СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc483468406)

[1. Обзор состояния вопроса 6](#_Toc483468407)

[1.1 Пространство имен System.IO 7](#_Toc483468408)

[1.2 Классы Directory и File 8](#_Toc483468409)

[1.3 Основные методы классов Directory и File 8](#_Toc483468410)

[1.3 Понятие сокетов 10](#_Toc483468411)

[2 Постановка задачи 11](#_Toc483468412)

[3 Проектирование программного обеспечения 12](#_Toc483468413)

[3.1 Архитектура ПО 12](#_Toc483468414)

[3.2 Алгоритмы работы модулей 12](#_Toc483468415)

[3.2.1 Описание работы модуля Client 13](#_Toc483468416)

[3.2.2 Описание работы модуля Server 14](#_Toc483468417)

[3.3 Интерфейс пользователя 15](#_Toc483468418)

[3.4 Библиотечные классы и методы 16](#_Toc483468419)

[3.5 Разработанные классы и методы 16](#_Toc483468420)

[3.5.1 Методы класса Server: 16](#_Toc483468421)

[3.5.2 Методы класса Client : 16](#_Toc483468422)

[4 Руководство пользователя 18](#_Toc483468423)

[ЗАКЛЮЧЕНИЕ 19](#_Toc483468424)

[Литература 20](#_Toc483468425)

[ПРИЛОЖЕНИЕ A 21](#_Toc483468426)

[ПРИЛОЖЕНИЕ B 23](#_Toc483468427)

# ВВЕДЕНИЕ

Главный объект, с которым работает любой компьютер – это информация. Как и человек, компьютер обрабатывает разного рода информацию: текст, графику, звук, видео и т.д. В представлении компьютера информация – это вереница из нулей и единиц, а файл – это хранилище информации определённого типа.

Каждый файл имеет размер и состоит из названия и расширения. Расширение файла – это символы, отделённые от названия файла точкой. Расширение помогает компьютеру и человеку понять, какой тип информации содержится в файле. Например, «.txt» — это текстовый файл, «.jpg» или «.bmp» — это графические файлы, «.mp3» или «.wav» — это музыкальные файлы, «.avi» или «.vob» — это видео файлы и т.д. По расширению компьютер понимает, какую программу надо запустить, чтобы обработать выбранный пользователем файл.

Для более эффективной и интуитивно понятной работы с файлами были разработаны файловые менеджеры. Файловый менеджер - компьютерная программа, предоставляющая интерфейс пользователя для работы с файловой системой и файлами. Файловый менеджер позволяет выполнять наиболее частые операции над файлами - создание, открытие, проигрывание, просмотр, редактирование, перемещение, переименование, копирование и другое.

В качестве среды разработки визуальной оболочки была выбрана Microsoft Visual Studio, язык программирования C#.

Цель курсового проекта – закрепление и углубление знаний, полученных при изучении курса «Компьютерные сети и системы» посредством разработки программного обеспечения на языке С#.

Для достижения цели курсового проекта были поставлены следующие задачи:

* Охарактеризовать технологию сокетов;
* Рассмотреть типы сокетов;
* Рассмотреть и охарактеризовать команды для работы с файлами в Windows;
* Разработать программу, реализующую функции файлового менеджера.

1. Обзор состояния вопроса

Файл является основной ячейкой информации. Любая информация (текст, графика, видео, звук, фото) хранится в виде файлов. В виде отдельного файла хранится каждый снимок, сделанный цифровой камерой. С другой стороны, целый фильм также хранится отдельным файлом.

Если сваливать все файлы в одну кучу, то потом сложно найти нужный файл. Поэтому группу файлов по какому-либо общему признаку объединяют в папку.,

Многие пользователи для работы с файлами и папками предпочитают использовать специализированные приложения, вместо стандартной программы Проводник. Такие программы для управления файлами называют файловыми менеджерами.

Файловый менеджер - один из самых популярных инструментов при работе с любой операционной системой. Им пользуются практически все. Выбор конкретной программы для повседневной работы с файлами у каждого человека очень субъективен. Но в первую очередь он зависит от широты и качества полезных функций.

Современный файловый менеджер должен:

* Обеспечивать удобную возможность работы с файлами;
* Копировать;
* Удалять,
* Перемещать;
* Создавать;
* Редактировать текстовые файлы;
* Позволять с легкостью и удобством работать файлами.

В языке C# есть пространство имен System.IO, в котором реализованы все необходимые классы для работы с файлами.

Всю работу с файлами можно разделить на 2 группы:

* работа с файлами, как с элементами файловой системы: найти файл, получить список файлов в директории, узнать дату изменения файла, различные его атрибуты, расширение, скопировать или удалить файл, создать новый;
* работа с содержимым файла: прочитать файл или записать в него что-нибудь.

Для выполнения всех этих операций существует несколько классов, расположенных в пространстве имён "System.IO".

Пространство имен System.IO — это область библиотек базовых классов, посвященная службам файлового ввода-вывода, а также ввода-вывода из памяти. Подобно любому пространству имен, в System.IO определен набор классов, интерфейсов, перечислений, структур и делегатов, большинство из которых находятся в mscorlib.dll. В дополнение к типам, содержащимся внутри mscorlib.dll, в сборке System.dll определены дополнительные члены пространства имен System.IO.

Многие типы из пространства имен System.IO сосредоточены на программных манипуляциях физическими каталогами и файлами. Дополнительные типы предоставляют поддержку чтения и записи данных в строковые буферы, а также области памяти.

В языке C# существует 2 класса File и FileInfo, которые предназначены для работы с файлом, как с частью файловой системы. Здесь также есть методы, которые позволяют работать с содержимым файла целиком.

* 1. Пространство имен System.IO

Классы работы с файловой системой находятся в пространстве имен **System.IO**. **IO** – сокращение от английского **Input/Output** (ввод/вывод, запись/чтение). Пространство имен **System.IO** содержит классы для работы с каталогами и дисками, такие как: **DriveInfo**, **Directory**, **DirectoryInfo**. Классы для работы с файлами: **File**, **FileInfo**. Класс для работы с путями: **Path**. В той или иной степени, в процессе своей деятельности, нам придется использовать все перечисленные классы (рис. 1.1). Но не стоит пугаться, для облегчения жизни программистов в **.NET Framework** есть специальные статичные классы, такие как **File** и **Directory**, которые имеют готовые, часто используемые функции и методы манипуляций с каталогами и файлами.

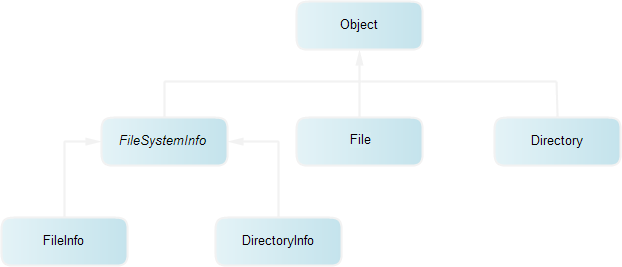


Рисунок 1.1 – Классы для работы с файлами и каталогами

* 1. Классы Directory и File

Классы Directory и File содержат только статические методы, а их экземпляры никогда не создаются. Для их использования нужно просто предоставить путь к интересующему объекту файловой системы при каждом вызове метода-члена. Если требуется выполнить только одну операцию в отношении папки или файла, то применение этих классов является наиболее эффективным подходом, поскольку позволяет избегать накладных расходов, связанных с созданием экземпляров.

Классы DirectoryInfo и FileInfo реализуют практически те же общедоступные методы, что и Directory, и File, а также некоторые общедоступные свойства и конструкторы, но поддерживают состояние, а их члены не являются статическими.

Сначала понадобится создать их экземпляр, который затем ассоциировать с определенной папкой или файлом. Это означает, что применение этих классов будет более эффективным подходом, если необходимо выполнять множество операций в отношении одного и того же объекта. Дело в том, что во время создания они считывают информацию об аутентификации и прочие сведения о соответствующем объекте файловой системы и больше не нуждаются в ее повторном чтении, сколько бы методов не вызывалось для данного объекта (экземпляра класса). В отличие от этого, классы, не поддерживающие состояние, нуждаются в проверке деталей, связанных с файлом или папкой, при каждом вызове какого-либо метода.

* 1. Основные методы классов Directory и File

Для создания пустого файла, в классе File есть метод Create(). Он принимает один аргумент – путь. Ниже приведен пример создания пустого текстового файла new\_file.txt на диске D:

static void Main(string[] args)  
{  
   File.Create("D:\\new\_file.txt");  
}

Если файл с таким именем уже существует, он будет переписан на новый пустой файл.

Метод AppendAllText() работает, как и метод WriteAllText() за исключением того, что новый текст дописывается в конец файла, а не переписывает всё что было в файле: 

static void Main(string[] args)  
{  
   File.AppendAllText("D:\\new\_file.txt", "текст метода AppendAllText ()");  
}

Метод Delete() удаляет файл по указаному пути:

static void Main(string[] args)  
{  
   File.Delete("d:\\test.txt"); //удаление файла   
}

Проверить существование файла

static void Main(string[] args)  
{  
   bool isExisFile = File.Exists(path);  
}

Метод File.GetCreationTime (String) возвращает дату и время создания заданного файла или каталога.

static void Main(string[] args)  
{

String time = File.GetCreationTime(dirName + name);

}

Метод Directory.SetCreationTime (String, DateTime) устанавливает дату и время создания заданного файла или каталога.

Directory.SetCreationTime(string path,DateTime creationTime), где path - файл или каталог, для которого требуется установить дату и время создания, а creationTime – дата и время.

Метод Directory.GetFiles(String) возвращает имена файлов (с указанием пути к ним) в указанном каталоге.

Метод Directory.GetDirectories(String) возвращает имена подкаталогов (включая пути) в указанном каталоге.

* 1. Понятие сокетов

Сокет (Socket - гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки сетевого соединения. Технология (интерфейс) сокетов – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут выполняться как на одном компьютере, так и на разных, связанных между собой сетью. Интерфейс сокетов был разработан в Калифорнийском университете в г. Беркли и встроен в ОС BSD Unix в 1983 г. Важнейшим преимуществом сокетов является предоставление единого независимого интерфейса сетевого программирования для различных сетевых протоколов. Библиотека Win32 Windows Sockets (Winsock) предоставляет механизмы программирования сокетов. В Microsoft .NET Framework имеется более высокий по отношению к Winsock уровень, благодаря чему управляемые приложения также могут взаимодействовать через сокетов.

Интерфейс сокетов расположен над транспортным уровнем стека TCP/IP, но в некоторых случаях может взаимодействовать напрямую с сетевым уровнем в обход транспортного.

* 1. Протокол. Описание протокола взаимодействия

Для создания клиент-серверного приложения был использован протокол транспортного уровня TCP. TCP (Transmission Control Protocol) – протокол управления передачей, один из самых широко распространенных протоколов транспортного уровня.

Протокол транспортного уровня TCP в данном курсовом проекте используется по следующим причинам:

* Это ориентированный на соединение протокол, предназначенный для обеспечения надежной передачи данных между процессами.
* Соединение с использованием TCP/IP сокетов является постоянным и определяется в двух направлениях. C помощью TCP/IP сокетов можно программировать подключение систем ввода/вывода к программам, расположенным на любом компьютере в сети.
* TCP выполняет функции контроля ошибок и управления потоком данных.

TCP не поддерживает широковещание и многоадресную рассылку. Он может использоваться только для соединений «один-к-одному». При этом протокол устанавливает дуплексный виртуальный канал передачи между конечными узлами. Каждый взаимодействующий процесс идентифицируется сокетом – парой IP-адрес и номер порта. Логическое соединение по протоколу TCP между двумя прикладными процессами идентифицируется парой сокетов. Каждый процесс одновременно может участвовать в нескольких соединениях.

Для работы с TCP протоколом среда .NET Framework предоставляет высокоуровневые классы TcpListener и TcpClient, относящиеся к пространству имен System.Net.Sockets.

Классы TcpListener и TcpClient используют потоковую модель. В них взаимодействие между клиентом и сервером основывается на потоке с применением класса NetworkStream.

Класс TcpClient обеспечивает работу с TCP-протоколом на стороне клиента. Он построен на классе Socket и обеспечивает TCP-сервисы на более высоком уровне. В классе TcpClient есть закрытый объект данных m\_ClientSocket, используемый для взаимодействия с сервером TCP.

Класс TcpListener обеспечивает работу с TCP-протоколом на стороне сервера. Он слушает запросы клиентов, принимает запрос и создает новый экземпляр класса Socket или TcpClient, который можно использовать для взаимодействия с клиентом. Класс TcpListener инкапсулирует закрытый объект Socket, m\_ServerSocket, доступный только для производных классов.

Всё взаимодействие сервера и клиента будет заключаться в специальных командах посылаемых друг другу в виде текстовых строк.

Каждая команда состоит из 4 текстовых полей (индекс функции, адрес директории, имя файла, текст), разделенных символом “|”. В общем виде это выглядит так:

< индекс функции >|< адрес директории > |< имя файла > |< текст >

Сервер принимает сообщение, после чего производит его расшифровку и вызов соответствующих функций с передачей соответствующих параметров.

2 Постановка задачи

В данном курсовом проекте необходимо написать программу, реализующую функции файлового менеджера. Реализовать взаимодействие на основе сокетов Windows. Архитектура клиент-сервер. Графический интерфейс.

Исходные данные для проекта:

* ОС MS Windows;
* MS Visual Studio .NET;
* язык программирования C#.

3 Проектирование программного обеспечения

3.1 Архитектура ПО

Общая структура клиент-серверного приложения представлена на рисунке 3.1.1.



Рисунок 3.1.1 - Общая схема взаимодействия клиент-серверного приложения

3.2 Алгоритмы работы модулей

Курсовой проект состоит из следующих проектов:

* Client
* Server

3.2.1 Описание работы модуля Client

В данном модуле был реализован класс Client.

У класса Client есть переменная client типа TcpClient, изначально инициализированная значением null. Конструктор класса принимает переменные port типа int и IP типа String.

У класса Client есть два метода:

* send(string text);
* receive();

Метод send принимает текст в виде строки. В методе создается массив байт, в который помещается строка, переданная в метод.

Данные передаются по сети между двумя узлами в форме непрерывного потока. Для обработки таких потоков в .NET имеется специальный класс NetworkStream. Класс NetworkStream входит в пространство имен System.Net.Sockets и используется для отправки и получения данных через сокеты. Объект NetworkStream – это небуферизованный поток, который не поддерживает произвольный доступ к данным.

Прежде чем отправлять и получать данные, нужно определить базовый поток. Для этих целей в классе TcpClient есть метод GetStream(). Он создает экземпляр класса NetworkStream и возвращает его вызывающей программе.

В методе создается поток для отправки массива байт серверу. С помощью метода write() происходит запись в поток. Он записывает в поток последовательность байтов и продвигает текущую позицию в потоке вперед на число записанных байтов.

Метод receive возвращает текст типа string, отправленный сервером. В методе создается массив байт, для принимаемого сообщения. Создается поток типа NetworkStream для сообщений сервера. С помощью метода read() происходит синхронное чтение из потока. Он считывает указанное число байтов и продвигает позицию в потоке вперед на число считанных байтов.

В главной функции модуля Client создается объект типа Client в конструктор, к которому передается номер порта и IP-адрес. В переменные menu, address, fileName, text типа string заносятся введенные пользователем данные. Где:

* address – путь к директории;
* filename – имя файла;
* menu – выбранный пользователем пункт меню;
* text – введенный пользователем текст.

Далее на объекте client вызывается метод send, которому передаются переменные menu, address, fileName, text с разделителем “|”, после чего клиент ждет ответа от сервера.

3.2.2 Описание работы модуля Server

В данном модуле был реализован класс Server.

Метод run() класса Server создает объект serverListener типа TcpListener.

С помощью метода start() на объекте serverListener экземпляр TcpListener переводится в режим прослушивания.

Далее создается поток для чтения и записи stream типа NetworkStream.

С помощью метода read(…) на объекте stream сервер считывает сообщения клиента в массив байт.

Сообщение, полученное от клиента, передается методу manage(), который обрабатывает данное сообщение. Обработка происходит путем разделения сообщения на 4 части, ранее разделенные символом “|”. Далее в методе manage() с помощью конструкции switch-case происходит вызов соответствующих методов класса Server для достижения желаемого пользователем результата.

Функциональные методы класса Server работают с встроенными классами File, Directory, DirectoryInfo, FileInfo.

Для отправки и получения сокетов используются методы send() и receive() соответственно, реализованные по такому же принципу, как и в классе Client.

3.3 Интерфейс пользователя

Интерфейс пользователя изображён на рисунке 3.3.

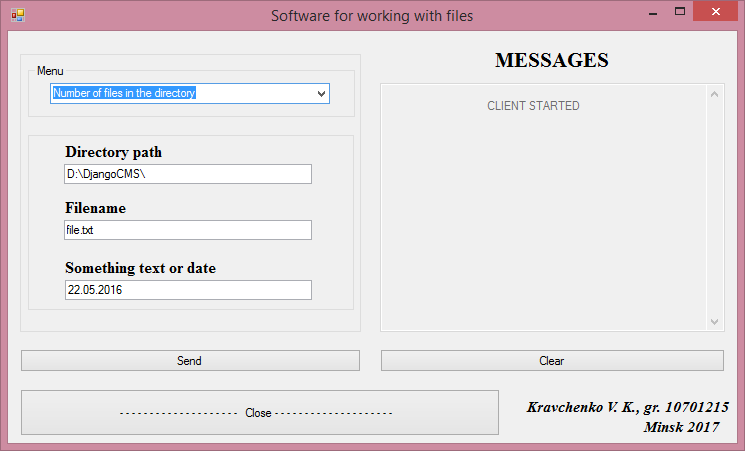


Рисунок 3.3 – интерфейс пользователя

Из предложенных пунктов меню пользователю необходимо выбрать интересующий. В поле «Directory path» необходимо указать путь к директории. В поле «Filename» необходимо указать имя файла. В зависимости от выбранного пункта меню в поле «Something text or date» пользователю предлагается ввести текст. Кнопка «Send» нужна для отправки сокета серверу. В поле «Messages» отобразиться ответ от сервера. После нажатия кнопки «Close» программа завершит свою работу.

3.4 Библиотечные классы и методы

В процессе проектирования ПО для работы с файлами были использованы следующие библиотечные классы:

* System.Net.Sockets;
* System.IO;

3.5 Разработанные классы и методы

В ходе выполнения курсового проекта были разработаны следующие основные классы и методы:

3.5.1 Методы класса Server:

Основные методы класса Server:

* run() – запускает сервер;
* send() – отправка сокета;
* showDirInDir() – показать все папки директории;
* sizeFile() – размер файла;
* createFile() – создать файл;
* deleteFile() – удалить файл;
* creationTimeFile() – время создания файла;
* SetCreationTimeFile() – изменить время создания файла;
* modificationTimeFile() – время последнего изменения файла;
* setModificationTimeFile() - изменить время последнего изменения файла;
* numberDirInDir() – количество папок в директории;
* numberFilesInDir() – количество файлов в директории;
* dirSize() – размер каталога.

3.5.2 Методы класса Client :

* send() – отправить сообщение;
* receive() – принять сообщение;

4 Руководство пользователя

Для запуска приложения необходимо иметь на компьютере ОС Windows версии 7 и выше и .Net Framework версии 4.6 и выше. Для запуска сервера нужно запустить Server.exe. Для запуска клиента запустить приложение Client.exe.

После запуска приложения появится главная форма (Рисунок 4.1)

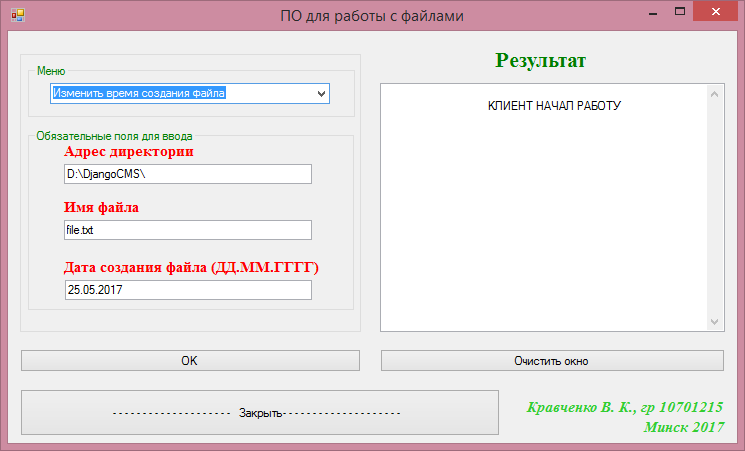


Рисунок 4.1 – Главное окно клиента.

Пользователю предоставляется возможность выбора одной из множества функций в поле «Меню».

Функции, предоставляемые пользователю:

* Показать все папки, выбранной директории
* Размер файла
* Создать файл
* Удалить файл
* Найти файл
* Узнать время создания файла
* Изменить время создания файла
* Узнать время последнего изменения файла
* Изменить время последнего изменения файла
* Количество папок в директории
* Количество файлов в директории
* Размер директории
* Показать все файлы, выбранной директории

После выбора интересующего пункта меню, необходимо заполнить поля, обязательные для заполнения.

Далее пользователю необходимо нажать кнопку «ОК», после чего в окне «Результат» отразится результат выполнения операции.

Кнопка «Очистить» очищает окно «Результат», а кнопка «Закрыть» завершает выполнение программы.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта были изучены: принципы работы с файлами посредством языка C#, средства .NET Framework для разработки клиент-серверного приложения на базе сокетов.

Разработано ПО для работы с файлами, демонстрирующая клиент-серверное приложение в ОС Windows.

Серверное приложение обладает такими возможностями, как: подключение к нему клиентов, прием и обработка запросов клиентов.

Был закреплен материал, пройденный на курсе «Компьютерных сетей и систем».

Данный программный продукт отвечает поставленным требованиям, имеет интуитивно понятный и удобный интерфейс.

Литература

1. Троелсен Э.У. C# и платформа .NET. Библиотека программиста. / Э.У. Троелсен. – СПб.: Питер, 2004. – 796 с.
2. Белова, С.В. Компьютерные сети: лабораторный практикум / С.В. Белова. – Минск: БНТУ, 2012. – 10-16 с.
3. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#, 3-е издание Санкт-Петербург: «БХВ-Петербург», 2012 – 200-540.
4. Работа с файлами C# [Электронный ресурс] – Режим доступа: http://mycsharp.ru/post/si-sharp\_klassy\_streamreader\_i\_streamwriter.html .

# ПРИЛОЖЕНИЕ A

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вырезка экрана | | | | | | | | | | |
|  |  |  |  |  | КП—10701215 03–2017–05 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Кравченко |  |  | Диаграмма классов проекта Client | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **1** | **3** |
| Консульт. | | Белова |  |  | 1070121511 БНТУ  г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | | Полосков |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вырезка экрана | | | | | | | | | | |
|  |  |  |  |  | КП—10701215 03–2017–05 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Кравченко |  |  | Диаграмма классов проекта Server | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **2** | **3** |
| Консульт. | | Белова |  |  | 1070121511 БНТУ  г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | | Полосков |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701215 03–2017–05 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Кравченко |  |  | Интерфейс пользователя и результат работы сервера | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **3** | **3** |
| Консульт. | | Белова |  |  | 1070121511 БНТУ  г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | | Полосков |  |  |

# ПРИЛОЖЕНИЕ B

Form1.cs (Client)

using System;

using System.IO;

using System.Windows.Forms;

namespace Client

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

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

{

int port = 8888;

String IP = "127.0.0.1";

Client client = new Client(port, IP);

String menu = comboBox1.SelectedIndex.ToString();

String addrres = Path.GetFullPath(textBox2.Text);

String fileName = textBox3.Text;

String text = textBox4.Text;

client.send(menu + "|" + addrres + "|" + fileName + "|" + text);

textBox1.Text += Environment.NewLine + client.receive();

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

textBox1.Text += Environment.NewLine + "\t\t СLIENT STARTED\n";

}

private void button2\_Click(object sender, EventArgs e)

{

textBox1.Clear();

}

private void button3\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Client.cs

using System;

using System.Net.Sockets;

namespace Client

{

class Client

{

private TcpClient client = null;

public Client() { }

public Client(int port, String IP)

{

try

{

client = new TcpClient(IP, port);

}

catch (ArgumentNullException exception)

{

Console.WriteLine("ArgumentNullException: {0}", exception);

}

catch (SocketException exception)

{

Console.WriteLine("SocketException: {0}", exception);

}

}

public void send(String message)

{

try {

Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

NetworkStream stream = client.GetStream();

stream.Write(data, 0, data.Length);

}

catch (ArgumentNullException exception)

{

Console.WriteLine("ArgumentNullException: {0}", exception);

}

catch (SocketException exception)

{

Console.WriteLine("SocketException: {0}", exception);

}

}

public String receive()

{

String responseData = String.Empty;

try

{

Byte [] data = new Byte[256];

NetworkStream stream = client.GetStream();

Int32 bytes = stream.Read(data, 0, data.Length);

responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);

stream.Close();

}

catch (ArgumentNullException exception)

{

Console.WriteLine("ArgumentNullException: {0}", exception);

}

catch (SocketException exception)

{

Console.WriteLine("SocketException: {0}", exception);

}

return responseData;

}

}

}

Server.cs

using System;

using System.Text;

using System.Net;

using Syastem.Net.Sockets;

using System.IO;

namespace Server

{

class Server

{

private int port = 1025;

private IPAddress localAddr = null;

private TcpClient client = null;

private NetworkStream stream = null;

public Server(int port, IPAddress localAddr)

{

this.port = port;

this.localAddr = localAddr;

}

public void run()

{

Console.WriteLine("SERVER RUNNING");

TcpListener serverListener = new TcpListener(localAddr, port);

try

{

while (true)

{

serverListener.Start();

Console.WriteLine("\nWaiting for connections... ");

client = serverListener.AcceptTcpClient();

// получаем входящее подключение

Console.WriteLine("Client is connected. Executing the query...");

// поток для чтения и записи

stream = client.GetStream();

Console.WriteLine("\*\*\*\*");

byte[] data2 = new byte[256];

String responseData = String.Empty;

Int32 bytes = stream.Read(data2, 0, data2.Length);

responseData = System.Text.Encoding.ASCII.GetString(data2, 0, bytes);

Console.WriteLine("Received: {0}", responseData);

manage(responseData);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

}

finally

{

if (serverListener != null)

serverListener.Stop();

Console.ReadLine();

}

}

public void manage(String msg)

{

String[] data = msg.Split('|');

int index = Convert.ToInt32(data[0]);

string directory = data[1];

string fileName = data[2];

string text = data[3];

switch (index)

{

case 0:

{

send(stream, showDirInDir(directory));

break;

}

case 1:

{

send(stream, sizeFile(directory, fileName));

break;

}

case 2:

{

send(stream, createFile(directory, fileName, text));

break;

}

case 3:

{

send(stream, deleteFile(directory, fileName));

break;

}

case 4:

{

send(stream, findFile(directory, fileName));

break;

}

case 5:

{

send(stream, creationTimeFile(directory, fileName));

break;

}

case 6:

{

send(stream, SetCreationTimeFile(directory, fileName, text));

break;

}

case 7:

{

send(stream, modificationTimeFile(directory, fileName));

break;

}

case 8:

{

send(stream, setModificationTimeFile(directory, fileName, text));

break;

}

case 9:

{

send(stream, numberDirInDir(directory));

break;

}

case 10:

{

send(stream, numberFilesInDir(directory));

break;

}

case 11:

{

send(stream, "Size of directory: " + dirSize(directory).ToString());

break;

}

default:

{

send(stream, "SOMETHING WRONG");

break;

}

}

}

private void send(NetworkStream stream, String message)

{

byte[] data = Encoding.UTF8.GetBytes(message);

// отправка сообщения

stream.Write(data, 0, data.Length);

Console.WriteLine("Sent: {0}", message);

}

//показать все папки директории

public static String showDirInDir(string dirName)

{

string[] dirs = null;

if (Directory.Exists(dirName))

{

dirs = Directory.GetDirectories(dirName);

}

StringBuilder builder = new StringBuilder();

for (int n = 0; n < dirs.Length; n++)

{

builder.Append(dirs[n]);

builder.Append(Environment.NewLine);

}

return "Directories:" + Environment.NewLine + builder.ToString();

}

//размер файла

public static String sizeFile(string dirName, string name)

{

String rezult = "SOMETHING WRONG";

if (File.Exists(dirName + name))

{

long fileSize = new FileInfo(dirName + name).Length;

rezult = "Size of file: " + fileSize.ToString() + " bytes;";

}

return rezult;

}

//создать файл

public static String createFile(string dirName, string name, string text)

{

String rezult = "SOMETHING WRONG";

if (Directory.Exists(dirName))

{

File.AppendAllText(dirName + name, text);

rezult = "File " + name + " created;";

}

return rezult;

}

//удалить файл

public static String deleteFile(string dirName, string name)

{

String rezult = "FILE DOES NOT EXIST";

if (File.Exists(dirName + name))

{

File.Delete(dirName + name);

rezult = "File " + name + " deleted;";

}

return rezult;

}

//поиск файла

public static String findFile(string dirName, string name)

{

String rezult = "File not found";

if (File.Exists(dirName + name))

{

rezult = "File " + name + " in " + dirName + " exist;";

}

return rezult;

}

//время создания файла

public static String creationTimeFile(string dirName, string name)

{

String rezult = "File not found";

if (File.Exists(dirName + name))

{

rezult = "File creation time: ";

rezult += File.GetCreationTime(dirName + name);

}

return rezult;

}

//изменение времени создания

public static String SetCreationTimeFile(string dirName, string name, string text)

{

String rezult = "File not found";

if (File.Exists(dirName + name))

{

string [] date = text.Split('.');

File.SetCreationTime(dirName + name, new DateTime(Convert.ToInt16(date[2]), Convert.ToInt16(date[1]), Convert.ToInt16(date[0])));

rezult = "Creation time changed.";

}

return rezult;

}

//время изменения файла

public static String modificationTimeFile(string dirName, string name)

{

String rezult = "File not found";

if (File.Exists(dirName + name))

{

rezult = "Last modification time: ";

rezult += File.GetLastWriteTime(dirName + name);

}

return rezult;

}

//изменение времени изменения файла

public static String setModificationTimeFile(string dirName, string name, string text)

{

String rezult = "File not found";

if (File.Exists(dirName + name))

{

string[] date = text.Split('.');

File.SetLastWriteTime(dirName + name, new DateTime(Convert.ToInt16(date[2]), Convert.ToInt16(date[1]), Convert.ToInt16(date[0])));

rezult = "Modification time changed.";

}

return rezult;

}

//количество папок в директории

public static String numberDirInDir(string dirName)

{

string rezult = "SOMETHING WRONG";

string[] dirs = null;

if (Directory.Exists(dirName))

{

dirs = Directory.GetDirectories(dirName);

rezult = "The number of directories in the directory - ";

rezult += Convert.ToString(dirs.Length);

rezult += ";";

}

return rezult;

}

//количество файлов в директории

public static String numberFilesInDir(string dirName)

{

string rezult = "SOMETHING WRONG";

string[] dirs = null;

if (Directory.Exists(dirName))

{

dirs = Directory.GetFiles(dirName);

rezult = "The number of directories in the directory - ";

rezult += Convert.ToString(dirs.Length);

rezult += ";";

}

return rezult;

}

//размер каталога

public long dirSize(string dirName)

{

long size = 0;

if (Directory.Exists(dirName))

{

DirectoryInfo DrInfo = new DirectoryInfo(dirName);

DirectoryInfo[] folder = DrInfo.GetDirectories();

FileInfo[] Fi = DrInfo.GetFiles();

foreach (FileInfo fl in Fi)

{

size += fl.Length;

}

for (int j = 0; j < folder.Length; j++)

{

size += dirSize(dirName + "\\" + folder[j].Name);

}

}

return size;

}

}

}