Лабораторная работа 5. Файловые системы

Цель работы – реализация доступа к линейному пространству данных как к иерархической файловой системе.

Данная работа предполагает два варианта исполнения:

а) создание программного блока, обеспечивающего работу с файловой системой, хранящейся в файле;

б) создание модулей для ОС UNIX, обеспечивающих подключение к иерархии файловой системы раздела с файловой системой, хранящегося в файле.

Постановка задачи для конкретного студента включает выбор варианта исполнения и параметры файловой системы.

В ходе выполнения лабораторной работы студент должен решить следующие задачи.

1. Изучить архитектуру драйвера файловой системы и его место в подсистеме управления файлами.

2. Определить обрабатываемые драйвером запросы и используемые функции доступа к линейному пространству данных.

3. Изучить предложенный тип файловой системы.

4. Реализовать модуль драйвера файловой системы.

5. Реализовать модуль инициализации файловой системы.

6. Выполнить компиляцию и сборку модулей.

7. Провести сравнительные эксперименты работы с файловыми системами различных типов с использованием соответствующих модулей.

Предлагаемые к реализации файловые системы

Во всех рассматриваемых файловых системах место на диске выделяется блоками. Размер блока, как правило, совпадает с аппаратным размером сектора (512 байт у большинства дисковых устройств), однако многие файловые системы могут использовать логические блоки, состоящие из нескольких секторов (так называемые кластеры).

Использование блоков и кластеров вместо адресации с точностью до байта обусловлено двумя причинами. Во-первых, у большинства устройств произвольного доступа доступ произволен лишь с точностью до сектора, т.е. нельзя произвольно считать или записать любой байт – нужно считывать или записывать весь сектор целиком. Во вторых, такой способ позволяет значительно расширить адресуемое пространство.

Файлы, хранящиеся в файловой системе можно разделить на два типа – пользовательские файлы и системные файлы (к которым относятся, например, директории).

Файловая система 1

Наиболее простой файловой системой можно считать структуру представленную на рисунке.

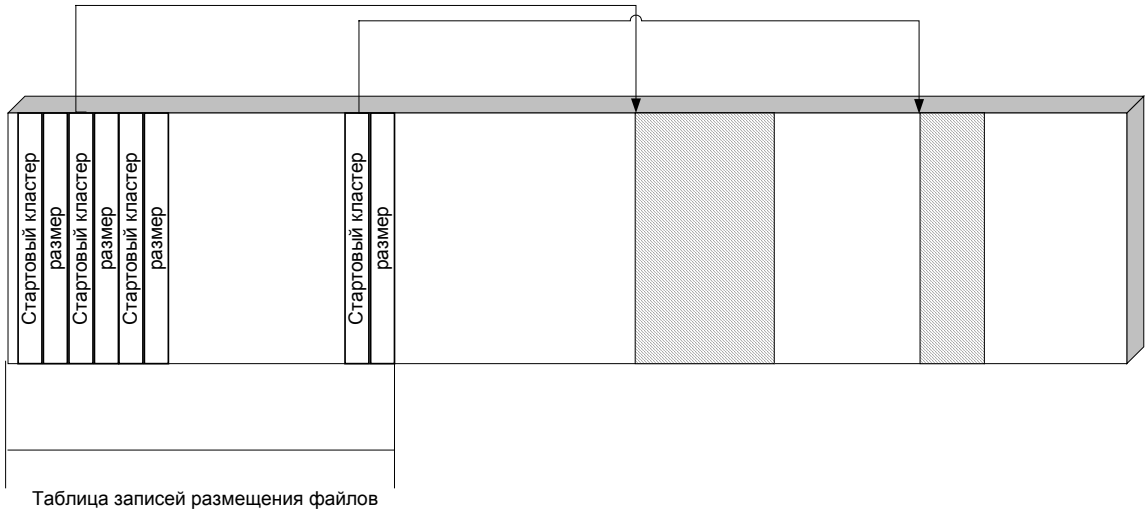


Рис. 58 Файловая система 1

В начале диска располагается таблица записей размещения файлов. Количество записей таблицы задается при форматировании файловой системы и не может быть изменено. Количество записей в таблице определяет число файлов, которые могут быть созданы. Каждому файлу выделяется непрерывная область на диске. Благодаря этому в записи таблицы, соответствующей файлу, достаточно хранить адрес первого кластера файла и его длину, также измеренную в кластерах.

Нулевая и первая запись в таблице зарезервированы. Нулевая запись представляет собой файл корневого каталога. Первая – файл, содержащий список элементов таблицы ссылающихся на свободные участки диска.

Каталог (директория) является файлом специального вида. Он содержит записи описывающие файлы входящие в каталог. Каждая запись представляет собой 32-байтовую структуру вида:

байт	0-7	имя файла
	8-10	расширение файла
	11	атрибут файла
	12-21	зарезервировано
	22-23	время последнего доступа к файлу
	24-25	дата последнего доступа к файлу
	26-27	индекс в таблице записей размещения файлов
	28-31	размер файла

Атрибут файла в частности определяет является ли файл подкаталогом (если младший бит = 1, то это подкаталог). Размер файла указывает точную длину файла в байтах (поскольку файл обычно не целиком занимает отведенный ему последний кластер).

Каталог свободных элементов имеет такую же структуру, однако в ней используется только индекс в таблице записей размещения файлов. В случае если свободного места нет, то элемент таблицы размещения файлов с индексом один будет содержать ноль в поле размер (каталог свободных элементов пуст).

Файловая система 2

Для работы этой файловой системы используется таблица, обычно называемая FAT (File Allocation Table, таблица размещения файлов), расположенная в начале диска. Таблица создается при форматировании файловой системы. В этой таблице каждому блоку (кластеру), предназначенному для хранения данных, соответствует 16-битовое значение. Если блок свободен, то значение будет нулевым. Если же блок принадлежит файлу, то значение равно индексу следующего блока этого файла в таблице размещения файлов. Если это последний блок в файле, то значение – FFFF. Запись с индексом ноль является первой записью файла, содержащего корневую директорию.

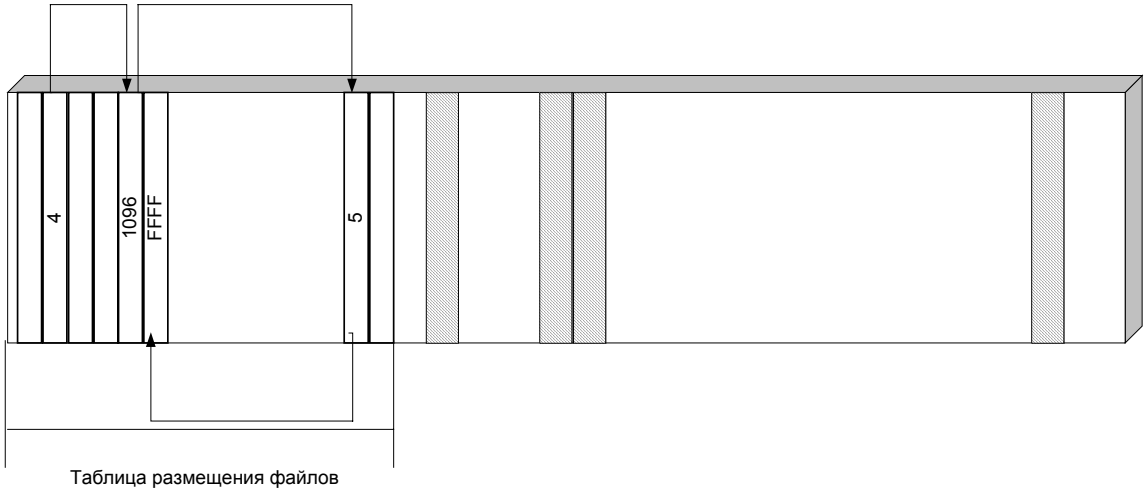


Рис. 59 Файловая система 2

Каталог (директория) является файлом специального вида. Он содержит записи описывающие файлы входящие в каталог. Каждая запись представляет собой 32-байтовую структуру вида:

байт	0-7	имя файла
	8-10	расширение файла
	11	атрибут файла
	12-21	зарезервировано
	22-23	время последнего доступа к файлу
	24-25	дата последнего доступа к файлу
	26-27	индекс первого кластера файла в таблице размещения файлов
	28-31	размер файла

Атрибут файла в частности определяет является ли файл подкаталогом (если младший бит = 1, то это подкаталог). Размер файла указывает точную длину файла в байтах (поскольку файл обычно не целиком занимает отведенный ему последний кластер).

Файловая система 3

Предполагается, что пространство разбито на участки равного размера (кластеры). Файл может занимать несколько несмежных областей. Области, занимаемые файлом, однозначно определяются *файловой записью*. Файловая запись представляет структуру следующего вида:

байт		экстент1
		экстент2
		экстент16

Каждый экстент представляет собой запись из двух полей – первое поле – адрес начала области (номер кластера), которую занимает файл, второе поле – размер этой области в кластерах. Файловые записи собраны в таблицу, размещаемую в начале диска. Размер таблицы задается при форматировании файловой системы и не изменяется.

В случае, если шестнадцати областей оказывается недостаточно для размещения файла, то из экстента может быть организована ссылка на файловую запись, содержащую "продолжение" файла. Для этого поле размер устанавливается в 0, при этом значение в первом поле трактуется как индекс в таблице файловых записей. Таким образом, массивы экстентов, описывающих размещение файлов, организованы в виде дерева.

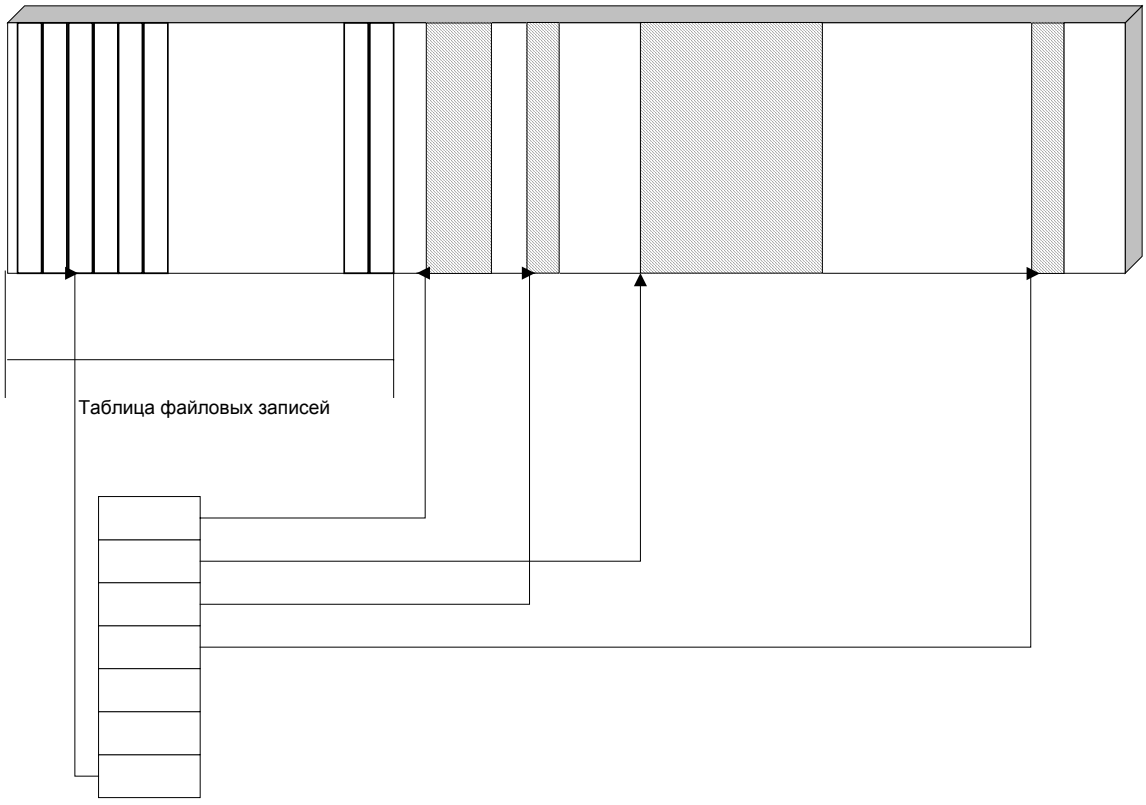


Рис. 60 Файловая система 3

Каталог (директория) является файлом специального вида. Он содержит записи, описывающие файлы, входящие в каталог. Каждая запись представляет собой 32-байтовую структуру вида:

байт	0-7	имя файла
	8-10	расширение файла
	11	атрибут файла
	12-21	зарезервировано

22-23	время последнего доступа к файлу
24-25	дата последнего доступа к файлу
26-27	индекс первой файловой записи в таблице файловых записей
28-31	размер файла

Атрибут файла в частности определяет является ли файл подкаталогом (если младший бит = 1, то это подкаталог). Размер файла указывает точную длину файла в байтах (поскольку файл обычно не целиком занимает отведенный ему последний кластер).

Файловая запись с индексом ноль содержит корневой каталог.

Симулятор работы с файловой системой

Постановка задачи

В рамках лабораторной работы ставится задача создания файловой системы, поддерживающей следующие операции:

Операции над файлами

- Create. Создание файла, не содержащего данных. Смысл данного вызова - объявить, что файл существует и присвоить ему ряд атрибутов по умолчанию.

- Delete. Удаление файла и освобождение занятого им дискового пространства.

- Open. Перед использованием файла процесс должен его открыть. Цель данного системного вызова разрешить системе проанализировать атрибуты файла и проверить права доступа к файлу, а также считать в оперативную память список адресов блоков файла для быстрого доступа к его данным.

- Close. Если работа с файлом завершена, его атрибуты и адреса блоков на диске больше не нужны. В этом случае файл нужно закрыть, чтобы освободить место во внутренних таблицах файловой системы.

- Seek. Дает возможность специфицировать место внутри файла, откуда будет производиться считывание (или запись) данных, то есть задать текущую позицию.

- Read. Чтение данных из файла. Обычно это происходит с текущей позиции. Пользователь должен задать объем считываемых данных и предоставить буфер для них.

- Write. Запись данных в файл с текущей позиции. Если текущая позиция находится в конце файла, его размер увеличивается, в противном случае запись осуществляется на место имеющихся данных, которые, таким образом, теряются.

- Get attributes. Предоставляет процессам нужные им сведения об атрибутах файла

- Set attributes. Дает возможность пользователю установить некоторые атрибуты. Наиболее очевидный пример - установка режима доступа к файлу.

- Rename. Данная операция может быть смоделирована копированием данного файла в файл с новым именем и последующим его удалением.

Операции над директориями

- Create. Создание директории.

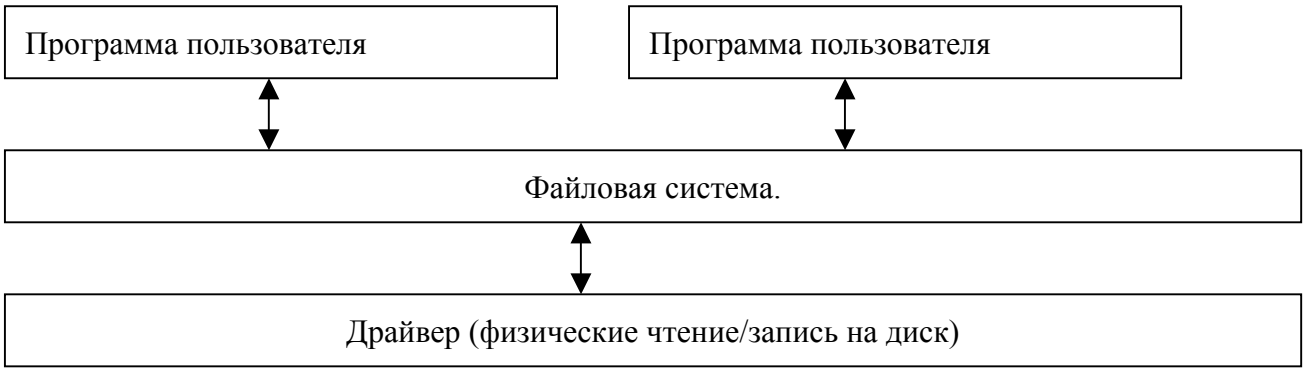
- Delete. Удаление директории. Удалена может быть только пустая директория.

- Files. Возвращает список файлов (и директорий), содержащихся в данной директории

- Rename. Имена директорий можно менять, также как и имена файлов.

Требования к лабораторной работе

Лабораторная работа должна быть выполнена с использованием программной лаборатории «Симулятор файловых систем». Программная лаборатория представляет собой набор подпрограмм и классов, реализованных на языке C++. Если рассмотреть пример типичного использования файловой системы, то можно выделить три уровня модулей, участвующих во взаимодействии



Целью лабораторной работы является реализация второго уровня – файловой системы, в то время как первый уровень (программы пользователя) и третий уровень («драйвер») поставляются в рамках программной лаборатории.

Пользовательские программы, включенные в программную лабораторию можно разделить на два типа – к первому относятся тесты файловой системы, каждый из которых проверяет тот или иной аспект функционирования файловой системы. К другому типу относится броузер файловой системы, который позволяет просмотреть структуру директорий. Тесты представляют собой консольные приложения. Броузер реализован в виде оконного приложения, представляющего структуру директорий в виде дерева. Сбор статистики по основным операциям (количество позиционирований, объем считанных/записанных байтов) производится «драйвером».

Таким образом, задача слушателя состоит в том, чтобы реализовать собственный класс, реализующий функциональность файловой системы определенного типа. Класс файловой системы должен быть потомком абстрактного класса VFS (`class VFS`). Описание класса предоставлено в сопровождающей документации.

Работа с программной лабораторией может осуществляться по следующему сценарию:

1. Получение задания на разработку файловой системы того или иного типа.

2. Реализация и оптимизация класса, реализующего функциональность файловой системы. Класс – потомок абстрактного класса VFS.Устранение ошибок.

3. Компиляция проекта, устранение ошибок.

4. Компиляция тестов.

5. Запуск тестов. В случае отрицательного результата теста – возврат к шагу 2.

6. Изучение файла статистики дисковых операций по результатам тестов. Сравнение с эталонными результатами.

7. Компиляция броузера реализованной файловой системы.

Архитектура программной лаборатории

Программная лаборатория предоставляет набор классов и подпрограмм, предназначенных для использования слушателем при создании симулятора файловой системы. Кроме того, она включает в себя средства тестирования и изучения файловых систем, реализованных слушателями.

Функционально программная лаборатория состоит из следующих логических блоков:

1) Блок визуализации – позволяет просмотреть структуру директорий предоставленной файловой системы.
2) Блок симуляции– реализуется слушателем. Представляет собой модуль, реализующий стандартные операции файловой системы.
3) Блок тестирования – включает в себя набор тестов для проверки правильности работы предоставленного модуля файловой системы
4) Блок статистики – отвечает за накопление результатов тестов.

Сценарий работы с программной лабораторией.

Программная лаборатория поставляется в виде структуры директорий, содержащих исходные тесты.

FSLab

|_____lab_viewer // Содержит код приложения, визуализирующего структуру директорий

|_____Tests // Содержит набор тестов

|_____FS // Содержит набор абстрактных классов FS, модуль работы с диском и модуль,

| //собирающий статистику.

|_____Docs // Некоторая документация

Задача слушателя состоит в том, чтобы реализовать собственный класс, реализующий функциональность файловой системы определенного типа. После чего осуществляется перекомпиляция исходных кодов программной лаборатории вместе с реализованным слушателем модулем.

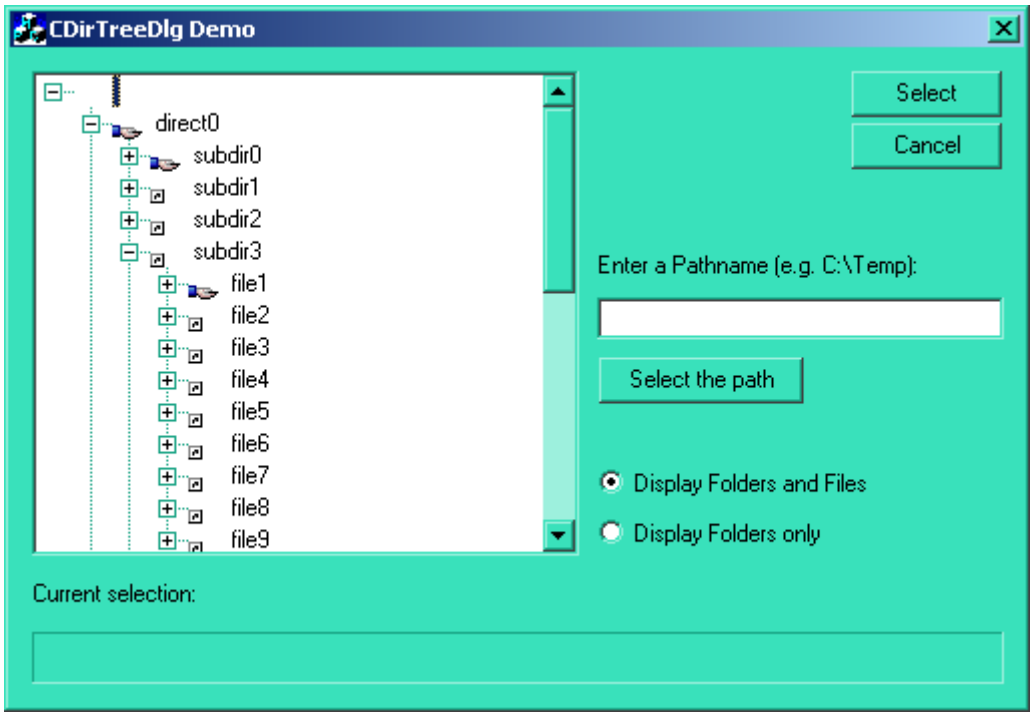


Рис. 61Окно программы – броузера файловой системы

Полученный симулятор файловый системы должен быть протестирован с помощью поставляемого вместе с лабораторией набора тестов на корректность работы. Кроме того, в результате тестирования собирается статистка по операциям чтения/записи, выполняемым модулем файловой системы – что может использоваться для оценки эффективности работы реализованного слушателем модуля.

Каталог FS содержит следующие файлы:

Driver.h	Содержит описание класса Driver, используемого для реализации операций чтения/записи на диск
Driver.cpp	Реализация класса Driver
VFS1.h	Содержит описание абстрактного класса VFS, который является базовым для реализуемых файловых систем

Класс слушателя, реализующий файловую систему, должен называться FS и располагаться в файлах FS.h и FS.cpp соответственно.

Каталог tests содержит следующие тесты

Тест1. Запись файла в файловую систему, чтение файла из файловой системы, сравнение записанного и прочитанного файлов.

Тест2. Последовательное создание 10 директорий, в каждой из директорий 10 поддиректорий, в каждой из них – 10 файлов размером по 50 байт. Чтение списков директорий. Сравнения списка записанных и прочитанных файлов.

Тест3. Несколько циклов создания/удаления большого количества мелких файлов.

Обзор архитектуры модуля поддержки файловой системы в Linux

Роль драйвера файловой системы состоит в выполнении низкоуровневых задач, используемых для распределения высокоуровневых операций VFS на физических устройствах. Интерфейс VFS достаточно универсален.

Драйвер создается из следующих компонентов, принадлежащих каталогу исходных текстов файловой системы или каталогу исходных текстов VFS:

1. Описание суперблока файловой системы - include/linux/type_fs.h описывает информацию в том виде, в каком она хранится на диске, include/linux/type_fs_sb.h содержит описание структуры, которая добавляется к стандартной структуре ядра, описывающей суперблок.

2. Описание inode - include/linux/type_fs_i.h содержит описание структуры inode на диске и структуры, которая добавляется к стандартной структуре ядра, описывающей inode.

3. Основной include файл - include/linux/type_fs.h.

4. Исходные тексты функций работы с inode - fs/fstype/inode.c. Структура inode_operations используется для операций с inode функциями VFS.

5. Исходные тексты функций работы с объектами файловой системы разных типов – файлами, каталогами, символьными ссылками и т.д. - fs/fstype/file.c, dir.c, symlink.c и т.д. Структуры типа struct file_operations используется для этих операций функциями VFS.

6. Операции, управляющие распределением и освобождением inode и блоков хранения данных - fs/fstype/bitmap.c

7. Операции с inode, связанные с именами, такие как создание и удаление имени, относящегося к inode, переименование и т.п.

Во время подключения файловой системы fstype_read_super заполняет структуру super_block информацией, полученной с поддерживаемого устройства. Поле s_op структуры содержит указатель на fstype_sops, используемый кодом VFS для выполнения операций над суперблоком.

Отключение происходит с помощью do_umount, включающем запуск fstype_put_super.

При доступе к файлу, fstype_read_inode заполняет общую системную inode структуру полями из fstype_inode. Поле inode.i_op заполняется указателями на реализации функций обработки inode для данной файловой системы.

Рассмотрение пунктов проще всего произвести на примере. Используем для этого файловую систему Minix.

Модули драйвера файловой системы Minix

Описание суперблока файловой системы

Формат хранения суперблока файловой системы на внешнем носителе содержится в файле include/linux/minix_fs.h. Суперблок содержит общую информацию о файловой системе – число файловых дескрипторов и т.д.

```
struct minix_super_block {
        __u16 s_ninodes;
        __u16 s_nzones;
        __u16 s_imap_blocks;
        __u16 s_zmap_blocks;
        __u16 s_firstdatazone;
        __u16 s_log_zone_size;
        __u32 s_max_size;
```

```
        __u16 s_magic;
        __u16 s_state;
        __u32 s_zones;
};
```

При монтировании файловой системы в ядре создается суперблок VFS для управления файловой системой. Его формат определен в файле linux/fs.h

```
struct super_block {
        struct list_head        s_list;
        kdev_t                  s_dev;
        unsigned long           s_blocksize;
        unsigned char           s_blocksize_bits;
        unsigned char           s_dirt;
          …
        union {
                struct minix_sb_info    minix_sb;
                struct ext2_sb_info     ext2_sb;
                struct ext3_sb_info     ext3_sb;
                struct msdos_sb_info    msdos_sb;
                    …
        } u;
};
```

Одним из полей структуры является объединение (union u;), предназначенное для хранения информации, зависимой от конкретного типа файловой системы. Состав и формат этих данных определяются в файле include/linux/ minix_fs_sb.h

```
struct minix_sb_info {
        unsigned long s_ninodes;
        unsigned long s_nzones;
        unsigned long s_imap_blocks;
        unsigned long s_zmap_blocks;
        unsigned long s_firstdatazone;
        unsigned long s_log_zone_size;
        unsigned long s_max_size;
        int s_dirsize;
        int s_namelen;
        int s_link_max;
        struct buffer_head ** s_imap;
        struct buffer_head ** s_zmap;
```

```
        struct buffer_head * s_sbh;
        struct minix_super_block * s_ms;
        unsigned short s_mount_state;
        unsigned short s_version;
};
```

Описание индексного дескриптора (inode)

Файл include/linux/minix_fs_i.h должен содержит описание структуры, которая добавляется к стандартной структуре ядра, описывающей индексный дескриптор.

```
struct minix_inode_info {
        union {
                __u16 i1_data[16];
                __u32 i2_data[16];
        } u;
};
```

Основной файл заголовков

Основной файл заголовков include/linux/minix_fs.h содержит следующие элементы

- определения констант, используемых при работе с файловой системой;

- описание формата хранения индексного дескриптора на диске

```
struct minix2_inode {
        __u16 i_mode;          // права доступа
        __u16 i_nlinks;        // число жестких ссылок (имен)
        __u16 i_uid;           // идентификатор владельца
        __u16 i_gid;           //идентификатор группового владельца
        __u32 i_size;          // размер
        __u32 i_atime;         // время последнего доступа
        __u32 i_mtime;         // время последнего изменения inode
        __u32 i_ctime;         // время последнего изменения данных
        __u32 i_zone[10];      // описания расположения файла на ВН
};
```

- описание формата хранения суперблока файловой системы на внешнем носителе (как мы уже сказали выше);

- описание формата хранения записи о дочернем объекта в директории (обратите внимание: в директории хранится только имя объекта файловой системы и номер его индексного дескриптора, все остальное хранится в индексном дескрипторе)

```
struct minix_dir_entry {
        __u16 inode;
```

```
        char name[0];
};
```

- прототипы всех функций работы с файловой системой, которые могут потребоваться подсистеме VFS;

- объявление структур, содержащих адреса функций работы с элементами файловой системы; эти структуры объявляются в файлах include/linux/file.c, include/linux/dir.c, include/linux/namei.c, используются при присвоении значений полям индексного дескриптора VFS: inode->i_op (операции над inode) и inode->f_op (операции над объектом файловой системы; впоследствии указанные поля используются подсистемой VFS для вызова функций чтения/записи файла и т.д.;

```
extern struct inode_operations minix_file_inode_operations;
extern struct inode_operations minix_dir_inode_operations;
extern struct file_operations minix_file_operations;
extern struct file_operations minix_dir_operations;
extern struct dentry_operations minix_dentry_operations;
```

Исходные тексты функций работы с индексным дескриптором

Файл fs/minix/inode.c содержит определение функций работы с индексным дескриптором (создание, удаление, чтение, запись и т.д.), функции для работы с суперблоком и определение структуры, в которой эти функции регистрируются.

```
static struct super_operations minix_sops = {
        read_inode:     minix_read_inode,
        write_inode:    minix_write_inode,
        delete_inode:   minix_delete_inode,
        put_super:      minix_put_super,
        write_super:    minix_write_super,
        statfs:         minix_statfs,
        remount_fs:     minix_remount,
};
```

Также в данном файле определены функции блочного ввода-вывода и структура с их перечислением.

```
static struct address_space_operations minix_aops = {
        readpage: minix_readpage,
        writepage: minix_writepage,
        sync_page: block_sync_page,
        prepare_write: minix_prepare_write,
        commit_write: generic_commit_write,
        bmap: minix_bmap
};
```

Исходные тексты функций работы с объектами ФС разных типов

Исходные тексты функций работы с файлами содержатся в файле fs/minix/file.c, каталогами - fs/minix/dir.c и fs/minix/namei.c. Набор реализованных операций является типичным.

```
struct file_operations minix_file_operations = {
        llseek:         generic_file_llseek,
        read:           generic_file_read,
        write:          generic_file_write,
        mmap:           generic_file_mmap,
        fsync:          minix_sync_file,
};
struct file_operations minix_dir_operations = {
        read:           generic_read_dir,
        readdir:        minix_readdir,
        fsync:          minix_sync_file,
};
struct inode_operations minix_dir_inode_operations = {
        create:         minix_create,
        lookup:         minix_lookup,
        link:           minix_link,
        unlink:         minix_unlink,
        symlink:        minix_symlink,
        mkdir:          minix_mkdir,
        rmdir:          minix_rmdir,
        mknod:          minix_mknod,
        rename:         minix_rename,
};
```

Другие модули

Функции, управляющие распределением и освобождением inode и блоков хранения данных определены в fs/minix/bitmap.c.

Операции с inode, связанные с именами, такие как создание и удаление имени, относящегося к inode, переименование и им подобные определены в fs/minix/namei.c.

Реализация драйвера файловой системы

При реализации драйвера файловой системы вы должны разработать структуру хранения ФС на диске и реализовать поддерживающий ее код. Простейший путь решения подобной задачи – модификация уже существующей реализации.

Объем исходных текстов драйвера файловой системы Minix составляет примерно 75Kb (~2000 строк, включая комментарии). Он поддерживает практически минимальный набор операций над объектами ФС и его код достаточно легок для понимания. Вы можете даже не добавлять новый тип файловой системы, а просто заменить реализацию MINIX.

Литература по лабораторной работе 5

1. Э. Таненбаум. Современные операционные системы. 2-е издание. СПб: Питер, 2002.

2. К. Грегори «Использование Visual C++ 6. Специальное издание»: Пер. с англ. – М.; СПб.; К.: Издательский дом «Вильямс», 2002.

3. David Rusling. The Linux Kernel (Электронный источник - www.linuxdoc.org/LDP/tlk/)

4. Cross-Referencing Linux (Электронный источник - lxr.linux.no)