МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

гвуз «донецкий Национальный Технический Университет»

КАФЕДРА ПРОГРАМНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

на тему: «Проектирование гипотетической операционной системы»

по курсу: «Операционные системы»

|  |  |
| --- | --- |
| Руководитель: | Выполнила: |
| \_\_\_\_\_\_\_\_ каф. ПИ | ст. гр. ПИ-16Б  Мамутова В.А. |
| Чернышева А.В. | “\_\_”\_\_\_\_\_\_ 2018 г. |
| “\_\_”\_\_\_\_\_\_ 2018 г. |  |
|  |  |
|  |  |

Донецк – 2018Реферат

Отчет по курсовому проекту содержит: 139 страниц, 23 рисунка, 25 таблиц, 18 приложений, 7 источников.

Объект исследования – операционные системы, их проектирование и эмуляция работы.

Задача исследования – изучить принципы работы операционных систем.

Цель исследования – выполнить обоснованный проект операционной системы, удовлетворяющей ряду специальных требований, приведенных в варианте задания, выполнить эмуляцию работы отдельных модулей спроектированной операционной системы.

Результат – проект операционной системы, состоящий из программы эмуляции файловой системы на языке программирования C# и программы для работы с межпроцессным взаимодействием на языке программирования Си.

Данная работа может быть использована в учебных целях для демонстрации работы файловой системы и межпроцессного взаимодействия.

ОПЕРАЦИОННАЯ СИСТЕМА, многопользовательская зашита, межпроцессное взаимодействие, КОМАНДНЫЙ ИНТЕРПРеТАТОР, ФАЙЛОВАЯ СИСТЕМА, ntfs, Эмуляция

Содержание

[Введение 5](#_Toc533593999)

[1 Структура файловой системы 6](#_Toc533594000)

[1.1 Общая организация файловой системы 6](#_Toc533594001)

[1.1.1 Структура записи Master File Table 9](#_Toc533594002)

[1.1.2 Структура данных служебного файла Volume 15](#_Toc533594003)

[1.1.3 Структура записи корневого каталога (/) 15](#_Toc533594004)

[1.1.4 Структура битовой карты свободных и занятых кластеров 15](#_Toc533594005)

[1.1.5 Структура списка пользователей системы (Users) 16](#_Toc533594006)

[1.1.6 Область файлов и директорий 16](#_Toc533594007)

[1.2 Виртуальные страницы 17](#_Toc533594008)

[1.3 Команды для работы с ФС 19](#_Toc533594009)

[1.4 Системные вызовы для работы с ФС 20](#_Toc533594010)

[1.5 Способ организации файлов 25](#_Toc533594011)

[1.6 Алгоритм работы некоторых системных вызовов ФС 26](#_Toc533594012)

[2 Процессы в ОС 31](#_Toc533594013)

[2.1 Команды для работы с процессами 32](#_Toc533594014)

[2.2 Системные вызовы управления процессами 33](#_Toc533594015)

[2.3 Диаграммы состояний процесса 35](#_Toc533594016)

[2.4 Приоритеты процессов 38](#_Toc533594017)

[2.5 Межпроцессное взаимодействие 39](#_Toc533594018)

[2.6 Выбор дисциплин обслуживания планировщика процессов. Алгоритм работы планировщика процессов в соответствии с выбранной дисциплиной обслуживания 44](#_Toc533594019)

[3 Режимы проектируемой ОС 46](#_Toc533594020)

[4.1 Мультипрограммный режим работы ОС 46](#_Toc533594021)

[4.2 Многопользовательская защита 47](#_Toc533594022)

[4.3 Интерактивный режим работы ОС 48](#_Toc533594023)

[4.4 Пакетный режим работы 49](#_Toc533594024)

[4 Структура операционной системы 50](#_Toc533594025)

[4.1 Общая структура проектируемой ОС 50](#_Toc533594026)

[4.2 Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами 51](#_Toc533594027)

[4.3 Структура управляющих блоков базы данных ОС 52](#_Toc533594028)

[5.1 Описание программных средств 59](#_Toc533594029)

[5.2 Разработка ФС 59](#_Toc533594030)

[5.3 Разработка командного интерпретатора 65](#_Toc533594031)

[5.4 Демонстрация работы с межпроцессным взаимодействием 69](#_Toc533594032)

[Приложение А Техническое задание 75](#_Toc533594033)

[Приложение Б. Руководство пользователя 78](#_Toc533594034)

[Приложение В. Экранные формы, отображающие результаты работ программ эмуляции 79](#_Toc533594035)

[Приложение Г. Листинг программы 87](#_Toc533594036)

# Введение

Операционная система (ОС) представляет собой комплекс управляющих и обрабатывающих программ, которые выполняют такие функции, как загрузка программ в оперативную память, их выполнение, управление оперативной памятью, управление доступом к данным. Также ОС обеспечивает стандартный доступ к периферийным устройствам, эффективно распределяет ресурсы вычислительной системы между процессами и т.д.

Предшественником ОС следует считать служебные программы (загрузчики и мониторы), а также библиотеки часто используемых подпрограмм, начавшие разрабатываться с появлением универсальных компьютеров 1-го поколения (конец 1940-х годов). Служебные программы минимизировали физические манипуляции оператора с оборудованием, а библиотеки позволяли избежать многократного программирования одних и тех же действий (осуществления операций ввода-вывода, вычисления математических функций и т.п.). В 1950-60-х годах сформировались и были реализованы основные идеи, определяющие функциональность ОС: пакетный режим, разделение времени и многозадачность, разделение полномочий, реальный масштаб времени, файловые структуры и файловые системы.

Для решения задач управления ресурсами разные ОС используют различные алгоритмы, особенности которых в конечном счете и определяют облик ОС. В число требований, предъявляемых сегодня к сетевым ОС, входят: функциональная полнота и эффективность управления ресурсами, модульность и расширяемость, переносимость и многоплатформенность, совместимость на уровне приложений и пользовательских интерфейсов, надежность отказоустойчивость, безопасность и производительность.

Целью данной курсовой работы является проектирование ОС, а также создание эмулятора, который будет демонстрировать работу файловой системы и реализовать межпроцессное взаимодействие процессов.

# 1 Структура файловой системы

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

* совокупность всех файлов на диске,
* наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
* комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

Если рассматривать диск с точки зрения операционной системы, то он представляет собой набор кластеров (логических единиц хранения данных). 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.

## Общая организация файловой системы

При проектировании файловой системы, в качестве основы была выбрана система NTFS (от англ. New Technology File System — «файловая система новой технологии»).

NTFS – файловая система, которая разрабатывалась Microsoft специально для Windows NT. Прежде всего, Windows NT – это ОС корпоративного уровня. Предыдущие файловые системы FAT и HPFS (ФС для OS/2) были способны удовлетворить потребностям рядовых пользователей, но не имели достаточных возможностей для корпоративной среды, таких как защищенность файлов, отказоустойчивость и восстанавливаемость. Устранить перечисленные недостатки призвана была новая операционная система NTFS. Это восстанавливаемая, защищенная и надежная файловая система, поддерживающая диски и файлы большого объёма. В этой файловой системе обеспечиваются множественные потоки данных, имена в UNICODE, универсальное средство индексации, переназначение плохих кластеров и поддержка POSIX. NTFS заимствовала лучшие свойства FAT и HPFS. От FAT система NTFS заимствовала философию “простота рождает эффективность”. От HPFS система NTFS унаследовала методы для повышения быстродействия и гибкости в файловой системе.

Из недостатков можно отметить существенные требования к оперативной памяти и то, что медленные диски и контроллеры сильно снижают быстродействия.

Учитывая все вышеизложенные плюсы и минусы, определение структуры файловой системы было сделано на основе NTFS.

Файловая система NTFS представляет собой выдающееся достижение структуризации: каждый элемент системы представляет собой файл - даже служебная информация. Самый главный файл в NTFS называется MFT (англ. Master File Table – «общая таблица файлов»). Он размещается в MFT-зоне и представляет собой централизованный каталог всех остальных файлов диска и себя самого. MFT-зона – служебное пространство, которое занимает первые 12% диска. MFT-зона всегда держится пустой - это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 88% диска представляют собой обычное пространство для хранения файлов.

MFT поделен на записи фиксированного размера (обычно 1 Кбайт), и каждая запись соответствует, какому-либо файлу (в общем смысле этого слова). Первые 16 файлов носят служебный характер и недоступны операционной системе - они называются метафайлами, причем самый первый метафайл - сам MFT. Эти первые 16 элементов MFT - единственная часть диска, имеющая фиксированное положение. Интересно, что вторая копия первых трех записей для надежности хранится ровно посередине диска. Остальной MFT-файл может располагаться, как и любой другой файл, в произвольных местах диска - восстановить его положение можно с помощью его самого, взяв за основу первый элемент MFT.

При проектировании файловой системы структура NTFS была упрощена.

MFT-пространство будет занимать 10 % от общего размера диска. Следовательно, при размере диска в 100 МБ MFT-зона будет занимать 100 \* 1024 \* 1024 \* 10 / 100 = 104 857 600 Б (размер тома) \* 0.1 (10 %) = 10 485 760 Б, при размере 200 МБ = 209 715 200 Б – 20 971 520 Б, при размере 400 МБ – 41 943 040 Б.

Размер Запись MFT-таблицы будет иметь фиксированный размер в 1 Кбайт (1024 байт). Всего в системе определено 6 служебных файлов, которые имеют фиксированное положение. Для большей наглядности и разъяснения данная структура представлена в таблице 1.1.

Таблица 1.1 - Логическая структура диска

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MFT-пространство | | | | | | |  | | |
| MFT-файл | | | | | |  |
| $MFT | $MFTMirr | $volume | / (корневой каталог) | $BitMap | $Users | … | Область данных | $MFTMirr (Копия первых записей $MFT) | Область данных |

Пояснения для служебных областей:

1. $MFT - сам MFT;
2. $MFTMirr - зеркальное отображение наиболее важных частей MFT. Используется для гарантирования доступа к MFT в случае повреждений в начале диска, в разрабатываемой файловой системе это копия первых 6 служебных записей MFT, размещенная посередине диска;
3. $Volume содержит служебную информацию – метка тома, версия файловой системы, состояние тома (повреждён/не повреждён), размер единицы распределения (кластера);
4. $/ (корневой каталог);
5. $BitMap – карта свободного места тома;
6. $Users – список пользователей.

Рассмотрим более подробно служебные файлы.

### Структура записи Master File Table

Сам MFT состоит из записей фиксированного размера по 1024 Б.

Условно структура записи делится на 2 части: заголовок и данные. Заголовок содержит 3 логических блока: Signature, File Name и Standard Information (см. табл. 1.2).

Таблица 1.2 – Структура записи файла $MFT, определяющей небольшой файл или каталог

|  |  |  |  |
| --- | --- | --- | --- |
| Header | | |  |
| Signature | File Name | Standard Information | Data |

Signature. Начало записи идентифицируется его сигнатурой - признаком записи, который указывает, используется данная запись или нет.

File Name. В данном поле определяется имя файла.

Standard information. В блоке содержится стандартная информация о файле: его атрибуты, расширение, размер, время создания и модификации файла, идентификатор владельца и права на файл.

Data. Здесь содержатся все данные файла.

Структура заголовка представлена в таблице 1.3.

Таблица 1.3 – Пример структуры записи $MFT

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Поле | Размерность | Назначение |
| Signature | Sign | 1 Б | Признак записи |
| string | FileName | 26 Б | Имя файла |
| int | Extension | 5 Б | Расширение |
| byte | Attributes | 1 Б | Атрибуты |
| int | Size | 4 Б | Размер |
| MyDateTime | CreationDate | 5 Б | Дата и время создания |
| MyDateTime | ModificationDate | 5 Б | Дата и время модификации |
| byte | UserId | 1 Б | User ID |
| Permission | Permissions | 2 Б | Права на файл |

Все указанные поля (первые три логические блока) в сумме занимают 50 байтов, значит под данные (Data) в одной записи выделяется 974 байта.

Поле «Признак записи» является перечислением и может иметь следующие значения, представленные в таблице 1.4.

Таблица 1.4 – Флаги для перечисления Признак записи

|  |  |
| --- | --- |
| Значение | Имя |
| 0 | NOT\_USED |
| 1 | IN\_USE |

Поля «Имя» и «Расширение» содержат набор символов, который представляет имя и расширение объекта файловой системы соответственно.

Предопределённые атрибуты файла представлены в таблице 1.5.

Таблица 1.5 – Атрибуты файла

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номера битов | 3-7 | 2 | 1 | 0 |
| Описание | Зарезервировано | Directory | Hidden | ReadOnly |

Если значение бита равно 1, файл обладает данным атрибутом. Если значение бита равно 0, файл не обладает данным атрибутом.

Поле «Размер файла» содержит целое число, указывающее на количество байт, которое занимает данный файл.

Поля «Дата и время создания» и «Дата и время модификации» содержат значения даты и полного времени создания и модификации файла соответственно. Для данных полей был определён собственный тип данных, представляющий массив из 5 байтов. Структура представлена в таблице 1.6.

Таблица 1.6 – Структура типа «MyDateTime»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Биты  Байты | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 4 | UTC | | | День | | | | |
| 3 | Месяц | | | | Год | | | |
| 2 | Год | | | | | | | Часы |
| 1 | Часы | | | | Минуты | | | |
| 0 | Минуты | | Секунды | | | | | |

Поле «User ID» содержит уникальный числовой идентификатор пользователя, который создал элемент файловой системы и является его владельцем.

Поле «Права доступа» состоит из двух байтов и содержит 3 набора битовых полей, которые представляют информацию о правах по каждой группе пользователей. В системе определено 3 группы пользователей: администраторы, владелец и все остальные. Структура данного поля представлена в таблице 1.7.

Таблица 1.7 – Структура поля «Права доступа»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номера битов | 12-16 | 8-11 | 4-7 | 0-3 |
| Тип пользователя | Зарезервировано | Администратор | Владелец | Все |

Набор из 4 битов, показывающих права конкретного типа пользователей, представлен в таблице 1.8.

Таблица 1.8 – Права пользователя

|  |  |  |
| --- | --- | --- |
| Тип прав | Бит | Описание |
| Полный доступ (Full Control) | 3 | Предоставляет полный доступ к файлу/папке, то есть допускаются все действия, предусмотренные всеми перечисленными ниже разрешениями. Дополнительно позволяется становиться владельцем файла/папки и изменять ее разрешения. |
| Изменение (Modify) | 2 | Разрешает все действия, которые позволяют выполнять «Чтение» и «Запись», а также разрешает изменение файла и удаление файла/папки. |
| Чтение (Read) | 1 | Разрешается чтение файла/просмотр вложенных папок и файлов, а также просмотр информации о них. |
| Запись (Write) | 0 | Разрешается запись файла/создание и размещение внутри папки новых файлов и подпапок, а также изменение информации о них (имени c расширением и атрибутов). |

Если значение бита равно 1, доступ по соответствующему критерию разрешён. Если значение бита равно 0, доступ запрещён. Если установлен в единицу бит, отвечающий за право изменения, то биты, отвечающие за права чтения и записи, также будет установлены в единицу. Аналогично, установленный в единицу бит прав полного доступа гарантирует установку битов, представляющих информацию о праве изменения, записи и чтения файла.

Если данные файла не помещаются в одну запись MFT, то атрибут Data будет нерезидентным (то есть таким, который не помещается в одну запись и находится в других местах раздела). В таком случае он будет хранить адресную информацию о блоках данных.

Под атрибут Data, как было отмечено выше, выделено 974 байта. Адрес блока данных представляет собой целочисленное значение и занимает 4 байта, значит запись MFT может хранить до [974 / 4] = 243 адресов блоков данных. Как следствие, размер элементов файловой системы ограничен, причём данное ограничение всецело зависит от размера блока. При размере одного блока, равному 1024 байт, размер элемента составляет 243 \* 1024 = 248 832 Б (если элемент – директория, то в ней могут храниться [248 832 / 36 (размер записи каталога)] = 6 912 объектов), при 2048 байт - 243 \* 2048 = 497 664 Б (13 824 объекта), при 4096 байт – 243 \* 4096 = 995 328 байтов (27 648 объектов).

Пример размещения большого файла приведён на рисунке 1.1.

|  |  |  |  |
| --- | --- | --- | --- |
| IN\_USE | File Name | Standard Information | Data  N1, N2, N3 |
| … | | | |
| … | | | |
| … | | | |
| Отрезок данных 1 | | | |
| … | | | |
| Отрезок данных 2 | | | |
| … | | | |
| … | | | |
| Отрезок данных 3 | | | |

Рисунок 1.1 – Размещение большого файла

### Структура данных служебного файла Volume

Структура данных файла $Volume представлена в таблице 1.9.

Таблица 1.9 – Структура $Volume

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Поле | Размерность | Назначение |
| string | VolumeName | 1 Б | Метка тома |
| string | FsVersion | 9 Б | Версия ФС |
| byte | State | 1 Б | Состояние |
| int | BytesPerCluster | 4 Б | Размер кластера |

### Структура записи корневого каталога (/)

Запись корневого каталога имеет следующую структуру, представленную в таблице. 1.10.

Таблица 1.10 – Структура записи корневого каталога

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Поле | Размерность | Назначение |
| string | FileName | 26 Б | Имя файла |
| int | Extension | 5 Б | Расширение |
| byte | Attributes | 1 Б | Атрибуты |
| int | MftEntry | 4 Б | Номер записи в MFT |

Каждая запись, находящаяся в каталоге, занимает 36 байтов.

### Структура битовой карты свободных и занятых кластеров

Битовая карта — служебная структура в сложных файловых системах, хранящая информацию о наличии и расположении свободного места. Битовая карта будет представлена битовым массивом, в котором каждому кластеру будет соответствовать 2 бита. Возможные значения состояния кластера представлены в таблице 1.11.

Таблица 1.11 – Значения битовой карты свободных и занятых кластеров

|  |  |
| --- | --- |
| Значение | Описание |
| 00 | свободен |
| 01 | повреждён (Bad-кластер) |
| 10 | служебный |
| 11 | занят |

### Структура списка пользователей системы (Users)

Запись в списке пользователей представлена в таблице 1.12.

Таблица 1.12 – Структура записи в списке пользователей

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Поле | Размерность | Назначение |
| string | Name | 26 Б | Имя пользователя |
| bool | IsAdministrator | 1 Б | Признак администратора |
| byte | Id | 1 Б | Уникальный идентификатор |
| byte[] | PasswordHash | 64 Б | Зашифрованный пароль |
| int | HomeDirectoryMftEntry | 4 Б | Номер Mft-записи домашней директории пользователя |

Общий размер одной записи – 96 Б.

### Область файлов и директорий

В разрабатываемой операционной системе данная область содержит в себе различные файлы. Размерность данной области - 90 % от области диска, то есть для размера тома, равного 100 МБ, данная область равна 90 МБ (94 371 840 Б), для 200 МБ – 180 МБ, для 400 МБ – 360 МБ, для 800 МБ – 720 МБ.

## Виртуальные страницы

Виртуальная память используется для обеспечения выполнения программ, требующих больше оперативной памяти, чем имеется в компьютере, а также для выделения каждому процессу собственного виртуального адресного пространства.

Данная технология разработана для многозадачных операционных систем. Она позволяет увеличить эффективность использования памяти несколькими одновременно работающими программами, организовав множество независимых адресных пространств, и обеспечить защиту памяти между различными приложениями. Использование виртуальной памяти упрощает и программирование.

Применение механизма виртуальной памяти позволяет:

* упростить адресацию памяти клиентским программным обеспечением;
* рационально управлять оперативной памятью компьютера (хранить в ней только активно используемые области памяти);
* изолировать процессы друг от друга.

В настоящее время эта технология имеет аппаратную поддержку на всех современных процессорах. В то же время во встраиваемых системах и в системах специального назначения, где требуется очень быстрая работа или есть ограничения на длительность отклика, виртуальная память используется относительно редко.

Современные процессоры, как правило, поддерживают объединение адресного пространства в области переменного размера (сегменты) и области фиксированного размера (страницы). В разработанной ОС используется страничная организация виртуальной памяти.

Страничный механизм обеспечивает гораздо большую гибкость, чем механизм сегментации. В нём все виртуальное адресное пространство разделено на блоки одинакового размера (а не переменного, как в случае сегментов), которые называются страницами. Так же, как и в случае сегментации, страница может либо присутствовать в оперативной памяти, либо находиться в области свопинга, или исполняемом файле процесса. Основное же преимущество такой схемы заключается в том, что система управления памятью оперирует областями достаточно малого размера для обеспечения эффективного распределения ресурсов памяти между процессами. Страничный механизм допускает, чтобы часть сегмента находилась в оперативной памяти, а часть отсутствовала. Это дает ядру возможность разместить в памяти только те страницы, которые в данное время используются процессом, тем самым значительно освобождая оперативную память. Еще одним преимуществом является то, что страницы сегмента могут располагаться в физической памяти в произвольном месте и порядке, что позволяет эффективно использовать свободное пространство.

При использовании страничного механизма линейный адрес, полученный в результате сложения базового адреса сегмента и смещения, также является логическим адресом, который дополнительно обрабатывается блоком страничной трансляции процессора. В этом случае линейный адрес рассматривается процессором как состоящий из трех частей, показанных на рисунок 1.2.

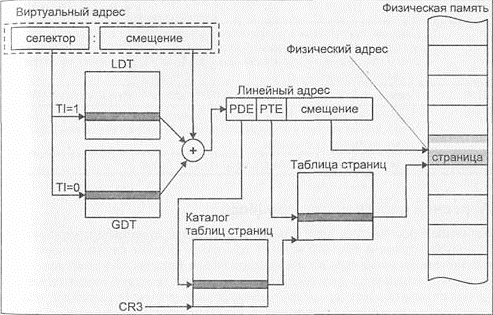


Рисунок 1.2 – Трансляция адреса с использованием страничного механизма

Процессор использует поле PDE (Page Directory Entry) адреса (старшие 10 бит) в качестве индекса в каталоге таблиц. Он имеет длину, равную одной странице, и содержит до 1024 указателей на таблицы страниц (page table). Второе поле линейного адреса, РТЕ (Page Table Entry), позволяет процессору выбрать нужный элемент таблицы, адресующий физическую страницу. Складывая адрес начала страницы со смещением, хранящимся в третьем поле и занимающем младшие 12 бит, процессор получает 32-битный физический адрес.

## Команды для работы с ФС

Разрабатываемая файловая система поддерживает команды, описание которых приведено в таблице 1.13.

Таблица 0.13 - Команды для работы с ФС

|  |  |  |
| --- | --- | --- |
| Имя команды | Параметры | Описание |
| createfile | fileName | Создаёт новый пустой файл с именем fileName |
| rename | fileName1, fileName2 | Переименование файла с именем fileName1 в файл с именем fileName2 |
| delete | fileName | Удаляет файл с именем fileName |
| copy | name, path | Копирует файл с именем fileName в path |

Так как разрабатываемая файловая система имеет графический интерфейс, то в ней нет вызовов таких команд напрямую, однако работа с элементами управления интерфейса подразумевает эмуляцию их вызова и выполнения.

Графические примеры вышеописанных команд создания и переименования файла представлены в приложениях В.4-В.6.

## Системные вызовы для работы с ФС

Системные вызовы – это способ вызова функции ядра ОС, это обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции. Системный вызов обычно выглядит как вызов подпрограммы или функции из системной библиотеки. Однако системный вызов как частный случай вызова такой функции или подпрограммы следует отличать от более общего обращения к системной библиотеке, поскольку последнее может и не требовать выполнения привилегированных операций.

Для работы с файловой системой разработаны системные вызовы, описание которых приведено в таблице 1.14.

Таблица 0.14 - Системные вызовы для работы с ФС

|  |  |  |  |
| --- | --- | --- | --- |
| Вызов | Параметры | Результат | Описание |
| Create | Path path, string name, string extension, MftHeader.Attribute attributes = MftHeader.Attribute.None, int userId = -1 | int mftEntry | Создание пустого файла или директории в зависимости от переданных атрибутов |
| Copy | Path path, DirectoryRecord record | void | Копирование в буфер обмена файл по указанному расположению |
| Paste | Path path | void | Вставка файла из буфера обмена в указанную директорию |
| CopyRecursively | Path sourcePath, Path destinationPath | List<DirectoryRecord> records | Копирование всего содержимого из указанного в целевой путь |
| Rename | Path path, DirectoryRecord record, string newFullName | void | Переименование указанного файла, находящегося в заданном расположении |
| Delete | Path path, DirectoryRecord record | void | Удаление файла по указанному пути |
| DeleteRecursively | Path directoryPath | int fileCount | Удаление содержимого из указанного расположения |
| Save | DirectoryRecord record, byte[] fileData | void | Сохранение данных файла |
| HasPermissions | UserRecord user, Permission.Rights rights | boot hasPermission | Проверка на разрешения к файлу |
| UpdateMftEntry | MftHeader mftHeader, int mftEntry | void | Обновление записи MFT по заданному номеру (удаление, добавление, изменение) |
| UpdateRecord<T> where T : IRecord | int targetMftEntry,  T before, T after) | void | Обновление записи в файле по указанному MFT-номеру (удаление, добавление, изменение) |
| GetRecord | string name, int targetMftEntry, int recordLength | IRecord | Получение записи из записи MFT с указанным номером |
| FindDirectoryMft Entry | Path path | int mftEntry | Получение MFT-номера конечной директории в полном пути |
| GetNewMftEntry | - | int mftEntry | Получение первого незанятого номера для новой Mft-записи |
| TryToAllocate Memory | int targetMftEntry, int recordLength | MemoryAllocation attributes | Получение атрибутов размещения новой записи в памяти |
| AllocateRecord | MemoryAllocation allocation, int targetMftEntry, int recordLength | void | Подготовка к записи нового файла: увеличение размеров необходимых файлов с учётом атрибутов распределения нового файла |
| GetFreeClusters | int requiredNumber | List<int> numbers | Получение заданное количество свободных кластеров |
| SetClustersState | List<int> clusterNumbers, ClusterState state | void | Установление состояния кластеров |
| WriteCluster Numbers | int mftEntry, int[] clusters | void | Запись в MFT адресов кластеров для указанного файла |
| ReadMftData | int mftEntry | Data dataAttributes | Чтение размера файла и массива кластеров |
| ReadFileData | int mftEntry | byte[] data | Чтение данных файла |
| WriteFileData | int mftEntry, byte[] fileData | void | Запись данных файла |
| SignUp | string name, string password, bool isAdministrator | void | Авторизация пользователя |
| GetUserById | int uid | UserRecord | Получение пользователя по его уникальному идентификатору |
| GetAvailableId | - | int | Получение первого доступного id для нового пользователя |
| ChangeOwnerToAdministrator | byte id | - | Присвоение всех файлов владельца в указанным id администратору |

Пояснение некоторых параметров:

Path – объект класса, содержащий полный путь;

MftHeader – объект класса, содержащий заголовок MFT-записи;

MftHeader.Attribute – перечисление атрибутов MFT-записи;

Permission.Rights – перечисление возможных прав пользователя;

IRecord – интерфейс для записи каталога и записи в списке пользователей;

DirectoryRecord - объект класса, содержащий структуру записи каталога (реализует интерфейс IRecord);

UserRecord – объект класса, содержащий структуру записи в списке пользователей (реализует интерфейс IRecord);

Data – объект класс, содержащий атрибут Data MFT-записи (размер файла и список адресов кластеров).

MemoryAllocation – объект класса, содержащий атрибуты размещения новой записи на диске (номер MFT-записи и дополнительные флаги: необходимо ли увеличить размер MFT-зоны, необходимо ли увеличить размер родительской директории файла, необходимо ли выделить новый кластер;

ClusterState – перечисление возможных состояний кластера.

## Способ организации файлов

Файл – именованная область данных на носителе информации. Файловая система поддерживает файлы с последовательным доступом, для организации хранения файлов на диске используется метод последовательного размещения и считывания записей.

Смысл последовательного доступа заключается в том, что в каждый момент времени доступна лишь одна компонента из всей последовательности. Для того, чтобы обратиться к компоненте с номером К, необходимо просмотреть от начала файла К-1 предшествующую компоненту. После обращения к компоненте с номером К можно обращаться к компоненте с номером К+1. Отсюда следует, что процессы формирования (записи) компонент файла и просмотра (чтения) не могут произвольно чередоваться. Таким образом, файл вначале строится при помощи последовательного добавления компонент в конец, а затем может последовательно просматриваться от начала до конца.

## Алгоритм работы некоторых системных вызовов ФС

Далее рассматривается выполнение следующих системных вызовов: создание, удаление и переименование записи.

Для наглядности представления алгоритмов был введён символ «(!)». В программе был разработан класс-наследник Exception (FsException) для продвинутой обработки исключений. Пометка «(!)» в конце строк алгоритмов означает возможный выброс данного исключения с кодом ошибки, которая будет обработан выше по иерархии.

Алгоритм системного вызова «Create», который выполняет операцию создания новой записи, изображен на рисунке 1.8.

Алгоритм Create

Входная информация:

(1) путь к папке, в которой необходимо создать запись

(2) имя записи

(3) расширение

(4) атрибуты

Выходная информация:

(1) номер записи MFT

{

Создаём заголовок MFT с указанными параметрами;

Получаем MFT-запись родительской директории по указанному пути;

Проверяем права на запись родительской директории; (!)

Получаем атрибуты распределения памяти в MFT-файле и в родительской директории для новой записи; (!)

Проверяем на существование файла с заданным именем по указанному пути; (!)

Добавляем новую запись в MFT;

Увеличиваем размеры MFT-файла, директории в соответствии с атрибутами распределения памяти;

Добавляем запись в родительскую директорию на пустое место.

}

Рисунок 1.3 - Алгоритм «Create»

В начале алгоритма создаётся заголовок новой MFT-записи с заданными параметрами (атрибуты определяют, какой элемент создаётся – файл или папка). Далее происходит поиск MFT-записи родительской директории по указанному пути (целевая директория – это последний элемент в пути). Проверяются права на доступ к директории. Если нет прав на запись, выбрасывается FsException с кодом ошибки и операция системного вызова завершается. Далее идёт определение атрибутов распределения памяти: 1) определяется номер первой неиспользуемой MFT-записи; 2) при необходимости устанавливается атрибут увеличения размера MFT-файла; если он не будет попадать в MFT-зону, выбрасывается FsException с кодом нехватки системного пространства; 3) если нет свободного места в директории, то устанавливается атрибут увеличения размера директории; 4) если этот атрибут был установлен, то выполняется проверка на попадание увеличенного размера директории в текущий блок данных (либо в атрибут Data в MFT-записи, либо в кластер в области файлов); 5) если проверка завершилась неуспешно, то проверяется количество свободных кластеров; если их не хватает, также выбрасывается FsException. Далее проверяется существование записи с данным полным именем в указанном расположении, если проверка была успешной, выбрасывается исключение FsException с кодом ошибки существования элемента с заданным именем. Если все предыдущие операции закончились успешно, добавляем созданный MFT-заголовок в MFT, увеличиваем при необходимости размеры директории и MFT и добавляем в директорию новую запись.

Алгоритм системного вызова «Delete», который выполняет операцию удаления записи, изображен на рисунке 1.4.

Алгоритм Delete

Входная информация:

(1) путь к папке, из которой необходимо удалить запись

(2) удаляемая запись

Выходная информация:

{

Получаем MFT-запись родительской директории по указанному пути;

Проверяем права на запись родительской директории; (!)

Получаем целевую MFT-запись;

Проверяем права на изменение удаляемой записи; (!)

Если (удаляемая запись – директория)

{

Рекурсивно удаляем содержимое директории;

Проверяем, совпадает ли количество удалённых элементов с количеством исходных дочерних элементов; (!)

Если (удалена домашняя директория пользователя)

Устанавливаем в -1 номер MFT-записи директории в заявке;

}

Очищаем кластеры;

Удаляем запись в родительской директории;

Ставим пометку о неиспользуемой записи MFT;

Изменяем время последней модификации родительской директории.

}

Рисунок 1.5 - Алгоритм «Delete»

В начале алгоритма получаем MFT-запись родительской директории по указанному пути (целевая директория – это последний элемент в пути). Проверяем права на запись данного элемента. В случае неуспешной проверки, выбрасывается FsException с кодом отсутствия требуемых прав. Далее получаем целевую MFT-запись для удаления. Аналогично, при отсутствии прав на изменение данного элемента, выбрасывается исключение FsException с кодом ошибки отсутствия требуемых прав. Если удаляемая запись является директорией, удаляем содержимое, получая при этом количество удалённых файлов в директории. Сравниваем это число с общим числом файлов директории. Если они не равны, выводим уведомление. Далее проверяем, не является ли удаляемая директория домашней директорией одного из пользователей. Если да, то изменяем домашнюю директорию на корневой каталог. Если все предыдущие операции закончились успешно, очищаем кластеры, занимаемые данной записью, удаляем данную запись из родительской директории, ставим пометку о неиспользуемой MFT-записи и изменяем время последней модификации этой родительской директории.

Алгоритм системного вызова «Rename», который выполняет операцию переименования записи, изображен на рисунке 1.6.

Алгоритм Rename

Входная информация:

(1) путь, в котором находится запись

(2) запись, которую необходимо переименовать

(3) новое полное имя

Выходная информация:

{

Получаем MFT-запись родительской директории по указанному пути;

Проверяем права на запись родительской директории; (!)

Получаем целевую MFT-запись

Проверяем права на запись заданного элемента; (!)

Проверяем на существование файла с заданным именем по указанному пути; (!)

Изменяем полное имя записи;

Обновляем MFT-запись;

Обновляем запись в родительской директории;

Изменяем время последней модификации родительской директории.

}

Рисунок 1.6 - Алгоритм «Rename»

В начале алгоритма получаем MFT-запись родительской директории по указанному пути (целевая директория – это последний элемент в пути). Проверяем права на запись данного элемента. В случае неуспешной проверки, выбрасывается FsException с кодом отсутствия требуемых прав. Далее получаем целевую MFT-запись. Аналогично, при отсутствии прав на запись данного элемента, выбрасывается исключение FsException с кодом ошибки отсутствия требуемых прав. Затем проверяется существование записи с данным полным именем в указанном расположении, если проверка была успешной, выбрасывается исключение FsException с кодом ошибки существования элемента с заданным именем. Если все предыдущие операции закончились успешно, изменяем полное имя, обновляем MFT-запись и данную запись в родительской директории и изменяем время последней модификации этой директории.

# 2 Процессы в ОС

Процесс – это совокупность программного кода и данных, загруженных в память ЭВМ. На первый взгляд процесс — это запущенная программа (приложение) или команда. Но это не совсем так. Некоторые приложения могут создавать несколько процессов одновременно.

Основной задачей любой операционной системы является управление процессами. Операционная система должна распределять между ними ресурсы, предоставлять им возможность совместно использовать информацию и обмениваться ею, защищать ресурсы, используемые одним процессом от их использования другими процессами, а также обеспечивать возможность синхронной работы процессов. Для этого операционная система должна поддерживать для каждого процесса свою структуру данных, в которой задается состояние данного процесса, и указываются ресурсы, которыми он владеет. Это позволяет операционной системе осуществлять управление процессами.

Каждому процессу в системе назначаются числовые идентификаторы (личные номера) в диапазоне от 1 до 65535 (PID - Process Identifier - идентификатор процесса) и идентификаторы родительского процесса (PPID - Parent Process Identifier - идентификатор родительского процесса). PID является именем процесса, по которому мы можем адресовать процесс в операционной системе при использовании различных средств просмотра и управления процессами. PPID определяет родственные отношения между процессами, которые, в значительной степени, определяют его свойства и возможности.

## Команды для работы с процессами

Рассмотрим команды для работы с процессами на основе операционной системы Linux.

Процесс в Linux (как и в UNIX) - это программа, которая выполняется в отдельном виртуальном адресном пространстве. Когда пользователь регистрируется в системе, автоматически создается процесс, в котором выполняется оболочка (shell), например, /bin/bash.

Команды для работы с процессами приведены в таблице 2.1.

Таблица 2.1 – Команды для работы с процессами

|  |  |
| --- | --- |
| Команда | Описание |
| kill (pid) | Уничтожает процесс с указанным идентификатором (pid). |
| ps [-t размер кванта] | Отображает список текущих процессов. Команда может работать в режиме трассировки планировщика, при этом надо указать размер кванта. |
| nice (pid) | Изменяет поправку приоритета процесса с идентификатором (pid) на указанное значение. Поправка находится в диапазоне от 0 до 10. |
| renice [[-p] PID] [[-g] grp] [[-u] user] | Служит для изменения значения nice для уже выполняющихся процессов |
| nohup команда & | Запуск процесса в фоновом режиме |

## Системные вызовы управления процессами

Для порождения нового процесса (процесс-потомок) используется системный вызов fork(). Формат вызова: int fork().

Порожденный таким образом процесс представляет собой точную копию своего процесса-предка. Единственное различие между ними заключается в том, что процесс-потомок в качестве возвращаемого значения системного вызова fork() получает 0, а процесс-предок - идентификатор процесса-потомка. Кроме того, процесс-потомок наследует и весь контекст программной среды, включая дескрипторы файлов, каналы и т.д. Наличие у процесса идентификатора дает возможность и ОС UNIX, и любому другому пользовательскому процессу получить информацию о функционирующих в данный момент процессах.

Далее рассмотрим подробнее алгоритм работы вызова fork():

1. Выделяется память для описателя нового процесса в таблице процессов.
2. Назначается идентификатор процесса PID.
3. Создается логическая копия процесса, который выполняет fork() - полное копирование содержимого виртуальной памяти родительского процесса, копирование составляющих ядерного статического и динамического контекстов процесса-предка.
4. Увеличиваются счетчики открытия файлов (порожденный процесс наследует все открытые файлы родительского процесса).
5. Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 - в процессе-потомке.

Ожидание завершения процесса-потомка родительским процессом выполняется с помощью системного вызова wait(): int wait(int \*status).

В результате осуществления процессом системного вызова wait() функционирование процесса приостанавливается до момента завершения порожденного им процесса-потомка. По завершении процесса-потомка процесс-предок пробуждается и в качестве возвращаемого значения системного вызова wait() получает идентификатор завершившегося процесса-потомка, что позволяет процессу-предку определить, какой из его процессов-потомков завершился (если он имел более одного процесса-потомка). Аргумент системного вызова wait() представляет собой указатель на целочисленную переменную status, которая после завершения выполнения этого системного вызова будет содержать в старшем байте код завершения процесса-потомка, установленный последним в качестве системного вызова exit(), а в младшем - индикатор причины завершения процесса-потомка.

Формат системного вызова exit(), предназначенного для завершения функционирования процесса: void exit(int status).

Аргумент status является статусом завершения, который передается отцу процесса, если он выполнял системный вызов wait().

Для получения собственного идентификатора процесса используется системный вызов getpid(), а для получения идентификатора процесса-отца - системный вызов getppid(): int getpid(), int getppid().

Вместе с идентификатором процесса каждому процессу в ОС UNIX ставится в соответствие также идентификатор группы процессов. В группу процессов объединяются все процессы, являющиеся процессами-потомками одного и того же процесса. Организация новой группы процессов выполняется системным вызовом getpgrp(), а получение собственного идентификатора группы процессов - системным вызовом getpgrp(). Их формат: int setpgrp(), int getpgrp().

Для завершения процесса используется вызов команды kill(pid).

Кроме завершения, процесс можно усыпить на определенное время. Для этого есть команда sleep(ms).

Если процесс уснул в ожидании какого-то ресурса и ресурс освободился, для пробуждения используется команда wakeup().

## Диаграммы состояний процесса

Жизненный цикл процесса может быть разбит на несколько состояний. Переход процесса из одного состояния в другое происходит в зависимости от наступления тех или иных событий в системе. На рисунке 2.1 показаны состояния, в которых процесс может находиться с момента создания до завершения выполнения.

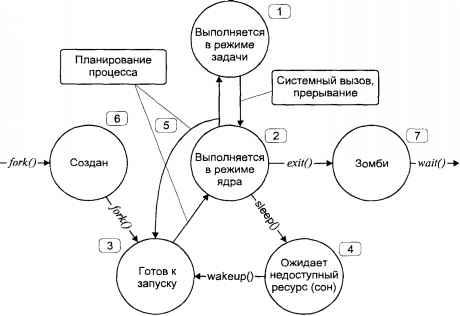


Рисунок 2.1 – Диаграмма состояний процесса

1. Процесс выполняется в режиме задачи. При этом процессором выполняются прикладные инструкции данного процесса.
2. Процесс выполняется в режиме ядра. При этом процессором выполняются системные инструкции ядра операционной системы от имени процесса.
3. Процесс не выполняется, но готов к запуску, как только планировщик выберет его (состояние runnable). Процесс находится в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме вычислительных.
4. Процесс находится в состоянии сна (asleep), ожидая недоступного в данный момент ресурса, например, завершения операции ввода/вывода.
5. Процесс возвращается из режима ядра в режим задачи, но ядро прерывает его и производит переключение контекста для запуска более высокоприоритетного процесса.
6. Процесс только что создан вызовом fork() и находится в переходном состоянии: он существует, но не готов к запуску и не находится в состоянии сна.
7. Процесс выполнил системный вызов exit() и перешел в состояние зомби (zombie, defunct). Как такового процесса не существует, но остаются записи, содержащие код возврата и временную статистику его выполнения, доступную для родительского процесса. Это состояние является конечным в жизненном цикле процесса.

Необходимо отметить, что не все процессы проходят через все множество состояний, приведенных выше.

Процесс начинает свой жизненный путь с состояния 6, когда родительский процесс выполняет системный вызов fork(). После того как создание процесса полностью завершено, процесс завершает "дочернюю часть" вызова fork() и переходит в состояние 3 готовности к запуску, ожидая своей очереди на выполнение. Когда планировщик выбирает процесс для выполнения, он переходит в состояние 1 и выполняется в режиме задачи.

Выполнение в режиме задачи завершается в результате системного вызова или прерывания, и процесс переходит режим ядра, в котором выполняется код системного вызова или прерывания. После этого процесс опять может вернуться в режим задачи. Однако во время выполнения системного вызова в режиме ядра процессу может понадобиться недоступный в данный момент ресурс. Для ожидания доступа к такому ресурсу, процесс вызывает функцию ядра sleep() и переходит в состояние сна (4). При этом процесс добровольно освобождает вычислительные ресурсы, которые предоставляются следующему наиболее приоритетному процессу. Когда ресурс становится доступным, ядро "пробуждает процесс", используя функцию wakeup(), помещает его в очередь на выполнение, и процесс переходит в состояние «готов к запуску».

При предоставлении процессу вычислительных ресурсов происходит переключение контекста (context switch), в результате которого сохраняется образ, или контекст, текущего процесса, и управление передается новому. Переключение контекста может произойти, например, если процесс перешел в состояние сна, или если в состоянии готовности к запуску находится процесс с более высоким приоритетом, чем текущий. В последнем случае ядро не может немедленно прервать текущий процесс и произвести переключение контекста. Дело в том, что переключению контекста при выполнении в режиме ядра может привести к нарушению целостности самой системы. Поэтому переключение контекста откладывается до момента перехода процесса из режима ядра в режим задачи, когда все системные операции завершены, и структуры данных ядра находятся в нормальном состоянии.

Таким образом, после того как планировщик выбрал процесс на запуск, последний начинает свое выполнение в режиме ядра, где завершает переключение контекста. Дальнейшее состояние процесса зависит от его предыстории: если процесс был только что создан или был прерван, возвращаясь в режим задачи, он немедленно переходит в этот режим. Если процесс начинает выполнение после состояния сна, он продолжает выполняться в режиме ядра, завершая системный вызов. Заметим, что такой процесс может быть прерван после завершения системного вызова в момент перехода из режима ядра в режим задачи, если в очереди существует более высокоприоритетный процесс.

Наконец, процесс выполняет системный вызов exit() и заканчивает свое выполнение. Процесс может быть также завершен вследствие получения сигнала. В обоих случаях ядро освобождает ресурсы, процессу, за исключением кода возврата и статистики его выполнения, и переводит процесс в состояние "зомби". В этом состоянии процесс находится до тех пор, пока родительский процесс не выполнит один из системных вызовов wait(), после чего вся информация о процессе будет уничтожена, а родитель получит код возврата завершившегося процесса.

## Приоритеты процессов

Приоритет — параметр, определяющий преимущества процесса в обладании ресурсами по сравнению с другими процессами. В разрабатываемой системе поддерживаются абсолютные, относительные и статические приоритеты.

Существуют две разновидности приоритетного планирования: обслуживание с относительными приоритетами и обслуживание с абсолютными приоритетами.

В обоих случаях выбор потока на выполнение из очереди готовых осуществляется одинаково: выбирается поток, имеющий наивысший приоритет. Однако проблема определения момента смены активного потока решается по-разному.

В системах с относительными приоритетами активный поток выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ожидания (или же произойдёт ошибка, или поток завершится).

В системах с абсолютными приоритетами выполнение активного потока прерывается кроме указанных выше причин, еще при одном условии: если в очереди готовых потоков появился поток, приоритет которого выше приоритета активного потока.

В системах, в которых планирование осуществляется на основе относительных приоритетов, минимизируются затраты на переключения процессора с одной работы на другую. С другой стороны, здесь могут возникать ситуации, когда одна задача занимает процессор долгое время. Ясно, что для систем разделения времени и реального времени такая дисциплина обслуживания не подходит: интерактивное приложение может ждать своей очереди часами, пока вычислительной задаче не потребуется ввод-вывод. А вот в системах пакетной обработки относительные приоритеты используются широко.

В системах с абсолютными приоритетами время ожидания потока в очередях может быть сведено к минимуму, если ему назначить самый высокий приоритет. Такой поток будет вытеснять из процессора все остальные потоки (кроме имеющих такой же наивысший приоритет). Это делает планирование на основе абсолютных приоритетов подходящим для систем управления объектами, в которых важна быстрая реакция на событие.

Динамический приоритет – приоритет, устанавливаемый и изменяемый операционной системой для обеспечения эффективной работы компьютера.

Основное отличие статических и динамических приоритетов в том, что статические приоритеты система только учитывает в своей работе, а динамические она формирует и модифицирует, исходя из общих критериев эффективности.

Статический приоритет – приоритет, который устанавливается и изменяется пользователем, то есть система не модифицирует приоритеты процессов во время работы. Однако следует учитывать, что такой механизм недостаточно гибок, т.к. не реагирует на изменение окружающей ситуации.

## Межпроцессное взаимодействие

Межпроцессное взаимодействие — обмен данными между потоками одного или разных процессов.

Каждый процесс в операционной системе выполняется самостоятельно. Для обеспечения взаимодействия между процессами используют сигналы, семафоры, механизмы разделяемой, отображаемой памяти, сообщения, каналы, именованные каналы (FIFO).

Проектируемая операционная система поддерживает такие средства взаимодействия процессов как каналы, именованные каналы и разделяемая память.

Канал - однонаправленное средство взаимодействия. Данные, записанные в канал со "стороны записи" читаются со "стороны чтения." Каналы - последовательные устройства; данные всегда читаются в том же порядке, в котором были записаны. Как правило, канал используется для взаимодействия двух потоков в одном процессе или между родительскими и дочерними процессами.

В командной оболочке Unix каналы создаются с помощью символа «|».

Рассмотрим команду cat file | wc. Она создает канал, соединяющий стандартный вывод процесса cat file со стандартным вводом процесса wc. В результате работы команды содержимое файла file будет послано команде wc, которая, в свою очередь, подсчитывает количество строк, слов и символов.

Емкость канала ограничена. Если процесс записи пишет быстрее, чем процесс чтения считывает данные, и, если канал не может больше сохранять данных, процесс записи блокируется до тех пор, пока канал не разгрузится. Если считывающий процесс пытается прочитать, но данные недоступны, он блокируется до тех пор, пока данные не появятся в канале. Таким образом, канал автоматически синхронизирует два процесса.

Для создания канала используется системный вызов pipe:

int pipe(int\* fildes);

Он возвращает два файловых дескриптора — fildes[0] для записи в канал и fildes[1] для чтения из канала. Теперь, если один процесс записывает данные в fildes[0], другой сможет получить эти данные из fildes[1]. Так как дочерний процесс наследует и разделяет все назначенные файловые дескрипторы родительского, то доступ к дескрипторам fildes канала может получить сам процесс, вызвавший pipe, и его дочерние процессы. В этом заключается серьезный недостаток каналов, поскольку они могут быть использованы для передачи данных только между родственными процессами. Каналы не могут использоваться в качестве средства межпроцессного взаимодействия между независимыми процессами.

Пример создания канала для ранее рассмотренной команды cat file | wc представлен на рисунке 2.2.

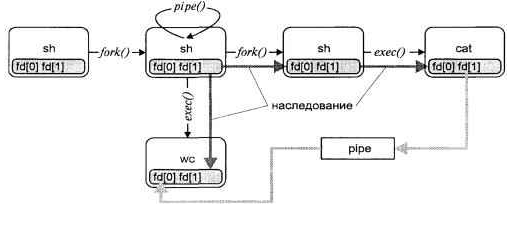


Рисунок 2.2 -  Создание канала между задачами cat(1) и wc(1)

Рассмотрим следующее изображение процесса и ядра после создания канала (рисунок 2.3).



Рисунок 2.3 – Работа канала на уровне ядра

Хотя канал создается одним процессом, он редко используется только этим процессом. Каналы обычно используются для связи между двумя процессами (родительским и дочерним) следующим образом: процесс создает канал, а затем вызывает fork, создавая свою копию — дочерний процесс. Затем родительский процесс закрывает открытый для чтения конец канала, а дочерний, в свою очередь, — открытый на запись конец канала. Это обеспечивает одностороннюю передачу данных между процессами (рисунок 2.4).



Рисунок 2.4 – Взаимодействие родительского и дочернего процесса

Пример работы с каналами на языке СИ представлен на рисунке 2.5

int main(void)

{

int fd[2], nbytes;

pid\_t childpid;

char string[] = "Hello, world!\n";

char readbuffer[80];

pipe(fd);

if((childpid = fork()) == -1)

{

perror("fork");

exit(1);

}

if(childpid == 0)

{

/\* Потомок закрывает вход \*/

close(fd[0]);

/\* Посылаем "string" через выход канала \*/

write(fd[1], string, strlen(string));

exit(0);

}

else

{

/\* Родитель закрывает выход \*/

close(fd[1]);

/\* Чтение строки из канала \*/

nbytes = read(fd[0], readbuffer, sizeof(readbuffer));

printf("Received string: %s", readbuffer);

}

return(0);

}

Рисунок 2.5 – Пример работа с каналом на языке СИ

Именованные каналы, в отличии от программных (неименованных), позволяет обмениваться данными между «чужими» процессами.

Именованный канал (FIFO) - канал, который имеет имя в файловой системе. Любой процесс может открыть или закрыть именованный канал; процессы на любом конце канала не должны быть связаны друг с другом.

С точки зрения ядра ОС FIFO является одним из вариантов реализации трубы. Системный вызов mkfifo() предоставляет процессу именованную трубу в виде объекта файловой системы. Как и для любого другого объекта, необходимо предоставлять процессам права доступа в FIFO, чтобы определить, кто может писать, и кто может читать данные. Несколько процессов могут записывать или читать FIFO одновременно. Режим работы с FIFO - полудуплексный, т.е. процессы могут общаться в одном из направлений. Типичное применение FIFO - разработка приложений ``клиент - сервер''.

Синхронизация выполняется с помощью разделяемой памяти.

Техника разделяемой памяти позволяет осуществлять обмен информацией через общий для процессов сегмент памяти без использования системных вызовов ядра. Сегмент разделяемой памяти подключается в свободную часть виртуального адресного пространства процесса. Таким образом, два разных процесса могут иметь разные адреса одной и той же ячейки подключенной разделяемой памяти.

## Выбор дисциплин обслуживания планировщика процессов. Алгоритм работы планировщика процессов в соответствии с выбранной дисциплиной обслуживания

В результате анализа требований к планировщику процессов (абсолютные приоритеты, относительные приоритеты и динамические приоритеты) был выбрани модифицирован смешанный алгоритм планирования.

Существует две очереди готовых процессов, каждая из которых содержит процессы определённого приоритета. Одна из них содержит процессы только с абсолютными приоритетами, а другая – не абсолютные. Очередь с абсолютными приоритетами выполняется до тех пор, пока в ней не закончатся процессы. И только после этого, инициатива передаётся очереди без абсолютных приоритетов. Но если во время исполнения процесса, состоящего в очереди не абсолютных приоритетов, появляется абсолютный или статический приоритет, то он вытесняет неабсолютный и начинает немедленно исполнятся. Это называется вытесняющее планирование. Очередь, содержащая процессы с абсолютными приоритетами, обрабатывается в соответствии с алгоритмом «Round Robin (RR)». Текущему процессу из очереди выделяется один квант времени. По истечении этого кванта, текущий процесс становится в конец очереди, а на выполнение поступает следующий за ним процесс. Абсолютные процессы имеют большую важность для функционирования системы, поэтому их своевременное и правильное исполнение играет важную роль. Увеличение выделяемого кванта времени для исполнения процесса оправдывается уменьшением сбоев при переключении процессов. В очереди, содержащей процессы с неабсолютными приоритетами, реализуется алгоритм приоритетного планирования. При приоритетном планировании каждый процесс имеет приоритет, в соответствии с которым ему выделяется процессор. При этом, приоритеты не остаются в одном состоянии. Каждый третий квант времени в очереди с неабсолютными процессами, у исполняемого в данный момент процесса понижается приоритет, а у самого последнего в очереди - повышается.

# Режимы проектируемой ОС

Рассмотрим режимы работы доступные в проектируемой гипотетической ОС.



## Мультипрограммный режим работы ОС

Мультипрограммный режим работы — режим функционирования ОС, при котором несколько процессов одновременно находятся на стадии выполнения. В нём организовано переключение выполнения одной программы на другую. Ресурсы вычислительной системы предоставляются каждому процессу из группы процессов, находящихся в системе, на интервалы времени, длительность и очередность предоставления которых определяется планировщиком ОС с целью обеспечения одновременной работы в интерактивном режиме.

Основные особенности мультипрограммного режима:

1. в оперативной памяти находятся несколько пользовательских программ в состояниях активности, ожидания или готовности;
2. время работы процессора разделяется между программами, находящимися в памяти в состоянии готовности;
3. параллельно с работой процессора происходит подготовка и обмен информацией с несколькими устройствами ввода-вывода.

Мультипрограммирование предназначено для повышения пропускной способности вычислительной системы путем более равномерной и полной загрузки всего ее оборудования, в первую очередь процессора. При этом скорость работы самого процессора и номинальная производительность ЭВМ не зависят от мультипрограммирования.

При наличии единственного центрального процессора параллелизм достигается за счёт асинхронной работы устройств. В то время как одна из программ выполняется на процессоре, другие программы могут выполнять операции на внешних устройствах. Таким образом, выигрыш эффективности по сравнению с однопрограммным режимом достигается за счёт параллельной работы устройств, входящих в состав компьютера.

## Многопользовательская защита

Многопользовательская защита — средства ОС, обеспечивающие идентификацию пользователей и различные уровни их привилегий, а также установку и защиту прав собственности на информационные ресурсы, создаваемые в среде ОС. Надежная защита возможна только при наличии специальных аппаратных средств, обеспечивающих защиту оперативной памяти и различные режимы работы процессора.

В разработанной файловой системе реализована многопользовательская защита. По умолчанию в системе присутствует только один пользователь, который является администратором системы. Регистрацию новых пользователей и их удаление может выполнять только он. Более того, только он имеет доступ к просмотру служебных файлов (MFT-файла, битовой карты, списка пользователей) и может форматировать систему.

В системе существует служебный файл $Users, который содержит список пользователей системы. Пользователь не может изменить свое учётное имя. Пароли в файле хранятся в закодированном виде, поэтому если пользователь забыл пароль, то он уже не сможет войти систему, так как его пароль не знает даже администратор.

Чтобы начать работать, пользователь должен "войти" в систему, введя свое учётное имя и пароль. После входа он перенаправляется в свою домашнюю директорию, к которой и в которой только он (и администратор) имеет полный доступ. Владелец элемента файловой системы может устанавливать на него права таким образом, что другие пользователи не смогут получить доступ к записи, чтению и изменению этого элемента, либо же наоборот, получат к нему полный доступ (станут его владельцем).

Права доступа в файловой системе указаны отдельно для каждого файла. Они представлены тремя наборами из 4 битов, первый из которых определяет права администраторов, второй – владельца, третий – всех остальных пользователей. Права предназначены для определения тех действий, которые может сделать с файлами текущий пользователь. Так, например, при чтении или записи файла, создании объектов и т.д. происходит проверка прав пользователя, который находится в системе. В зависимости от чего, ему либо предоставляется доступ к указанному действию, либо выводится сообщение об ошибке доступа.

Если администратор удаляет пользователя, то он становится владельцем всех созданных им объектов.

## Интерактивный режим работы ОС

Интерактивный режим — режим работы ОС, при котором основными источниками команд являются пользователи, взаимодействующие с компьютером посредством терминалов. В интерактивном режиме необходимо обеспечить приемлемое для пользователя время исполнения команды (как правило, в пределах нескольких секунд).

В данной курсовой работе при работе с файловой системой и процессами пользователь общается с системой посредствами графического интерфейса, реализованного c помощью технологии WPF (Windows Presentation Foundation). Пользователь выбирает необходимый пункт меню либо нажимает необходимую горячую клавишу и вводит необходимые параметры. Если все верно, то выполнится заданный метод и пользователь получит требуемые данные, а в случае ошибки будет выведено соответствующее сообщение.

## Пакетный режим работы

Пакетный режим - режим работы ОС, при котором основными источниками команд являются специальные пакетные файлы, предварительно подготовленные пользователями. В пакетном режиме время исполнения конкретной команды не столь существенно как в интерактивном. Главным критерием работы пакетной ОС является, как правило, максимальная загрузка оборудования.

Главным критерием работы пакетной ОС является, как правило, максимальная загрузка оборудования. Для достижения этой цели в системах пакетной обработки используются следующая схема функционирования:

1. Формируется пакет заданий, в котором каждое содержит требование к системным ресурсам.
2. Из пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач.
3. Выбираются задачи, предъявляющие отличающиеся требования к ресурсам, так, чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины.

Для обеспечения интерактивности работы разрабатываемая ОС должна совмещать в себе свойства систем разных типов, а именно: часть задач может выполняться в режиме пакетной обработки, а часть - в режиме реального времени или в режиме разделения времени.

# 4 Структура операционной системы



## Общая структура проектируемой ОС

Компоненты операционной системы:

* загрузчик;
* ядро;
* базовая система ввода-вывода;
* драйверы устройств;
* графическая оболочка (интерфейс).

Загрузчик - системное программное обеспечение (ПО), обеспечивающее загрузку операционной системы непосредственно после включения компьютера: обеспечивает необходимые средства для диалога с пользователем компьютера, загружает ядро операционной системы в ОЗУ, передаёт управление ядру операционной системы.

Ядро - центральная часть операционной системы (ОС), обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации. Также обычно ядро предоставляет сервисы файловой системы и сетевых протоколов. Как основополагающий элемент ОС, ядро представляет собой наиболее низкий уровень абстракции для доступа приложений к ресурсам системы, необходимым для их работы.

Базовая система ввода-вывода (BIOS) – компонента ОС, реализующая программный интерфейс для взаимодействия с минимальным набором устройств ввода-вывода и периферийных устройств компьютера, набор программ управления основными функциями и устройствами компьютера.

Драйверы устройств - компьютерное программное обеспечение, с помощью которого другое программное обеспечение (ОС) получает доступ к аппаратному обеспечению некоторого устройства. Обычно с операционными системами поставляются драйверы для ключевых компонентов аппаратного обеспечения, без которых система не сможет работать. Однако для некоторых устройств (таких, как видеокарта или принтер) могут потребоваться специальные драйверы, обычно предоставляемые производителем устройства.

Графическая оболочка - тип интерпретатора команд операционной системы, обеспечивающего интерфейс для взаимодействия пользователя с функциями системы. Работа с графической оболочкой разработанной ОС осуществляется с помощью клавиатуры и мыши.

## Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами

За основу ядра проектируемой операционной системы была взята и модифицирована структура ядра UNIX. Структура ядра UNIX представлена на рисунке 4.1.



Рисунок 0.1 - Структура ядра UNIX

Ядро ОС состоит из следующих основных подсистем:

* Файловая подсистема
* Подсистема управления памятью и процессами
* Подсистема управления устройствами

Файловая подсистема обеспечивает унифицированный интерфейс доступа к данным, расположенным на дисковых накопителях, и к периферийным устройствам. Файловая подсистема контролирует права доступа к файлу, выполняет операции размещения и удаления файла, а также выполняет запись/чтение данных файла

Подсистема управления памятью и процессами представляет собой совокупность модуля планировщика процессов и модуля управления памятью. Планировщик разрешает конфликты между процессами в конкуренции за системные ресурсы (процессор, память, устройство ввода-вывода). Модуль памяти управления памятью обеспечивает предоставление оперативной памяти для прикладных задач и управляет виртуальной памятью процессов.

Подсистема управления устройствами обеспечивает взаимодействие с устройствами с использованием драйверов устройств – специальных модулей ядра, которые занимаются непосредственно обслуживанием конкретного устройства [5]

## Структура управляющих блоков базы данных ОС

База данных ОС содержит всю информацию, необходимую ей для функционирования. БД состоит из статической части и динамической. Размер статической части не изменяется, а динамическая может изменять свой размер в процессе работы системы. Динамическая часть загружается за пределы системного пула. В БД содержится информация о запущенных процессах, открытых файлах, подключенных пользователях.

БД должна содержать следующие управляющие блоки:

* Блок управления пользователями (Таблица 4.1)
* Блок управления файлами (Таблица 4.2)
* Блок управления памятью (Таблица 4.3)
* Блок управления каталога страниц памяти (Таблица 4.4)
* Блок управления процессами (Таблица 4.5)
* Блок управления устройствами (Таблица 4.6)
* Блок управления сообщениями (Рисунок 4.3)

При входе в систему блок управления пользователями регистрирует пользователя, при выходе пользователя из системы запись о нем удаляется из системы.

При запуске процесса блок управления процессами регистрирует процесс, при завершении процесса, запись о нем удаляется.

При обращении к ОП блок управления памятью регистрирует управление обращение, определяет по ряду признаков возможность доступа и права пользователя, при завершении работы процесса с этим сегментом блок выставляет флаг доступа в 1, разрешая тем самым другому процессу занять этот сегмент.

Блок управления устройствами регистрирует все устройства, их текущее состояние, изменяя его в ходе работы.

Блок управления файлами заносит запись об открытом файле в таблицу, устанавливает признаки, при закрытии файла запись удаляется из таблицы.

Общая схема организации БД приведена на рисунке 4.2.

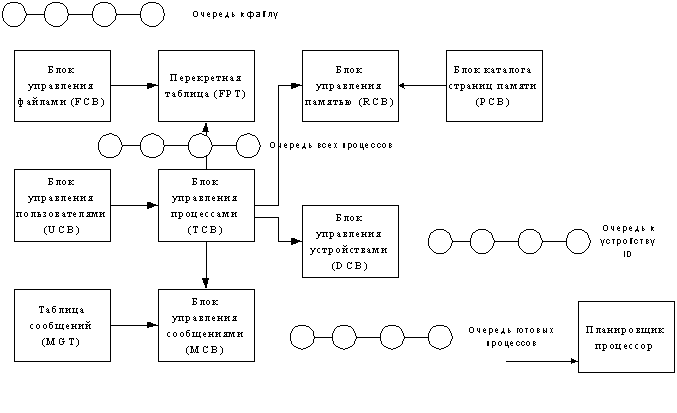


Рисунок 0.2 - Общая схема организации БД

Таблица 4.1 - Блок управления пользователями (User Control Block)

|  |  |
| --- | --- |
| UCB\_ID | ID пользователя; |
| UCB\_Name | имя пользователя |
| UCB\_GROUP | группа пользователя (текущая) |
| UCB\_RIGHT | права пользователя |

Таблица 4.2 - Блок управления файлами (File Control Block)

|  |  |
| --- | --- |
| FCB\_ID | ID файла; |
| FCB\_NAME | имя файла |
| FCB\_FPT | адрес перекрестной таблицы |
| FCB\_FPCOUNT | количество связей файла |
| FCB\_DEVID | ID устройства |
| FCB\_SIZE | размер файла |
| FCB\_UID | идентификатор владельца |

Продолжение таблицы 4.2 - Блок управления файлами (File Control Block)

|  |  |
| --- | --- |
| FCB\_GUID | идентификатор группы |
| FCB\_DATE | дата создания |
| FCB\_TIME | время создания |
| FCB\_HDATE | дата изменения |
| FCB HTIME | время изменения |
| FCB\_START | начальный кластер |
| FCB\_ATTR | атрибуты файла |
| FCB\_NEXT | указатель на следующий FCB |

Информация об оперативной памяти располагается в статической части БД и не является набором структур, в блоке управления памятью хранится информация об общем размере, начальные адреса физических сегментов.

Таблица 4.3 - Блок управления памятью (Memory Control Block)

|  |  |
| --- | --- |
| RCB\_HANDLE | дескриптор блока |
| RCB\_SIZE | размер блока |
| RCB\_CADR | текущий адрес блока |
| RCB\_RIGHT | права доступа |

Таблица 4.4 - Блок каталога страниц памяти. (Page Mapping Table Control Block)

|  |  |
| --- | --- |
| PCB\_TID | ID таблицы страниц |
| PCB\_DESCRIPT | описание страницы |

Максимально количество одновременно запущенных процессов определяется системной переменной SYSP\_MAXP, которая может быть изменена привилегированными пользователем. Для того, чтобы избежать необходимости перезагрузки системы после изменения значения переменной, дескрипторы процессов располагаются в динамической части БД. В статической части располагается информация о количестве запущенных процессов и адреса таблицы дескрипторов процессов.

Таблица 4.5 - Блок управления процессами (Task Control Block)

|  |  |
| --- | --- |
| TCB\_ID | системный процесс. |
| ID=01xxxxxx | резидентный процесс. |
| TCB\_NAME | имя процесса. |
| TCB\_QID | идентификатор очереди, в которой стоит процесс |
| TCB\_PPID | идентификатор процесса родителя |
| TCB\_UID | идентификатор пользователя |
| TCB\_INP | точка входа процесса (начальный адрес сегмента) |
| TCB\_RPIO | приоритет процесса, первый бит признак абсолютного приоритета |
| TCB\_CONT | контекст процесса (состояние регистров, указатель стека) и объем занимаемой памяти |

В статической части БД хранится только флаг наличия устройства в системе DCB\_ST, название устройства DCB\_NAME и адрес драйвера в памяти DCB\_DRV.

ОС автоматически, через некоторый промежуток времени проверяет статус устройства (DCB\_ST). При отсутствии устройства, которое было подключено, драйвер выгружается из памяти. При обнаружении нового устройства, ОС получает данные об устройстве и загружает соответствующий драйвер. При отсутствии необходимого драйвера (например, устройство не поддерживается системой по умолчанию), привилегированному пользователю предлагается указать местоположение драйвера.

Каждое устройство имеет свой запрос на прерывание и порт ввода вывода. При загрузке драйвера ему сообщается эта информация, и дальнейшая работа с устройством ведется только посредством драйвера. Для незанятых прерываний формируются "заглушки" - пустые устройства, статус которых "отсутствует".

Таблица 4.6 - Блок управления устройствами (Device Control Block)

|  |  |
| --- | --- |
| DCB\_ID | ID устройства |
| DCB\_NAME | имя устройства |
| DCB\_ST | статус устройства |
| DCB\_INT | таблица точек входа |
| DCB\_INIT | инициализация |
| DCB\_WRITE | флаг записи |
| DCB\_READ | флаг чтения |
| DCB\_PROP | параметры устройства |
| DCB\_DRV | адрес драйвера устройства |
| DCB\_NEXT | адрес следующего DCB |

Блок управления сообщениями - динамическая часть БД. Производит обработку входящих сообщений ОС. Для каждого процесса создается отдельная таблица входящих сообщений, содержащая ссылку на общую таблицу сообщений (MGT).

Каждый раз, когда возникает системное сообщение, в дескрипторы процессов, подключенных к этому сообщению, добавляется номер сообщения и сопутствующая информация. При получении процессом кванта времени, сначала вызываются обработчики пришедших сообщений. Процесс продолжает выполнение только после этого. Кроме того, процесс может послать сообщение другому процессу.

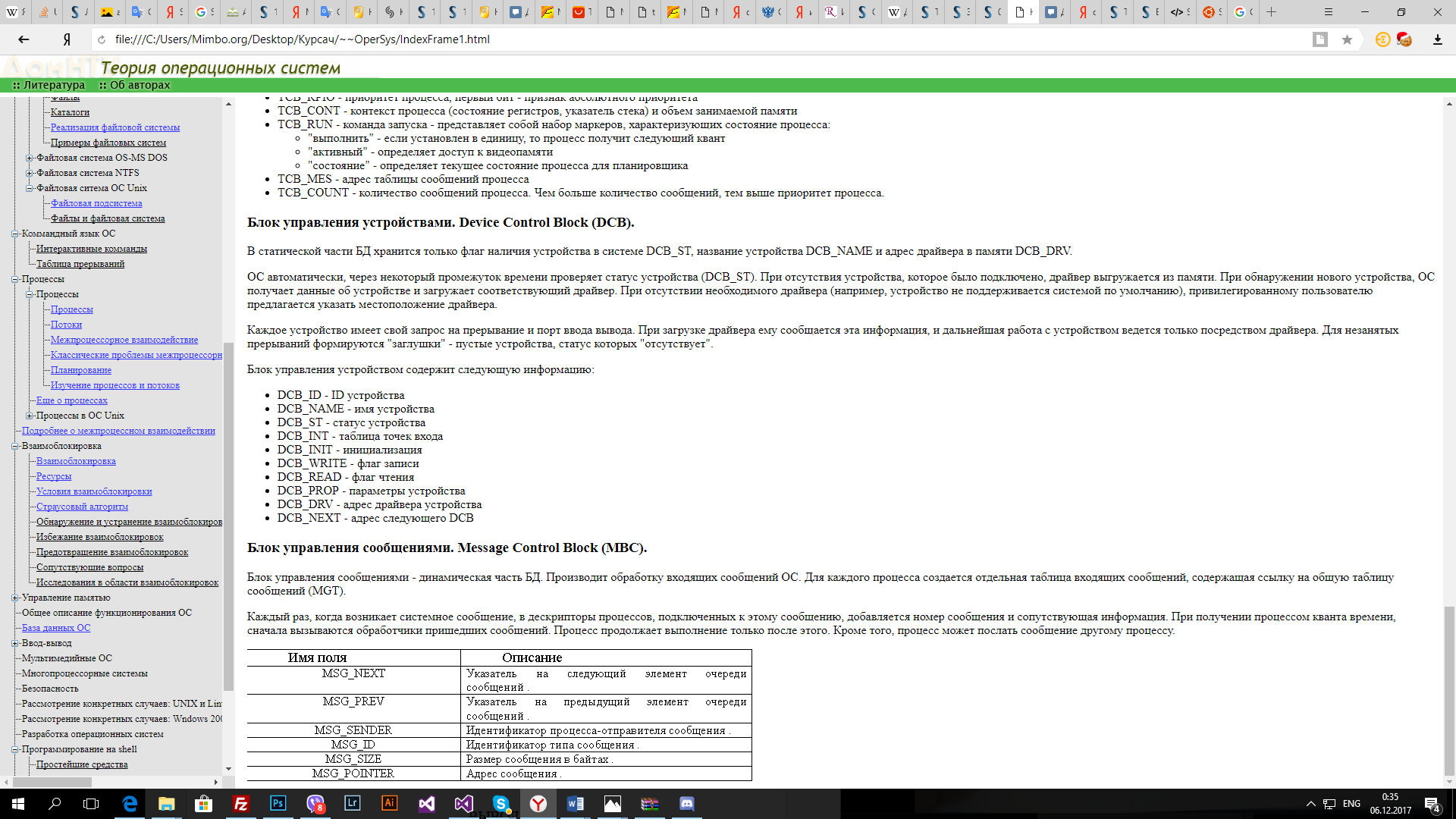


Рисунок 4.3 - Блок управления сообщениями(Message Control Block)

5 Разработка программ эмуляции ОС

## Описание программных средств

Для разработки программного обеспечения эмуляции ОС была выбрана платформа пользовательского интерфейса Windows Presentation Foundation (WPF), которая является частью платформы .NET Framework, и язык программирования C#. Разработка велась с использованием интегрированной среды разработки Microsoft Visual Studio.

## Разработка ФС

В разработанной ФС применён принцип ООП. Все модули распределены по классам, и реализует все необходимые функции для работы.

Эмулятор модуля файловой системы разработанной ОС содержит необходимый функционал для реализации системных вызовов ОС. Данный модуль содержит 2 основных класса, которые обеспечивают полную работу с файловой системой:

* FileSystem
* SystemCalls

Диаграмма классов эмулятора файловой системы представлена на рисунке 5.1.

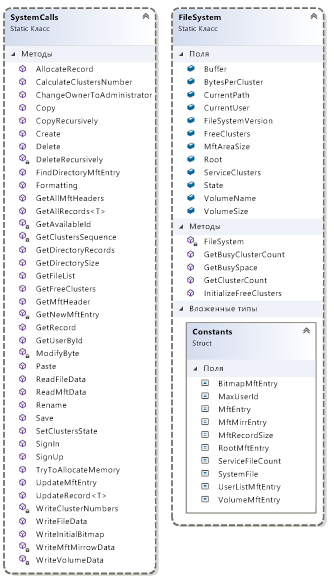


Рисунок 5.1 - Диаграмма классов основного модуля ФС

Класс SystemCalls является статическим и содержит определения для всех системных вызовов разрабатываемой ОС. Он обеспечивает взаимодействие с дисковым накопителем, предоставляя удобный программный интерфейс, который позволит ядру ОС работать с файлами и каталогами, не вникая в особенности хранения объектов и служебных структур файловой системы. Методы программного интерфейса класса описаны в таблице 5.1.

Таблица 5.1 - Методы класса «SystemCalls»

|  |  |
| --- | --- |
| AllocateRecord | Получает атрибуты размещения новой записи в памяти |
| CalculateClustersNumber | Подсчитывает количество кластеров по размеру файла |
| ChangeOwnerToAdministrator | Изменяет id владельца файла на id администратора |
| Copy | Копирует файл или папку |
| CopyRecursively | Делает рекурсивную копию содержимого папки, вызывается из Paste, если копируемым элементом была папка |
| Create | Создаёт файл или папку |
| Delete | Удаляет файл или папку |
| DeleteRecursively | Делает рекурсивное удаление содержимого папки, вызывается из Delete, если удаляемым элементом была папка |
| FindDirectoryMftEntry | Находит номер Mft-записи папки по указанному пути |
| Formatting | Форматирует том с указанными параметрами |
| GetAllMftHeaders | Получает список всех Mft-записей |

Продолжение таблицы 5.1 - Методы класса «SystemCalls»

|  |  |
| --- | --- |
| GetAllRecords | Получает список всех записей (либо учётных записей пользователей, либо файлов и папок выбранной директории) |
| GetAvailableId | Получает первый доступный id для создания нового пользователя |
| GetClustersSequence | Генерирует последовательный массив номеров кластеров с указаннного индекса |
| GetDirectoryRecords | Получает список всех записей директории |
| GetDirectorySize | Рассчитывает размер директории |
| GetFileList | Получает список файлов и папки для отображения его в проводнике |
| GetFreeClusters | Возвращает последовательность требуемого количества свободных кластеров |
| GetMftHeader | Получает Mft-заголовок по номеру |
| GetNewMftEntry | Получает первый свободный номер Mft-записи |
| GetRecord | Получает запись по имени (либо пользовательская учётная запись, либо запись директории) |
| GetUserById | Получает пользователя по указанному id |
| ModifyByte | Записывает в байт состояния четырёх кластеров и возвращает его |
| Paste | Вставляет скопированный элемент по указанному пути |
| ReadFileData | Считывает содержимое файла |
| ReadMftData | Читает атрибут Data Mft-записи |
| Rename | Переименовует выбранный элемент |

Продолжение таблицы 5.1 - Методы класса «SystemCalls»

|  |  |
| --- | --- |
| Save | Сохраняет содержимое файла |
| SetClustersState | Устанавливает состояние последовательности кластеров по их номерами |
| SignIn | Авторизирует пользователя |
| SignUp | Регистрирует пользователя |
| TryToAllocateMemory | Возвращает атрибуты размещения новой записи в памяти |
| UpdateMftEntry | Обновляет Mft-запись по номеру |
| UpdateRecord | Обновляет запись |
| WriteClusterNumbers | Записывает номера кластеров в Mft-запись |
| WriteFileData | Считывает содержимое файла |
| WriteInitialBitmap | Записывает начальные данные битовой карты |
| WriteMftMirrowData | Записывает данные копии Mft |
| WriteVolumeData | Записывает информацию о томе |

Класс FileSystem представляет собой настройки системы. Он содержит структуру списка констант файловой системы (размер MFT-записи, номера MFT-записей метафайлов), статический конструктор, несколько полей, доступных только для чтения (размер кластера, метка тома, размер MFT-зоны и т.д), и несколько изменяемых на протяжении всего выполнения программы полей (количество свободных кластеров, текущий путь, буфер обмена). Данный класс является статическим и инициализируется сразу после входа пользователя в систему. Он не делает никаких изменений на диске и имеет несколько методов для получения системных настроек. Они представлены в таблице 5.2.

Таблица 5.2 – Методы класса «FileSystem»

|  |  |
| --- | --- |
| GetBusyClusterCount | Получение количества занятых кластеров. |
| GetBusySpace | Расчёт занятого пространства в системе. |
| GetClusterCount | Получение общего количества MFT-зоны. |
| InitializeFreeClusters | Инициализируем число свободных и служебных кластеров. |

Для организации работы двух основных классов файловой системы (FileSystem и SystemCalls) в программе были разработаны собственные типы данных – классы, структуры, перечисления (рисунок 5.2).

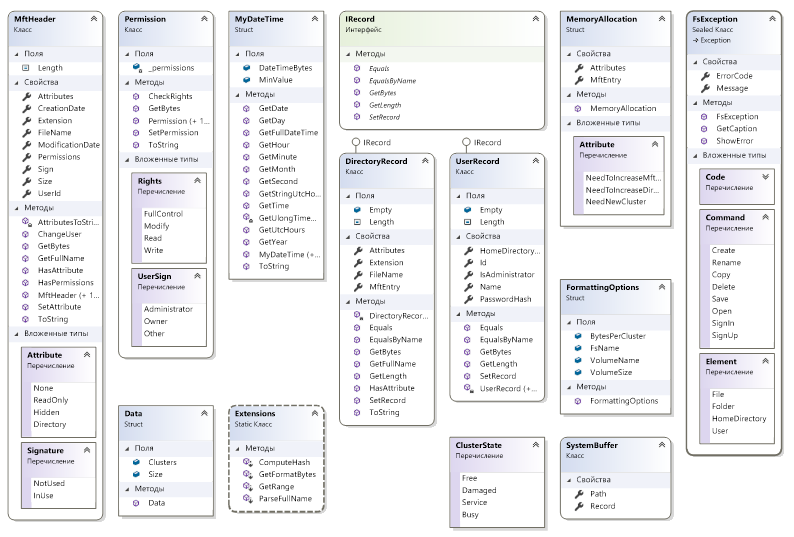


Рисунок 5.2 – Типы данных разработанной файловой системы

Описание данных классов и структур представлено в таблице 5.3.

Таблица 5.3 – Типы данных ФС

|  |  |
| --- | --- |
| Название | Описание |
| MftHeader | Описывает структуру MFT-заголовка записи (50 Б). |
| Permission | Описывает права пользователя (2Б ). |
| MyDateTime | Описывает собственный тип данных для времени и даты. |
| Data | Описывает атрибут Data Mft-записи. |
| Extensions | Содержит методы для получения или установления полей некоторых классов |
| IRecord | Описывает общие методы для записи диретктории и полтховетасльноской. |
| DirectoryRecord : IRecord | Описываем запись директории. |
| UserRecord : IRecord | Описываем пользовательскую запись. |
| MemoryAllocation | Описывает атрибуты распределения память под новы |
| FormattingOptions | Получение общего количества MFT-зоны. |
| SystemBuffer | Инициализируем число свободных и служебных кластеров. |
| ClusterState | Описывает перечисление возможных состояний кластера. |
| FsException | Описывает буфер обмена файловой системы. |

## Разработка командного интерпретатора

Для представления командного интерпретатора был выбран графический интерфейс пользователя (GUI), который наиболее удобен обычному пользователю. Взаимодействие с таким интерфейсом производится с использованием клавиатуры и мыши.

Основу модуля командного интерпретатора составляют 8 классов:

* App
* MainWindow
* PropertiesWindow
* NameWindow
* TextEditorWindow
* FormattingWindow
* ControlWindow
* EditFile

Диаграмма вариантов использования представлена на рисунке 5.3.



Рисунок 5.3 – Диаграмма вариантов использования системы

Диаграмма классов модуля командного интерпретатора изображена на рисунке 5.4.

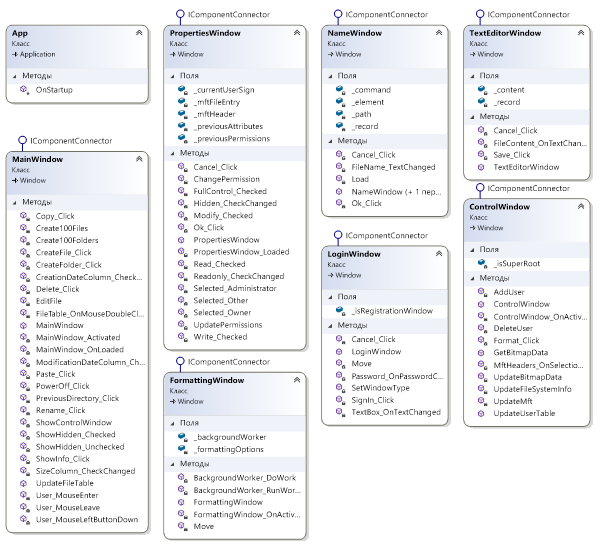


Рисунок 5.4 - Диаграмма классов модуля командного интерпретатора

Для визуализации некоторых моделей в виде пользовательского интерфейсе были созданы дополнительные классы, представленные на рисунке 5.5.

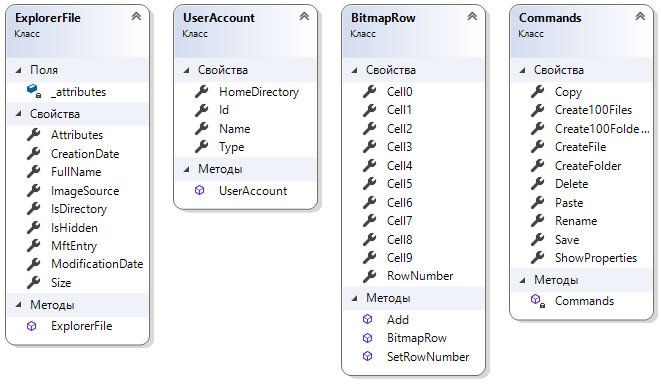


Рисунок 5.5 – Диаграмма классов моделей интерфейса

Работа программы начинается с класса App, который обеспечивает функционирование командного интерпретатора ОС. В нём проверяется существование файла ФС. Если он отсутствует, происходит форматирование с параметрами по умолчанию с запуском анимационной формы FormattingWindow (приложение В.16). Далее App запускает форму авторизации LoginWindow, в котором происходит вход пользователя в систему. Если он был успешен, окно возвращает положительный ответ и App запускает основной класс программы MainForm, в противном случае – закрывает программу.

Класс ControlWindow отвечает за управление системой. Для обычного пользователя в нём доступна только вкладка с информацией о системе (приложение В.10). Администратору же доступно ещё четыре вкладки (приложения В.12 – В.15). Во вкладке «Учётные запиис» он может просмотреть список всех зарегистрированных пользователей, а также добавить новых или удалить старых. Во вкладке с главной таблицей файлов он может просмотреть записи файла $MFT, во вкладке «Битовая карта» просмотреть состояние кластеров файловой системы в удобочитаемом формате, а во вкладке «Форматирование» произвести полный сброс настроек системы и произвести форматирование.

Класс NameWindow отвечает за указание имени к новому файлу или папке и за переименование существующего файла или папки. Класс PropertiesWindow показывает пользователю свойства выбранного файла, а также позволяет изменять служебную информацию о нём. Класс TextFditorWindow позволяет пользователю изменить содержимое выбранного файла.

## Демонстрация работы с межпроцессным взаимодействием

Для демонстрации работы с межпроцессным взаимодействием было разработано три программы программы, запускающиеся в стандартном терминале Linux.

Writer\_fd и reader\_fd эмулируют работу с именованными каналами. Writer\_fd создает именованный канал и записывает в него введенный текст. После этого программа ожидает, пока другая программа считает данные из канала, после чего завершит свое выполнение. Reader\_fd открывает созданный writer\_fd канал и считывает оттуда текст.

Shm – программа, эмулирующая работу с разделяемой памятью. Она работает в двух режимах – чтение и запись. В режиме записи создается раздел памяти, в который записываются данные. В режиме чтения, программа считывает данные и выводит их в терминал, если раздел по данному ключу существует.

Примеры работы программ представлены в приложениях В.17 – В.18.

6 Тестирование программы. Анализ результатов

Тестирование – это проверка правильности работы написанной программистом программы и выявления недочетов. Иначе говоря, установление соответствия программы заданным требованиям. Испытываемая программа является одним из этапов работ общего процесса разработки программ. Цель тестирования состоит в выявлении наличия ошибок в программе, описываемых выше.

Разработанное программное обеспечение для эмуляции ОС было протестировано в ОС Windows 10. Тестирование программы проводилось путем ввода различных входных значений и проверки полученных выходных результатос. На основе этого и подтверждались выводы о правильности работы программы.

Минимальные системные предствалены в таблице 6.1.

Таблица 6.1 – Минимальные требования

|  |  |
| --- | --- |
| Характеристики | Описание |
| OS | Microsoft Windows 10 |
| Программная платформа | Microsoft .NET Framework 4 |
| ОП | 1024 мб |

В разработанном ПО предусмотрена возможность обработки исключительных ситуаций, которые могут возникнуть в результате неверного ввода данных или неверных действий пользователя.

В окне входа в систему в случае введения неверных данных для авторизации система выведет сообщения, которое изображено на рисунке 6.1. После этого пользователю будет предоставлена возможность повторного ввода данных для авторизации.

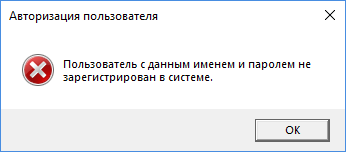


Рисунок 0.3 - Ошибка авторизации

В окне менеджера пользователей при попытке удаления системного пользователя с именем «root» также будет выдано сообщение об ошибке, которое представлено на рисунке 6.2.

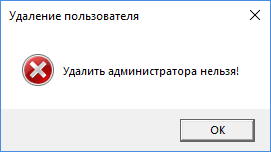


Рисунок 0.4 - Невозможность удаления пользователя root

Также в системе предусмотрена обработка исключений для прав доступа, рисунок 6.3.

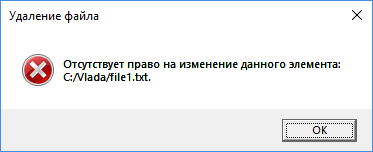


Рисунок 0.5 - Обработка исключений прав доступа

При попытке создания файла в каталоге с именем, которое занято другим файлом, будет выведено соответствующее сообщение, представленное на рисунке 6.4.

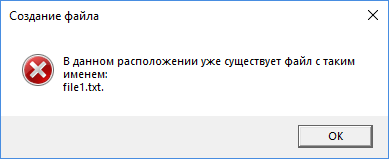


Рисунок 0.6 - Добавление файла с существующим названием

Выводы

Современная операционная система — сложный комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем. Программный интерфейс операционных систем позволяет уменьшить размер конкретной программы, упростить ее работу со всеми компонентами вычислительной системы.

В курсовой работе были спроектированы гипотетическая операционная система и разработана программа для работы с межпроцессным взаимодействием.

В процессе работы над курсовым проектом были закреплены теоретические знания курса «Операционные системы» и получены навыки в создании командных интерфейсов.

В качестве развития системы – будут дописываться новые методы для более удобного взаимодействия с ФС, а также будут добавляться новые модули и новые инструменты.

Перечень ссылок

1. Артём Баранов. Записки исследователя NTFS. 2009 г. [Электронный ресурс]. Режим доступа: http://citforum.ru/operating\_systems/windows/ntfs/#top
2. Гордеев А.В. Операционные системы: Учебник для вузов — 2-е изд. — СПб.: «Питер», 2007. — 416 с.
3. Карпов В.Е., Коньков К.А. Основы операционных систем. Курс лекций. Учебное пособие - М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2005 - 536 с.
4. Методические указания Технологии доступа к данным в информационных системах: Файловая система NTFS. Часть II. — Самара: СамИИТ, 2002. – 22 с.
5. Разрешения NTFS. 2015 г. [Электронный ресурс]. Режим доступа: https://winnote.ru/security/93-razresheniya-ntfs.html
6. Робачевский А. М., Операционная система Unix. - СПб.: BHV-Санкт-Петербург, 1998. -528 С., Ил.
7. Таненбаум Э.С. Современные операционные системы — 2-е изд. — СПб.: «Питер», 2002. — 1040 с.

# Приложение А Техническое задание

Утверждено зав. кафедрой ПИ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Федяев О.И.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

к курсовому проекту

по дисциплине «Операционные системы»

студента группы ПИ-16Б, Мамутовой Влады Александровны

Тема: “Проектирование гипотетической операционной системы

и программная эмуляция отдельных модулей”

Вариант № 69

Задание:

1. Режим работы ОС: мультипрограммный, многопользовательская защита, интерактивный, пакетный.
2. Управление процессами: абсолютные приоритеты, относительные приоритеты, статические приоритеты.
3. Средства взаимодействия процессов: каналы, именованные каналы, разделяемая память.

4. Организация оперативной памяти: страничная.

5. Перечень внешних устройств: видеотерминал, НМД.

6. Организация файловой системы: многоуровневая, битовая карта свободных/занятых кластеров.

7. Команды интерпретатора: копирование, вход и выход в систему, переименование, создание и удаление.

8. Эмуляция работы: эмуляция межпроцессного взаимодействия.

График выполнения курсового проекта:

|  |  |
| --- | --- |
| Неделя | Работа |
| 1-2 | Выдача задания и изучение задания |
| 3 | Анализ требований к системе и способов их реализации |
| 4-5 | Проектирование файловой системы |
| 6-7 | Организация процессов |
| 8 | Поддержка режимов работы, проектирование общей структуры |
| 9-13 | Разработка программ эмуляции |
| 14 | Оформление пояснительной записки |
| 15-17 | Защита курсового проекта |

Дата выдачи задания 04.09.2018

Задание принял Мамутова В.А.

Руководитель проекта Чернышова А.В.

# Приложение Б. Руководство пользователя

Системные требования: компьютер под управлением операционной системы MS Windows XP/Vista/7/8.1/10, монитор, клавиатура.

Программа запускается с помощью файла «MyOS.exe». Если файл c ФС VMFS.bin не существует, то начнётся выполнение форматирования тома с параметрами по умолчанию и созданием единственного пользователя - администатора. Далее запускается форма авторизации, в которой будет предложено ввести имя пользователя и пароль. Если было произведено форматирование, то в систему можно войти только с учётной записи администратора, в этом случае следует ввести “admin” в поле «Имя» и “admin” в поле «Пароль».

Если логин и пароль введены правильно, то произойдет разблокирования всех основных команд. Можно будет производить различные операции с файлами и пользователями, перечень которых был описан выше, с учётом типа пользователя.

При открытии приложения для демонстрации работы межпроцессного взаимодействия, пользователь сможет выбрать режим работы программы, опираясь на подсказки и записать какие-либо данные, с последующим чтением.

# Приложение В. Экранные формы, отображающие результаты работ программ эмуляции

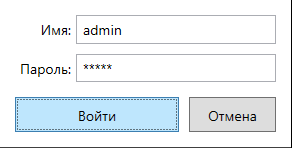


Рисунок В.1 – Форма авторизации

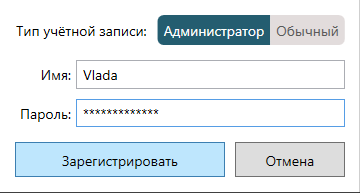


Рисунок В.2 – Форма регистрации

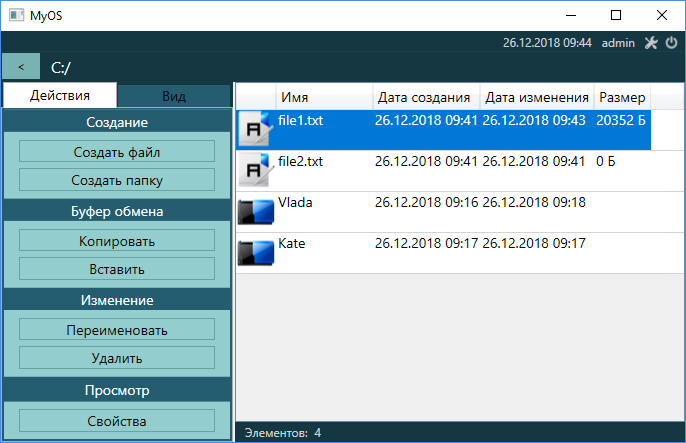


Рисунок В.3 – Рабочая область

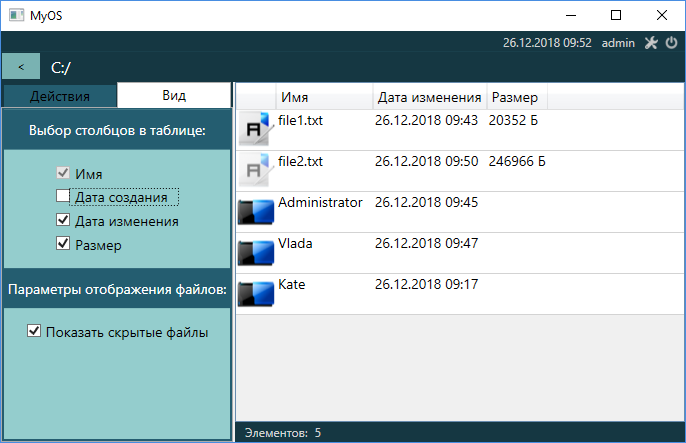


Рисунок В.4 – Вкладка «Вид» на главной форме

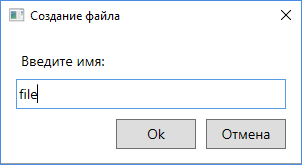


Рисунок В.5 – Создание файла

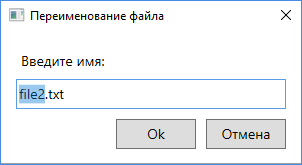


Рисунок В.6 – Переименование файла

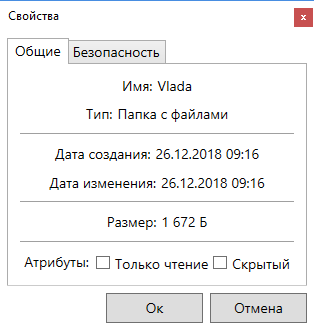


Рисунок В.7 – Свойства папки

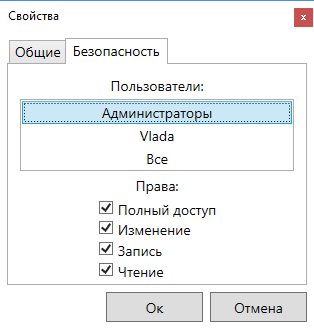


Рисунок В.8 – Вкладка «Безопасность» в свойствах папки

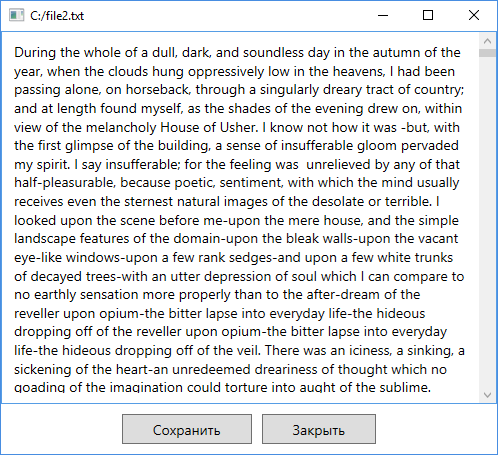


Рисунок В.9 – Текстовый редактор

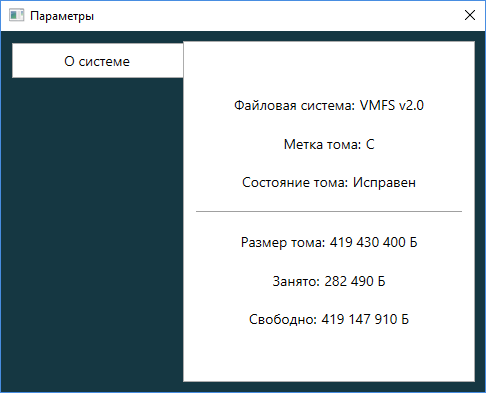


Рисунок В.10 – Параметры системы в режиме обычного пользователя

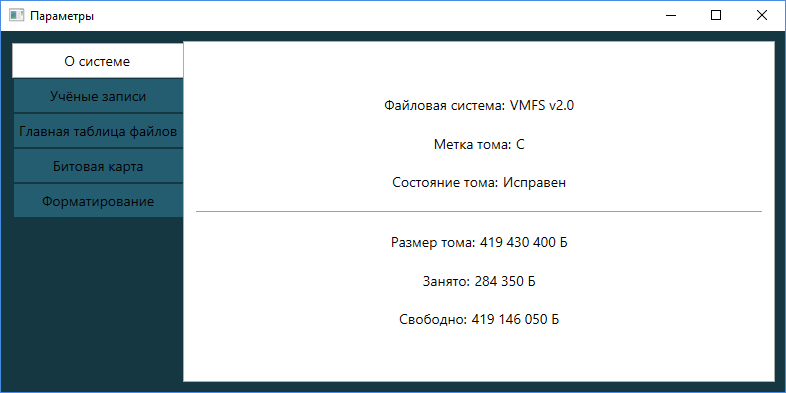


Рисунок В.11 – Параметры системы в учётной записи «admin»

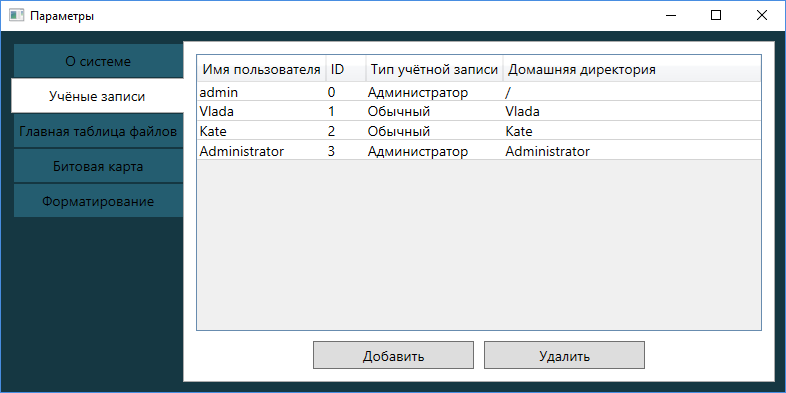


Рисунок В.12 – Учётные записи пользователей

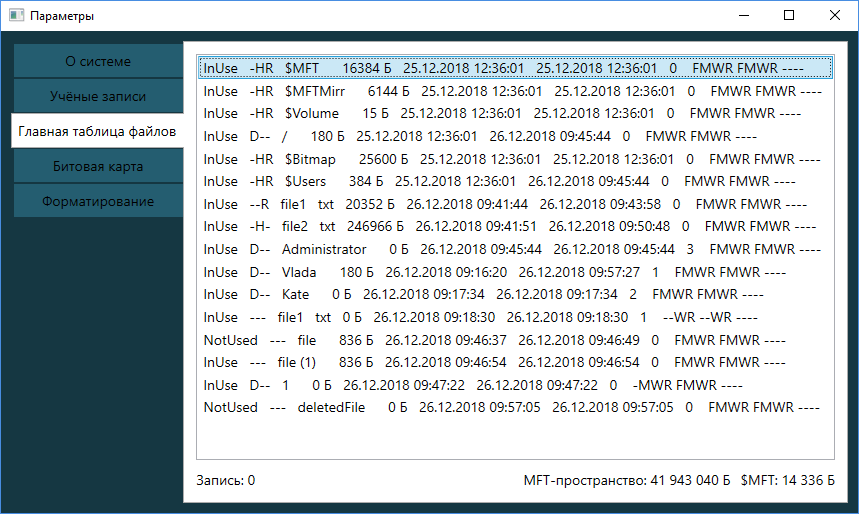


Рисунок В.13 – Главная таблица файлов системы

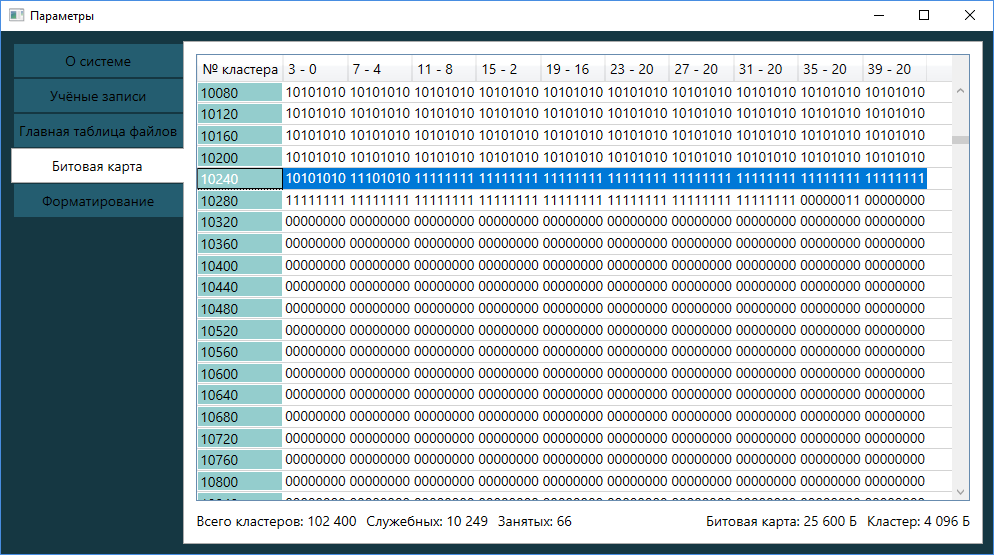


Рисунок В.14 – Битовая карта системы

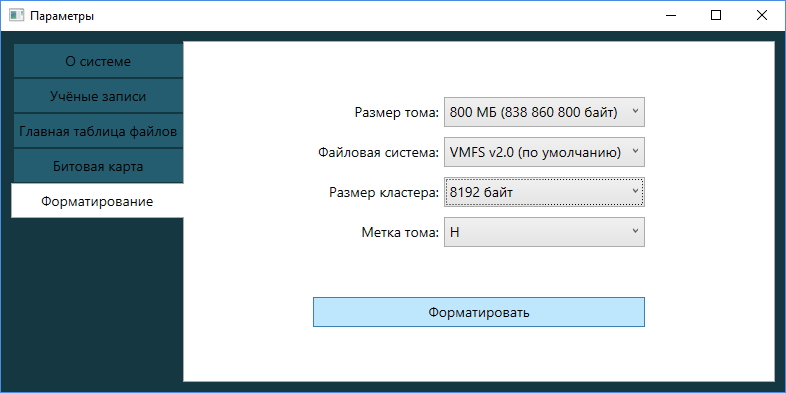


Рисунок В.15 – Форматирование системы

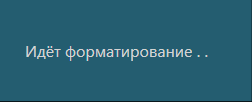


Рисунок В.16 – Процесс форматирования

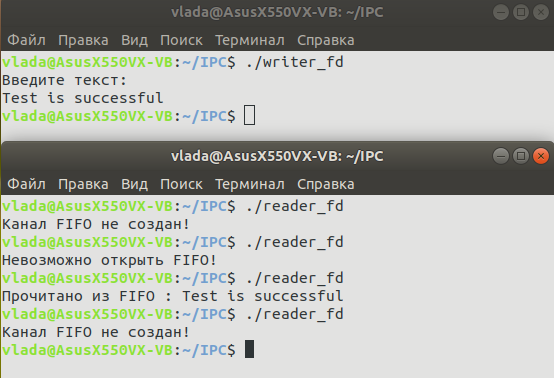


Рисунок В.17 – Демонстрация работы программы для FIFO

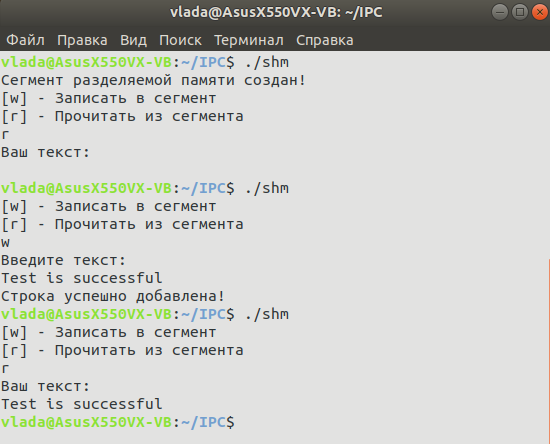


Рисунок В.18 – Демонстрация работы программы для разделяемой памяти

# Приложение Г. Листинг программы

// Классы файловой системы

// FileSystem.cs

using System;

using System.IO;

using System.Linq;

using System.Text;

using MyOS.FileSystem.SpecialDataTypes;

using Path = MyOS.FileSystem.SpecialDataTypes.Path;

namespace MyOS.FileSystem

{

static class FileSystem

{

/// <summary>

/// Системные константы.

/// </summary>

public struct Constants

{

public const string SystemFile = "VMFS.bin";

public const ushort MftRecordSize = 1024; // Размер записи в MFT.

public const byte ServiceFileCount = 6; // Всего служебных метафайлов.

public const byte MaxUserId = Byte.MaxValue; // Максимальное число пользователей в системе.

// Фиксированные номера метафайлов в MFT.

public const byte MftEntry = 0;

public const byte MftMirrEntry = 1;

public const byte VolumeMftEntry = 2;

public const byte RootMftEntry = 3;

public const byte BitmapMftEntry = 4;

public const byte UserListMftEntry = 5;

}

public static readonly char VolumeName; // Метка тома.

public static readonly char Root; // Имя корневого каталога.

public static readonly string FileSystemVersion; // Версия файловой системы

public static readonly byte State; // Состояние тома.

public static readonly int VolumeSize; // Размер тома.

public static readonly int MftAreaSize; // Размер MFT-пространства (10% от всего размера).

public static readonly int BytesPerCluster; // Размер кластера в байтах.

public static int FreeClusters;

public static int ServiceClusters;

public static UserRecord CurrentUser; // Текущий пользователь системы.

public static SystemBuffer Buffer; // Системный буфер.

public static Path CurrentPath;

static FileSystem()

{

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

VolumeSize = (int)br.BaseStream.Length;

// Вычисляем размер Mft-пространства, делая его кратным размеру Mft-записи.

MftAreaSize = VolumeSize / 10 - VolumeSize / 10 % Constants.MftRecordSize;

br.BaseStream.Seek(Constants.VolumeMftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

VolumeName = br.ReadChar();

FileSystemVersion = Encoding.UTF8.GetString(br.ReadBytes(9));

State = br.ReadByte();

BytesPerCluster = BitConverter.ToInt32(br.ReadBytes(4), 0);

br.BaseStream.Seek(Constants.RootMftEntry \* Constants.MftRecordSize + 1, SeekOrigin.Begin);

Root = br.ReadChar();

}

InitializeFreeClusters();

}

public static void InitializeFreeClusters()

{

Data bitmap = SystemCalls.ReadMftData(Constants.BitmapMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int byteNumber = 0; // Номер прочитанного байта в битовой карте.

foreach (var cluster in bitmap.Clusters)

{

br.BaseStream.Seek(cluster \* BytesPerCluster, SeekOrigin.Begin);

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

{

if (byteNumber >= bitmap.Size) return;

byte fourClustersInfo = br.ReadByte();

// Кластер представляется 2 битами, значит в байте - информация о 4 кластерах.

for (int j = 0; j < 4; j++)

{

// Обнуляем первые шесть битов, при этом получая 2 младших бита,

// представляющих информацию об одном кластере.

if ((fourClustersInfo & 0b00000011) == (byte)ClusterState.Free)

FreeClusters++; // Увеличиваем счётчик свободных класетров

else if ((fourClustersInfo & 0b00000011) == (byte)ClusterState.Service)

ServiceClusters++;

// Сдвигаемся на 2 бита вправо для получения информации о следующем кластере.

fourClustersInfo = (byte)(fourClustersInfo >> 2);

}

byteNumber++;

}

}

}

}

public static int GetBusySpace()

{

return SystemCalls.GetAllMftHeaders().Count(mftHeader => mftHeader.Sign == MftHeader.Signature.InUse) \*

Constants.MftRecordSize +

SystemCalls.GetDirectorySize(Constants.RootMftEntry);

}

public static int GetClusterCount() => VolumeSize / BytesPerCluster;

public static int GetBusyClusterCount() => GetClusterCount() - ServiceClusters - FreeClusters;

}

}

// SystemCalls.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using MyOS.FileSystem.SpecialDataTypes;

using MyOS.ViewModels;

using Path = MyOS.FileSystem.SpecialDataTypes.Path;

using Constants = MyOS.FileSystem.FileSystem.Constants;

namespace MyOS.FileSystem

{

static class SystemCalls

{

public static void Formatting(FormattingOptions options)

{

int mftAreaSize = options.VolumeSize / 10 - options.VolumeSize / 10 % Constants.MftRecordSize;

using (var fileStream = new FileStream(Constants.SystemFile, FileMode.Create, FileAccess.Write, FileShare.None))

fileStream.SetLength(options.VolumeSize);

// По умолчанию в системе создаётся администратор.

UserRecord administrator = new UserRecord("admin", "admin", 0, true, Constants.RootMftEntry);

MftHeader.Attribute systemReadOnlyHidden = MftHeader.Attribute.ReadOnly | MftHeader.Attribute.Hidden;

// Создание файла $MFT, представляющего централизованный каталог

// всех остальных файлов диска и себя самого.

// Первые 6 записей - метафайлы (имеют фиксированное положение на диске).

// 1 файл - запись о самом MFT.

// 2 файл - запись о копии первых записей MFT.

// 3 файл - $Volume.

// 4 файл - / (корневой каталог).

// 5 файл - $Bitmap (битовая карта).

// 6 файл - $Users (список пользователей системы).

MftHeader mft = new MftHeader("$MFT", "",

size: Constants.ServiceFileCount \* Constants.MftRecordSize,

attributes: systemReadOnlyHidden, userId: administrator.Id);

MftHeader mftMirr = new MftHeader("$MFTMirr", "",

size: Constants.ServiceFileCount \* Constants.MftRecordSize,

attributes: systemReadOnlyHidden, userId: administrator.Id);

MftHeader volume = new MftHeader("$Volume", "", size: 15, attributes: systemReadOnlyHidden,

userId: administrator.Id);

MftHeader root = new MftHeader("/", "", MftHeader.Attribute.Directory, userId: administrator.Id);

// Размер битовой карты = (количество\_кластеров \* 2 бита / 8 битов в байте = 25600 байтов).

int bitmapSize = options.VolumeSize / options.BytesPerCluster \* 2 / 8;

MftHeader bitmap = new MftHeader("$Bitmap", "", size: bitmapSize, attributes: systemReadOnlyHidden,

userId: administrator.Id);

MftHeader users = new MftHeader("$Users", "", size: UserRecord.Length, attributes: systemReadOnlyHidden,

userId: administrator.Id);

#region Записываем служебные записи в главную файловую таблицу MFT.

UpdateMftEntry(mft, Constants.MftEntry);

UpdateMftEntry(mftMirr, Constants.MftMirrEntry);

UpdateMftEntry(volume, Constants.VolumeMftEntry);

UpdateMftEntry(root, Constants.RootMftEntry);

UpdateMftEntry(bitmap, Constants.BitmapMftEntry);

UpdateMftEntry(users, Constants.UserListMftEntry);

UpdateRecord(Constants.UserListMftEntry, UserRecord.Empty, administrator);

#endregion

#region Записываем системную информацию в файлы.

List<int> mftMirrowClusters = new List<int>(GetClustersSequence(

CalculateClustersNumber(Constants.ServiceFileCount \* Constants.MftRecordSize, options.BytesPerCluster),

options.VolumeSize / options.BytesPerCluster / 2));

WriteClusterNumbers(Constants.MftMirrEntry, mftMirrowClusters.ToArray());

WriteMftMirrowData(new[] { mft, mftMirr, volume, root, bitmap, users }, mftMirrowClusters.ToArray(),

options.BytesPerCluster);

WriteVolumeData(options.VolumeName, options.FsName, options.BytesPerCluster);

List<int> bitmapClusters =

new List<int>(GetClustersSequence(CalculateClustersNumber(bitmapSize, options.BytesPerCluster),

mftAreaSize / options.BytesPerCluster));

WriteClusterNumbers(Constants.BitmapMftEntry, bitmapClusters.ToArray());

WriteInitialBitmap(bitmapClusters.ToArray(), options.VolumeSize, options.BytesPerCluster,

mftMirrowClusters.ToArray());

#endregion

}

static void WriteMftMirrowData(MftHeader[] mftHeaders, int[] mftMirrowClusters, int bytesPerCluster)

{

List<byte> bytes = new List<byte>();

foreach (var mftHeader in mftHeaders)

{

bytes.AddRange(mftHeader.GetBytes());

bytes.AddRange(new byte[Constants.MftRecordSize - MftHeader.Length]);

}

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

int needToWriteBytes = bytes.Count;

foreach (var block in mftMirrowClusters)

{

int byteCount = needToWriteBytes < bytesPerCluster

? needToWriteBytes

: bytesPerCluster;

bw.Seek(block \* bytesPerCluster, SeekOrigin.Begin);

bw.Write(bytes.GetRange(bytes.Count - needToWriteBytes, byteCount).ToArray());

needToWriteBytes -= bytesPerCluster;

}

}

}

static void WriteVolumeData(string volumeName, string fsName, int bytesPerCluster)

{

List<byte> volumeData = new List<byte>();

volumeData.AddRange((volumeName + fsName).GetFormatBytes(10));

volumeData.Add(0); // Том исправен.

volumeData.AddRange(BitConverter.GetBytes(bytesPerCluster));

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

bw.BaseStream.Seek(Constants.VolumeMftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

bw.Write(volumeData.ToArray());

}

}

static int[] GetClustersSequence(int number, int firstBlock)

{

int[] dataBlocks = new int[number];

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

dataBlocks[i] = firstBlock + i;

return dataBlocks;

}

public static void WriteInitialBitmap(int[] bitmapClusters, int volumeSize, int bytesPerCluster, int[] mftMirrowClusters)

{

int mftZoneSize = volumeSize / 10 - volumeSize / 10 % Constants.MftRecordSize;

int mftClusterСount = mftZoneSize / bytesPerCluster;

int clusterCount = volumeSize / bytesPerCluster;

int currentCluster = 0;

byte serviceCluster = 0b10;

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

foreach (var block in bitmapClusters)

{

bw.BaseStream.Seek(block \* bytesPerCluster, SeekOrigin.Begin);

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

{

// Кластер представляется 2 битами, значит в байте - информация о 4 кластерах.

byte[] clustersInfo = new byte[4]; // Массив, представляющий информацию о 4 кластерах.

for (int j = 0; j < 4; j++)

{

clustersInfo[j] = currentCluster < mftClusterСount ||

mftMirrowClusters.Contains(currentCluster) ||

bitmapClusters.Contains(currentCluster)

? serviceCluster

: (byte)0;

currentCluster++;

if (currentCluster >= clusterCount) return;

}

byte infoByte = 0;

// Заполняем побитово байт с информацией о кластерах.

// Порядок кластеров - обратный, для удобства считывания.

for (int k = clustersInfo.Length - 1; k >= 0; k--)

{

// В последние два бита записываем информацию о кластере.

infoByte = (byte)(infoByte | clustersInfo[k]);

// Сдвигаем байт влево на два бита, осаобождая место под информацию о следующем кластере.

if (k != 0) infoByte = (byte)(infoByte << 2);

}

bw.Write(infoByte);

}

}

}

}

public static int CalculateClustersNumber(int dataSize, int bytesPerCluster = 0)

{

if (dataSize <= Constants.MftRecordSize - MftHeader.Length)

return 0;

return (int)Math.Ceiling((double)dataSize / (bytesPerCluster == 0 ?

FileSystem.BytesPerCluster : bytesPerCluster));

}

public static List<int> GetFreeClusters(int requiredNumber)

{

List<int> clusterNumbers = new List<int>();

Data data = ReadMftData(Constants.BitmapMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int byteNumber = 0; // Номер прочитанного байта в битовой карте.

foreach (var block in data.Clusters)

{

br.BaseStream.Seek(block \* FileSystem.BytesPerCluster, SeekOrigin.Begin);

for (int i = 0; i < FileSystem.BytesPerCluster; i++)

{

if (byteNumber >= data.Size) return clusterNumbers;

byte fourClustersInfo = br.ReadByte();

// Кластер представляется 2 битами, значит в байте - информация о 4 кластерах.

for (int j = 0; j < 4; j++)

{

// Обнуляем первые шесть битов, при этом получая 2 младших бита,

// представляющих информацию об одном кластере.

if ((fourClustersInfo & 0b00000011) == (byte)ClusterState.Free)

{

clusterNumbers.Add(byteNumber \* 4 + j); // Добавляем номер свободного кластера.

if (clusterNumbers.Count == requiredNumber)

return clusterNumbers;

}

// Сдвигаемся на 2 бита вправо для получения информации о следующем кластере.

fourClustersInfo = (byte)(fourClustersInfo >> 2);

}

byteNumber++;

}

}

}

return clusterNumbers;

}

public static void SetClustersState(List<int> clusterNumbers, ClusterState state)

{

Data data = ReadMftData(Constants.BitmapMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

// Смещаемся на поле данных (на атрибут Data) записи $Bitmap.

foreach (var clusterNumber in clusterNumbers)

{

// Получаем номер байта, в котором записан данный кластер.

// В одном байте - информация о 4 кластерах.

int clusterByteNumber = clusterNumber / 4;

// Получаем порядковый номер кластера в данном байте.

int clusterOffset = clusterNumber % 4;

// Получаем номер блока данных битовой карты, в которой находится необходимый байт,

// и порядковый номер байта в блоке.

int blockNumber = clusterByteNumber / FileSystem.BytesPerCluster;

int byteOffset = clusterByteNumber % FileSystem.BytesPerCluster;

br.BaseStream.Seek(data.Clusters[blockNumber] \* FileSystem.BytesPerCluster + byteOffset,

SeekOrigin.Begin);

byte modifiedByte = ModifyByte(br.ReadByte(), clusterOffset, state);

BinaryWriter bw = new BinaryWriter(br.BaseStream);

bw.BaseStream.Position -= 1; // Смещаемся на один байт назад для его перезаписи.

bw.Write(modifiedByte);

if (state == ClusterState.Free) FileSystem.FreeClusters++;

else FileSystem.FreeClusters--;

}

}

}

private static byte ModifyByte(byte sourceByte, int clusterNumberInByte, ClusterState state)

{

// Создаём байт, содержащий в двух младших битах информацию о состоянии кластера.

byte stateByte = (byte)state;

// Сдвигаем биты кластера на заданное смещение.

for (int j = clusterNumberInByte; j > 0; j--)

stateByte = (byte)(stateByte << 2);

byte byteMask = 0;

if (clusterNumberInByte == 0)

byteMask = 0b11\_11\_11\_00;

if (clusterNumberInByte == 1)

byteMask = 0b11\_11\_00\_11;

if (clusterNumberInByte == 2)

byteMask = 0b11\_00\_11\_11;

if (clusterNumberInByte == 3)

byteMask = 0b00\_11\_11\_11;

sourceByte &= byteMask;

sourceByte |= stateByte;

return sourceByte;

}

public static void UpdateMftEntry(MftHeader mftHeader, int mftEntry)

{

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

bw.BaseStream.Seek(mftEntry \* Constants.MftRecordSize, SeekOrigin.Begin);

if (mftHeader == null)

bw.Write((byte)MftHeader.Signature.NotUsed);

else bw.Write(mftHeader.GetBytes());

}

}

public static List<MftHeader> GetAllMftHeaders()

{

List<MftHeader> mftHeaders = new List<MftHeader>();

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

br.BaseStream.Seek(Constants.MftEntry \* Constants.MftRecordSize, SeekOrigin.Begin);

MftHeader mft = new MftHeader(br.ReadBytes(MftHeader.Length));

mftHeaders.Add(mft);

for (int i = Constants.MftRecordSize; i < mft.Size;)

{

br.BaseStream.Seek(Constants.MftRecordSize - MftHeader.Length, SeekOrigin.Current);

mftHeaders.Add(new MftHeader(br.ReadBytes(MftHeader.Length)));

i += Constants.MftRecordSize;

}

return mftHeaders;

}

}

public static MftHeader GetMftHeader(int mftEntry)

{

if (mftEntry < 0) return null;

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

br.BaseStream.Seek(mftEntry \* Constants.MftRecordSize, SeekOrigin.Begin);

MftHeader mftHeader = new MftHeader(br.ReadBytes(MftHeader.Length));

if (mftHeader.Sign == MftHeader.Signature.InUse)

return mftHeader;

return null;

}

}

public static int FindDirectoryMftEntry(Path path)

{

string[] directoryList = path.DirectoriesList.ToArray();

int mftEntry = Constants.RootMftEntry;

bool directoryFound = directoryList.Length == 0;

// Проходим по всем директориям указанного пути

// для поиска конечной папки.

foreach (var directoryName in directoryList)

{

directoryFound = false;

DirectoryRecord directoryRecord = (DirectoryRecord)GetRecord(directoryName,

mftEntry, DirectoryRecord.Length);

if (directoryRecord == null) return -1;

if (directoryRecord.HasAttribute(MftHeader.Attribute.Directory))

{

directoryFound = true;

mftEntry = directoryRecord.MftEntry;

}

}

if (directoryFound) return mftEntry;

return -1;

}

public static IRecord GetRecord(string name, int targetMftEntry, int recordLength)

{

if (targetMftEntry < 0) return null;

Data data = ReadMftData(targetMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int offset = 0; // Смещение относительно начала блока данных Mft-записи.

if (data.Clusters.Count == 0) // Данные содержатся в одной Mft-записи.

{

br.BaseStream.Seek(targetMftEntry \* Constants.MftRecordSize + MftHeader.Length,

SeekOrigin.Begin);

while (offset < data.Size)

{

byte[] recordBytes = br.ReadBytes(recordLength);

// Если данная последовательность байтов не пустая, считываем

// и сравниваем запись по имени записи (либо имя пользователя, либо имя папки/файла).

if (!recordBytes.SequenceEqual(new byte[recordLength]))

{

IRecord record;

if (recordBytes.Length == UserRecord.Length)

record = new UserRecord(recordBytes);

else record = new DirectoryRecord(recordBytes);

if (record.EqualsByName(name))

return record;

}

offset += recordLength;

}

}

else

{ // Данные содержатся в области файлов.

int blockIndex = 0;

br.BaseStream.Seek(data.Clusters[blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

while (offset < data.Size)

{

int availableBytes = FileSystem.BytesPerCluster - offset % FileSystem.BytesPerCluster;

List<byte> recordBytes = new List<byte>();

if (availableBytes >= recordLength) // Следующая запись помещается в текущий блок данных.

recordBytes.AddRange(br.ReadBytes(recordLength));

else

{ // Следующая запись по частям содержится в двух блоках.

recordBytes.AddRange(br.ReadBytes(availableBytes));

br.BaseStream.Seek(data.Clusters[++blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

if (recordLength - recordBytes.Count > 0)

recordBytes.AddRange(br.ReadBytes(recordLength - recordBytes.Count));

}

if (!recordBytes.ToArray().SequenceEqual(new byte[recordLength]))

{

IRecord record;

if (recordBytes.Count == UserRecord.Length)

record = new UserRecord(recordBytes.ToArray());

else record = new DirectoryRecord(recordBytes.ToArray());

if (record.EqualsByName(name))

return record;

}

offset += recordLength;

}

}

}

return null;

}

public static List<ExplorerFile> GetFileList(Path path)

{

// Получаем весь список файлов для отображения в проводнике по указанному пути.

List<DirectoryRecord> directoryRecords = GetDirectoryRecords(path);

List<ExplorerFile> files = new List<ExplorerFile>();

foreach (var record in directoryRecords)

{

MftHeader mftHeader = GetMftHeader(record.MftEntry);

files.Add(new ExplorerFile(mftHeader, record.MftEntry));

}

return files;

}

/// <summary>

/// Получает файлы директории по указанному пути.

/// </summary>

/// <param name="path">Заданный путь.</param>

/// <returns></returns>

public static List<DirectoryRecord> GetDirectoryRecords(Path path)

{

List<DirectoryRecord> directoryRecords = new List<DirectoryRecord>();

int directoryMftEntry = FindDirectoryMftEntry(path);

if (directoryMftEntry == -1) return directoryRecords;

return GetAllRecords<DirectoryRecord>(directoryMftEntry);

}

/// <summary>

/// Получает все записи в Mft-записи по указанному номеру

/// </summary>

/// <typeparam name="T">Тип записи: пользователь или запись директории.</typeparam>

/// <param name="mftEntry">Номер Mft-записи, информацию которой необходимо получить.</param>

/// <returns></returns>

public static List<T> GetAllRecords<T>(int mftEntry) where T : IRecord

{

int recordLength = typeof(T) == typeof(UserRecord) ? UserRecord.Length : DirectoryRecord.Length;

List<T> records = new List<T>();

Data data = ReadMftData(mftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int offset = 0;

if (data.Clusters.Count == 0)

{

br.BaseStream.Seek(mftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

while (offset < data.Size)

{

byte[] recordBytes = br.ReadBytes(recordLength);

if (!recordBytes.SequenceEqual(new byte[recordLength]))

{

IRecord record;

if (recordLength == UserRecord.Length)

record = new UserRecord(recordBytes.ToArray());

else record = new DirectoryRecord(recordBytes.ToArray());

records.Add((T)record);

}

offset += recordLength;

}

}

else

{

int blockIndex = 0;

br.BaseStream.Seek(data.Clusters[blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

while (offset < data.Size)

{

int availableBytes = FileSystem.BytesPerCluster - offset % FileSystem.BytesPerCluster;

List<byte> recordBytes = new List<byte>();

if (availableBytes >= recordLength)

recordBytes.AddRange(br.ReadBytes(recordLength));

else

{

recordBytes.AddRange(br.ReadBytes(availableBytes));

br.BaseStream.Seek(data.Clusters[++blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

recordBytes.AddRange(br.ReadBytes(recordLength - recordBytes.Count));

}

if (!recordBytes.SequenceEqual(new byte[recordLength]))

{

IRecord record;

if (recordLength == UserRecord.Length)

record = new UserRecord(recordBytes.ToArray());

else record = new DirectoryRecord(recordBytes.ToArray());

records.Add((T)record);

}

offset += recordLength;

}

}

}

return records;

}

public static int GetDirectorySize(int mftEntry)

{

int directorySize = 0;

List<DirectoryRecord> records = GetAllRecords<DirectoryRecord>(mftEntry);

foreach (var record in records)

{

if (record.HasAttribute(MftHeader.Attribute.Directory))

directorySize += GetDirectorySize(record.MftEntry);

else directorySize += GetMftHeader(record.MftEntry).Size;

}

return directorySize;

}

/// <summary>

/// Обновляет запись на диске.

/// </summary>

/// <typeparam name="T">Тип записи: пользователь или запись директории.</typeparam>

/// <param name="targetMftEntry">Номер Mft-записи, которую необходимо обновить.</param>

/// <param name="before">Текущая запись.</param>

/// <param name="after">Обновлённая запсиь.</param>

public static void UpdateRecord<T>(int targetMftEntry, T before, T after) where T : IRecord

{

if (before == null || after == null)

throw new ArgumentException("Обновляемая или обновлённая записи не определены.");

if (targetMftEntry < 0)

throw new ArgumentException("Файл, в котором следует обновить запись, не найден!");

if (before.GetType() != after.GetType())

throw new ArgumentException("Обновляемая и обновлённая записи имеют разные типы.");

int recordLength = before.GetBytes().Length;

Data data = ReadMftData(targetMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int offset = 0;

if (data.Clusters.Count == 0)

{

br.BaseStream.Seek(targetMftEntry \* Constants.MftRecordSize + MftHeader.Length,

SeekOrigin.Begin);

while (offset < data.Size)

{

if (before.Equals(br.ReadBytes(recordLength)))

{

using (BinaryWriter bw = new BinaryWriter(br.BaseStream))

{

bw.BaseStream.Position -= recordLength;

bw.Write(after.GetBytes());

}

return;

}

offset += recordLength;

}

}

else

{

int blockIndex = 0;

br.BaseStream.Seek(data.Clusters[blockIndex] \* FileSystem.BytesPerCluster, SeekOrigin.Begin);

while (offset < data.Size)

{

int availableBytes = FileSystem.BytesPerCluster - offset % FileSystem.BytesPerCluster;

if (availableBytes >= recordLength)

{

if (before.Equals(br.ReadBytes(recordLength)))

{

using (BinaryWriter bw = new BinaryWriter(br.BaseStream))

{

bw.BaseStream.Position -= recordLength;

bw.Write(after.GetBytes());

}

return;

}

}

else

{

List<byte> recordBytes = new List<byte>(br.ReadBytes(availableBytes));

br.BaseStream.Seek(data.Clusters[++blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

recordBytes.AddRange(br.ReadBytes(recordLength - availableBytes));

if (before.Equals(recordBytes.ToArray()))

{

using (BinaryWriter bw = new BinaryWriter(br.BaseStream))

{

bw.BaseStream.Seek(data.Clusters[--blockIndex] \* FileSystem.BytesPerCluster +

offset % FileSystem.BytesPerCluster, SeekOrigin.Begin);

bw.Write(after.GetBytes().GetRange(0, availableBytes));

bw.BaseStream.Seek(data.Clusters[++blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

bw.Write(after.GetBytes().GetRange(availableBytes, recordLength - availableBytes));

}

return;

}

}

offset += recordLength;

}

}

}

}

private static void WriteClusterNumbers(int mftEntry, int[] clusters)

{

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

bw.Seek(mftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

foreach (var cluster in clusters)

bw.Write(cluster);

}

}

public static void WriteFileData(int mftEntry, byte[] fileData)

{

if (fileData == null) return;

Data data = ReadMftData(mftEntry);

using (BinaryWriter bw = new BinaryWriter(File.Open(Constants.SystemFile, FileMode.Open)))

{

bw.Seek(mftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

if (data.Clusters.Count == 0)

bw.Write(fileData);

else

{

int needToWriteBytes = fileData.Length;

foreach (var block in data.Clusters)

{

int byteCount = needToWriteBytes < FileSystem.BytesPerCluster

? needToWriteBytes

: FileSystem.BytesPerCluster;

bw.Seek(block \* FileSystem.BytesPerCluster, SeekOrigin.Begin);

bw.Write(fileData.GetRange(fileData.Length - needToWriteBytes, byteCount));

needToWriteBytes -= byteCount;

}

}

}

}

public static byte[] ReadFileData(int mftEntry)

{

List<byte> dataBytes = new List<byte>();

Data data = ReadMftData(mftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

br.BaseStream.Seek(mftEntry \* Constants.MftRecordSize + MftHeader.Length,

SeekOrigin.Begin);

if (data.Clusters.Count == 0)

dataBytes.AddRange(br.ReadBytes(data.Size));

else

{

int needToReadBytes = data.Size;

foreach (var block in data.Clusters)

{

int byteCount = needToReadBytes > FileSystem.BytesPerCluster

? FileSystem.BytesPerCluster

: needToReadBytes;

br.BaseStream.Seek(block \* FileSystem.BytesPerCluster, SeekOrigin.Begin);

dataBytes.AddRange(br.ReadBytes(byteCount));

needToReadBytes -= byteCount;

}

}

return dataBytes.ToArray();

}

}

public static Data ReadMftData(int mftEntry)

{

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

br.BaseStream.Seek(mftEntry \* Constants.MftRecordSize + 33, SeekOrigin.Begin);

int dataSize = br.ReadInt32();

br.BaseStream.Seek(13, SeekOrigin.Current);

int clustersCount = CalculateClustersNumber(dataSize);

int[] blocks = new int[clustersCount]; // Список номеров блоков, в которых содержатся данные.

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

blocks[i] = br.ReadInt32();

return new Data(dataSize, blocks);

}

}

/// <summary>

/// Получает незанятый номер для новой Mft-записи.

/// </summary>

/// <returns></returns>

static int GetNewMftEntry()

{

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

br.BaseStream.Seek(Constants.MftEntry \* Constants.MftRecordSize, SeekOrigin.Begin);

MftHeader mftItselfHeader = new MftHeader(br.ReadBytes(MftHeader.Length));

// Просматриваем все записи MFT, начиная с первого доступного номера, т.е. пропуская метафайлы.

int mftEntry = Constants.ServiceFileCount;

br.BaseStream.Seek(mftEntry \* Constants.MftRecordSize, SeekOrigin.Begin);

while (mftEntry \* Constants.MftRecordSize < mftItselfHeader.Size)

{

if (br.ReadByte() == (byte)MftHeader.Signature.NotUsed)

return mftEntry;

br.BaseStream.Seek(Constants.MftRecordSize - 1, SeekOrigin.Current);

mftEntry++;

}

// Проверяем, будет ли главный файл MFT помещаться в MFT зону,

// если увечилить его размер ещё на одну запись.

if ((mftEntry + 1) \* Constants.MftRecordSize <= FileSystem.MftAreaSize)

return mftEntry;

return -1;

}

}

/// <summary>

/// Получает атрибуты размещения новой записи в памяти.

/// </summary>

/// <exception cref="FsException">Недостаточно свободного пространства в Mft-зоне.</exception>

/// <exception cref="FsException">Превышен максимально допустимый размер директории.</exception>

/// <exception cref="FsException">Недостаточно свободного пространства в области файлов.</exception>

/// <param name="targetMftEntry">Номер Mft-записи, в которую следует помесить данную запись.</param>

/// <param name="recordLength">Длина в байтах данной записи.</param>

/// <returns></returns>

public static MemoryAllocation TryToAllocateMemory(int targetMftEntry, int recordLength)

{

int mftEntry = GetNewMftEntry();

if (mftEntry == -1)

throw new FsException(FsException.Code.NoSystemSpace, "");

MemoryAllocation.Attribute allocationAttributes = 0;

// Устанавливаем флаг увеличения Mft, если новая Mft-запись не помещается в Mft-файл.

if (GetMftHeader(Constants.MftEntry).Size == mftEntry \* Constants.MftRecordSize)

allocationAttributes |= MemoryAllocation.Attribute.NeedToIncreaseMftSize;

Data data = ReadMftData(targetMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int offset = 0;

if (data.Clusters.Count == 0)

{

br.BaseStream.Seek(targetMftEntry \* Constants.MftRecordSize + MftHeader.Length,

SeekOrigin.Begin);

while (offset < data.Size)

{

if (br.ReadBytes(recordLength).SequenceEqual(new byte[recordLength]))

return new MemoryAllocation(mftEntry, allocationAttributes);

offset += recordLength;

}

// Проверяем, поместится ли новая запись в атрибут Data Mft-записи директории.

if (offset + recordLength > Constants.MftRecordSize - MftHeader.Length)

allocationAttributes |= MemoryAllocation.Attribute.NeedNewCluster;

}

else

{

int blockIndex = 0;

br.BaseStream.Seek(data.Clusters[blockIndex] \* FileSystem.BytesPerCluster, SeekOrigin.Begin);

while (offset < data.Size)

{

int availableBytes = FileSystem.BytesPerCluster - offset % FileSystem.BytesPerCluster;

if (availableBytes >= recordLength)

{

if (br.ReadBytes(recordLength).SequenceEqual(new byte[recordLength]))

return new MemoryAllocation(mftEntry, allocationAttributes);

}

else

{

List<byte> recordBytes = new List<byte>(br.ReadBytes(availableBytes));

br.BaseStream.Seek(data.Clusters[blockIndex + 1] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

recordBytes.AddRange(br.ReadBytes(recordLength - availableBytes));

if (recordBytes.SequenceEqual(new byte[recordLength]))

return new MemoryAllocation(mftEntry, allocationAttributes);

}

offset += recordLength;

}

// Проверяем, поместится ли новая запись в уже выделенные кластеры.

if (offset + recordLength > data.Clusters.Count \* FileSystem.BytesPerCluster)

allocationAttributes |= MemoryAllocation.Attribute.NeedNewCluster;

}

// Так как прошли по всей директории и не нашли свободного места,

// то устанавливаем флаг необходимости увеличения размера директории.

allocationAttributes |= MemoryAllocation.Attribute.NeedToIncreaseDirectorySize;

// Проверяем, есть ли свободные кластеры в области файлов,

// если был установлен флаг выделения нового кластера.

if ((allocationAttributes & MemoryAllocation.Attribute.NeedNewCluster) != 0)

{

if (FileSystem.FreeClusters == 0) throw new FsException(FsException.Code.NoFreeSpace, "");

// Проверяем, поместится ли в атрибут Data ссылка на ещё один выделенный кластер.

if (data.Clusters.Count + 1 > (Constants.MftRecordSize - MftHeader.Length) / 4)

throw new FsException(FsException.Code.NoSystemSpace, "");

}

return new MemoryAllocation(mftEntry, allocationAttributes);

}

}

/// <summary>

/// Увеличивает размеры необходимых файлов с учётом атрибутов распределения нового файла.

/// </summary>

/// <param name="allocation">Атрибуты распределения файла в памяти.</param>

/// <param name="targetMftEntry">Номер Mft-записи, в которую следует поместить данную запись.</param>

/// <param name="recordLength">Номер новой записи.</param>

public static void AllocateRecord(MemoryAllocation allocation, int targetMftEntry, int recordLength)

{

Data data = ReadMftData(targetMftEntry);

byte[] dataBytes = null;

// При необходимости выделяем новый кластер под подительскую директорию.

if ((allocation.Attributes & MemoryAllocation.Attribute.NeedNewCluster) != 0)

{

int newBlock = GetFreeClusters(1)[0];

SetClustersState(new List<int> { newBlock }, ClusterState.Busy);

if (data.Clusters.Count == 0)

dataBytes = ReadFileData(targetMftEntry);

data.Clusters.Add(newBlock);

WriteClusterNumbers(targetMftEntry, data.Clusters.ToArray());

}

MftHeader directoryMftHeader = GetMftHeader(targetMftEntry);

directoryMftHeader.ModificationDate = new MyDateTime(DateTime.Now);

// При необходимости увеличиваем размер родительской директории.

if ((allocation.Attributes & MemoryAllocation.Attribute.NeedToIncreaseDirectorySize) != 0)

directoryMftHeader.Size += recordLength;

UpdateMftEntry(directoryMftHeader, targetMftEntry);

// Если было необходимо выделить новый кластер для директории и он был первым

// (то есть до этого данные полностью помещались в Mft-запись),

// то перезаписываем их в область файлов.

if (dataBytes != null) WriteFileData(targetMftEntry, dataBytes);

// При необходимости увеличиваем размер зоны Mft.

if ((allocation.Attributes & MemoryAllocation.Attribute.NeedToIncreaseMftSize) != 0)

{

MftHeader mftHeader = GetMftHeader(Constants.MftEntry);

mftHeader.Size += Constants.MftRecordSize;

UpdateMftEntry(mftHeader, Constants.MftEntry);

}

}

public static int Create(Path path, string name, string extension,

MftHeader.Attribute attributes = MftHeader.Attribute.None, int userId = -1)

{

MftHeader newMftHeader = new MftHeader(name, extension, attributes, userId: userId);

// Получаем родительскую директорию.

int directoryMftEntry = FindDirectoryMftEntry(path);

if (directoryMftEntry == -1)

throw new FsException(FsException.Code.NotFound, path.GetAbsolutePath(newMftHeader.FileName));

MftHeader parentDirectory = GetMftHeader(directoryMftEntry);

// Проверяем права доступа для директории.

if (!parentDirectory.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Write))

throw new FsException(FsException.Code.NoWritePermission,

path.CurrentPath);

// Получаем параметры распределения памяти.

MemoryAllocation allocation = TryToAllocateMemory(directoryMftEntry, DirectoryRecord.Length);

// Проверяем, не существует ли уже файл с таким именем.

DirectoryRecord record =

(DirectoryRecord)GetRecord(newMftHeader.GetFullName(), directoryMftEntry, DirectoryRecord.Length);

if (record != null)

throw new FsException(record.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Code.FolderExists

: FsException.Code.FileExists, record.GetFullName());

// Добавляем новую запись в Mft.

UpdateMftEntry(newMftHeader, allocation.MftEntry);

// Увеличиваем при необходимости размеры файлов.

AllocateRecord(allocation, directoryMftEntry, DirectoryRecord.Length);

// Добавляем запись в родительскую директорию на пустое место.

UpdateRecord(directoryMftEntry, DirectoryRecord.Empty,

new DirectoryRecord(newMftHeader, allocation.MftEntry));

return allocation.MftEntry;

}

public static void Copy(Path path, DirectoryRecord record)

{

// Получаем файл для копирования.

DirectoryRecord bufferRecord =

(DirectoryRecord)GetRecord(record.GetFullName(), FindDirectoryMftEntry(path), DirectoryRecord.Length);

MftHeader mftHeader = GetMftHeader(bufferRecord.MftEntry);

// Пользователь должен обладать полным доступом к файлу для его копирования.

if (!mftHeader.HasPermissions(FileSystem.CurrentUser, Permission.Rights.FullControl))

throw new FsException(FsException.Code.NoFullControlPermission, path.GetAbsolutePath(mftHeader.GetFullName()));

FileSystem.Buffer = new SystemBuffer { Path = new Path(path), Record = new DirectoryRecord(bufferRecord) };

}

public static List<DirectoryRecord> CopyRecursively(Path sourcePath, Path destinationPath)

{

List<DirectoryRecord> copies = new List<DirectoryRecord>();

List<DirectoryRecord> records = GetDirectoryRecords(sourcePath);

foreach (var record in records)

{

MftHeader source = GetMftHeader(record.MftEntry);

try

{

// Проверяем, существует ли файл в исходном расположении.

if (source == null)

throw new FsException(FsException.Code.NotFound,

sourcePath.GetAbsolutePath(record.GetFullName()));

// Проверяем доступ к файлу. Пользователь должен обладать полным доступом к файлу для его копирования.

if (!source.HasPermissions(FileSystem.CurrentUser, Permission.Rights.FullControl))

throw new FsException(FsException.Code.NoFullControlPermission,

sourcePath.GetAbsolutePath(source.GetFullName()));

// Пробуем создать копию файла в заданном располжении.

var mftEntry = Create(destinationPath, source.FileName, source.Extension,

(MftHeader.Attribute) source.Attributes);

byte[] data;

// В зависимости от типа записи (директория или файл), получаем её данные.

if (record.HasAttribute(MftHeader.Attribute.Directory))

{ // Для директории копируем перечень файлов.

Path from = new Path(sourcePath);

from.Add(record.FileName);

Path to = new Path(destinationPath);

to.Add(source.FileName);

// Получаем список дочерних файлов.

List<DirectoryRecord> childCopyFileList = CopyRecursively(from, to);

// Создаём содержимое директории-копии.

List<byte> directoryBytes = new List<byte>();

foreach (var childFileCopy in childCopyFileList)

directoryBytes.AddRange(childFileCopy.GetBytes());

data = directoryBytes.ToArray();

}

else data = ReadFileData(record.MftEntry); // Копируем данные файла.

MftHeader copyMftHeader = GetMftHeader(mftEntry);

// Записываем данные записи.

Save(new DirectoryRecord(copyMftHeader, mftEntry), data);

// Обновляем созданную копию записи, изменяя её размер.

copyMftHeader.Size = data.Length;

UpdateMftEntry(copyMftHeader, mftEntry);

copies.Add(new DirectoryRecord(copyMftHeader, mftEntry));

}

catch (FsException fsException)

{

// При возникновении исключения, выводим сообщение об ошибке,

// пропускаем файл и идём дальше по перечню файлов директории.

fsException.ShowError(FsException.Command.Copy,

record.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Element.Folder

: FsException.Element.File);

}

}

return copies;

}

public static void Paste(Path path)

{

if (FileSystem.Buffer.Path.GetAbsolutePath(FileSystem.Buffer.Record.GetFullName()) == path.CurrentPath)

throw new FsException(FsException.Code.EndlessCopy,

FileSystem.Buffer.Record.GetFullName());

MftHeader source = GetMftHeader(FileSystem.Buffer.Record.MftEntry);

// Проверяем, существует ли файл в исходном расположении.

if (source == null)

throw new FsException(FsException.Code.NotFound,

path.GetAbsolutePath(FileSystem.Buffer.Record.GetFullName()));

int directoryMftEntry = FindDirectoryMftEntry(path);

// Проверяем доступ к файлу. Пользователь должен обладать полным доступом к файлу для его копирования.

if (!source.HasPermissions(FileSystem.CurrentUser, Permission.Rights.FullControl))

throw new FsException(FsException.Code.NoFullControlPermission,

path.GetAbsolutePath(source.GetFullName()));

// Проверяем, существует ли в заданном расположении файл или директория с указанным именем.

// Пока находим такие файлы и длина имени не превосходит максимальную в системе, изменяем имя.

string sourceFullName = source.GetFullName();

string name = source.FileName;

int attempt = 1;

while (GetRecord(source.GetFullName(), directoryMftEntry, DirectoryRecord.Length) != null)

{

source.FileName = name + " (" + attempt++ + ")";

if (source.FileName.Length > 26)

throw new FsException(FsException.Code.NameTooLong, source.FileName);

}

// Пробуем создать файл по указанному пути.

var mftEntry = Create(path, source.FileName, source.Extension,

(MftHeader.Attribute)source.Attributes);

byte[] data;

if (source.HasAttribute(MftHeader.Attribute.Directory))

{

// Если данный файл - директория, то копируем всё её содержимое.

Path from = new Path(FileSystem.Buffer.Path);

from.Add(FileSystem.Buffer.Record.FileName);

Path to = new Path(path);

to.Add(source.FileName);

// Получаем список дочерних файлов.

List<DirectoryRecord> childCopyFileList = CopyRecursively(from, to);

// Записываем перечень файлов в директрию.

List<byte> directoryBytes = new List<byte>();

foreach (var childFileCopy in childCopyFileList)

directoryBytes.AddRange(childFileCopy.GetBytes());

data = directoryBytes.ToArray();

if (data.Length != GetAllRecords<DirectoryRecord>(FileSystem.Buffer.Record.MftEntry).Count \*

DirectoryRecord.Length)

throw new FsException(FsException.Code.IncompleteCopying,

FileSystem.Buffer.Path.GetAbsolutePath(sourceFullName));

}

else data = ReadFileData(FileSystem.Buffer.Record.MftEntry);

MftHeader copyMftHeader = GetMftHeader(mftEntry);

Save(new DirectoryRecord(source, mftEntry), data);

copyMftHeader.Size = data.Length;

UpdateMftEntry(copyMftHeader, mftEntry);

}

public static void Rename(Path path, DirectoryRecord record, string newFullName)

{

// Проверяем права доступа для директории.

int directoryMftEntry = FindDirectoryMftEntry(path);

MftHeader parentDirectory = GetMftHeader(directoryMftEntry);

if (!parentDirectory.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Write))

throw new FsException(FsException.Code.NoWritePermission, path.CurrentPath);

// Проверяем права доступа для заданной записи.

MftHeader file = GetMftHeader(record.MftEntry);

if (!file.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Write))

throw new FsException(FsException.Code.NoWritePermission, path.GetAbsolutePath(file.GetFullName()));

DirectoryRecord newRecord = (DirectoryRecord)GetRecord(newFullName, directoryMftEntry, DirectoryRecord.Length);

if (newRecord != null)

{

throw new FsException(newRecord.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Code.FolderExists

: FsException.Code.FileExists, newRecord.GetFullName());

}

newFullName.ParseFullName(record.HasAttribute(MftHeader.Attribute.Directory), out var name, out var extension);

file.FileName = name;

file.Extension = extension;

UpdateMftEntry(file, record.MftEntry);

UpdateRecord(directoryMftEntry, record, new DirectoryRecord(file, record.MftEntry));

}

public static void Save(DirectoryRecord record, byte[] fileData)

{

Data data = ReadMftData(record.MftEntry);

int requiredClusterCount = CalculateClustersNumber(fileData.Length) - data.Clusters.Count;

if (requiredClusterCount > 0)

{

List<int> clusterNumbers = GetFreeClusters(requiredClusterCount);

if (clusterNumbers.Count < requiredClusterCount)

throw new FsException(FsException.Code.NoFreeSpace, "");

if (data.Clusters.Count + requiredClusterCount > (Constants.MftRecordSize - MftHeader.Length) / 4)

throw new FsException(FsException.Code.NoSystemSpace, "");

data.Clusters.AddRange(clusterNumbers);

SetClustersState(clusterNumbers, ClusterState.Busy);

}

else if (requiredClusterCount < 0)

{

SetClustersState(data.Clusters.GetRange(data.Clusters.Count + requiredClusterCount,

Math.Abs(requiredClusterCount)), ClusterState.Free);

data.Clusters.RemoveRange(data.Clusters.Count + requiredClusterCount,

Math.Abs(requiredClusterCount));

}

MftHeader file = GetMftHeader(record.MftEntry);

if (file.HasAttribute(MftHeader.Attribute.ReadOnly))

throw new FsException(FsException.Code.ReadOnly, file.GetFullName());

if (!file.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Write))

throw new FsException(FsException.Code.NoWritePermission, file.GetFullName());

file.ModificationDate = new MyDateTime(DateTime.Now);

file.Size = fileData.Length;

UpdateMftEntry(file, record.MftEntry);

WriteClusterNumbers(record.MftEntry, data.Clusters.ToArray());

WriteFileData(record.MftEntry, fileData);

}

public static void Delete(Path path, DirectoryRecord record)

{

// Проверяем, есть ли право на запись родительской директории.

int directoryMftEntry = FindDirectoryMftEntry(path);

MftHeader parentDirectory = GetMftHeader(directoryMftEntry);

if (directoryMftEntry != Constants.RootMftEntry &&

!parentDirectory.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Write))

throw new FsException(FsException.Code.NoWritePermission, path.CurrentPath);

// Проверяем, есть ли право на изменение файла.

MftHeader mftHeader = GetMftHeader(record.MftEntry);

if (!mftHeader.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Modify))

throw new FsException(FsException.Code.NoModifyPermission,

path.GetAbsolutePath(mftHeader.GetFullName()));

if (record.HasAttribute(MftHeader.Attribute.Directory))

{ // Если удаляемая запись является директорией, удаляем её содержимое.

Path from = new Path(path);

from.Add(record.FileName);

int deletedFileCount = DeleteRecursively(from);

// Если не все файлы директории были удалены, прекращаем операцию удаления,

// поднимаясь на верхний уровень.

if (deletedFileCount < GetAllRecords<DirectoryRecord>(record.MftEntry).Count)

throw new FsException(FsException.Code.IncompleteDeleting, record.FileName);

// Если была удалена домашняя директория пользователя,

// то изменяем номер Mft-записи в его заявке.

List<UserRecord> users = GetAllRecords<UserRecord>(Constants.UserListMftEntry);

foreach (var user in users)

{

if (user.HomeDirectoryMftEntry == record.MftEntry)

{

UserRecord updatedUserRecord =

new UserRecord(user) {HomeDirectoryMftEntry = Constants.RootMftEntry};

UpdateRecord(Constants.UserListMftEntry, user, updatedUserRecord);

}

}

}

// Очищаем кластеры.

Data data = ReadMftData(record.MftEntry);

SetClustersState(data.Clusters, ClusterState.Free);

// Удаляем запись в родительской директории.

UpdateRecord(directoryMftEntry, GetRecord(record.GetFullName(), directoryMftEntry,

DirectoryRecord.Length), DirectoryRecord.Empty);

// Ставим пометку о неиспользуемой записи MFT.

UpdateMftEntry(null, record.MftEntry);

// Изменяем время последней модификации родительской директории

MftHeader directoryHeader = GetMftHeader(directoryMftEntry);

directoryHeader.ModificationDate = new MyDateTime(DateTime.Now);

UpdateMftEntry(directoryHeader, directoryMftEntry);

}

static int DeleteRecursively(Path directoryPath)

{

int deletedRecordCount = 0;

bool canDelete = true;

List<DirectoryRecord> records = GetDirectoryRecords(directoryPath);

foreach (var record in records)

{

MftHeader mftHeader = GetMftHeader(record.MftEntry);

if (!mftHeader.HasPermissions(FileSystem.CurrentUser, Permission.Rights.Modify))

{

// Если не хватает прав на изменения данного элемента,

// выводим сообщений об ошибке и отменяем удаление этой записи.

new FsException(FsException.Code.NoModifyPermission,

directoryPath.GetAbsolutePath(mftHeader.GetFullName())).ShowError(FsException.Command.Delete,

record.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Element.Folder

: FsException.Element.File);

canDelete = false;

}

if (record.HasAttribute(MftHeader.Attribute.Directory))

{

// Если удаляемая запись является директорией, удаляем её содержимое.

Path from = new Path(directoryPath);

from.Add(record.FileName);

int childRecordCount = DeleteRecursively(from);

if (childRecordCount \* DirectoryRecord.Length < mftHeader.Size)

// Если не все файлы папки были удалены, отменяем удаление только этой записи.

canDelete = false;

}

if (canDelete)

{

// Очищаем кластеры.

Data data = ReadMftData(record.MftEntry);

SetClustersState(data.Clusters, ClusterState.Free);

// Удаляем запись в родительской директории.

int directoryMftEntry = FindDirectoryMftEntry(new Path(directoryPath));

UpdateRecord(directoryMftEntry,

GetRecord(record.GetFullName(), directoryMftEntry, DirectoryRecord.Length),

DirectoryRecord.Empty);

// Ставим пометку о неиспользуемой записи MFT.

UpdateMftEntry(null, record.MftEntry);

// Изменяем время последней модификации родительской директории

MftHeader directoryHeader = GetMftHeader(directoryMftEntry);

directoryHeader.ModificationDate = new MyDateTime(DateTime.Now);

UpdateMftEntry(directoryHeader, directoryMftEntry);

deletedRecordCount++;

}

}

return deletedRecordCount;

}

public static void SignIn(string name, string password)

{

UserRecord user = (UserRecord)GetRecord(name, Constants.UserListMftEntry, UserRecord.Length);

if (user == null || !user.PasswordHash.SequenceEqual(password.ComputeHash())) throw new FsException(FsException.Code.UserNull, "");

FileSystem.CurrentUser = user;

}

public static void SignUp(string name, string password, bool isAdministrator)

{

if (GetRecord(name, Constants.UserListMftEntry, UserRecord.Length) != null)

throw new FsException(FsException.Code.UserExists, name);

int id = GetAvailableId();

if (id > Constants.MaxUserId) throw new FsException(FsException.Code.MaxUserCount, "");

MemoryAllocation fileAllocation;

try

{

fileAllocation = TryToAllocateMemory(Constants.UserListMftEntry, UserRecord.Length);

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Create, FsException.Element.User);

return;

}

int homeDirectoryMftEntry;

try

{

homeDirectoryMftEntry = Create(new Path(), name, "", MftHeader.Attribute.Directory, id);

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Create, FsException.Element.HomeDirectory);

return;

}

AllocateRecord(fileAllocation, Constants.UserListMftEntry, UserRecord.Length);

UpdateRecord(Constants.UserListMftEntry, UserRecord.Empty, new UserRecord(name, password, id, isAdministrator, homeDirectoryMftEntry));

}

public static UserRecord GetUserById(int uid)

{

Data data = ReadMftData(Constants.UserListMftEntry);

using (BinaryReader br = new BinaryReader(File.Open(Constants.SystemFile, FileMode.Open)))

{

int offset = 0;

if (data.Clusters.Count == 0)

{

br.BaseStream.Seek(Constants.UserListMftEntry \* Constants.MftRecordSize + MftHeader.Length, SeekOrigin.Begin);

while (offset < data.Size)

{

UserRecord user = new UserRecord(br.ReadBytes(UserRecord.Length));

if (!user.Equals(UserRecord.Empty) && user.Id == uid) return user;

offset += UserRecord.Length;

}

}

else

{

int blockIndex = 0;

br.BaseStream.Seek(data.Clusters[blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

while (offset < data.Size)

{

int availableBytes = FileSystem.BytesPerCluster - offset % FileSystem.BytesPerCluster;

List<byte> userBytes = new List<byte>();

if (availableBytes >= UserRecord.Length)

userBytes.AddRange(br.ReadBytes(UserRecord.Length));

else

{

userBytes.AddRange(br.ReadBytes(availableBytes));

br.BaseStream.Seek(data.Clusters[++blockIndex] \* FileSystem.BytesPerCluster,

SeekOrigin.Begin);

userBytes.AddRange(br.ReadBytes(UserRecord.Length - userBytes.Count));

}

UserRecord user = new UserRecord(userBytes.ToArray());

if (!user.Equals(UserRecord.Empty) && user.Id == uid) return user;

offset += UserRecord.Length;

}

}

}

return null;

}

static int GetAvailableId()

{

List<UserRecord> users = GetAllRecords<UserRecord>(Constants.UserListMftEntry);

List<byte> idList = users.Select(user => user.Id).ToList();

for (byte i = 0; i < Constants.MaxUserId; i++)

if (!idList.Contains(i))

return i;

return Constants.MaxUserId + 1;

}

public static void ChangeOwnerToAdministrator(byte id)

{

List<MftHeader> mftHeaders = GetAllMftHeaders();

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

{

if (mftHeaders[i].UserId == id)

{

mftHeaders[i].ChangeUser(0);

UpdateMftEntry(mftHeaders[i], i);

}

}

}

}

}

// Типы данных файловой системы

// SystemBuffer.cs

class SystemBuffer

{

public Path Path { get; set; }

public DirectoryRecord Record { get; set; }

}

// ClusterState.cs

public enum ClusterState : byte

{

Free = 0,

Damaged = 1,

Service = 0b10,

Busy = 0b11

}

// Data.cs

/// <summary>

/// Атрибут Data в Mft-записи.

/// </summary>

struct Data

{

public int Size;

public List<int> Clusters;

public Data(int size, int[] clusters)

{

Size = size;

Clusters = new List<int>(clusters);

}

}

// DirectoryRecord.cs

/// <summary>

/// Запись корневого каталога.

/// </summary>

public class DirectoryRecord : IRecord

{

public const int Length = 36;

public string FileName { get; set; } // Имя файла.

public byte Attributes { get; set; } // Атрибуты.

public string Extension { get; set; } // Расширение.

public int MftEntry { get; set; } // Номер записи в MFT.

public static readonly DirectoryRecord Empty;

static DirectoryRecord() { Empty = new DirectoryRecord(new byte[Length]); }

public DirectoryRecord() { }

public DirectoryRecord(byte[] recordBytes)

{

if (recordBytes.Length != Length) return;

FileName = Encoding.UTF8.GetString(recordBytes.GetRange(0, 26)).Trim('\0');

Extension = Encoding.UTF8.GetString(recordBytes.GetRange(26, 5)).Trim('\0');

Attributes = recordBytes[31];

MftEntry = BitConverter.ToInt32(recordBytes, 32);

}

public int GetLength() => Length;

public void SetRecord(byte[] recordBytes)

{

if (recordBytes.Length != Length) return;

FileName = Encoding.UTF8.GetString(recordBytes.GetRange(0, 26)).Trim('\0');

Extension = Encoding.UTF8.GetString(recordBytes.GetRange(27, 5)).Trim('\0');

Attributes = recordBytes[31];

MftEntry = BitConverter.ToInt32(recordBytes, 32);

}

public bool Equals(byte[] recordBytes) => GetBytes().SequenceEqual(recordBytes);

public bool EqualsByName(string name) { return GetFullName() == name; }

public DirectoryRecord(MftHeader mftEntry, int mftEntryNumber)

{

FileName = mftEntry.FileName;

Extension = mftEntry.Extension;

Attributes = mftEntry.Attributes;

MftEntry = mftEntryNumber;

}

public DirectoryRecord(DirectoryRecord record)

{

FileName = record.FileName;

Extension = record.Extension;

Attributes = record.Attributes;

MftEntry = record.MftEntry;

}

public DirectoryRecord(ExplorerFile record)

{

record.FullName.ParseFullName(record.IsDirectory, out var fileName, out var extension);

FileName = fileName;

Extension = extension;

Attributes = record.Attributes;

MftEntry = record.MftEntry;

}

public byte[] GetBytes()

{

List<byte> recordBytes = new List<byte>();

recordBytes.AddRange(FileName.GetFormatBytes(26));

recordBytes.AddRange(Extension.GetFormatBytes(5));

recordBytes.Add(Attributes);

recordBytes.AddRange(BitConverter.GetBytes(MftEntry));

return recordBytes.ToArray();

}

public string GetFullName()

{

return FileName + (Extension != "" ? "." + Extension : "");

}

public override string ToString()

{

return GetFullName() + " " + MftEntry;

}

public bool HasAttribute(MftHeader.Attribute attribute) => Attributes == (Attributes | (byte)attribute);

}

// Extensions.cs

static class Extensions

{

/// <summary>

/// Возвращает диапазон байтов массива.

/// </summary>

/// <param name="array">Массив, из которого следует получить диапазон байтов.</param>

/// <param name="startIndex">Начальный индекс диапазона.</param>

/// <param name="count">Количество байтов диапазона.</param>

/// <returns></returns>

public static byte[] GetRange(this byte[] array, int startIndex, int count)

{

byte[] arrayRange = new byte[count];

if (startIndex < 0 || array.Length < startIndex + count) return arrayRange;

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

arrayRange[i] = array[startIndex + i];

return arrayRange;

}

/// <summary>

/// Возвращает массив байтов заданной длины, представляющий коды символов переданной строки.

/// </summary>

/// <param name="source">Строка, содержащая символы для кодирования.</param>

/// <param name="resultSize">Размер результирующего массива байтов, должен быть

/// больше либо равен количеству символов в строке для кодирования.</param>

/// <returns></returns>

public static byte[] GetFormatBytes(this string source, int resultSize)

{

if (source == null) return new byte[0];

byte[] resultBytes = new byte[resultSize];

byte[] resourceBytes = Encoding.UTF8.GetBytes(source);

if (resultSize < resourceBytes.Length) Array.Copy(resourceBytes, resultBytes, resultSize);

else resourceBytes.CopyTo(resultBytes, 0);

return resultBytes;

}

/// <summary>

/// Шифрует пароль, хещируя его с помощью алгоритма SHA512.

/// </summary>

/// <param name="password"></param>

/// <returns></returns>

public static byte[] ComputeHash(this string password)

{

return SHA512.Create().ComputeHash(Encoding.UTF8.GetBytes(password));

}

public static void ParseFullName(this string fullName, bool isDirectory, out string name, out string extension)

{

if (isDirectory)

name = fullName;

else

{

if (fullName.Contains("."))

name = fullName.Substring(0, fullName.LastIndexOf('.'));

else name = fullName;

}

extension = name.Length + 1 < fullName.Length ? fullName.Substring(name.Length + 1) : "";

}

}

// FornattingOptions.cs

public struct FormattingOptions

{

public int VolumeSize;

public int BytesPerCluster;

public string FsName;

public string VolumeName;

public FormattingOptions(int volumeSize, int bytesPerCluster, string fsName, string volumeName)

{

VolumeSize = volumeSize;

BytesPerCluster = bytesPerCluster;

FsName = fsName;

VolumeName = volumeName;

}

}

// FsException.cs

public sealed class FsException : Exception

{

public enum Command

{

Create,

Rename,

Copy,

Delete,

Save,

Open,

SignIn,

SignUp

}

public enum Code

{

NotFound,

NoReadPermission,

ReadOnly,

NoWritePermission,

NoModifyPermission,

NoFullControlPermission,

NoFreeSpace,

NoSystemSpace,

MaxSize,

FileExists,

FolderExists,

NameTooLong,

ExtensionTooLong,

MaxUserCount,

UserNull,

UserExists,

IncompleteCopying,

EndlessCopy,

IncompleteDeleting

}

public enum Element

{

File,

Folder,

HomeDirectory,

User

}

public override string Message { get; }

public Code ErrorCode { get; }

public static string GetCaption(Command command, Element element)

{

string caption = "";

switch (command)

{

case Command.Create: caption = "Создание"; break;

case Command.Rename: caption = "Переименование"; break;

case Command.Copy: caption = "Копирование"; break;

case Command.Delete: caption = "Удаление"; break;

case Command.Save: caption = "Сохранение"; break;

case Command.Open: caption = "Открытие"; break;

case Command.SignIn: caption = "Авторизация"; break;

case Command.SignUp: caption = "Регистрация"; break;

}

switch (element)

{

case Element.File: caption += " файла"; break;

case Element.Folder: caption += " папки"; break;

case Element.HomeDirectory: caption += " домашней директории"; break;

case Element.User: caption += " пользователя"; break;

}

return caption;

}

public void ShowError(Command command, Element element)

{

MessageBox.Show(Message, GetCaption(command, element), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

public FsException(Code code, string element)

{

ErrorCode = code;

switch (code)

{

case Code.NotFound: Message = "Не удалось найти этот элемент: "; break;

case Code.NoReadPermission: Message = "Отсутствует право на чтение данного элемента:"; break;

case Code.NoWritePermission: Message = "Отсутствует право на запись данного элемента:"; break;

case Code.NoModifyPermission: Message = "Отсутствует право на изменение данного элемента:"; break;

case Code.NoFullControlPermission: Message = "Отсутствует полный доступ к данному элементу:"; break;

case Code.ReadOnly: Message = "Данный файл доступен только для чтения."; break;

case Code.NoFreeSpace: Message = $"Элемент: {element} {Environment.NewLine}" +

"Недостаточно места на диске для завершения текущей операции."; return;

case Code.MaxSize: Message = $"Элемент: {element} {Environment.NewLine}" +

"Превышен максимально допустимый размер "; return;

case Code.FileExists: Message = "В данном расположении уже существует файл с таким именем:"; break;

case Code.FolderExists: Message = "В данном расположении уже существует папка с таким именем:"; break;

case Code.NameTooLong: Message = $"Имя файла слишком длинное: {element} {Environment.NewLine}" +

"Оно не должно превышать 26 символов."; return;

case Code.ExtensionTooLong: Message = $"Расширение слишком длинное: {element} {Environment.NewLine}" +

"Оно не должно превышать 5 символов."; return;

case Code.MaxUserCount: Message = "Достигнуто максимальное число пользователей в системе."; return;

case Code.UserNull: Message = "Пользователь с данным именем и паролем не зарегистрирован в системе."; return;

case Code.UserExists: Message = "Пользователь с таким именем уже существует в системе:"; return;

case Code.IncompleteCopying:

Message = "Не все файлы из директории были скопированы:"; break;

case Code.IncompleteDeleting: Message = "Данная папка полностью не удалена:"; break;

case Code.EndlessCopy:

Message = "Конечная папка, в которую следует поместить файлы, является" +

" дочерней для папки, в которой они находятся:"; break;

case Code.NoSystemSpace:

Message = "Недостаточно системного пространства для завершения текущей операции."; return;

}

Message += Environment.NewLine + element + ".";

}

}

public interface IRecord

{

byte[] GetBytes();

void SetRecord(byte[] recordBytes);

int GetLength();

bool Equals(byte[] recordBytes);

bool EqualsByName(string name);

}

struct MemoryAllocation

{

[Flags]

public enum Attribute : byte

{

NeedToIncreaseMftSize = 0b1,

NeedToIncreaseDirectorySize = 0b10,

NeedNewCluster = 0b100

}

public int MftEntry { get; set; }

public Attribute Attributes { get; }

public MemoryAllocation(int mftEntry, Attribute attributes)

{

MftEntry = mftEntry;

Attributes = attributes;

}

}

// MftHeader.cs

/// <summary>

/// Заголовок MFT-записи.

/// </summary>

public class MftHeader

{

public const int Length = 50;

// Перечисление признаков записи.

public enum Signature : byte

{

NotUsed = 0,

InUse = 1

}

[Flags]

public enum Attribute : byte

{

None = 0,

ReadOnly = 0b1,

Hidden = 0b10,

Directory = 0b100

}

public Signature Sign { get; set; } // Признак записи.

public string FileName { get; set; } // Имя файла.

public string Extension { get; set; } // Расширение файла.

public byte Attributes { get; set; } // Атрибуты.

public int Size { get; set; } // Размер.

public MyDateTime CreationDate { get; set; } // Дата создания файла.

public MyDateTime ModificationDate { get; set; } // Дата последней модификации файла.

public byte UserId { get; private set; } // Уникальный идентификатор владельца файла.

public Permission Permissions { get; set; } // Права на файл.

public MftHeader(string fileName, string extension, Attribute attributes = Attribute.None, int size = 0, int userId = -1)

{

Sign = Signature.InUse;

FileName = fileName;

Extension = extension;

Attributes = (byte)attributes;

Size = size;

CreationDate = ModificationDate = new MyDateTime(DateTime.Now);

if (userId < 0) UserId = MyOS.FileSystem.FileSystem.CurrentUser.Id;

else UserId = (byte)userId;

Permissions = new Permission();

}

public MftHeader(byte[] recordHeader)

{

Sign = (Signature) recordHeader[0];

FileName = Encoding.UTF8.GetString(recordHeader.GetRange(1, 26)).Trim('\0');

Extension = Encoding.UTF8.GetString(recordHeader.GetRange(27, 5)).Trim('\0');

Attributes = recordHeader[32];

Size = BitConverter.ToInt32(recordHeader, 33);

CreationDate = new MyDateTime(recordHeader.GetRange(37, 5));

ModificationDate = new MyDateTime(recordHeader.GetRange(42, 5));

UserId = recordHeader[47];

Permissions = new Permission(recordHeader.GetRange(48, 2));

}

public byte[] GetBytes()

{

List<byte> headerBytes = new List<byte> {(byte) Sign};

headerBytes.AddRange(FileName.GetFormatBytes(26));

headerBytes.AddRange(Extension.GetFormatBytes(5));

headerBytes.Add(Attributes);

headerBytes.AddRange(BitConverter.GetBytes(Size));

headerBytes.AddRange(CreationDate.DateTimeBytes);

headerBytes.AddRange(ModificationDate.DateTimeBytes);

headerBytes.Add(UserId);

headerBytes.AddRange(Permissions.GetBytes());

return headerBytes.ToArray();

}

public bool HasPermissions(UserRecord user, Permission.Rights rights)

{

if (user.Id == UserId)

return Permissions.CheckRights(Permission.UserSign.Owner, rights);

return Permissions.CheckRights(

user.IsAdministrator ? Permission.UserSign.Administrator : Permission.UserSign.Other, rights);

}

public void ChangeUser(int newUserId)

{

UserId = (byte) newUserId;

}

public string GetFullName()

{

return FileName + (Extension != "" ? "." + Extension : "");

}

public bool HasAttribute(Attribute attribute) => Attributes == (Attributes | (byte)attribute);

public void SetAttribute(Attribute attribute, bool state)

{

if (state) Attributes |= (byte)attribute;

else Attributes = (byte)(Attributes & (byte)~attribute);

}

private string AttributesToString()

{

string attributes = "";

attributes += HasAttribute(Attribute.Directory) ? "D" : "-";

attributes += HasAttribute(Attribute.Hidden) ? "H" : "-";

return attributes + (HasAttribute(Attribute.ReadOnly) ? "R" : "-");

}

public override string ToString()

{

return (Sign == Signature.NotUsed

? "NotUsed"

: "InUse") + $" {AttributesToString()} {FileName} " +

$"{Extension} {Size} Б " +

$"{CreationDate.GetFullDateTime()} {ModificationDate.GetFullDateTime()} " +

$"{UserId} {Permissions}";

}

}

// MyDateTime.cs

/// <summary>

/// Представляет 5-байтную пользовательскую структуру даты и времени.

/// </summary>

public struct MyDateTime

{

public byte[] DateTimeBytes;

public static readonly MyDateTime MinValue = new MyDateTime(new byte[5]);

public MyDateTime(byte[] dateTimeBytes)

{

DateTimeBytes = new byte[5];

if (dateTimeBytes.Length == 5)

DateTimeBytes = dateTimeBytes;

}

public MyDateTime(DateTime dateTime)

{

DateTimeBytes = new byte[5];

uint utc = (uint)(dateTime.Hour - DateTime.UtcNow.Hour);

// Записываем разницу часов текущего времени от всемирного

// координированного времени в последние четыре бита.

ulong ulongDateTime = utc;

// Сдвигаем число на 5 битов влево, освобождая конечные биты под компонент дня даты.

ulongDateTime = ulongDateTime << 5;

// Записываем компонент дня в младшие 5 разрядов.

ulongDateTime = ulongDateTime | (uint)dateTime.Day;

// Сдвигаем число на 4 бита влево, освобождая конечные биты под компонент месяца даты,

// и записываем компонент месяца в младшие 4 разряда.

ulongDateTime = ulongDateTime << 4 | (uint)dateTime.Month;

ulongDateTime = ulongDateTime << 10 | (uint) (dateTime.Year - 2000);

ulongDateTime = ulongDateTime << 5 | (uint)dateTime.Hour;

ulongDateTime = ulongDateTime << 6 | (uint)dateTime.Minute;

ulongDateTime = ulongDateTime << 6 | (uint)dateTime.Second;

byte[] longDateTimeBytes = BitConverter.GetBytes(ulongDateTime);

for (int i = 0; i < 5; i++)

DateTimeBytes[i] = longDateTimeBytes[i];

}

private ulong GetUlongTimeDate()

{

byte[] ulongDateTimeBytes = new byte[8];

DateTimeBytes.CopyTo(ulongDateTimeBytes, 0);

return BitConverter.ToUInt64(ulongDateTimeBytes, 0);

}

public int GetUtcHours() => (int)(GetUlongTimeDate() >> 36 & 0xF);

public string GetStringUtcHours() => "UTC +" + GetUtcHours() + " часа";

public int GetDay() => (int)(GetUlongTimeDate() >> 31 & 0x1F);

public int GetMonth() => (int)(GetUlongTimeDate() >> 27 & 0xF);

public int GetYear() => (int)(GetUlongTimeDate() >> 17 & 0x3FF) + 2000;

public int GetHour() => (int)(GetUlongTimeDate() >> 12 & 0x1F);

public int GetMinute() => (int)(GetUlongTimeDate() >> 6 & 0x3F);

public int GetSecond() => (int)(GetUlongTimeDate() & 0x3F);

public override string ToString()

{

return GetDate() + $" {GetHour():d2}:{GetMinute():d2}";

}

public string GetFullDateTime()

{

return $"{GetDay():d2}.{GetMonth():d2}.{GetYear()} {GetHour():d2}:{GetMinute():d2}:{GetSecond():d2}";

}

public string GetDate() => $"{GetDay():d2}.{GetMonth():d2}.{GetYear()}";

public string GetTime() => $"{GetHour():d2}:{GetMinute():d2}:{GetSecond():d2}";

}

}

// Path.cs

public class Path :INotifyPropertyChanged

{

private const char Separator = '/';

public List<string> DirectoriesList;

private string \_fullPath;

public string CurrentPath

{

get => \_fullPath;

set {

if (value == \_fullPath)

return;

\_fullPath = value;

OnPropertyChanged(CurrentPath);

}

}

public static readonly string RootPath;

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName]string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

static Path()

{

RootPath = FileSystem.VolumeName + ":" + FileSystem.Root;

}

public Path()

{

DirectoriesList = new List<string>();

CurrentPath = RootPath;

}

public Path(List<string> directories)

{

DirectoriesList = new List<string>(directories);

CurrentPath = RootPath + string.Join(Separator.ToString(), DirectoriesList);

}

public Path(Path path)

{

DirectoriesList = new List<string>(path.DirectoriesList);

CurrentPath = path.CurrentPath;

}

public void Add(string directory)

{

DirectoriesList.Add(directory);

CurrentPath = RootPath + string.Join(Separator.ToString(), DirectoriesList);

}

public void GetPreviosFolder()

{

if (DirectoriesList.Count > 0)

{

DirectoriesList.RemoveAt(DirectoriesList.Count - 1);

CurrentPath = FileSystem.VolumeName + ":" + FileSystem.Root + string.Join(Separator.ToString(), DirectoriesList);

}

}

public string GetAbsolutePath(string fileName)

{

if (DirectoriesList.Count > 0)

return CurrentPath + Separator + fileName;

return CurrentPath + fileName;

}

}

// Permission.cs

public class Permission

{

[Flags]

public enum Rights : byte

{

FullControl = 0b1111, // Все права (Full Control = Modify + назначение прав другим пользователям).

Modify = 0b0111, // Изменение (Modify = Read + Write).

Read = 0b0010, // Чтение (Read).

Write = 0b0001 // Запись (Write).

}

/// <summary>

/// Признак пользователя, показывающий смещение битов, отвечающих за его права.

/// </summary>

public enum UserSign

{

Administrator = 8, // Номер бита, начиная с которого записаны 4 бита прав администратора.

Owner = 4, // Номер бита, начиная с которого записаны права админастратора.

Other = 0 // В третьих - права всех остальных пользователей.

}

private ushort \_permissions;

public Permission(byte[] newPermissions)

{

if (newPermissions.Length != 2) throw new ArgumentException(nameof(newPermissions));

// Поочередно записываем байты, содержащие права пользователей.

ushort ushortPermissions = newPermissions[0];

ushortPermissions = (ushort) (ushortPermissions << 8); // Сдвигаем влево на ещё один байт.

ushortPermissions = (ushort) (ushortPermissions | newPermissions[1]);

\_permissions = ushortPermissions;

}

/// <summary>

/// По умолчанию для администраторов и владельца определяются полные права.

/// </summary>

public Permission()

{

// Записываем права администратора и сдвигаем на 4 бита влево,

// освобождая место под права владельца.

\_permissions = (ushort)Rights.FullControl << 4;

\_permissions |= (byte)Rights.FullControl; // Записываем права владельца в младшие четыре бита.

\_permissions <<= 4; // Снова сдвигаем на 4 бита для поля прав остальных пользователей.

}

public bool CheckRights(UserSign userSign, Rights right)

{

return \_permissions == (\_permissions | ((ushort)right << (ushort)userSign));

}

public byte[] GetBytes()

{

byte[] permissionsBytes = new byte[2];

permissionsBytes[0] = (byte) (\_permissions >> 8);

permissionsBytes[1] = (byte) \_permissions;

return permissionsBytes;

}

public void SetPermission(UserSign userSign, Rights right, bool state)

{

byte allUserRights = (byte)((\_permissions >> (ushort)userSign) & 0b0000\_1111);

if (state) allUserRights |= (byte)right;

else

{

switch (right)

{

case Rights.FullControl: allUserRights &= 0b0111; break;

case Rights.Modify: allUserRights &= 0b0011; break;

case Rights.Read: allUserRights &= 0b0001; break;

case Rights.Write: allUserRights &= 0b0010; break;

}

}

switch (userSign)

{

case UserSign.Administrator: \_permissions &= 0b1111\_0000\_1111\_1111; break;

case UserSign.Owner: \_permissions &= 0b1111\_1111\_0000\_1111; break;

case UserSign.Other: \_permissions &= 0b1111\_1111\_1111\_0000; break;

}

// Обновляем разрешения, добавляя к текущим новые права доступа.

\_permissions |= (ushort)(allUserRights << (ushort)userSign);

}

public override string ToString()

{

string permission = "";

for (int i = 0; i < 12; i++)

{

permission = permission.Insert(0, ((\_permissions >> i) | 1) == \_permissions >> i++ ? "R" : "-");

permission = permission.Insert(0, ((\_permissions >> i) | 1) == \_permissions >> i++ ? "W" : "-");

permission = permission.Insert(0, ((\_permissions >> i) | 1) == \_permissions >> i++ ? "M" : "-");

permission = permission.Insert(0, ((\_permissions >> i) | 1) == \_permissions >> i ? " F" : " -");

}

return permission;

}

}

// UserRecord.cs

/// <summary>

/// Запись в списке пользователей, представляющая полную информацию об одном пользователе системы.

/// </summary>

public class UserRecord : IRecord

{

public const byte Length = 96; // Размер заявки пользователя в списке пользователей.

public string Name { get; set; } // Имя пользователя.

public bool IsAdministrator { get; set; } // Признак администратора.

public byte Id { get; set; } // Уникальный идентификатор.

public byte[] PasswordHash { get; set; } // Пароль.

public int HomeDirectoryMftEntry { get; set; } // Номер Mft-записи домашней директории.

public static readonly UserRecord Empty;

static UserRecord()

{

Empty = new UserRecord(new byte[Length]);

}

public UserRecord(string name, string password, int id, bool isAdministrator, int homeDirectoryMftEntry)

{

Id = (byte)id;

IsAdministrator = isAdministrator;

Name = name;

PasswordHash = password.ComputeHash();

HomeDirectoryMftEntry = homeDirectoryMftEntry;

}

public UserRecord(UserRecord user)

{

Id = user.Id;

IsAdministrator = user.IsAdministrator;

Name = user.Name;

PasswordHash = user.PasswordHash;

HomeDirectoryMftEntry = user.HomeDirectoryMftEntry;

}

public UserRecord(byte[] userBytes)

{

if(userBytes.Length != Length) return;

Name = Encoding.UTF8.GetString(userBytes.GetRange(0, 26)).Trim('\0');

Id = userBytes[26];

IsAdministrator = userBytes[27] == 1;

PasswordHash = userBytes.GetRange(28, 64);

HomeDirectoryMftEntry = BitConverter.ToInt32(userBytes, 92);

}

public void SetRecord(byte[] recordBytes)

{

if (recordBytes.Length != Length) return;

Name = Encoding.UTF8.GetString(recordBytes.GetRange(0, 26)).Trim('\0');

Id = recordBytes[26];

IsAdministrator = recordBytes[27] == 1;

PasswordHash = recordBytes.GetRange(28, 64);

HomeDirectoryMftEntry = BitConverter.ToInt32(recordBytes, 92);

}

public int GetLength() => Length;

public bool Equals(byte[] recordBytes) => GetBytes().SequenceEqual(recordBytes);

public bool EqualsByName(string name) { return Name == name; }

public byte[] GetBytes()

{

List<byte> userBytes = new List<byte>();

userBytes.AddRange(Name.GetFormatBytes(26));

userBytes.Add(Id);

userBytes.Add(IsAdministrator ? (byte)1 : (byte)0);

userBytes.AddRange(PasswordHash);

userBytes.AddRange(BitConverter.GetBytes(HomeDirectoryMftEntry));

return userBytes.ToArray();

}

}

// Классы моделей для интерфейса

// BitmapRow.cs

public class BitmapRow

{

public int RowNumber { get; private set; }

public string Cell0 { get; private set; }

public string Cell1 { get; private set; }

public string Cell2 { get; private set; }

public string Cell3 { get; private set; }

public string Cell4 { get; private set; }

public string Cell5 { get; private set; }

public string Cell6 { get; private set; }

public string Cell7 { get; private set; }

public string Cell8 { get; private set; }

public string Cell9 { get; private set; }

public BitmapRow() { }

public void SetRowNumber(int i) { RowNumber = i; }

public void Add(int i, string cell)

{

switch (i)

{

case 0: Cell0 = cell; return;

case 1: Cell1 = cell; return;

case 2: Cell2 = cell; return;

case 3: Cell3 = cell; return;

case 4: Cell4 = cell; return;

case 5: Cell5 = cell; return;

case 6: Cell6 = cell; return;

case 7: Cell7 = cell; return;

case 8: Cell8 = cell; return;

case 9: Cell9 = cell; return;

}

}

}

// Commands.cs

static Commands()

{

// Создаём и инициализируем команды.

InputGestureCollection inputs =

new InputGestureCollection {new KeyGesture(Key.C, ModifierKeys.Control | ModifierKeys.Shift, "Ctrl + Shift + C")};

Copy = new RoutedUICommand("Copy", "Copy", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.V, ModifierKeys.Control | ModifierKeys.Shift, "Ctrl + Shift + V") };

Paste = new RoutedUICommand("Paste", "Paste", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.F2, 0, "F2") };

Rename = new RoutedUICommand("Rename", "Rename", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.D, ModifierKeys.Control | ModifierKeys.Shift, "Ctrl + Shift + D") };

Delete = new RoutedUICommand("Delete", "Delete", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.S, ModifierKeys.Control | ModifierKeys.Shift, "Ctrl + Shift + S") };

Save = new RoutedUICommand("Save", "Save", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.Apps, 0, "Apps") };

ShowProperties = new RoutedUICommand("ShowProperties", "ShowProperties", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.F, ModifierKeys.Control, "Ctrl + F") };

CreateFile = new RoutedUICommand("CreateFile", "CreateFile", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.D, ModifierKeys.Control, "Ctrl + D") };

CreateFolder = new RoutedUICommand("CreateFolder", "CreateFolder", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.F, ModifierKeys.Control | ModifierKeys.Alt, "Ctrl + Alt + F") };

Create100Files = new RoutedUICommand("Create100Files", "Create100Files", typeof(Commands), inputs);

inputs =

new InputGestureCollection { new KeyGesture(Key.D, ModifierKeys.Control | ModifierKeys.Alt, "Ctrl + Alt + D") };

Create100Folders = new RoutedUICommand("Create100Folders", "Create100Folders", typeof(Commands), inputs);

}

public static RoutedUICommand Copy { get; }

public static RoutedUICommand Paste { get; }

public static RoutedUICommand Rename { get; }

public static RoutedUICommand Delete { get; }

public static RoutedUICommand Save { get; }

public static RoutedUICommand CreateFile { get; }

public static RoutedUICommand CreateFolder { get; }

public static RoutedUICommand Create100Files { get; }

public static RoutedUICommand Create100Folders { get; }

public static RoutedUICommand ShowProperties { get; }

}

// Explorer.cs

public class ExplorerFile

{

private byte \_attributes;

public byte Attributes

{

get => \_attributes;

set

{

\_attributes = value;

IsHidden = value == (value | (byte)MftHeader.Attribute.Hidden);

IsDirectory = value == (value | (byte)MftHeader.Attribute.Directory);

string uri = IsDirectory

? (IsHidden ? "pack://application:,,,/Resources/hidden\_folder.png" : "pack://application:,,,/Resources/folder.png")

: (IsHidden

? "pack://application:,,,/Resources/hidden\_file.png"

: "pack://application:,,,/Resources/file.png");

ImageSource = new BitmapImage(new Uri(uri, UriKind.Absolute));

}

} // Атрибуты.

public string FullName { get; set; } // Имя файла с расширением.

public MyDateTime CreationDate { get; set; } // Дата создания.

public MyDateTime ModificationDate { get; set; } // Дата последнего изменения.

public string Size { get; set; } // Размер файла.

public int MftEntry { get; set; } // Номер записи MFT.

public bool IsHidden { get; private set; }

public bool IsDirectory { get; private set; }

public ImageSource ImageSource { get; private set; }

public ExplorerFile(MftHeader mftHeader, int mftEntry)

{

Attributes = mftHeader.Attributes;

FullName = mftHeader.GetFullName();

CreationDate = mftHeader.CreationDate;

ModificationDate = mftHeader.ModificationDate;

Size = IsDirectory ? "" : mftHeader.Size + " Б";

MftEntry = mftEntry;

}

}

class UserAccount

{

public string Name { get; set; } // Имя пользователя.

public byte Id { get; set; } // Уникальный идентификатор.

public string Type { get; } // Тип пользователя.

public string HomeDirectory { get; set; } // Имя домашней директории.

public UserAccount(UserRecord user)

{

Name = user.Name;

Id = user.Id;

Type = user.IsAdministrator ? "Администратор" : "Обычный";

HomeDirectory = SystemCalls.GetMftHeader(user.HomeDirectoryMftEntry).FileName;

}

}

// Классы-окна системы

// App.xaml.cs

public partial class App

{

protected override void OnStartup(StartupEventArgs e)

{

base.OnStartup(e);

MainWindow = new MainWindow();

if (!File.Exists(FileSystem.FileSystem.Constants.SystemFile))

{

// Если файл тома не найден, то форматируем его с параметрами по умолчанию.

FormattingWindow formattingWindow = new FormattingWindow(new FormattingOptions(419430400, 4096, "VMFS v2.0", "C"));

formattingWindow.ShowDialog();

}

LoginWindow loginWindow = new LoginWindow(false);

if (loginWindow.ShowDialog() == true)

MainWindow.Show();

else MainWindow.Close();

}

}

// Control.xaml.cs

public partial class ControlWindow

{

private readonly bool \_isSuperRoot;

public ControlWindow(bool isSuperRoot)

{

InitializeComponent();

\_isSuperRoot = isSuperRoot;

if (!\_isSuperRoot)

{

AccountsItem.Visibility = Visibility.Collapsed;

MftItem.Visibility = Visibility.Collapsed;

BitmapItem.Visibility = Visibility.Collapsed;

FormattingItem.Visibility = Visibility.Collapsed;

ResizeMode = ResizeMode.NoResize;

}

else MinWidth = 800;

}

private void ControlWindow\_OnActivated(object sender, EventArgs e)

{

UpdateFileSystemInfo();

if (!\_isSuperRoot) return;

UpdateUserTable();

UpdateMft();

UpdateBitmapData();

}

private void UpdateFileSystemInfo()

{

FsName.Text = FileSystem.FileSystem.FileSystemVersion;

VolumeName.Text = FileSystem.FileSystem.VolumeName.ToString();

VolumeState.Text = FileSystem.FileSystem.State == 0 ? "Исправен" : "Повреждён";

VolumeSize.Text = FileSystem.FileSystem.VolumeSize.ToString("N0") + " Б";

int busySpace = FileSystem.FileSystem.GetBusySpace();

BusySpace.Text = busySpace.ToString("N0") + " Б";

FreeSpace.Text = (FileSystem.FileSystem.VolumeSize - busySpace).ToString("N0") + " Б";

}

private void UpdateMft()

{

List<MftHeader> allMftHeaders = SystemCalls.GetAllMftHeaders();

MftHeaders.ItemsSource = allMftHeaders;

MftZoneSize.Text = FileSystem.FileSystem.MftAreaSize.ToString("N0") + " Б";

MftSize.Text =

(allMftHeaders.Count(mftHeader => mftHeader.Sign == MftHeader.Signature.InUse) \*

FileSystem.FileSystem.Constants.MftRecordSize).ToString("N0") + " Б";

MftHeaders.SelectedIndex = 0;

}

private void UpdateBitmapData()

{

ClustersCount.Text = FileSystem.FileSystem.GetClusterCount().ToString("N0");

ServiceClusters.Text = FileSystem.FileSystem.ServiceClusters.ToString("N0");

BusyClusters.Text = FileSystem.FileSystem.GetBusyClusterCount().ToString("N0");

ClusterSize.Text = FileSystem.FileSystem.BytesPerCluster.ToString("N0") + " Б";

GetBitmapData();

}

public void GetBitmapData()

{

byte[] bytes = SystemCalls.ReadFileData(FileSystem.FileSystem.Constants.BitmapMftEntry);

BitmapSize.Text = bytes.Length.ToString("N0") + " Б";

List<BitmapRow> grid = new List<BitmapRow>();

for (int i = 0; i < bytes.Length;)

{

BitmapRow row = new BitmapRow();

row.SetRowNumber(i \* 4);

for (int j = 0; j < 10 && i < bytes.Length; j++)

row.Add(j, Convert.ToString(bytes[i++], 2).PadLeft(8, '0'));

grid.Add(row);

}

BitmapDataGrid.ItemsSource = grid;

}

private void Format\_Click(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Форматирование приведёт к безвозвратной потере всех данных," +

" в том числе всех учётных записей пользователей." + Environment.NewLine + "Продолжить?",

"Форматирование", MessageBoxButton.YesNo, MessageBoxImage.Information) == MessageBoxResult.Yes)

{

// Размер тома - // 1/2/4/8 \* 100 (МБ) \* 1024 (КБ) \* 1024 Б.

int volumeSize = (1 << VolumeSizeComboBox.SelectedIndex) \* 100 \* 1024 \* 1024;

int bytesPerCluster = (1 << BytesPerClusterComboBox.SelectedIndex) \* 1024;

string fsName = FsName.Text;

string volumeName = ((ComboBoxItem) VolumeNameComboBox.SelectedItem).Content.ToString();

FormattingWindow formattingWindow = new FormattingWindow(new FormattingOptions(volumeSize, bytesPerCluster, fsName, volumeName));

formattingWindow.ShowDialog();

System.Diagnostics.Process.Start(Application.ResourceAssembly.Location);

Application.Current.Shutdown();

}

}

public void UpdateUserTable()

{

List<UserAccount> userAccounts = new List<UserAccount>();

var allUsers = SystemCalls.GetAllRecords<UserRecord>(FileSystem.FileSystem.Constants.UserListMftEntry);

foreach (var user in allUsers)

userAccounts.Add(new UserAccount(user));

UserTable.ItemsSource = userAccounts;

}

private void AddUser(object sender, RoutedEventArgs e)

{

LoginWindow loginWindow = new LoginWindow(true);

if (loginWindow.ShowDialog() == true)

UpdateUserTable();

}

private void DeleteUser(object sender, RoutedEventArgs e)

{

if (UserTable.CurrentCell.Item is DataGridCellInfo ||

!(UserTable.SelectedItem is UserAccount user)) return;

if (user.Id == 0) MessageBox.Show("Удалить администратора нельзя!", "Удаление пользователя",

MessageBoxButton.OK, MessageBoxImage.Error);

else if (user.Id == FileSystem.FileSystem.CurrentUser.Id)

{

if (MessageBox.Show(

"Вы пытаетесь удалить свою текущую учётную запись. " +

"При завершении данной операции будет осуществлён выход из системы. Продолжить?",

"Удаление учётной записи пользователя", MessageBoxButton.YesNo, MessageBoxImage.Question) ==

MessageBoxResult.Yes)

{

DialogResult = true;

}

}

else

{

SystemCalls.UpdateRecord(FileSystem.FileSystem.Constants.UserListMftEntry, SystemCalls.GetUserById(user.Id),

UserRecord.Empty);

SystemCalls.ChangeOwnerToAdministrator(user.Id);

UpdateUserTable();

}

}

private void MftHeaders\_OnSelectionChanged(object sender, SelectionChangedEventArgs e)

{

MftEnty.Text = MftHeaders.SelectedIndex.ToString();

}

}

// FormattingWindow.xaml.cs

public partial class FormattingWindow

{

private readonly BackgroundWorker \_backgroundWorker;

private readonly FormattingOptions \_formattingOptions;

public FormattingWindow( FormattingOptions formattingOptions)

{

InitializeComponent();

\_formattingOptions = formattingOptions;

\_backgroundWorker = new BackgroundWorker();

\_backgroundWorker.DoWork += BackgroundWorker\_DoWork;

\_backgroundWorker.RunWorkerCompleted += BackgroundWorker\_RunWorkerCompleted;

}

private void Move(object sender, MouseButtonEventArgs e)

{

DragMove();

}

private void BackgroundWorker\_DoWork(object sender, DoWorkEventArgs e)

{

SystemCalls.Formatting((FormattingOptions)e.Argument);

}

private void BackgroundWorker\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

Close();

}

private void FormattingWindow\_OnActivated(object sender, EventArgs e)

{

string[] animation = { ". ", ". . ", ". . ." };

int second = 0;

var timer = new System.Windows.Threading.DispatcherTimer

{

Interval = new TimeSpan(0, 0, 1),

IsEnabled = true

};

timer.Tick += (o, t) =>

{

Animation.Text = animation[second];

if (second == 2) second = 0;

else second++;

};

timer.Start();

if(!\_backgroundWorker.IsBusy)

\_backgroundWorker.RunWorkerAsync(\_formattingOptions);

}

}

// LoginWindow.xaml.cs

public partial class LoginWindow

{

public LoginWindow(bool isRegistration)

{

InitializeComponent();

\_isRegistrationWindow = isRegistration;

SetWindowType();

}

private readonly bool \_isRegistrationWindow;

private void SetWindowType()

{

if (\_isRegistrationWindow)

{

SignIn.Content = "Зарегистрировать";

UserTypeRow.Visibility = Visibility.Visible;

}

else

{

SignIn.Content = "Войти";

UserTypeRow.Visibility = Visibility.Collapsed;

}

}

private void SignIn\_Click(object sender, RoutedEventArgs e)

{

if (!\_isRegistrationWindow)

{

try

{

SystemCalls.SignIn(UserName.Text, Password.Password);

DialogResult = true;

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.SignIn, FsException.Element.User);

}

}

else

{

try

{

SystemCalls.SignUp(UserName.Text, Password.Password,

UserTypeRow.IsChecked != null && UserTypeRow.IsChecked.Value);

DialogResult = true;

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.SignUp, FsException.Element.User);

}

}

}

private void Cancel\_Click(object sender, RoutedEventArgs e)

{

DialogResult = false;

}

private void Move(object sender, MouseButtonEventArgs e)

{

DragMove();

}

private void TextBox\_OnTextChanged(object sender, TextChangedEventArgs e)

{

SignIn.IsEnabled = UserName?.Text.TrimEnd() != "" && Password?.Password != "";

}

private void Password\_OnPasswordChanged(object sender, RoutedEventArgs e)

{

SignIn.IsEnabled = UserName?.Text.TrimEnd() != "" && Password?.Password != "";

}

}

// MainWindow.xaml.cs

public partial class MainWindow

{

public MainWindow()

{

InitializeComponent();

}

public void UpdateFileTable()

{

List<ExplorerFile> fileList = new List<ExplorerFile>();

var allFiles = SystemCalls.GetFileList(FileSystem.FileSystem.CurrentPath);

if (ShowHidden.IsChecked != null && ShowHidden.IsChecked.Value)

fileList = allFiles;

else

{

foreach (var file in allFiles)

if (!file.IsHidden)

fileList.Add(file);

}

FileTable.ItemsSource = fileList;

FileCount.Text = FileTable.Items.Count.ToString();

}

private void ShowHidden\_Checked(object sender, RoutedEventArgs e)

{

FileTable.ItemsSource = SystemCalls.GetFileList((Path) CurrentDirectory.DataContext);

FileCount.Text = FileTable.Items.Count.ToString();

}

private void ShowHidden\_Unchecked(object sender, RoutedEventArgs e)

{

List<ExplorerFile> fileList = new List<ExplorerFile>();

var allFiles = SystemCalls.GetFileList((Path) CurrentDirectory.DataContext);

foreach (var file in allFiles)

if (!file.IsHidden)

fileList.Add(file);

FileCount.Text = fileList.Count.ToString();

FileTable.ItemsSource = fileList;

}

private void Create100Files(object sender, RoutedEventArgs e)

{

for (int i = 0; i < 100; i++)

{

try

{

SystemCalls.Create((Path)CurrentDirectory.DataContext, "file" + (i + 1), "txt");

}

catch (FsException fs)

{

fs.ShowError(FsException.Command.Create, FsException.Element.File);

}

}

UpdateFileTable();

}

private void Create100Folders(object sender, RoutedEventArgs e)

{

for (int i = 0; i < 100; i++)

{

try

{

SystemCalls.Create((Path) CurrentDirectory.DataContext, "folder" + (i + 1), "",

MftHeader.Attribute.Directory);

}

catch (FsException fs)

{

fs.ShowError(FsException.Command.Create, FsException.Element.Folder);

}

}

UpdateFileTable();

}

private void CreateFile\_Click(object sender, RoutedEventArgs e)

{

NameWindow name =

new NameWindow(FsException.Element.File, (Path) CurrentDirectory.DataContext);

name.ShowDialog();

}

private void CreateFolder\_Click(object sender, RoutedEventArgs e)

{

NameWindow name =

new NameWindow(FsException.Element.Folder, (Path) CurrentDirectory.DataContext);

name.ShowDialog();

}

private void FileTable\_OnMouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

if (file.Attributes == (file.Attributes | (byte) MftHeader.Attribute.Directory))

{

if (!SystemCalls.GetMftHeader(file.MftEntry).HasPermissions(FileSystem.FileSystem.CurrentUser, Permission.Rights.Read))

{

new FsException(FsException.Code.NoReadPermission, file.FullName).ShowError(

FsException.Command.Open, FsException.Element.Folder);

return;

}

((Path) CurrentDirectory.DataContext).Add(file.FullName);

CurrentDirectory.Text = ((Path) CurrentDirectory.DataContext).CurrentPath;

UpdateFileTable();

}

else EditFile();

}

private void PreviousDirectory\_Click(object sender, RoutedEventArgs e)

{

Path path = (Path) CurrentDirectory.DataContext;

path.GetPreviosFolder();

CurrentDirectory.DataContext = path;

CurrentDirectory.Text = ((Path) CurrentDirectory.DataContext).CurrentPath;

UpdateFileTable();

}

private void Copy\_Click(object sender, RoutedEventArgs e)

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

try

{

SystemCalls.Copy((Path) CurrentDirectory.DataContext, new DirectoryRecord(file));

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Copy,

file.IsDirectory ? FsException.Element.Folder : FsException.Element.File);

}

}

private void Paste\_Click(object sender, RoutedEventArgs e)

{

if (FileSystem.FileSystem.Buffer == null)

{

MessageBox.Show("Буфер обмена пуст!");

return;

}

try

{

SystemCalls.Paste((Path) CurrentDirectory.DataContext);

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Copy,

FileSystem.FileSystem.Buffer.Record.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Element.Folder

: FsException.Element.File);

}

UpdateFileTable();

}

private void EditFile()

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

if (!SystemCalls.GetMftHeader(file.MftEntry).HasPermissions(FileSystem.FileSystem.CurrentUser, Permission.Rights.Read))

{

new FsException(FsException.Code.NoReadPermission, file.FullName).ShowError(FsException.Command.Open,

FsException.Element.File);

return;

}

string fullName = ((Path)CurrentDirectory.DataContext).GetAbsolutePath(file.FullName);

foreach (Window window in Application.Current.Windows)

{

if (window is TextEditorWindow && window.Title == fullName)

{

window.Activate();

return;

}

}

TextEditorWindow editWindow = new TextEditorWindow(file, fullName)

{Owner = this};

editWindow.Show();

}

private void Rename\_Click(object sender, RoutedEventArgs e)

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

NameWindow name =

new NameWindow(file, (Path) CurrentDirectory.DataContext);

name.ShowDialog();

}

private void Delete\_Click(object sender, RoutedEventArgs e)

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

try

{

SystemCalls.Delete((Path) CurrentDirectory.DataContext, new DirectoryRecord(file));

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Delete,

file.IsDirectory ? FsException.Element.Folder : FsException.Element.File);

}

UpdateFileTable();

}

private void MainWindow\_Activated(object sender, EventArgs e)

{

if ((Path)CurrentDirectory.DataContext != null)

UpdateFileTable();

}

private void MainWindow\_OnLoaded(object sender, RoutedEventArgs e)

{

FileSystem.FileSystem.CurrentPath = new Path();

User.Text = FileSystem.FileSystem.CurrentUser.Name;

if (FileSystem.FileSystem.CurrentUser.HomeDirectoryMftEntry != FileSystem.FileSystem.Constants.RootMftEntry)

FileSystem.FileSystem.CurrentPath.Add(SystemCalls.GetMftHeader(FileSystem.FileSystem.CurrentUser.HomeDirectoryMftEntry).FileName);

CurrentDirectory.DataContext = FileSystem.FileSystem.CurrentPath;

CurrentDirectory.Text = FileSystem.FileSystem.CurrentPath.CurrentPath;

Time.Text = new MyDateTime(DateTime.Now).ToString();

UpdateFileTable();

var timer = new System.Windows.Threading.DispatcherTimer

{

Interval = new TimeSpan(0, 1, 0),

IsEnabled = true

};

timer.Tick += (o, t) => { Time.Text = new MyDateTime(DateTime.Now).ToString(); };

timer.Start();

CreationDateColumn.IsChecked = true;

ModificationDateColumn.IsChecked = true;

SizeColumn.IsChecked = true;

ShowHidden.IsChecked = false;

}

private void CreationDateColumn\_CheckChanged(object sender, RoutedEventArgs e)

{

FileTable.Columns[2].Visibility = CreationDateColumn.IsChecked != null && CreationDateColumn.IsChecked.Value

? Visibility.Visible

: Visibility.Collapsed;

}

private void ModificationDateColumn\_CheckChanged(object sender, RoutedEventArgs e)

{

FileTable.Columns[3].Visibility =

ModificationDateColumn.IsChecked != null && ModificationDateColumn.IsChecked.Value

? Visibility.Visible

: Visibility.Collapsed;

}

private void SizeColumn\_CheckChanged(object sender, RoutedEventArgs e)

{

FileTable.Columns[4].Visibility = SizeColumn.IsChecked != null && SizeColumn.IsChecked.Value

? Visibility.Visible

: Visibility.Collapsed;

}

private void User\_MouseEnter(object sender, MouseEventArgs e)

{

User.Foreground = Brushes.White;

}

private void User\_MouseLeave(object sender, MouseEventArgs e)

{

User.Foreground = Brushes.LightGray;

}

private void ShowInfo\_Click(object sender, RoutedEventArgs e)

{

if (FileTable.CurrentCell.Item is DataGridCellInfo ||

!(FileTable.SelectedItem is ExplorerFile file)) return;

PropertiesWindow propertiesWindow = new PropertiesWindow(file.MftEntry);

propertiesWindow.ShowDialog();

UpdateFileTable();

}

private void ShowControlWindow(object sender, RoutedEventArgs e)

{

foreach (Window window in Application.Current.Windows)

{

if (window is ControlWindow)

{

window.Activate();

return;

}

}

ControlWindow controlWindow = new ControlWindow(FileSystem.FileSystem.CurrentUser.Id == 0) {Owner = this};

controlWindow.Show();

}

private void User\_MouseLeftButtonDown(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Вы точно хотите выйти из учётной записи?", "Выход из системы", MessageBoxButton.YesNo,

MessageBoxImage.None) == MessageBoxResult.Yes)

{

System.Diagnostics.Process.Start(Application.ResourceAssembly.Location);

Application.Current.Shutdown();

}

}

private void PowerOff\_Click(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Вы точно хотите завершить сеанс?", "Завершение работы", MessageBoxButton.YesNo,

MessageBoxImage.None) == MessageBoxResult.Yes)

Close();

}

}

// NameWindow.xaml.cs

public partial class NameWindow

{

private readonly FsException.Command \_command;

private readonly FsException.Element \_element;

private readonly DirectoryRecord \_record;

private readonly Path \_path;

public NameWindow(FsException.Element element, Path path)

{

InitializeComponent();

\_command = FsException.Command.Create;

\_element = element;

\_path = path;

Title = FsException.GetCaption(\_command, \_element);

}

public NameWindow(ExplorerFile file, Path path)

{

InitializeComponent();

\_command = FsException.Command.Rename;

\_element = file.IsDirectory ? FsException.Element.Folder : FsException.Element.File;

\_record = new DirectoryRecord(file);

\_path = path;

Title = FsException.GetCaption(\_command, \_element);

FileName.Text = file.FullName;

}

private void Load(object sender, RoutedEventArgs e)

{

FileName.Select(0,

FileName.Text.Contains(".") && \_element == FsException.Element.File

? FileName.Text.LastIndexOf('.')

: FileName.Text.Length);

FileName.Focus();

}

private void Ok\_Click(object sender, RoutedEventArgs e)

{

try

{

FileName.Text.ParseFullName(\_element == FsException.Element.Folder, out var fileName,

out var extension);

if (fileName.Length > 26) throw new FsException(FsException.Code.NameTooLong, fileName);

if (extension.Length > 5) throw new FsException(FsException.Code.ExtensionTooLong, "." + extension);

if (\_command == FsException.Command.Create)

{

SystemCalls.Create(\_path, fileName, extension,

\_element == FsException.Element.Folder

? MftHeader.Attribute.Directory

: MftHeader.Attribute.None);

}

else SystemCalls.Rename(\_path, \_record, FileName.Text);

DialogResult = true;

}

catch (FsException ex)

{

ex.ShowError(\_command, \_element);

Close();

}

}

private void FileName\_TextChanged(object sender, TextChangedEventArgs e)

{

if (\_record != null)

Оk.IsEnabled = FileName.Text.TrimEnd().TrimEnd().Length > 0 && \_record.FileName != FileName.Text;

else Оk.IsEnabled = FileName.Text.TrimEnd().TrimEnd().Length > 0;

}

private void Cancel\_Click(object sender, RoutedEventArgs e)

{

DialogResult = false;

}

}

// PropertiesWindow.xaml.cs

public partial class PropertiesWindow

{

public PropertiesWindow(int mftEntry)

{

InitializeComponent();

if(mftEntry < 0) throw new ArgumentException(nameof(mftEntry));

\_mftHeader = SystemCalls.GetMftHeader(mftEntry);

\_mftFileEntry = mftEntry;

\_previousPermissions = new Permission(\_mftHeader.Permissions.GetBytes());

\_previousAttributes = \_mftHeader.Attributes;

}

private readonly MftHeader \_mftHeader;

private readonly int \_mftFileEntry;

private readonly Permission \_previousPermissions;

private readonly byte \_previousAttributes;

private Permission.UserSign \_currentUserSign;

private void PropertiesWindow\_Loaded(object sender, RoutedEventArgs e)

{

FileName.Text = \_mftHeader.GetFullName();

bool isDirectory = \_mftHeader.HasAttribute(MftHeader.Attribute.Directory);

Type.Text = isDirectory ? "Папка с файлами" :

\_mftHeader.Extension == "" ? "Файл" : \_mftHeader.Extension;

CreationDateTime.Text = \_mftHeader.CreationDate.ToString();

ModificationDateTime.Text = \_mftHeader.CreationDate.ToString();

Size.Text = (isDirectory ? SystemCalls.GetDirectorySize(\_mftFileEntry) : \_mftHeader.Size).ToString("N0") + " Б";

Hidden.IsChecked = \_mftHeader.HasAttribute(MftHeader.Attribute.Hidden);

ReadOnly.IsChecked = \_mftHeader.HasAttribute(MftHeader.Attribute.ReadOnly);

OwnerName.Content = SystemCalls.GetUserById(\_mftHeader.UserId).Name;

\_currentUserSign = \_mftHeader.UserId == FileSystem.FileSystem.CurrentUser.Id ? Permission.UserSign.Owner :

FileSystem.FileSystem.CurrentUser.IsAdministrator ? Permission.UserSign.Administrator : Permission.UserSign.Other;

switch (\_currentUserSign)

{

case Permission.UserSign.Administrator: UserList.SelectedItem = Administrator;

break;

case Permission.UserSign.Owner:

UserList.SelectedItem = OwnerName;

break;

case Permission.UserSign.Other:

UserList.SelectedItem = AllUsers;

break;

}

Permissions.IsEnabled = \_mftHeader.HasPermissions(FileSystem.FileSystem.CurrentUser, Permission.Rights.FullControl) ||

FileSystem.FileSystem.CurrentUser.IsAdministrator;

ReadOnly.IsEnabled = Hidden.IsEnabled =

\_mftHeader.HasPermissions(FileSystem.FileSystem.CurrentUser, Permission.Rights.Write);

}

private void UpdatePermissions()

{

FullControl.IsChecked = \_mftHeader.Permissions.CheckRights(\_currentUserSign, Permission.Rights.FullControl);

Modify.IsChecked = \_mftHeader.Permissions.CheckRights(\_currentUserSign, Permission.Rights.Modify);

Write.IsChecked = \_mftHeader.Permissions.CheckRights(\_currentUserSign, Permission.Rights.Write);

Read.IsChecked = \_mftHeader.Permissions.CheckRights(\_currentUserSign, Permission.Rights.Read);

}

private void Selected\_Administrator(object sender, RoutedEventArgs e)

{

\_currentUserSign = Permission.UserSign.Administrator;

UpdatePermissions();

}

private void Selected\_Owner(object sender, RoutedEventArgs e)

{

\_currentUserSign = Permission.UserSign.Owner;

UpdatePermissions();

}

private void Selected\_Other(object sender, RoutedEventArgs e)

{

\_currentUserSign = Permission.UserSign.Other;

UpdatePermissions();

}

private void ChangePermission(Permission.Rights right, bool state)

{

\_mftHeader.Permissions.SetPermission(\_currentUserSign, right, state);

UpdatePermissions();

}

private void FullControl\_Checked(object sender, RoutedEventArgs e)

{

ChangePermission(Permission.Rights.FullControl, FullControl.IsChecked != null && FullControl.IsChecked.Value);

}

private void Modify\_Checked(object sender, RoutedEventArgs e)

{

ChangePermission(Permission.Rights.Modify,

Modify.IsChecked != null && Modify.IsChecked.Value);

}

private void Write\_Checked(object sender, RoutedEventArgs e)

{

ChangePermission(Permission.Rights.Write,

Write.IsChecked != null && Write.IsChecked.Value);

}

private void Read\_Checked(object sender, RoutedEventArgs e)

{

ChangePermission(Permission.Rights.Read, Read.IsChecked != null && Read.IsChecked.Value);

}

private void Readonly\_CheckChanged(object sender, RoutedEventArgs e)

{

\_mftHeader.SetAttribute(MftHeader.Attribute.ReadOnly,

ReadOnly.IsChecked != null && ReadOnly.IsChecked.Value);

}

private void Hidden\_CheckChanged(object sender, RoutedEventArgs e)

{

\_mftHeader.SetAttribute(MftHeader.Attribute.Hidden, Hidden.IsChecked != null && Hidden.IsChecked.Value);

}

private void Ok\_Click(object sender, RoutedEventArgs e)

{

if (!\_mftHeader.Permissions.GetBytes().SequenceEqual(\_previousPermissions.GetBytes()) ||

\_mftHeader.Attributes != \_previousAttributes)

SystemCalls.UpdateMftEntry(\_mftHeader, \_mftFileEntry);

DialogResult = true;

}

private void Cancel\_Click(object sender, RoutedEventArgs e)

{

if (!\_mftHeader.Permissions.GetBytes().SequenceEqual(\_previousPermissions.GetBytes()) ||

\_mftHeader.Attributes != \_previousAttributes)

if (MessageBox.Show("Сохранить изменения?", "Сохранения свойств объекта",

MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

SystemCalls.UpdateMftEntry(\_mftHeader, \_mftFileEntry);

DialogResult = true;

}

Close();

}

}

// TextEditorWindow.xaml.cs

public partial class TextEditorWindow

{

public TextEditorWindow(ExplorerFile file, string fullName)

{

InitializeComponent();

Title = fullName;

FileContent.Text = \_content = Encoding.UTF8.GetString(SystemCalls.ReadFileData(file.MftEntry));

\_record = new DirectoryRecord(file);

}

private readonly DirectoryRecord \_record;

private readonly string \_content;

private void Save\_Click(object sender, RoutedEventArgs e)

{

try

{

SystemCalls.Save(\_record, Encoding.UTF8.GetBytes(FileContent.Text));

Save.IsEnabled = false;

}

catch (FsException fsException)

{

fsException.ShowError(FsException.Command.Save,

\_record.HasAttribute(MftHeader.Attribute.Directory)

? FsException.Element.Folder

: FsException.Element.File);

}

}

private void Cancel\_Click(object sender, RoutedEventArgs e)

{

Close();

}

private void FileContent\_OnTextChanged(object sender, TextChangedEventArgs e)

{

Save.IsEnabled = \_content != FileContent.Text;

}

}

// Эмуляция работы межпроцессного взаимодействия

// writer\_fd.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/shm.h>

#include <sys/ipc.h>

int main()

{

int shmid; // Дескриптор сегмента.

char \*segptr; // Указатель на сегмент.

key\_t key = ftok(".", 'A');// Создаём ключ сегмента.

shmctl(shmid, IPC\_RMID, 0);

do{

// обнаружение или создание сегмента разделяемой памяти

if((shmid = shmget(key, 1, IPC\_CREAT|IPC\_EXCL|0777) == -1))

{

if((shmid = shmget(key, 1, 0)) == -1)

perror("shmget");

}

}while((segptr = shmat(shmid, 0, 0)) == -1);

// Привязываем или размещеним сегмент в адресном пространстве процесса.

segptr = "1";

int fd\_fifo; // Дескриптор FIFO.

char text[1000];

char temp[1];

printf("Введите текст: \n");

scanf ("%[^\n]s", text);

unlink("fifo"); // Удаляем файл, если он уже создан.

// Создаем FIFO.

if((mkfifo("fifo", 0777)) == -1)

{

fprintf(stderr, "Невозможно создать FIFO!\n");

exit(0);

}

//Открываем fifo для чтения и записи.

if((fd\_fifo=open("fifo", O\_RDWR)) == - 1)

{

fprintf(stderr, "Невозможно открыть FIFO!\n");

exit(0);

}

write(fd\_fifo,text,strlen(text));

while((shmat(shmid, 0, 0) != -1) && (fd\_fifo=open("fifo", O\_RDWR)) != - 1){

sleep(1);

}

shmctl(shmid, IPC\_RMID, 0);

unlink("fifo"); // Удаляем fifo.

return 0;

}

// reader\_fd.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/shm.h>

#include <sys/ipc.h>

int main()

{

int shmid; // Дескриптор сегмента.

char \*segptr; // Указатель на сегмент.

key\_t key = ftok(".", 'A');// Создание ключа.

// Обнаружение или создание сегмента разделяемой памяти.

if((shmid = shmget(key, 1, 0)) == -1)

{

fprintf(stderr, "Канал FIFO не создан!\n");

exit(0);

}

// Привязка или размещение сегмента в адресном пространстве процесса.

if((segptr = shmat(shmid, 0, 0)) == -1)

{

perror("shmat");

exit(1);

}

int fd\_fifo; // Дескриптор FIFO.

char buf[100];

// Открываем fifo для чтения и записи.

if((fd\_fifo=open("fifo", O\_RDWR)) == - 1)

{

fprintf(stderr, "Невозможно открыть FIFO!\n");

exit(0);

}

if(read(fd\_fifo, &buf, sizeof(buf)) == -1)

fprintf(stderr, "Невозможно прочесть из FIFO!\n");

else

printf("Прочитано из FIFO : %s\n",buf);

segptr = "0";

shmctl(shmid, IPC\_RMID, 0);

unlink("fifo");

return 0;

}

// shm.c

#include <stdio.h>

#include <string.h>

#include <sys/shm.h> // библиотека для работы с разделяемой памятью

#include <sys/ipc.h> // библиотека для работы с межпроцессным взаимодействием

#include <sys/types.h> // библиотека для работы с разными типами данных

#include <ctype.h>

#define SEGSIZE 100

main()

{

key\_t key; // Ключ сегмента.

int shmid; // Дескриптор сегмента.

char \*segptr; // Указатель на сегмент.

key = ftok(".", 'А');// Создание ключа.

// Обнаружение или создание сегмента разделяемой памяти.

if((shmid = shmget(key, SEGSIZE, IPC\_CREAT|IPC\_EXCL|0777)) == -1)

{

if((shmid = shmget(key, SEGSIZE, 0)) == -1)

{

perror("shmget");

exit(1);

}

}

else

printf("Сегмент разделяемой памяти создан! \n");

// Привязка или размещение сегмента в адресном пространстве процесса.

if((segptr = shmat(shmid, 0, 0)) == -1)

{

perror("shmat");

exit(1);

}

// Вывод текста.

printf("[w] - Записать в сегмент \n");

printf("[r] - Прочитать из сегмента \n");

char cKey;

cKey = getchar();

switch(tolower(cKey))

{

char text[1000];

char temp[1];

case 'w':

printf("Введите текст: \n");

scanf ("%c", temp);

fflush(stdin);

scanf ("%[^\n]s", text);

writeShm(shmid, segptr, text);

break;

case 'r':

readShm(shmid, segptr);

break;

}

}

writeShm(int shmid, char \*segptr, char string[])

{

strcpy(segptr, string);

printf("Строка успешно добавлена! \n");

}

readShm(int shmid, char \*segptr)

{

printf("Ваш текст:\n%s \n", segptr);

}