Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Специальность (направление, профиль) программная инженерия

Курсовой проект

защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Б. Целебровский

(подпись руководителя проекта) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Курсовой проект

Моделирование структуры и работы файловой системы FAT

(тема курсового проекта)

Пояснительная записка

по дисциплине Курсовая работа по операционным системам

КП 09.03.04.23.000 ПЗ

(обозначение документа)

Студент группы ПИ-02 Чередов Р.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.\_\_.2023

(фамилия, имя, отчество) (подпись) (дата)

Руководитель проекта ассистент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Б. Целебровский

(должность, ученое звание) (подпись) (инициалы, фамилия)

БАРНАУЛ 2023

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный технический университет

имени И.И. Ползунова»

Факультет информационных технологий

Кафедра «Прикладная математика»

З А Д А Н И Е

на курсовой проект по дисциплине «Курсовая работа по операционным системам»

студенту группы ПИ-02 Чередов Роману Алексеевичу

Тема курсового проекта: «Моделирование структуры и работы файловой системы FAT».

Календарный план работы:

|  |  |  |
| --- | --- | --- |
| № этапа | Содержание этапа | Недели семестра |
| 1 | Получение задания | 1 |
| 2 | Описание предметной области и постановка задачи | 2 |
| 3 | Проектирование программы | 3-4 |
| 4 | Реализация программы | 5-13 |
| 5 | Оформление пояснительной записки | 14 |
| 6 | Защита курсового проекта | 15-16 |

Руководитель проекта ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Б. Целебровский подпись

Дата выдачи задания «9» февраля 2023 г.

число месяц год

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Чередов Р.А.

подпись

# Оглавление

[Оглавление 3](#_Toc148714895)

[Введение 4](#_Toc148714896)

[1 Описание предметной области и постановка задачи 5](#_Toc148714897)

[1.1 Описание предметной области 5](#_Toc148714898)

[1.2 Постановка задачи 7](#_Toc148714899)

[2 Проектирование 9](#_Toc148714900)

[2.1 Описание структуры классов 9](#_Toc148714901)

[2.2 Диаграмма классов 10](#_Toc148714902)

[3 Разработка программного обеспечения 11](#_Toc148714903)

[3.1 Инструменты и технологии 11](#_Toc148714904)

[3.2 Реализация 11](#_Toc148714905)

[Заключение 12](#_Toc148714906)

[Список использованных источников 13](#_Toc148714907)

[Приложение А. Тестирование программного обеспечения 14](#_Toc148714908)

# Введение

Главный элемент системы пользования компьютера – Файл. Именно его используют для взаимодействия с системой, а именно хранение и обработка данных. Для выполнений операций над файлами (открытие, закрытие, чтение, редактирование и т.д.) используют файловую систему.

Файловая система – порядок, определяющий способ хранения, организации и именования информации в компьютере, а также в любых других электронном оборудовании. Файловая система определяет формат содержимого и способ хранения информации, которую принято формировать в виде каталогов. Данная файловая система определяет размер имен файлов, максимально возможный вес файла и раздела и их набор атрибутов.

При помощи функционала взаимодействия программ API, файловая система связывает носитель информации с программным обеспечением, организуя доступ к конкретным файлам. При обращении к файлу программа использует данные о его имени, размере и атрибутах. Вся остальная информация получается от драйвера файловой системы.

Данный проект передает не весь функционал файловой структуры FAT, но даст достаточной информации для знакомства с характерными особенностями и понимания их сути.

# 1 Описание предметной области и постановка задачи

# 1.1 Описание предметной области

**FAT** –классическая архитектура файловой системы, которая из-за своей простоты до сих пор широко применяется для флэш-накопителей. Используется в дискетах, ssd картах и некоторых других носителей, ранее находило применение и на жестких дисках.

В этой файловой системе смежные сектора дискового пространства объединяются в кластеры. Количество секторов в кластере равняется степени двойки. Для хранения данных файла отводится минимум один кластер.

Пространство тома FAT делится на несколько **смежных областей**:

* **Зарезервированная область** – используется при инициализации тома.
* **Область таблицы FAT** – содержит массив индексов соответствующих кластеров.
* **Область данных** – содержимое файлов.

Если кластер принадлежит файлу, то соответствующая ему ячейка в таблице FAT содержит номер следующего кластера этого же файла. Если ячейка - последний кластер файл, то она имеет специальное значение (0xFFFF для FAT16). Таким образом выстраивается цепочка кластеров файла. Неиспользуемые кластеры – нули в таблице.

Между кластерами и индексными указателями таблицы имеется взаимно однозначное соответствие — N-й указатель соответствует кластеру с тем же номером. Первому кластеру области данных присваивается номер 2. Значение индексного указателя соответствует состоянию соответствующего кластера. Возможны следующие состояния:

* Кластер свободен – обнуление указателя
* Кластер занят (не является последним кластером файла) – указатель содержит номер следующего кластера файла.
* Кластер последний кластер файла – указатель содержит метку EOC (end of clusterchain), значение зависит от версии FAT: FAT16 – значение> или = 0xFFF8(по умолчанию 0xFFFF), FAT32 – значение> или = 0x0FFFFFF8(по умолчанию 0x0FFFFFFF).

После окончания последней таблицы FAT следует область данных, содержащая файлы и папки. Каталог FAT является обычным файлом, помеченным специальным атрибутом. Содержимым такого файла в любой версии FAT является цепочка 32-байтовых файловых записей. Каталог не может штатно содержать два файла с одинаковым именем: если программа проверки диска обнаруживает искусственно созданную пару файлов с идентичным именем в одном каталоге, один из них переименовывается.

Единственным обязательно присутствующим каталогом является корневой каталог. В FAT32 корневой каталог, имеет переменный размер и является цепочкой кластеров, и имеет следующие отличия от других каталогов тома FAT:

* Нет меток даты и времени
* Нет собственного имени (исключение – «\»)
* Единственный каталог, в котором может штатно располагаться файл метки тома.

В структуре каталогов файлу отводится первый незанятый кластер, доступный на томе. Номер начального кластера позволяет определить местонахождение файла: каждый кластер содержит указатель на следующий кластер или значение FFFF, указывающее на то, что это последний кластер в цепочке кластеров, занимаемых файлом.

Файловая запись FAT32 состоит из следующего:

* **DIR\_Name –** 11 байтовое поле по адресу 0, содержащие имя файла
* **DIR\_Attr –** байт отвечающий за атрибуты файла
* **DIR\_NTRes –** байт, использующийся в windows NT
* **DIR\_CrtTimeTenth –** байт, счетчик десятков миллисекунд времени создания файла, допускаются значения от 0 до 199
* **DIR\_CrtTime –** 2 байта, время создания файла, точность до 2х секунд
* **DIR\_CrtDate –** 2 байта, дата создания файла
* **DIR\_LstAccDate –** 2 байта, дата последнего доступа к файла( когда его последний раз «трогали»)
* **DIR\_FstClusHI –** 2 байта, номер первого кластера файла
* **DIR\_WrtTime –** 2 байта, время последней записи файла
* **DIR\_WrtDate –** 2 байта, время последней записи файла
* **DIR\_FstClusLO –** 2 байта, номер первого кластера файла
* **DIR\_FileSize –** 4 байта, содержит значение размера файла в байтах.

Ограничение FAT32 – максимальное значение размера файла составляет 0xFFFFFFFF( или 4 Гбайта – 1 байт).

Если первый байт записи FAT (то есть DIR\_Name[0]) содержит 0xE5 или 0x05, это значит, что запись свободна (соответствующий файл был удален). Ноль в DIR\_Name[0] означает, что свободна не только эта запись, но и все следующие записи каталога; Windows не анализирует остаток каталога после обнуленной записи.

В байте атрибутов верхние два бита являются резервными и всегда должны быть обнулены. Остальные биты распределяются таким образом, что значение 0x01 соответствует атрибуту «только для чтения», 0x02 – «скрытый», 0x04 – «системный», 0x20 – «архивный». Набор нескольких атрибутов составляется суммированием основных значений. Кроме этих стандартных атрибутов используются следующие: 0x10 – свидетельствует, что файл является каталогом (контейнером для других файлов); 0x08 – ATTR\_VOLUME\_ID, специальный атрибут уникального файла нулевого размера в корневом каталоге, имя которого считается меткой тома.

Поле DIR\_Name логически разбивается на первые 8 символов, образующие имя файла, и последние 3, образующие расширение. Точка-разделитель добавляется на уровне операционной системы и не хранится в поле имени. Если имя и расширение файла не заполняют отведённое для них место, остальные байты поля DIR\_Name заполняются пробелами (0x20). Все буквенные символы 8.3 имени всегда переводятся и сохраняются в поле DIR\_Name в верхнем регистре. Имя и расширение файла могут содержать любую комбинацию букв, цифр или символов с ASCII-кодами свыше 127; специальные символы распределяются на три группы:

* Разрешенные: ! # $ % & () - @ ^ \_ ` { } ~ '
* Запрещенные: +.; = [ ]
* Служебные: \* ? <: > / \ | "

Двухбайтовая отметка даты имеет следующий формат:

* Биты 0-4 это день месяца, значения от 1-31
* Биты 5-8 это месяц года, значения от 1-12
* Биты 9-15 это год, от 0 до 127(то есть от 1820 до 2107).

Временные отметки задаются при создании и не изменяются при записи в него, но могут изменяться если переименовать и переместить. Дата последнего доступа к файлу обновляется при каждом обращении, например, при смене атрибутов или переименовании файла.

# 1.2 Постановка задачи

Создать модель файловой системы FAT, реализующую следующие функции:

При запуске программы открывается окно, на нем выводится таблица кластеров (из которых два первых заняты системой – отображается графически), поле с логом и два поля с имеющимися в системе фалами и каталогами соответственно.

Так же есть меню с пунктами для файлов и каталогов, которые реализуют следующие функции:

* **Создание каталога –** создается запись с атрибутом dir, ей можно задать имя и установить флаг.
* **Создание файла –** создается запись с атрибутом fil, ей можно задать имя, тип расширения, размер и установить флаги, указанное количество кластеров заполняется одной из цифр (2-8), присвоенных этому файлу.
* **Редактирование имени каталога –** выбранному каталогу присваивается новое имя, при этом обновляется отметка даты и времени.
* **Редактирование имени файла** – выбранному файлу присваивается новое имя, при этом обновляется отметка даты и времени
* **Перемещение каталога –** выбранный каталог перемещается в указанный каталог, обновляется отметки даты и времени.
* **Перемещение файла –** выбранный файл перемещается в указанный каталог, при этом обновляются отметки даты и времени модификации.
* **Просмотр информации о каталоге –** выводится информация, без возможности редактирования.
* **Просмотр информации о файле -**  выводится информация, без возможности редактирования.
* **Удаление каталога –** удаляется выбранный каталог, при этом его дочерние файлы перемещаются в корневой каталог.
* **Удаление файла –** удаляется данный файл, при этом очищая занятые им кластеры.

При каждом завершении операции обновляется лог, куда записывается информация о совершенных действиях и список текущих файлов и каталогов системы.

Информация о каталоге и о файле отображаются в текстовых полях.

# 2 Проектирование

# 2.1 Описание структуры классов

Класс Disk – основной класс, представляет собой том. Через него ведется управление всей системой файлов.

Класс BPB – класс отвечающий за структуру.

Класс FSInfo – информация о пустых кластерах.

Класс FAT – представляет собой структуру FAT и отвечает за всю работу с ней.

Класс FileStruct – это файловая запись.

Класс DirectoryCluster – это кластер, занятый директорией.

# 2.2 Диаграмма классов

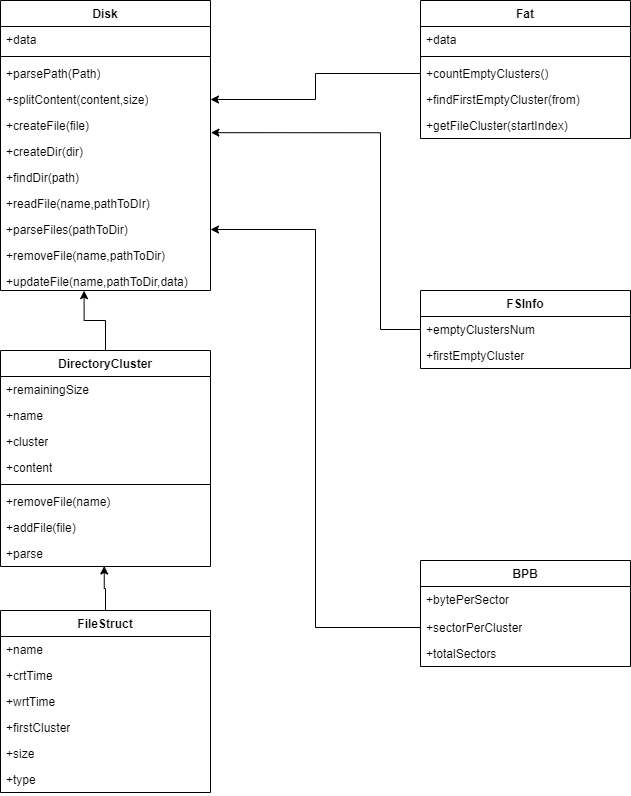


Рисунок 2.1 - Диаграмма классов

# 3 Разработка программного обеспечения

# 3.1 Инструменты и технологии

Для разработки модели был выбран стек JavaScript, HTML, CSS.

JavaScript - язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации ECMAScript. Основные архитектурные черты: динамическая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

HTML – язык гипертекстововй разметки документов для веб-страниц.

CSS – формальный язык декорирования и описания внешнего вида веб-страницы. Использует язык разметки.

Данный стек был выбран для возможности реализации модели в веб-версии

# 3.2 Реализация

При открытии веб-страницы инициализируется новый диск, содержащий 1 папку – My computer и два файла – Disk C и Disk D. После инициализации возможно создавать свои файлы, папки и посмотреть какие кластеры они занимают, есть возможность удалить эти файлы и отредактировать их. Все это реализовано в визуальном интерфейсе.

# Заключение

Разработанная программа моделирования работы файловой системы FAT, которая предоставляет полную возможность работы с файлами. С помощью графического интерфейса пользователь может отследить занятые файлами кластеры.

Программа обладает следующими функциональными возможностями:

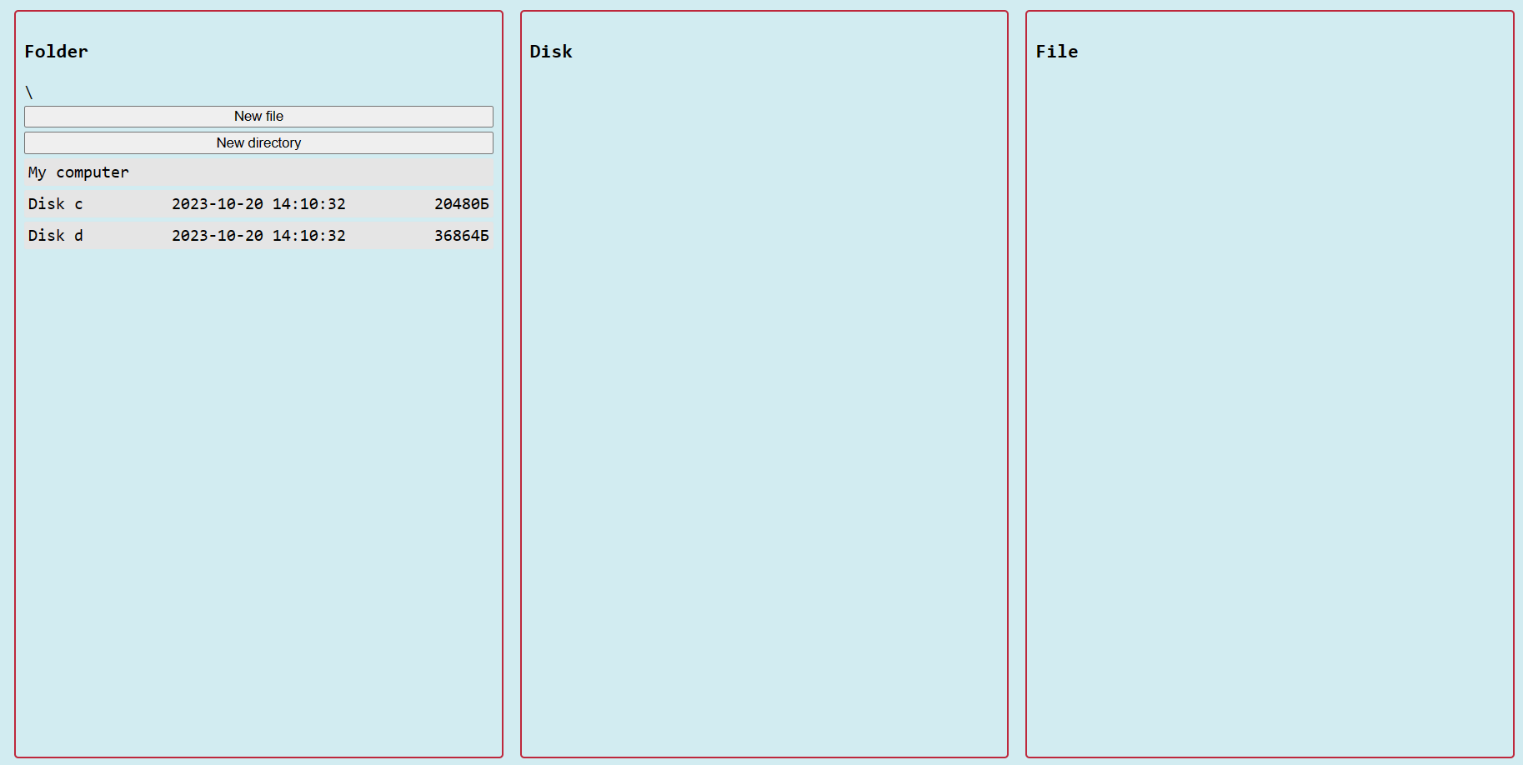
* Просмотр файлов и директорий – пользователь имеет возможность просмотреть информацию о файлах и директориях
* Создание файлов и директорий – пользователь может создать новые файлы и директории через интерфейс
* Редактирование и удаление файлов - – пользователь может удалять и редактировать файлы через интерфейс

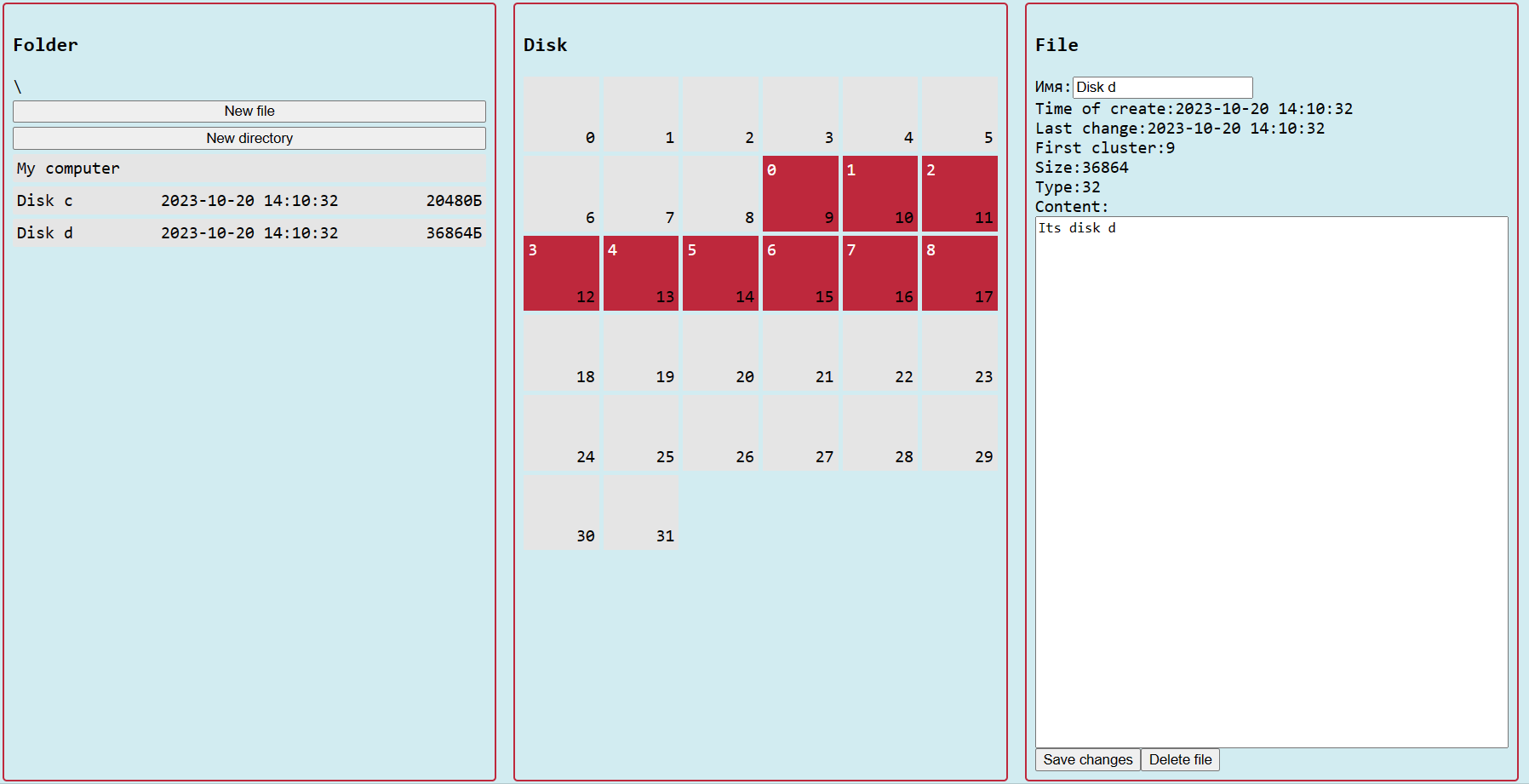
Разработанная программа моделирования обладает интуитивным и удобным интерфейсом, который позволяет пользователям легко взаимодействовать с системой.

# Список использованных источников

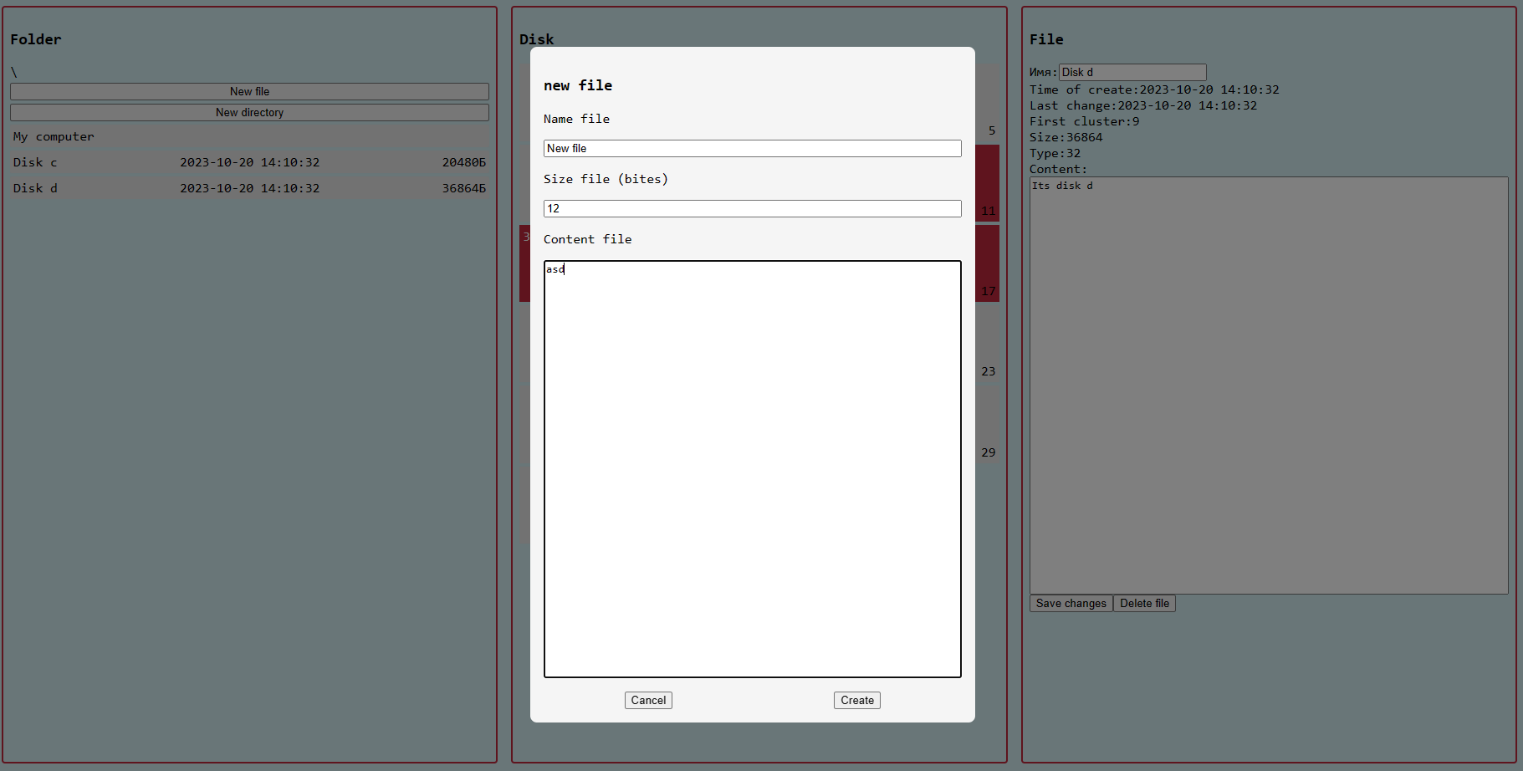
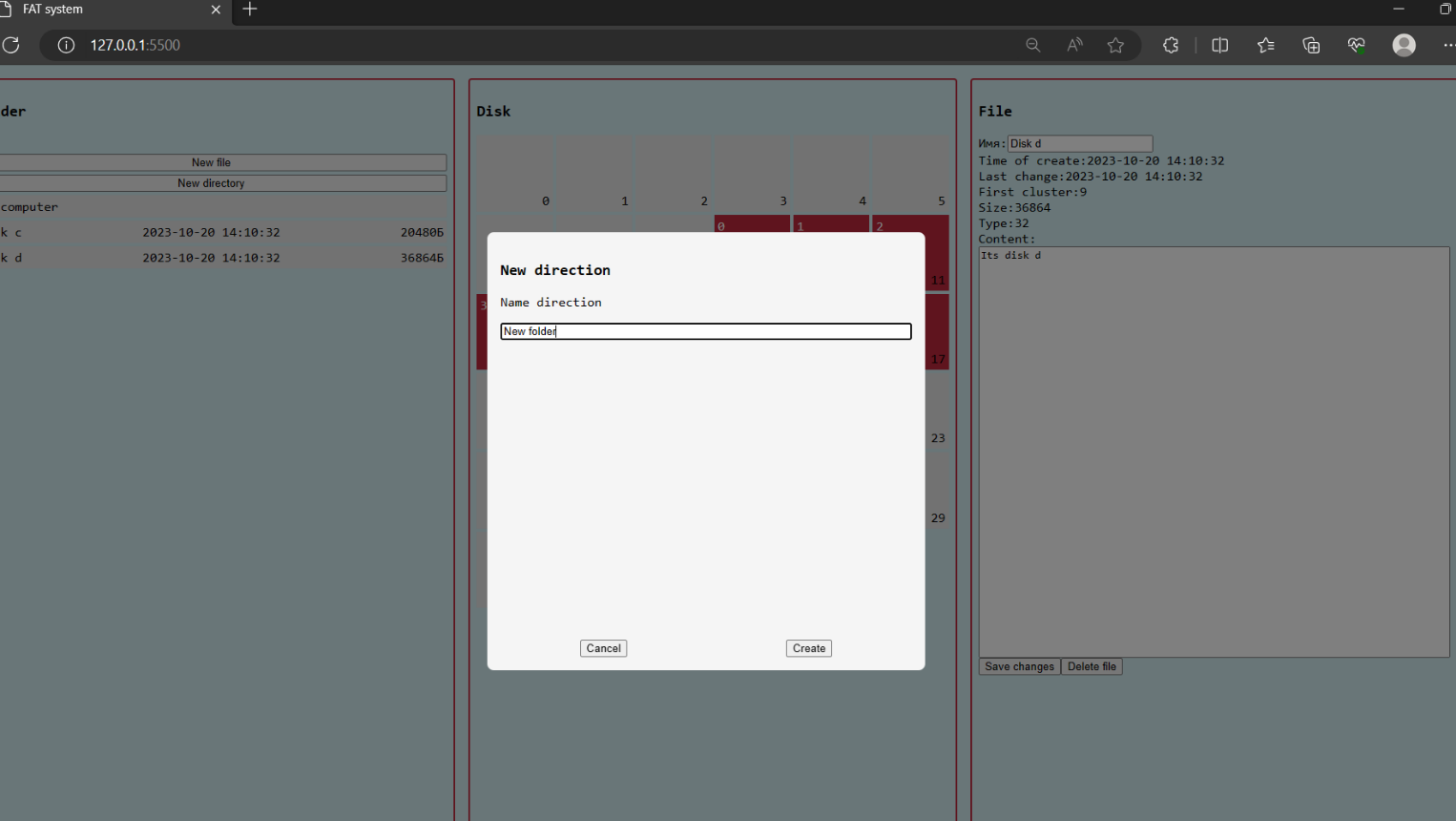
1. Microsoft FAT Specification. — Текст : электронный // CBA : [сайт]. — URL: [Microsoft FAT Specification : Microsoft : Free Download, Borrow, and Streaming : Internet Archive](https://archive.org/details/FAT_Specification)
2. Emulation FAT32 on C — Текст : электронный // CBA : [сайт]. — URL: [FAT32/FAT32.c at main · quinnhipp/FAT32 · GitHub](https://github.com/quinnhipp/FAT32/blob/main/FAT32.c)
3. Emulation FAT32 on C — Текст : электронный // CBA : [сайт]. — URL: [FAT32/FAT32.c at main · quinnhipp/FAT32 · GitHub](https://github.com/quinnhipp/FAT32/blob/main/FAT32.c)
4. Emulation FAT16/FAT32 on javaScript — Текст : электронный // CBA : [сайт]. — URL: [GitHub - natevw/fatfs: Standalone FAT16/FAT32 filesystem implementation in JavaScript](https://github.com/natevw/fatfs)
5. File Allocation Table — Текст : электронный // CBA : [сайт]. — URL: [File Allocation Table - Wikipedia](https://en.wikipedia.org/wiki/File_Allocation_Table)

# Приложение А. Тестирование программного обеспечения

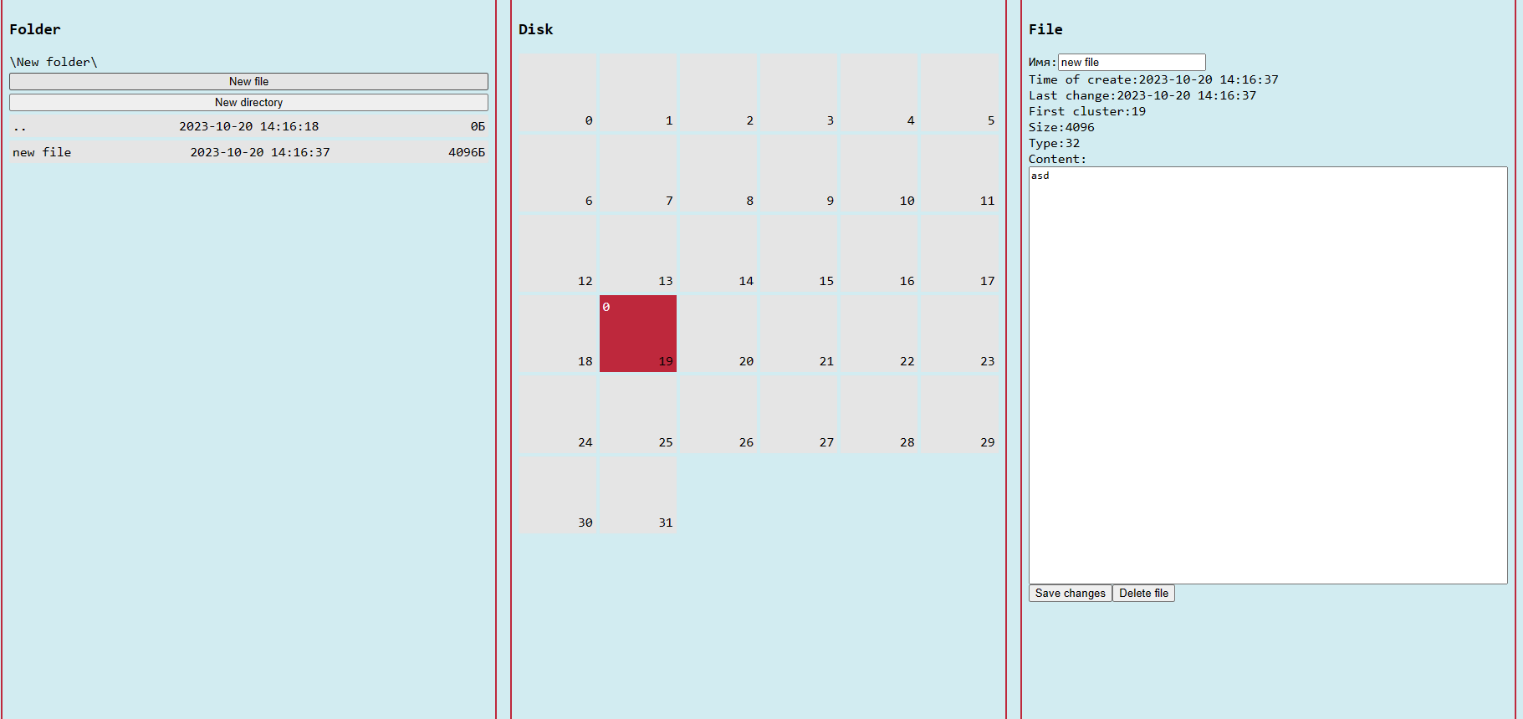
При инициализации можно увидеть на диске 2 файла и директорию. Открываем один из них.

При открытии файла появляется информация о файле, а также показывается занятые им кластеры.

Окно создания нового файла в текущей директории

Также можно создать новую директорию

Попробуем создать новую директорию и в ней создать новый файл

**Приложение Б. Исходный текст программы**

**Index.himl**

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport"

          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <link href="style.css" rel="stylesheet">

    <title>FAT system</title>

</head>

<body>

    <div class="content">

        <div id="modals"></div>

        <div class="column">

            <h3 class="column\_\_h">Folder</h3>

            <div id="file-tree" class="file-tree"></div>

        </div>

        <div class="column">

            <h3 class="column\_\_h">Disk</h3>

            <div id="clusters-info"></div>

        </div>

        <div class="column">

            <h3 class="column\_\_h">File</h3>

            <div id="file-info"></div>

        </div>

    </div>

    <script type="module" src="./index.js"></script>

</body>

</html>

**Style.css**

html, body {

    padding: 0;

    margin: 0;

    background: #d2ecf1;

    font-family: "Consolas", sans-serif;

}

.content {

    display: flex;

    gap: 1em;

    min-height: 100vh;

    padding: 1em;

    box-sizing: border-box;

}

.column {

    flex: 1;

    border: 2px solid #be283c;

    padding: 0.5em;

    border-radius: 0.25em;

}

.file-tree {

    display: flex;

    flex-direction: column;

    gap: 0.25em;

}

.file {

    display: flex;

    width: 100%;

    border-radius: 2px;

    background: #e5e5e5;

    padding: 0.25em;

    box-sizing: border-box;

    justify-content: space-between;

}

.file:hover {

    background: #cbcbcb;

    cursor: pointer;

}

.file:active {

    background: #b2b2b2;

}

.file-content {

    display: flex;

    flex-direction: column;

    width: 100%;

}

.file-content\_\_text {

    width: 100%;

    box-sizing: border-box;

    height: 500px;

    resize: none;

}

.clusters {

    display: grid;

    grid-template-columns: repeat(6, 1fr);

    gap: 0.25em;

}

.cluster {

    box-sizing: border-box;

    background: #e5e5e5;

    aspect-ratio: 1 / 1;

    display: flex;

    justify-content: end;

    align-items: end;

    padding: 0.25em;

    position: relative;

}

.cluster\_full {

    background: #be283c;

}

.cluster\_\_index-in-chain {

    position: absolute;

    left: 0.25em;

    top: 0.25em;

    color: white;

}

.modal {

    position: absolute;

    left: 0;

    top: 0;

    right: 0;

    bottom: 0;

    z-index: 10000;

    background: rgba(0,0,0,0.5);

    display: flex;

    justify-content: center;

    align-items: center;

}

.modal-content {

    min-height: 500px;

    min-width: 500px;

    background: #f5f5f5;

    padding: 1em;

    border-radius: 0.5em;

    display: flex;

    flex-direction: column;

}

.form {

    display: flex;

    flex-direction: column;

    gap: 1em;

    flex: 1 0;

}

.form\_\_actions {

    display: flex;

    justify-content: space-around;

    flex: 1 0;

    align-items: end;

}

**Bpb.js**

export default class BPB {

    constructor(size, sectorSize, clusterSize) {

      this.bytePerSector = sectorSize

      this.sectorPerCluster = clusterSize

      this.totalSectors = size / sectorSize

    }

  }

**directoryCluster.js**

export default class DirectoryCluster {

    constructor(size, name, cluster) {

      this.remainingSize = size

      this.name = name

      this.content = []

      this.cluster = cluster

    }

    removeFile(name) {

      const index = this.content.findIndex(file => file.name === name)

      if (index !== -1) {

        this.content[index].name = String.fromCharCode(0x0E5) + this.content[index].name.slice(1)

      }

    }

    addFile(file) {

      const indexOfFree = this.content.findIndex(file => file.name[0] === String.fromCharCode(0x0E5))

      if (indexOfFree === -1) {

        if (this.remainingSize < 32) return 1

        this.content.push(file)

        this.remainingSize -= 32

      }

      else {

        this.content[indexOfFree] = file

      }

      return 0

    }

    parse() {

      return this.content.map(file => ({

        dir: this.name,

        name: file.name,

        size: file.size,

        type: file.type,

        wrtTime: file.wrtTime,

      })).filter(file => file.name[0] !== String.fromCharCode(0x0E5))}

  }

**Disk.js**

import BPB from "./BPB.js";

import FSInfo from "./FSInfo.js";

import Fat from "./Fat.js";

import FileStruct from "./FileStruct.js";

import DirectoryCluster from "./DirectoryCluster.js";

export default class Disk {

  constructor(size, sectorSize, clusterSize) {

    this.data = new Array(size / (sectorSize \* clusterSize)).fill(0)

    this.data[0] = [new BPB(size, sectorSize, clusterSize)]

    this.data[0].push(new FSInfo())

    this.data[0].push(new Fat(this.getTotalSectors(), this.getSectorPerCluster()))

    this.data[1] = new FileStruct("\\", null, null, 2, 0, 0x10);

    this.data[2] = new DirectoryCluster(this.getSectorSize() \* this.getSectorPerCluster(), "\\", 2)

  }

  getBPB = () => this.data[0][0]

  getFSInfo = () => this.data[0][1]

  getFat = () => this.data[0][2]

  getTotalSectors = () => this.getBPB().totalSectors;

  getSectorPerCluster = () => this.getBPB().sectorPerCluster;

  getSectorSize = () => this.getBPB().bytePerSector;

  findFirstEmptyCluster = (from = 3) => this.getFSInfo().firstEmptyCluster = this.getFat().findFirstEmptyCluster(from);

  countEmptyClusters = () => this.getFSInfo().emptyClustersNum = this.getFat().countEmptyClusters()

  parsePath(path) {

    return ["\\", ...path.split("\\").slice(1, -1)]

  }

  splitContent(content, size) {

    const clustersNum = Math.ceil(size / (this.getSectorSize() \* this.getSectorPerCluster()))

    const chunkLen = Math.ceil(content.length / clustersNum)

    const chunks = []

    for (let i = 0; i < clustersNum; i++) {

      const end = chunkLen \* (i + 1) > content.length ? content.length : chunkLen \* (i + 1)

      chunks.push(content.slice(chunkLen \* i, end))

    }

    return chunks

  }

  createFile({name, content, size, type, pathToDir}) {

    this.findFirstEmptyCluster()

    this.countEmptyClusters()

    const clustersNum = Math.ceil(size / (this.getSectorSize() \* this.getSectorPerCluster()))

    if (clustersNum > this.getFSInfo().emptyClustersNum) {

      alert("Not enough disk space")

      return

    }

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    if (dirStruct[dirStruct.length - 1].content.findIndex(f => f.name === name) !== -1) {

      alert("There is already such a file in this directory")

      return

    }

    let index = this.getFSInfo().firstEmptyCluster

    const ans = dirStruct[dirStruct.length - 1].addFile(new FileStruct(name, new Date(), new Date(), index, size, type))

    if (ans === 1) {

      alert("Not enough space in the directory")

      return

    }

    if (size<=content.length)

    {

      alert("out of space")

      return

    }

    const contentChunks = this.splitContent(content, size)

    let num = 0

    let remainingSize = size

    while (remainingSize > 0) {

      const nextFreeIndex = this.findFirstEmptyCluster(index + 1)

      const isLastCluster = remainingSize - this.getSectorSize() \* this.getSectorPerCluster() <= 0

      this.getFat().data[index] = isLastCluster ? 0xFFFFFFFF : nextFreeIndex

      this.data[index] = contentChunks[num]

      num++

      index = nextFreeIndex

      remainingSize -= this.getSectorSize() \* this.getSectorPerCluster()

    }

  }

  createDir({name, pathToDir}) {

    this.findFirstEmptyCluster()

    this.countEmptyClusters()

    if (this.getFSInfo().emptyClustersNum === 0) {

      alert("Not enough disk space")

      return

    }

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    if (dirStruct[dirStruct.length - 1].content.findIndex(f => f.name === name) !== -1) {

      alert("There is already such a file in this directory")

      return

    }

    let index = this.getFSInfo().firstEmptyCluster

    const ans = dirStruct[dirStruct.length - 1].addFile(new FileStruct(name, new Date(), new Date(), index, 0, 0x10))

    if (ans === 1) {

      alert("Not enough space in the directory")

      return

    }

    this.getFat().data[index] = 0xFFFFFFFF

    this.data[index] = new DirectoryCluster(this.getSectorSize() \* this.getSectorPerCluster(), name, index)

    this.data[index].addFile(new FileStruct("..", new Date(), new Date(), dirStruct[dirStruct.length - 1].cluster, 0, 0x01))

  }

  findDir(path = "\\") {

    const parsedPath = this.parsePath(path)

    const chain = [this.data[2]]

    for (let i = 1; i < parsedPath.length; i++) {

      const fileStruct = chain[i-1].content.find(str => str.name === parsedPath[i])

      if (fileStruct) {

        chain.push(this.data[fileStruct.firstCluster])

      }

      else {

        return -1

      }

    }

    return chain

  }

  readFile(name, pathToDir) {

    let content = ''

    let file = {}

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    file = dirStruct[dirStruct.length - 1].content.find(f => f.name === name)

    let index = file.firstCluster

    while (true) {

      content = content + this.data[index]

      index = this.getFat().data[index]

      if (index === 0xFFFFFFFF) break

    }

    const clusters = this.getFat().getFileClusters(file.firstCluster)

    return {...file, content, clusters}

  }

  parseFiles(pathToDir = "\\") {

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    return {files: dirStruct[dirStruct.length - 1].parse(), path: pathToDir}

  }

  removeFile(name, pathToDir) {

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    const file = dirStruct[dirStruct.length - 1].content.find(file => file.name === name)

    dirStruct[dirStruct.length - 1].removeFile(name)

    let index = file.firstCluster

    while (true) {

      const tmp = index

      index = this.getFat().data[index]

      this.getFat().data[tmp] = 0

      if (index === 0xFFFFFFFF) break

    }

  }

  updateFile(name, pathToDir, data = {}) {

    const dirStruct = this.findDir(pathToDir)

    if (dirStruct === -1) return;

    const file = dirStruct[dirStruct.length - 1].content.find(file => file.name === name)

    if (data?.name || data?.content) file.wrtTime = new Date()

    if (data?.name) file.name = data.name

    if (data?.content) {

      let index = file.firstCluster

      while (true) {

        const tmp = index

        index = this.getFat().data[index]

        this.getFat().data[tmp] = 0

        if (index === 0xFFFFFFFF) break

      }

      this.findFirstEmptyCluster()

      index = this.getFSInfo().firstEmptyCluster

      file.firstCluster = index

      const contentChunks = this.splitContent(data.content, file.size)

      let num = 0

      let remainingSize = file.size

      while (remainingSize > 0) {

        const nextFreeIndex = this.findFirstEmptyCluster(index + 1)

        const isLastCluster = remainingSize - this.getSectorSize() \* this.getSectorPerCluster() <= 0

        this.getFat().data[index] = isLastCluster ? 0xFFFFFFFF : nextFreeIndex

        this.data[index] = contentChunks[num]

        num++

        index = nextFreeIndex

        remainingSize -= this.getSectorSize() \* this.getSectorPerCluster()

        if(file.size<=content.length)

        {

          alert("out of space")

          return

        }

      }

    }

  }

}

**Fat.js**

export default class Fat {

    data = []

    constructor(sectorsNum, sectorsPerCluster) {

      this.data = new Array(sectorsNum / sectorsPerCluster).fill(0)

      this.data[0] = 0xFFFFF00

      this.data[1] = 0xFFFFFFFF

      this.data[2] = 0xFFFFFFFF

    }

    countEmptyClusters() {

      return this.data.reduce((acc, cluster) => {

        if (cluster === 0) acc++

        return acc

      }, 0)

    }

    findFirstEmptyCluster(from = 0) {

      for (let i = from; i < this.data.length; i++) {

        if (this.data[i] === 0) return i

      }

      return null

    }

    getFileClusters(startIndex) {

      if (this.data[startIndex] === 0xFFFFFFFF) return [startIndex]

      return [startIndex, ...this.getFileClusters(this.data[startIndex])]

    }

  }

**FileStruct.js**

export default class FileStruct {

    constructor(name, crtTime, wrtTime, cluster, size, type) {

      this.name = name;

      this.crtTime = crtTime;

      this.wrtTime = wrtTime;

      this.firstCluster = cluster;

      this.size = size;

      this.type = type;

    }

  }

**Format.js**

export const formatDate = (date) => date.toISOString().split("T").join(" ").split(".")[0]

**FSInfo.js**

export default class FSInfo {

    constructor() {

      this.emptyClustersNum = null

      this.firstEmptyCluster = null

    }

  }

**Index.js**

import Disk from "./Disk.js";

import {formatDate} from "./format.js";

let currentPath = "\\"

const disk = new Disk(131072, 512, 8)

disk.createDir({name: "My computer", pathToDir: currentPath})

disk.createFile({name: "Disk c", content: "Its disk c", size: 20480, type: 0x20, pathToDir: currentPath})

disk.createFile({name: "Disk d", content: "Its disk d", size: 36864, type: 0x20, pathToDir: currentPath})

updateTree()

function updateTree() {

  const {files, path} = disk.parseFiles(currentPath)

  renderList(files, path)

}

function renderInput(label, id, type = "text", value = "") {

  const lab = document.createElement("label")

  lab.setAttribute("for", id)

  lab.innerText = label

  const input = document.createElement("input")

  input.setAttribute("id", id)

  input.setAttribute("type", type)

  input.value = value

  return [lab, input]

}

function renderTextarea(label, id, className = "") {

  const lab = document.createElement("label")

  lab.setAttribute("for", id)

  lab.innerText = label

  const input = document.createElement("textarea")

  input.setAttribute("id", id)

  input.className = className

  return [lab, input]

}

function renderModal(path) {

  const modal = document.createElement("div")

  modal.className = "modal"

  const modalContent = document.createElement("modalContent")

  modalContent.className = "modal-content"

  modal.append(modalContent)

  const h = document.createElement("h3")

  h.innerText = "new file"

  modalContent.append(h)

  const form = document.createElement("div")

  form.className = "form"

  modalContent.append(form)

  form.append(...renderInput("Name file", "nameInput"))

  form.append(...renderInput("Size file (bites)", "sizeInput"))

  form.append(...renderTextarea("Content file", "contentInput", "file-content\_\_text"))

  const actions = document.createElement("div")

  actions.className = "form\_\_actions"

  form.append(actions)

  const back = document.createElement("button")

  back.innerText = "Cancel"

  back.addEventListener("click", (event) => {

    modals.innerHTML = ""

  })

  actions.append(back)

  const apply = document.createElement("button")

  apply.setAttribute("data-path", path)

  apply.innerText = "Create"

  apply.addEventListener("click", (event) => {

    const name = document.getElementById("nameInput").value

    const size = +document.getElementById("sizeInput").value

    const content = document.getElementById("contentInput").value

    if (name === "" || name === null) return

    if (size <= 0) return

    disk.createFile({name, content, size, type: 0x20, pathToDir: currentPath})

    modals.innerHTML = ""

    updateTree()

  })

  actions.append(apply)

  const modals = document.getElementById("modals")

  modals.innerHTML = ""

  modals.append(modal)

}

function renderDirectoryDialog(path) {

  const modal = document.createElement("div")

  modal.className = "modal"

  const modalContent = document.createElement("modalContent")

  modalContent.className = "modal-content"

  modal.append(modalContent)

  const h = document.createElement("h3")

  h.innerText = "New direction"

  modalContent.append(h)

  const form = document.createElement("div")

  form.className = "form"

  modalContent.append(form)

  form.append(...renderInput("Name direction", "dirNameInput"))

  const actions = document.createElement("div")

  actions.className = "form\_\_actions"

  form.append(actions)

  const back = document.createElement("button")

  back.innerText = "Cancel"

  back.addEventListener("click", (event) => {

    modals.innerHTML = ""

  })

  actions.append(back)

  const apply = document.createElement("button")

  apply.setAttribute("data-path", path)

  apply.innerText = "Create"

  apply.addEventListener("click", (event) => {

    const name = document.getElementById("dirNameInput").value

    const path = event.target.getAttribute("data-path")

    if (name === "" || name === null) return

    disk.createDir({name, pathToDir: path})

    modals.innerHTML = ""

    updateTree()

  })

  actions.append(apply)

  const modals = document.getElementById("modals")

  modals.innerHTML = ""

  modals.append(modal)

}

function renderList(files, path) {

  const fileTree = document.getElementById("file-tree")

  fileTree.innerHTML = ""

  const pathElem = document.createElement("span")

  pathElem.className = "file-tree\_\_path"

  pathElem.innerText = path

  fileTree.append(pathElem)

  const createButton = document.createElement("button")

  createButton.innerText = "New file"

  createButton.setAttribute("data-path", path)

  createButton.addEventListener("click", (event) => {

    renderModal(event.target.getAttribute("data-path"))

  })

  fileTree.append(createButton)

  const createDirButton = document.createElement("button")

  createDirButton.innerText = "New directory"

  createDirButton.setAttribute("data-path", path)

  createDirButton.addEventListener("click", (event) => {

    renderDirectoryDialog(event.target.getAttribute("data-path"))

  })

  fileTree.append(createDirButton)

  files.forEach(file => {

    const div = document.createElement('div')

    div.className = "file"

    const name = document.createElement('span')

    name.className = "file\_\_name"

    name.innerText = file.name

    div.append(name)

    if (file.type !== 0x10) {

      const date = document.createElement('span')

      date.className = "file\_\_wrt-time"

      date.innerText = formatDate(file.wrtTime)

      div.append(date)

      const size = document.createElement('span')

      size.className = "file\_\_size"

      size.innerText = file.size + "Б"

      div.append(size)

    }

    div.setAttribute("name", file.name)

    div.addEventListener("click", (event) => {

      const name = event.target.getAttribute("name") || event.target.parentElement.getAttribute("name")

      const file = disk.readFile(name, currentPath)

      if (file.type === 0x10) {

        currentPath = currentPath + file.name + "\\"

        const {files, path} = disk.parseFiles(currentPath)

        renderList(files, path)

      } else if (file.name === "..") {

        currentPath = "\\" + currentPath.split("\\").slice(1, -2).join("\\")

        if (currentPath[currentPath.length - 1] !== "\\") currentPath = currentPath + "\\"

        const {files, path} = disk.parseFiles(currentPath)

        renderList(files, path)

      } else {

        renderFileInfo(file, currentPath)

        renderFileClusters(file)

      }

    })

    fileTree.append(div)

  })

}

function renderFileInfo(file, dir) {

  function getInfoLine(label, val) {

    const line = document.createElement('div')

    line.className = "info"

    const nameLabel = document.createElement('span')

    nameLabel.className = "info-name"

    nameLabel.innerText = label

    const nameValue = document.createElement('span')

    nameValue.className = "info-value"

    nameValue.innerText = val

    line.append(nameLabel, nameValue)

    return line

  }

  function getContentLine(label, val) {

    const line = document.createElement('div')

    line.className = "file-content"

    const labelElem = document.createElement("span")

    labelElem.className = "file-content\_\_label"

    labelElem.innerText = label

    const valueElem = document.createElement("textarea")

    valueElem.className = "file-content\_\_text"

    valueElem.value = val

    valueElem.setAttribute("id", "textAreaInfo")

    line.append(labelElem, valueElem)

    return line

  }

  const fileInfo = document.getElementById("file-info")

  fileInfo.innerHTML = ""

  const name = renderInput("Имя:", "nameInfo", "text", file.name)

  const crtTime = getInfoLine("Time of create:", formatDate(file.crtTime))

  const wrtTime = getInfoLine("Last change:", formatDate(file.wrtTime))

  const firstCluster = getInfoLine("First cluster:", file.firstCluster)

  const size = getInfoLine("Size:", file.size)

  const type = getInfoLine("Type:", file.type)

  const content = getContentLine("Content:", file.content)

  const addButton = document.createElement("button")

  addButton.innerText = "Save changes"

  addButton.addEventListener("click", event => {

    const data = {}

    const nameInValue = document.getElementById("nameInfo").value

    if (file.name !== nameInValue) data.name = nameInValue

    const contentInValue = document.getElementById("textAreaInfo").value

    if (file.content !== contentInValue) data.content = contentInValue

    disk.updateFile(file.name, dir, data)

    updateTree()

    const nfile = disk.readFile(nameInValue, dir)

    renderFileInfo(nfile, dir)

    renderFileClusters(nfile)

  })

  const removeButton = document.createElement("button")

  removeButton.innerText = "Delete file"

  removeButton.addEventListener("click", event => {

    disk.removeFile(file.name, dir)

    updateTree()

    clearFileInfo()

    clearClustersInfo()

  })

  fileInfo.append(...name, crtTime, wrtTime, firstCluster, size, type, content, addButton, removeButton)

}

function clearFileInfo() {

  const fileInfo = document.getElementById("file-info")

  fileInfo.innerHTML = ""

}

function clearClustersInfo() {

  const clustersInfo = document.getElementById("clusters-info")

  clustersInfo.innerHTML = ""

}

function renderFileClusters(file) {

  function renderCluster(index, indexInChain) {

    const cluster = document.createElement('div')

    cluster.className =  indexInChain !== -1 ? "cluster cluster\_full" : "cluster"

    cluster.innerText = index

    if (indexInChain !== -1) {

      const span = document.createElement("span")

      span.className = "cluster\_\_index-in-chain"

      span.innerText = indexInChain

      cluster.append(span)

    }

    return cluster

  }

  const clustersInfo = document.getElementById("clusters-info")

  clustersInfo.innerHTML = ""

  const clustersGrid = document.createElement("div")

  clustersGrid.className = "clusters"

  for (let i = 0; i < disk.data.length; i++) {

    const cluster = renderCluster(i, file.clusters.indexOf(i))

    clustersGrid.append(cluster)

  }

  clustersInfo.append(clustersGrid)

}