# ФАЙЛОВЫЕ СИСТЕМЫ UNIX

26.04.2011
11:19:03

# Файловые системы UNIX System V Release 4



В UNIX System V Release 4 реализован механизм виртуальной файловой системы VFS (Virtual File System), который позволяет ядру системы одновременно поддерживать несколько различных типов файловых систем.

Типы файловых систем, поддерживаемых в UNIX System V Release 4:

- *s5* - традиционная файловая система UNIX System V, поддерживаемая в ранних версиях UNIX System V от AT&T;
- *ufs* - файловая система, используемая по умолчанию в UNIX System V Release 4, которая ведет происхождение от файловой системы SunOS, которая в свою очередь, происходит от файловой системы Berkeley Fast File System (FFS);
- *nfs* - адаптация известной файловой системы NFS фирмы Sun Microsystems, которая позволяет разделять файлы и каталоги в гетерогенных сетях;
- *rfs* - файловая система Remote File Sharing из UNIX System V Release 3. По функциональным возможностям близка к NFS, но требует на каждом компьютере установки UNIX System V Release 3 или более поздних версий

# Файловые системы UNIX System V Release 4



- *Veritas* - отказоустойчивая файловая система с транзакционным механизмом операций;
- *specfs* - этот новый тип файловой системы обеспечивает единый интерфейс ко всем специальным файлам, описываемым в каталоге /dev;
- *fifofs* - эта новая файловая система использует механизм VFS для реализации файлов FIFO, также известных как конвейеры (pipes), в среде STREAMS;
- *bfs* - загрузочная файловая система. Предназначена для быстрой и простой загрузки и поэтому представляет собой очень простую плоскую файловую систему, состоящую из одного каталога;
- */proc* - файловая система этого типа обеспечивает доступ к образу адресного пространства каждого активного процесса системы, обычно используется для отладки и трассировки;
- */dev/fd* - этот тип файловой системы обеспечивает удобный метод ссылок на дескрипторы открытых файлов.

# Традиционная файловая система s5

# Типы файлов



Файловая система UNIX s5 поддерживает логическую организацию файла в виде последовательности байтов. По функциональному назначению различаются обычные файлы, каталоги и специальные файлы.

- *Обычные файлы* содержат ту информацию, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ, то есть ОС не накладывает никаких ограничений на структуру и характер информации, хранимой в обычных файлах.
- *Каталог* - файл, содержащий служебную информацию файловой системы о группе файлов, входящих в данный каталог. В каталог могут входить обычные, специальные файлы и каталоги более низкого уровня.
- *Специальный файл* - фиктивный файл, ассоциируемый с каким-либо устройством ввода-вывода, используется для унификации механизма доступа к файлам и внешним устройствам.

# Структура файловой системы



- Файловая система s5 имеет иерархическую структуру, в которой уровни создаются за счет каталогов, содержащих информацию о файлах более низкого уровня.
- Корневой каталог файловой системы всегда располагается на системном устройстве (диск, имеющий такой признак). Однако это не означает, что и все остальные файлы могут содержаться только на нем. Для связи иерархий файлов, расположенных на разных носителях, применяется монтирование файловой системы, выполняемое системным вызовом mount.
- Операция монтирования заключается в следующем: в корневой файловой системе выбирается некоторый существующий каталог, содержащий один пустой файл. После выполнения монтирования выбранный каталог становится корневым каталогом второй файловой системы. Через этот каталог смонтированная файловая система подсоединяется как поддерево к общему дереву.

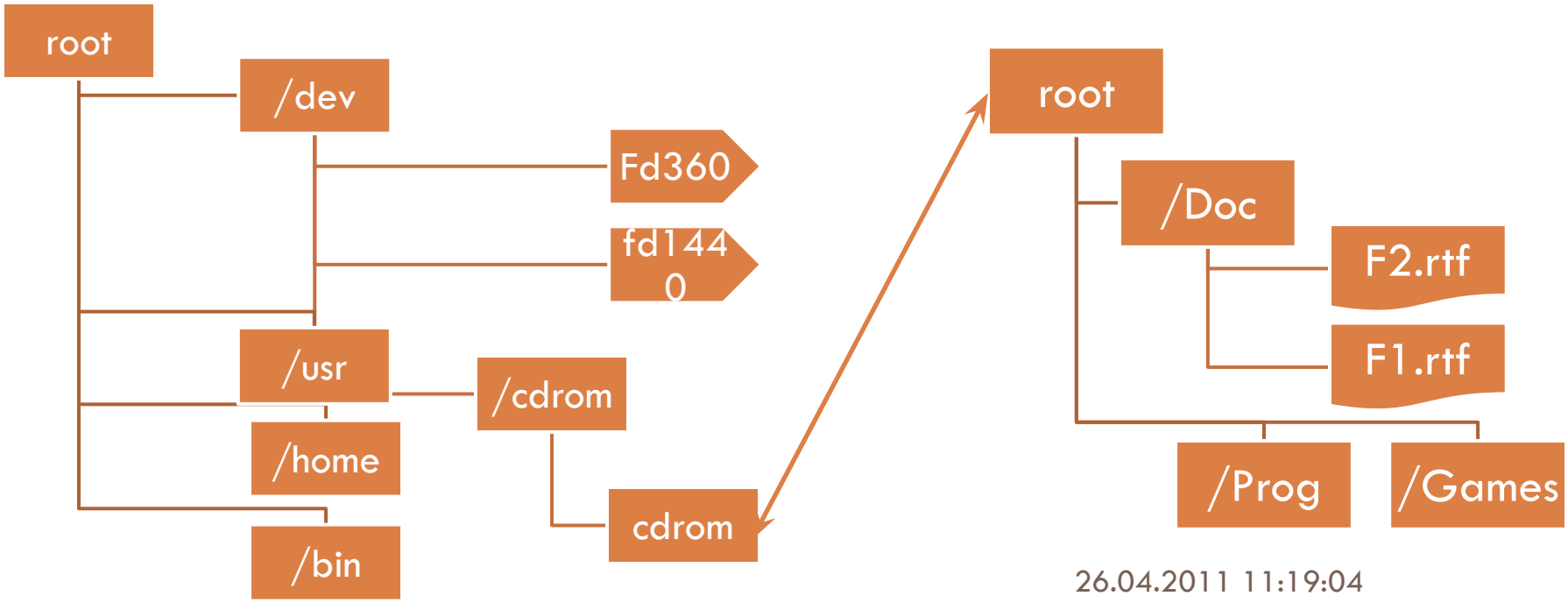
# Операция монтирования



mount -t iso9660 -o ro /dev/cdrom /cdrom

Здесь -t iso9660 - это тип файловой системы монтируемого устройства. Опция -o ro сообщает о том, что устройство используется только для чтения. /dev/cdrom - это название монтируемого устройства, а /cdrom - точка монтирования в файловой системе.

При извлечении устройства следует выполнить команду # umount /dev/cdrom



# Имена файлов



В UNIX для файла существует три типа имени - краткое, полное и относительное.

- Краткое имя идентифицирует файл в пределах одного каталога. Оно может состоять не более чем из 14 символов и содержать так называемый суффикс, отделяемый точкой.
- Полное имя однозначно определяет файл. Оно состоит из цепочки имен каталогов, через которые проходит маршрут от корневого каталога до данного файла. Имена каталогов разделяются символами "/", при этом имя корневого каталога не указывается, например, /mnt/rk2/test.c, где mnt и rk2 - имена каталогов, а test.c - имя файла. Каждому полному имени в ОС соответствует только один файл, однако файл может иметь несколько различных имен (жесткие связи).
- Относительное имя файла связано с понятием "текущий каталог", то есть каталог, имя которого задавать не нужно, так как оно подразумевается по умолчанию. Имя файла редставляет собой цепочку имен каталогов, через которые проходит маршрут от текущего каталога до данного файла. Относительное имя не начинается с символа "/". Если в предыдущем примере принять за текущий каталог /mnt, то относительное имя файла test.c будет rk2/test.c

# Привилегии доступа



- В UNIX s5 все пользователи по отношению к данному файлу делятся на три категории: владелец, член группы владельца и все остальные. Группа - это пользователи, которые объединены по какому-либо признаку, например, по принадлежности к одной разработке. Кроме этого, в системе существует суперпользователь, обладающий абсолютными правами по доступу ко всем файлам системы.
- Определены также три вида доступа к файлу - чтение, запись и выполнение. Привилегии доступа к каждому файлу определены для каждой из трех категорий пользователей и для каждой из трех операций доступа. Начальные значения прав доступа к файлу устанавливаются при его создании операционной системой и могут изменяться его владельцем или суперпользователем.

# Физическая организация файла



В общем случае файл может располагаться в несмежных блоках дисковой памяти. Логическая последовательность блоков в файле задается набором из 13 элементов. Первые 10 элементов предназначаются для непосредственного указания номеров первых 10 блоков файла. Если размер файла превышает 10 блоков, то в 11 элементе указывается номер блока, в котором содержится список следующих 128 блоков файла. Если файл имеет размер более, чем 10+128 блоков, то используется 12-й элемент, содержащий номер блока, в котором указываются номера 128 блоков, каждый из которых может содержать еще по 128 номеров блоков файла. Таким образом, 12-й элемент используется для двухуровневой косвенной адресации. В случае, если файл больше, чем $10+128+128^2$ блоков, то используется 13 элемент для трехуровневой косвенной адресации. При таком способе адресации предельный размер файла составляет 2 113 674 блока. Традиционная файловая система s5 поддерживает размеры блоков 512, 1024 или 2048 байт.

# Информация о файле. Индексные дескрипторы



Вся необходимая операционной системе информация о файле, кроме его символьного имени, хранится в специальной системной таблице, называемой индексным дескриптором (inode) файла. Индексные дескрипторы всех файлов имеют одинаковый размер - 64 байта и содержат данные о типе файла, о физическом расположении файла на диске (описанные выше 13 элементов), размере в байтах, о дате создания, последней модификации и последнего обращения к файлу, о привилегиях доступа и некоторую другую информацию. Индексные дескрипторы пронумерованы и хранятся в специальной области файловой системы. Номер индексного дескриптора является уникальным именем файла. Соответствие между полными символьными именами файлов и их уникальными именами устанавливается с помощью иерархии каталогов.

# Структура каталога



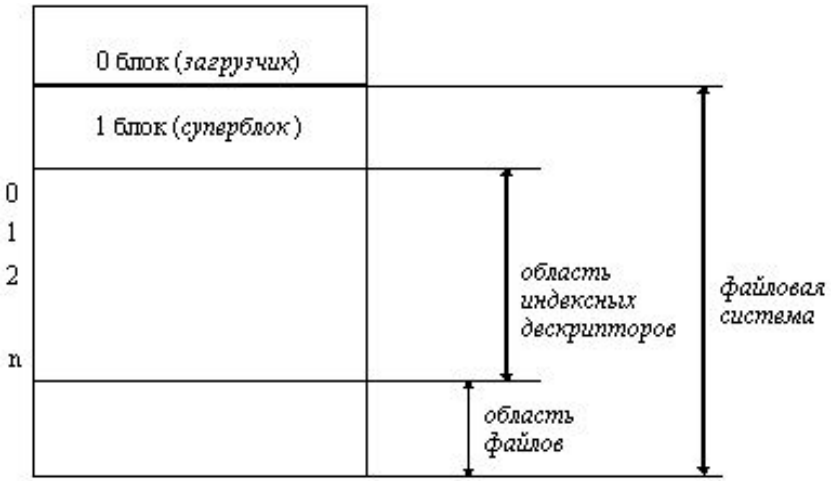
Каталог представляет собой совокупность записей обо всех файлах и каталогах, входящих в него. Каждая запись состоит из 16 байтов, 14 байтов отводится под короткое символьное имя файла или каталога, а 2 байта - под номер индексного дескриптора этого файла. В каталоге файловой системы s5 непосредственно не указываются характеристики файлов. Такая организация файловой системы позволяет с меньшими затратами перестраивать систему каталогов. Например, при включении или исключении файла из каталога идет манипулирование меньшими объемами информации. Кроме того, при включении одного и того же файла в разные каталоги не нужно иметь несколько копий как характеристик, так и самих файлов. С этой целью в индексном дескрипторе ведется учет ссылок на этот файл из всех каталогов. Как только число ссылок становится равным нулю, индексный дескриптор данного файла уничтожается.

# Структура диска s5



Все дисковое пространство, отведенное под файловую систему, делится на четыре области:

- *загрузочный блок (boot)*, в котором хранится загрузчик операционной системы;
- *суперблок (superblock)* - содержит самую общую информацию о файловой системе: размер файловой системы, размер области индексных дескрипторов, число индексных дескрипторов, список свободных блоков и список свободных индексных дескрипторов, а также другую административную информацию;
- *область индексных дескрипторов*, поря
- *область данных*, в которой дескрипторов в которой соответствует и расположены как обычные файлы, так и файлы-каталоги. Специальные файлы представлены в файловой системе только записями в соответствующих каталогах и индексными дескрипторами специального формата, но места в области данных не занимают.



# Доступ к файлу



Доступ к файлу осуществляется путем последовательного просмотра всей цепочки каталогов, входящих в полное имя файла, и соответствующих им индексных дескрипторов. Поиск завершается после получения всех характеристик из индексного дескриптора заданного файла. Эта процедура требует в общем случае нескольких обращений к диску, пропорционально числу составляющих в полном имени файла. Для уменьшения среднего времени доступа к файлу его дескриптор копируется в специальную системную область памяти. Копирование индексного дескриптора входит в процедуру открытия файла.

# Открытие файла



При открытии файла ядро выполняет следующие действия:

- Проверяет, существует ли файл; если не существует, то можно ли его создать. Если существует, то разрешен ли к нему доступ требуемого вида.
- Копирует индексный дескриптор с диска в оперативную память; если с указанным файлом уже ведется работа, то новая копия индексного дескриптора не создается.
- Создает в области ядра структуру, предназначенную для отображения текущего состояния операции обмена данными с указанным файлом. Эта структура, называемая file, содержит данные о типе операции (чтение, запись или чтение и запись), о числе считанных или записанных байтов, указатель на байт файла, с которым проводится операция.
- Делает отметку в контексте процесса, выдавшего системный вызов на операцию с данным файлом.

# Виртуальная файловая система VFS

# Идеология VFS



VFS не ориентируется на какую-либо конкретную файловую систему, механизмы реализации файловой системы полностью скрыты как от пользователя, так и от приложений. В ОС нет системных вызовов, предназначенных для работы со специфическими типами файловой системы, а имеются абстрактные вызовы типа open, read, write и другие, которые имеют содержательное описание, обобщающее некоторым образом содержание этих операций в наиболее популярных типах файловых систем (например, s5, ufs, nfs и т.п.). VFS также предоставляет ядру возможность оперирования файловой системой, как с единым целым: операции монтирования и демонтирования, а также операции получения общих характеристик конкретной файловой системы (размера блока, количества свободных и занятых блоков и т.п.) в единой форме. Если конкретный тип файловой системы не поддерживает какую-то абстрактную операцию VFS, то файловая система должна вернуть ядру код возврата, извещающий об этом факте.

# Информация о файлах и типы файлов VFS



В VFS вся информация о файлах разделена на две части - не зависящую от типа файловой системы, которая хранится в специальной структуре ядра - структуре vnode, и зависящую от типа файловой системы - структура inode, формат которой на уровне VFS не определен, а используется только ссылка на нее в структуре vnode. Имя inode не означает, что эта структура совпадает со структурой индексного дескриптора inode файловой системы s5.

Виртуальная файловая система VFS поддерживает следующие типы файлов:

- обычные файлы,
- каталоги,
- специальные файлы,
- именованные конвейеры,
- символьные связи.

Содержательное описание обычных файлов, каталогов и специальных файлов и связей не отличается от их описания в файловой системе s5.

# Символьные связи



Мягкая связь, называемая символьной связью и реализуемая с помощью системного вызова symlink. Символьная связь - это файл данных, содержащий имя файла, с которым предполагается установить связь. Символьная связь может быть создана даже с несуществующим файлом. При создании символьной связи образуется как новый вход в каталоге, так и новый индексный дескриптор inode. Кроме этого, резервируется отдельный блок данных для хранения полного имени файла, на который он ссылается.

Имеются три системных вызова, которые имеют отношение к символьным связям:

- readlink - чтение полного имени файла или каталога, на который ссылается символьная связь. Эта информация хранится в блоке, связанном с символьной связью.
- lstat - аналогичен системному вызову stat, но используется для получения информации о самой связи.
- lchown - аналогичен системному вызову chown, но используется для изменения владельца самой символьной связи.

# Реализация файловой системы VFS



UNIX System V Release 4 имеет массив структур vfssw [ ], каждая из которых описывает файловую систему конкретного типа, которая может быть установлена в системе. Структура vfssw состоит из четырех полей:

- символьного имени файловой системы;
- указателя на функцию инициализации файловой системы;
- указателя на структуру, описывающую функции, реализующие абстрактные операции VFS в данной конкретной файловой системе;
- флаги, которые не используются в описываемой версии UNIX.

Пример инициализированного массива структур vfssw:

```
struct vfssw vfssw[] = {{0, 0 , 0 ,0 }, // 0-й элемент не используется
{"spec",  specint,  &spec_vfsops,    0},  - SPEC
{"vxfs",  vx_init,  &vx_vfsops,      0},  - Veritas
{"cdfs",  cdfsinit, &cdfs_vfsops,    0},  - CD ROM
{"ufs",   ufsinit,  &ufs_vfsops,     0},  - UFS
{"s5",    vx_init,  &vx_vfsops,      0},  - S5
{"fifo",  fifoinit, &fifovfsops,     0},  - FIFO
{"dos",   dosinit,  &dos_vfsops,     0},  - MS-DOS
```

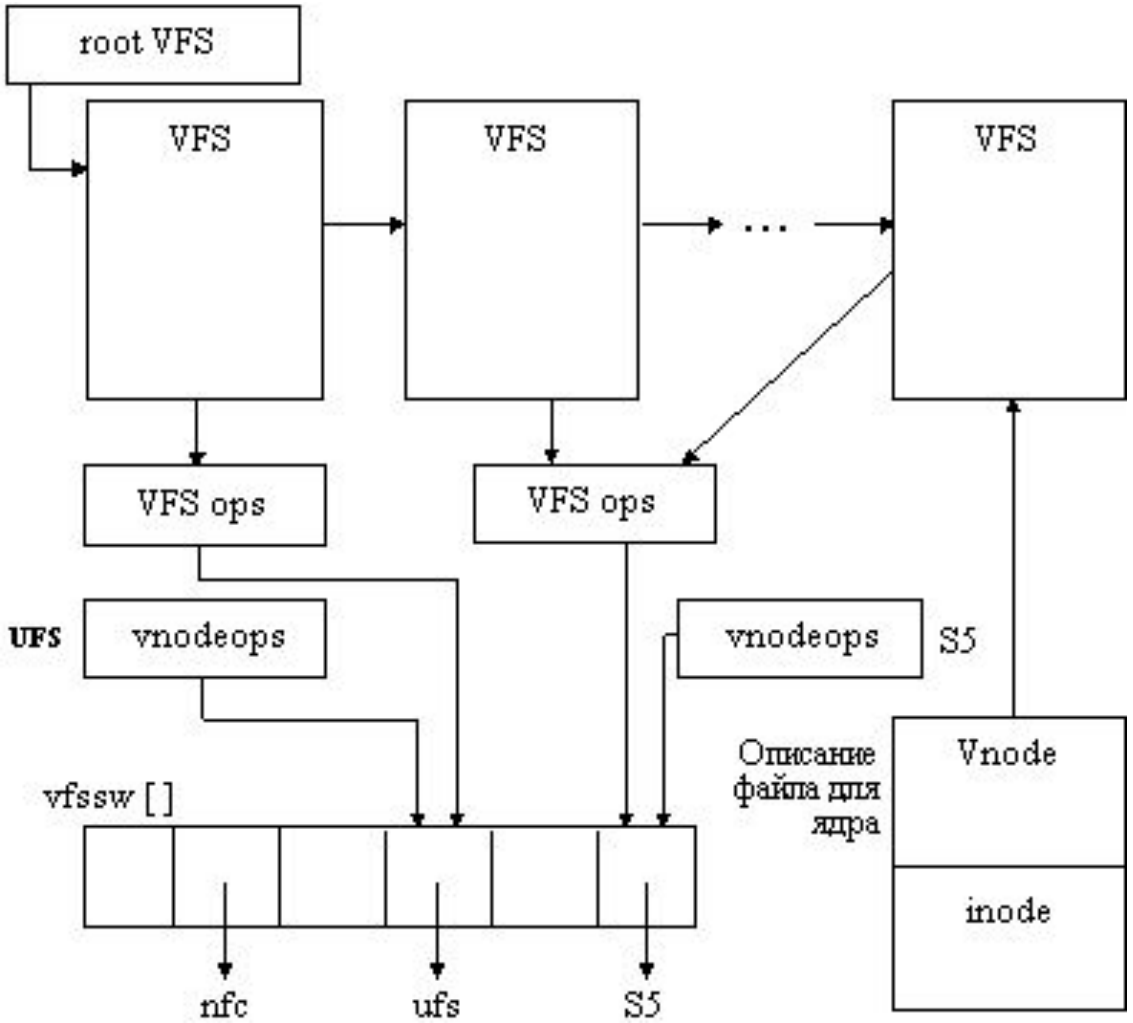
# Операции над файловой системой



| | |
|---|---|
| VFS_MOUNT | монтирование файловой системы, |
| VFS_UNMOUNT | размонтирование файловой системы, |
| VFS_ROOT | получение vnode для корня файловой системы, |
| VFS_STATVFS | получение статистики файловой системы, |
| VFS_SYNC | выталкивание буферов файловой системы на диск, |
| VFS_VGET | получение vnode по номеру дескриптора файла, |
| VFS_MOUNTROOT | монтирование корневой файловой системы. |

# Монтирование файловых систем в VFS





# Абстрактные операции над файлами



| | |
|---|---|
| VOP_OPEN | - открыть файл |
| VOP_CLOSE | - закрыть файл |
| VOP_READ | - читать из файла |
| VOP_WRITE | - записать в файл |
| VOP_IOCTL | - управление в/в |
| VOP_SETFL | - установить флаги статуса |
| VOP_GETATTR | - получить атрибуты файла |
| VOP_SETATTR | - установить атрибуты файла |
| VOP_LOOKUP | - найти vnode по имени файла |
| VOP_CREATE | - создать файл |
| VOP_REMOVE | - удалить файл |
| VOP_LINK | - связать файл |
| VOP_MAP | - отобразить файл в память |

# Структура vnodeops

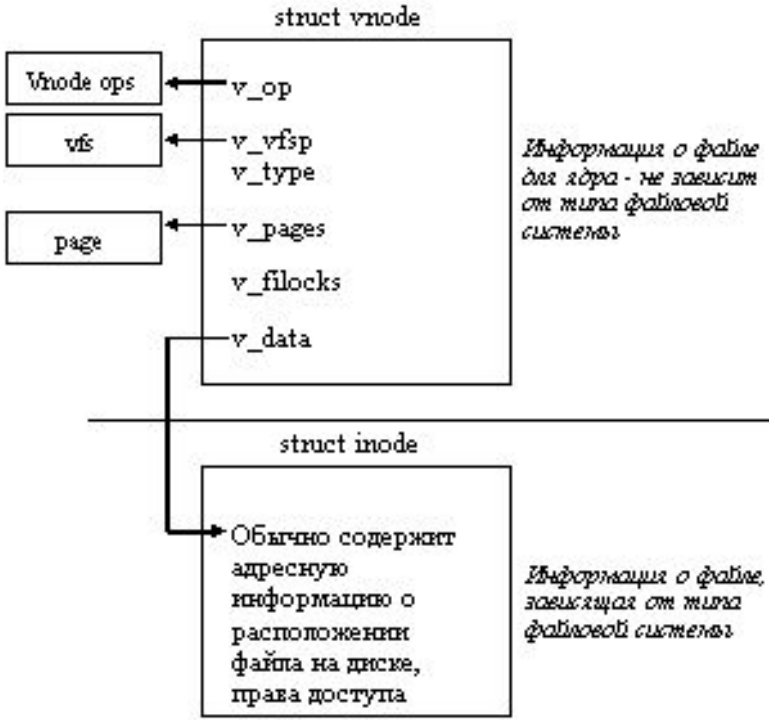


Кроме операций над файловой системой в целом, для каждого типа файловой системы (s5, ufs и т.д.), установленной в ОС, необходимо описать способ реализации абстрактных операций над файлами, которые допускаются в VFS. Этот способ описывается для каждого типа файловой системы в структуре vnodeops. Как видно из состава списка абстрактных операций, они образованы объединением операций, характерных для наиболее популярных файловых систем UNIX. Для того, чтобы обращение к специфическим функциям не зависело от типа файловой системы, для каждой операции в vnodeops определен макрос с общим для всех типов файловых систем именем, например, VOP_OPEN, VOP_CLOSE, VOP_READ и т.п. Эти макросы определены в файле <sys/vnode.h> и соответствуют системным вызовам. Таким образом, в структуре vnodeops скрыты зависящие от типа файловой системы реализации стандартного набора операций над файлами. Даже если файловая система какого-либо конкретного типа не поддерживает определенную операцию над своими файлами, она должна создать соответствующую функцию, которая выполняет некоторый минимум действий: или сразу возвращает успешный код завершения, или возвращает код ошибки. Для анализа и обработки полного имени файла в VFS используется операция VOP_LOOKUP, позволяющая по имени файла найти ссылку на его

# Структура vnode



Структура vnode используется ядром для связи файла с определенным типом файловой системы через поле v_vfsp и конкретными реализациями файловых операций через поле v_op. Поле v_pages используется для указания на таблицу физических страниц памяти в случае, когда файл отображается в физическую память (этот механизм описан в разделе, описывающем организацию виртуальной памяти). В vnode также содержится тип файла и указатель на зависимую от типа файловой системы часть описания характеристик файла - структуру inode, обычно содержащую адресную информацию о расположении файла на носителе и о правах доступа к файлу. Кроме этого, vnode используется ядром для хранения информации о блокировках (locks), примененных процессами к отдельным



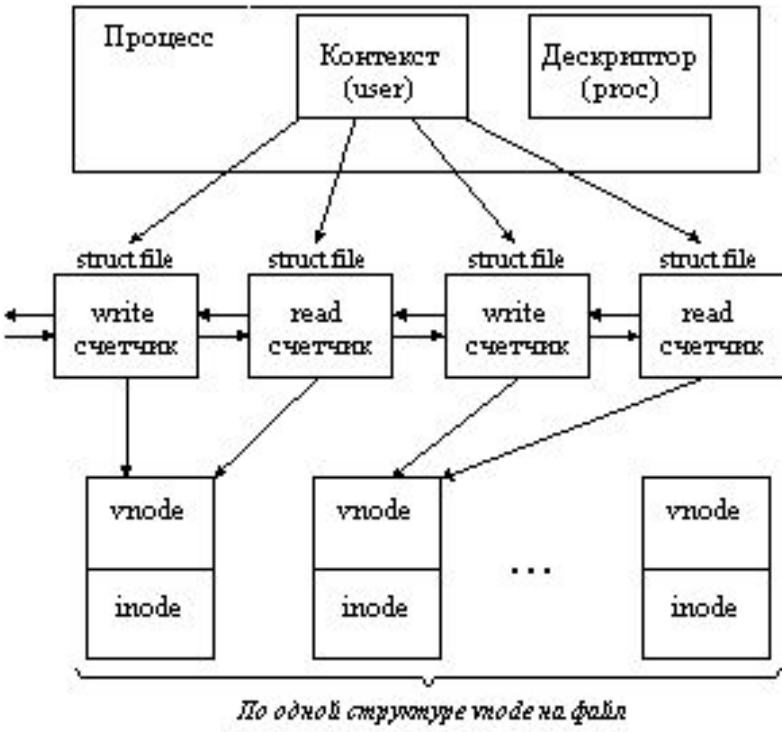
# Структура FILE



При каждом открытии процессом файла ядро создает в системной области памяти новую структуру типа file, которая, как и в случае традиционной файловой системы s5, описывает как открытый файл, так и операции, которые процесс собирается производить с файлом (например, чтение).

Структура file содержит такие поля, как:

- flag - определение режима открытия (только для чтения, для чтения и записи и т.п.);
- S truct vnode * f_vnode - указатель на структуру vnode (заменивший по сравнению с s5 указатель на inode);
- offset - смещение в файле при операциях чтения/записи;
- struct cred * f_cred - указатель на структуру, содержащую права процесса, открывшего файл (структура находится в дескрипторе процесса);
- указатели на предыдущую и последующую структуру типа file, связывающие все такие структуры в список.



# File И vnode



В отличие от структур типа file структуры типа vnode заводятся операционной системой для каждого активного (открытого) файла в единственном экземпляре, поэтому структуры file могут ссылаться на одну и ту же структуру vnode.

Структуры vnode не связаны в какой-либо список. Они появляются по требованию в системном пуле памяти и присоединяются к структуре данных, которая инициировала появление этого vnode, с помощью соответствующего указателя. Например, в случае структуры file в ней используется указатель f_vnode на соответствующую структуру vnode, описывающую нужный файл. Аналогично, если файл связан с образом процесса (то есть это файл, содержащий выполняемый модуль), то отображение сегмента памяти, содержащего части этого файла, осуществляется посредством указателя vp (в структуре segvn_data) на vnode этого файла.

Все операции с файлами в UNIX System V Release 4 производятся с помощью связанной с файлом структуры vnode. Когда процесс запрашивает операцию с файлом (например, операцию open), то независимая от типа файловой системы часть ОС передает управление зависимой от типа файловой системы части ОС для выполнения операции. Если зависимая часть обнаруживает, что структуры vnode, описывающей нужный файл, нет в оперативной памяти, то зависимая часть заводит для него новую структуру vnode.

# Система ввода-вывода в ОС Unix System V

# Подсистема буферизации



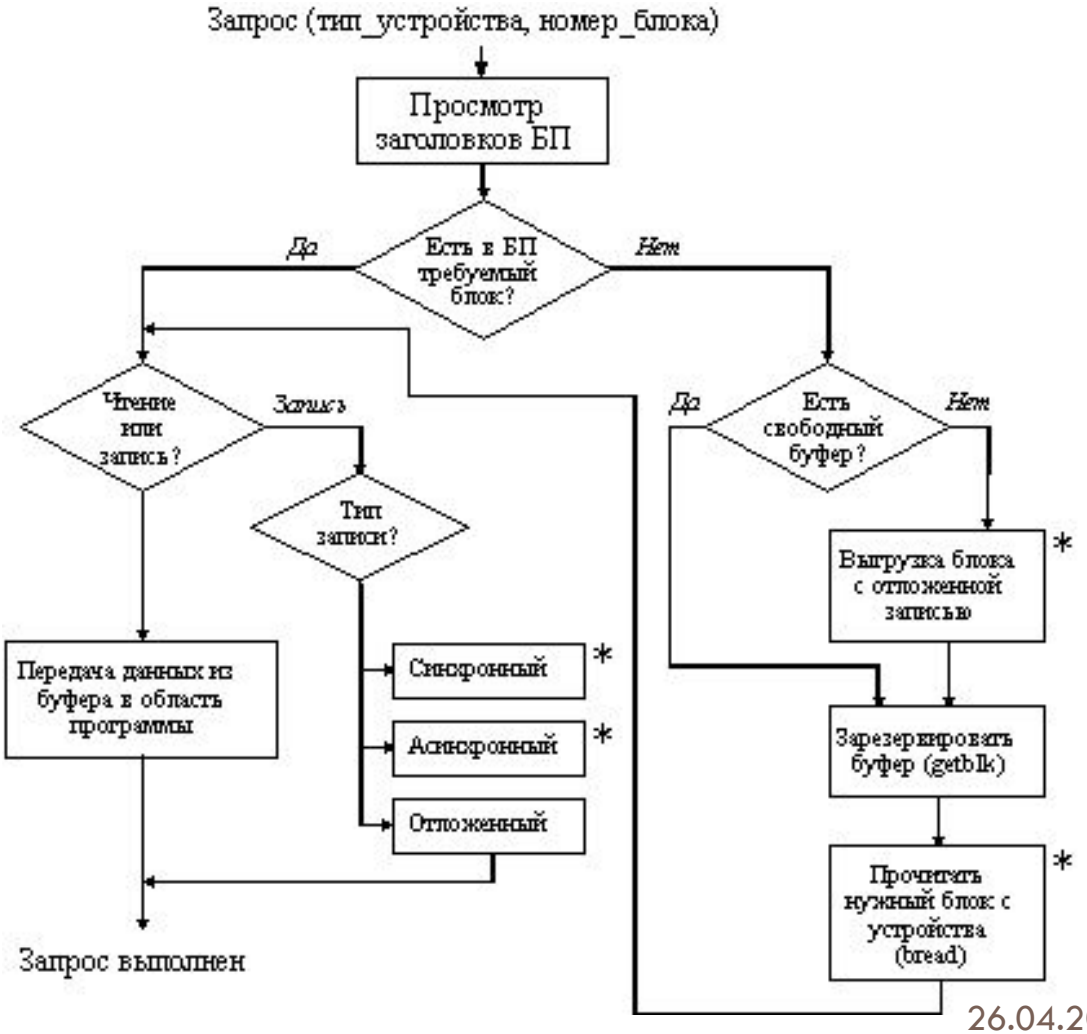
Основу системы ввода-вывода ОС UNIX составляют драйверы внешних устройств и средства буферизации данных. ОС UNIX использует два различных интерфейса с внешними устройствами: байт-ориентированный и блок-ориентированный.

Любой запрос на ввод-вывод к блок-ориентированному устройству преобразуется в запрос к подсистеме буферизации, которая представляет собой буферный пул и комплекс программ управления этим пулом. Буферный пул состоит из буферов, находящихся в области ядра. Размер отдельного буфера равен размеру блока данных на диске. С каждым буфером связана специальная структура - заголовок буфера, в котором содержится следующая информация:

| Данные о состоянии буфера: | Данные об устройстве - источнике информации, находящейся в этом буфере: |
|---|---|
| - занят/свободен,<br>- чтение/запись,<br>- признак отложенной записи,<br>- ошибка ввода-вывода. | - тип устройства,<br>- номер устройства,<br>- номер блока на устройстве.<br>-Адрес буфера. |

# Алгоритм выполнения запроса к подсистеме буферизации





# Драйверы



Драйвер - это совокупность программ (секций), предназначенная для управления передачей данных между внешним устройством и оперативной памятью.

Связь ядра системы с драйверами обеспечивается с помощью двух системных таблиц:

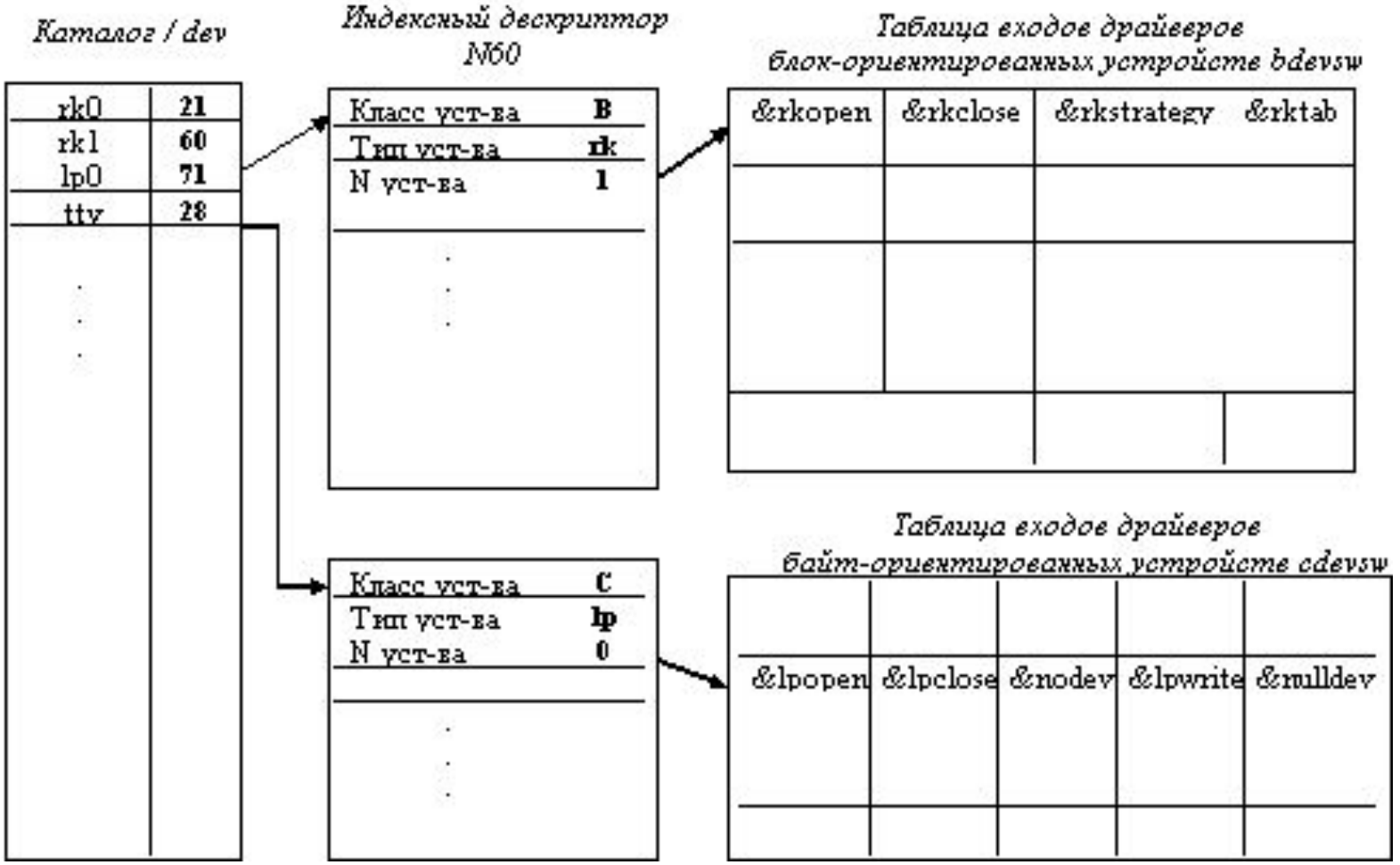
- bdevsw - таблица блок-ориентированных устройств и
- cdevsw - таблица байт-ориентированных устройств.

Для связи используется следующая информация из индексных дескрипторов специальных файлов:

- класс устройства (байт-ориентированное или блок-ориентированное),
- тип устройства (лента, гибкий диск, жесткий диск, устройство печати, дисплей, канал связи и т.д.)
- номер устройства.

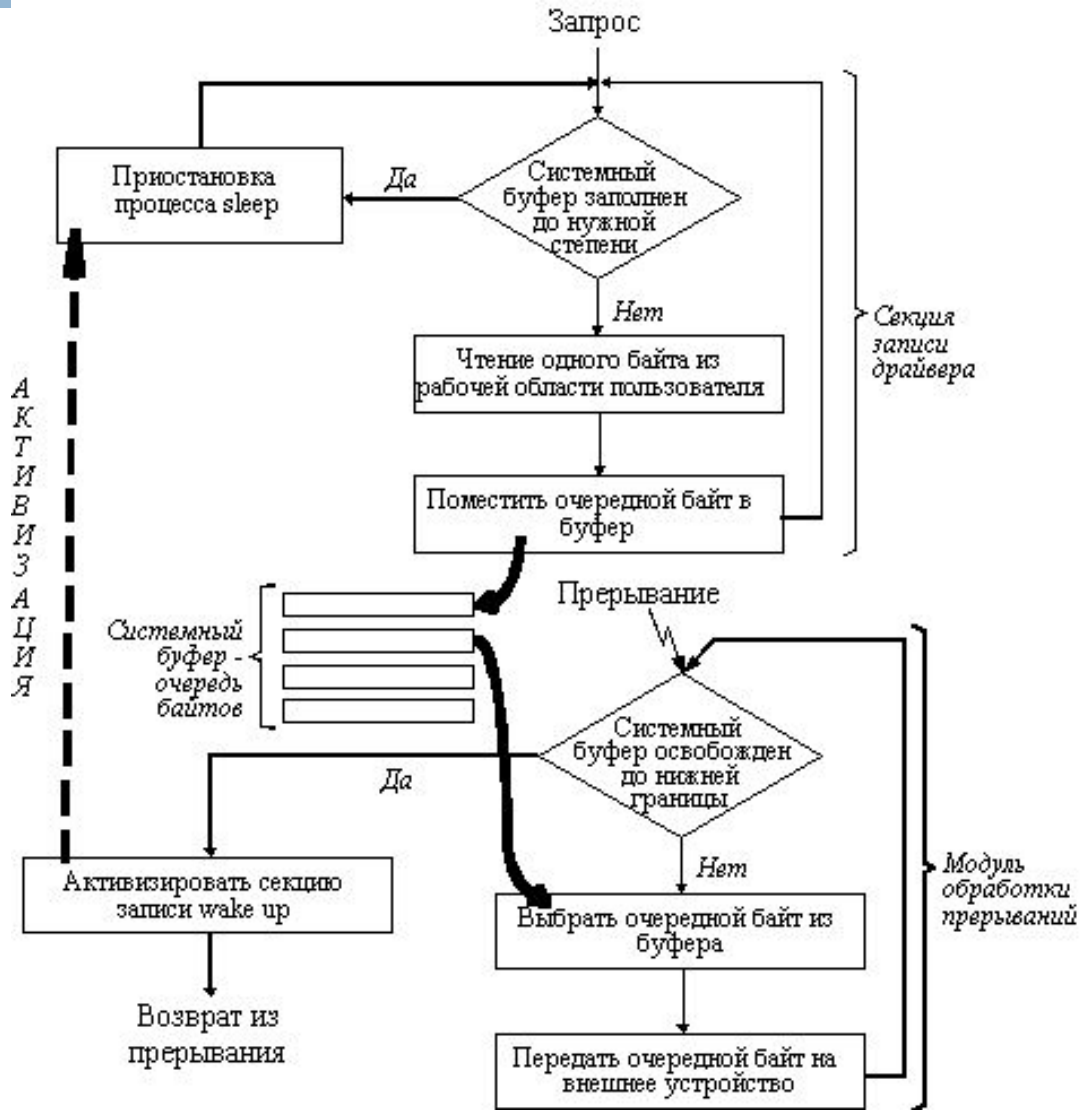
# Организация связи ядра с драйверами





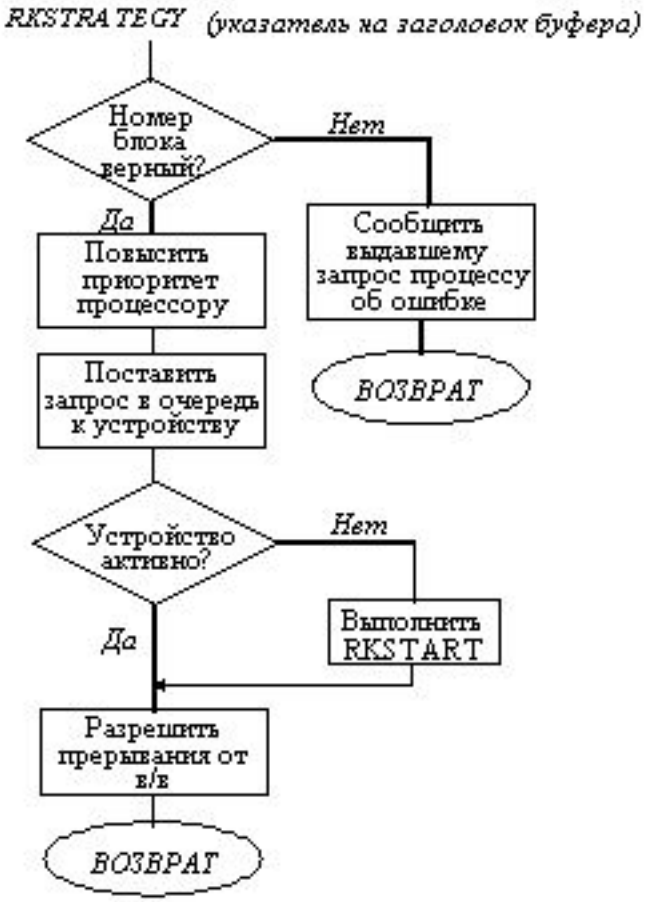
# Взаимодействие секции записи драйвера с модулем обработки прерывания

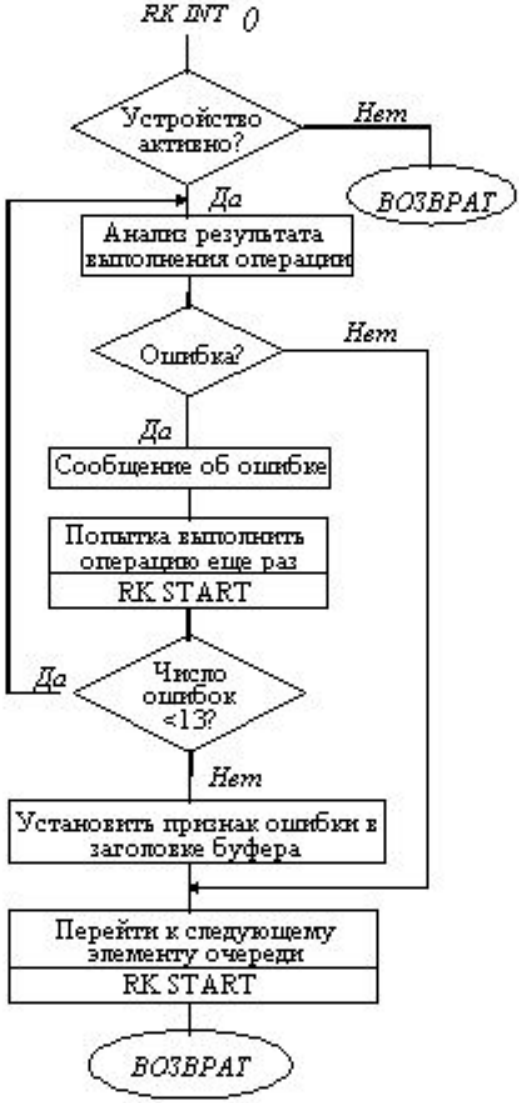


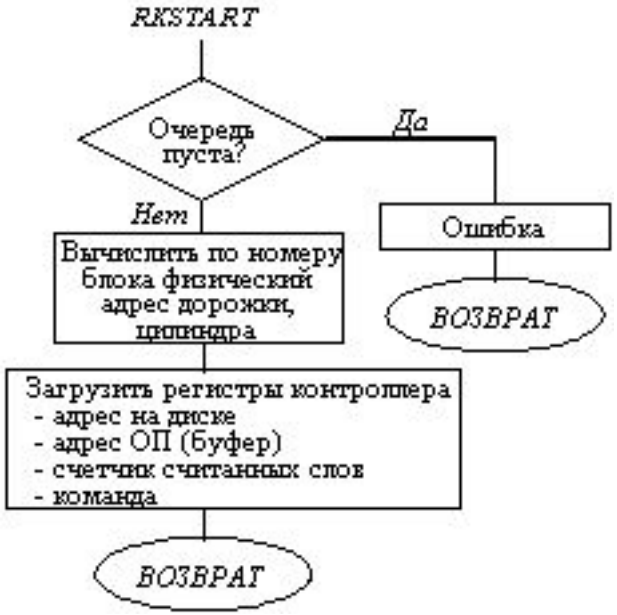


# Структурная схема драйвера диска типа RK





# Файловые системы Linux

# Файловая система ext2fs

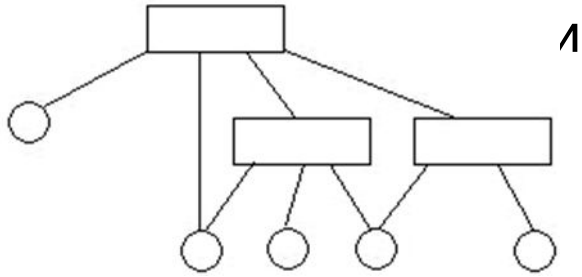


- На заре развития Linux использовала файловую систему ОС Minix. Она была довольно стабильна, но оставалась 16-разрядной и, как следствие, имела жёсткое ограничение в 64 Мегабайта на раздел. Также присутствовало ограничение на максимальную длину имени файла: оно составляло 14 символов. Эти и другие ограничения послужили стимулом к разработке «расширенной файловой системы» (англ. Extended File System), решавшей две главные проблемы Minix. Новая файловая система была представлена в апреле 1992 года. Ext расширила ограничения на размер файла до 2 гигабайт[2] и установила предельную длину имени файла в 255 символов.
- Тем не менее, оставалось ещё много нерешённых проблем: не было поддержки раздельного доступа, временных меток модификации данных. Именно эти проблемы послужили инициативой для создания следующей версии расширенной файловой системы ext2 (англ. Second Extended File System), разработанной в январе 1993 года. В ext2 были также реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов.

# Логическая организация файловой системы ext2



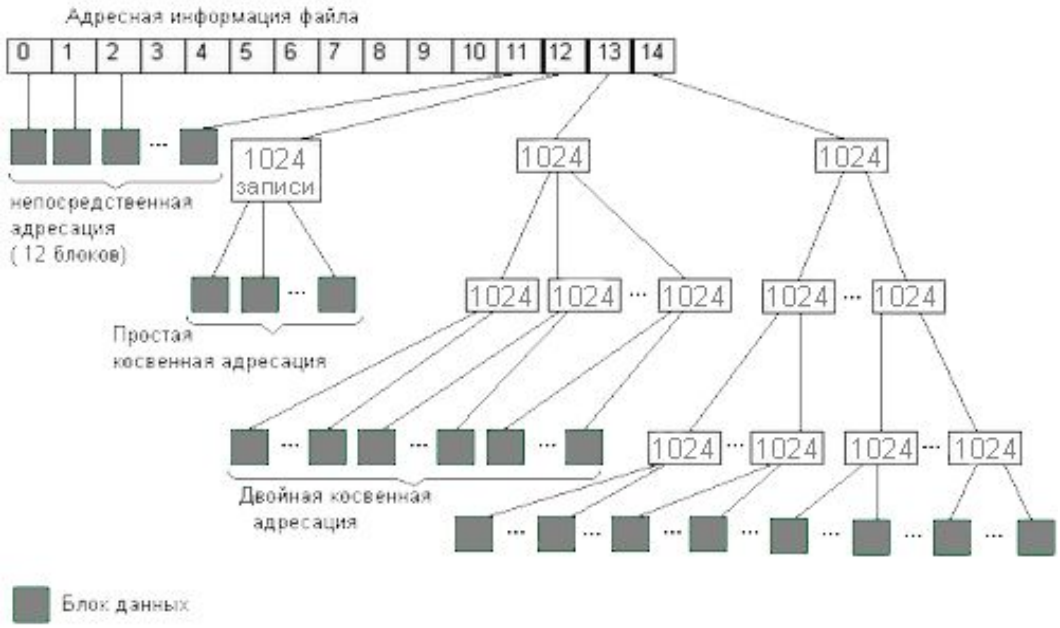
- Граф, описывающий иерархию каталогов файловой системы ext2, представляет собой сеть. Причиной такой организации является то, что один файл может входить сразу в несколько каталогов.
- Все типы файлов имеют символьные имена. В иерархически организованных файловых системах обычно используются три типа имен: простые, составные и относительные. Не является исключением и ext2. Ограничения на простое имя состоят в том что, его длина не должна превышать 255 символов, а также в имени не должны присутствовать символ NULL и слеш. Ограничения на символ NULL связаны с представлением строк в языке Си, а на символ слеш — с тем. что он используются как разделительный сим л.



# Физическая организация файловой системы ext2



| |
|---|
| Суперблок (Superblock) |
| Описани группы блоков (Group Descriptors) |
| Битовая карта блоков (Block Bitmap) |
| Битовая карта индексных дескрипторов (Inode Bitmap) |
| Таблица индексных дескрипторов (Inode Table) |
| Данные (Data) |



# Файловая система ext3fs



- Third Extended File System (третья версия расширенной файловой системы),— журналируемая файловая система, используемая в операционных системах на ядре Linux, является файловой системой по умолчанию во многих дистрибутивах. Основана на ФС ext2, начало разработки которой положил Стивен Твиди.
- Основное отличие от ext2 состоит в том, что ext3 журналируема, то есть в ней предусмотрена запись некоторых данных, позволяющих восстановить файловую систему при сбоях в работе компьютера.
- Стандартом предусмотрено три режима журналирования:
  - writeback: в журнал записываются только метаданные файловой системы, то есть информация о её изменении. Не может гарантировать целостности данных, но уже заметно сокращает время проверки по сравнению с ext2;
  - ordered: то же, что и writeback, но запись данных в файл производится гарантированно до записи информации о изменении этого файла. Немного снижает производительность, также не может гарантировать целостности данных (хотя и увеличивает вероятность их сохранности при дописывании в конец существующего файла);
  - journal: полное журналирование как метаданных ФС, так и пользовательских данных. Самый медленный, но и самый безопасный режим; может гарантировать целостность данных при хранении журнала на отдельном разделе (а лучше — на отдельном жёстком диске).

# Типовая структура диска



Disk /dev/hda: 30.0 GB, 30005821440 bytes 240 heads, 63 sectors/track, 3876 cylinders Units = cylinders of 15120 * 512 = 7741440 bytes Device Boot Start End Blocks Id System

/dev/hda1 * 1 14 105808+ 83 Linux

/dev/hda2 15 81 506520 82 Linux swap

/dev/hda3 82 3876 28690200 83 Linux

# Типовая структура диска



- Первым является маленький раздел (/dev/hda1) в начале диска, называемый загрузочным разделом. Цель загрузочного раздела – это хранение всех важных данных, связанных с загрузкой – загрузчик GRUB, а также ваше Linux ядро (ядра). Загрузочный раздел обеспечивает нам безопасное место для хранения любой информации, связанной с загрузкой. При нормальной работе загрузочный раздел должен оставаться отмонтированным для безопасности. Рекомендовалось держать загрузочный раздел (содержащий всё необходимое для загрузки) в начале диска. Это не обязательно, так как берет свои истоки из прошлого, когда загрузчик LILO не мог загружать ядро с файловых систем, которые располагались за 1024 цилиндром диска.
- Второй раздел (/dev/hda2) используется для подкачки. Ядро использует дисковое пространство подкачки как виртуальную память, когда места в ОЗУ мало. Размер раздела сравнительно не очень большой, как правило около 512 МБ.
- Третий раздел (/dev/hda3) большого размера и занимает весь остальной диск. Этот раздел будет нашим корневым разделом, и будет служить для хранения главной файловой системы Linux.

# Файловая система Ext4fs



- Fourth Extended File System (четвёртая версия расширенной файловой системы), сокр. ext4, или ext4fs — журналируемая файловая система, используемая в ОС с ядром Linux. Основана на файловой системе ext3, которая является файловой системой по умолчанию во многих дистрибутивах Linux.
- Впервые экспериментальная поддержка ext4 была выпущена в виде патча для Linux версий 2.6.19-rc1-mm1 и 2.6.19-rc1-git8 10 октября 2006 года программистом Эндрю Мортоном
- Основной особенностью стало увеличение максимального объёма одного раздела диска до 1 эксабайта (260 байт) при размере блока 4Kb, и увеличение размера одного файла до 16 терабайт. Кроме того, в ext4 представлен механизм пространственной (extent) записи файлов (новая информация добавляется в конец заранее выделенной по соседству области файла), уменьшающий фрагментацию и повышающий производительность.

# Файловая система ReiserFS



- ReiserFS – это файловая система, основанная на B-дереве, которая имеет очень хорошую производительность и значительно превосходит ext2 и ext3 при работе с небольшими файлами (файлы менее 4 кБ), часто в 10-15 раз. А также ReiserFS отлично масштабируется и имеет журналирование метаданных. Начиная с ядра версии 2.4.18 и выше, ReiserFS является стабильной и рекомендуется, как в качестве ФС общего назначения, так и в крайних случаях, таких как создание больших файловых систем, использование для множества маленьких файлов, для огромных файлов, а также для каталогов с десятками тысяч файлов.
- ФС ReiserFS рекомендуется для использования по умолчанию для всех не загрузочных разделов.