ext2 (Second Extended File System)

ext2	
Полное название	Second extended file system
Содержимое каталога	Table
Распределение файлов	bitmap (free space), table (metadata)
Сбойные блоки	Table
Ограничения	
Макс. размер тома	2–32 TiB
Макс. количество файлов	10^{18}
Макс. длина имени файла	255 bytes
Разрешенные символы в именах файлов	All bytes except NUL ('\0') and '/'
Особенности	
Даты зарегистрирован	modification (mtime),

	attribute modification (ctime), access (atime)
Диапазон дат	December 14, 1901 - January 18, 2038
Дата резолюции	1 s
Разрешения файловой системы	POSIX
Другие	
Операционная система	Linux, BSD, Windows (through an IFS), Mac OS X (through an IFS)

ext2 (англ. ***S**econd **E**xtended **F**ile **S**ystem*) (дословно: «вторая расширенная файловая система»), сокращённо **ext2** (иногда **ext2fs**) — файловая система ядра Linux. Была разработана Реми Кардом (англ.) взамен существующей тогда ext. По скорости и производительности работы она может служить эталоном в тестах производительности файловых систем. Так, в тестах на скорость последовательного чтения и записи, проведённых The Dell

TechCenter, файловая система ext2 обгоняет ext3, и уступает лишь более современной ext4 в тесте на чтение.

Файловая система ext2 по-прежнему используется на флеш-картах и твердотельных накопителях (SSD), так как отсутствие журналирования является преимуществом при работе с накопителями, имеющими ограничение на количество циклов записи.

Архитектура ext2

Файловую систему можно представить в виде следующей структурной схемы:

- Суперблок (Superblock)
- Описание группы блоков (Block Group Descriptors)
- Битовая карта блоков (Blocks Bitmap)
- Битовая карта индексных дескрипторов (Inode Bitmap)
- Таблица индексных дескрипторов (Inode Table)
- Данные

Все пространство раздела диска разбивается на так называемые блоки, имеющие порядковые номера. Блоки имеют фиксированный размер и кратны размеру сектора (1024,2048,4096...). Для уменьшения количества перемещений головки жесткого диска блоки объединяют в группы блоков.

Одним из базовых понятий ext2 является понятие индексного дескриптора (inode, инод, айнод). Это специальная структура, содержащая в себе информацию об физическом расположении файла/директории/ссылки и т.п. и его атрибутах.

Суперблок

Суперблок - основной элемент файловой системы ext2. Он содержит следующую информацию о файловой системе (список не полный):

- общее число блоков и inode-ов в файловой системе
- число свободных блоков и inode-ов в файловой системе
- размер блока файловой системы
- количество блоков и inode-ов в группе
- размер inode-а

- идентификатор файловой системы
- номер первого блока данных. Другими словами, это номер блока, содержащего суперблок. Этот номер всегда равен 0, если размер блока файловой системы больше 1024 байт, и 1, если размер блока равен 1024 байт

Описание группы блоков

BGD таблица содержит индексные дескрипторы для каждой группы блоков файловой системы и распологается сразу после суперблока. Таким образом, если указано, что размер блока составляет 1 кб, то блок с таблицей начинается в втором блоке. Блоки нумеруются с нуля, и номера блоков не соответствуют их физическому нахождению.

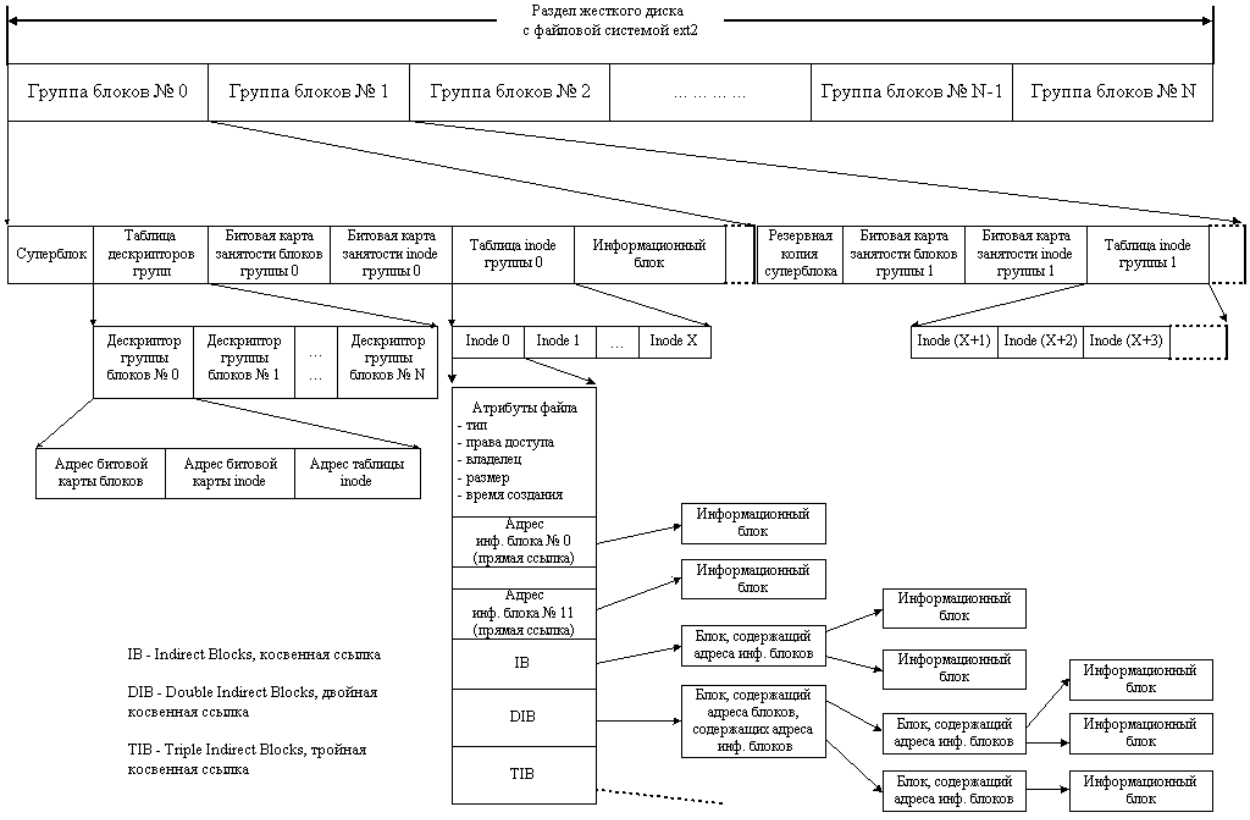
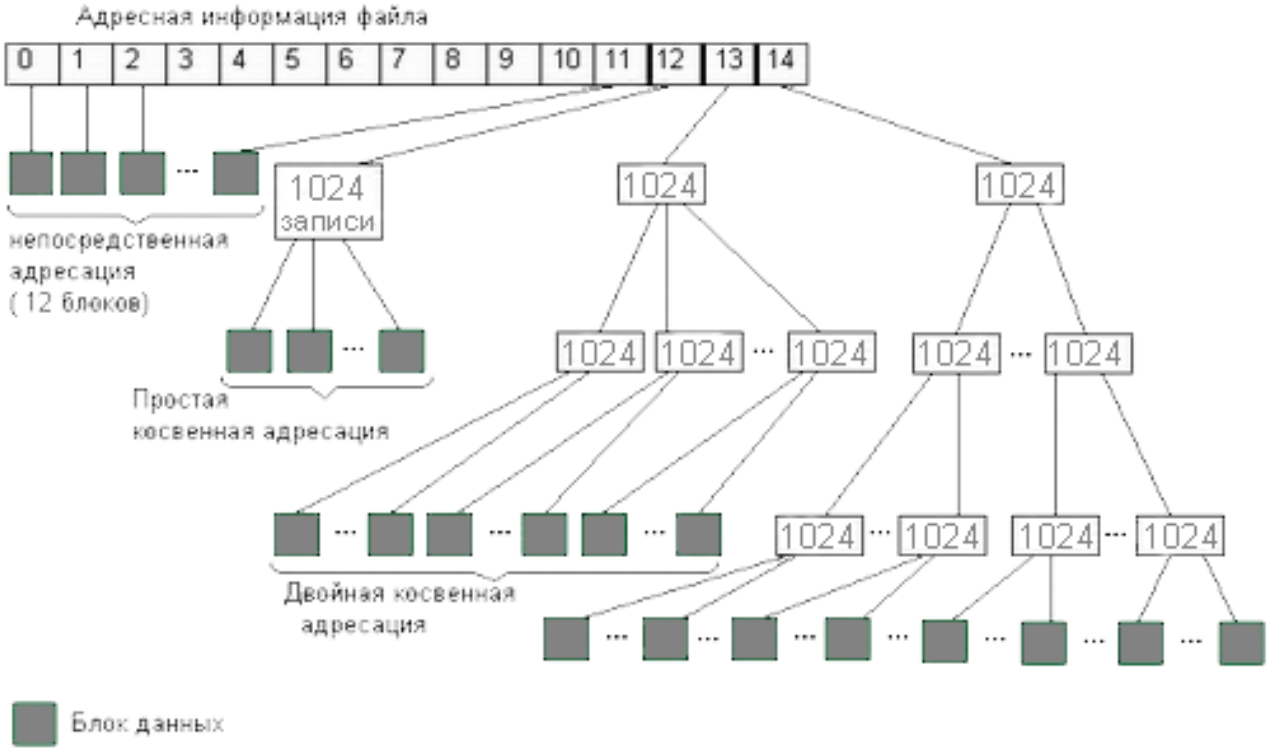


Рис. 3. Обобщенная структурная схема файловой системы ext2

Система адресации данных

Система адресации данных — это одна из самых важных составляющих файловой системы. Именно она позволяет находить нужный файл среди множества как пустых, так и занятых блоков на диске. Для хранения адреса файла выделено 15 полей по 4 байта. Если файл умещается в пределах 12 блоков, то в первых 12 полях адреса перечислены номера соответствующих кластеров, иначе следующее поле используется для косвенной адресации. Возможна ситуация, когда размер файла превышает 1024+12 блоков. Тогда 14 поле используется для двойной косвенной адресации, но если вдруг файл включает в себя более 1049612 блоков, в дело вступает тройная косвенная адресация и 15 блок. Такая организация позволяет при максимальном размере блока в 4Кб оперировать файлами, размер которых превышает 2Тб.



Применение

- ext2 применяется в системах на которые проблематично установить современные файловые системы
- Файловые системы для boot-раздела, в случаях, когда на корневом разделе используется экзотическая ФС, которую не поддержит загрузчик или ядро
- Для флэш-накопителей
- В кэшах, временных файлах и т. д. Так как, там нужен максимум скорости при ненужной сохранности данных
- При восстановлении удалённых файлов

Преимущества и недостатки

Преимущества

- Так как, в системе нет журналирования, она может снизить интенсивность записи и износ ячеек флэш-памяти
- Высокая производительность
- Обладает отличной совместимостью, т. е. будет прочитана любой Linux-системой, большинством BSD-систем

Недостатки

Главный недостаток ext2 (и одна из причин демонстрации столь высокой производительности) заключается в том, что она не является журналируемой файловой системой. Он был устранён в файловой системе ext3 — следующей версии Extended File System, полностью совместимой с ext2. Но для ssd это скорее плюс, продлевает жизнь накопителя. Это основная причина, почему EXT2 до сих пор поддерживается в Anaconda и Ubiquity.

ext3 (англ. *Third Extended File system*) – это расширенная версия одноименной файловой системы **Extended file system** (**ext**). **Ext3** журналируема, является расширением к ext2 файловой системе. Она обратно совместима с ext2 и преобразование из ext2 в ext3 является очень простым. Используется в операционных системах на ядре Linux, является файловой системой по умолчанию во многих дистрибутивах. Для ее использования не требуется никаких установок, все необходимые пакеты доступны в основной LFS системе. Дата представления: ноябрь 2001 года (Linux 2.4.15).

Описание

Стандартом предусмотрено три режима журналирования:

1. **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;
2. **ordered**: то же, что и *writeback*, но запись данных в файл производится гарантированно до записи информации об изменении этого файла. Немного снижает производительность, также не может гарантировать целостности данных (хотя и увеличивает вероятность их сохранности при дописывании в конец существующего файла);
3. **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).

Указывается режим журналирования в строке параметров для программы mount, например:

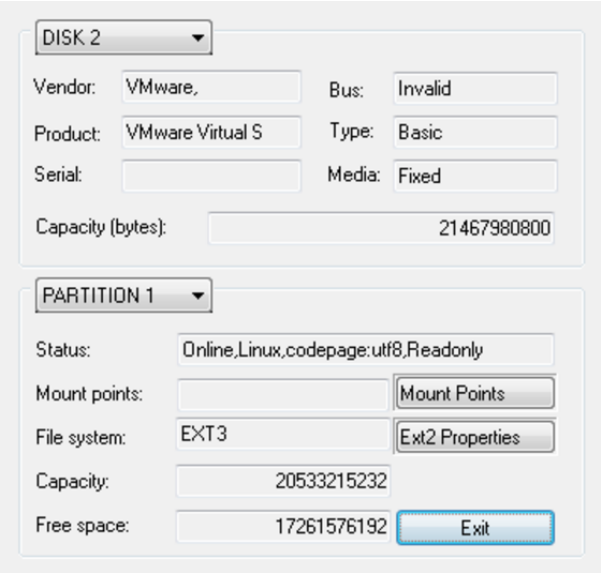
```
mount /dev/hda6 /mnt/disc -t ext3 -o data=<режим>
```

либо в файле /etc/fstab.

Файловая система **ext3** может поддерживать файлы размером до 1 ТБ. С Linux-ядром 2.4 объём файловой системы ограничен максимальным размером блочного устройства, что составляет 2 терабайта. В Linux 2.6 (для 32-разрядных процессоров)

максимальный размер блочных устройств составляет 16 ТБ, однако **ext3** поддерживает только до 4 ТБ.

Ext3 имеет одно значительно преимущество перед другими журналируемыми файловыми системами - она полностью совместима с файловой системой ext2. Это делает возможным использование всех существующих приложений разработанных для манипуляции и настройки файловой системы ext2. **Ext3** поддерживается ядрами Linux версии 2.4.16 и более поздними, и должна быть активизирована использованием диалога конфигурации файловых систем (Filesystems Configuration) при сборке ядра. В Linux дистрибутивы, такие как Red Hat 7.2 и SuSE 7.3 уже включена встроенная поддержка файловой системы **ext3**. Вы можете использовать файловую систему ext3 только в том случае, когда поддержка ext3 встроена в ваше ядро и у вас есть последние версии утилит "mount" и "e2fsprogs". Описываемая файловая система предоставляет значительные преимущества для большого круга пользователей Linux: минимизирует задержки при перезагрузке системы, сводит к минимуму возможность появления ошибок в ФС, является высокопроизводительной, а ее утилиты делают перевод системы из ext2 в **ext3** очень простым. Эта совместимость так же увеличивает возможность использование всех утилит, созданных для работы с ext2.



Из ext2 в ext3

В большинстве случаев перевод файловых систем из одного формата в другой влечет за собой резервное копирование всех содержащихся данных, переформатирование разделов или логических томов, содержащих файловую систему, и затем восстановление всех данных на эту файловую систему. В связи с совместимостью файловых систем **ext2** и **ext3**, все эти действия можно не проводить, и перевод может быть сделать с помощью одной команды (запущенной с полномочиями *root*):

```
# /sbin/tune2fs -j <имя-раздела >
```

Например, перевод файловой системы ext2 расположенной на разделе /dev/hda5 в файловую систему **ext3** может быть осуществлен с помощью следующей команды:

```
# /sbin/tune2fs -j /dev/hda5
```

Опция '-j' команды 'tune2fs' создает журнал **ext3** на существующей ext2 файловой системе. После перевода файловой системы ext2 в **ext3**, нужно так же должны внести изменения в записи файла /etc/fstab, для указания что теперь раздел является файловой системой 'ext3'. Также, можно использовать авто определение типа раздела (опция "auto"), но все же рекомендуется явно указывать тип файловой системы. Следующий пример файл

/etc/fstab показывает изменения до и после перевода файловой системы для раздела */dev/hda5*:

До:

```
/dev/hda5 /opt ext2 defaults 1 2
```

После:

```
/dev/hda5 /opt ext3 defaults 1 0
```

Последнее поле в */etc/fstab* указывает этап в загрузке, во время которого целостность файловой системы должна быть проверена с помощью утилиты "fsck". При использовании файловой системы **ext3**, вы можете установить это значение в '0', как показано на предыдущем примере. Это означает что программа 'fsck' никогда не будет проверять целостность файловой системе, в связи с тем что целостность файловой системы гарантируется путем отката в журнале.

Перевод корневой файловой системы в **ext3** требует особого подхода, и лучше всего его проводить в режиме одного пользователя *(single user mode)* после создания RAM диска поддерживающего файловую систему **ext3**.

Ограничения размеров

Максимальное число блоков для **ext3** равняется 2^{32}. Размер блока может быть различным, что влияет на максимальное число файлов и максимальный размер файла в файловой системе.

Размер блока	**Max размер файла**	**Max размер ФС**
1 KiB	16 GiB	до 2 TiB
2 KiB	256 GiB	до 8 TiB
4 KiB	2 TiB	до 16 TiB
8 KiB **(+прим.)**	2 TiB	до 32 TiB

(+прим.) Размер данного блока в **Linux** доступен только на архитектурах, поддерживающих страницы в 8 KiB, например, **Alpha**.

ext4 (англ. *Fourth Extended File system* - четвёртая расширенная файловая система, сокр. **ext4**, или **ext4fs)** — журналируемая файловая система (ФС), используемая в операционных системах с ядром Linux. Основана на ФС ext3, ранее использовавшейся по умолчанию во многих дистрибутивах GNU/Linux. Отличается от ext3 поддержкой extent'ов, групп смежных физических блоков, управляемых как единое целое; повышенной скоростью проверки целостности и рядом других усовершенствований.

История создания

Ext4 — это результат эволюции Ext3, наиболее популярной файловой системы в Linux. Во многих аспектах Ext4 представляет собой больший шаг вперёд по сравнению с Ext3, чем Ext3 была по отношению к Ext2. Наиболее значительным усовершенствованием Ext3 по сравнению с Ext2 было журналирование, в то время как Ext4 предполагает изменения в важных структурах данных, таких как, например, предназначенных для хранения данных файлов. Это позволило создать файловую систему с более продвинутым дизайном, более производительную и стабильную и с обширным набором функций.

Первая экспериментальная реализация ext4 была написана Эндрю Мортоном и выпущена 10 октября 2006 года в виде патча к ядрам Linux версий 2.6.19-rc1-mm1 и 2.6.19-rc1-git8 .

В октябре 2008 была переименована из ext4dev в ext4, что символизирует то, что с точки зрения разработчиков она достаточно стабильна. В ядре 2.6.28 (вышедшем 25.12.2008) файловая система уже называется ext4 и считается стабильной. Файловая система ext4 рассматривается как промежуточный шаг на пути к файловой системе следующего поколения Btrfs, которая претендует на звание основной файловой системы Linux в будущем.

Характеристика

Основные изменения по сравнению с ext3:

- увеличение максимального объёма одного раздела диска до 1 эксбибайта (2^{60} байт) при размере блока 4 кибибайт;

- увеличение размера одного файла до 16 тебибайт (2^{44} байт);
- введение механизма пространственной (extent) записи файлов, уменьшающего фрагментацию и повышающего производительность. Суть механизма заключается в том, что новая информация добавляется в конец области диска, выделенной заранее по соседству с областью, занятой содержимым файла.

Возможности ext4

- ***Использование экстентов*** (англ. *extent*). В файловой системе ext3 адресация данных выполнялась традиционным образом, поблочно. Такой способ адресации становится менее эффективным с ростом размера файлов. Экстенты позволяют адресовать большое количество (до 128 MB) последовательно идущих блоков одним дескриптором. До 4х указателей на экстенты может размещаться непосредственно в inode, что достаточно для файлов маленького и среднего размера.
- ***48-и битные номера блоков.*** На сегодняшний день максимальный размер файловой системы Ext3 равен 16 терабайтам, а размер файла ограничен 2 терабайтами. В Ext4 добавлена 48-битная адресация блоков, что означает, что максимальный размер этой файловой системы равен одному экзабайту, и файлы могут быть размером до 16 терабайт. 1 EB (экзабайт) = 1,048,576 TB (терабайт), 1 EB = 1024 PB (петабайт), 1 PB = 1024 TB, 1 TB = 1024 GB.
- ***Выделение блоков группами*** (англ. *multiblock allocation*). Файловая система хранит не только информацию о местоположении свободных блоков, но и количество свободных блоков, идущих друг за другом. При выделении места файловая система находит такой фрагмент, в который данные могут быть записаны без фрагментации. Это снижает уровень фрагментации ФС в целом.
- ***Отложенное выделение блоков*** (англ. *delayed allocation*). Выделение блоков для хранения данных файла происходят непосредственно перед физической записью на диск

(например, при вызове **sync**), а не при вызове *write*. В результате, операции выделения блоков можно делать не по одной, а группами, что в свою очередь минимизирует фрагментацию и ускоряет процесс выделения блоков. С другой стороны, увеличивает риск потери данных в случае внезапного пропадания питания.

- ***Масштабируемость подкаталогов***. В настоящий момент один каталог Ext3 не может содержать более, чем 32000 подкаталогов. Ext4 снимает это ограничение и позволяет создавать неограниченное количество подкаталогов.
- ***Резервирование inode'ов при создании каталога*** (англ. *directory inodes reservation*). При создании каталога резервируется несколько inode'ов. Впоследствии, при создании файлов в этом каталоге сначала используются зарезервированные inode'ы, и если таких не осталось, выполняется обычная процедура.
- ***Размер inode***. Размер inode (по умолчанию) увеличен с 128 до 256 байтов. Это дало возможность реализовать те преимущества, которые перечислены ниже.
- ***Временные метки с наносекундной точностью*** (англ. *nanosecond timestamps*). Точность временных меток, хранящихся в inode, повышена до наносекунд. Диапазон значений тоже расширен: у ext3 верхней границей хранимого времени было 18 января 2038 года, а у ext4 — 25 апреля 2514 года.
- ***Версия inode.*** В inode появился номер, который увеличивается при каждом изменении inode файла. Это будет использоваться, например, в NFSv4, для того чтобы узнавать, изменился ли файл.
- ***Хранение расширенных атрибутов в inode*** (англ. *extended attributes (EA) in inode*). Хранение расширенных атрибутов, таких как ACL, атрибутов SELinux и прочих, позволяет повысить производительность. Атрибуты, для которых недостаточно места в inode, хранятся в отдельном блоке размером 4KB. Предполагается снять это ограничение в будущем.
- ***Контрольное суммирование в журнале*** (Journal checksumming). Контрольные суммы журнальных транзакций.

Позволяют лучше найти и (иногда) исправить ошибки при проверке целостности системы после сбоя.

- ***Режим без журналирования***. Журналирование обеспечивает целостность файловой системы путём протоколирования всех происходящих на диске изменений. Но оно также вводит дополнительные накладные расходы на дисковые операции. В некоторых особых ситуациях журналирование и предоставляемые им преимущества могут оказаться излишними. Ext4 позволяет отключить журналирование, что приводит к небольшому приросту производительности.
- ***Предварительное выделение*** (англ. *persistent preallocation*). Сейчас для того, чтобы приложению гарантированно занять место в файловой системе, оно заполняет его нулями. В ext4 появилась возможность зарезервировать множество блоков для записи и не тратить на инициализацию лишнее время. Если приложение попробует прочитать данные, оно получит сообщение о том, что они не проинициализированы. Таким образом, несанкционированно прочитать удалённые данные не получится.
- ***Дефрагментация без размонтирования*** (англ. *online defragmentation*). Дефрагментация выполняется утилитой e4defrag, поставляемой в составе пакета e2fsprogs с 2011 года.
- ***Неинициализированные блоки*** (англ. *uninitialised groups*). Возможность пока не реализована и предназначена для ускорения проверки целостности ФС утилитой fsck. Блоки, отмеченные как неиспользуемые, будут проверяться группами, и детальная проверка будет производится только если проверка группы показала наличие повреждений. Предполагается, что время проверки будет составлять от 1/2 до 1/10 от нынешнего в зависимости от способа размещения данных.
- ***Прямая и обратная совместимость с ext2/ext3***. Файловые системы ext2/ext3 можно монтировать как файловую систему ext4. Наоборот — монтировать файловую ext4 как ext3 — можно только в том случае, если на ext4 не используются экстенты[2].

Дефрагментация

Файловые системы в Linux изначально спроектированы так, чтобы фрагментация файлов была как можно меньше. Тем не менее, фрагментированность файлов имеет место быть, и в некоторых случаях может привести к заметному падению производительности дисковой подсистемы. Для Ext4 есть несколько дефрагментаторов:

- **e2fsprogs**. Онлайн дефрагментатор из стандартного набора утилит.
- **Defrag**. Дефрагментатор от Кона Коливаса. Случаются перерывы в разработке, но проект живой, в 2010 появился PPA-репозиторий: ppa:e2defrag/ppa.
- **Shake**. Онлайн-дефрагментатор, PPA-репозиторий: ppa:un-brice/ppa.

e2fsprogs

- Проверка необходимости дефрагментации:

```
$ sudo e4defrag -c /dev/<xxx>
```

- запуск дефрагментации:

```
$ sudo e4defrag /dev/<xxx>
```

- проверка результата (в последней строке вывода должно быть ⩽0.3% non-contiguous):

```
$ sudo fsck -n /dev/<xxx>
```

Оптимизация дискового пространства

На свежесозданном разделе с Ext4 обычно выделяется заметно меньше свободного дискового пространства чем размер раздела. Это происходит потому, что используется порядка 1,8% на служебные нужды: заголовки групп блоков, бинарные поля для учета свободного места, индексные дискрипторы (i-node), основной и множество резервных копий суперблока. Также по умолчанию, дополнительно резервируется 5% от объема раздела для нужд учетной записи суперпользователя (root) и системных служб (daemons). Обычно это полезно для системного корневого раздела, но на пользовательских разделах необходимости в этом резерве нет, особенно на разделах большого размера, где 5% превращаются в очень приличные объемы (недоступные пользователю).

Изменение процента зарезервированных блоков для нужд суперпользователя

По умолчанию резервируется 5% от объема раздела или диска. Значение 0 отключает резервирование блоков, значение 5 - устанавливает резервирование на 5%.

```
$ sudo tune2fs -m 0 /dev/<xxx>
```

Изменение числа зарезервированных под служебные нужды блоков

Значение 0 отключает резервирование блоков, значение 5 - устанавливает резервирование 5-и блоков.

```
$ sudo tune2fs -r 0 /dev/<xxx>
```

Использование ext4

Для того чтобы использовать ext4, необходимо:

- поддержка со стороны ядра
- поддержка со стороны программ
- при создании файловой системы с нуля используется **mkfs.ext3** с ключом -E test_fs

```
# mkfs.ext3 -E test_fs file.img<console>
```

- для подготовки существующей файловой системы ext3 к монтированию использовать **debugfs**

```
# debugfs -w file.img
```

debugfs 1.40 (29-Jun-2007) debugfs: set_super_value s_flags 4 debugfs: quit

- при монтировании использовать тип файловой системы ext4dev

```
# mount -t ext4dev -o loop file.img /mnt
```

Ниже во всех деталях описывается процесс создания файловой системы ext4.

Для примера в качестве носителя файловой системы используется обычный файл. Создание файловой системы не в файле, а на дисковом разделе происходит аналогично, с той разницей, что не нужно использовать модуль *loop*.

Для того чтобы использовать файловую систему ext4, необходимо чтобы в ядре

Linux была соответствующая поддержка в ядре.

```
# uname -a
Linux dhcp 2.6.25-2-xen-686 #1 SMP Tue May 27 17:30:39 UTC 2008 i686 GNU/Linux

# modinfo ext4dev
filename:       /lib/modules/2.6.25-2-xen-686/kernel/fs/ext4/ext4dev.ko
license:        GPL
description:    Fourth Extended Filesystem with extents
author:         Remy Card, Stephen Tweedie, Andrew Morton, Andreas Dilger, Theodore Ts'o and others
depends:        mbcache,jbd2,crc16
vermagic:       2.6.25-2-xen-686 SMP mod_unload 686
```

Модуль ядра, который отвечает за поддержку ext4, называется *ext4dev*.

Создаём пустой файл размером 100MB,
который чуть позже будет форматироваться под ext4.

```
# dd if=/dev/zero of=file.img count=100 bs=1024k
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.43877 seconds, 239 MB/s
```

Создаём обычную файловую систему ext3:

```
# mkfs.ext3 file.img
mke2fs 1.40 (29-Jun-2007)
file.img is not a block special device.
Proceed anyway? (y,n) y
...
```

После того как файловая система создана, можно попробовать её подмонтировать. Поскольку мы создавали файловую систему в файле, а не на блочном устройстве, для того чтобы смонтировать её, файл нужно связать с loopback-устройством.

Это можно сделать или при помощи команды **losetup** или просто, использовав опцию loop при монтировании.

```
# mount -t ext4dev -o loop file.img /mnt
```

Если при выполнении команды возникает ошибка, в которое сообщается о проблемах с устройством /dev/loop, возможно, у вас просто не загружен модуль loop или не установлен udev:

```
# mount -t ext4dev -o loop file.img /mnt
mount: could not find any device /dev/loop#

# modprobe loop
[   11.636764] loop: module loaded

# mount -t ext4dev -o loop file.img /mnt
mount: could not find any device /dev/loop#

# apt-get install udev
```

Может возникнуть и другая ошибка, которая имеет непосредственное отношение

к файловой системе ext4.

```
# mount -t ext4dev -o loop file.img /mnt
[    14.531109] EXT4-fs: loop0: not marked OK to use with test code.
```

Модуль файловой системы ext4 ядра Linux сообщает о том, что файловая система не помечена как ext4. Проверка добавлена скорее из психологических чем технических побуждений — чтобы файловые системы ext3 не смонтировали случайно как ext4.

Добавить пометку, извещающую систему о том, что файловую систему можно смело монтировать как ext4

можно так:

```
# debugfs -w file.img
debugfs 1.40 (29-Jun-2007)
debugfs:
debugfs:  set_super_value s_flags 4
debugfs:  quit
```

Вообще, можно было попросить ставить пометку ещё при создании файловой системы.

Для этого нужно было использовать ключ -E test_fs:

```
# mkfs.ext3 -E test_fs file.img
```

Тип файловой системы, указываемый при монтировании: ext4dev.

```
# mount -t ext4dev -o loop file.img /mnt
[   16.769093] kjournald2 starting.  Commit interval 5 seconds
[   16.769093] EXT4 FS on loop0, internal journal
[    16.769093] EXT4-fs: mounted filesystem with ordered data mode.
[   16.769093] EXT4-fs: file extents enabled
[   16.769093] EXT4-fs: mballoc enabled
```

Файловая система успешно смонтирована:

```
# mount
/dev/xvda1 on / type ext3 (rw,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
udev on /dev type tmpfs (rw,mode=0755)
nfsd on /proc/fs/nfsd type nfsd (rw)
rpc_pipefs on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/root/file.img on /mnt type ext4dev (rw,loop=/dev/loop0)
```

После того как работа с файловой системой окончена, её нужно размонтировать[2].

```
# umount /mnt
[   16.974054] EXT4-fs: mballoc: 0 blocks 0 reqs (0 success)
[    16.974070] EXT4-fs: mballoc: 0 extents scanned, 0 goal hits, 0 2^N hits, 0 breaks, 0 lost
[   16.974075] EXT4-fs: mballoc: 0 generated and it took 0
[   16.974085] EXT4-fs: mballoc: 0 preallocated, 0 discarded
```

Поддержка в операционных системах

- coLinux — технология, позволяющая запустить ядро Linux под управлением ОС Windows. Возможен доступ к томам с ФС ext2/ext3/ext4.

- Ext2read — программа, предоставляющая доступ к ФС ext2/ext3/ext4 и LVM2, но в режиме только для чтения, с интерфейсом, напоминающим проводник Windows.
- Ext2Fsd — экспериментальный драйвер, добавляющий поддержку ФС ext2/ext3/ext4 для ОС Windows. Драйвер поддерживает не все возможности ФС.
- Paragon ExtFS for Windows — бесплатное для личного пользования проприетарное ПО, предоставляющее доступ к ФС ext2/ext3/ext4 для чтения и записи.

Поддержка ext4 в дистрибутивах Linux

- **Ubuntu:**
  - 9.04: ext4 доступна для выбора по желанию пользователя;
  - 9.10+: ext4 стала ФС по умолчанию.
- **Debian:**
  - 6.0: ext4 доступна для выбора по желанию пользователя;
  - 7.0+: ext4 предлагается, как ФС по умолчанию.
- **Fedora:**
  - >=9: ext4 доступна для выбора по желанию пользователя;
  - 11+: ext4 — ФС по умолчанию.

11.8. Файловая система Windows NT

Windows поддерживает несколько файловых систем, самыми важными из которых являются **FAT-16**, **FAT-32** и **NTFS** (NT File System). FAT-16 — это старая файловая система операционной системы MS-DOS. Она использует 16-битные дисковые адреса, что ограничивает размер дисковых разделов 2 Гбайт. В основном она применяется для доступа к флоппи-дискам (для тех клиентов, которые их до сих пор используют). FAT- 32 использует 32-битные дисковые адреса и поддерживает дисковые разделы размером до 2 Тбайт. FAT-32 не имеет никакой системы безопасности, и на сегодняшний день она фактически используется только для переносных носителей (таких, как флеш-диски). Файловая система NTFS была разработана специально для версии Windows NT. Начи-

ная с Windows XP ее по умолчанию устанавливает большинство производителей компьютеров, она существенно увеличивает безопасность и функциональность Windows. NTFS использует 64-битные дисковые адреса и теоретически может поддерживать дисковые разделы размером до 2^{64} байт (однако некоторые соображения ограничивают этот размер до более низких значений).

В этом разделе мы изучим файловую систему NTFS, так как это современная файловая система со многими интересными функциональными возможностями и конструктивными новшествами. Это большая и сложная файловая система, но объем книги не позволяет описать все ее функциональные возможности, однако приведенный далее материал должен дать вам довольно хорошее представление о ней.

11.8.1. Фундаментальные концепции

Имена файлов в NTFS ограничены 255 символами; размер полного маршрута ограничен 32 767 символами. Имена файлов хранятся к кодировке Unicode, что позволяет в тех странах, где не используется латинский алфавит (например, Греции, Японии, Индии, России и Израиле), писать имена файлов на своем языке. Например, фЛе — это допустимое имя файла. NTFS полностью поддерживает чувствительные к регистру имена (так что foo отличается от Foo и FOO). Интерфейс прикладного программиро

вания Win32 не полностью поддерживает чувствительность к регистру имен файлов и вовсе не поддерживает ее для имен каталогов. Поддержка чувствительности к регистру имеется при работе подсистемы POSIX (для совместимости с UNIX). Win32 не является чувствительным к регистру, но сохраняет регистр, так что имена файлов могут иметь в своем составе буквы разных регистров. Несмотря на то что чувствительность к регистру хорошо знакома пользователям UNIX, она очень неудобна для обычных пользователей, которые обычно не делают таких различий. Например, почти весь современный Интернет не имеет чувствительности к регистру.

Файл в NTFS — это не просто линейная последовательность байтов (как файлы в FAT- 32 и UNIX). Файл состоит из множества атрибутов, каждый из которых представлен потоком байтов. Большинство файлов имеет несколько коротких потоков (таких, как название файла и его 64-битный идентификатор объекта) плюс один длинный (неименованный) поток с данными. Однако файл может иметь также два или более длинных потока данных. Каждый поток имеет имя, состоящее из имени файла, двоеточия и имени потока (как, например, foo:stream1). Каждый поток имеет размер и может блокироваться независимо от всех остальных потоков. Идея множества потоков в NTFS не нова. Файловая система компьютеров Apple Macintosh использует два потока на файл (ветвь данных и ветвь ресурсов). Первоначально потоки в NTFS применялись для того, чтобы файловый сервер NT мог обслуживать клиентов Macintosh. Множественность потоков данных используется также для того, чтобы представлять метаданные файлов, такие как контрольные картинки изображений в формате JPEG (которые есть в графическом интерфейсе пользователя Windows). Однако, к сожалению, множественные потоки данных уязвимы, и они часто теряются при переносе в другие файловые системы или по сети (и даже при резервном копировании и последующем восстановлении, поскольку многие утилиты игнорируют их).

NTFS — это иерархическая файловая система, похожая на

файловую систему UNIX. Однако разделителем компонентов имени является знак «\», а не «/» (это атавизм, унаследованный от требований совместимости с операционной системой CP/M в период создания MS-DOS, поскольку в CP/M прямой слеш использовался для ключей командной строки). В отличие от UNIX, здесь концепции текущего рабочего каталога, жестких ссылок на текущий каталог (.) и родительский каталог (..) реализованы как соглашения, а не как фундаментальная часть файловой системы. Жесткие ссылки поддерживаются, но используются только для подсистемы POSIX, так же как и поддержка проверки обхода каталогов (разрешение ‘x’ в UNIX).

Символические ссылки для NTFS поддерживаются. Создание символических ссылок обычно разрешается только администраторам (во избежание проблем с безопасностью типа спуфинга, которые появились в UNIX, когда в версии BSD 4.2 были введены символические ссылки). Реализация символических ссылок использует функциональную возможность NTFS под названием **«точка повторной обработки»** (reparse points), которая обсуждается далее в этом разделе. Кроме того, поддерживаются также сжатие, шифрование, отказоустойчивость, журналирование и разреженные файлы. Это функциональные возможности, и их реализацию мы скоро обсудим.

11.8.2. Реализация файловой системы NTFS

NTFS — это очень сложная файловая система, которая была разработана специально для NT как альтернатива файловой системе HPFS, которая была разработана для OS/2.

В то время как большая часть NT была создана на суше, NTFS является уникальным компонентом операционной системы, потому что большая часть ее проектирования происходила на борту парусной шлюпки в проливе Puget Sound (причем соблюдался строгий протокол: утром — работа, после обеда — отдых). Далее мы изучим функциональные возможности NTFS, начиная с ее структуры, затем перейдем к поиску имен файлов, сжатию файлов, журналированию и шифрованию файлов.

Структура файловой системы

Каждый том NTFS (например, дисковый раздел) содержит файлы, каталоги, битовые массивы и другие структуры данных. Каждый том организован как линейная последовательность блоков (которые в терминологии компании Microsoft называются кластерами), причем размер блоков для каждого тома фиксирован (в зависимости от размера тома он может изменяться от 512 байт до 64 Кбайт). Большинство дисков NTFS использует блоки размером 4 Кбайт — это компромисс между применением больших блоков (для эффективной передачи данных) и использованием маленьких блоков (для снижения внутренней фрагментации). Ссылки на блоки делаются с использованием смещения от начала тома (при помощи 64-битных чисел).

Главная структура данных каждого тома — это **MFT** (Master File Table — главная таблица файлов), которая является линейной последовательностью записей фиксированного размера (1 Кбайт). Каждая запись MFT описывает один файл или один каталог. Она содержит атрибуты файла (такие, как его имя и временная метка), а также список дисковых адресов (где расположены его блоки). Если файл очень большой, то иногда приходится использовать две или более записи MFT (чтобы разместить в них список всех блоков). В этом случае первая запись в MFT, называемая **основной записью** (base record), указывает на дополнительные записи в MFT. Такая схема переполнения ведет свое начало из CP/M, где каждый элемент каталога назывался экстентом. Битовый массив отслеживает свободные элементы MFT.

Сама MFT также является файлом и в качестве такового может быть размещена в любом месте тома (таким образом устраняется проблема наличия дефектных секторов на первой дорожке). Более того, при необходимости этот файл может расти (до максимального размера в 2^{48} записей).

MFT показана на рис. 11.24. Каждая запись MFT состоит из последовательности пар (заголовок атрибута — значение). Каждый атрибут начинается с заголовка, рассказывающего о том, что это за

атрибут и какую длину имеет его значение. Некоторые значения атрибутов (такие, как имя файла и его данные) имеют переменную длину. Если значение атрибута достаточно короткое для того, чтобы уместиться в записи MFT, то оно помещается именно туда. Если же значение слишком длинное, то оно размещается на диске, а в запись MFT помещается указатель на него. Это делает систему NTFS очень эффективной для небольших полей, которые могут разместиться в самой записи MFT.

Первые 16 записей MFT резервируются для файлов метаданных NTFS (см. рис. 11.24). Каждая из этих записей описывает нормальный файл, который имеет атрибуты и блоки данных (как и любой другой файл). Каждый из этих файлов имеет имя, которое начинается со знака доллара (чтобы обозначить его как файл метаданных). Первая запись описывает сам файл MFT. В частности, в ней говорится, где находятся блоки файла MFT (чтобы система могла найти файл MFT). Очевидно, что Windows нужен способ нахождения первого блока файла MFT, чтобы найти остальную информацию по

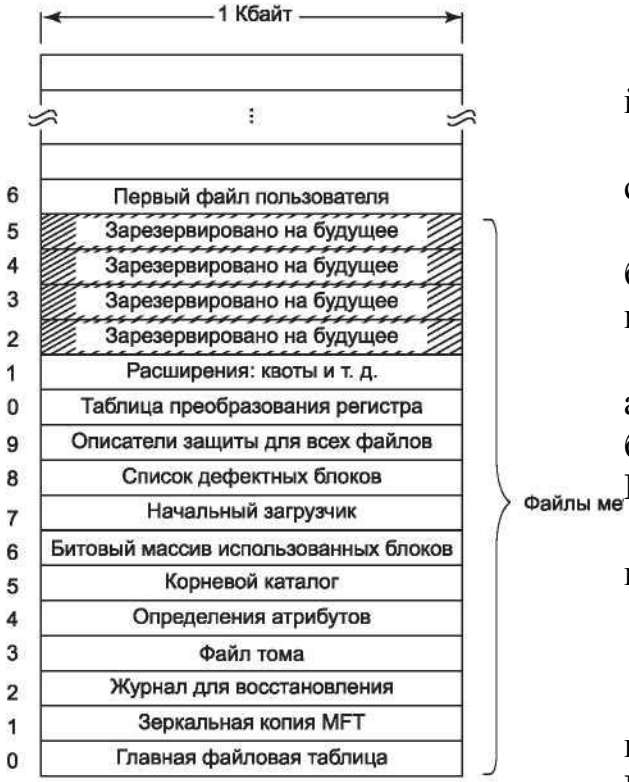


Рис. 11.24. Главная таблица файлов NTFS

файловой системе. Windows смотрит в загрузочном блоке — именно туда записывается адрес первого блока файла MFT при форматировании тома.

Запись 1 является дубликатом начала файла MFT. Эта информация настолько ценная, что наличие второй копии может быть просто критическим (в том случае, если один из первых блоков MFT перестанет читаться). Вторая запись — файл журнала. Когда в файловой системе происходят структурные изменения (такие, как добавление нового или удаление существующего каталога), то такое действие журналируется здесь до его выполнения (чтобы повысить вероятность корректного восстановления в случае сбоя во время операции, например такого, как отказ системы). Здесь также журналируются изменения в файловых атрибутах. Фактически не журналируются здесь только изменения в пользовательских данных. Запись 3 содержит информацию о томе (такую, как его размер, метка и версия).

Как уже упоминалось, каждая запись MFT содержит последовательность пар «заголовок атрибута — значение».

Атрибуты определяются в файле $AttrDef. Информация об этом файле содержится в MFT (в записи 4). Затем идет корневой каталог, который сам является файлом и может расти до произвольного размера. Он описывается записью номер 5 в MFT.

Свободное пространство тома отслеживается при помощи битового массива. Сам битовый массив — тоже файл, его атрибуты и дисковые адреса даны в записи 6 в MFT.

Следующая запись MFT указывает на файл начального загрузчика. Запись 8 используется для того, чтобы связать вместе все плохие блоки (чтобы обеспечить невозможность их использования для файлов). Запись 9 содержит информацию безопасности. Запись 10 используется для установления соответствия регистра. Для латинских букв A — Z соответствие регистра очевидно (по крайней мере для тех, кто разговаривает на романских языках). Однако соответствие регистров для других языков (таких, как греческий, армянский или грузинский) для говорящих на романских языках не столь очевидно, поэтому данный файл рассказывает, как это сделать. И наконец, запись 11 — это каталог, содержащий различные файлы для таких вещей, как дисковые квоты, идентификаторы объектов, точки повторной обработки и т. д. Последние четыре записи MFT зарезервированы для использования в будущем.

Каждая запись MFT состоит из заголовка записи, за которым следуют пары «заголовок атрибута — значение». Заголовок записи содержит системный код, используемый для проверки достоверности, последовательный номер (обновляемый каждый раз, когда запись используется для нового файла), счетчик количества ссылок на файл, фактическое количество использованных в записи байтов, идентификатор (индекс, порядковый номер) основной записи (используется только для записей расширения), а также некоторые другие поля.

NTFS определяет 13 атрибутов, которые могут появиться в записях MFT. Они перечислены в табл. 11.16. Каждый заголовок атрибута идентифицирует атрибут и содержит длину и местоположение поля значения, а также разнообразные флаги и прочую информацию. Обычно значения атрибутов следуют непосредственно за своими заголовками атрибутов, но если значение слишком длинное для того, чтобы поместиться в запись

Атрибу	**Описание**
Standar д	Биты флагов, временные метки и т. д.
File	Имя файла в Unicode, может повторяться
Security	Устарел. Информация безопасности

MFT, то оно может быть размещено в отдельных дисковых блоках. Такой атрибут называется **нерезидентным атрибутом** (nonresident attribute). Очевидно, что таким атрибутом является атрибут данных.

Таблица 11.16. Используемые в записях MFT атрибуты

	исание
Standar д	Биты флагов, временные метки и т. д.
File	Имя файла в Unicode, может повторяться
Security	Устарел. Информация безопасности
Attribut	Местоположение дополнительных
Object	Уникальный для данного тома 64-битный
Reparse	Используется для монтирования и
Volume	Название данного тома (используется
Volume	Версия тома (используется только в
Index	Используется для каталогов
Index	Используется для очень больших
Bitmap	Используется для очень больших
Logged	Управляет журналированием в $LogFile
Data	Данные потока, могут повторяться

Некоторые атрибуты (такие, как имя) могут

повторяться, но все атрибуты должны присутствовать в записи MFT в определенном порядке. Заголовки резидентных атрибутов имеют длину 24 байта, заголовки нерезидентных атрибутов длиннее (поскольку они содержат информацию о том, где нужно искать атрибут на диске).

Стандартное информационное поле содержит: сведения о владельце файла, информацию безопасности, нужные для POSIX временные метки, количество жестких ссылок, биты архивирования и «только для чтения» и т. д. Это поле имеет фиксированную длину и присутствует всегда. Имя файла — это строка переменной длины в коде Unicode. Для того чтобы файлы с не соответствующими правилам MS-DOS именами могли быть доступны старым 16-битным программам, они могут иметь **короткие имена** (short name) по принятой в MS-DOS схеме 8 + 3. Если реальное имя файла соответствует схеме именования в MS-DOS (8 + 3), то второе имя MS-DOS не нужно.

В NT 4.0 информация безопасности размещалась в атрибуте, но в Windows 2000 и более поздних вся информация безопасности размещается в одном файле (чтобы она могла совместно использоваться многими файлами). Это приводит к существенной экономии места во многих записях MFT и в файловой системе в целом, поскольку информация безопасности идентична для большого количества принадлежащих одному пользователю файлов.

Список атрибутов нужен в том случае, когда атрибуты не помещаются в запись MFT. Из этого атрибута можно узнать, где искать записи расширения. Каждый элемент списка содержит 48-битный индекс по MFT (который говорит о том, где находится запись расширения) и 16-битный порядковый номер (для проверки того, что запись расширения соответствует базовой записи).

Файлы NTFS имеют связанный с ними идентификатор, который подобен номеру узла i-node в UNIX. Файлы можно открывать по идентификатору, но присваиваемый файловой системой NTFS идентификатор не всегда можно использовать, поскольку он основан на записи MFT и может измениться при перемещении записи для данного файла (например, если файл восстанавливается из резервной копии). NTFS позволяет использовать отдельный атрибут «идентификатор объекта», который может быть установлен для файла и который нет необходимости изменять. Его можно сохранить вместе с файлом (например, если он копируется на новый том).

Точка повторной обработки сообщает разбирающей имя файла процедуре о необходимости сделать что-то особенное. Этот механизм используется для явного монтирования файловых систем и для символических ссылок. Два атрибута тома используются только для идентификации томов. Следующие три атрибута работают с реализацией каталогов. Маленькие каталоги — это просто списки файлов, а большие реализованы как деревья B+. Атрибут logged utility stream используется шифрующей файловой системой.

И наконец, мы подошли к атрибуту, который важнее всех: потоку (или

потокам) данных. Файл в NTFS имеет один (или несколько) связанных с ним потоков данных. Именно здесь находится его полезное содержание. **Поток данных по умолчанию** (default data stream) названия не имеет (например, *dirpath\filename::$DATA),* но **альтернативные потоки данных** (alternate data stream) имеют имена, например: *dirpath\ filename:streamname:$DA TA.*

Имя каждого потока (если оно имеется) находится в заголовке этого атрибута. Следом за заголовком идет либо список дисковых адресов (это содержащиеся в потоке блоки), либо (для потоков всего в несколько сотен байтов, а таких много) сам поток.

Размещенные в записи MFT реальные данные потока называются **непосредственным файлом** — immediate file (Mullender and Tanenbaum, 1984).

Конечно, в основном данные не помещаются в запись MFT, поэтому данный атрибут обычно нерезидентный. Теперь давайте рассмотрим, как NTFS отслеживает местоположение нерезидентных атрибутов.

Выделение дискового пространства

Модель отслеживания дисковых блоков состоит в том, что они выделяются последовательными участками, насколько это возможно (из соображений эффективности). Например, если первый логический блок потока помещен в блок 20 диска, то система будет очень стараться поместить второй логический блок в блок 21, третий логический блок — в блок 22 и т. д. Одним из способов достижения непрерывности этих участков является выделение дискового пространства по несколько блоков за один раз (по мере возможности).

Блоки потока описываются последовательностью записей, каждая их которых описывает последовательность логически смежных блоков. Для потока без пропусков будет только одна такая запись. К этой категории принадлежат такие потоки, которые записаны по порядку с начала и до конца. Для потока с одним пропуском (например, определены только блоки 0-49 и блоки 60-79) будет две записи. Такой поток может быть получен при помощи записи первых 50 блоков, а затем пропуска до 60-го блока и записи еще 20 блоков. Когда такой пропуск считывается, все недостающие байты — нулевые. Файлы с пропусками называются **разреженными файлами** (sparse files).

Каждая запись начинается с заголовка, в котором дается смещение первого блока потока. Затем идет смещение первого не описанного данной записью блока. В приведенном ранее примере в первой записи будет заголовок (0, 50) и будут даны дисковые адреса этих 50 блоков. Во второй записи будет заголовок (60, 80) и дисковые адреса этих 20 блоков.

За заголовком записи следует одна или несколько пар (в каждой даются дисковый адрес и длина участка). Дисковый адрес — это смещение дискового блока от начала раздела, длина участка — это количество блоков в участке. В записи участка может быть столько пар, сколько необходимо. Использование этой схемы для потока из трех участков и девяти блоков показано на рис. 11.25.

На этом рисунке у нас есть запись MFT для короткого потока из девяти блоков (заголовок 0-8). Он состоит из трех участков последовательных блоков на диске. Первый участок — блоки 20-23, второй — блоки 64-65, третий — блоки 80-82. Каждый из этих участков заносится в запись MFT как пара (дисковый адрес, количество блоков). Количество участков зависит от того, насколько хорошо справился со своей работой модуль выделения блоков при создании потока. Для потока из *n* блоков количество участков может составлять от 1 до n.

Здесь нужно сделать несколько замечаний. Во-первых, для представленных таким способом потоков нет верхнего ограничения размера. Если не использовать сжатие адресов, то для каждой пары требуется два 64-битных числа (всего 16 байт). Однако пара может представлять 1 млн (или более) смежных дисковых блоков. Фактически состоящий из 20 отдельных участков (каждый по 1 млн блоков размером 1 Кбайт) поток размером 20 Мбайт легко помещается в одну запись MFT, а разбросанный по 60 изолированным блокам поток размером 60 Кбайт в одну запись MFT не помещается.

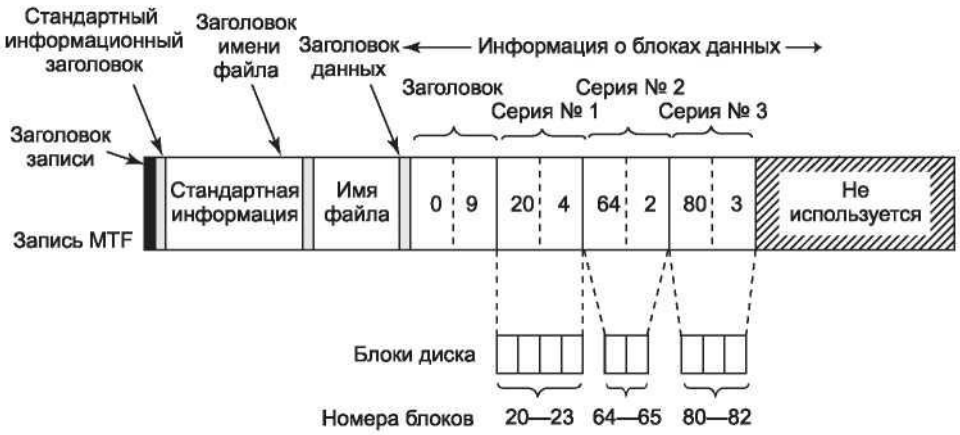


Рис. 11.25. Запись MFT для потока из трех участков и девяти блоков

Во-вторых, в то время как самый простой способ представления каждой пары требует 2 • 8 байт, имеется также метод сжатия, который уменьшает размер пары меньше чем до 16 байт. Многие дисковые адреса имеют нулевые старшие байты. Их можно опустить. Заголовок данных сообщает о том, сколько их было опущено (то есть сколько байтов реально используется на один адрес). Применяются и другие виды сжатия. На практике пара часто занимает только 4 байта.

Наш первый пример был простым: вся информация файла уместилась в одной записи MFT. Что произойдет, если файл настолько большой или так сильно фрагментирован, что информация о блоках не помещается в одну запись MFT? Ответ простой: используются две или более записи MFT. На рис. 11.26 мы видим файл, основная запись которого находится в записи 102 в MFT. Он имеет слишком много (для одной записи MFT) участков, поэтому вычисляется количество нужных записей расширения (например, две) и их индексы помещаются в основную запись. Остальная часть записи используется для первых *k* участков данных.

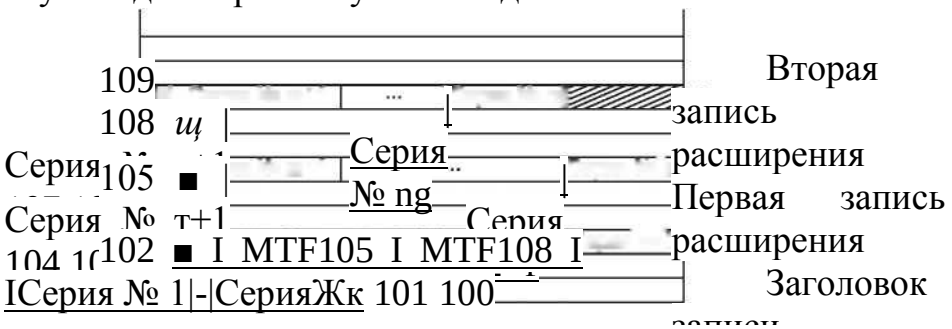


Рис. 11.26. Файл, которому требуется три записи MFT для хранения всех его участков

Обратите внимание на то, что на рис. 11.26 имеется некоторая избыточность. В теории не должно быть необходимости указывать конец последовательности участков, поскольку эту информацию можно вычислить по парам для участков. Цель избыточного

Обратите внимание на то, что на рис. 11.26 имеется некоторая избыточность. В теории не должно быть необходимости указывать конец последовательности участков, поскольку эту информацию можно вычислить по парам для участков. Цель избыточного указания этой информации в том, чтобы сделать поиск более эффективным: чтобы

найти блок по заданному смещению в файле, нужно обследовать только заголовки записей (а не пары для участков).

Когда все пространство записи 102 будет использовано, сохранение участков продолжится в записи 105 в MFT. В нее будет записано столько участков, сколько поместится. Когда эта запись также заполнится, остальные участки попадут в запись 108 в MFT. Таким образом можно использовать много записей MFT (для работы с большими фрагментированными файлами).

Если нужно очень много записей MFT, то появляется проблема: может не хватить места в основной MFT для размещения всех их индексов. Для этой проблемы также есть решение: список записей расширения MFT делается нерезидентным, то есть хранится в других дисковых блоках (вместо основной записи MFT). В этом случае он может увеличиваться настолько, насколько это нужно.

Элемент MFT для небольшого каталога показан на рис. 11.27. Запись содержит некоторое количество элементов каталога, каждый из которых описывает один файл или каталог. Каждый элемент содержит структуру фиксированной длины, за которой следует имя файла (переменной длины). Фиксированная часть содержит индекс элемента MFT для данного файла, длину имени файла, а также разнообразные прочие поля и флаги. Поиск элемента каталога состоит из опроса всех имен файлов по очереди. Большие каталоги используют другой формат. Вместо линейного перечисления файлов используется дерево B+ (чтобы сделать возможным алфавитный поиск и облегчить вставку новых имен в нужное место каталога).

Рис. 11.27. Запись MFT для небольшого каталога

В NTFS разбор маршрута \foo\bar начинается в корневом каталоге C:, блоки которого можно определить из элемента 5 в MFT (см. рис. 11.24). Строка «foo» ищется в корневом каталоге, который возвращает индекс в MFT для каталога foo. В этом каталоге затем выполняется поиск строки «bar», которая ссылается на запись MFT для данного файла. NTFS выполняет проверки доступа (обращаясь к монитору безопасности) и, если все в порядке, ищет запись MFT для атрибута *::$DATA*, который является потоком данных по умолчанию.

Обнаружив файл bar, NTFS установит указатели на свои метаданные в объекте файла, переданном из диспетчера ввода-вывода. Метаданные включают указатель на запись MFT, информацию по сжатию и блокировке диапазонов, различные подробности о совместном использовании и т. д.

Большинство этих метаданных содержится в структурах данных, совместно используемых всеми ссылающимися на этот файл объектами файлов. Несколько полей специфичны только для текущего открытого файла: например, следует ли файл удалить после его закрытия. После того как открытие успешно произошло,

NTFS вызывает *IoCompleteRequest* для передачи IRP обратно вверх по стеку ввода-вывода (в диспетчеры ввода-вывода и объектов). В итоге описатель для объекта файла помещается в таблицу описателей для текущего процесса, и управление передается обратно в пользовательский режим. При последующих вызовах *ReadFile* приложение может предоставлять описатель, указывая, что этот объект файла для C:\foo\bar следует включать в запрос чтения, который передается вниз по стеку устройства C: в NTFS.

В дополнение к обычным файлам и каталогам NTFS поддерживает и жесткие ссылки (в UNIX-смысле), а также символические ссылки (с использованием механизма под названием **точка повторной обработки** — reparse points). NTFS поддерживает пометку файла или каталога как точки повторной обработки и ассоциирование с ней блока данных. Когда такой файл или каталог встречается во время разбора имени файла, операция заканчивается неудачей и диспетчеру объектов возвращается этот блок данных. Диспетчер объектов может интерпретировать данные как представляющие альтернативный маршрут, после чего он обновляет строку для разбора и повторяет операцию ввода-вывода. Этот механизм используется для поддержки как символических ссылок, так и смонтированных файловых систем, выполняя перенаправление поиска в другую часть иерархии каталогов или даже в другой раздел диска.

Точки повторной обработки используются также для пометки отдельных файлов для драйверов-фильтров файловой системы. На рис. 11.11 мы показали, как фильтры файловой системы можно установить между диспетчером ввода-вывода и файловой системой. Запросы ввода-вывода завершаются при помощи вызова *IoCompleteRequest,* передающего управление процедурам завершения, которые каждый драйвер представил в стеке устройств (вставленном в IRP во время запроса). Желающий пометить файл драйвер ассоциирует тег повторной обработки, а затем отслеживает закончившиеся неудачно (потому что им встретилась точка повторной обработки) запросы на завершение операций открывания файлов. Из блока (передаваемых обратно с IRP) данных драйвер может определить, тот ли это блок данных, который сам драйвер ассоциировал с файлом. Если это так, то драйвер останавливает обработку завершения и продолжает обработку исходного запроса ввода-вывода. Обычно при этом продолжается выполнение запроса на открытие, но имеется флаг, который дает указание NTFS игнорировать точку повторной обработки и открыть файл.

NTFS поддерживает прозрачное сжатие файлов. Файл может создаваться в сжатом режиме, а это означает, что NTFS пытается автоматически сжать блоки при их записи на диск и автоматически распаковывает их при чтении обратно. Те процессы, которые читают или

пишут сжатые файлы, совершенно не в курсе того факта, что происходит сжатие или распаковка.

Сжатие работает следующим образом. Когда NTFS пишет файл (помеченный как сжатый) на диск, то она изучает первые 16 (логических) блоков файла — независимо от того, сколько участков они занимают. Затем запускает по ним алгоритм сжатия. Если полученные данные можно записать в 15 или менее блоков, то сжатые данные записываются на диск (по возможности — одним участком). Если сжатые данные по-прежнему занимают 16 блоков, то эти 16 блоков записываются в несжатом виде. Затем исследуются блоки 16-31, чтобы узнать, можно ли их сжать до размера 15 блоков (или менее), и т. д.

На рис. 11.28, *а* показан файл, в котором первые 16 блоков успешно сжались до 8 блоков, вторые 16 блоков не сжались, а третьи 16 блоков также сжались на 50 %. Эти три части были записаны как три участка и сохранены в записи MFT. «Отсутствующие» блоки хранятся в элементе MFT с дисковым адресом 0 (рис. 11.28, б). Здесь за заго-

ловком (0, 48) следует пять пар: две для первого (сжатого) участка, одна для несжатого участка и две для по

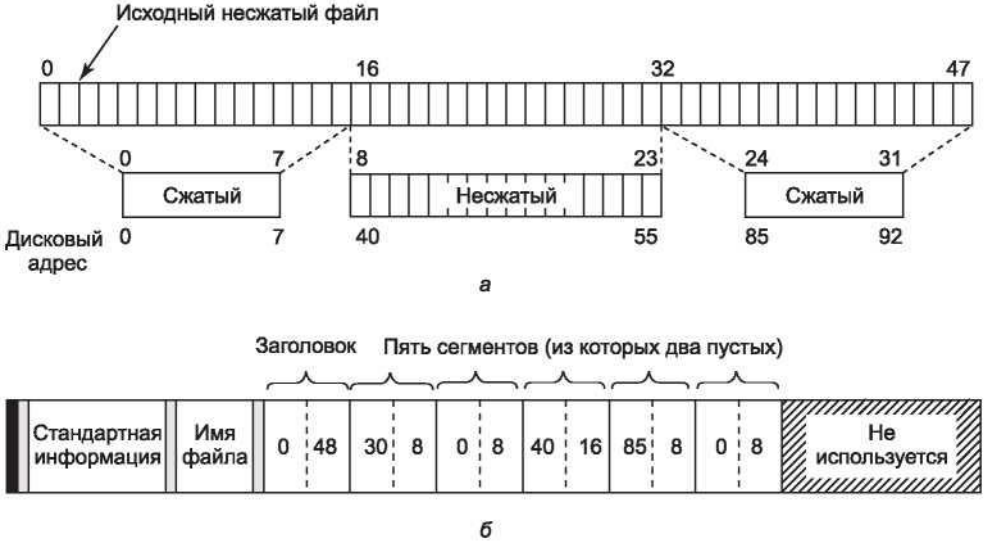


Рис. 11.28. *а* — пример сжатия файла из 48 блоков до размера в 32 блока; *б* — запись MFT для файла после сжатия

Когда файл считывается обратно, NTFS должна знать, какие участки сжаты, а какие — нет. Она может определить это по дисковым адресам. Дисковый адрес 0 указывает на то, что это последняя часть 16 сжатых блоков. Дисковый блок 0 не может использоваться для хранения данных (во избежание неоднозначности). Поскольку блок 0 тома содержит загрузочный сектор, использование его для данных в любом случае невозможно.

Получить произвольный доступ к сжатым файлам возможно, но это сложно. Предположим, что процесс выполняет поиск блока 35 (см. рис. 11.28). Как NTFS найдет блок 35 в сжатом файле? Сначала ей нужно прочитать и распаковать весь участок. Затем она узнает, где находится блок 35, и сможет передать его в любой процесс (который его прочитает). Выбор в качестве единицы сжатия 16 блоков — это компромисс: меньший размер снизил бы эффективность сжатия, больший сделал бы произвольный доступ еще более дорогим.

Журналирование

NTFS поддерживает два механизма, при помощи которых программы могут обнаружить изменения в файлах и каталогах. Первый — это операция *NtNotifyChangeDirectoryFile,* передающая системе буфер, который возвращается после обнаружения изменения в каталоге или подкаталоге. В результате ввода-вывода буфер заполняется списком записей об изменениях. Если он очень маленький, записи теряются.

Второй механизм — это журнал изменений NTFS. NTFS содержит список всех записей об изменениях для каталогов и файлов тома в специальном файле, который программы могут читать при помощи специальных операций управления файловой системой

(опция *FSCTL_QUERY_USN_JOURNAL* вызова *NtFsControlFile).* Файл журнала обычно очень большой, поэтому вероятность затирания записей до того, как они будут изучены, очень мала.

Шифрование файлов

В наши дни компьютеры используются для хранения самой разнообразной конфиденциальной информации, в том числе планов корпоративных поглощений, налоговой информации, любовных писем (которые их владельцы не хотят никому показывать). Потеря информации может произойти при утрате или краже ноутбука, перезагрузке настольного компьютера с флоппи-диска MS-DOS (чтобы обойти систему безопасности Windows) либо при физическом переносе жесткого диска с одного компьютера на другой (с небезопасной операционной системой).

Windows решает эти проблемы при помощи опции шифрования файлов, чтобы даже в случае кражи или загрузки в MS-DOS файлы были нечитаемыми. Для использования шифрования обычным способом необходимо пометить нужные каталоги как зашифрованные, после чего все находящиеся в них файлы будут зашифрованы, а новые (переносимые или создаваемые файлы) также будут шифроваться. Шифрование и расшифровка управляются не файловой системой NTFS, а драйвером под названием **EFS** (Encryption File System), который регистрирует обратные вызовы в NTFS.

EFS шифрует конкретные файлы и каталоги. В Windows есть еще одно средство шифрования под названием **BitLocker**, которое шифрует почти все данные тома, что может помочь защитить данные несмотря ни на что — если только пользователь использует механизмы сильных ключей. С учетом того, что множество компьютеров теряют или крадут, а также высокой опасности «кражи личности» (identify theft) обеспечение защиты секретов является очень важным