Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

“Создание виртуальной файловой системы”

Студент ИУ7-76Б _____ Нгуен Ф. С.
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсового проекта _____ Рязанова Н. Ю.
(Подпись, дата) (И. О. Фамилия)

2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7
(Индекс)

_____

И.В.Рудаков
(И.О.Фамилия)

« ___ » _____ 2021 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине_____ Операционные системы_____

создание файловой системы
(Тема курсового проекта)

Студент_____ Нгуен Ф. С. гр. ИУ7-76Б_____
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к _4 нед., 50% к _7_ нед., 75% к _11нед., 100% к 14 нед.

1. Техническое задание

Спроектировать и разработать в операционной системе Linux файловую систему с операциями: Монтирование, Создание, Удаление, Переименование, ...

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.)__На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ____ » _____ 20__ г.

Руководитель курсового проекта _____ ____**Рязанова Н. Ю.**
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____**Нгуен Ф. С.**
(Подпись,дата) (И.О.Фамилия)

Оглавление

Введение

Один из ключей к успеху Linux - это его способность комфортно сосуществовать с другими системами. Вы можете прозрачно монтировать диски или разделы, на которых размещены форматы файлов, используемые Windows, другими системами Unix или даже системами с небольшой долей рынка, такими как Amiga. Linux удается поддерживать несколько типов дисков так же, как и другие варианты Unix, благодаря концепции, называемой виртуальной файловой системой.

Идея виртуальной файловой системы заключается в том, что внутренние объекты, представляющие файлы и файловые системы в памяти ядра, содержат широкий спектр информации; есть поле или функция для поддержки любой операции, предоставляемой любой реальной файловой системой, поддерживаемой Linux. Для каждой вызываемой функции чтения, записи или другой вызываемой функции ядро заменяет фактическую функцию, которая поддерживает собственную файловую систему Linux, файловую систему NT или любую другую файловую систему, в которой находится файл.

В этой работе обсуждаются цели, структура и реализация виртуальной файловой системы Linux. Он фокусируется на трех из пяти стандартных типов файлов Unix, а именно на обычных файлах, каталогах и символических ссылках.

1. Аналитическая часть

В данном разделе будет проанализирована поставленная задача и рассмотрены различные способы ее реализации.

1.1. Постановка задачи

Необходимо спроектировать и разработать веб-приложение популярного туристического направления с возможностью поиска (фильтра) по городом и по категориям, отметки любимых мест, оценки и комментарий собственного опыта.

В соответствии с заданием на курсовую работу по курсу ОС, необходимо создать загрузочный модуль (виртуальную файловую систему) со следующими возможностями:

Обычный файл: создать новый / удалить / переименовать / открыть / прочитать / записать файл.

Папка: Создать / Удалить / Переименовать / составлять список.

Ссылка: Создать / Удалить / Переименовать.

1.2. Загружаемые модули

Одной из важных особенностей ОС Linux является способность расширения функциональности ядра во время работы. Это означает, что вы можете добавить функциональность в ядро, а также и убрать её, когда система запущена и работает, без перезагрузки. Объект добавляющий дополнительный функционал в ядро, во время работы, называется модулем. Ядро Linux предлагает поддержку довольно большого числа типов (классов) модулей, включая, драйвера устройств. Каждый модуль является подготовленным объектным кодом, который может быть динамически подключен в работающее ядро командой «insmod» и отключен командой «rmmod».

Зарегистрировать файловую систему можно с помощью системного вызова **register_filesystem().** Регистрация файловой системы выполняется в функции инициализации модуля.

Для дерегистрации файловой системы используется функция **unregister_filesystem(),** которая вызывается в функции выхода загружаемого модуля.

Обе функции принимают как параметр указатель на структуру **file_system_type**, которая "описывает" создаваемую файловую систему. Эта структура описана в файле include / linux / fs.h.

Листинг 1. Описание структуры file_system_type

```
1. struct file_system_type {
2.     struct module *owner;
3.     const char *name;
4.     struct dentry *(*mount) (struct file system type *, int,
5.                 const char *, void *);
6.     void (*kill_sb) (struct super_block *);
7.     int fs_flags;
8.     struct file_system_type * next;
9.         }
```

Поле owner отвечает за счетчик ссылок на модуль, чтобы его нельзя было случайно выгрузить.

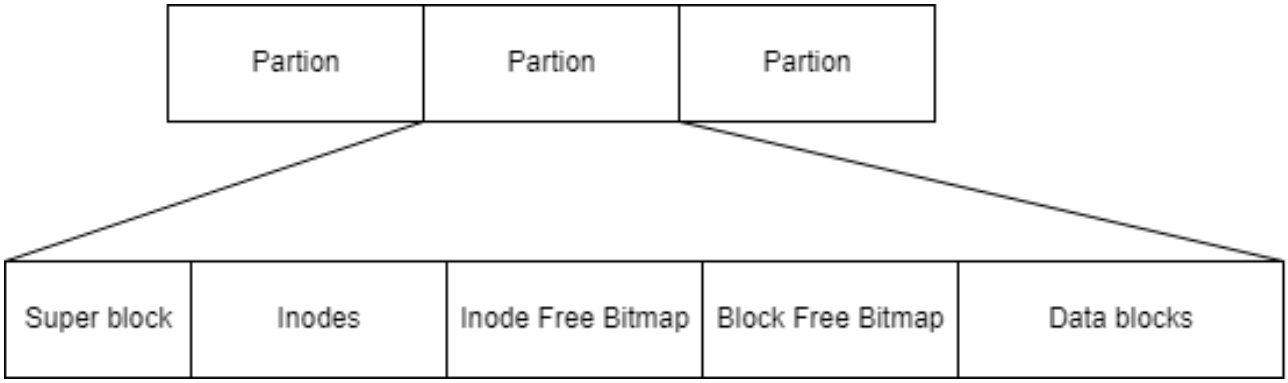
Поле name хранит название файловой системы. Именно это название будет использоваться при ее монтировании.

mount функция будет вызвана при монтировании файловой системы

kill_sb функция будет вызвана при размонтированиифайловой системы

1.3. **Распределение памяти:**

Каждый раздел (partition) представляет собой файловую систему. ФС содержит последовательность блоков. Каждый блок имеет размер 4 KiB



Суперблок - это первый блок раздела (блок 0). Он содержит метаданные раздела, такие как количество блоков, количество inodes, количество свободных inodes / блоков,...

Inode Store - Содержит все inodes раздела. Максимальное количество inodes равно количеству блоков раздела.

Два bitmaps будут использоваться для управления свободным пространством:

- inode free bitmap - для свободных inodes
- block free bitmap - для свободных блоков данных

1.4. **Объекты суперблока**

Эти структуры используются как «шлюз» к драйверам файловой системы, беря абстрактные идеи файловой системы и предоставляя ссылку на реализацию этих идей для каждой из файловых систем.

Листинг 2. Описание структуры super_block

```
1. struct super_block {
2.     unsigned long       s_blocksize;
3.     unsigned char       s_blocksize_bits;
4.     unsigned long       s_magic;
5.     const struct super_operations   *s_op;
6.     …
7. }
```

S_magic - магическое число, по которому драйвер файловой системы может проверить, что на диске хранится именно та самая файловая система, а не что-то еще или прочие данные;

s_blocksize : размер блока в байтах

s_blocksize_bits: размер блока в битах

s_op: указатель на структуру super_operations, которая содержит специальные методы связанные с суперблоком

Листинг 3. Описание структуры super_operations

```
1. struct super_operations {
2.     struct inode *(*alloc_inode)(struct super_block *sb);
3.     void (*destroy_inode)(struct inode *);
4.     int (*statfs) (struct dentry *, struct kstatfs *);
5.     void (*put_super) (struct super_block *);
6.     void (*write_inode) (struct inode *, int);
7.     int (*sync_fs) (struct super_block *);
8. }
```

1.5. **Объекты Inode**

Inode представлен структурой struct inode и операциями с ним, определенными в структуре **struct inode_operations**.

Вся информация, необходимая файловой системе для обработки файла, включается в структуру данных, называемую индексным дескриптором. Имя файла - это случайно назначенная метка, которую можно изменить, но индексный дескриптор уникален для файла и остается неизменным, пока файл существует.

Листинг 4. Описание структуры inode

```
1. struct inode {
2.     owner
3.     umode_t i_mode
4.     const struct inode_operations   *i_op;
5.     const struct file_operations    *i_fop;
6.     loff_t          i_size;
7.     struct timespec64   i_atime;
```

```
8.      struct timespec64   i_mtime;
9.      struct timespec64   i_ctime;
10.     void          *i_private;
11. }
```

I_mode : Тип файла и права доступа

I_size : Длина файла в байтах

I_atime : Время последнего доступа к файлу

I_mtime: Время последней записи файла

I_ctime: Время последнего изменения inode

I_op: Методы, связанные с объектом inode

I_fop: Методы, связанные с объектом file

Ядро Linux не может жестко запрограммировать конкретную функцию для обработки операции. Вместо этого он должен использовать указатель для каждой операции; указатель указывает на правильную функцию для конкретной файловой системы, к которой осуществляется доступ.

1.6. **Структура inode_operations**

Листинг 5. Описание структуры inode_operations

```
1.  struct inode_operations {
2.      struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
3.      int (*create) (struct inode *,struct dentry *, umode_t, bool);
4.      int (*link) (struct dentry *,struct inode *,struct dentry *);
5.      int (*unlink) (struct inode *,struct dentry *);
6.      int (*symlink) (struct inode *,struct dentry *, const char *);
7.      int (*mkdir) (struct inode *,struct dentry *, umode_t);
8.      int (*rmdir) (struct inode *,struct dentry *);
9.      int (*rename) (struct inode *, struct dentry *, struct inode *, struct dentry *,
    unsigned int);
```

Только что перечисленные методы доступны для всех возможных индексных дескрипторов и типов файловых систем. Тем не мение, только часть из них применима к любому данному inode и файловой системе; поля, соответствующие нереализованные методы устанавливаются в NULL. Нам надо реализовать пропущенные методы.

Create(dir, dentry, mode, excl): Создает новый inode связанного с объектом dentry в некотором каталоге.

Lookup(dir, dentry, flags): Ищет в каталоге индексный дескриптор, соответствующий имени файла, включенному в объект dentry.

Mkdir(dir, dentry, mode): Создает новый inode для каталога, связанного с объектом dentry в некотором каталоге.

Rmdir (dir, dentry): Удаляет из каталога подкаталог, имя которого включено в объект dentry.

Link(old_dentry, dir, new_dentry): Создает новую жесткую ссылку, которая ссылается на файл, указанный в old_dentry в каталоге dir; новая жесткая ссылка имеет имя, указанное в new_dentry.

Unlink(dir, dentry): Удаляет жесткую ссылку на файл, указанный объектом dentry, из каталога.

Symlink(dir, dentry, symname): Создает новый inode для символической ссылки, связанной с объектом dentry в некотором каталоге.

Rename(old_dir, old_dentry, new_dir, new_dentry, flags): Перемещает файл, идентифицированный old_entry, из каталога old_dir в каталог new_dir. Новое имя файла включается в объект dentry, на который указывает new_dentry.

1.7. **Объекты файлов**

Файловый объект описывает, как процесс взаимодействует с файлом, который он открыл. Объект создается при открытии файла и состоит из файловой структуры. Основная информация, хранящаяся в файловом объекте, - это указатель файла, то есть текущая позиция в файле, с которой будет выполняться следующая операция. Поскольку несколько процессов могут обращаться к одному и тому же файлу одновременно, указатель файла не может храниться в объекте inode.

Листинг 6. Описание структуры file

```
1.  struct file {
2.      struct path     f_path;
3.      struct inode *  inode
4.      struct file_operations*     f_ops;
5.      loff_t  f_pos;
6.      void *  private_data
7.      spinlock_t  f_lock;
8.      ….
9.  }
```

F_pos : Текущее смещение файла (указатель файла)

F_op: Указатель на таблицу операций с файлами

Листинг 7. Описание структуры file_operations

```
1.  struct file_operations {
2.      ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
3.      ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
4.      int (*open) (struct inode *, struct file *);
5.      int (*mmap) (struct file *, struct vm_area_struct *);
6.      int (*fsync) (struct file *, loff_t, loff_t, int datasync);
7.  }
```

Fsync - Функция предназначена для активации или деактивации уведомления об асинхронном вводе-выводе.

Llseek(file, offset, whence) - Обновляет указатель файла.

1.8. **Объект dentry**

Объект dentry создается ядром для каждого компонента пути, который ищет процесс; объект dentry связывает компонент с его соответствующим индексом. Например, при поиске пути /tmp/test ядро создает объект dentry для корневого каталога /, второй объект dentry для записи tmp корневого каталога и третий объект dentry для записи test в /tmp каталог.

Листинг 8. Описание структуры dentry

```
1.  Struct dentry {
2.      unsigned int d_flags;
3.      struct qstr d_name;
4.      struct inode *d_inode;
5.      struct list_head d_child;   /* child of parent list */
6.      struct list_head d_subdirs;
7.      struct dentry *d_parent;
8.  }
```

Объекты Dentry хранятся в кэше распределителя slab, называемом dentry_cache; Таким образом, объекты dentry создаются и уничтожаются вызовом kmem_cache_alloc () и kmem_cache_free ().

1.9. **Управление адресным пространством**

Перейдя inode→i_mapping, можно получить связанное address_space для региона. address_space содержит всю специфичную для файловой системы информацию, необходимую для выполнения страничных операций на диске.

```
1.  struct address_space {
2.          ...
3.          struct address_space_operations * a_ops;
4.          ...
5.  };
```

Периодически диспетчеру памяти нужно сбрасывать информацию на диск. Диспетчер памяти не знает и не заботится о том, как информация записывается на диск, поэтому для вызова соответствующих функций используется структура address_space_operations.

```
1.  struct address_space_operations {
2.       int (*writepage)(struct page * page, struct writeback_control wbc);
3.       int (*readpage)(struct file *file, struct page * page);
4.
```

```
5.      int (*write_begin)(struct file *file, struct address_space *mapping,
6.                  loff_t pos, unsigned len, unsigned flags,
7.                  struct page **pagep, void **fsdata);
8.      int (*write_end)(struct file *file, struct address_space *mapping,
9.                 loff_t pos, unsigned len, unsigned copied,
10.                struct page *page, void *fsdata);
11. };
```

Readpage - Вызывается page catch для чтения страницы с физического диска и отображения ее в памяти.

Writepage - Вызывается page catch для записи грязной страницы на физический диск (при вызове синхронизации или когда требуется память).

Write_begin - Вызывается VFS, когда для файла происходит системный вызов write() перед записью данных в кэш страницы. Эта функция проверяет, сможет ли запись завершиться, и выделяет необходимые блоки с помощью block_write_begin().

Write_end - Вызывается VFS после записи данных из системного вызова write() в кеш страницы. Эта функция обновляет метаданные inode и при необходимости усекает файл.

1.10. Другие структуры

Inode_Store Содержит все inode раздела (структура myfs_inode). Каждый inode (struct myfs_inode) содержит: стандартные данные, такие как размер файла и количество используемых блоков, а также специфичное для myfs поле объединения, содержащее dir_block и ei_block. Этот блок содержит:

- Для каталога (dir_block): список файлов в этом каталоге (struct myfs_file). блок описывается структурой myfs_dir_block;
- Для файла (ei_block): список экстентов, содержащих фактические данные этого файла. блок описывается структурой myfs_file_ei_block.

Структура myfs_file содержит информацию о файле, включая ino и имя файла, зная значение ino, мы можем найти соответствующий индекс (struct inode).

Экстент охватывает последовательные блоки, мы выделяем для него последовательные блоки диска одновременно. Он описывается структурой myfs_extent, которая содержит три члена:

- Ee_block: первый логический блок.
- Ee_len: количество блоков, покрываемых экстентом.
- Ee_start: первый физический блок.

2. Конструкторская часть

2.1. **Взаимодействие между процессами и объектами VFS**

Файл представлен файловой структурой данных в памяти ядра. Эта структура данных содержит поле с именем f_op, которое содержит указатели на функции, специфичные для файлов MS-DOS, включая функцию, которая читает файл.

Короче говоря, ядро отвечает за назначение правильного набора

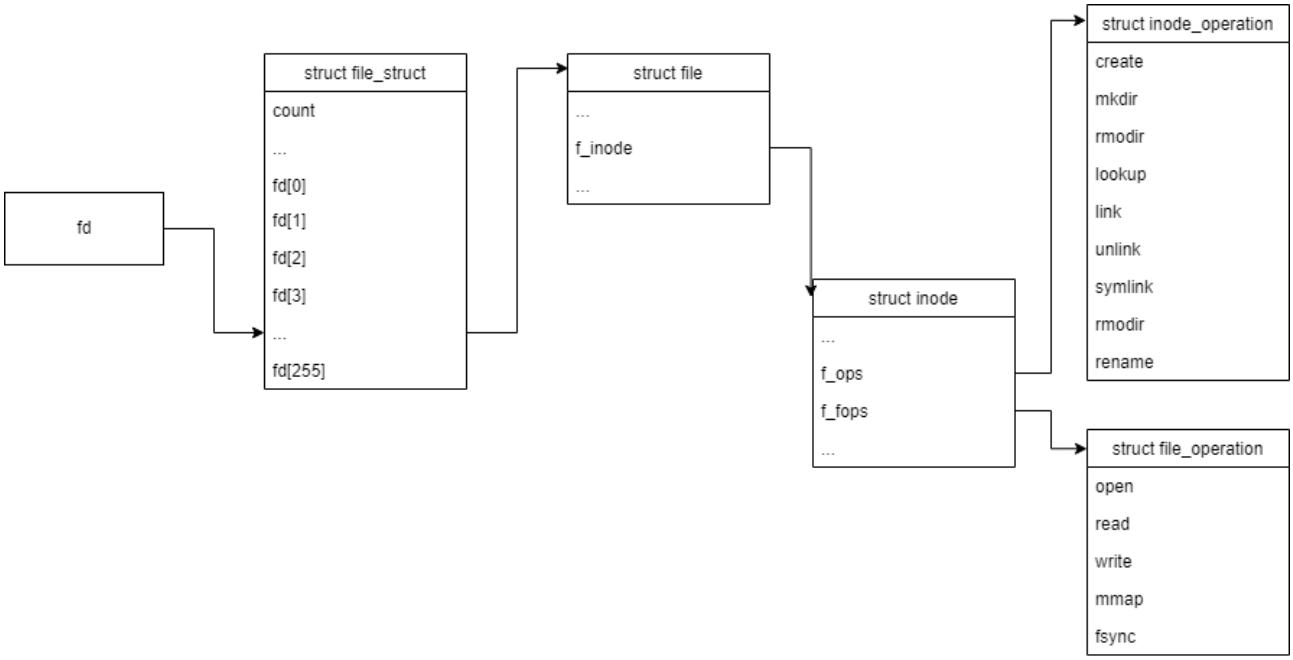


Рисунок 1. Взаимодействие между процессами и объектами VFS

2.2. **Некоторая дополнительная структура**

Листинг 9. Описание структуры myfs_inode

```
1.  struct myfs_inode {
2.      uint32_t i_mode;   /* File mode */
3.      uint32_t i_uid;    /* Owner id */
4.      uint32_t i_gid;    /* Group id */
5.      uint32_t i_size;   /* Size in bytes */
6.      uint32_t i_ctime;  /* Inode change time */
7.      uint32_t i_atime;  /* Access time */
8.      uint32_t i_mtime;  /* Modification time */
9.      uint32_t i_blocks; /* Block count */
10.     uint32_t i_nlink;  /* Hard links count */
11.     union {
12.         uint32_t ei_block;  /*Block with list of extents for this file */
13.         uint32_t dir_block; /*Block with list of files for this directory */
14.     };
15.     char i_data[32]; /* store symlink content */
16. };
```

Листинг 10. Описание структуры myfs_sb_info

```
1.  struct myfs_sb_info {
2.      uint32_t magic; /* Magic number */
3.
4.      uint32_t nr_blocks; /* Total number of blocks (incl sb & inodes) */
5.      uint32_t nr_inodes; /* Total number of inodes */
6.
7.      uint32_t nr_istore_blocks; /* Number of inode store blocks */
8.      uint32_t nr_ifree_blocks;  /* Number of inode free bitmap blocks */
9.      uint32_t nr_bfree_blocks;  /* Number of block free bitmap blocks */
10.
11.     uint32_t nr_free_inodes; /* Number of free inodes */
12.     uint32_t nr_free_blocks; /* Number of free blocks */
13. }
```

Листинг 11. Описание структуры myfs_inode_info

```
1.  struct myfs_inode_info {
2.      union {
3.          uint32_t ei_block;  /* Block with list of extents for this file */
4.          uint32_t dir_block; /* Block with list of files for this directory */
5.      };
6.      char i_data[32];
7.      struct inode vfs_inode;
8.  };
```

Листинг 12. Описание структуры myfs_dir_block

```
1.  struct myfs_dir_block {
2.      struct myfs_file {
3.          uint32_t inode;
4.          char filename[MYFS_FILENAME_LEN];
5.      } files[MYFS_MAX_SUBFILES];
6.  };
```

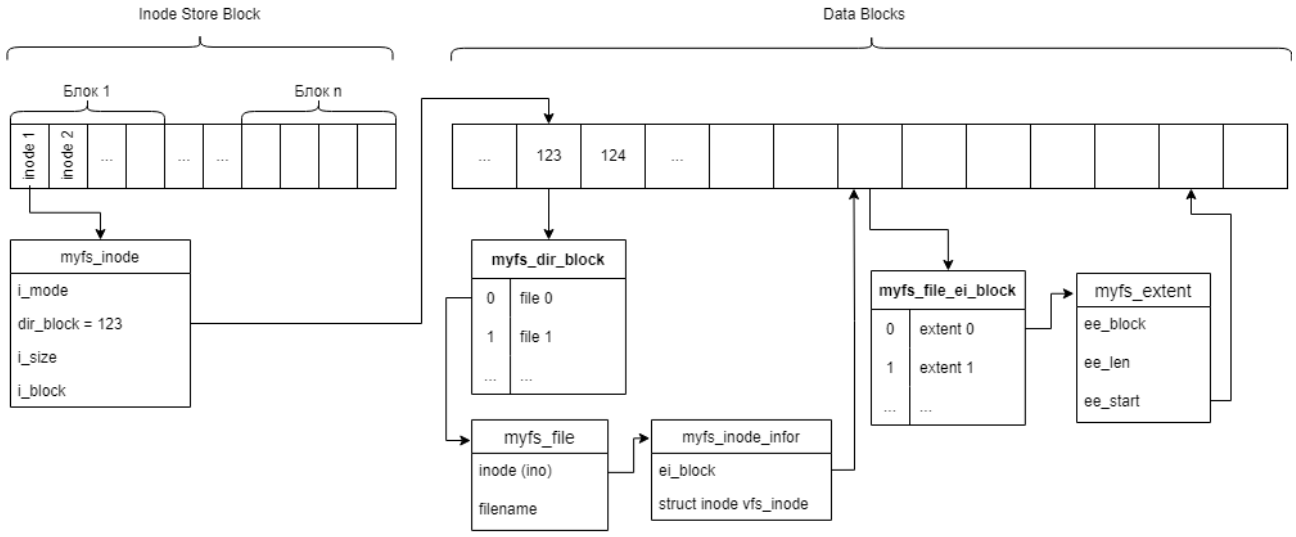


Рисунок 2. отношения между структурами

2.3. **Инициализация и установка суперблока**

Struct **file_system_type** - основная структура данных, описывающая файловую систему в ядре. **Mount** и **kill_sb** - 2 функции управления суперблоком.

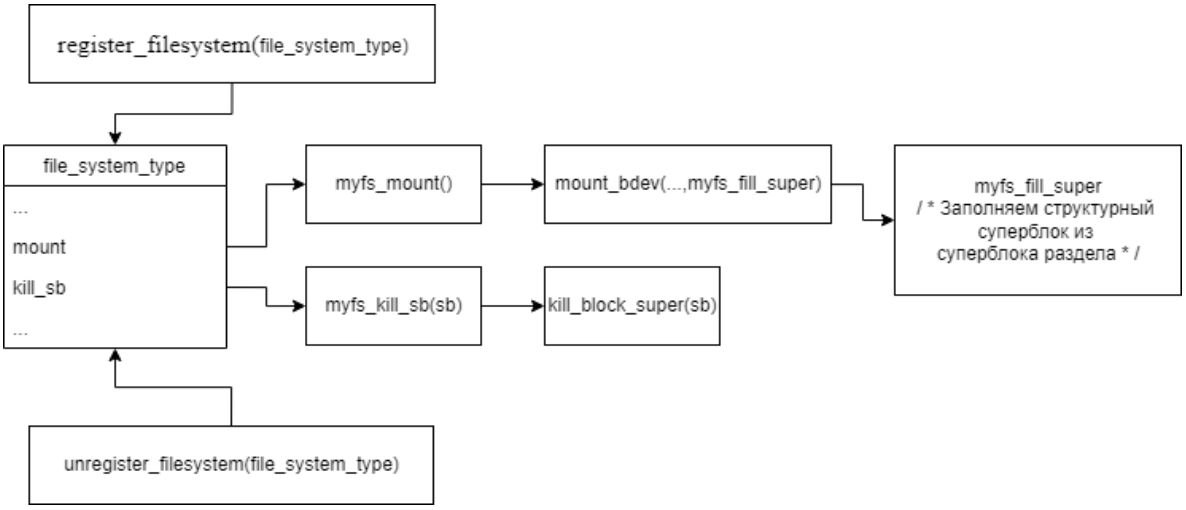


Рисунок 3. Инициализация и установка суперблока

Рисунок 4. алгоритм работы функции myfs_fill_super

2.4. **Методы, связанные с объектом inode**

Описание структуры inode_operations, используемой в программе:

Листинг 13. Описание структуры inode_operations

```
static const struct inode_operations myfs_inode_ops = {
    .lookup = myfs_lookup,
    .create = myfs_create,
    .unlink = myfs_unlink,
    .mkdir = myfs_mkdir,
    .rmdir = myfs_rmdir,
    .rename = myfs_rename,
    .link = myfs_link,
    .symlink = myfs_symlink,
};
```

В следующем абзаце приведено описание конкретных функций в этой структуре.

2.4.1. Lookup

Искать dentry в справочнике. заполнить dentry значением NULL, если он не находится в каталоге, или соответствующим индексным дескриптором, если он найден. В случае успеха возвращает NULL.

На рисунке показан алгоритм работы операции Lookup().

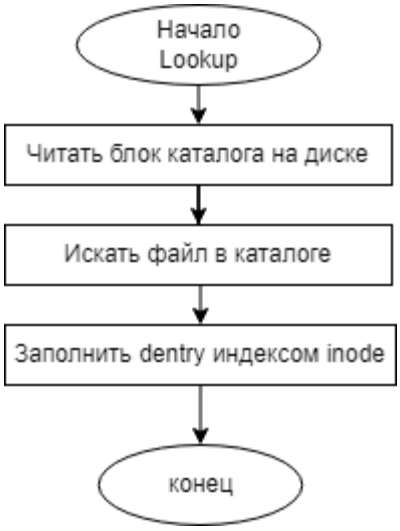


Рисунок 5. алгоритм работы операции Lookup()

2.4.2. Create

Create(dir, dentry, mode, excl): Создает новый inode связанного с объектом dentry в некотором каталоге.

На рисунке показан алгоритм работы операции create().

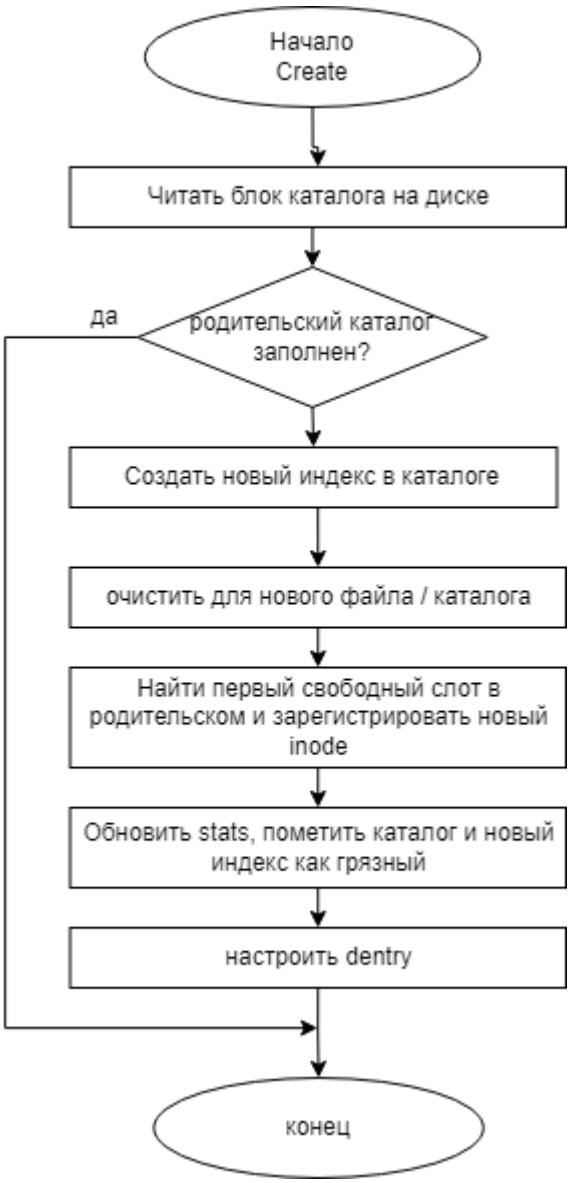


Рисунок 6. алгоритм работы операции create()

2.4.3. Mkdir

Mkdir(dir, dentry, mode): Создает новый inode для каталога, связанного с объектом dentry в некотором каталоге.

Создать каталог, вызвав функцию create с установкой флага S_IFDIR

2.4.4. Rmdir

Rmdir (dir, dentry): Удаляет из каталога подкаталог, имя которого включено в объект dentry.

На рисунке показан алгоритм работы операции rmdir().

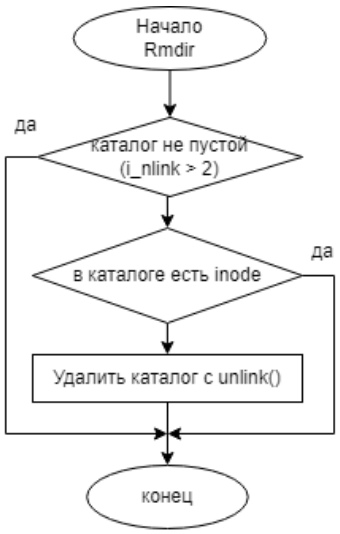


Рисунок 7. алгоритм работы операции rmdir()

2.4.5. Rename

Rename(old_dir, old_dentry, new_dir, new_dentry, flags): Перемещает файл, идентифицированный old_entry, из каталога old_dir в каталог new_dir. Новое имя файла включается в объект dentry, на который указывает new_dentry.

На рисунке показан алгоритм работы операции rename().

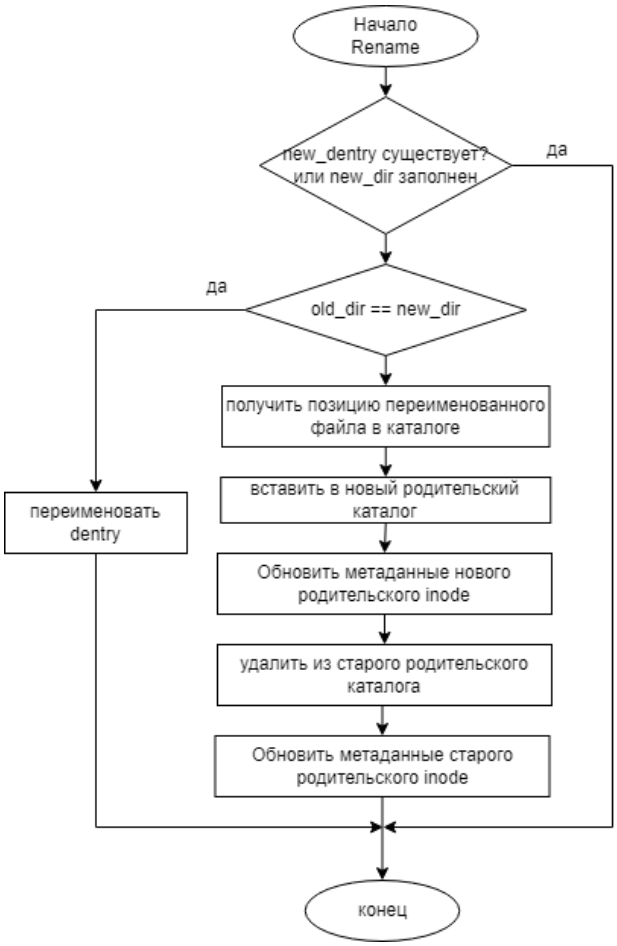


Рисунок 8. алгоритм работы операции rename()

2.4.6. Link

Link(old_dentry, dir, new_dentry): Создает новую жесткую ссылку, которая ссылается на файл, указанный в old_dentry в каталоге dir; новая жесткая ссылка имеет имя, указанное в new_dentry.

На рисунке показан алгоритм работы операции link()

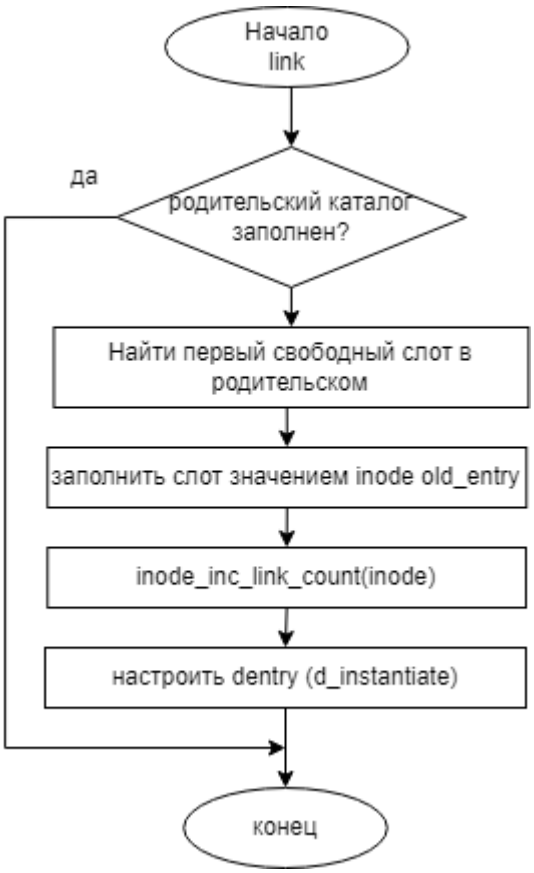


Рисунок 9. алгоритм работы операции link()

2.4.7. Unlink

Unlink(dir, dentry): Удаляет жесткую ссылку на файл, указанный объектом dentry, из каталога.

На рисунке показан алгоритм работы операции unlink()

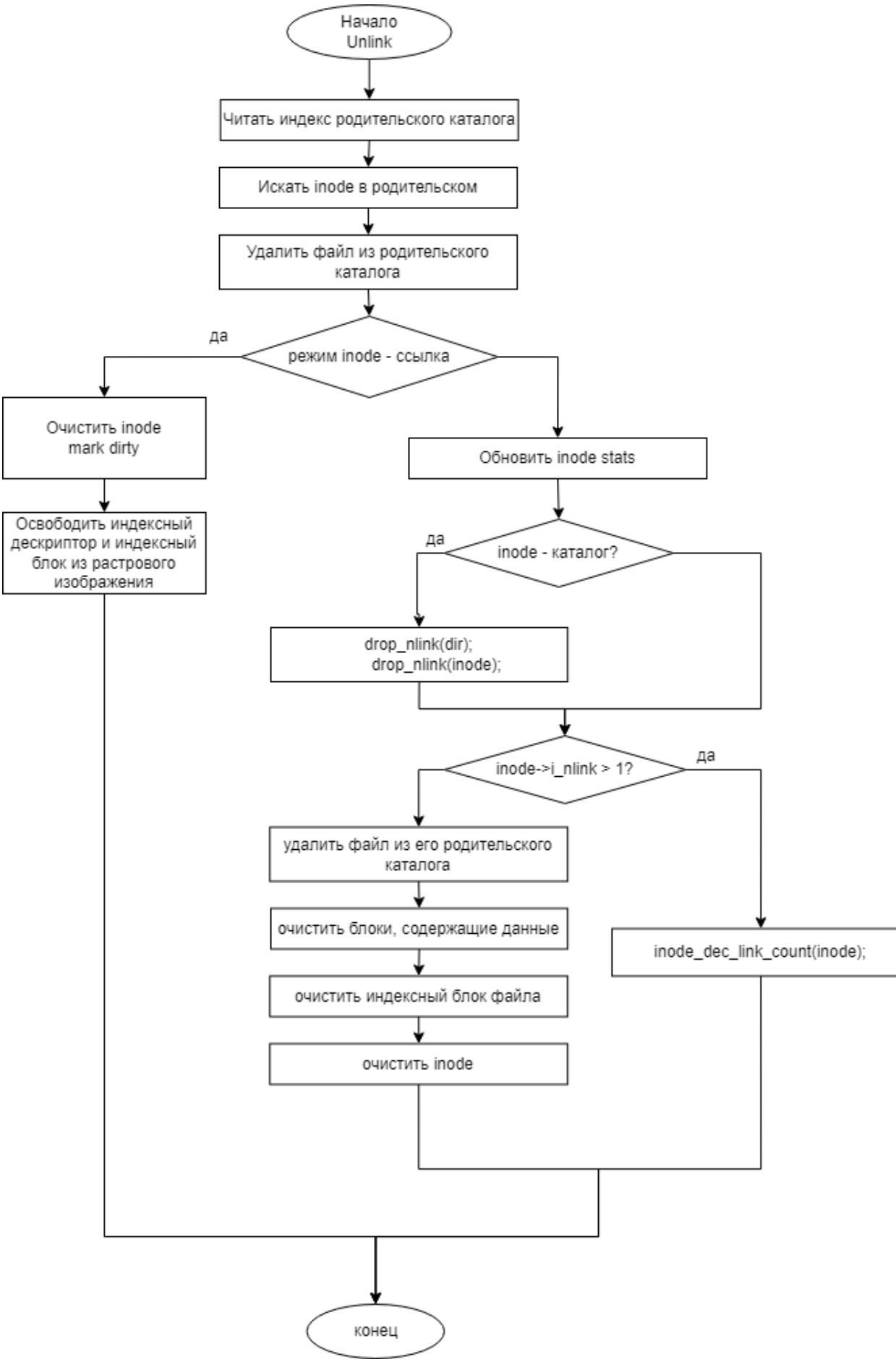


Рисунок 10. алгоритм работы операции unlink()

2.4.8. Symlink

Symlink(dir, dentry, symname): Создает новый inode для символической ссылки, связанной с объектом dentry в некотором каталоге.

на рисунке показан алгоритм работы операции symlink()

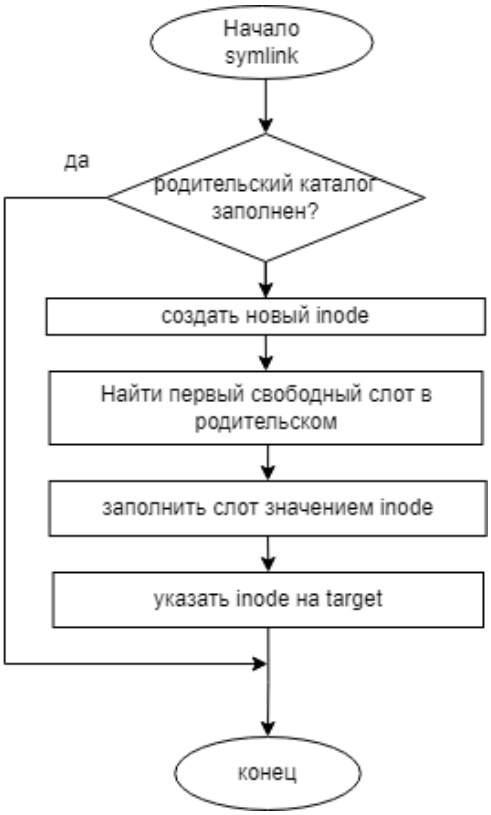


Рисунок 11. алгоритм работы операции symlink()

3. Технологическая часть

В данном разделе производится выбор средств для разработки и рассматривается реализация программного обеспечения.

3.1. **Выбор языка программирования**

В качестве языка программирования был выбран язык C. На этом языке реализованы все модули ядра и драйверы операционной системы Linux. Компилятор -- gcc.

3.2. **Исходный код программы**

Листинг 14. Инициализация модуля

```
static int __init myfs_init(void)
{
    int ret = myfs_init_inode_cache();
    if (ret) {
        pr_err("inode cache creation failed\n");
        goto end;
    }

    ret = register_filesystem(&myfs_file_system_type);
    if (ret) {
        pr_err("register_filesystem() failed\n");
        goto end;
    }

    pr_info("module loaded\n");
end:
    return ret;
}
```

Листинг 15. Выход из модуля

```
static void __exit myfs_exit(void)
{
    int ret = unregister_filesystem(&myfs_file_system_type);
    if (ret)
        pr_err("unregister_filesystem() failed\n");

    myfs_destroy_inode_cache();

    pr_info("module unloaded\n");
}
```

Листинг 16. Описание структуры file_system_type

```
struct dentry *myfs_mount(struct file_system_type *fs_type,
                          int flags,
                          const char *dev_name,
                          void *data)
{
    struct dentry *dentry =
        mount_bdev(fs_type, flags, dev_name, data, myfs_fill_super);
    if (IS_ERR(dentry))
        pr_err("'%s' mount failure\n", dev_name);
    else
        pr_info("'%s' mount success\n", dev_name);

    return dentry;
}

/* Unmount a myfs partition */
void myfs_kill_sb(struct super_block *sb)
{
    kill_block_super(sb);

    pr_info("unmounted disk\n");
}

static struct file_system_type myfs_file_system_type = {
    .owner = THIS_MODULE,
    .name = "myfs",
    .mount = myfs_mount,
    .kill_sb = myfs_kill_sb,
    .fs_flags = FS_REQUIRES_DEV,
    .next = NULL,
}
```

Листинг 17. Описание структуры inode_operations

```
static const struct inode_operations myfs_inode_ops = {
    .lookup = myfs_lookup,
    .create = myfs_create,
    .unlink = myfs_unlink,
    .mkdir = myfs_mkdir,
    .rmdir = myfs_rmdir,
    .rename = myfs_rename,
    .link = myfs_link,
    .symlink = myfs_symlink,
};

static struct dentry *myfs_lookup(struct inode *dir, struct dentry *dentry,
                                                        unsigned int flags);

static int myfs_create(struct inode *dir, struct dentry *dentry, umode_t mode, bool excl);

static int myfs_unlink(struct inode *dir, struct dentry *dentry);

static int myfs_mkdir(struct inode *dir, struct dentry *dentry, umode_t mode);

static int myfs_rmdir(struct inode *dir, struct dentry *dentry);

static int myfs_link(struct dentry *old_dentry,struct inode *dir, struct dentry *dentry);

static int myfs_symlink(struct inode *dir, struct dentry *dentry, const char *symname);
```

Листинг 18. . Описание структуры address_space_operations

```
const struct address_space_operations myfs_aops = {
    .readpage = myfs_readpage,
    .writepage = myfs_writepage,
    .write_begin = myfs_write_begin,
    .write_end = myfs_write_end,
};

static int myfs_readpage(struct file *file, struct page *page)
{
    return mpage_readpage(page, myfs_file_get_block);
}

static int myfs_writepage(struct page *page, struct writeback_control *wbc)
{
    return block_write_full_page(page, myfs_file_get_block, wbc);
}

static int myfs_write_begin(struct file *file, struct address_space *mapping, loff_t pos,
                            unsigned int len, unsigned int flags, struct page **pagep,
                            void **fsdata);

static int myfs_write_end(struct file *file, struct address_space *mapping, loff_t pos,
                          unsigned int len, unsigned int copied, struct page *page,
                              void *fsdata);
```

4. Экспериментальная часть

4.1. **Условия эксперимента**

Исследование результатов выполнения программы производилось при следующем аппаратном обеспечении, выделенном виртуальной машине:

- Процессор Intel(R) Core(TM) i5-7200U CPU @ 2.50ghz 2.70 ghz
- Оперативная память 4GB
- OS Linux Ubuntu 20.4

4.2. **Загрузка модуля и интерфейс программы**

Sudo insmod myfs.ko

Mkdir -p myfs-Root

$ dd if=/dev/zero of=myfs.img bs=1M count=50

Sudo mount -o loop -t myfs myfs.img myfs-Root

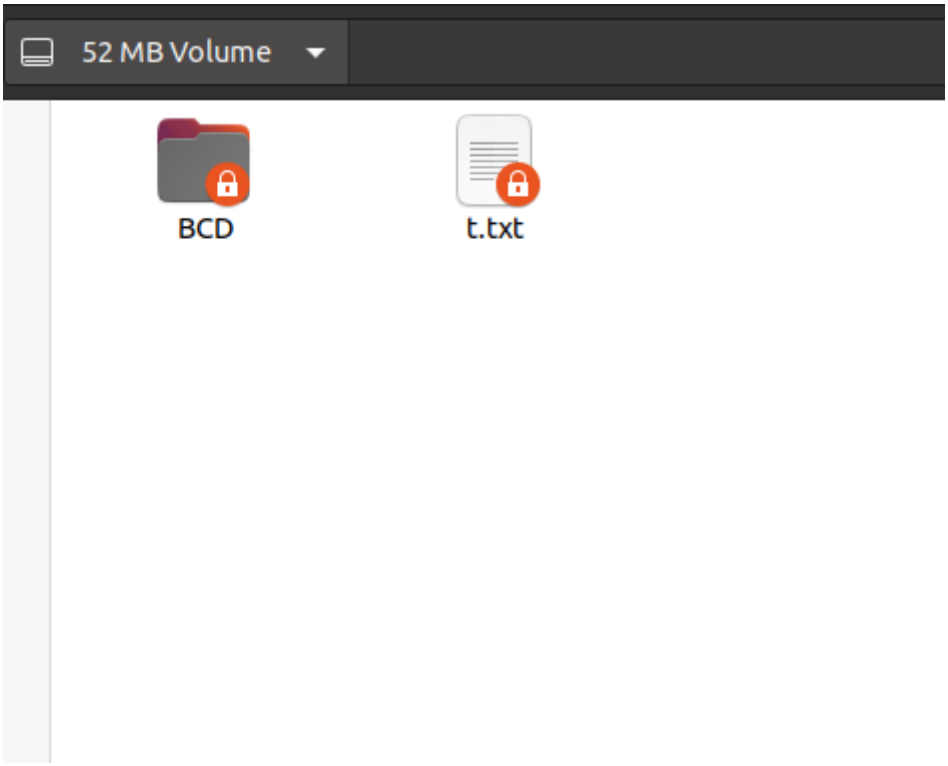


Рисунок 12. интерфейс программы

4.3. **Результат работы программы**

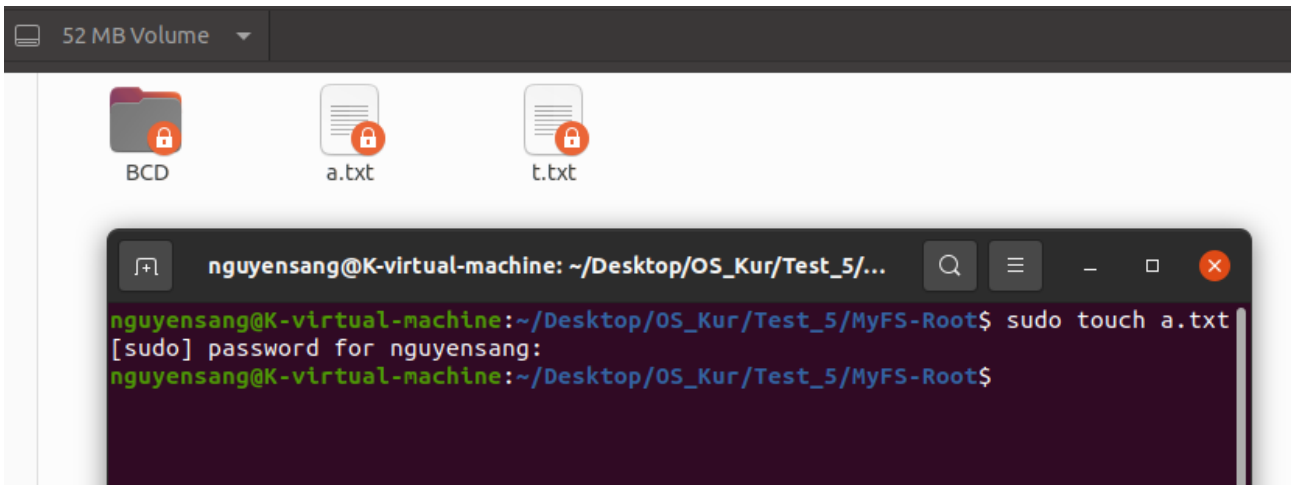


Рисунок 13. Работа операции create()

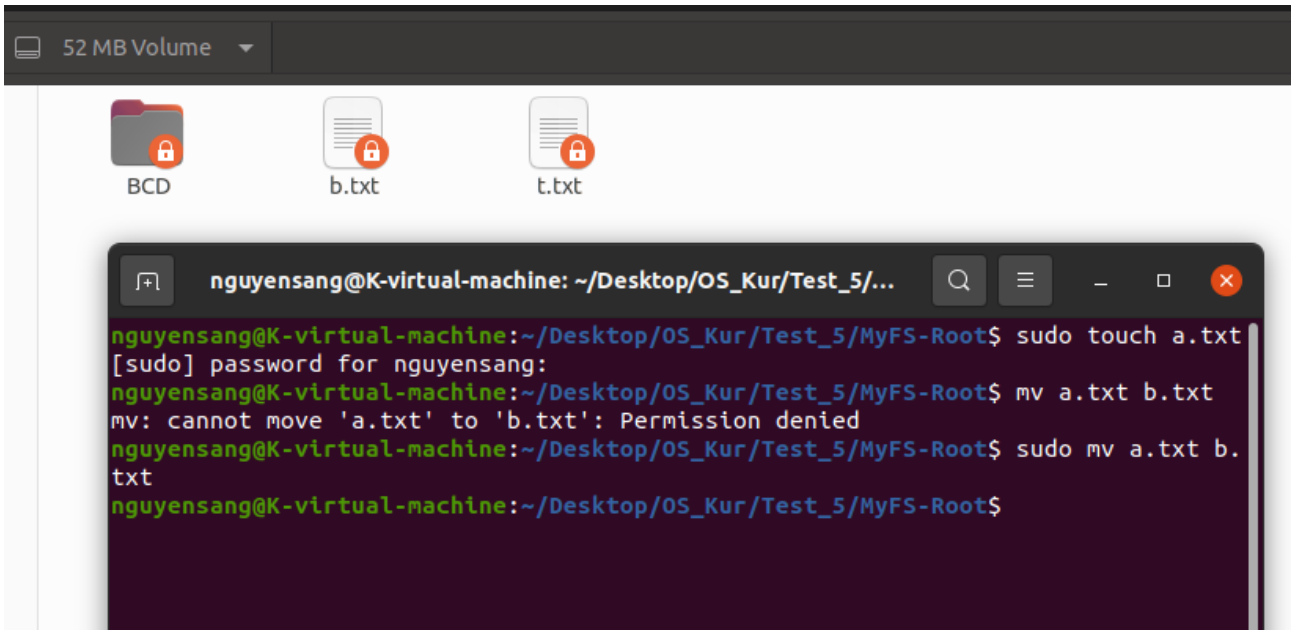


Рисунок 14. Работа операции rename

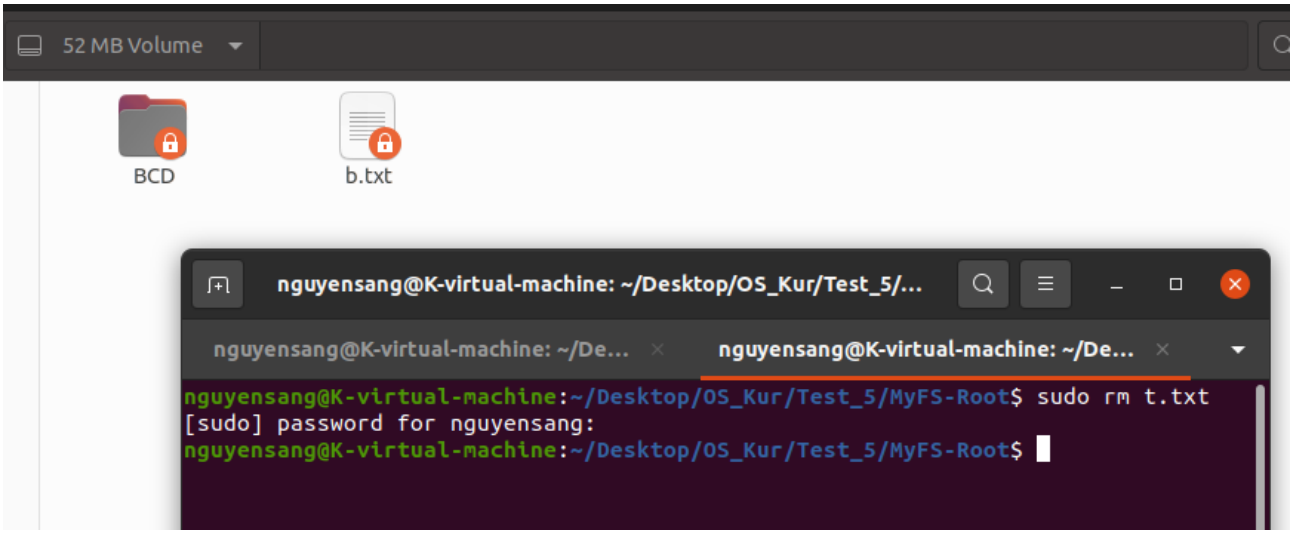


Рисунок 15. Работы операции rmdir()

4.4. **Выгрузка модуля**

Unount –t myfs MyFS-Root

sudo rmmod myfs

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта был изучен метод создания виртуальной файловой системы, работа с ядром и ядерными функциями.

Разработана программа , в соответствии с техническим заданием. Разработанный программный продукт удовлетворяет поставленной задаче.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Overview of the Linux Virtual File System *[https://www.kernel.org/doc/html/latest/filesystems/vfs.html]*
2. конспект лекций по курсу "Операционные системы"
3. The Linux Kernel's VFS Layer *[https://www.usenix.org/legacy/publications/library/proceedings/usenix01/full_papers/kroeger/kroeger_html/node8.html]*
4. Using the page cache *[https://cs4118.github.io/pantryfs/page-cache-overview.pdf]*
5. Linux File System: Virtual File System (VFS) *[https://emmanuelbashorun.medium.com/linux-file-system-virtual-file-system-vfs-layer-part-3-79235c40a499]*
6. Виртуальная файловая система
7. *[http://www.cs.vsu.ru/~svv/ux/lecture%205.pdf]*
8. How to write a Linux VFS filesystem module *[http://pages.cpsc.ucalgary.ca/~crwth/programming/VFS/VFS.php]*
9. Understanding the Linux Kernel, Daniel P. Bovet, Marco Cesati *[https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch12s02.html#:~:text=Each%20VFS%20object%20is%20stored,specialized%20behavior%20for%20the%20object.]*
10. Linux VFS *[https://titanwolf.org/]*

Приложение

Makefile

```
obj-m += simplefs.o
simplefs-objs := fs.o super.o inode.o file.o dir.o extent.o

KDIR ?= /lib/modules/$(shell uname -r)/build

MKFS = mkfs.myfs

all: $(MKFS)
    make -C $(KDIR) M=$(PWD) modules

IMAGE ?= myfs.img
IMAGESIZE ?= 50

$(MKFS): mkfs.c
    $(CC) -std=gnu99 -Wall -o $@ $<

$(IMAGE): $(MKFS)
    dd if=/dev/zero of=${IMAGE} bs=1M count=${IMAGESIZE}
    ./$< $(IMAGE)

check: all
    script/test.sh $(IMAGE) $(IMAGESIZE) $(MKFS)

clean:
    make -C $(KDIR) M=$(PWD) clean
    rm -f *~ $(PWD)/*.ur-safe
    rm -f $(MKFS) $(IMAGE)

.PHONY: all clean
```