МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

ОТЧЁТ

по Курсовой работе по дисциплине

Принципы и методы организации системных программных средств

Тема: «Создание одноуровневой файловой системы»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Викулова Е.Н

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уткин А.О

21-ПО

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2024

Оглавление

[Введение 2](#_Toc167791795)

[Цель работы 4](#_Toc167791796)

[Проблемы и их решения 4](#_Toc167791797)

[Обзор c# 6](#_Toc167791798)

[Планирование системы 7](#_Toc167791799)

[Описание GUI 9](#_Toc167791800)

[Алгоритмы работы файловой системы 10](#_Toc167791801)

[Форматирование 10](#_Toc167791802)

[Создание нового файла 11](#_Toc167791803)

[Удаление и восстановление 13](#_Toc167791804)

[Чтение 14](#_Toc167791805)

[Использование паттерна «Витрина данных» 15](#_Toc167791806)

[Структура проекта 16](#_Toc167791807)

[Тестирование 17](#_Toc167791808)

[Вывод 21](#_Toc167791809)

# Введение

**Фа́йловая систе́ма** (англ. *file system*) — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов (и каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Файловая система связывает носитель информации с одной стороны и API для доступа к файлам — с другой. Когда прикладная программа обращается к файлу, она не имеет никакого представления о том, каким образом расположена информация в конкретном файле, так же как и о том, на каком физическом типе носителя (CD, жёстком диске, магнитной ленте, блоке флеш-памяти или другом) он записан. Всё, что знает программа — это имя файла, его размер и атрибуты. Эти данные она получает от драйвера файловой системы. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жёстком диске).

С точки зрения операционной системы (ОС), весь диск представляет собой набор кластеров (как правило, размером, кратным 512 байт). Драйверы файловой системы организуют кластеры в файлы и каталоги (реально являющиеся файлами, содержащими список файлов в этом каталоге). Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

Однако файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы, а также сетевые файловые системы, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

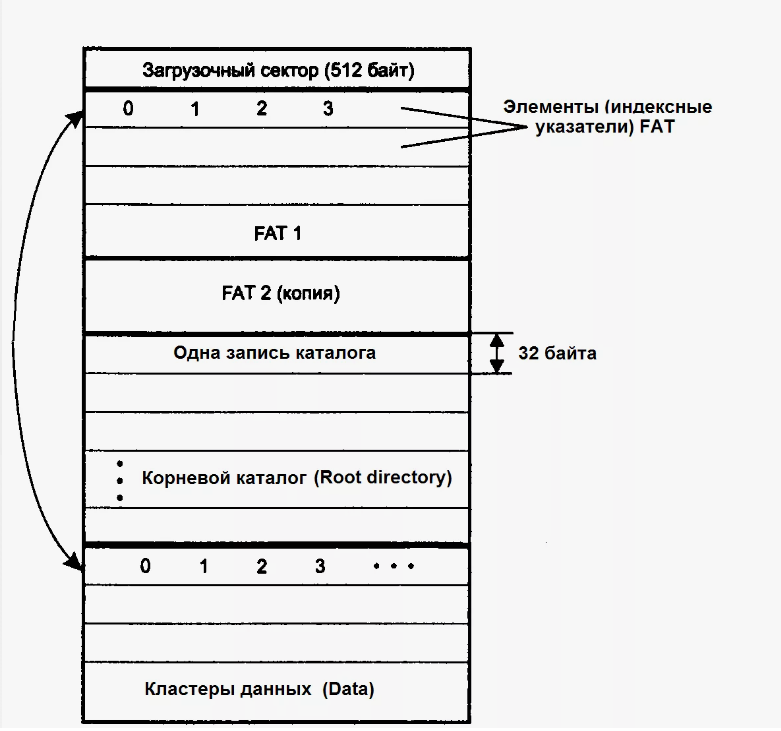


Рис 1.1 «Структура FAT»

**FAT** (англ. *File Allocation Table* «таблица размещения файлов») — классическая архитектура файловой системы, которая из-за своей простоты всё ещё широко применяется для флеш-накопителей. Ранее находила применение на дискетах и на жёстких дисках.

Разработана Биллом Гейтсом и Марком МакДональдом в 1976—1977 годах. Использовалась в качестве основной файловой системы в операционных системах семейств MS-DOS и Windows 9x.

В файловой системе FAT смежные секторы диска объединяются в единицы, называемые кластерами. Количество секторов в кластере равно степени двойки (см. далее). Для хранения данных файла отводится целое число кластеров (минимум один), так что, например, если размер файла составляет 40 байт, а размер кластера 4 Кбайт, реально занят информацией файла будет лишь 1 % отведённого для него места. Во избежание подобных ситуаций целесообразно уменьшать размер кластеров, а для сокращения объёма адресной информации и повышения скорости файловых операций — наоборот. На практике выбирают некоторый компромисс. Так как ёмкость диска вполне может и не выражаться целым числом кластеров, обычно в конце тома присутствуют так называемые surplus sectors — «остаток» размером менее кластера, который не может отводиться ОС для хранения информации.

Пространство тома FAT32 логически разделено на три смежные области:

* Зарезервированная область. Содержит служебные структуры, которые принадлежат загрузочной записи раздела (Partition Boot Record — PBR, для отличия от master boot record — главной загрузочной записи диска; также PBR часто некорректно называется загрузочным сектором) и используются при инициализации тома;
* Область таблицы FAT, содержащая массив индексных указателей («ячеек»), соответствующих кластерам области данных. Для повышения надёжности на диске обычно представлено две копии таблицы FAT;
* Область данных, где записано собственно содержимое файлов — то есть текст текстовых файлов, кодированное изображение для файлов рисунков, оцифрованный звук для аудиофайлов и т. д.

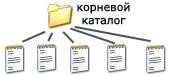
В FAT12 и FAT16 также специально выделяется область корневого каталога. Она имеет фиксированное положение (непосредственно после последнего элемента таблицы FAT) и фиксированный размер в 32-байтных элементах, то есть при описании в Partition Boot Record указывается именно количество 32-байтных элементов, каждый из которых описывает какой-либо элемент корневого каталога (будь то файл или другой вложенный каталог). 

Рис 1.2 «Одноуровневая файловая система»

Для дисков с небольшим количеством файлов (до нескольких десятков) удобно применять *одноуровневую файловую систему*, когда каталог (оглавление диска) представляет собой линейную последовательность имен файлов. Для отыскания файла на диске достаточно указать лишь имя файла.

# Цель работы

Разработать структуру одноуровневой файловой системы, реализуемой на сменном или виртуальном диске. Написать программу, позволяющую формировать каталог, просматривать содержимое каталога, добавлять удалять, восстанавливать оценивать объем занятой и свободной информации.

# Проблемы и их решения

Изначально планировалось написать на ассемблере небольшую операционную систему на ассемблере под которой и работала бы новая файловая система. Она должна была состоять из загрузчика, ядра (по сути что-то типа оболочки) и текстового редактора. Код писался на диалекте FASM и bin файл конвертировался в .iso файл с помочью Ultra iso, после чего тестировался в VirtualBox. Довольно скоро стало понятно, что разрабатывать программу таким образом очень тяжело и малоэффективно. Стало появляться понимание, что что-то более менее сложное таким образом разработать не удастся. Кроме того возникало множество проблем не поддававшихся объяснению из-за недостатка понимания предметной области. Получилась не ОС но программа работающая, как говорится «на голом железе».

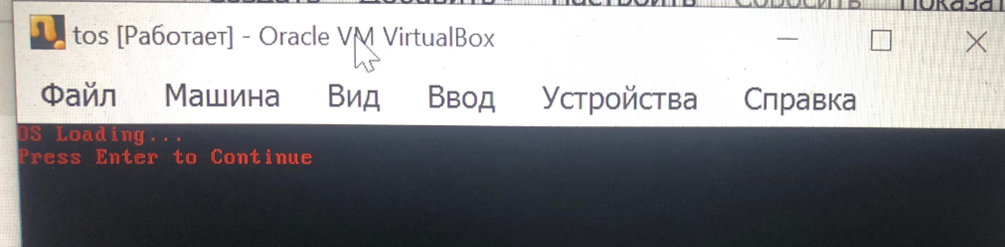


Рис 2.1 «Исполнение загрузчика»

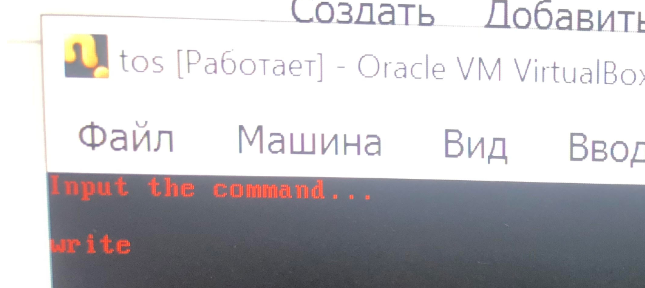


Рис 2.2 «Работа ядра»

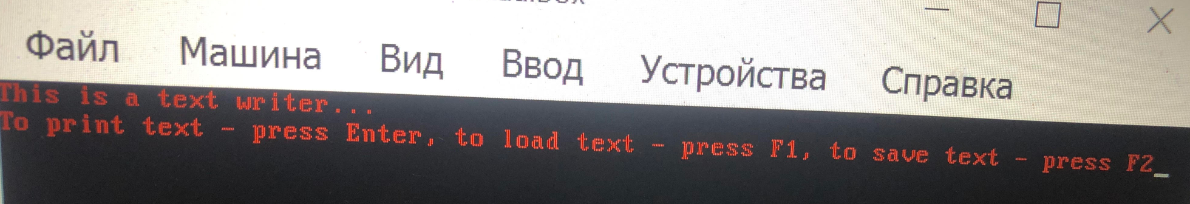


Рис 2.3 «Не работающий текстовый редактор»

По сначала неясным причинам загрузчик не мог записать ядро на диск виртуальной машины, а текстовый редактор и вовсе не работал. Проведя ряд экспериментов оказалось, что прерывание bios int 13h работает некорректно. Ниже примеры ошибок.

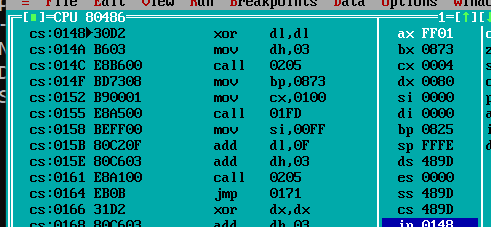


Рис 2.4а «al=FF ошибка операции считывания»

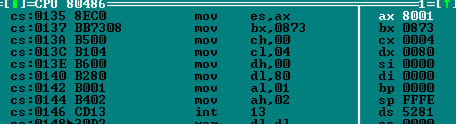


Рис 2.4а «al=80 таймаут»

Далее мне удалось узнать, что получить низкоуровневый доступ к физическому или даже к виртуальнлму iso диску можно получить также с помощью языков семейства C.

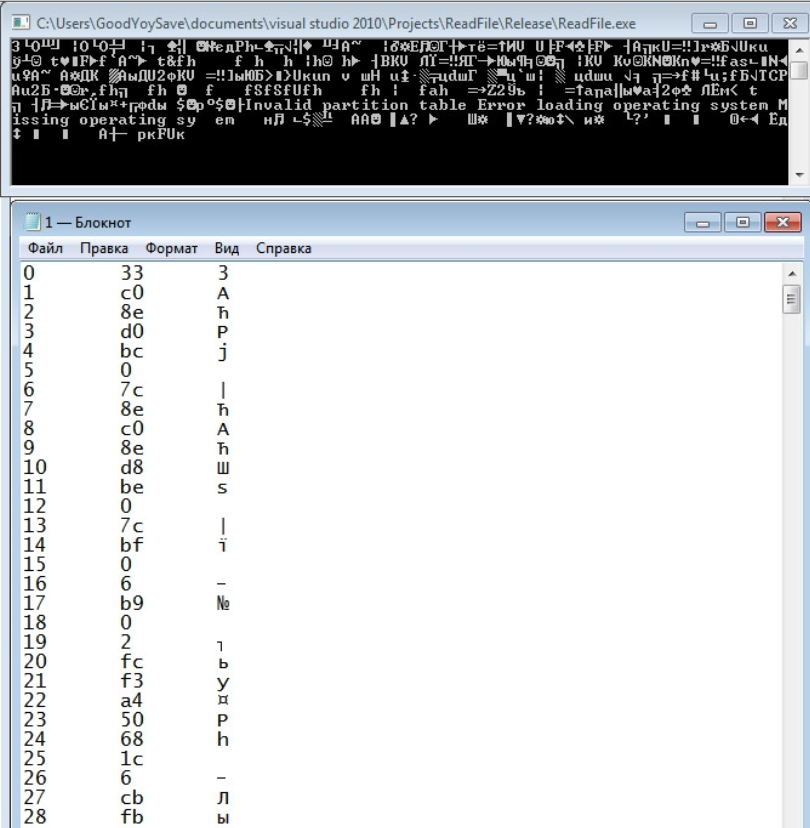


Рис 2.5 «Программа на c++, считывающая байты с диска в txt файл.

Таким образом было принято решение разрабатывать программу на языке c# с помощью среды Visual Studio, что гораздо удобнее чем первоначальный подход.

# Обзор c#

C# (произносится си шарп) — объектно-ориентированный язык программирования общего назначения. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Delphi, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

Название «Си шарп» (от англ. sharp — диез) происходит от буквенной музыкальной нотации, где латинской букве C соответствует нота До, а знак диез (англ. sharp) означает повышение соответствующего ноте звука на полутон, что аналогично названию языка C++, где «++» обозначает инкремент переменной. Название также является игрой с цепочкой C → C++ → C++++(C#), так как символ «#» можно представить состоящим из 4 знаков «+».

# Планирование системы



Рис 4.1-«Модель моей файловой системы»

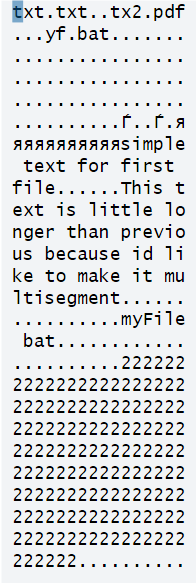


Рис 4.2-«Моя файловая система в программе Hex Editor Neo»

Размер моей файловой системы составляет 672 байта. Она специально сделана такой маленькой чтобы на ней легко было тестировать возможности контроля переполнения, многосегментности, а также для большей наглядности и более легкого тестирования.

Для корневого каталога предоставлены первые 144 байта для 16 записей по 9 байтов.

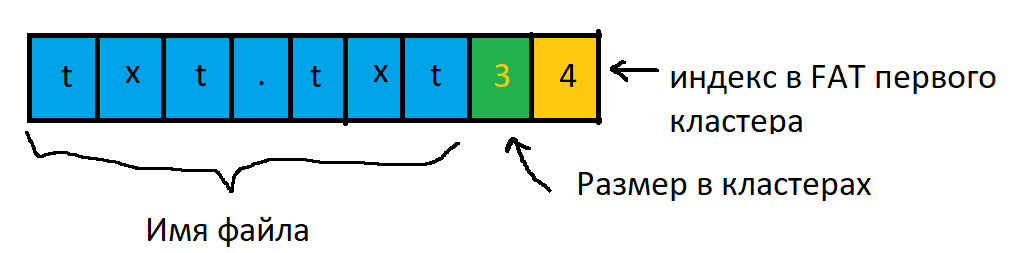


Рис 4.3-«Структура записи о файле»

Первые 7 байт- имя файла, размер в кластерах и индекс первого кластера. 16- записей на случай если надо будет заполнить диск однокластерными файлами.

В FAT таблице хранятся индексы фат таблицы указывающие также и на кластеры. В файле может содержаться число от 0 до 16 указывающее на байт, байт 255 или ff для пустых кластеров, 129 для конечных кластеров и 127 для конечного кластера удаленного файла, а также 248-254 для удаленных файлов.

# Описание GUI

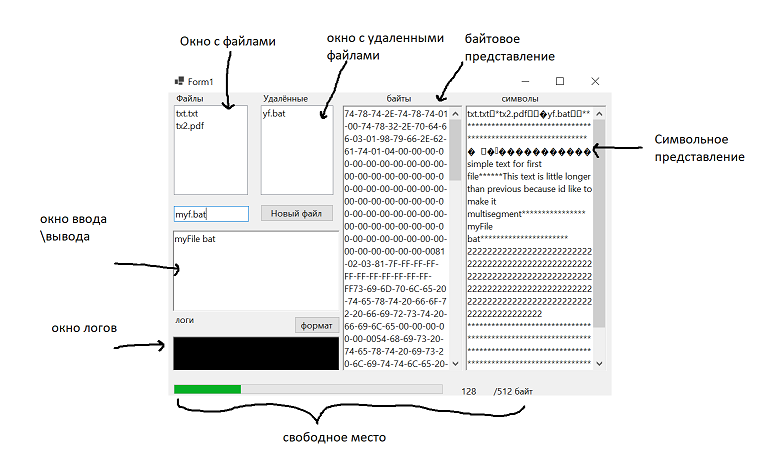


Рис 5.1-«Основные элементы GUI»

Окно с файлами- отображает файлы доступные для чтения и удаления для задействования соответствующих функций необходимо открыть контекстное меню.

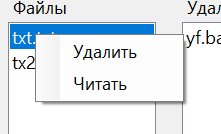


Рис 5.2-«Контекстное меню»

Окно с удаленными файлами- показывает удаленные файлы некоторые из которых возможно восстановить. Для использования соответствующей функции необходимо воспользоваться контекстным меню.

Окно ввода/вывода- используется одновременно для добавления текста нового файла и чтения уже имеющихся

Кнопка новый файл- создаёт новый файл, для этого надо ввести название в малое текстовое поле, ввести содержимое файла в окно ввода/вывода и нажать на кнопку новый файл.

Кнопка формат- форматирует диск.

Байтовое и символьное представление- байтовое показывает все байты как они есть, символьное конвертирует байты в соответствии с таблицей ASCII.

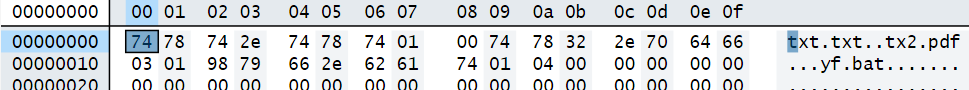


Рис 5.3а-«Анализ диска с помощью hex editor neo для проверки корректности вывода»

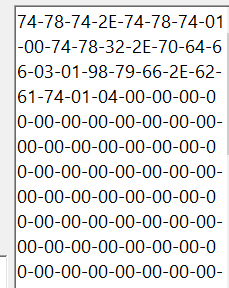


Рис 5.3а-«Байтовое представление в моей программе»

Снизу формы расположен progresbar, показывающий сколько места осталось наглдно. Справа от неё надпись которая делает тоже самое текстом.

# Алгоритмы работы файловой системы

## Форматирование

Для форматирования я использую так называемое быстрое форматирование, а точнее просто зануляю корневой каталог, и делаю все ячейки fat=255.

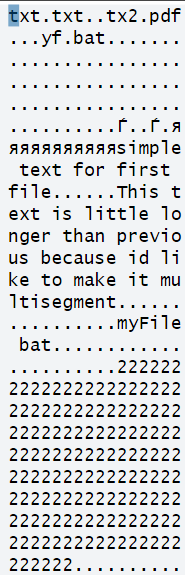
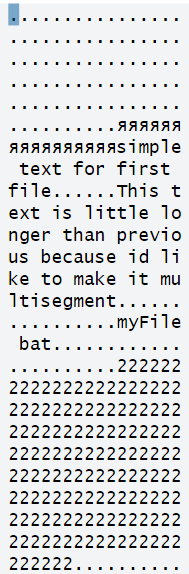
 

Рис 5.1.1а и Рис 5.1.1б -«До и после форматирования»

## Создание нового файла

Чтобы создать алгоритм создания файла сначала был разработан алгоритм поиска следующего свободного кластера:

Функция int СледующийКластер()

{

Для(int i=0; i<16;i++)

{

Если(myFAT[i]==255)

{

Вернуть i

}

}

//Проверим есть ли доступные удаленные кластеры

Для(int i=0; i<16;i++)

{

Если(myFAT[i]>255 или myFAT[i]==127 )

{

Вернуть i

}

}

Вернуть нет места

}

**Алгоритм создания файла:**

Функция void СоздатьФайл(string name, string content)

{

Int a=0;

Пока(a!=144)

{

Если (RootDir[a]==0)

{

RootDir[a:a+6]=name;

RootDir[a+7]=СледующийКластер()

RootDir[a+8]=ОкруглениеВБольшуюСт(content.Lenght/32)

Всё Пока

}

a\*=9

}

Int size= ОкруглениеВБольшуюСт(content.Lenght/32)

Int claster= RootDir[a+7]

Int num=0

Int clast=claster

Для(int i=0; i<content.Lenght;i++)

{

Clasters[clast][num]=content[i]

Num++;

Если(num==32)

{

Num=0

Clast=СледующийКластер

}

}

}

## Удаление и восстановление

Удаляя файл мы сначала заменяем первый байт в названии на 152. Почему не E5- 229?

Для этого нам придёте обратиться к таблице ASCII.

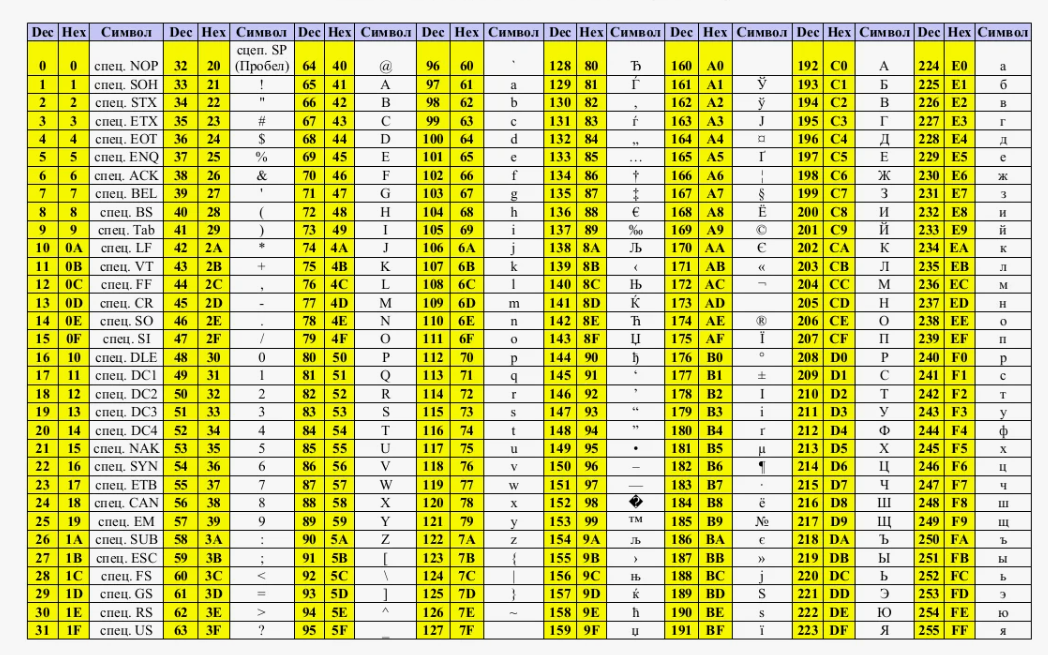


Рис 5.3.1 -«Таблица ASCII»

Согласно этой таблице E5 соответствует русская строчная «е». Я решил заменить этот байт на байт которому соответствует символ который нельзя нажать на клавиатуре.

Далее для удаления файла мы последовательно вычитаем из 256 индекс содержащийся в кластере и помещаем его туда.

В процессе восстановления файла мы делаем всё тоже самое.

## Чтение

Алгоритм чтения файлов:

Функция string чтение(string name)

{

Int a=0;

Пока(a!=144)

{

Если (RootDir[a]!=0)

{

Если(RootDir[a:a+6]==name)

{

claster= RootDir[a+7]

size=RootDir[a+8]

Всё пока

}

a\*=9

}

String txt=””

Для(int i=0; i<size;i++)

{

Txt+=clasters[claster]

Claster=myFAT[claster]

}

Вернуть txt

}

# Использование паттерна «Витрина данных»

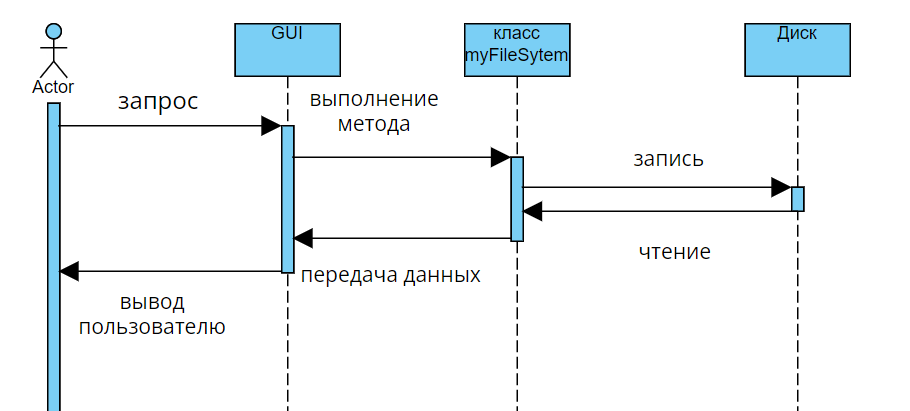


Рис 6.1-«Диограмма последовательности»

В программе использован паттерн витрина данных т.е часть данных дублированы в поля классов для более удобной работы с данными. У этого подхода есть как преимущества так и недостатки:

Достоинства:

* Программе не требуется жесткая связь с диском- нужно лишь иногда читать и записывать данные, что даёт возможность параллельно например читать содержимое диска hex редактором
* Более удобная обработка данных

Недостатки:

* Большие требования к оперативной памяти
* Этот подход не подходит для настоящих файловых систем так как использует довольно сложную модель взаимодействия с данными, за счет чего падает скорость.

Данный паттерн был выбран мной из-за более легкого тестирования ПО, и возможности анализировать диск в промежутках между исполнениями команд.

# Структура проекта

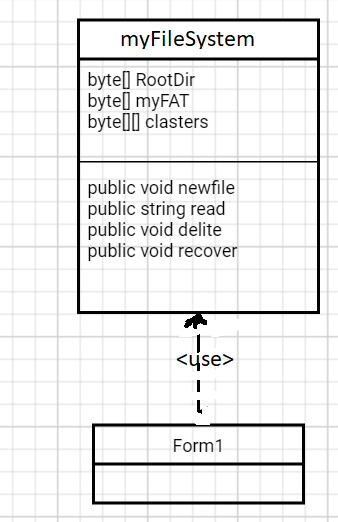


Рис 7.1-«UML- диаграмма»

Проект состоит из двух классов: myFileSystem- осуществляющий работу с диском и Form1 отвечающий за GUI

myFileSystem содержит много разных методов среди которых важнейшими являются:

Save- сохраняет изменения в файловой системе

Refresh- записывает данные диска в витрину данных

Formatting – форматирование диска

CountFreeClasters – считает свободные кластеры

NextFreeClaster – находит следующий кластер пригодный для записи

CleanClaster- зануляет все байты кластера

NewFile- создаёт новый файл

Read- считывает содержимое файла

Delete- удаление файла

CanBeRecovered- проверяет возможно ли восстановить файл

Recovery- восстановление файла

Display – возвращает список имен файлов

ReDisp- возвращает список имен удаленных файлов

ReadHex- возвращает строку показывающую все байты с диска

ReadChar- возвращает байты конвертированные в Char, где все 0 заменены на \*(т.к. байт 0 в c# string означает конец строки)

Form1 содержит методы связанные с работой GUI

UpdateFiles- обновляет все поля

DeleteItem- функция подвязанная к контекстному меню вызывает метод Delete класса myFileSystem для выбранного элемента

RecoverItem- функция подвязанная к контекстному меню вызывает метод Recover класса myFileSystem для выбранного элемента

ReadItem- функция подвязанная к контекстному меню вызывает метод Read класса myFileSystem для выбранного элемента

button1\_Click – поддержка создания файла

button2\_Click – поддержка форматирования

Как видно из перечисления методов Form1 и myFileSystem связаны зависимостью типа Dependency, так как первый использует второй.

# Тестирование

Протестируем функцию создания файла:

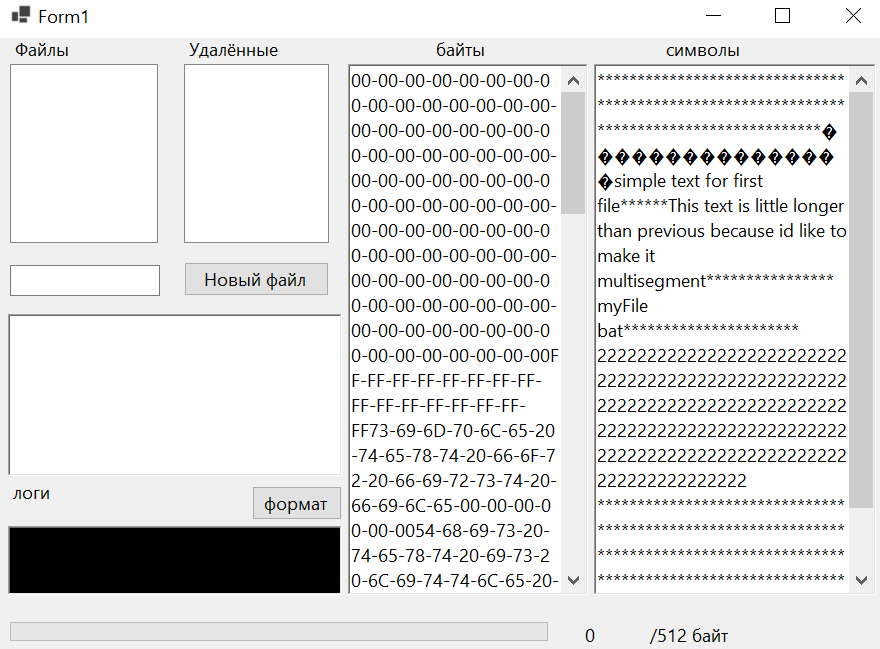


Рис 8.1а-«До создания файла»

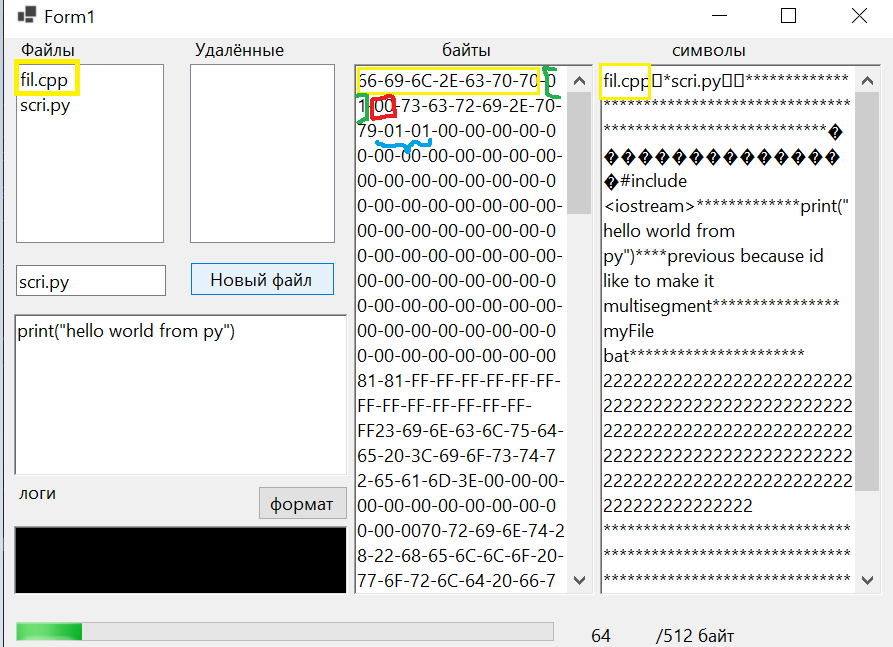


Рис 8.1б-«После создания файлов»

Как видно желтым обозначено имя файла, зелёным –размер в кластерах, красным –первый индекс таблицы FAT для данного файла это 0, и синим первый индекс и длинна второго файла как видно там тоже 1 и 1.

Создадим многокластерный файл.

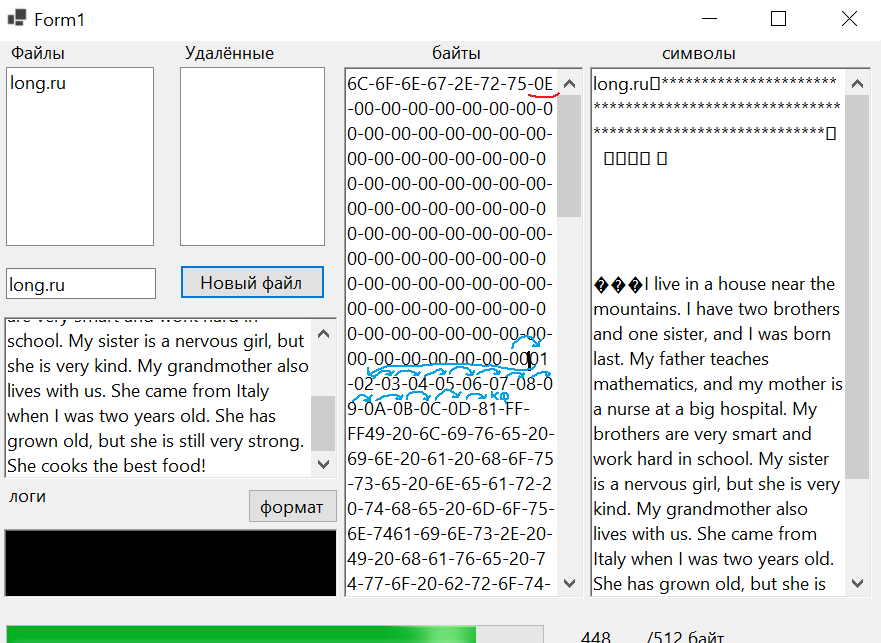


Рис 8.2-«многокластерная программа»

Как видно по числу подчеркнутому красным цветом в файле 14 сегментов, а синей стрелкой показана работа FAT таблицы.

Удалим файл

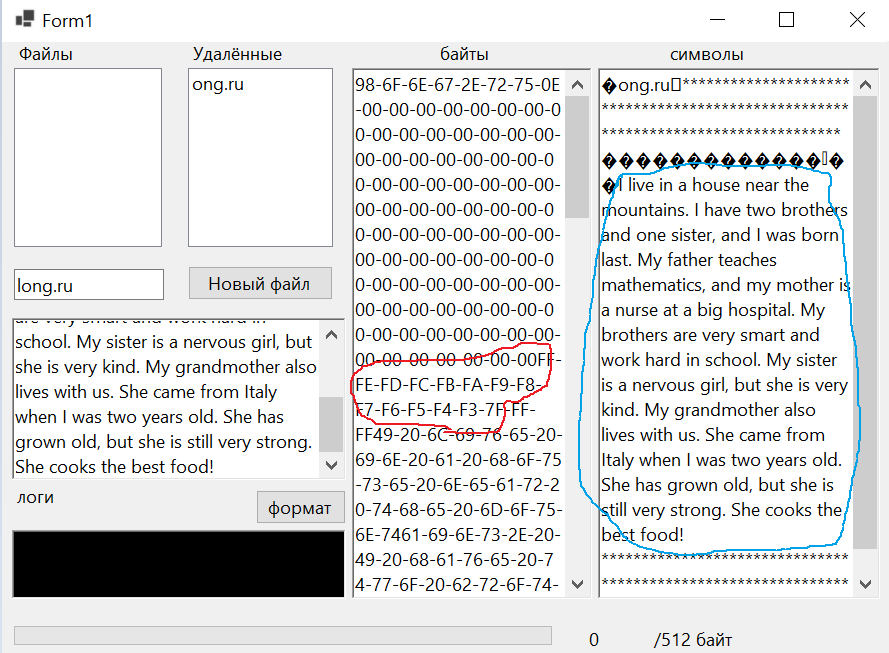


Рис 8.3-«Удалённая программа»

Как видно программа перенеслась в удалённые индексы инвертировались а данные в кластерах сохранены.

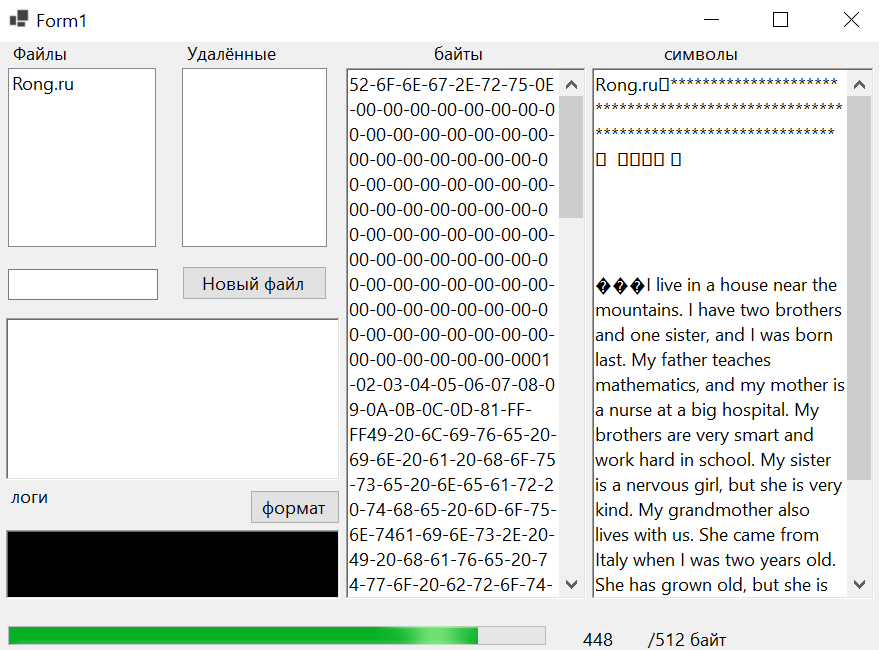


Рис 8.4-«Восстановленная программа»

Первая буква названия файла заменилась на R- это не баг всё так и задумывалось.

Контроль переполнения

Как видно из рисунка 8.5 программа не допускает переполнения выводя в поле логов too long

Также поле логов может предупредить если имя вводимого файла не соответствует стандартам программы. И сообщит если файл нельзя восстановить.

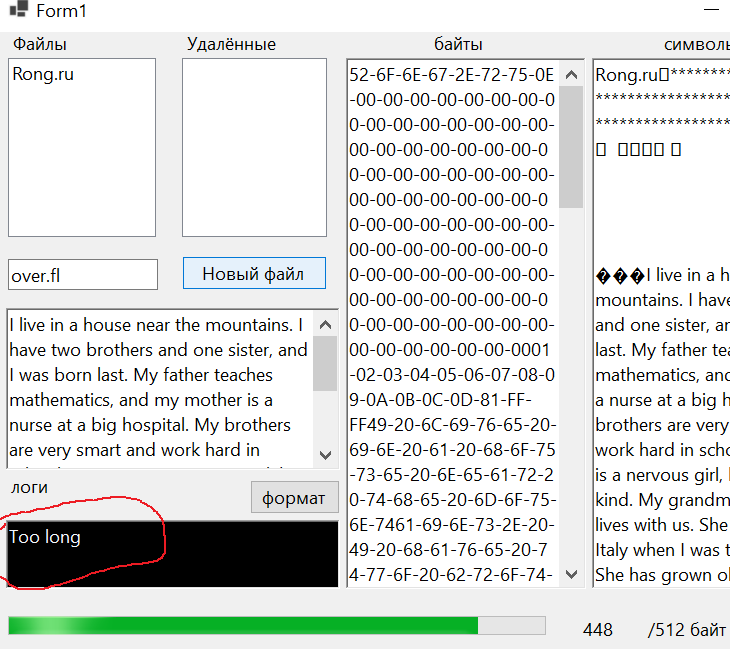


Рис 8.5-«Контроль переполнения программы»

Создадим фрагментированный файл

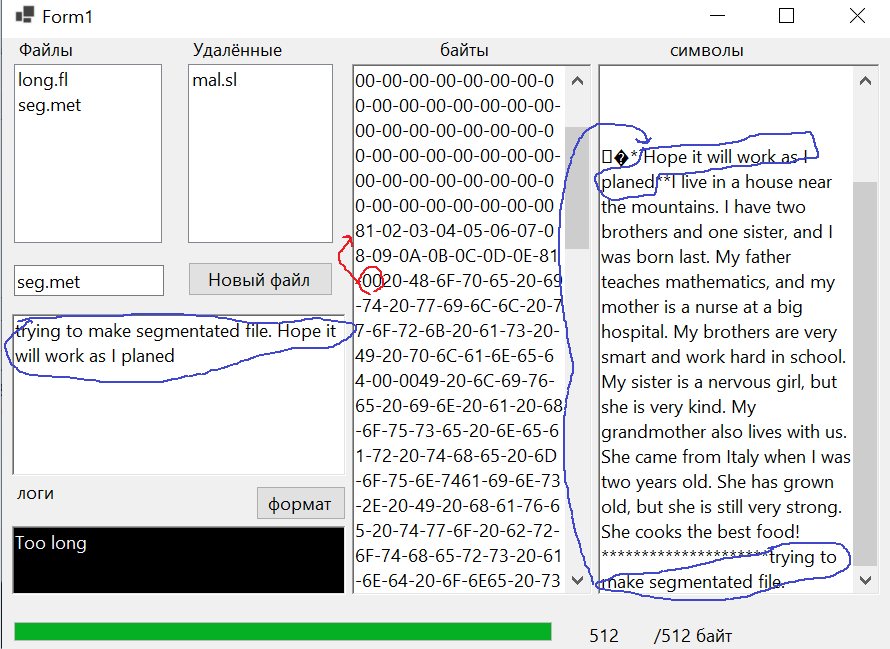


Рис 8.6-«Фрагментированный файл»

Был создан файл smal.sl из 1 сегмента далее был создан файл long.fl на 14 кластеров, удален файл smal.ml и создан файл seg.met который затер 0 сектор бывшего smal.ml. smal.ml уже не восстановить.



Рис 8.7-«mal.sl уже не восстановить»

# Вывод

В ходе курсовой работы мне удалось создать небольшую одноуровневую файловую систему, в которой можно создавать, читать, удалять и восстанавливать файлы.

# Список литературы

1. Andpro.ru
2. Blog.skillfactory.ru
3. Magicunraiser.com
4. Habr.com
5. Stackoverflow.com

# Приложение 1 код

Файл Form1.cs

namespace Fsys

{

public partial class Form1 : Form

{

myFileSystem fs = new myFileSystem(@"c:/app/fsys.iso");

public Form1()

{

InitializeComponent();

myFileSystem fs = new myFileSystem(@"c:/app/fsys.iso");

List<string> files = fs.Display();

listBox1.BeginUpdate();

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

{

listBox1.Items.Add(files[i]);

}

listBox1.EndUpdate();

ContextMenuStrip contextMenu = new ContextMenuStrip();

contextMenu.Items.Add("Удалить", null, DeleteItem);

contextMenu.Items.Add("Читать", null, ReadItem);

// Привязываем контекстное меню к ListBox

listBox1.ContextMenuStrip = contextMenu;

listBox2.BeginUpdate();

files = fs.ReDisp();

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

{

listBox2.Items.Add(files[i]);

}

listBox2.EndUpdate();

ContextMenuStrip contextMenuR = new ContextMenuStrip();

contextMenuR.Items.Add("Восстановить", null, RecoverItem);

// Привязываем контекстное меню к ListBox

listBox2.ContextMenuStrip = contextMenuR;

richTextBox2.Text = fs.ReadHex();

richTextBox3.Text = fs.ReadChar();

progressBar1.Minimum = 0;

progressBar1.Maximum = 16;

int occuped=16 -fs.FreeClasters(); ;

progressBar1.Value = occuped;

label6.Text = Convert.ToString(occuped \* 32);

}

private void UpdateFiles()

{

listBox1.Items.Clear();

List<string> files = fs.Display();

listBox1.BeginUpdate();

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

{

listBox1.Items.Add(files[i]);

}

listBox1.EndUpdate();

listBox2.Items.Clear();

files = fs.ReDisp();

listBox2.BeginUpdate();

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

{

listBox2.Items.Add(files[i]);

}

listBox2.EndUpdate();

richTextBox2.Text = fs.ReadHex();

richTextBox3.Text = fs.ReadChar();

progressBar1.Minimum = 0;

progressBar1.Maximum = 16;

int occuped = 16 - fs.FreeClasters(); ;

progressBar1.Value = occuped;

label6.Text = Convert.ToString(occuped \* 32);

}

private void DeleteItem(object sender, EventArgs e)

{

// Удаление выбранного элемента из ListBox

if (listBox1.SelectedIndex != -1)

{

string name = listBox1.SelectedItem.ToString();

fs.Delete(name);

}

UpdateFiles();

}

private void RecoverItem(object sender, EventArgs e)

{

// Удаление выбранного элемента из ListBox

if (listBox2.SelectedIndex != -1)

{

string name = listBox2.SelectedItem.ToString();

if (fs.CanBeRecovered(name))

{

fs.Recovery(name);

}

else

{

richTextBox4.Text = name + " cant be recovered";

}

}

UpdateFiles();

}

private void ReadItem(object sender, EventArgs e)

{

if (listBox1.SelectedIndex != -1)

{

string name = listBox1.SelectedItem.ToString();

string content = fs.Read(name);

richTextBox1.Text = content;

}

UpdateFiles();

}

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

{

string name = textBox1.Text;

if (name.Length != 7)

{

richTextBox4.Text = "Invalid name";

}

else

{

string content = richTextBox1.Text;

if (fs.convert((double)content.Length/32) > fs.FreeClasters())

{

richTextBox4.Text = "Too long";

}

else

{

fs.NewFile(name, content);

UpdateFiles();

}

}

}

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

{

fs.Formatting();

UpdateFiles();

}

}

}

Файл myFileSystem.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fsys

{

class myFileSystem

{

private byte[] RootDir;

private byte[] myFAT;

private byte[][] clasters;

string disk;

public myFileSystem(string disk)

{

this.disk = disk;

RootDir = new byte[90];

myFAT = new byte[16];

clasters = new byte[16][];

for (int i = 0; i < 16; i++)

{

this.clasters[i] = new byte[32];

}

this.Refresh();

}

public int convert(double a)

{

int b = Convert.ToInt32(a);

if (a > b) b++;

return b;

}

public void Save()

{

byte[] newData = new byte[618];

for (int i = 0; i < 90; i++)

{

newData[i] = this.RootDir[i];

}

for (int i = 90; i < 106; i++)

{

newData[i] = this.myFAT[i - 90];

}

int it = 106;

for (int j = 0; j < 16; j++)

{

for (int k = 0; k < 32; k++)

{

newData[it] = this.clasters[j][k];

it++;

}

}

ZipInputStream.WriteAllBytes(disk, newData);

}

public void Refresh()

{

byte[] data = ZipInputStream .ReadAllBytes(disk);

for (int i = 0; i < 90; i++)

{

this.RootDir[i] = data[i];

}

for (int i = 90; i < 106; i++)

{

this.myFAT[i - 90] = data[i];

}

int it = 106;

for (int j = 0; j < 16; j++)

{

for (int k = 0; k < 32; k++)

{

this.clasters[j][k] = data[it];

it++;

}

}

}

public void Formatting()

{

for (int i = 0; i < 16; i++)

{

this.myFAT[i] = 255;

}

for (int i = 0; i < 90; i++)

{

this.RootDir[i] = 0;

}

this.Save();

}

public int CountFreeClasters(int clasters)

{

int FreeClasters = 0;

bool flag = false;

for (int i = 0; i < 16; i++)

{

if (myFAT[i] == 255)

{

FreeClasters++;

}

}

if (!flag)

{

for (int i = 0; i < 16; i++)

{

if (myFAT[i] > 200)

{

FreeClasters++;

}

}

}

return FreeClasters;

}

public int NextFreeClaster()

{

int FirstClaster = 0;

bool flag = false;

for (int i = 0; i < 16; i++)

{

if (myFAT[i] == 255)

{

FirstClaster = i;

flag = true;

break;

}

}

if (!flag)

{

for (int i = 0; i < 16; i++)

{

if (myFAT[i] > 200)

{

FirstClaster = i;

flag = true;

break;

}

}

}

return FirstClaster;

}

public void CleanClaster(int claster)

{

for (int i = 0; i < 32; i++)

{

clasters[claster][i] = 0;

}

}

public void NewFile(string name, string content)

{

int size = this.convert((double)content.Length / 32);

int FirstClaster = NextFreeClaster();

char[] ChName = name.ToCharArray();

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] == 0)

{

for (int j = 0; j < 7; j++)

{

RootDir[a + j] = Convert.ToByte(ChName[j]);

}

RootDir[a + 7] = Convert.ToByte(size);

RootDir[a + 8] = Convert.ToByte(FirstClaster);

break;

}

}

int it = 0;

int clast = FirstClaster;

int num = 0;

int CurClaster = FirstClaster;

char[] ContentChar = content.ToCharArray();

this.CleanClaster(FirstClaster);

while (it < content.Length)

{

clasters[clast][num] = Convert.ToByte(ContentChar[it]);

num++;

it++;

if (num == 32)

{

num = 0;

myFAT[CurClaster] = 129;

clast = NextFreeClaster();

myFAT[CurClaster] = Convert.ToByte(clast);

clast = NextFreeClaster();

CurClaster = clast;

this.CleanClaster(clast);

}

}

myFAT[clast] = 129;

this.Save();

}

public string Read(string name)

{

int claster = 0;

int size = 0;

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] != 0 && RootDir[a] != 152)

{

string str = Encoding.UTF8.GetString([RootDir[a], RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

if (str == name)

{

size = RootDir[a + 7];

claster = RootDir[a + 8];

break;

}

}

}

string txt = "";

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

{

txt += Encoding.UTF8.GetString(clasters[claster]);

claster = myFAT[claster];

}

return txt;

}

public void Delete(string name)

{

int claster = 0;

int size = 0;

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] != 0 && RootDir[a] != 152)

{

string str = Encoding.UTF8.GetString([RootDir[a], RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

if (str == name)

{

size = RootDir[a + 7];

claster = RootDir[a + 8];

RootDir[a] = 152;

break;

}

}

}

int NextClaster;

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

{

NextClaster = myFAT[claster];

myFAT[claster] = (byte)(256 - myFAT[claster]);

claster = NextClaster;

}

this.Save();

}

public bool CanBeRecovered(string name)

{

int claster = 0;

int size = 0;

bool flag = false;

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] == 152)

{

string str = Encoding.UTF8.GetString([RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

if (str == name)

{

//RootDir[a] = 82;

size = RootDir[a + 7];

claster = RootDir[a + 8];

flag = true;

break;

}

}

}

int NextClaster;

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

{

NextClaster = myFAT[claster];

if(NextClaster <200 )

{

if (NextClaster != 127)

{

flag = false;

break;

}

}

//myFAT[claster] = (byte)(256 - myFAT[claster]);

claster = 256-NextClaster;

}

return flag;

}

public void Recovery(string name)

{

int claster = 0;

int size = 0;

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] == 152)

{

string str = Encoding.UTF8.GetString([RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

if (str == name)

{

RootDir[a] = 82;

size = RootDir[a + 7];

claster = RootDir[a + 8];

break;

}

}

}

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

{

myFAT[claster] = (byte)(256 - myFAT[claster]);

claster = myFAT[claster];

}

this.Save();

}

public List<string> Display()

{

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

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] != 0 && RootDir[a] != 152)

{

string str = Encoding.UTF8.GetString([RootDir[a], RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

list.Add(str);

}

}

return list;

}

public List<string> ReDisp()

{

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

for (int i = 0; i < 10; i++)

{

int a = i \* 9;

if (RootDir[a] == 152)

{

string str = Encoding.UTF8.GetString([ RootDir[a + 1], RootDir[a + 2], RootDir[a + 3], RootDir[a + 4], RootDir[a + 5], RootDir[a + 6]]);

list.Add(str);

}

}

return list;

}

public string ReadHex()

{

string hex= BitConverter.ToString(RootDir);

hex+= BitConverter.ToString(myFAT);

for(int i = 0;i < 16;i++)

{

hex+= BitConverter.ToString(clasters[i]);

}

return hex;

}

public string ReadChar()

{

for(int i=0;i<RootDir.Length;i++)

{

if (RootDir[i] ==0)

{

RootDir[i] = 42;

}

}

string Char = Encoding.Default.GetString(RootDir);

for (int i = 0; i < myFAT.Length; i++)

{

if (myFAT[i] == 0)

{

myFAT[i] = 42;

}

}

Char += Encoding.Default.GetString(myFAT);

for (int i = 0; i < 16; i++)

{

for (int j = 0; j < 32; j++)

{

if (clasters[i][j] == 0)

{

clasters[i][j] = 42;

}

}

Char += Encoding.Default.GetString(clasters[i]);

}

this.Refresh();

return Char;

}

public int FreeClasters()

{

int count = 0;

for (int i = 0; i < 16; i++)

{

if (myFAT[i] >200 || myFAT[i]==127)

{

count++;

}

}

return count;

}

}

}