# Тема 4. Теоретическая часть

Данный учебный материал предполагает, что предыдушие темы студентом изучены и закреплены на практике тремя лабораторными работами. Имея полученные зна- ния, можно провести нужный уровень детализации в изучении дисциплины. Здесь, такая детализация проводится для темы «Управление файловыми системами ОС».

## Устройства компьютера

*Впервую очередь*, термин *устройства компьютера* ассоциируются с аппаратурой ЭВМ, что в общем случае так и есть. Но в предмете нашей дисциплины, аппарату- ра компьютера не является непосредственно доступной для ползователя или прог- рамм, работающим в пользовательском режиме. Как ранее было отмечено:

* *доступ к устройствам* ЭВМ возможен только через ядро ОС;
* *все устройства* ЭВМ имеют отображение в виде имен в ФС ОС.

*С другой стороны*, устройства компьютера являются *ресурсами ОС*, которые управ-

ляются ядром ОС.

*С третьей стороны*, *сами файловые системы ОС* располагаются на устройствах компьютера.

*Указанное выше противоречие*, заключающееся в том, что устройства отображают- ся в ФС, которые сами находятся на устройствах, разрешается следующим образом:

* *все устройства ОС* делятся на *блочные* и *символьные*;
* *блочные устройства ОС* могут содержать файловые системы;
* *символьные устройства ОС* — устройства ОС, не являющиеся блочными;
* *в ядре ОС создаются блочные устройства*, которые не относятся к аппарат- ным средствам ЭВМ: *псевдоустройства ядра ОС* или устройства *nodev*;
* *псевдоустройства ядра ОС* имеют имена, которые почти все совпадают с именами соответсвующих файловых систем; эти имена соответствуют вер- шинам файловых систем (ФС);
* *псевдоустройство с именем rootfs* является корнем *виртуальной файловой системы* (***VFS — Virtual File System***).

*Когда GRUB загрузил ядро ОС* и передал ему файл с временной файловой сис- темой, то после инициации внутренних параметров ядра проводится *монтирова- ние (подключение) корневой файловой системы*. Это подключение (монтирование) проводится проводится самим ядром ОС относительно *внутренней точки ядра с именем* ***rootfs***.

*Далее*, ядро ОС создает первый процесс, которым является интерпретатор shell. Он, от имени пользователя ***root*** (UID=0) и группы GID=-1, начинает выполнять сцена- рий **/init**. Поскольку пользователь ***root*** работает *в режиме пользователя*, то он восп- ринимает точку ядра ***rootfs*** как **/**.

*В процессе выполнения* сценария **/init**, к директориям корневой ФС монтируются

*другие файловые системы*.

На рисунке 1.1 выведены типы файловых систем, поддерживаемых текущим сеан-

сом ОС УПК АСУ. Все *псевдоустройства ядра ОС* обозначены как *nodev*.

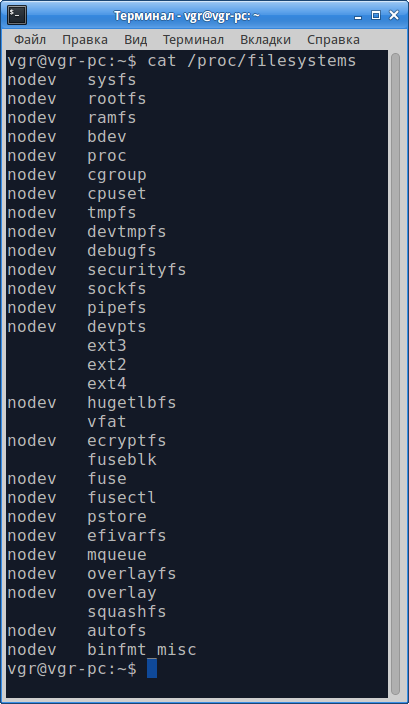


Рисунок 1.1 — Список файловых систем ядра ОС УПК АСУ

На рисунке 1.2 представлен вывод части сообщений команды ***mount***. Хорошо вид- но, монтирование некоторых псевдоустройств на директории корневой ФС.

По традиции, все ОС UNIX и Linux *отображают имена файлов устройств* в директорию ***/dev*** корневой файловой системы. Для устройств, точкой монтирования ядра ОС является ***dev***, что также хорошо видно из рисунка 1.2.

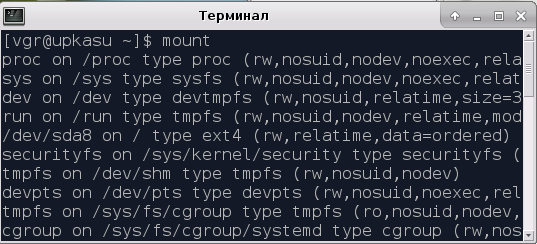


Рисунок 1.2 — Примеры монтирования псевдоустройств ядра

*Общая семантика* слова *устройство* соответствует семантике английского слова *device*. Реальные файловые системы ОС, на которых долговременно хранится ин- формация, находятся на физических устройствах ЭВМ. Поскольку физические утройства типа винчестера разбиваются на разделы, то *реальные ФС создаются в разделах блочных устройств*. Именно разделы блочных устройств *монтируются к директориям VFS*.

*В терминологии ядра*, *имя устройства*, которое отображается в директории ***/dev***, называется ***узел*** (***node***).

**NODE** - *узел* - специальная *именованная структура*, отображаемая в директории

***/dev***, создаваемая для связи ядра ОС с физическим устройством ЭВМ.

**Замечание** Чтобы ядро ОС могло работать с физическими устройствами ЭВМ, *узлы (node)* должны быть созданы заранее и иметь отображение в директории ***/dev***.

*Узлы создаются* как для *блочных*, так и для *символьных устройств*. *Каждый узел* имеет *имя, тип, старший\_номер и младший\_номер*.

*Все узлы* хранятся в *специальной таблице* (специальной файловой системе). Напри- мер, из рисунка 1.2 видно, что директория ***/dev*** монтирована в точку ядра ОС с име- нем ***dev***, имеющей тип файловой системы ***devtmpfs***.

Для создания устройств используется утилита **mknod**, в формате;

### mknod [OPTION]... NAME TYPE MAJOR MINOR

где NAME — имя устройства;

TYPE равно **b** для блочного устройства; TYPE равно **c** для символьного устройства; TYPE равно **p** для канала FIFO;

MAJOR, MINOR — старший и младший номер устройства.

Например,

[ -e /dev/console ] || mknod -m 0600 /dev/console c 5 1

[ -e /dev/null ]

|| mknod /dev/null c 1 3

В этом примере, для устройства ***консоли*** и устройства ***null***, проверяется наличие узла по его имени, и если узлы отсутствуют, то они создаются.

На рисунке 1.3, с помощью команды ***ls -l*** и утилиты фильтра ***grep***, показан вывод информации об узлах *console* и *null*.

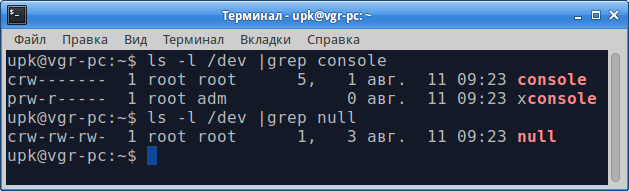


Рисунок 1.3 — Вывод информации об узлах устройств

**Замечание** Для создания узлов необходимо знать не только имена и типы устройств, но и их номера (младшие и старшие). Наличие большого количества различных физических устройств ЭВМ и их различная группировка, создает ряд известных проблем, при создании и сопровождении узлов. Более подробно, эти проблемы обсуждаются в подразделе 1.10

«Три концепции работы с устройствами».

В данном курсе, подробная работа с символьным устройствами не рассматривается, поскольку она всегда имеет свою специфику и детальное знание самих устройств. Поэтому информация об использовании символьных устройств дается только в общем контексте их применения в ОС.

## BOOT-сектор и разделы винчестера

*Учитывая тематику* изучаемого материала, мы ограничимся рассмотрением конк- ретного физического устройства ЭВМ, известного как *винчестер* или «*жесткий диск*», который является *основным блочным устройством* для долговременного хранения информации и одновременно - *оперативно используется в работе ОС*. *Кроме того*, именно на одном из разделов винчестера размещается файловая систе- ма, рассматриваемая ядром ОС как *корневая ФС*.

*Все блочные устройства*, на которых размещаются конкретные ФС имеют свою

структуру, определяемую физическим типом устройства. Такую структуру имеют и физические устройства, называемые *винчестер*. Широкое применение этого обору- дования привело к стандартизации его общей структуры, которая уже была рас- смотрена ранее в теме 2 [6, подраздел 1.6 «Винчестер и загрузочные устройства»]:

* *классическая* структура MBR;
* *новая структура* GPT.

*Широкое и успешное применение* применение устройств типа винчестер и их об- щих структур привело к тому, что эти структуры стали переноситься на другие уст- ройства, физически отличающиеся от винчестера. Примером таких устройств явля- ется flashUSB, которое является микросхемой и не имеет дисков, головок и секто- ров. Тем не менее, перенос структуры винчестера на flashUSB позволяет использо- вать это устройство как блочное, с наименьшими изменениями в системном ПО ОС.

*Ограничиваясь только классической структурой MBR*, можно выделить:

* *главный загрузочный сектор* (MBR), который не зависит от типа ОС;
* *загрузочные секторы* (блоки) логических дисков (***разделов***), которые зависят от ОС только в плане поддерживамых ей типов ФС;
* *специальные области разметки* и *корневой каталог*, зависящие от типа фай- ловой системы;
* *область данных* – файлы и каталоги конкретной файловой системы;
* *цилиндр* для выполнения диагностических операций чтения-записи.

*Указанная структура* демонстрируется схемой, представленной на рисунке 1.4.

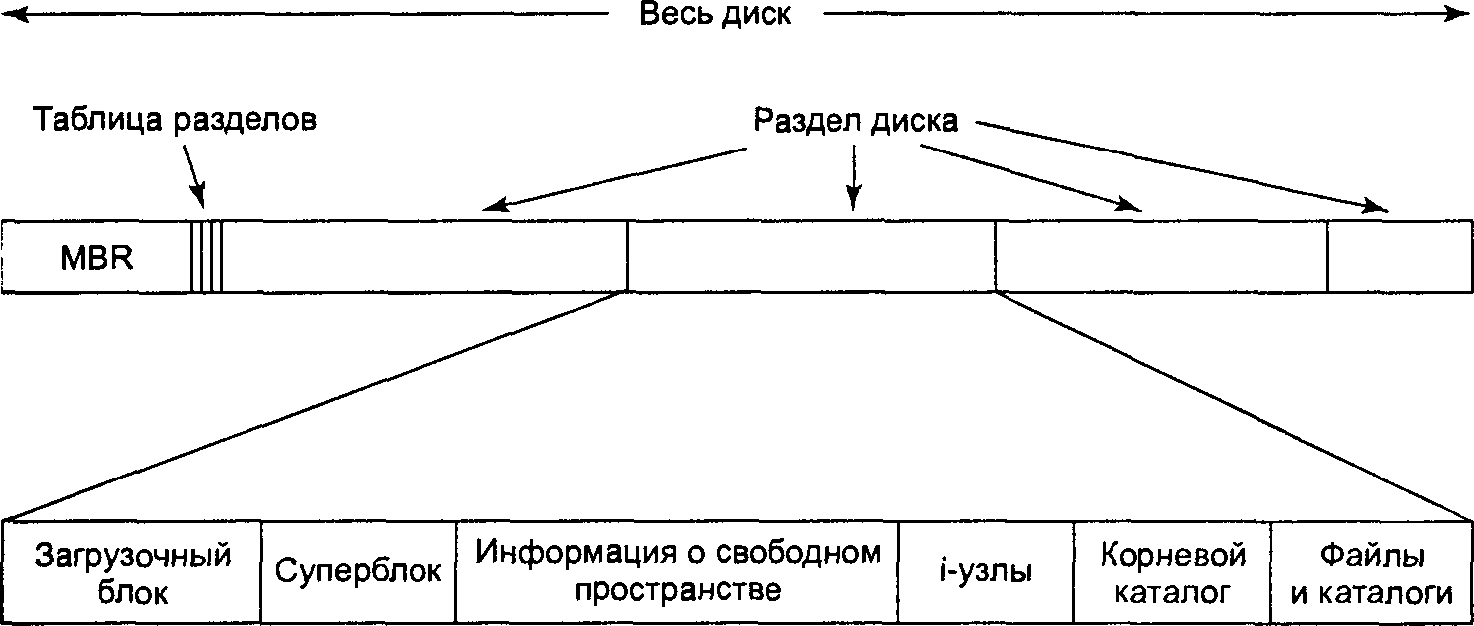


Рисунок 1.4 - Общая структура винчестера

*Для поддрежки такой структуры* блочных устройств, каждая ОС имеет специаль- ные *системные утилиты*.

**Для примера**, в ОС УПК АСУ, такими утилитами являются:

* **fdisk** - позволяет показать все блочные устройства ЭВМ, а также *обеспечить*

*разбиение устройства* на нужное количество разделов;

* **mkfs** — позволяет *создать файловую систему на заданном разделе* заданного блочного устройства;
* **mknod —** позволяет *создавать узлы в ФС*, которые именуют блочные и сим- вольные устройства для доступа к ним ПО пользовательского режима;
* **mount** — позволяет *монтировать (подключить) конкретную ФС* к древовид- ной структуре файлов ядра ОС;
* **umount** — позволяет *размонтировать (отключить) конкретную ФС* от дре- вовидной структуры файлов ядра ОС.

*Следуя основной парадигме UNIX-систем*: «**Все есть файл**», разработчики ОС создавали ***свои файловые системы***, которые ***монтировались*** при загрузке ОС и использовались, в дальнейшем, как ***уникальные*** хранилища информации.

*Фактически*, *задача загрузки первых реализаций ОС* сводилась (см. рисунок 1.4):

* *к использованию загрузочного блока* для загрузки ядра и передачи ему *ин- формации о разделе конкретного блочного устройства*;
* *ядро ОС*, проведя инициализацию внутренних параметров и внешних уст- ройств ЭВМ, *находила этот раздел* и *монтировала его* как *корневую файловую систему*.

*Хотя каждая ФС* была оптимизирована под ядро конкретной ОС, подобный под- ход имел две существенные проблемы:

* *правильное монтирование* корневой ФС было возможным только после пред- варительной инсталляции ОС, *посредством специального инсталлятора*;
* *перенос данных* с других («**не родных**») ФС был крайне затруднительным.

**Замечание** Многие корпорации такая ситуация вполне устраивала, но ***в 1985 году***, компания Sun Microsystems для ОС SunOS предложила **VFS** — Virtual File System (File System Switch) - *Виртуальнаую Файловую Систему*.

## Загрузочные сектора разделов

Для блочного устройства типа винчестер, минимальным физически читаемым объемом информации (данных) является сектор, который для современного обору- дования ЭВМ равен **512 байт**.

*Понятие сектора* связано физической организацией обмена данными в блочных устройствах. Файловые системы (ФС), которые стали создаваться для различных нужд ОС, стали использовать *понятие блока*.

*Понятие блока* связано с логической организацией обмена данными между ОС и блочными устройствами. Соответственно размер блока может быть равен *одному сектору* или *кратен целому числу таких секторов*.

*С другой стороны*, понятие блока стали вводить с целью преодоления ограничений,

которые накладываются способом адресации разделов в таблице MBR и ПО BIOS, которые ограничивали возможный объем используемых блочных устройств. Например, в ОС MS Windows для форматирования блочных устройств стало ис- пользоваться понятие кластер.

*Кластер* — это последовательный набор секторов винчестера, который на логичес- ком уровне рассматривается как единица обмена данными между ОС и блочным устройством.

*С третьей стороны*, конструктивные особенности блочных устройств типа вин- честера связаны с тем, что линейная скорость движения головки чтения/записи по внешним трекам диска выше, чем по внутренним. Соответственно, на внешних треках можно расположить больше секторов, чем на внутреннем, что позволяет более эффективно использовать поверхность диска, но делает непригодной систе- му адресации секторов: (***CHS***). Выход из данной ситуации был найден с помощью ***LBA***, которая саму адресацию секторов перекладывает на контроллер винчестера, а для ОС предоставляется упорядоченная последовательность логических блоков.

*Наконец*, разработка новой структуры блочных устройств GPT снимает проблему логической адресации болших объемов устройств, *расширив формат указателей адреса и стандартизировав размер LBA на уровне одного сектора (512 байт)*.

*Другой круг проблем* связан с ПО самой BIOS, рассмотренной нами в [6, разделы

1.2 и 1.3]. Дело в том, что функции и прерывания BIOS могут работать только на уровне секторов и LBA, а также понимают структуру MBR. По этой причине, для загрузки ОС необходимо в разделе корневой файловой системы *выделить область для ПО загрузчика*, а в программный код первых ***446 байт*** MBR *прописать адрес загрузчика*. Очевидно, что такой адрес должен указывать на фиксированное место- положение ПО загрузчика относительно начала раздела. Поэтому, практически все ФС включают в свою структуру некоторый блок загрузки, как это показано на ри- сунке 1.4. Как следствие, такой подход позволяет загружать только одну ОС с одно- го блочного устройства.

*Альтернативный подход* предполагает *смещение начала первого раздела* относи- тельно сектора MBR. Такое смещение позволяет записать стразу же после сектора MBR ПО загрузчика, что и используется нами при установке ***ПО GRUB***.

**Замечание** Использование альтернативных загрузчиков делает не нужным использование загрузоч- ных секторов разделов, но, в силу преемственности структур ФС, данные сектора оста- ются.

*Очевидно*, что структуры различных ФС могут сильно отличаться друг от друга. На рисунке 1.4 приведена обобщенная структура ФС, характерная для ОС UNIX. Да- лее, на примере двух ФС — *FAT32* и *ext2fs*, мы рассморим их основные особеннос- ти, а затем проведем их сравнительный анализ.

## Структура файловой системы FAT32 (VFAT)

**FAT32** - файловая система компании ***Microsoft***, разработанная ***в августе 1996 года***. Она является преемницей файловых систем ***FAT8***, ***FAT12*** и ***FAT16***, начало разра- ботки которых положено Биллом Гейтсом и Марком МакДональдом, ***в 1976 - 1977 годах***.

**VFAT** — расширение ***FAT***, появившееся в ***MS Windows 95*** и дающее возможность *использовать имена файлов длиной до 255 символов*, в кодировке ***UTF-16LE***. **Фактически**, ***VFAT*** и используется во всех ОС, как универсальный хранитель ин- формации.

**FAT** – *File Allocation Table* или *Таблица размещения файлов*.

**32** – *число бит*, используемое для нумерации *кластеров* (блоков) раздела файловой системы (на самом деле используется ***28 бит***).

**Кластер** – это объем данных, которыми оперирует файловая система. Обычно, для 8 Гбайт раздела используется 4 КБайтные кластеры (8 секторов диска).

**Один сектор** диска – 512 байт.

*Загрузочный сектор раздела* содержит:

* *блок параметров диска (BPB)*, в котором содержится информация о разделе: размер и количество секторов, размер кластера, метка тома и другие;
* *загрузочный код* – программу, с которой начинается процесс загрузки опера- ционной системы (для MS-DOS и MS Windows-9X – файл ***Io.sys***).

*На этапе логического форматирования* каждого раздела (*логического диска*) созда- ются четыре логических области:

* *загрузочный сектор* (boot sector);
* *таблица размещения файлов* (FAT1 и FAT2);
* *каталог*;
* *область данных*.

***Загрузочный сектор*** содержит в себе, *кроме кода загрузчика*, *таблицу BPB* и двухбайтовую сигнатуру ***55AA*** (0xAA55).

**BPB** — *BIOS Parameter Block* — таблица содержашая множество параметров, определяющих характеристики блочного устройства и самой файловой системы.

### Замечание

Для нужд FAT32, таблица BPB была расширена и названа ***BF\_BPB***.

**BF\_BPB** — *Big FAT BIOS Parameter Block*. Размер загрузочного сектора:

* ***один*** физический сектор для FAT16;
* ***три*** физических сектора для FAT32.

Далее, в таблице 1.1, представлена структура первого сектора (***boot sector***) файло- вой системы FAT32.

*Таблица 1.1 - Первый сектор раздела FAT32*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Смеще- ние, байт*** | ***Длина поля,***  ***байт*** | ***Обозначе- ние*** | ***Содержание*** |
| 0х00 (0) | 3 | *JUMP 90* | Безусловный переход на начало загрузчика |
| 0x03 (3) | 8 |  | *Системный идентификатор* — начало *BF\_BPB* |
| 0x0B (11) | 2 | *SectSize* | Размер сектора, байт (обычно 512) |
| 0x0D (13) | 1 | *ClastSize* | Число секторов в кластере |
| 0x0E (14) | 2 | *ResSecs* | Число зарезервированных секторов, равно 32 |
| 0x10 (16) | 1 | *FATcnt* | Число копий FAT, обычно 2 |
| 0x11 (17) | 2 | *RootSize* | Количество элементов в корневом каталоге |
| 0x13 (19) | 2 | *TotSect* | Общее число секторов. 0, если размер больше 32 Мб |
| 0x15 (21) | 1 | *Media* | Дескриптор носителя |
| 0x16 (22) | 2 | *FATsize* | Количество секторов на элемент таблицы FAT |
| 0x18 (24) | 2 | *TrkSecs* | Число секторов на дорожке |
| 0x1A (26) | 2 | *HeadCnt* | Число рабочих поверхностей (число головок) |
| 0x1C (28) | 4 | *HidnSecs* | Число скрытых секторов, которые расположены перед загрузочным сектором. Используются при загрузке для вычисления абсолютного смещения корневого каталога и данных |
| 0x20 (32) | 4 |  | Число секторов на логическом диске |
| 0x24 (36) | 4 |  | Число секторов в таблице FAT |
| 0x28 (40) | 2 |  | Расширенные флаги |
| 0x2A (42) | 2 |  | Версия файловой системы |
| 0x2C (46) | 4 |  | Номер кластера для первого кластера каталога. Обычно, значение этого поля равно 2. |
| 0x30 (48) | 2 |  | Номер информационного сектора ФС |
| 0x34 (50) | 2 |  | Номер сектора с резервной копией загр. сектора |
| 0x34 (52) | 12 |  | Зарезервировано FAT32 (расширение BF\_BPB) |
| 0x40 (64) | 1 |  | Физический номер устройства: 0х00 — флоппи диск;  0х80 — жесткий диск. |
| 0x41 (65) | 1 | 0 | Зарезервировано для FAT32 |
| 0x42 (66) | 1 |  | Сигнатура расширенного загрузочного сектора. |
| 0x43 (67) | 4 |  | Серийный номер тома. Случайное число, которое генерируется при фоматировании |
| 0x47 (71) | 11 |  | Метка тома |
| 0x52 (82) | 8 |  | Тип файловой системы (12-, 16-, 32-разрядная) |
| 0x5A (90) | 420 |  | *Начало кода системного загрузчика* |
| 0x1FE (510) | 2 |  | Сигнатура (слово AA55) |

Поскольку содержимое загрузочной записи имеет для ОС важное значение, то

*обычно имеются резервные копии этой записи*.

*Резервная загрузочная запись*, как правило, располагается *в секторах 7-9 раздела*.

**Таблица размещения файлов (FAT)** – это *массив целых чисел* с длинной, равной количеству кластеров раздела файловой системы.

*Номер элемента этого массива* – это *номер кластера в разделе файловой системы*. *Отдельный элемент массива*:

* для ***FAT16*** — это 2-х байтовые числа;
* для ***FAT32* —** это 4-х байтовые числа.

*Значение элемента таблицы FAT* **—** это *ссылка на следующий номер элемента*

*таблицы FAT*. В таблице 1.2, представлены возможные значения таблицы FAT.

*Таблица 1.2 - Значения элементов таблицы FAT32 (используется 28 бит)*

|  |  |
| --- | --- |
| ***Значение*** | ***Описание*** |
| 0x00000000 | Свободный кластер |
| 0x00000002 – 0x0FFFFFEF | Номер следующего кластера в цепочке |
| 0x0FFFFFF0 – 0x0FFFFFF6 | Зарезервированный кластер |
| 0x0FFFFFF7 | Плохой кластер |
| 0x0FFFFFF8 – 0x0FFFFFFF | Последний кластер в цепочке |

**Замечание** В файловой системе FAT, между копиями загрузочной записи раздела и корневым каталогом, *находятся две копии таблиц FAT*.

При загрузке ОС таблица **FAT** загружается в оперативную память компьютера. Для больших файловых систем таблица **FAT** может загружаться частями.

**За таблицами FAT** располагается *корневой каталог файловой системы* (***Root Directory***), размером:

* 512 байт, для **FAT16**.
* 2048 байт, для **FAT32**, сейчас — до 65535 элементов записей по 32 байта.

**Корневой каталог файловой системы** – список записей по ***32 байта***. *Исторически, сложилась ситуация*, что файловая система FAT содержит ***два типа имен***:

* *короткие имена* — до восьми байт имя и три байта расширение имени;
* *длинные имена* — до 255 символов в формате ***UTF-16LE***.

*Формат записи короткого имени файла* вмещается в одну 32-байтовую запись.

*Структура этого формата* представлена в таблице 1.3.

*Одинадцатый байт записи* содержит поле «***Атрибуты***», значения которого пред- ставлены в таблице 1.4.

**Для длинных имен файлов**, в формате ***Unicode-16***, используется несколько 32- байтовых записей каталога в специальном формате. Один элемент, такого формата записи, показан на рисунке 1.5 и имеет несколько полей.

*Таблица 1.3 - Основной вариант записи для короткого имени FAT32*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Смещение*** | | | ***Описание*** |
| ***Hex*** | ***Dec*** | ***Длина поля*** |
| 00h | 0 | 8 байт | Имя файла |
| 08h | 8 | 3 байт | Расширение файла |
| 0Bh | 11 | 1 байт | *Атрибуты файла* |
| 0Ch | 12 | 2 байт | Зарезервировано для NT |
| 0Eh | 14 | 2 байт | Время создания файла |
| 10h | 16 | 2 байт | Дата создания файла |
| 12h | 18 | 2 байт | Дата последнего доступа |
| 14h | 20 | 2 байт | *Старшее слово номера начального кластера* |
| 16h | 22 | 2 байт | Время последней записи |
| 18h | 24 | 2 байт | Дата последней записи |
| 1Ah | 26 | 2 байт | *Младшее слово начального кластера* |
| 1Ch | 28 | 4 байт | Размер файла в байтах |

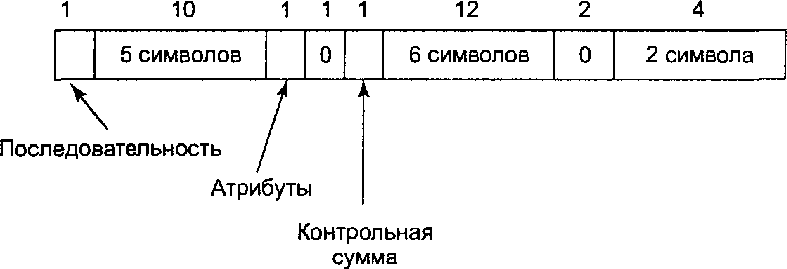


Рис. 1.5 - Структура элемента записи для длинного имени файла

*Таблица 1.4 - Значения 11-го байта атрибутов*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***7*** | ***6*** | ***5*** | ***4*** | ***3*** | ***2*** | ***1*** | ***0*** | ***Hex*** | ***Значение*** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01h | Только чтение |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02h | Скрытый |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 04h | Системный |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 08h | Метка тома |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10h | Подкаталог |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 20h | Архивный (измененный) |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 40h | Зарезервировано |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80h | Зарезервировано |

**Область данных** — все остальные кластеры раздела. Они используются для хра- нения *подкатологов* и *файлов*.

*Подкатологи* – это файлы, содержащие последовательности 32-битных записей

*имен файлов и подкаталогов*, указанного выше формата.

*Содержимое файлов* — данные *некоторой последовательности кластеров*. *Последовательность кластеров* — произвольная и неупорядоченная, но *отраже- на в виде цепочки*, в таблице FAT.

**Замечание** Чем больше размер кластера, тем больший размер раздела может адресоваться и повы- шается скорость обработки файлов, но уменьшается эффективность их хранения.

## Структура файловой системы EXT2FS

**ext2 (ext2fs)** - *Second Extended File System* — базовая файловая система ОС Linux.

**Разработана** Реми Кардом и представлена *в январе 1993 года*.

**Поддерживает** файлы размером *2 ГБайт — 2 ТБайт*.

**Размер файловой системы** - *2 — 32 ТБайт*.

**Является** преемницей ***ext*** - *Extended File System*, представленной *в апреле 1992 года*.

**Ext** - поддерживала файлы размером *до 2 Гбайт* и длину имени файла до *255 байт*.

**Структура дискового раздела** ext2fs представляет *последовательность группы блоков*, которые нумеруются, начиная с 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Загрузочная запись**  – принадлежит 1-й группе блоков | Группа блоков 1 | Группа блоков 2 | ... | Группа блоков n |

*Каждая группа блоков* состоит из *последовательности блоков*, которые также нумеруются, начиная с 1.

*Каждая группа блоков* имеет *одинаковое число блоков*, кроме последней, которая может быть неполной.

*Начало каждой группы блоков* имеет *адрес*, который может быть получен как:

**Address = (*номер\_группы* — 1)\* (*число\_блоков\_в\_группе*)**

*Размер блока* может быть *1, 2 или 4 килобайта*, что определяется в момент форма- тирования – при создании файловой системы.

*Блок* является *адресуемой единицей дискового пространства*.

**Каждая группа блоков** имеет одинаковое строение:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Супер- блок | Описание группы блоков (Group Descriptors) | Битовая карта блоков (Block Bitmap) | Битовая карта индексных дескрипторов (Inode Bitmap) | Таблица индексных дескрипторов (Inode Table) | Область блоков данных |

Такая структура служит повышению производительности файловой системы:

* сокращается расстояние между таблицей индексных дескрипторов и блоками данных;
* сокращается время поиска нужного места головками в процессе операций записи/считывания файла.

**Первый элемент** группы блоков - *суперблок*, который одинаков для всех групп.

**Все остальные элементы** - индивидуальны для каждой группы.

**Первые 1024 байта раздела** занимает *загрузочная запись раздела*. *Суперблок*, хранится в первой группе блоков и имеет ***смещение 1024 байта***.

*Наличие нескольких копий* суперблока объясняется чрезвычайной важностью этого элемента файловой системы.

*Дубликаты суперблока* используются при восстановлении файловой системы после сбоев.

*Информация*, хранимая в суперблоке, используется для организации доступа к остальным данным на диске. В суперблоке определяется размер файловой системы, максимальное число файлов в разделе, объем свободного пространства и содержит- ся информация о том, где искать незанятые участки.

**При запуске ОС**, суперблок считывается в память и все изменения файловой системы вначале находят отображение в копии суперблока, находящейся в ОЗУ, и записываются на диск только периодически.

**При выключении системы**, суперблок обязательно должен быть записан на диск.

*Таблица 1.5 - Структура суперблока*

|  |  |  |
| --- | --- | --- |
| ***Название поля*** | ***Тип*** | ***Комментарий*** |
| s\_inodes\_count | ULONG | Число индексных дескрипторов в файловой системе |
| s\_blocks\_count | ULONG | Число блоков в файловой системе |
| s\_r\_blocks\_count | ULONG | Число блоков, зарезервированных для *суперпользователя* |
| s\_free\_blocks\_count | ULONG | Счетчик числа свободных блоков |
| s\_free\_inodes\_count | ULONG | Счетчик числа свободных индексных дескрипторов |
| s\_first\_data\_block | ULONG | Первый блок, который содержит данные. В зависимости от размера блока, это поле может быть равно 0 или 1. |
| s\_log\_block\_size | ULONG | Индикатор размера логического блока: 0 = 1 Кб; 1 = 2  Кб; 2 = 4 Кб. |
| s\_log\_frag\_size | LONG | Индикатор размера фрагментов |

|  |  |  |
| --- | --- | --- |
| s\_blocks\_per\_group | ULONG | Число блоков в каждой группе блоков |
| s\_frags\_per\_group | ULONG | Число фрагментов в каждой группе блоков |
| s\_inodes\_per\_group | ULONG | Число индексных дескрипторов (inodes) в каждой группе блоков |
| s\_mtime | ULONG | Время, когда в последний раз была смонтирована файловая система. |
| s\_wtime | ULONG | Время, когда в последний раз производилась запись в файловую систему |
| s\_mnt\_count | USHORT | *Счетчик числа монтирований* файловой системы. Если этот счетчик достигает значения, указанного в следующем поле (s\_max\_mnt\_count), файловая система должна быть проверена (это делается при перезапуске), а счетчик обнуляется. |
| s\_max\_mnt\_count | SHORT | Число, определяющее, сколько раз может быть смонтирована файловая система |
| s\_magic | USHORT | "Магическое число" (***0xEF53***), указывающее, что файловая система принадлежит к типу ex2fs |
| s\_state | USHORT | Флаги, указывающее текущее состояние файловой системы: является ли она чистой (clean) и т.п. |
| s\_errors | USHORT | Флаги, задающие процедуры обработки сообщений об ошибках (что делать, если найдены ошибки). |
| s\_pad | USHORT | Заполнение |
| s\_lastcheck | ULONG | Время последней проверки файловой системы |
| s\_checkinterval | ULONG | Максимальный период времени между проверками файловой системы |
| s\_creator\_os | ULONG | Указание на тип ОС, в которой создана файловая система |
| s\_rev\_level | ULONG | Версия (revision level) файловой системы. |
| s\_reserved | ULONG[ 235] | Заполнение до 1024 байт (235 \* 4 = 940) |

**Вслед за суперблоком** расположено *описание группы блоков* (Group Descriptors). Это описание имеет *структуру длиной 32 байта*.

*Таблица 1.6 - Структура описания группы блоков*

|  |  |  |
| --- | --- | --- |
| ***Название поля*** | ***Тип*** | ***Назначение*** |
| bg\_block\_bitmap | ULONG | Адрес блока, содержащего битовую карту блоков (block bitmap) данной группы |
| bg\_inode\_bitmap | ULONG | Адрес блока, содержащего битовую карту индексных дескрипторов (inode bitmap) данной группы |
| bg\_inode\_table | ULONG | Адрес блока, содержащего таблицу индексных дескрипто- ров (inode table) данной группы |
| bg\_free\_blocks\_c ount | USHORT | Счетчик числа свободных блоков в данной группе |
| bg\_free\_inodes\_count | USHORT | Число свободных индексных дескрипторов в данной группе |
| bg\_used\_dirs\_count | USHORT | Число индексных дескрипторов в данной группе, которые являются каталогами |
| bg\_pad | USHORT | Заполнение |
| bg\_reserved | ULONG[ 3] | Заполнение |

*Размер описания группы блоков* можно вычислить как:

**N= (размер\_группы\_блоков\_в\_ext2 \* число\_групп) / размер\_блока**

*Информация*, которая хранится в описании группы, используется, чтобы найти:

* битовые карты блоков,
* индексных дескрипторов,
* таблицу индексных дескрипторов.

**Битовая карта блоков** (***block bitmap***) - это структура, каждый бит которой показывает, отведен ли соответствующий ему блок какому-либо файлу.

*Если бит равен 1*, то блок занят.

*Эта карта* также служит для поиска свободных блоков в тех случаях, когда надо выделить место под файл.

*Битовая карта блоков* занимает число блоков, равное:

### Bloks = (число\_блоков\_в\_группе / 8) / размер\_блока

**Битовая карта индексных дескрипторов** выполняет аналогичную функцию по отношению к таблице индексных дескрипторов: показывает *какие именно дескрип- торы заняты*.

### Таблица индексных дескрипторов:

*После битовой карты индексных дескрипторов*, следует *таблица индексных деск- рипторов файлов* **- *Inode Table***.

Каждая строка ***Inode Table*** имеет *размер 128 байт* и имеет *структуру индексного*

*дескриптора файлов*, рассмотренного ниже.

Количество строк ***Inode Table***, а следовательно, и размер таблицы, *задается при создании файловой системы*.

**Замечание**

Каждый файл требует одной записи в таблицу дескрипторов. Поэтому *размер этих таб- лиц нужно предусмотреть заранее*.

**Среди индексных дескрипторов** имеется *несколько значений дескрипторов*, кото- рые *зарезервированы для специальных целей* и играют особую роль в файловой системе.

**Замечание**

Значения этих дескрипторов, которые перечислены в таблице 1.7, не могут быть исполь- зованы для других целей.

*Таблица 1.7 - Специальные индексные дескрипторы*

|  |  |  |
| --- | --- | --- |
| ***Идентификатор*** | ***Знач.*** | ***Описание*** |
| EXT2\_BAD\_INO | 1 | Индексный дескриптор, в котором перечисле- ны адреса дефектных блоков на диске (Bad blocks inode) |
| EXT2\_ROOT\_INO | 2 | Индексный дескриптор корневого каталога файловой системы (Root inode) |
| EXT2\_ACL\_IDX\_INO | 3 | ACL inode |
| EXT2\_ACL\_DATA\_INO | 4 | ACL inode |
| EXT2\_BOOT\_LOADER\_INO | 5 | Индексный дескриптор загрузчика (Boot loader inode) |
| EXT2\_UNDEL\_DIR\_INO | 6 | Undelete directory inode |
| EXT2\_FIRST\_INO | 11 | Первый незарезервированный индексный деск- риптор |

*Самый важный дескриптор в этом списке* - *дескриптор корневого каталога*. *Каталог файловой системы* - файл, состоящий из записей переменной длины.

*Таблица 1.8 - Структура файла корневого каталога*

|  |  |  |
| --- | --- | --- |
| ***Название поля*** | ***Тип*** | ***Описание*** |
| inode | ULONG | Номер индексного дескриптора (индекс) файла |
| rec\_len | USHORT | Длина этой записи |
| name\_len | USHORT | Длина имени файла |
| name | CHAR[0] | Первый символ имени файла |

**Замечание** Отдельная запись в каталоге не может пересекать границу блока, то есть должна быть расположена целиком внутри одного блока.

Если очередная запись каталога не помещается целиком в данном блоке, то она пе- реносится в следующий блок, а предыдущая запись продолжается таким образом, чтобы она заполнила блок до конца.

### Индексные дескрипторы файлов:

Вся информация о файле хранится в индексном десткрипторе (***Inode***).

Число файлов, которое может быть создано в файловой системе, *ограничено числом индексных дескрипторов*, которое:

* *либо явно задается*, при создании файловой системы;
* *либо вычисляется*, исходя из физического объема дискового раздела.

*Размер* индексного дескриптора = 32 \* 4 = ***128 байт***.

*Структура* индексного дескриптора приведена в таблице 1.9.

*Таблица 1.9 - Структура индексного дескриптора файла*

|  |  |  |
| --- | --- | --- |
| ***Название поля*** | ***Тип*** | ***Описание*** |
| i\_mode | USHORT | Тип и права доступа к данному файлу. |
| i\_uid | USHORT | Идентификатор владельца файла (Owner Uid). |
| i\_size | ULONG | Размер файла в байтах. |
| i\_atime | ULONG | Время последнего обращения к файлу (Access time). |
| i\_ctime | ULONG | Время создания файла. |
| i\_mtime | ULONG | Время последней модификации файла. |
| i\_dtime | ULONG | Время удаления файла. |
| i\_gid | USHORT | Идентификатор группы (GID). |
| i\_links\_count | USHORT | Счетчик числа связей (Links count). |
| i\_blocks | ULONG | Число блоков, занимаемых файлом. |
| i\_flags | ULONG | Флаги файла (File flags) |
| i\_reserved1 | ULONG | Зарезервировано для ОС |
| i\_block | ULONG[1 5] | Указатели на блоки данных, которые рассмотрены далее в разделе «Система адресации данных» |
| i\_version | ULONG | Версия файла (для NFS) |
| i\_file\_acl | ULONG | ACL файла - *Access Control List – список доступа* |
| i\_dir\_acl | ULONG | ACL каталога - *Access Control List – список доступа* |
| i\_faddr | ULONG | Адрес фрагмента (Fragment address) |
| i\_frag | UCHAR | Номер фрагмента (Fragment number) |
| i\_fsize | UCHAR | Размер фрагмента (Fragment size) |

|  |  |  |
| --- | --- | --- |
| i\_pad1 | USHORT | Заполнение |
| i\_reserved2 | ULONG[2  ] | Зарезервировано |

Среди множества полей индесного десткриптора файла, более подробно мы рас- смотрим ***i\_mode*** и ***i\_block***.

Поле **i\_mode** - тип и права доступа к файлу.

Оно является *двух-байтовым словом*, каждый бит которого служит *флагом*, инди- цирующим *отношение файла к определенному типу* или *установку одного конк- ретного права на файл* (таблица 1.10).

*Таблица 1.10 - Поле прав доступа к файлу*

|  |  |  |
| --- | --- | --- |
| ***Идентификатор*** | ***Значение*** | ***Назначение флага (поля)*** |
| **S\_IFMT** | **F000** | **Маска для типа файла** |
| S\_IFSOCK | A000 | Доменное гнездо (socket) |
| S\_IFLNK | C000 | Символическая ссылка |
| S\_IFREG | 8000 | Обычный (regular) файл |
| S\_IFBLK | 6000 | Блок-ориентированное устройство |
| S\_IFDIR | 4000 | Каталог |
| S\_IFCHR | 2000 | Байт-ориентированное (символьное) устройство |
| S\_IFIFO | 1000 | Именованный канал (fifo) |
|  |  |  |
| S\_ISUID | 0800 | SUID - бит смены владельца |
| S\_ISGID | 0400 | SGID - бит смены группы |
| S\_ISVTX | 0200 | Бит сохранения задачи (sticky bit) |
|  |  |  |
| **S\_IRWXU** | **01C0** | **Маска прав владельца файла** |
| S\_IRUSR | 0100 | Право на чтение |
| S\_IWUSR | 0080 | Право на запись |
| S\_IXUSR | 0040 | Право на выполнение |
|  |  |  |
| **S\_IRWXG** | **0038** | **Маска прав группы** |
| S\_IRGRP | 0020 | Право на чтение |
| S\_IWGRP | 0010 | Право на запись |
| S\_IXGRP | 0008 | Право на выполнение |
|  |  |  |
| **S\_IRWXO** | **0007** | **Маска прав остальных пользователей** |

|  |  |  |
| --- | --- | --- |
| S\_IROTH | 0004 | Право на чтение |
| S\_IWOTH | 0002 | Право на запись |
| S\_IXOTH | 0001 | Право на выполнение |

### Система адресации данных:

*Система адресации данных* - это одна из самых существенных составных частей файловой системы. Именно она позволяет находить нужный файл среди множества как пустых, так и занятых блоков на диске.

*В ext2fs*, система адресации реализуется полем ***i\_block*** индексного дескриптора файла. Поле ***i\_block*** в индексном дескрипторе файла представляет собой *массив из 15 адресов блоков*, что наглядно демонстрируется рисунком 1.6.

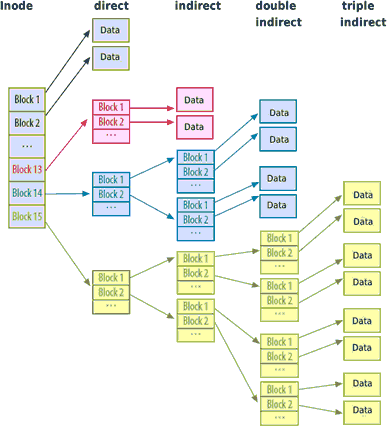


Рисунок 1.6 - Адресация блоков файла с помощью i\_block

*Первые 12 адресов* в этом массиве **i\_block[0 - 11]** = EXT2\_NDIR\_BLOCKS [0 - 11] -

***прямые ссылки*** (адреса) на номера блоков, в которых хранятся данные из файла. *Следующий адрес* **i\_block[12]** = EXT2\_IND\_BLOCK является ***косвенной ссылкой на блок***, в котором хранится ***список следующих адресов*** блоков с данными для

этого файла:

### Количество адресов = (размер\_блока / размер\_ULONG)

*Следующий адрес* **i\_block[13]** = EXT2\_DIND\_BLOCK указывает на ***блок двойной косвенной адресации*** (double indirect block). Этот блок содержит список адресов блоков, которые в свою очередь содержат списки адресов следующих блоков данных того файла, который задается данным индексным дескриптором:

**Количество адресов = (размер\_блока / размер\_ULONG)2**

*Последний адрес* **i\_block[14]** = EXT2\_TIND\_BLOCK указывает на ***блок тройной косвенной адресации***:

**Количество адресов = (размер\_блока / размер\_ULONG)3**

**Все оставшееся место**, в группе блоков, отводится для хранения файлов.

*Таким образом*, прочитав суперблок, мы можем:

* определить адреса групп блоков данных прочитать групповые дескрипторы;
* из групповых дескрипторов определяем положение и размер индексных дескрипторов файлов;
* просматривая таблицу индексных дескрипторов файлов, мы определяем, что описывает дескриптор: файл, каталог, символьное или блочное устройство, другие объекты;
* по дескрипторам каталогов мы читаем блоки данных, в которых записаны имена файлов, каталогов, других устройств и соответствующие номера записей в таблице индексных дескрипторов файлов;
* номера свободных блоков и индексных дескрипторов файлов мы определяет по соответствующим битовым таблицам (картам).

## Сравнение файловых систем

Рассмотренные выше структуры файловых систем ***FAT32*** и ***ext2fs*** демострируют два разных подхода к обработке данных:

* *простоту управления*, характерную для FAT32;
* *скорость обработки больших файлов*, характерную для ext2fs.

*В любом случае*, обе файловые системы имеют общую метаструктуру, которая опре-

деляется наличием:

* *секторов* загрузчика ОС;
* *суперблока*, характеризующего файловую систему и раздел в котором она рас- положена;
* *набора таблиц* для разметки и учета использованных блоков данных;
* *корневого каталога*, с которого начинается логическое построение ФС;
* *набора блоков данных*, в которых размещаются подкатологи и файлы ФС.

*Фактически*, все известные файловые системы отличаются только *способом*

*реализации этих элементов метаструктуры*.

Естественным образом, файловые системы ***FAT32*** и ***ext2fs*** имеют как свои преиму- щества, так и недостатки.

*Преимущества* FAT32:

* *простота реализации* и эффективность использования всего физического пространства блочных устройств малой емкости;
* *широкая известность* и распространенность, делающая ее «*универсальным хранилищем*» временных ФС.

*Недостатки* FAT32:

* практическая ограниченность размера ФС и слабая защищенность, делающая ее непригодной для современных ОС и технологий хранения информации;
* высокая фрагментированность ФС в процессе эксплуатации, что снижает скорость ее работы и негативно воздействует на устройства хранеия данных.

*Именно эти недостатки* потребовали от корпорации Microsoft, ***в марте 1993 года***, замены FAT32 на более сложную и современную ФС: ***ntfs***.

**Файловая система** ext2 берет свое начало от ФС ***minix***. Файловая система ***ext*** - это ***Extended minix***.

*Преимущества* ext2:

* *высокое быстродействие*;
* *поддержка файлов до 2 Гбайт*. *Недостатки* ext2:
* *отсутствие журналирования*, снижающее ее надежность;
* *недостаточные*, по современным меркам, *размеры* поддерживаемых разде- лов ФС и размера файлов.

*Указанные недостатки* ext2 потребовали дальнейшее ее усовершенствование:

* **ext3** — *журналируемая ФС* появилась в ноябре 2001 года; поддерживает раз- мер файлов до 16 Тбайт и размер раздела до 32 Тбайт;
* **ext4** — *журналируемая ФС* появилась в октябре 2008 года; поддерживает размер файлов до 16 Тбайт и размер раздела до 1 Эбайт.

*Журналируемая файловая система* — это ФС, которая сохраняет список измене- ний, проводимых с файловой системой, перед фактическим их осуществлением.

*Записи будущих фактических изменений* хранятся в отдельной части ФС, называе- мой журналом (*journal*) или логом (*log*). Как только изменения файловой системы внесены в журнал, она применяет эти изменения к файлам или метаданным, а затем удаляет эти записи из журнала.

*Записи журнала* организованы в наборы связанных изменений файловой системы.

**Замечание** Журналируемые записи первоначально появились в управлении базами данных, для которых характерно целостное управление большими массивами данных. С помощью их также реализуется *механизм транзакций*.

## Стандартизация структуры ФС

*Предыдущий учебный материал* дает описание блочного устройства на уровне:

* *секторов, LBA и разделов* винчестера: подразделы 1.1 и 1.2;
* *специализированных блоков отдельных ФС*, размещаемых в отдельном разде- ле винчестера: подразделы 1.3 — 1.6;

*Данный учебный материал* рассматривает структуру ФС как организованый набор файлов и директорий, видимый программами (процессами) пользовательского ре- жима работы отдельных ОС.

*В общем случае*, структура ФС на уровне файлов и каталогов может быть произ- вольной, за некоторыми ограничениями, например:

* *обязательно наличие корневой ФС* ОС;
* *обязательно наличие корневого каталога* отдельных ФС.

На рисунке 1.7 приведен пример дерева каталогов и файлов некоторой ФС типа

UNIX.

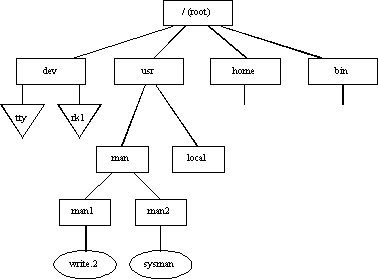


Рисунок 1.7 - Пример структуры ФС типа UNIX

Хорошо видно, что это дерево ФС логически структурировано в прикладном плане, например:

* *dev* — директория размещения узлов устройств ЭВМ (devices);
* *usr* — директория ориентированная на универсальные ресурсы ОС;
* *home* — директория для рабочих (домашних) областей пользователей;
* *bin* — директория расположения исполняемых файлов ОС (binary).

*В августе 1993 года*, в рамках проекта ***GNU***, стал разрабатываться стандарт на структуру файловой системы (*Filesystem Standard*), который был выпущен *в фев- рале 1994 года* и получил обозначение ***FSSTND***.

*Позже, в 1996 году*, была организована группа ***Free Standard Group*** (***FSG***), к кото- рой присоединились и другие разработчики ОС.

***FSG*** продолжила разработку структуры иерархии ФС, которая была бы пригодна для всех UNIX-подобных систем. Такой стандарт получил название ***Filesystem Hierarchy Standard*** (**FHS**).

***В общем случае***, файловая система Linux разделена на *три крупных уровня иерар- хий*. Кратко рассмотрим каждый из этих уровней.

*Первый уровень иерархии ФС* составляют каталоги, которые управляются пользова- телем ***root*** и другими *администраторами системы*.

|  |  |
| --- | --- |
| / | *Корневой каталог*. Должен содержать все, что необходимо для стар- та и запуска ОС, а также на случай восстановления ее из резервной копии. |
| /bin | Необходимые программы (бинарные файлы). |
| /boot | Файлы загрузчика LILO (GRUB) |
| /dev | Устройства (все устройства доступны в виде файлов). |
| /etc | Файлы конфигурации. |
| /etc/X11 | Файлы конфигурации для X Window. |
| /home | Домашние каталоги пользователей. |
| /home/reznik | Домашний каталог пользователя reznik. |
| /lib | Разделяемые библиотеки, необходимые для работы программ, нахо- дящиеся в каталогах ***/bin***, ***/sbin*** и др. |
| /lib/modules | Модули ядра. |
| /media | Директория монтирования ***временных ФС*** (floppy и т.д.). |
| /mnt | Точка для временного монтирования файловых систем. |
| /opt | Каталог для пакетов программ (опциональная установка). |
| /opt/Office50 | Пакет Star Office 5.0 (в данном случае). |
| /proc | *Специальные виртуальные файлы*, обеспечивающие доступ к функ- циям ядра Linux. |
| /root | Домашний каталог пользователя root. |
| /sbin | Необходимые системные программы (бинарные файлы). |

|  |  |
| --- | --- |
| /tmp | *Временные файлы* для всех программ. Эти файлы должны уничто- жаться системой при каждой перезагрузке или периодически. |
| /usr | ***Второй уровень иерархии***. |
| /var | Изменяемые данные (сохраняемые при перезагрузке ОС). |

*Второй уровень иерархии ФС*, по замыслу, содержит *данные, которые доступны только для чтения* и могут использоваться всеми пользователями или несколькими компьютерами сети. В идеале, модифицироваться могут только файлы каталога

***/usr/local***, в который администратор может устанавливать новые программы, не поставляющиеся с системой.

|  |  |
| --- | --- |
| /usr/X11R6 | Система X Window, версия 11.6 |
| /usr/X386 | Система X Window, версия 11.5 |
| /usr/bin | Программы, предназначенные для пользователей. |
| /usr/games | Игры и развлечения. |
| /usr/include | Файлы заголовков для программ на языке С. |
| /usr/lib | Библиотеки. |
| /usr/local | Локальная иерархия (***корень третьего уровня иерархии***). Обычно сюда монтируются всякие дополнения. |
| /usr/sbin | Системные программы. |
| /usr/share | **Архитектурно независимые данные**: |
| /usr/share/dict | Словари. |
| /usr/share/doc | Документация о свободных форматах. |
| /usr/share/games | Игры. |
| /usr/share/info | Документация в формате GNU info. |
| /usr/share/locale | Данные для системной локали. |
| /usr/share/man | Документация в формате **man**. |
| /usr/share/nls | Данные для поддержки ***национальных языков*** (NLS). |
| /usr/share/misc | Все остальные данные. |
| /usr/share/terminfo | База данных terminfo. |
| /usr/share/tmac | Макросы для troff. |
| /usr/share/zoneinfo | База данных и ***конфигурация местного времени***. |
| /usr/src | Исходные тексты (ядро ОС). |
| /usr/src/linux | Исходные тексты ядра Linux. |

*Третий уровень иерархии ФС* — *подобен второму* и продолжается в директории

#### /usr/local.

**Далее**, более подробно рассмотрим корневую директорию изменяемых данных

***/var***, поскольку она содержит множество системных данных ОС.

|  |  |
| --- | --- |
| /var | *Каталог*, содержимое которого может меняться при работе сис- темных программ. |
| /var/cache | Временно сохраняемые файлы различных программ (браузеров Internet, программы man, сервера шрифтов и другие). |
| /var/games | Изменяемые файлы для игр из ***/usr/games***. |
| /var/lock | Lock-файлы, указывающие, что то или иное устройство (обычно модем) занято. |
| /var/log | Системные журнальные файлы. |
| /var/mail | Почтовые ящики пользователей. |
| /var/opt | Изменяемые файлы для программ из **/opt**. |
| /var/run | Файлы, имеющие отношение к запущенным в настоящий мо- мент программам, например, хранящие номера процессов. |
| /var/spool | *Данные, обработка которых отложена* (статьи из групп новос- тей, документы посланные на печать, письма, которые не уда- лось сразу послать из-за проблем со связью и другие). |
| /var/state | Данные, имеющие отношение к состоянию программ, напри- мер, **backup**-файлы, файлы текстовых редакторов, конфигура- ции, имеющие отношение ко всей системе и прочее. |
| /var/tmp | Временные файлы, которые ***не должны*** регулярно уничтожать- ся системой. |
| /var/yp | Файлы системы ***NIS*** (Network Information Services). |

### Замечание

Видимое пользователям дерево ФС отличается от дерева ФС доступного ядру ОС:

* ядро ОС «видит» дерево виртуальной ФС (VFS);
* пользователь «видит» только ветвь дерева, объявленного для пользователя как корень root;
* современная тенденция контейнерная технология склоняется к минимизации ви- димой части ФС, повышая ее защищенность.

*Управление видимостью дерева ФС* осуществляется с помощью утилиты ***chroot***. В частности, она используется во время выполнения процедуры ***login***.

*С другой стороны*, видимая для пользователя ФС, создает для него *иерархическую систему координат*, в пределах которой он принципиально может адресовать любой файл.

*Следующий аспект работы пользователя* состоит в том, что он всегда находится *в некоторой точке ФС*, которая связана с некоторой ее директорией:

* чтобы определить, где находится пользователь, следует использовать команду

***pwd***, которая в качестве результата возвратит имя директории;

* чтобы перемещаться по ФС, пользователь должен использовать команду:

**cd [директория],**

где аргумент команды обрабатывается по следующим правилам:

* отсутствие аргумента *делает текущей директорию*, заданную системной переменной **HOME**;
* аргумент, начинающийся символом **/**, *задает абсолютные координаты ди- ректории*, начиная от корня ФС;
* аргумент, начинающийся любым допустимым символом, отличным от **/**, задает *относительные координаты директории*, начиная от текущей дирек- тории;
* *для уточнения* использования относительной адресации, можно использовать аргумент ***./директория***;
* *для относительной адресации* от родительской директории, можно использо- вать аргумент ***../директория***

*Наиболее распространенные команды* для работы с директориями:

* **mkdir директория;** - создание директории;
* **rmdir директория;** - удаление директории;
* **ls [опции] [шаблон];** - вывод на консоль списка файлов и директорий;
* **cat файл;** - вывод содержимого файла на консоль.

Например,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $ ls -l |  | | | |
| итого 864 |
| drwxrwxrwx | 5 root root | 4096 окт. 6 | 2013 | asu11upk3 |
| drwxrwxrwx | 3 root root | 4096 июля 28 | 2013 | asu12upk |
| -rw-rw-r-- | 1 vgr vgr | 18 дек. 22 | 2012 | asu\_tmp |
| drwxrwxr-x | 2 vgr vgr | 4096 июня 1 | 02:32 | bin |
| drwxrwxr-x | 2 vgr vgr | 4096 дек. 26 | 2012 | bin.old |
| -rw-r--r-- | 1 vgr vgr | 5860 янв. 5 | 2013 | casper.log |
| -rw-rw-r-- | 1 vgr vgr | 180 сент. 18 | 2012 | demo2.txt |
| -rw-rw-r-- | 1 vgr vgr | 490 сент. 18 | 2012 | demo.txt |
| … |  |  |  |  |
| $ |  |  |  |  |

Кроме специального назначения директории **/dev**, о которой говорилось ранее, ФС имеет директорию **/proc**, в которую монтирована файловая система **proc**. В ней со- держатся множество файлов, содержащих информацию о параметрах или функцио- нальных возможностях ядра ОС.

Например, чтобы узнать, какие типы ФС поддерживает ядро ОС, можно выпол- нить:

$ ***cat /proc/filesystems***

nodev sysfs

nodev nodev nodev nodev nodev nodev nodev nodev nodev nodev nodev nodev nodev

nodev nodev

nodev

nodev nodev nodev nodev

$

rootfs bdev proc cgroup cpuset tmpfs devtmpfs debugfs securityfs sockfs pipefs

anon\_inodefs devpts

ext3 ext4 ramfs

hugetlbfs vfat ecryptfs fuseblk fuse fusectl pstore mqueue

Данный пример хорошо показывает, что *не все типы ФС* подразумевают наличие

«физических» (***блочных***) устройств.

Другой пример, - файл ***/proc/cmdline***, который содержит список параметров, пере- данных ядру во время загрузки ОС:

$ ***cat /proc/cmdline***

BOOT\_IMAGE=/boot/vmlinuz-3.5.0-21-generic root=/dev/sda5 ro

$

**Замечание**

Последний пример - чисто демонстрационный. Студенту самостоятельно следует опре- делить и описать содержимое файла /proc/cmdline.

## Модули и драйверы ОС

**Ядро ОС** — это большая программа, работающая в собственном *защищенном (привелигированном) режиме*.

**Любой процесс** работает в *режиме пользователя*. Потому, *любой процесс имеет доступ к устройствам только через ядро ОС*.

Чтобы работать с конкретным устройством, ядро ОС имеет специальное ПО, кото- рое называется *драйвером*.

*ПО драйвера* имеет два интерфейса:

* *интерфейс вызова драйвера ядром ОС*, который использует концепцию узла ФС или *node*, подразделяя устройства на *символьные* и *блочные*;
* *интерфейс устройства*, который использует сам драйвер для работы с конк- ретным устройством *через общую шину ЭВМ*.

*Принципиальная разница* между этими интерфейсами:

* *интерфейс вызова драйвера* ядром ОС *определяется разработчиками ОС* и различается от ее типа: разный интерфейс для разных ОС;
* *интерфейс устройства* определяется *архитектурой аппаратной части ЭВМ*

и *конструктивными особенностями* конкретного устройства.

*Учитывая большое разнообразие* архитектур аппаратной части ЭВМ, *драйверы традиционно пишутся в виде модулей*, которые хранятся в ФС ОС и *загружаются по мере необходимости*.

Поскольку без некоторой части драйверов ОС не сможет даже загрузиться, такие драйверы включаются в ядро ОС ***статически***. Обычно, перед компиляцией ядра ОС, запускается **программа-конфигуратор**, с помощью которой определяется: какие драйверы будут включаются в ядро ***статически***, а какие будут подгружаться ***динамически:*** во время загрузки ОС или по мере необходимости.

**Идея использования модулей ядра** — очень привлекательна:

* *для разработки любого ПО* используются средства разработки, которые *сами являются ПО* и могут функционировать *только в среде уже загруженной ОС*;
* *статически скомпилированный драйвер* может вступить в конфликт с архи- тектурой аппаратной части ЭВМ, что может сделать ядро ОС *неработоспо- собным*;
* *стремление уменьшить размер ядра ОС*, которое является актуальным для *встроенных (embedded) систем* или в *проектах микроядерной архитектуры ОС*.

**Для создания** *модулей в ОС Linux* разработана специальная технология, которая широко освещена в соответствующей литературе. Например, корпорация IBM на сайте <http://www.ibm.com/developerworks/ru/library/l-linux_kernel_01/>выпустила бо- лее 70-ти статей, посвященных этой тематике.

*В общем случае*, **модуль** — *системное ПО ОС*, которое не обязательно является *драйвером*. Главное, что это ПО предназначено для работы в *защищенном режиме пространства ядра ОС*.

*Использование модулей* предполагает учет их особенностей:

* *управление модулями* из режима пользователя осуществляется утилитами

***insmod***, ***rmmod***, ***lsmod***, ***modinfo*** и ***modprobe***;

* *разработка модулей* предполагает знание не только технологии их написания и отладки, но также - *архитектуры* и *ПО ядра* ОС;
* *вывод информации в пространство пользователя* модуль осуществляет пос- редством функции ***printk(...)***, которая записывает сообщение в файл */var/log/ messages*;
* *для просмотра сообщений модулей ядра* используется утилита ***dmesg***;
* *файлы конфигурации и сами модули* находятся в директориях ФС: */etc* и

*/lib/modules*.

## Системные вызовы ОС по управлению устройствами и ФС

*Один из общепринятых способов повышения мобильности ПО* - обеспечение стан- дартизации окружения приложений. Это подразумевает стандартизацию *ПО ОС*, *утилит* и *программных интерфейсов* самих приложений. Таким средством явля- ется стандарт **POSIX** (*Portable Operating System Interface*), который описывает раз- личные интерфейсы операционной системы.

**Название POSIX** предложено известным специалистом, основателем Фонда сво- бодного программного обеспечения, Ричардом Столмэном. Наиболее современная версия стандарта POSIX, ***в редакции 2003 г.***, основана на Техническом стандарте *Open Group IEEE Std 1003.1* и на международном стандарте *ISO/IEC 9945*.

*По состоянию на 2001 год*, стандарт содержал следующие *четыре части*:

* + 1. Основные определения (термины, концепции и интерфейсы, общие для всех частей);
    2. Описание прикладного программного C-интерфейса к системным сервисам;
    3. Описание интерфейса к системным сервисам на уровне командного языка и служебных программ;
    4. Детальное разъяснение положений стандарта, обоснование принятых реше- ний.

*В дальнейшем*, накапливались и были внесены многие мелкие исправления, учтен- ные *в редакции 2003 года*.

**Стандарт POSIX** - обязательный элемент современной дисциплины разработки прикладных систем.

*Таким образом*, рассмотрев основные понятия и требования стандарта POSIX, мы можем перейти к непосредственному обсуждению системных вызовов ОС. На ри- сунке 1.9, приведен список системных вызовов ОС, разделенных на группы:

* управление процессами;
* управление файлами;
* управление каталогами и файловыми системами;
* разные, которые нельзя полностью отнести к предыдущим группам.

*Непосредственно к нашей теме*, относятся группы *управление файлами*, а также

*управление каталогами и файловыми системами*.

*Примеры предыдущего раздела* показывают, что зная структуру необходимых дан- ных, можно одни функции реализовать через другие, «более мелкие», но общая па- радигма системного ПО требует отношение к ним как к функциям одного уровня.

**Замечание** Непосредственное применение системных вызовов предполагает изучение правил их вызова с использованием утилиты ***man***, которая, собственно говоря, и была создана для этих целей.

Наряду с низкоуровневыми системными вызовами, для работы с файлами и файловыми системами, используются более высокоуровневые функции, которые обеспечивают буферизованный ввод-вывод и подробно изложены в любом учебнике по языку ***С***.

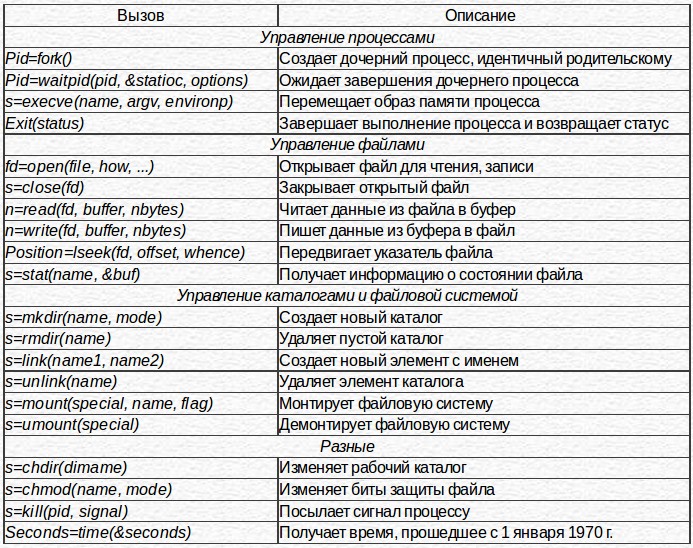


Рисунок 1.9 - Системные вызовы стандарта POSIX

## Три концепции работы с устройствами

*Для того чтобы*, чтобы уже созданные ФС стали доступны приложениям ОС,

*должны быть созданы узлы* (*node*), которые расположены в директории /dev.

*Узлы ОС* могут быть созданы из пространства пользователя:

* *с помощью системного вызова* ***mknod(...)*** из программы на языке ***С***;
* *с помощью утилиты* ***mknod*** посредством команды языка ***shell***.

Хотя любой из указанных способов предполагает указание всего четырех парамет- ров - ***имя\_узла***, ***тип\_узла***, ***старший\_номер\_узла*** и ***младший\_номер\_узла***, такая ситуация создает множество проблем:

* *какое имя дать узлу*, чтобы оно мнемонически отображало *тип устройства*

и было не очень длинным, для удобства работы с ним;

* *какой старший номер* присвоить устройству, чтобы он соответствовал типу устройства и *был стандартным для этого типа устройств*;
* *сколько должно быть младших номеров устройства*, чтобы удовлетворить все варианты использования ОС;
* *что делать с узлами устройств*, которых нет в архитектуре ЭВМ;
* *как узнать момент*, когда к ЭВМ устройство подключается динамически (*временное подключение и отключение устройств*).

*Исторически, сложилось три концепции (технологии)* именования и отслеживания актуальности наличия устройств: ***manual***, ***udev*** и ***systemd***.

*Концепция manual (ручная)* появилась первой. Она предполагает использование, упомянутых выше, системного вызова ***mknod(...)*** или утилиты ***mknod***.

*Обычно*, созданием устройств занимается *программа-инсталлятор ОС*, которая проверяет архитектуру компьютера на наличие имеющихся устройств, а также создает узлы устройств: автоматически или по требованию администратора ОС. *Первоначально*, именование и параметры узлов придумывались разработчиками ОС и были достаточно разнообразны. Со временем, стала проводиться стандарти- зация, учитывающая известные типы оборудования. Например:

* *почти все ОС* имеют устройство консоли **/dev/console**;
* *винчестеры, с контроллерами* ***ATA***, стали именоваться **/dev/hda**, **/dev/hdb** и далее, а разделы, на которых фактически и создаются ФС - **/dev/hda1**,

**/dev/hda2**, …, **/dev/hdb1**, **/dev/hdb2**, …

* *разделы винчестеров, с контроллерам* ***SATA или SCSI***, стали именоваться -

**/dev/sda1**, **/dev/sda2**, …, **/dev/sdb1**, **/dev/sdb2**, ...

*Окончательно*, процесс стандартизации завершился созданием скрипта shell — **MAKEDEV**, который упорядочил использование имен и других параметров узлов. Часто, найти этот скрипт можно в директории **/dev** или **/sbin**.

**Замечание** Используя скрипт **MAKEDEV**, можно легко создать узлы с корректными именами и пара- метрами, но невозможно решить проблемы эффективного их использования. Особенно

остро этот вопрос встал, когда корпорация Microsoft стала интенсивно призвать произ- водителей оборудования внедрять *технологию Plug and Play*.

Принципиальный аспект этой проблемы состоит в том, что драйвера устройств работают в пространстве ядра ОС. И хотя они могут определить факт подключения к ЭВМ нового устройства, *они не предназначены для посылки сигналов процессам*, работающим в режиме пользователя.

**С начала 2000-х годов**, в технологиях ОС, стала применяться файловая система

*devfs*.

**Цель проекта** - решить основную проблему ОС UNIX/Linux, связанную как с катастрофически возрастающим объемом директории */dev*, так и с поддержкой технологии *PnP*.

Данный проект предполагал использование специального демона *devfsd*, который периодически обращался к ядру ОС с целью контроля за состоянием имеющегося оборудования:

* *при обнаружении подключения* нового оборудования, этот демон создавал соответствующий узел (inode) в директории */dev*;
* *в случае обнаружения ситуации*, что оборудование отключилось, он соответ- ственно удалял узел из директории */dev*.

**Существенным недостатком** *devfs* является *неэффективное расходование ресур- сов ЭВМ*, связанное с его постоянным циклическим запуском, поэтому современ- ные системы ***ОС UNIX/Linux*** перешли на более новую технологию, известную как *udev*, используя для директории **/dev** файловую систему *devtmpfs*.

**Концепция udev**, была реализована ***в ноябре 2003 года***. Она решила проблему внедрения *технологии Plug and Play*.

**Замечание** *Здесь также следует отметить*, что широкое внедрение графических оболочек ОС потребовало организовать *эффективный обмен сообщениями между демонами (фоно- выми процессами) рабочего стола*.

Для этого была разработана *технология D-Bus* - специальная система межпроцессного взаимодействия, которая предоставляет *несколько шин (логических шин)*:

* *системную шину*, которая создается *при старте демона D-Bus* и обеспечивает взаимодействие демонов системных процессов;
* *сессионную шину*, которая создается для каждого пользователя, авторизовавше- гося в системе, и обеспечивает взаимодействие его личных процессов.

**Таким образом**, при старте ОС *запускается* ***демон udevd***, который, используя шину D-Bus, *отслеживает подключение новых устройств и создает для них нуж- ные узлы*.

Когда устройства отключается от ЭВМ, ***udevd*** удаляет соответствующие им узлы, кроме тех, которые были созданы в ручную.

**Одновременно**, также решается *проблема поддержания в активном состоянии неиспользованных узлов*.

**С апреля 2010 года**, применительно к ОС Linux, стала внедряться «*Концепция systemd*».

*Ее цель* — ***унификация*** взаимодействия между процессами режима пользователя.

*В частности*:

* *отслеживание динамики* подключения устройств (вместо ***udevd***);
* *эффективное отслеживание групп процессов*, порожденных системными вызовами ***fork(...)*** и объединенных одной прикладной целью **(*cgroups*).**

**Замечание** *ОС УПК АСУ*, для работы с узлами устройств, использует эту современную концепцию systemd.

## Разделы дисков и работа с ними

**Когда созданы все ФС**, на всех разделах блочных устройств, работа с файловой системой ОС становится достаточно простой:

* *необходимо определить точки монтирования* (директори) корневой ФС ОС для подключения к ним (монтирования) нужных сопуствующих ФС и затем,

правильно записать желаемую конфигурацию в файле /etc/fstab;

* *выполнить команду* ***mount -a***, и желаемая структура ФС ОС будет создана для последующей эксплуатации.

*Хотя такой подход требует определенной квалификации* пользователя и ориенти- рован больше на администраторов серверов, он вполне приемлем для многих случаев, когда аппаратная конфигурация ЭВМ является стабильной и не требует серьезных обновлений.

*Ситуация обостряется и становится критичной*, когда необходимо провести мо- дернизацию свзанную с:

* *динамическим подключением других* («не родных») ФС;
* *подключением к сетевым ресурсам данных*, которые кроме проблем самой сети вынуджают использовать ФС, построенные на иных концепциях, напри-

мер, ФС типа ***FAT32*** и ***ntfs***, которые не используют атрибутов пользователя, группы и других.

*В общем случае*, эти проблемы решаются разработкой и использованием необходи- мых модулей ядра ОС, что уже было рассмотрено в подразделе 1.8. Но поскольку структура ФС достаточно сложна, а модули подключаются к ядру ОС, то:

* *размер ядра ОС* значительно увеличивается;
* *время разработки и последующей отладки ПО* модулей становится неприем- лемо большим.

*В 1985 году*, компания Sun Microsystems предложила для ОС SunOS: ***VFS*** — Virtual File System (File System Switch) - *Виртуальнаую Файловую Систему*.

### Виртуальная файловая система VFS:

*Суть идеи* — расположить VFS между *приложениями* и *конкретными файловыми системами на внешних носителях*.

Это позволяет *сконцентрировать в одном месте* пользовательский интерфес для разных типов ФС, что *создает утифицированный эталон для разработки драйве- ров ОС*.

**Таким образом**, **VFS** обеспечивает *унифицированный программный интерфейс* к услугам файловой системы, причем безотносительно к тому, какой тип файловой системы (vfat, ext2fs, nfs и другие) имеется на конкретном физическом носителе.

*В результате*, VFS:

* *упрощает процесс создания драйверов ФС*, что повышает их надежность;
* *сокращает время разработки*, что создает конкурентные преимущества са- мой ОС;
* *уменьшает время отладки и исправления ошибок*, что повышает актуаль- ность их применения.

**В любом случае**, возможность применения тех или иных типов ФС зависит как от потребностей самих вычислительных систем, реализованных на ПО ОС, так и от возможности правильного управления имеющимися ФС.

**Все равно**, основу уравления ФС составляют утилиты ***mount*** и ***umount***, которые мы рассмотрим в следующем подразделе.

## Монтирование и демонтирование устройств

**Монтирование файловой системы** — *подключение конкретного раздела* внешне- го блочного устройства *к конкретному каталогу ФС*, в видимой пользователю общей части дерева корневой ФС.

**Демонтирование файловой системы** — *отключение конкретного раздела* внеш- него блочного устройства от *конкретного каталога ФС*, в видимой пользователю общей части дерева корневой ФС.

**Общее правило** монтирования/демонтирования ФС:

* *монтирование осуществляется командой* ***mount*** с указанием узла ФС и ди- ректории монтирования, при этом: *старое содержимое директории стано- вится невидимым, а монтируемая ФС — видимой*;
* *демонтирование осуществляется командой* ***umount*** с указанием или узла ФС или директории монтирования, при этом: *старое содержимое директории становится видимым, а демонтируемая ФС — невидимой*;

***Простейший общий вид команды*** монтирования:

**mount -t Тип Устройство Директория;**

**Например**, монтирование устройства ***/dev/sda2***, типа ***ext2*** в каталог ***/mnt***, выглядит так:

**$ mount -t ext2 /dev/sda2 /mnt**

**$**

Команда **mount***, без параметров*, выдает список монтированных в данный момент устройств:

|  |  |  |
| --- | --- | --- |
| /dev/sda5 | on / | type ext4 (rw,errors=remount-ro) |
| proc | on /proc | type proc (rw,noexec,nosuid,nodev) |
| sysfs | on /sys | type sysfs (rw,noexec,nosuid,nodev) |
| none on /sys/fs/fuse/connections type fusectl (rw)  …....  /dev/sda2 on /mnt type ext2 (rw) | | |

Команда **umount***, для указанного примера*, может быть выполнена в двух вариан- тах:

**umount /dev/sda2; umount /mnt;**

**Замечание**

Информация о действиях команды mount оперативно заносится в файл **/etc/mtab** в формате:

**Устройство on Каталог type Тип (Опции)**

**Для централизованного управления** *монтированием ФС*, используется файл /etc/ fstab. Структура файла имеет вид:

**Устройство Каталог Тип Опции fs\_freq fs passno**

где

***fs\_freq* —** определяет, должна ли создаваться резервная копия этого раздела с помощью команды ***dump***; значение **0** отменяет ***dump***;

***fs passno* —** указывает в какую очередь данная ФС проверяется на целостность, при запуске Linux: 0 — проверка не требуется; 1 — для корневой ФС; 2 — для других ФС.

Демонстрационный пример **/etc/fstab**:

# /etc/fstab: static file system information. #

# Use 'blkid' to print the universally unique identifier for a

# device; this may be used with UUID= as a more robust way to name devices # that works even if disks are added and removed. See fstab(5).

#

# <file system> <mount point> <type> <options> <dump> <pass> # / was on /dev/sda2 during installation

/dev/sda4 / ext2 errors=remount-ro 0 1

# swap was on /dev/sda6 during installation

|  |  |  |  |
| --- | --- | --- | --- |
| /dev/sda6 none swap | sw | 0 | 0 |
| /dev/fd0 /media/floppy0 auto | rw,user,noauto,exec,utf8 | 0 | 0 |

**Замечание** Исторически, ***опции***, с которыми монтируются ФС, создавались разработчиками разных ФС. Такая ситуация требует отдельного изучения опций для каждой ФС.

Кроме того, ***/etc/fstab*** имеет свои специфические опции.

*Набор наиболее распространенных опций*:

|  |  |
| --- | --- |
| async | Весь ввод/вывод осуществляется асинхронно. |
| sync | Весь ввод/вывод осуществляется синхронно. |
| atime | Обновлять времена обращения (по умолчанию). |
| noatime | Времена обращения не обновляются. |
| auto | Система ***может*** быть смонтирована с опцией **-а (для всех).** |
| noauto | Система ***не может*** быть смонтирована с опцией **-а (для всех).** |
| defaults | Аналог ***rw,suid,dev,exec,auto,nouser,asunc.*** |
| dev | Файлы устройств ФС интерпретируются как устройства . |
| nodev | Файлы устройств запрещены. |
| exec | Право на запуск исполняемых файлов. |
| noexec | Запрет на запуск исполняемых файлов. |
| suid | Биты SUID и SGID ***действуют.*** |
| nosuid | Биты SUID и SGID **не *действуют.*** |
| user | Обычный пользователь может смонтировать систему. Подразумевает опции: ***noexec,nosuid,nodev.*** |
| nouser | Только **root** может монтировать систему. |
| remount | Перемонтировать уже смонтированную систему (например, для измене- ния опций системы). |
| ro | Смонтировать ФС только для чтения. |
| rw | Смонтировать ФС для чтения и записи. |

## Файловые системы loopback, squashfs, overlayfs и fuse

Наряду с рассмотренными выше ФС на разделах дисков и в ядре ОС, используется множество специальных ФС. Мы рассмотрим четыре из них: *loopback*, *squashfs*, *overlayfs* и *fuse*.

*Термин* ***loopback*** означает - «обратная петля». Применительно к нашей тематике он будет означать:

* *циклическое устройство*, когда мы будем говорить о блочном устройстве;
* *циклическая ФС*, когда мы будем говорить о файловой системе, которая соз- дана в отдельном файле.

*Узлы циклических устройств* находятся в директории **/dev**. Следующий пример показывает вариант создания десяти циклических устройств: ***/dev/loop1***, …,

#### /dev/loop9.

for x in 0 1 2 3 4 5 6 7 8 9; do

[ -e /dev/loop$x ] || mknod /dev/loop$x

done

b

5 $x

*Циклические ФС* обычно создаются в файлах, которые могут быть *сжатыми* и

*зашифрованными*. Для примера рассмотрим создание ФС типа ***ext2fs*** в файле

*/tmp/ramdisk* и последующее монтирование созданной ФС в директорию */mnt/initrd*:

# Используемые константы BLKSIZE=1024 # Размер блока в байтах RDSIZE=4000 # Количество блоков в ФС RBLOCK=0 # Число резервных блоков

# Создаем файл, забитый нулями

# (dd — преобразование и копирование файлов)

dd if=/dev/zero of=/tmp/ramdisk bs=$BLKSIZE count=$RDSIZE

# Создаем ФС в файле ext2fs

/sbin/mke2fs -F -m $RBLOCK -b $BLKSIZE /tmp/ramdisk $RDSIZE

# Монтируем ФС в файле к директории через устройство /dev/loop0

mount /tmp/ramdisk /mnt/initrd -t ext2 -o loop=/dev/loop0

*Широкое распространение* получила технология хранения файловых систем в *сильно сжатых файлах* для *live-дистрибутивов*. Такой подход позволяет практи- чески на прямую использовать технологии созданные для работы с ФС на CD дисках. Например, если устройство ***/dev/sda1*** содержит файловую систему ***ntfs***, в которой имеется файл */asu64upk/upkasu/usrfs.sfs*, то монтирование такой ФС (типа ***squashfs***) к директории */usr* осуществляется двумя командами:

# Монтируем устройство /dev/sda1 в директорию /run*/*basefs mount /dev/sda1 /run*/*basefs -t ntfs -o rw

# Монтируем ФС в файле к директории /new\_root через устройство

/dev/loop1

mount /run/basefs/asu64upk/upkasu/usrfs.sfs /usr \

-t squashfs -o loop=/dev/loop1

**Другая специальная ФС** - **overlayfs** — является вариантом реализации *каскадной файловой системы* для ОС ***Ubuntu*** и ее клонов.

*Каскадная файловая система (КФС)* — это виртуальная ФС, позволяющая

*«прозрачно видеть» две изолированные ФС как одну*.

*Работы над КФС* ведутся ***с середины 90-х годов***. ***В январе 2007 года***, в некоторые

дистрибутивы Linux, была включена КФС, названная *UnionFS*. Альтернативная разработка *Aufs (AnotherUnionFS)* ведется ***с 2006 года***.

**Каскадно-объединенное монтирование** — *одновременное монтирование разных нескольких файловых систем как одну*, например для объединения ФС нескольких сайтов.

***Overlayfs*** обеспечивает надежное монтирование двух файловых систем на разных уровнях. Рассмотрим подробнее этот вопрос на примере:

* пусть устройство CDROM монтировано в директорию **/cdrom** с возмож- ностью ***только чтение***;
* пусть в директорию **/mnt/initrd** монтирован, созданный ранее ***ramdisk***, кото- рый имеет возможность ***чтения и записи***;
* проведем каскадно-объединенное монтирование в директорию **/tmp/mnt**, предполагая, что ***CDROM*** будет монтирован на нижнем уровне, а ***ramdisk*** — на верхнем;
* учтем, что в ядре ОС имеется имя **overlayfs**, которое не является устрой- ством, но позволяет монтировать ФС.

*Тогда*, команда монтирования будет иметь вид:

# Команда каскадно-объединенного монтирования mount -t overlay overlay /tmp/mnt -o \ rw,upperdir=/mnt/initrd,lowerdir=/cdrom

**В результате** такого действия, в директории /tmp/mnt, мы будем видеть все файлы директории /cdrom.

**Кроме того**, мы можем редактировать и использовать любой файл директории

/cdrom, но сохраняться эти файлы будут в памяти *ramdisk*.

**Последня специальная ФС**, которую мы кратко рассмотрим — ***FUSE***.

**FUSE** — *Filesystem in Userspace* — файловая система в пространстве пользователя. *FUSE* реализована в виде модуля для UNIX-подобных ОС. Этот модуль *позволяет пользователям без привилегий создавать свои ФС*, без необходимости переписы- вать код ядра ОС. Это достигается за счет запуска кода файловой системы в пространстве пользователя, в то время как *модуль FUSE предоставляет мост* для актуальных интерфейсов ядра.

**На рисунке 1.8** показана схема взаимодействия системных вызовов пользователей с ядром ОС и ПО в пространстве пользователя. Хорошо видно, что:

* *схема взаимодействия* системного и прикладного ПО напоминает аналогич- ное взаимодействие ***микроядра*** и ***серверных сервисов***;
* *кроме системной библиотеки* ***glibc*** необходима библиотека ***libfuse***.

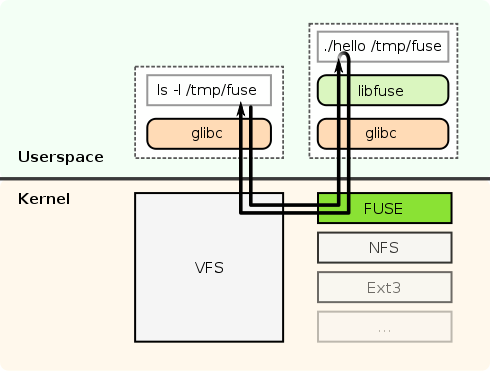


Рисунок 1.8 - Схема работы модуля FUSE

*Для операций монтирования и демонтирования* используется специальная утилита

***fusermount***, использующая файл конфигурации */etc/fuse.conf*:

### fusermount [OPTIONS] MOUNTPOINT

*Опция* ***-u*** используется для демонтирования FUSE.

*Следует заметить*, что многие видные разработчики ОС критикуют FUSE, но эти вопросы выходят за рамки нашей дисциплины.

### Замечание

Разделы винчестера, содержащие ФС ***ntfs***, монтируются в Linux с типом ***fuseblk***.

## Дисковые квоты

**В заключение темы**, рассмотрим вопросы ограничения размеров, которые систем- но можно накладывать на файловые системы.

*Рано или поздно может случиться так*, что некоторая ФС на внешнем носителе, подключенная к древовидной структуре ядра ОС, *заполнится и блокирует нор- мальную работу ОС*.

*По умолчанию*, ФС ***ext2fs*** резервирует ***5%*** места для пользователя ***root***.

*Принято считать*, что для нормальной работы ФС требуется ***свободное прост- ранство***:

* *не менее 10%* дискового пространства (пространство раздела);
* *не менее наибольшего файла* в файловой системе.

*Рекомендуется* размешать ***/home, /opt, /tmp, /var, /usr/local*** в разных разделах диска.

*Определить* свободное пространство диска можно командой ***df***:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| $ df  Файл.система | 1K-блоков | Использовано | Доступно | Использовано% | Cмонтировано в |
| /dev/sda5 | 21356772 | 14841812 | 5430084 | 74% | / |
| udev | 1669124 | 12 | 1669112 | 1% | /dev |
| tmpfs | 670748 | 780 | 669968 | 1% | /run |
| none | 5120 | 0 | 5120 | 0% | /run/lock |
| none | 1676860 | 84 | 1676776 | 1% | /run/shm |
| none | 102400 | 12 | 102388 | 1% | /run/user |
| /dev/sda1  $ | 51199120 | 26771180 | 24427940 | 53% | /cdrom |

*Администратор ОС* должен отслеживать наличие достаточного дискового прост- ранства ФС. Для этих целей имеется специальное системное ПО под общим названием **quota**. Естественно, ядро ОС должно поддерживать команды этого пакета.

*Чтобы включить квоты* для конкретной файловой системы, нужно в корень ФС поместить бинарные файлы:

* *quota.user* — для персональных квот;
* *quota.group* — для групповых квот.

*Управление квотами* выполняется с помощью команд:

|  |  |
| --- | --- |
| quotastats | Проверка поддержки квот ядром ОС. |
| quotacheck | Сканирование заданной ФС и первоначальное создание файлов  *quota.user и quota.group*. |
| edquota | Редактор квот. |
| quotaon | Активация настроек квот для ФС. |
| quotaoff | Деактивация настроек квот для ФС. |
| quota | Проверка пользователем, установленных для него квот. |