Лекция 4. (2 учебных часа - 1 ч 20 мин) Файловая система

1. Введение

Файловая система - это часть операционной системы, обеспечивающей организацию хранения и доступа к информации на различных носителях, пользовательский интерфейс при работе с данными, и обеспечения совместного использования файлов несколькими пользователями и процессами.

Рассмотрим файловые системы для наиболее распространенных в наше время носителей информации - магнитных дисков. Информация на жестком диске хранится в секторах и само устройство может выполнять лишь команды считать/записать информацию в определенный сектор на диске. В отличие от этого файловая система позволяет пользователю оперировать с более удобным для него понятием - файл. Файловая система берет на себя организацию взаимодействия программ с файлами, расположенными на дисках. Для идентификации файлов используются имена. Современные файловые системы предоставляют пользователям возможность давать файлам достаточно длинные мнемонические названия.

Под каталогом в ФС понимается, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений, с другой стороны каталог - это файл, содержащий системную информацию о группе составляющих его файлов. Файловые системы обычно имеют иерархическую структуру, в которой уровни создаются за счет каталогов, содержащих информацию о файлах и каталогах более низкого уровня.

1. Имена файлов

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими. Так в популярной файловой системе FAT длина имен ограничивается известной схемой 8.3 (8 символов - собственно имя, 3 символа - расширение имени). Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, в Windows в файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющем собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уникальное имя, так. чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы. Примером такого уникального имени файла в системе UNIX является inode - сокращение от Information NODE (информационный узел). Inode'bi хранятся на диске в таблице inode. Они существуют для всех типов файлов, которые могут храниться в файловой системе, включая каталоги, именованные каналы, файлы символьного режима и так далее. Отсюда вытекает другая известная фраза: Inode - это файл». При помощи inode'oв UNIX® идентифицирует файл уникальным способом.

4.3. Типы файлов

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы- каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - это файлы, ассоциированные с устройствами ввода- вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Каталог - это. с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр. или файлы, составляющие один программный пакет), а с другой стороны - это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

* информация о разрешенном доступе,
* пароль для доступа к файлу,
* владелец файла
* создатель файла,
* признак "только для чтения".
* признак "скрытый файл".

признак "системный файл", признак "архивный файл", признак "двоичный/символьный",

признак "временный" (удалить после завершения процесса), признак блокировки, длина записи,

указатель на ключевое поле в записи, длина ключа,

Имя файла

времена создания, последнего доступа и последнего изменения, текущий размер файла, максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в ОС UNIX (рисунок 4.1). Каталоги могут образовывать иерархическую структуру за счет тою, что каталог более низкого уровня может входить в каталог более высокого уровня (рисунок 4.2).

8 3 1 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Мня файла | | Расширение | | Атри­буты | Резервные |
| Резервные | Время | Дата | N первого блока | | Размер |

(а)

14

N индексного дескриптора

(б)

Рис. 4.1. Структура каталогов: а - структура записи каталога MS-DOS (32 байта); б - структура записи каталога ОС UNIX

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру, а в UNIX'e - сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

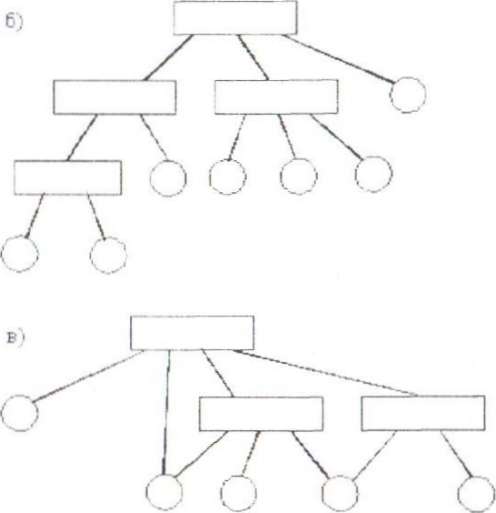
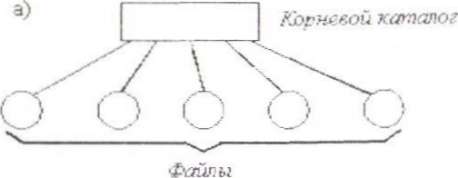


Рис. 4.2. Логическая организация файловой системы а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)

4.4. Логическая организация файла

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рисунке 4.3 показаны несколько схем логической организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В файловых системах ОС UNIX и MS-DOS файл имеет простейшую логическую структуру - последовательность однобайтовых записей.

I

I i

I I I I

Последовательность логических записей фиксированной длимы

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ш |  | 12 |  | 13 |  | 14 |  | 15 |  |
|  | и  4 |  | 12  « |  | 13  4 |  | 14  4 1 |  | 15 i 1 |

Последовательность логических записей переменной длины

ш

Индекс = ключ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  | 8 |  | б |  | 2 |  | 5 |  |

V

Индексная таблица запись I запись 2 запись 3 запись 4 запись 5 Индексная логическая организация

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Индекс | 1 | 2 | о я | 4 | 5 | 6 |
| Адрес | 21 | 201 | 315 | 661 | 670 | 715 |

Рис. 4.3. Способы логической организации файлов 4.5. Физическая организация и адрес файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей - блоков. Блок - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.

Непрерывное размещение - простейший вариант физической организации (рисунок 4.4. а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти (рисунок 4.4. б ). При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом - номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла: для того, чтобы прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся водном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программ! • читают данные блоками, размер которых равен степени двойки.

|  |  |
| --- | --- |
| 1 | |
| 4 | 2 |
|  | 3 |
| 6 | : |
| 9 | 6  7 |
| 8 | |
| - | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 |  | в | я |  |
|  |  |  |  |  |
|  |  |  |  | 1 |
| 2 | | | | |
|  |  |  |  | 3 |
|  |  |  |  | 4 |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |

г)

1

Облестъ j индексов

Файл

2,4.5.7

6

Рис. 4.4. Физическая организация файла а - непрерывное размещение; б - связанный список блоков: в - связанный список индексов; г - перечень номеров блоков

Популярным способом, используемым, например, в файловой системе FAT операционной системы MS-DOS. является использование связанного списка

г

индексов. С каждым блоком связывается некоторый элемент - индекс. Индексы располагаются в отдельной области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит имеют объем, равный степени двойки.

4.6. Права доступа к файлу

Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

* создание файла,
* уничтожение файла,
* открытие файла.
* закрытие файла,
* чтение файла,
* запись в файл,
* дополнение файла,
* поиск в файле,
* получение атрибутов файла.
* установление новых значений атрибутов,
* переименование,
* выполнение файла,
* чтение каталога,
* и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рисунок 4.5). В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

Имела файлов

class .Л f

Ч1ш: ppt

пю dem.txt

kna

выполнять

выполнять

genya

выполнять

выполнять читать

о

nataly

выполнять читать

victor

сиздать

читать писать

Рис. 4.5. Матрица прав доступа

Различают два основных подхода к определению прав доступа:

* 1. избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции:
  2. мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости оттого, к какой группе пользователь отнесен.

По сравнению с Windows® и большинством других операционных систем, в GNU/Linux работа с файлами организована совсем по-другому. Основные различия являются прямым следствием того факта, что Linux - это многопользовательская система: каждый файл является исключительной собственностью одного пользователя и одной группы. Еще один момент о пользователях и группах, который мы не упомянули, состоит в том, что каждый из них владеет личным каталогом (называемым домашним каталогом). \ 1ользователь является владельцем этого каталога и всех создаваемых в нем файлов. С ними также ассоциируется группа, которая является основной группой, к которой принадлежит пользователь. Пользователь может быть членом нескольких групп одновременно. Как владелец файла, пользователь может устанавливать права на файлы. Эти нрава распределяю геи между тремя категориями пользователей: владельцем файла; всеми пользователями, являющимися членами группы, ассоциированной с файлом (также называемой группой владельца), но не являющимися владельцами; и остальными, куда входят все остальные пользователи, которые не являются ни владельцами, ни членами группы владельца.

Существует три разновидности прав:

* + 1. Права на чтение (Read, г): пользователю разрешается читать содержимое файла. По отношению к каталогу это означает, что пользователь может просмотреть его содержимое (т.е. список файлов этого каталога).
    2. Права на запись (Write, w): разрешает изменять содержимое файла. По отношению к каталогу право на запись дает пользователю возможность добавлять или удалять файлы из этого каталога, даже если он не является владельцем этих файлов.
    3. Права на выполнение (eXecute, х): разрешает запуск файла (обычно только исполняемые файлы имеют этот тип прав доступа). По отношению к каталогу это дает пользователю возможность проходить его, что означает войти в этот каталог или пройти сквозь него. Обратите внимание, что это отличается от доступа на чтение: вы в состоянии пройти через каталог, но прочитать его содержимое все-таки не можете!

Возможны любые комбинации этих прав. Например, вы можете разрешить только чтение файла для себя и запретить доступ для всех других пользователей. Как владелец файла вы также можете изменить его группу (только если вы член устанавливаемой группы).

Ниже представлено выполнение команды Is -I в командной строке: Is -I total 1

-rw-r 1 queen users 0 Jul 8 14:11 a\_file

drwxr-xr— 2 peter users 1024 Jul 8 14:11 adirectory/

Результаты выполнения команды Is -I (слева направо):

* Первые десять символов представляют тип файла и назначенные ему права. Первый символ - это тип файла: если это обычный файл, вы увидите тире (-). Если это каталог, крайним левым символом будет d. Существуют и другие типы файлов, которые мы обсудим позже. Следующие девять символов представляют собой права доступа для данного файла. Эти девять символов на самом деле являются тремя группами из трех прав. Первая группа представляет права владельца файла; следующие три символа касаются всех пользователей, принадлежащих к группе владельца; и последние три символа относятся ко всем остальным. Знак тире (-) означает, что права доступа не установлены.
* Далее следует количество ссылок на файл. Позже мы увидим, что уникальный идентификатор файла - это не имя. а его номер (номер inode), и существует возможность иметь на диске несколько имен для одного файла. Для каталога количество ссылок имеет специальное значение, что также будет рассмотрено несколько позже.
* Следующая часть информации - это имя владельца файла и имя группы.
* И, наконец, далее показаны размер файла (в байтах), время его последнего изменения и имя самого файла или каталога в самом конце строки.

Давайте поближе рассмотрим права доступа для каждого из этих файлов: сначала мы должны отбросить первый символ, представляющий тип файла, и для файла afile мы получим следующие права: rw-r . Ниже представлена схема организации прав:

* первые три символа (rw-) - это права владельца, которым в данном случае является queen. Следовательно, queen может читать файл (г), изменять его содержимое (w). но не может запускать его (-).
* следующие три символа (г--) относятся к любому пользователю, кроме queen, который является членом группы users. Он будет в состоянии прочитать файл (г), но не сможет ни записать, ни выполнить его (--).
* последние три символа (—) относятся к любому пользователю, кроме queen и всех кто входит в группу users. Эти пользователи вообще не имеют никаких прав на этот файл.

Для каталога a\_directory права выглядят так rwxr-xr--, отсюда:

* peter, как владелец каталога, может получить список находящихся в нем файлов (г), добавить или удалить файлы из этого каталога (w) и может пройти через него (х):
* Каждый пользователь, кроме peter, который входит в группу users, будет в состоянии получить список файлов в этом каталоге (г), но не сможет удалить или добавить файлы (-), а также сможет проходить его (х).
* Любой другой пользователь сможет только получить список содержимого этого каталога (г). Но поскольку у него нет прав wx, он не сможет записать файлы или войти в каталог.

Есть одно исключение из этих правил - root, root может изменять атрибуты (права доступа, владельца и группу) всех файлов, даже если он не является владельцем, и поэтому сможет сделать себя владельцем файла! root может читать файлы, для которых у него нет прав на чтение, проходить через каталоги, к которым у него\* будь он обычным пользователем, не было бы доступа и т.д. И если root'y не хватает прав, ему нужно просто добавить их. root имеет полный контроль над системой, что влечет за собой определенный уровень доверия к человеку, знающего его пароль.

И в заключение, не стоит беспокоиться из-за различий между именами файлов в мирах UNIX® и Windows®. Первый - UNIX® - предоставляет значительно большую гибкость и имеет меньше ограничений.

* Имя файла может содержать любые символы, включая непечатаемые, за исключением ASCII-символа 0. который означает конец строки, и /, который является разделителем каталога. Кроме того, вследствие чувствительности к регистру в UNIX® файлы readme и Readme будут разными, потому что под буквами г и R в системах на базе UNIX® подразумеваются два разных символа.
* Имя файла не обязательно должно иметь расширение, если только вам не захочется так называть свои файлы. В GNU/Linux расширения файлов не определяют их содержимого, а также на большинстве операционных систем. Тем не менее, так называемые «расширения файлов» довольно удобны. В UNIX® точка (.) - это просто один из символов, но он также имеет одно специальное назначение. В UNIX® файлы с именами, начинающимися с точки, являются «скрытыми»; это также касается и каталогов, чьи имена начинаются с.

4.7. Кэширование диска

Кэш-память, или просто кэш (cache), — это способ совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ позволяет, с одной стороны, уменьшить среднее время доступа к данным, а с другой стороны, экономить более дорогую быстродействующую память.

Неотъемлемым свойством кэш-памяти является ее прозрачность для программ и пользователей. Система не требует никакой внешней информации об интенсивности использования данных: ни пользователи, ни программы не принимают никакого участия в перемещении данных из ЗУ одного типа в ЗУ другого типа, все это делается автоматически системными средствами.

Кэш-памятью, или кэшем, часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств — «быстрое» ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. «Медленное» ЗУ далее будем называть основной памятью, противопоставляя ее вспомогательной кэш-памяти.

Кэширование — это универсальный метод, пригодный для ускорения доступа к оперативной памяти, к диску и к другим видам запоминающих устройств. Если кэширование применяется для уменьшения среднего времени доступа к оперативной памяти, то в качестве кэша используют быстродействующую статическую память. Если кэширование используется системой ввода-вывода для ускорения доступа к данным, хранящимся на диске, то в этом случае роль кэш-памяти выполняют буферы в оперативной памяти, в которых оседают наиболее активно используемые данные. Виртуальную память также можно считать одним из вариантов реализации принципа кэширования данных, при котором оперативная память выступает в роли кэша по отношению к внешней памяти — жесткому диску. Правда, в этом случае кэширование используется не для того, чтобы уменьшить время доступа к данным, а для того, чтобы заставить диск частично подменить оперативную память за счет перемещения временно неиспользуемого кода и данных на диск с целью освобождения места для активных процессов. В результате наиболее интенсивно используемые данные «оседают» в оперативной памяти, остальная же информация хранится в более объемной и менее дорогостоящей внешней памяти.

В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (диски, ленты), перехватываются промежуточным программным слоем-подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

4.8. Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рисунок 4.6). в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам. в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Запрос к файлу (операция, имя файла, логическая запись)



К подсистеме веода-вывода

Рис. 4.6. Общая модель файловой системы

Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, MS-DOS), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла. В файловой системе UNIX, например, уникальным именем является номер индексного дескриптора файла (i-node).

На следующем, базовом уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых файловых системах (например, HPFS) при открытии файла вместе с его характеристиками в оперативную память перемещаются несколько первых блоков файла, содержащих данные.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байта4 ) от начала файла находится требуемая логическая запись. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины 1, то n- ая логическая запись имеет смещение 1(п~ байт. Для определения координат логической записи в файле с индексно - последо вательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.

файл

V V V v"\*4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | N |  |  |  |  |  |  |
|  | S | | Is» | логичес | | хая запись | |
|  | | |  | |

Рис. 4.7. Функции физического уровня файловой системы

Исходные данные:

* V - размер блока
* N - номер первого блока файла
* S - смешение логической записи в файле

Требуется определить на физическом уровне:

* п - номер блока, содержащего требуемую логическую запись
* s - смещение логической записи в пределах блока
* n = N + [S/V], где [S/V] - целая часть числа S/V
* s = R [S/V] - дробная часть числа S/V

На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня - смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока. Рисунок 4.7 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физическою уровня решается независимо оттого, как был логически организован файл.

После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

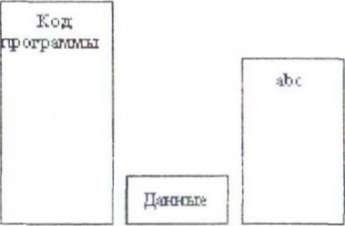
4.9. Отображаемые в память файлы

По сравнению с доступом к памяти, традиционный доступ к файлам выглядит запутанным и неудобным. По этой причине некоторые ОС, обеспечивают отображение файлов в адресное пространство выполняемого процесса. Этовыражается в появлении двух новых системных вызовов: MAP (отобразить) и UNMAP (отменить отображение). Первый вызов передает операционной системе в качестве параметров имя файла и виртуальный адрес, и операционная система отображает указанный файл в виртуальное адресное пространство по указанному адресу.

Предположим, например, что файл f имеет длину 64 К и отображается на область виртуального адресного пространства с начальным адресом 512 К. После этого любая машинная команда, которая читает содержимое байта по адресу 512 К. получает 0-ой байт этого файла и т.д. Очевидно, что запись по адресу 512 К + 1100 изменяет 1100 байт файла. При завершении процесса на диске остается модифицированная версия файла, как если бы он был изменен комбинацией вызовов SEEK и WRITE.

В действительности при отображении файла внутренние системные таблицы изменяются так, чтобы данный файл служил хранилищем страниц виртуальной памяти на диске. Таким образом, чтение по адресу 512 К вызывает страничный отказ, в результате чего страница 0 переносится в физическую память. Аналогично, запись по адресу 512 К + 1100 вызывает страничный отказ, в результате которого страница, содержащая этот адрес, перемещается в память, после чего осуществляется запись в память по требуемому адресу. Если эта страница вытесняется из памяти алгоритмом замены страниц, то она записывается обратно в файл в соответствующее его место. При завершении процесса все отображенные и модифицированные страницы переписываются из памяти в файл.

Отображение файлов лучше всего работает в системе, которая поддерживает сегментацию. В такой системе каждый файл может быть отображен в свой собственный сегмент, так что k-ый байт в файле является к-ым байтом сегмента. На рисунке 4.8, а изображен процесс, который имеет два сегмента-кода и данных. Предположим, что этот процесс копирует файлы. Для этого он сначала отображает файл-источник, например, abc. Затем он создает пустой сегмент и отображает на него файл назначения, например, файл ddd.



ddd

(5)

С этого момента процесс может копировать сегмент-источник в сегмент-приемник с помощью обычного программного цикла, использующего команды пересылки в памяти типа шоу. Никакие вызовы READ или WRITE не нужны. После выполнения копирования процесс может выполнить вызов UNMAP для удаления файла из адресного пространства, а затем завершиться. Выходной файл ddd будет существовать на диске, как если бы он был создан обычным способом.

Хотя отображение файлов исключает потребность в выполнении ввода-вывода и тем самым облегчает программирование, этот способ порождает и некоторые новые проблемы. Во-первых, для системы сложно узнать точную длину выходного файла, в данном примере ddd. Проще указать наибольший номер записанной страницы, но нет способа узнать, сколько байт в этой странице было записано. Предположим, что программа использует только страницу номер 0, и после выполнения все байты все еще установлены в значение 0 (их начальное значение). Быть может, файл состоит из 10 нулей. А может быть, он состоит из 100 нулей. Как это определить? Операционная система не может это сообщить. Все, что она может сделать, так это создать файл, длина которого равна размеру страницы.

Код программы

дшньк

СЮ

Рис. 4.Р (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) Процесс после отображения существующего файла abc в один сегмент и создания нового

сегмента для файла ddd

Вторая проблема проявляется (потенциально), если один процесс отображает файл, а другой процесс открывает его для обычного файлового доступа. Если первый процесс изменяет страницу, то это изменение не будет отражено в файле на диске до тех пор, пока страница не будет вытеснена на диск. Поддержание согласованности данных файла для этих двух процессов требует от системы больших забот.

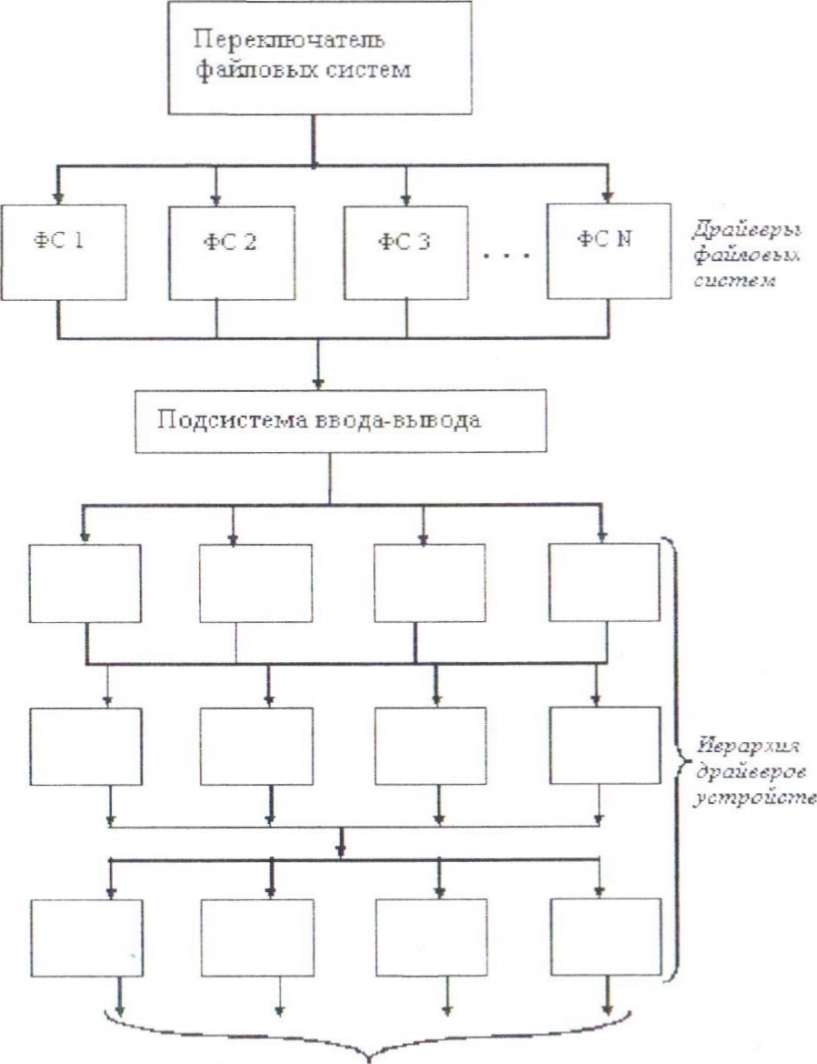
Третья проблема состоит в том, что файл может быть больше, чем сегмент, и лаже больше, чем все виртуальное адресное пространство. Единственный способ ее решения

состоит в реализации вызова MAP таким образом, чтобы он мог отображать не весь файл, а его часть. Хотя такая работа, очевидно, менее удобна, чем отображение целого файла.

4.10. Современные архитектуры файловых систем

Разработчики операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В современном понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Современная файловая система имеет многоуровневую структуру (рисунок 4.9), на верхнем уровне которой располагается гак называемый переключатель файловых систем (в Windows 95, например, га кой переключатель называется устанавливаемым диспетчером файловой системы - installable filesystem manager, IFS). Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой обращается это приложение. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем - уровнем файловых систем.



К -аппаратный средствам

Рис. 4.9. Архитектура современной файловой системы

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную организацию файловой системы.

Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую.

Драйвер файловой системы может быть написан в виде реентерабельного кода, что позволяет сразу нескольким приложениям выполнять операции с файлами. Каждый драйвер файловой системы в процессе собственной инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях к файловой системе.

Для выполнения своих функций драйверы файловых систем обращаются к подсистеме ввода-вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода вывода - это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы. Обычно эти модули представляют собой драйверы портов, которые непосредственно занимаются работой с аппаратными средствами. Кроме этого подсистема ввода-вывода обеспечивает некоторый сервис драйверам файловой системы, что позволяет им осуществлять запросы к конкретным устройствам.

Подсистема ввода-вывода должна постоянно присутствовать в памяти и организовывать совместную работу иерархии драйверов устройств. В эту иерархию могут входить драйверы устройств определенного типа (драйверы жестких дисков или накопителей на лентах), драйверы, поддерживаемые поставщиками (такие драйверы перехватывают запросы к блочным устройствам и могут частично изменить поведение существующего драйвера этого устройства, например, зашифровать данные), драйверы портов, которые управляют конкретными адаптерами.

Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость - драйвер может получить управление на любом этапе выполнения запроса - от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством цепочек вызова.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в цепочку вызова устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса, подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше - в измененном или в неизмененном виде - на следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

Приложение 1. Файловая система FAT

FAT

Файловая система FAT (File Allocation Table) была разработана Биллом Гейтсом и Марком МакДональдом в 1977 году и первоначально использовалась в операционной системе 86- DOS. Чтобы добиться переносимости программ из операционной системы СР/М в 86-DOS. в ней были сохранены ранее принятые ограничения на имена файлов. В дальнейшем 86-DOS была приобретена Microsoft и стала основой для ОС MS-DOS 1.0. выпущенной в августе 1981 года. FAT была предназначена для работы с гибкими дисками размером менее 1 Мбайта, и вначале не предусматривала поддержки жестких дисков. В настоящее время FAT поддерживает файлы и разделы размеров до 2 Гбайт.

В FAT применяются следующие соглашения по именам файлов:

* имя должно начинаться с буквы или цифры и может содержать любой символ ASCII, за исключением пробела и символов "А[]:;|=,А\*?
* Длина имени не превышает 8 символов, за ним следует точка и необязательное расширение длиной до 3 символов.
* регистр символов в именах файлов не различается и не сохраняется.

Структура раздела FAT изображена на рисунке 1. В блоке параметров BIOS содержится необходимая BIOS информация о физических характеристиках жесткого диска. Файловая система FAT не может контролировать отдельно каждый сектор, поэтому она объединяет смежные сектора в кластеры (clusters). Таким образом, уменьшается общее количество единиц хранения, за которыми должна следить файловая система. Размер кластера в FAT является степенью двойки и определяется размером тома при форматировании диска (табл. 1.1). Кластер представляет собой минимальное пространство, которое может занимать файл. Это приводит к тому, что часть пространства диска расходуется впустую. В состав операционной системы входят различные утилиты (DoubleSpace, DriveSpace), предназначенные для уплотнения данных на диске.

Загрузочный сектор FAT Корневой Область

Блок параметров BIOS (ВРВ) FAT (копия) каталог файлов

Рисунок 1.

Свое название FAT получила от одноименной таблицы размещения файлов. В таблице размещения файлов хранится информация о кластерах логического диска. Каждому кластеру в FAT соответствует отдельная запись, которая показывает, свободен ли он. занят ли данными файла, или помечен как сбойный (испорченный). Если кластер занят под файл, то в соответствующей записи в таблице размещения файлов указывается адрес кластера, содержащего следующую часть файла. Из-за этого FAT называют файловой системой со связанными списками. Оригинальная версия FAT, разработанная для DOS 1.00. использовала 12-битную таблицу размещения файлов и поддерживала разделы объемом до 16 Мб (в DOS можно создать не более двух разделов FAT). Для поддержки жестких дисков размером более 32 Мб разрядность FAT была увеличена до 16 бит. а размер кластера - до 64 секторов (32 Кб). Так как каждому кластеру может быть присвоен уникальный 16-разрядный номер, то FAT поддерживает максимально 2\*6, или 65536 кластеров на одном томе.

|  |  |  |
| --- | --- | --- |
| Размер раздела | Размер кластера | Тин FAT |
| < 16 Мб | 4 Кб | FAT 12 |
| 16 Мб- 127 Мб | 2 Кб | FAT 16 |
| 128 Мб-255 Мб | 4 Кб | FAT 16 |
| 256 Мб-511 Мб | 8 Кб | FAT 16 |
| 512 Мб-1023 Мб | 16 Кб | FAT 16 |
| 1 Г6-2Г6 | 32 Кб | FAT 16 |

Таблица 1.

Поскольку загрузочная запись слишком мала для хранения алгоритма поиска системных файлов на диске, то системные файлы должны находиться в определенном месте, чтобы загрузочная запись могла их найти. Фиксированное положение системных файлов в начале области данных накладывает жесткое ограничение на размеры корневого каталога и таблицы размещения файлов. Вследствие этого общее число файлов и подкаталогов в корневом каталоге на диске FAT ограничено 512.

Каждому файлу и подкаталогу в FAT соответствует 32-байтный элемент каталога (directory entry), содержащий имя файла, его атрибуты (архивный, скрытый, системный и "только для чтения"), дату и время создания (или внесения в него последних изменений), а также прочую информацию (табл. 2).

|  |  |
| --- | --- |
| Содержание | Размер(байт) |
| Имя файла | 8 |
| Расширение | 3 |
| Байт атрибутов | 1 |
| Зарезервировано | 10 |
| Время | 2 |
| Дата | 2 |
| Номер начального кластера с | 2 |
| данными |  |
| Размер файла | 4 |

Табл. 2 Элемент каталога

Файловая система FAT всегда заполняет свободное место на диске последовательно от начала к концу. При создании нового файла или увеличении уже существующего она ищет самый первый свободный кластер в таблице размещения файлов. Если в процессе работы одни файлы были удалены, а другие изменились в размере, то появляющиеся в результате пустые кластеры будут рассеяны по диску. Если кластеры, содержащие данные файла, расположены не подряд, то файл оказывается фрагментированным. Сильно фрагментированные файлы значительно снижают эффективность работы, так как головки чтения/записи при поиске очередной записи файла должны будут перемещаться от одной области диска к другой. В состав операционных систем, поддерживающих FAT, обычно входят специальные утилиты дефрагментации диска, предназначенные повысить производительность файловых операций.

Еще один недостаток FAT заключается в том, что ее производительность сильно зависит от количества файлов, хранящихся в одном каталоге. При большом количестве файлов (около тысячи), выполнение операции считывания списка файлов в каталоге может занять несколько минут. Это обусловлено тем, что в FAT каталог имеет линейную неупорядоченную структуру, и имена файлов в каталогах идут в порядке их создания. В результате, чем больше в каталоге записей, тем медленнее работают программы, так как при поиске файла требуется просмотреть последовательно все записи в каталоге.

Поскольку FAT изначально проектировалась для однопользовательской операционной системы DOS, то она не предусматривает хранения такой информации, как сведения о владельце или полномочия доступа к файлу/каталогу.

FAT является наиболее распространенной файловой системой и ее в той или иной степени поддерживают большинство современных ОС. Благодаря своей универсальности FAT может применяться на томах, с которыми работают разные операционные системы.

Хотя нет никаких препятствий использовать при форматировании дискет любую другую файловую систему, большинство ОС для совместимости используют FAT. Отчасти это можно объяснить тем, что простая структура FAT требует меньше места для хранения служебных данных, чем остальные системы. Преимущества других файловых систем становятся заметны только при использовании их на носителях объемом более 100 Мб.

Надо отметить, что FAT - простая файловая система, не предотвращающая порчи файлов из- за ненормального завершения работы компьютера. В состав операционных систем, поддерживающих FAT, входят специальные утилиты проверяющие структуру и корректирующие несоответствия в файловой системе.

Приложение 2. Файловая система FAT32 FAT 32

FAT32 - файловая система, производная системы FAT. FAT32 поддерживает меньшие размеры кластеров, что позволяет более эффективно использовать дисковое пространство по сравнению с FAT. Это файловая система используется в современных операционных системах на DOS основе - Windows 98 и Windows Me.

FAT - File Allocation Table (таблица размещения файлов) - этот термин относится к одному из способов организации файловой системы на диске. Эта таблица хранит информацию о файлах на жестком диске в виде последовательности чисел, определяющих, где находится каждая часть каждого файла. С ее помощью операционная система выясняет, какие кластеры занимает нужный файл. FAT - является самой распространенной файловой системой и поддерживается подавляющим большинством операционных систем. Сначала FAT была 12- разрядной и позволяла работать с дискетами и логическими дисками объемом не более 16 Мбайт. В MS-DOS версии 3.0 таблица FAT стала 16-разрядной для поддержки дисков большей емкости, а для дисков объемом до 2 047 Гбайт используется 32-разрядная таблица FAT.

Система FAT32 - более новая файловая система на основе формата FAT, она поддерживается Windows 95 OSR2. Windows 98 и Windows Millennium Edition. FAT32 использует 32-разрядные идентификаторы кластеров, но при этом резервирует старшие 4 бита, так что эффективный размер идентификатора кластера составляет 28 бит. Поскольку максимальный размер кластеров FAT32 равен 32 Кбайт, теоретически FAT32 может работать с 8-терабайтными томами. Windows 2000 ограничивает размер новых томов FAT32 до 32 Гбайт, хотя поддерживает существующие тома FAT32 большего размера (созданные в других операционных системах). Большее число кластеров, поддерживаемое FAT32, позволяет ей управлять дисками более эффективно, чем FAT 16. FAT32 может использовать 512-байтовые кластеры для томов размером до 128 Мбайт.

Файловая система FAT 32 в Windows 98 используется в качестве основной. С этой операционной системой поставляется специальная программа преобразования диска из FAT 16 в FAT 32. Windows 2000 и Windows ХР тоже могут использовать файловую систему FAT, 1-1 поэтому можно загрузить компьютер с DOS-диска и иметь полный доступ ко всем файлам. Однако некоторые из самых прогрессивных возможностей Windows 2000 и Windows ХР обеспечиваются ее собственной файловой системой NTFS (NT File System). NTFS позволяет создавать на диске разделы объемом до 2 Тбайт (как и FAT 32), но. кроме этого, в нее встроены функции сжатия файлов, безопасности и аудита, необходимые при работе в сетевой среде. А в Windows 2000, как и в Windows ХР реализуется поддержка файловой системы FAT 32. Данные этих операционных систем можно хранить на диске FAT, но по желанию пользователя диск может быть конвертирован в формат NTFS.

Для этого можно воспользоваться утилитой Convert.exe, поставляемой вместе с операционной системой. Преобразованный к системе NTFS раздел диска становится недоступным для других операционных систем. Чтобы вернуться в DOS, Windows 95, Windows 98 или Me, нужно удалить раздел NTFS, а вместо него создать раздел FAT. Windows 2000, как и в Windows ХР можно устанавливать на диск с файловой системой FAT 32 и NTFS.

Возможности файловых систем FAT32 гораздо шире возможностей FAT 16. Самая важная ее особенность в том, что она поддерживает диски объемом до 2 047 Гбайт и работает с кластерами меньшего размера, благодаря чему существенно сокращает объемы неиспользуемого дискового пространства. Например, жесткий диск объемом 2 Гбайт в FAT 16 использует кластеры размером по 32 Кбайт, а в FAT32 - кластеры размером по 4 Кбайт. Чтобы по возможности сохранить совместимость с существующими программами, сетями и драйверами устройств, FAT32 реализована с минимальными изменениями в архитектуре. API-интерфейсах, структурах внутренних данных и дисковом формате. Но, так как размер элементов таблицы FAT32 теперь составляет четыре байта, многие внутренние и дисковые структуры данных, а также API-интерфейсы пришлось пересмотреть или расширить. Отдельные API на НАТ32-дисках блокируются, чтобы унаследованные дисковые утилиты не повредили содержимое РАТ32-дисков. На большинстве программ эти изменения никак не скажутся. Существующие инструментальные средства и драйверы будут работать и на 1-'АТ32-дисках. Однако драйверы блочных устройств MS-DOS (например. Aspidisk.sys) и дисковые утилиты нуждаются в модификации для поддержки FAT32. Все дисковые утилиты, поставляемые Microsoft (Format, Fdisk, Defrag, а также ScanDisk для реального и защищенного режимов), переработаны и полностью поддерживают FAT32. Кроме того, Microsoft помогает ведущим поставщикам дисковых утилит и драйверов устройств в модификации их продуктов для поддержки FAT32. F/^T32 эффективнее FAT 16 при работе с дисками большего объема и не требует их разбиения на разделы по 2 Гбайт. Windows 98 обязательно поддерживает FAT 16, так как именно эта файловая система совместима с другими операционными системами, в том числе сторонних компании. В MS-DOS реального режима и в безопасном режиме Windows 98, файловая система FAT32 работает значительно медленнее, чем FAT16. Поэтому, при запуске программ в режиме MS DOS желательно включить в файл Autoexec.bat или PlF-файл команду для загрузки Smartdrv.exe, что ускорит дисковые операции. Некоторые устаревшие программы, рассчитанные на спецификацию FAT 16. могут сообщать неправильную информацию об объеме свободного или общего дискового пространства, если он больше 2 Гбайт. Windows 98 предоставляет новые API- интерфейсы для MS-DOS и Win32, которые позволяют корректно определять эти показатели. В табл. I приведены сравнительные характеристики FAT 16 и FAT32.

г

Таблица 1. Сравнение файловых систем FAT16 и FAT32

|  |  |
| --- | --- |
| FAT 16 | FAT32 |
| Реализована и используется большинством операционных систем (MS-DOS, Windows 95/98/Ме. Windows 2000 и Windows ХР, OS/2, UNIX). | На данный момент поддерживается только в Windows 95/98/Ме. Windows 2000 и Windows ХР. |
| Очень эффективна для логических дисков размером менее 256 Мбайт. | Не работает с дисками объемом менее 512 Мбайт. |
| Поддерживает сжатие дисков, например по алгоритму DriveSpace. | Не поддерживает сжатие дисков. |
| Обрабатывает максимум 65 525 кластеров, размер которых зависит от объема логического диска. Так как максимальный размер кластеров равен 32 Кбайт, FAT16 может работать с логическими дисками объемом не более 2 Гбайт. | Способна работать с логическими дисками объемом до 2 047 Гбайт при максимальном размере кластеров в 32 Кбайт. |
| Чем больше размер логическою диска, тем меньше эффективность хранения файлов в FAT'16-системе, так как увеличивается и размер кластеров. Пространство для файлов выделяется кластерами, и поэтому при максимальном объеме логического диска файл размером 10 Кбайт потребует 32 Кбайт, а 22 Кбайт дискового пространства пропадет впустую. | На логических дисках объемом менее 8 Гбайт размер кластеров составляет 4 Кбайт. |

Максимально возможная длина файла в FAT32 равна 4 Гбайт за вычетом 2 байтов. Win32- приложения могут открывать файлы такой длины без специальное обработки. Остальные приложения должны использовать прерывание Int 21h, функцию 716С (FAT32) с флагом открытия, равным EXTEND-SIZE (ЮООИ).

В файловой системе FAT32 на каждый кластер в таблице размещения файлов отводится по 4 байта, тогда как в FAT 16 - по 2, а в FAT12 - по 1,5.

Старшие 4 бита 32-разрядного элемента таблицы FAT32 зарезервированы и не участвуют в формировании номера кластера. Программы, напрямую считывающие РАТ32-таблицу, должны маскировать эти биты и предохранять их от изменения при записи новых значений.

Итак, FAT32 обладает следующими преимуществами в сравнении с прежними реализациями файловой системы FAT:

* поддерживает диски объемом до 2 Гбайт;
* эффективнее организует дисковое пространство. FAT32 использует кластеры меньшего размера (4 Кбайт для дисков объемом до 8 Гбайт), что позволяет сэкономить до 10-15% пространства на больших дисках по сравнению с FAT;
* корневой каталог FAT 32, как и все остальные каталоги, теперь не ограничен, он состоит из цепочки кластеров и может быть расположен в любом месте диска;
* имеет более высокую надежность: FAT32 способна перемещать корневой каталог и работать с резервной копией FAT, кроме того, загрузочная запись на FАТЗ2-дисках расширена и теперь включает резервную копию критически важных структур данных, а это означает, что РАТ32-диски менее чувствительны к возникновению отдельных сбойных участков, чем существующие FAT-тома:
* программы загружаются на 50% быстрее.

Таблица 2. Сравнение размеров кластеров

|  |  |  |
| --- | --- | --- |
| Объем диска | Размер кластеров в FAT16, Кбайт | Размер кластеров в FAT32, Кбайт |
| 256 Мбайт-511 Мбайт | 8 | Не поддерживается |
| 512 Мбайт-1023 Мбайт | 16 | 4 |
| 1024 Мбайт - 2 Гбайт | 32 | 4 |
| 2 Гбайт - 8 Гбайт | Не поддерживается | 4 |
| 8 Гбайт-16 Гбайт | Не поддерживается | 8 |
| 16 Гбайт-32 Гбайт | Не поддерживается | 16 |
| Более 32 Гбайт | Не поддерживается | 32 |

Усовершенствованная утилита дефрагментации дисков оптимизирует размещение файлов приложения, загружаемых в момент его запуска. Возможно преобразование диска в FAT32 с помощью утилиты Drive Converter (FAT32), но после этого рекомендуется запустить утилиту Disk Defragmenter, - иначе компьютер будет работать с диском медленнее, чем раньше. Благодаря этому на больших дисках удается высвободить десятки и даже сотни мегабайтов, а в сочетании с усовершенствованной утилитой дефрагментации дисков FAT32 значительно сокращает время загрузки приложений. Процедура преобразования файловой системы на жестком диске в FAT32 с помощью Drive Converter (FAT32) достаточно проста. Для этого последовательно необходимо открыть меню Start (Пуск), подменю Programs (Программы), Accessories (Стандартные), System Tools (Служебные) и выбрать команду Drive Converter (FAT32) (Преобразование диска в FAT32). Преобразование может повлиять на функции спящего режима (hibernate features) (сохранения состояния компьютера на диск), предусмотренные во многих компьютерах. Системы, в которых режим сна реализован через АРМ BIOS или ACPI (Advanced Configuration and Power Interface) S4/BIOS, должны поддерживать FAT32, - только тогда они будут корректно работать в Windows 98 и Me.

Большинство изготовителей BIOS включают в нее средства зашиты от вирусов, отслеживающие изменения в главной загрузочной записи MBR (Master Boot Record). Кроме того, устаревшие антивирусные утилиты, устанавливаемые как резидентные программы или

драйверы реального режима, могут обнаруживать изменение MBR при загрузке MS-DOS. Так как преобразование в FAT32 приводит к неизбежной модификации MBR, некоторые средства проверки на вирусы могут ошибочно счесть это признаком инфицирования

системы.

Лучше всего удалить антивирусное программное обеспечение и отключить встроенные в BIOS средства защиты от вирусов перед преобразованием диска в FAT32. Потом можно вновь установить антивирусную утилиту и активизировать встроенные в BIOS средства защиты от вирусов.

Главная загрузочная запись (MBR)

Форматирование жестких дисков выполняется в три этапа:

* низкоуровневое форматирование (физическая разметка диска на цилиндры, дорожки, секторы);
* разбиение диска на разделы (логические устройства):
* высокоуровневое (логическое) форматирование каждого раздела.

На этапе низкоуровневого форматирования процессор, выполняя программу форматирования, поочередно передает в контроллер жесткого диска сначала команду "Поиск" для установки головок накопителя на нужный цилиндр, а затем посылает команду "Форматировать дорожку". Выполняя команду "Форматировать дорожку" контроллер жесткого диска, получив из накопителя импульс "Индекс" (начало дорожки), производит запись служебного формата дорожки, который разбивает ее на секторы. Каждый сектор содержит в себе блок данных (512 байт), обрамленный служебным форматом сектора (содержание к размер служебного формата определяется конкретной фирмой-разработчиком данного устройства).

Служебный формат дорожки и секторов необходим контроллеру жесткого диска при выполнении команд. Читая и расшифровывая поля служебного формата, контроллер находит на диске нужный цилиндр, "поверхность, сектор и блок данных внутри сектора. На следующих этапах форматирования в блоки данных ряда секторов записывается системная информация, которая обеспечивает организацию разделов на диске, автоматическую загрузку операционной системы и поддержку файловой системы на диске.

На э тапе разбиения диска на разделы в блоке данных первого физического сектора диска (О цилиндр, 0 поверхность, 1 сектор) с адреса IBEh формируется таблица разделов (Partition table), состоящая из 4-х шестнадцати байтных строк. Обычно системную информацию, записанную в блок данных этого сектора в процессе форматирования, называют Master Boot Record (MBR).

С самого начала блока данных этого сектора располагается программа (IPL 1). Переход на программу IPL 1 процессор осуществляет после успешного завершения POST и программы "Начального загрузчика", выполняя которую процессор загружает с диска в память MBR. и передает управление на начало MBR (на программу IPL 1). продолжая действия ведущие к загрузке операционной системы. Программа IPL 1 (загрузчик), находящаяся в MBR просматривает строки таблицы разделов в поисках активного раздела с которого возможна загрузка операционной системы. Если в таблице разделов нет активного раздела, выдается сообщение об ошибке. Если хотя бы один раздел содержит неправильную метку, либо несколько разделов помечены как активные, выдается сообщение об ошибке Invalid partition table, и процесс загрузки останавливается. Если активный раздел обнаружен, то анализируется загрузочный сектор этого раздела. Если найден только один активный раздел, то содержимое блока данных его загрузочного сектора (BOOT) читается в память по адресу Ю00:7С00 и управление передается по этому адресу, если загрузочный сектор активного раздела не читается за пять попыток, выдается сообщение об ошибке: Error loading operating system и система останавливается: проверяется сигнатура считанного загрузочного сектора активного раздела и если последних два его байта не соответствуют сигнатуре 55ААИ, выдается сообщение об ошибке: Missing operating system и система останавливается).

Процессор читает по адресу 0000:7000 команду J MP. выполняя ее, передает управление на начат' программы IPL 2, которая осушествляет проверку, действительно ли раздел активный: 1PL 2 проверяет имена и расширения двух файлов в корневом каталоге - это должны быть файлы IO.SYS и MSDOS.SYS (NTLDR для Windows ХР), загружает их и. т. д.

Система Windows 9х/Ме во многом основана на тех же концепциях, что и DOS, но в ней эти концепции получили дальнейшее логическое развитие. Те же два системных файла IO.SYS и MSDOS.SYS, но теперь вся системная программа находится в IO.SYS , а второй файл MSDOS.SYS содержит ASCII-текст с установками, управляющими поведением системы при загрузке. Эквиваленты программ Himem.sys. Ifshlp.sys и Setver.exe автоматически загружаются программой IO.SYS при запуске системы. Как и прежде, для загрузки в память драйверов и резидентных программ можно использовать файлы Config.sys и Autoexec.bat, но загрузку 32-разрядных драйверов устройств, которые разработаны специально для Windows 9х. теперь обеспечивают записи в системном реестре. Когда вся предварительная работа выполнена, запускается файл Win.com, и Windows 9х/Ме загружается и предоставляет свои возможности через графическое меню.

Системный реестр является базой данных, в которой Windows 9х/Ме хранит информацию обо всех настройках, конфигурационных установках и параметрах, необходимых для работы ее собственных модулей и отдельных приложений. Системный реестр как бы выполняет функции Config.sys, Autoexec.bat и ini-файлов Windows 3.1 вместе взятых. На диске компьютера реестр хранится в виде двух отдельных файлов: System.dat и User.dat. В первом из них содержатся всевозможные аппаратные установки, а во втором - данные о работающих в системе пользователях и используемых ими конфигурациях. Каждый пользователь может иметь свой файл User.dat, т.е. собственную рабочую среду, которую он настраивает по своему вкусу и потребностям. Системный реестр можно импортировать, экспортировать, а также создавать его резервные копии и. используя их. восстанавливать сохраненные данные - одним словом, это довольно мощный механизм управления системными параметрами и их защиты от потерь и повреждений.

В данный момент FAT32 поддерживается в следующих ОС: Windows 95 OSR2. Windows 98, Windows ME, Windows 2000 и Windows ХР.

Приложение 3. Файловая система NTFS

NTFS

NTFS (New Technology File System) - наиболее предпочтительная файловая система при работе с ОС Windows NT (Windows 2000 и ХР также являются NT системами), поскольку она была специально разработана для данной системы. В состав Windows NT входит утилита convert, осуществляющая конвертирование томов с FAT и HPFS в тома NTFS. В NTFS значительно расширены возможности по управлению доступом к отдельным файлам и каталогам, введено большое число атрибутов, реализована отказоустойчивость, средства динамического сжатия файлов, поддержка требований стандарта POSIX. NTFS позволяет использовать имена файлов длиной до 255 символов, при этом она использует тот же алгоритм для генерации короткого имени, что и VFAT. NTFS обладает возможностью самостоятельного восстановления в случае сбоя ОС или оборудования, так что дисковый том остается доступным, а структура каталогов не нарушается.

Каждый файл на томе NTFS представлен записью в специальном файле - главной файловой таблице MFT (Master File Table). NTFS резервирует первые 16 записей таблицы размером около 1 Мб для специальной информации. Первая запись таблицы описывает непосредственно саму главную файловую таблицу. За ней следует зеркальная запись MFT. Если первая запись MFT разрушена, NTFS считывает вторую запись, чтобы отыскать зеркальный файл MFT, первая запись которого идентична первой записи MFT. Местоположение сегментов данных MFT и зеркального файла MFT хранится в секторе начальной загрузки. Копия сектора начальной загрузки находится в логическом центре диска. Третья запись MFT содержит файл регистрации, применяемый для восстановления файлов. Семнадцатая и последующие записи главной файловой таблицы используются собственно файлами и каталогами на томе.

В журнале транзакций (log file) регистрируются все операции, влияющие на структуру тома, включая создание файла и любые команды, изменяющие структуру каталогов. Журнал транзакций применяется для восстановления тома NTFS после сбоя системы. Запись для корневого каталога содержит список файлов и каталогов, хранящихся в корневом каталоге.

Схема распределения пространства на томе хранится в файле битовой карты (bitmap file). Атрибут данных этого файла содержит битовую карту, каждый бит которой представляет один кластер тома и указывает, свободен ли данный кластер или занят некоторым файлом.

В загрузочном файле (boot file) хранится код начального загрузчика Windows NT.

NTFS также поддерживает файл плохих кластеров (bad cluster tile) для регистрации поврежденных участков на томе и файл тома (volume file), содержащий имя тома, версию NTFS и бит, который устанавливается при повреждении тома. Наконец, имеется файл, содержащий таблицу определения атрибутов (attribute definition table), которая задает типы атрибутов, поддерживаемые на томе, и указывает можно ли их индексировать, восстанавливать операцией восстановления системы и т.д.

NTFS распределяет пространство кластерами и использует для их нумерации 64 разряда, что дает возможность иметь кластеров, каждый размером до 64 Кбайт. Как и в FAT размер кластера может меняться, но необязательно возрастает пропорционально размеру диска. Размеры кластеров, устанавливаемые по умолчанию при форматировании раздела, приведены в табл. 6.

|  |  |
| --- | --- |
| Размер раздела | Размер кластера |
| <512 Мб | 512 байт |
| 513 Мб - 1024„Мб (1 Гб) | 1 Кб |
| 1 Гб - 2 Гб | 2 Кб |
| 2 Гб - 4 Гб | 4 Кб |
| 4 Гб - 8 Гб | 8 Кб |
| 8 Гб - 16 Гб | 16 Кб |
| 16 Гб-32 Гб | 32 Кб |
| >32 Гб | 64 Кб |

Табл. 6

NTFS позволяет хранить файлы размером до 16 эксабайт (2"^ байт) и располагает встроенным средством уплотнения файлов в реальном времени. Сжатие является одним из атрибутов файла или каталога и подобно любому атрибуту может быть снято или установлено в любой момент (сжатие возможно на разделах с размером кластера не более 4 Кб). При уплотнении файла, в отличие от схем уплотнения используемых в FAT, применяется пофайловое уплотнение, таким образом, порча небольшого участка диска не приводит к потере информации в других файлах.

Для уменьшения фрагментации NTFS всегда пытается сохранить файлы в непрерывных блоках. Эта система использует структуру каталогов в виде В-дерева, аналогичную высокопроизводительной файловой системе HPFS, а не структуре со связанным списком применяемой в FAT. Благодаря этому поиск файлов в каталоге осуществляется быстрее, поскольку имена файлов хранятся сортированными в лексикографическом порядке.

NTFS была разработана как восстанавливаемая файловая система, использующая модель обработки транзакций. Каждая операция ввода-вывода, изменяющая файл на томе NTFS, рассматривается системой как транзакция и может выполняться как неделимый блок. При модификации файла пользователем сервис файла регистрации фиксирует всю информацию необходимую для повторения или отката транзакции. Если транзакция завершена успешно, производится модификация файла. Если нет. NTFS производит откат транзакции.

Несмотря на наличие защиты от несанкционированного доступа к данным NTFS не обеспечивает необходимую конфиденциальность хранимой информации. Для г :> лучения доступа к файлам достаточно загрузить компьютер в DOS с дискеты и воспользоваться каким-нибудь сторонним драйвером NTFS для этой системы.

Начиная с версии Windows 2000 Microsoft поддерживает новую файловую систему NTFS 5.0. В новой версии NTFS были введены дополнительные атрибуты файлов; наряду с правом доступа введено понятие запрета доступа, позволяющее, например, при наследовании пользователем прав группы на какой-нибудь файл, запретить ему возможность изменять его содержимое. Новая система также позволяет:

* вводить ограничения (квоты) на размер дискового пространства, предоставленного пользователям;
* проецировать любой каталог (как на локальном, так и на удаленном компьютере) в подкаталог на локальном диске.

Интересной возможностью новой версии Windows NT является динамическое шифрование файлов и каталогов, повышающее надежность хранения информации. В состав Windows 2000 и Windows ХР входит файловая система с шифрованием (Encrypting File System, EFS), использующая алгоритмы шифрования с общим ключом. Если для файла установлен атрибут шифрования, то при обращении пользовательской программы к файлу для записи или чтения происходит прозрачное для программы кодирование и декодирование файла.

Структура раздела - общий взгляд

Как и любая другая система. NTFS делит все полезное место на кластеры - блоки данных, используемые единовременно. NTFS поддерживает почти любые размеры кластеров - от 512 байт до 64 Кбайт, неким стандартом же считается кластер размером 4 Кбайт. Никаких аномалий кластерной структуры NTFS не имеет, поэтому на эту, в общем-то, довольно банальную тему, сказать особо нечего.

I

Диск NTFS условно делится на две части. Первые 12% диска отводятся под так называемую MFT зону - пространство, в которое растет метафайл MFT (об этом ниже). Запись каких- либо данных в эту область невозможна. MFT-зона всегда держится пустой - это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 88% диска представляют собой обычное пространство для хранения файлов.

—— MFT зона (сюда теоретически, растет MFT)

|  |  |  |
| --- | --- | --- |
|  | место под файлы | место под файлы |
| t 1 | | к |

— MFT Копия первых записей MFT

Свободное место диска, однако, включает в себя всё физически свободное место - незаполненные куски MFT-зоны туда тоже включаются. Механизм использования MFT- зоны таков: когда файлы уже нельзя записывать в обычное пространство, MFT-зона просто сокращается (в текущих версиях операционных систем ровно в два раза), освобождая таким образом место для записи файлов. При освобождении места в обычной области MFT зона может снова расширится. При этом не исключена ситуация, когда в этой зоне остались и обычные файлы: никакой аномалии тут нет. Что ж, система старалась оставить её свободной, но ничего не получилось. Жизнь продолжается... Метафайл MFT все-таки может фрагментироваться, хоть это и было бы нежелательно.

MFT и его структура

Файловая 'система NTFS представляет собой выдающееся достижение структуризации: каждый элемент системы представляет собой файл - даже служебная информация. Самый главный файл на NTFS называется MFT, или Master File Table - общая таблица файлов. Именно он размещается в MFT зоне и представляет собой централизованный каталог всех остальных файлов диска, и, как не парадоксально, себя самого. MFT поделен на записи фиксированного размера (обычно 1 Кбайт), и каждая запись соответствует какому либо файлу (в общем смысле этого слова). Первые 16 файлов носят служебный характер и недоступны операционной системе - они называются метафайлами, причем самый первый метафайл - сам MFT. Эти первые 16 элементов MFT - единственная часть диска, имеющая фиксированное положение. Интересно, что вторая копия первых трех записей, для надежности - они очень важны - хранится ровно посередине диска. Остальной MFT-файл может располагаться, как и любой другой файл, в произвольных местах диска - восстановить его положение можно с помощью его самого, "зацепившись" за самую основу - за первый элемент MFT.

Журнал про вам не

NTFS - отказоустойчивая система, которая вполне может привести себя в корректное состояние при практически любых сбоях. Любая современная файловая система основана на таком понятии, как транзакция - действие, совершаемое целиком и корректно или не совершаемое вообще. У NTFS просто не бывает промежуточных (ошибочных или некорректных) состояний - квант изменения данных не может быть поделен на до и после сбоя, принося разрушения и путаницу - он либо совершен, либо отменен.

Пример 1: осуществляется запись данных на диск. Вдруг выясняется, что в то место, куда мы только что решили записать очередную порцию данных, писать не удалось - физическое повреждение поверхности. Поведение NTFS в этом случае довольно логично: транзакция записи откатывается целиком - запись не производится. Место помечается как сбойное, а данные записываются в другое место - начинается новая транзакция.

Пример 2: более сложный случай - идет запись данных на диск. Вдруг, бах - отключается питание и система перезагружается. На какой фазе остановилась запись, где есть данные, а где данные не корректны? На помощь приходит другой механизм системы - журнал транзакций. Система отмечает в метафайле SLogFile свое состояние и при перезагрузке это файл проверяется на предмет наличия незавершенных транзакций, которые были прерваны из-за аварии и результат, которых непредсказуем. Все эти транзакции отменяются: место, в которое осуществлялась запись, помечается снова как свободное, индексы и элементы MFT приводятся в с состояние, в котором они были до сбоя, и система в целом остается стабильна. Если произошла ошибка при записи в журнал - ничего страшного: транзакция либо еще и не начиналась (идет только попытка записать намерения ее произвести), либо уже закончилась - то есть идет попытка записать, что транзакция на самом деле уже выполнена. В последнем случае при следующей загрузке система сама вполне разберется, что на самом деле всё и так записано корректно, и не обратит внимания на "незаконченную" транзакцию.

И все-таки помните, что журналирование - не абсолютная панацея, а лишь средство существенно сократить число ошибок и сбоев системы. Вряд ли рядовой пользователь NTFS хоть когда-нибудь заметит ошибку системы или вынужден будет запускать chkdsk - опыт показывает, что NTFS восстанавливается в полностью корректное состояние даже при сбоях в очень загруженные дисковой активностью моменты. Вы можете даже оптимизировать диск и в самый разгар этого процесса нажать reset - вероятность потерь данных даже в этом случае будет очень низка. Важно понимать, однако, что система восстановления NTFS гарантирует корректность файловой системы, а не ваших данных. Если вы производили запись на диск и получили аварию - ваши данные могут и не записаться. Чудес не бывает.

Сжатие

Файлы NTFS имеют один довольно полезный атрибут - "сжатый". Дело в том, что NTFS имеет встроенную поддержку сжатия дисков - то, для чего раньше приходилось использовать Stacker или DoubleSpace. Любой файл или каталог в индивидуальном порядке может хранится на диске в сжатом виде - этот процесс совершенно прозрачен для приложений. Сжатие файлов имеет очень высокую скорость и только одно большое отрицательное свойство - огромная виртуальная фрагментация сжатых файлов, которая, правда, никому особо не мешает. Сжатие осуществляется блоками по 16 кластеров и использует так называемые "виртуальные кластеры" - опять же предельно гибкое решение, позволяющее добиться интересных эффектов - например, половина файла может быть сжата, а половина - нет. Это достигается благодаря тому, что хранение информации о компрессированности определенных фрагментов очень похоже на обычную фрагментацию файлов: например, типичная запись физической раскладки для реального, несжатого, файла:

* кластеры файла с 1 по 43-й хранятся в кластерах диска начиная с 400-го
* кластеры файла с 44 по 52-й хранятся в кластерах диска начиная с 8530-го

Физическая раскладка типичного сжатого файла:

* кластеры файла с 1 по 9-й хранятся в кластерах диска начиная с 400-го
* кластеры файла с 10 по 16-й нигде не хранятся
* кластеры файла с 17 по 18-й хранятся в кластерах диска начиная с 409-го « кластеры файла с 19 по 36-й нигде не хранятся

Блок 16 кластеров ►

виртуальные

реальные кластеры

е

виртуальные

9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Q

4 5 6

9|10|11|12|13[14|15|1б[17|18

физические кластеры (на диске)

Видно, что сжатый файл имеет "виртуальные" кластеры, реальной информации в которых нет. Как только система видит такие виртуальные кластеры, она тут же понимает, что данные предыдущего блока кратного 16-ти, должны быть разжаты, а получившиеся данные как раз заполнят виртуальные кластеры - вот, по сути, и весь алгоритм.

Безопасность

NTFS содержит множество средств разграничения прав объектов - есть мнение, что это самая совершенная файловая система из всех ныне существующих. В теории это, без сомнения, так, но в текущих реализациях, к сожалению, система прав достаточно далека от идеала и представляет собой хоть и жесткий, но не всегда логичный набор характеристик. Права, назначаемые любому объекту и однозначно соблюдаемые системой, эволюционируют - крупные изменения и дополнения прав осуществлялись уже несколько раз и к Windows 2000 все-таки они пришли к достаточно разумному набору.

Права файловой системы NTFS неразрывно связаны с самой системой - то есть они, вообще говоря, необязательны к соблюдению другой системой, если ей дать физический доступ к диску. Для предотвращения физического доступа в Windows2000 всё же ввели стандартную возможность - об этом см. ниже. Система прав в своем текущем состоянии достаточно сложна, и я сомневаюсь, что смогу сказать широкому читателю что-нибудь интересное и полезное ему в обычной жизни. Если вас интересует эта тема - вы найдете множество книг по сетевой архитектуре NT, в которых это описано более чем подробно.

На этом описание строение файловой системы можно закончить, осталось описать лишь некоторое количество просто практичных или оригинальных вещей.

Hard Links

Эта штука была в NTFS с незапамятных времен, но использовалась очень редко - и тем не менее: Hard Link - это когда один и тот же файл имеет два имени (несколько указателей файла-каталога или разных каталогов указывают на одну и ту же MFT запись). Допустим, один и тот же файл имеет имена l.txt и 2.txt: если пользователь сотрет файл 1, останется файл 2. Если сотрет 2 - останется файл 1, то есть оба имени, с момента создания, совершенно равноправны. Файл физически стирается лишь тогда, когда будет удалено его последнее имя.

Symbolic Links (NT5)

Гораздо более практичная возможность, позволяющая делать виртуальные каталоги - ровно гак же, как и виртуальные диски командой subst в DOSe. Применения достаточно разнообразны: во-первых, упрощение системы каталогов. Если вам не нравится каталог Documents and settings\Administrator\Documents, вы можете прилинковать его в корневой каталог - система будет по прежнему общаться с каталогом с дремучим путем, а вы - с гораздо более коротким именем, полностью ему эквивалентным. Для создания гаких связей можно воспользоваться программой junction, которую написал известный специалист Mark Russinovich (<http://www.sysinternals.com/>). Программа работает только в Windows 2000. как и сама возможность.

Для удаления связи можно воспользоваться стандартной командой rd. ВНИМАНИЕ: Попытка удаления связи с помощью проводника или других файловых менеджеров, не понимающих виртуальную природу каталога (например, FAR), приведет к удалению данных, на которые ссылается ссылка! Будьте осторожны.

к

Шифрование (EFS)

Полезная возможность для людей, которые беспокоятся за свои секреты - каждый файл или каталог может также быть зашифрован, что не даст возможность прочесть его другой инсталляцией NT. В сочетании со стандартным и практически непрошибаемым паролем на загрузку самой системы, эта возможность обеспечивает достаточную для большинства применений безопасность избранных вами важных данных.

Особенности дефрагментации NTFS

Вернемся к одному достаточно интересному и важному моменту - фрагментации и дефрагментации NTFS. Дело в том, что ситуация, сложившаяся с этими двумя понятиями в настоящий момент, никак не может быть названа удовлетворительной. В самом начале утверждалось, что NTFS не подвержена фрагментации файлов. Это оказалось не совсем так, и утверждение сменили - NTFS препятствует фрагментации. Оказалось, что и это не совсем так. То есть она, конечно, препятствует, но толк от этого близок к нулю... Сейчас уже понятно, что NTFS - система, которая как никакая другая предрасположена к фрагментации, что бы ни утверждалось официально. Единственное что - логически она не очень от этого страдает. Все внутренние структуры построены таким образом, что фрагментация не мешает быстро находить фрагменты данных. Но от физического последствия фрагментации - лишних движений головок - она, конечно, не спасает. И поэтому - вперед и с песней...

К истокам проблемы...

Как известно, система сильнее всего фрагментирует файлы когда свободное место кончается, когда приходится использовать мелкие дырки, оставшиеся от других файлов. Тут возникает первое свойство NTFS, которое прямо способствует серьезной фрагментации.

Диск NTFS поделен на две зоны. В начала диска идет MFT зона - зона, куда растет MFT, Master File Table. Зона занимает минимум 12% диска, и запись данных в эту зону невозможна. Это сделано для того, чтобы не фрагментировался хотя бы MFT. Но когда весь остальной диск заполняется - зона сокращается ровно в два раза :). И так далее. Таким образом мы имеем не один заход окончания диска, а несколько. В результате если NTFS работает при диске, заполненном на около 90% - фрагментация растет как бешенная.

Попутное следствие - диск, заполненный более чем на 88%, дефрагментировать почти невозможно - даже API дефрагментации не может перемешать данные в MFT зону. Может оказаться так, что у нас не будет свободного места для маневра.

Далее. NTFS работает себе и работает, и всё таки фрагментируется - даже в том случае, если свободное место далеко от истощения. Этому способствует странный алгоритм нахождения свободного места для записи файлов - второе серьезное упущение. Алгоритм действий при любой записи такой: берется какой-то определенный объем диска и заполняется файлом до упора. Причем по очень интересному алгоритму: сначала заполняются большие дырки, потом маленькие. Т.е. типичное распределение фрагментов файла по размеру на фрагментированной NTFS выглядит так (размеры фрагментов):

16-16-16-16- 16-[скачек назад] - 15-15- 15-[назад] - 14-14- 14.... 1 - 1 - I -I -1...

Так процесс идет до самых мелких дырок в 1 кластер, несмотря на то, что на диске наверняка есть и гораздо более большие куски свободного места.

Вспомните сжатые файлы - при активной перезаписи больших объемов сжатой информации на NTFS образуется гигантское количество "дырок" из-за перераспределения на диске сжатых объемов - если какой-либо участок файла стал сжиматься лучше или хуже, его приходится либо изымать из непрерывной цепочки и размещать в другом месте, либо стягивать в объеме, оставляя за собой дырку.

Смысл в сего этого вступления в пояснении того простого факта, что никак нельзя сказать, что NTFS препятствует фрагментации файлов. Наоборот, она с радостью их фрагментирует. Фрагментация NTFS через пол года работы доведет до искреннего удивления любого человека, знакомого с работой файловой системой. Поэтому приходится запускать дефрагментатор. Но на этом все наши проблемы не заканчиваются, а, увы. только начинаются...

Средства решения?

В NT существует стандартное API дефрагментации, обладающее интересным ограничением для перемещения блоков файлов: за один раз можно перемещать не менее 16 кластеров (!). причем начинаться эти кластеры должны с позиции, кратной 16 кластерам в файле. В общем, операция осуществляется исключительно по 16 кластеров. Следствия:

* 1. В дырку свободного места менее 16 кластеров нельзя ничего переместить (кроме сжатых файлов, но это неинтересные в данный момент тонкости).
  2. Файл, будучи перемещенный в другое место, оставляет после себя (на новом месте) "временно занятое место", дополняющее его по размеру до кратности 16 кластерам.
  3. При попытке как-то неправильно ("не кратно 16") переместить файл результат часто непредсказуем. Что-то округляется, что-то просто не перемещается... Тем не менее, всё место действия щедро рассыпается "временно занятым местом".

"Временно занятое место" служит для облегчения восстановления системы в случае аппаратного сбоя и освобождается через некоторое время, обычно где-то пол минуты.

Тем не менее, логично было бы использовать это API, раз он есть. Его и используют. Поэтому процесс стандартной дефрагментации, с поправками на ограниченность API, состоит из следующих фаз (не обязательно в этом порядке):

• Вынимание файлов из MFT зоны. Не специально - просто обратно туда их положить не представляется возможным — безобидная фаза, и даже в чем-то полезная.

* Дефрагментация файлов. Безусловно, полезный процесс, несколько, правда, осложняемый ограничениями кратности перемещений - файлы часто приходится перекладывать сильнее, чем это было бы логично сделать по уму.
* Дефрагментация MFT, виртуалки (pagefile.sys) и каталогов. Данная операция возможна через API только в Windows2000. иначе - при перезагрузке, отдельным процессом, как в старом Diskeeper-e.
* Складывание файлов ближе к началу - дефрагментация свободного места.

Допустим, мы хотим положить файлы подряд в начало диска. Кладем один файл. Он оставляет хвост занятости дополнения до кратности 16. Кладем следующий - после хвоста, естественно. Через некоторое время, по освобождению хвоста, имеем дырку <16 кластеров размером. Которую потом невозможно заполнить через API дефрагментации! В результате, до оптимизации картина свободного места выглядела так: много дырок примерно одинакового размера. После оптимизации - одна дыра в конце диска, и много маленьких <16 кластеров дырок в заполненном файлами участке. Какие места в первую очередь заполняются? Правильно, находящиеся ближе к началу диска мелкие дырки <16 кластеров... Любой файл, плавно созданный на прооптимизированном диске, будет состоять из дикого числа фрагментов. Да, диск потом можно оптимизировать снова. А потом еще раз., и еще., и гак - желательно каждую неделю. Бред? Реальность.

Таким образом, имеется два примерно равнозначных варианта. Первый - часто оптимизировать диск таким дефрагментатором, смиряясь при этом с дикой фрагментацией заново созданных файлов. Второй вариант - вообще ничего не трогать, и смириться с равномерной, но гораздо более слабой фрагментацией всех файлов на диске.

Пока есть всего один дефрагментатор, который игнорирует API дефрагментации и работает как-то более напрямую - Norton Speeddisk. Когда его пытаются сравнить со всеми остальными - Diskeeper, О&О defrag, т.д. - не упоминают этого главного, самого принципиального, отличия. Просто потому, что эта проблема тщательно скрывается, по крайней мере не афишируется. Speeddisk - единственная на сегодняшний день программа, которая может оптимизировать диск полностью, не создавая маленьких незаполненных фрагментов свободного места. Стоит добавить также, что при помощи стандартного API невозможно дефрагментировать тома NTFS с кластером более 4 Кбайт, a SpeedDisk и это может.

К сожалению, в Windows 2000 и Windows ХР поместили дефрагментатор, который работает через API, и, соответственно, плодит дырки <16 кластеров. Поэтому рекомендуется использовать Speeddisk из состава Norton Utilities.

Как некоторый вывод из всего этого: все остальные дефрагментаторы при одноразовом применении просто вредны. Если вы запускали его хоть раз - нужно запускать его потом хотя бы раз в месяц, чтобы избавится от фрагментации новоприбывающих файлов. В этом основная суть сложности дефрагментации NTFS теми средствами, которые сложились исторически.

Что выбрать?

Любая из представленных ныне файловых систем уходит своими корнями в глубокое прошлое - еще к 80-м годам. Да, NTFS, как это не странно - очень старая система! Дело в том, что долгое время персональные компьютеры пользовались лишь операционной системой DOS, которой и обязана своим появлением FAT. Но параллельно разрабатывались и тихо существовали системы, нацеленные на будущее. Две таких системы, получившие всё же широкое признание - NTFS, созданная для операционной системы Windows NT 3.1 еще в незапамятные времена, и HPFS - верная спутница OS/2.

Внедрение новых систем шло трудно - еще в 95м году, с выходом Windows95, ни у кого не было и мыслей о том, что что-то нужно менять - FAT получил второе дыхание посредством налепленной сверху заплатки "длинные имена", реализация которых там хоть и близка к идеально возможной без изменения системы, но всё же довольно бестолкова. Но в последующие годы необходимость перемен назрела окончательно, поскольку естественные ограничения FAT стали давать о себе знать. FAT32, появившаяся в Windows 95 OSR2, просто сдвинула рамки - не изменив сути системы, которая просто не дает возможности организовать эффективную работу с большим количеством данных.

HPFS (High Performance File System), активно применяемая до сих пор пользователями OS/2, показала себя достаточно удачной системой, но и она имела существенные недостатки - полное отсутствие средств автоматической восстанавливаемости, излишнюю сложность организации данных и невысокую гибкость.

NTFS же долго не могла завоевать персональные компьютеры из-за того, что для организации эффективной работы с её структурами данных требовались значительные объемы памяти. Системы с 4 или 8 Мбайт (стандарт 95-96 годов) были просто неспособны получить хоть какой-либо плюс от NTFS, поэтому за ней закрепилась не очень правильная репутация медленной и громоздкой системы. На самом деле это не соответствует действительности - современные компьютерные системы с памятью более 64 Мб получают просто огромный прирост производительности от использования NTFS.

В данной таблице сведены воедино все существенные плюсы и минусы распространенных в наше время систем, таких как FAT32, FAT и NTFS. Вряд ли разумно обсуждать другие системы, так как в настоящее время 97% пользователей делают выбор между Windows 2000 и Windows ХР, а других вариантов там просто нет.

|  |  |  |  |
| --- | --- | --- | --- |
|  | FAT | FAT32 | NTFS |
| Системы, её поддерживающие | DOS, Windows 95/98/Ме, Windows NT/2000/X Р/Vista | Windows 98/Ме. Windows 2000/ХР/Vista | Windows  NT/2000/XP/Vista |
| Максимальный размер тома | 2 Гбайт | практически неограничен | практически неограничен |
| Макс, число файлов на томе | примерно 65 тысяч | практически не ограничено | практически не ограничено |
| Имя файла | с поддержкой длинных имен - 255 символов, системный набор символов | с поддержкой длинных имен - 255 символов, системный набор символов | 255 символов, любые символы любых алфавитов (65 тысяч разных начертаний) |
| Возможные атрибуты файла | Базовый набор | Базовый набор | всё, что придет в голову производителям програм много обеспечения |
| Безопасность | нет | нет | да (начиная с Windows 2000 встроена возможность физически шифровать данные) |
| Сжатие | да(только программные средства MSDOS - DoubleSpace, DriveSpace, Stacker) | нет | да |
| Устойчивость к сбоям | средняя (система слишком проста и поэтому ломаться особо нечему :)) | плохая (средства оптимизации по скорости привели к появлению слабых по | полная - автоматическое восстановление системы при любых сбоях (не считая физические ошибки записи, когда |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | надежности мест) | пишется одно, а на самом деле записывается другое) |
| Экономичность | минимальная (огромные размеры кластеров на больших дисках) | улучшена за счет уменьшения размеров кластеров | максимальна. Очень эффективная и разнообразная система хранения данных |
| Быстродействие | высокое для малого числа файлов, но быстро  уменьшается с появлением большого количества файлов в каталогах, результат - для слабо заполненных дисков -  максимальное, для заполненных - плохое | полностью аналогично FAT. но на дисках большого размера (десятки гигабайт) начинаются серьезные проблемы с общей организацией данных | система не очень эффективна для малых и простых разделов (до 1 Гбайт), но работа с огромными массивами данных и внушительными каталогами организована как нельзя более эффективно и очень сильно превосходит по скорости другие системы |

Хотелось бы сказать, что если ваша операционная система - Windows 2000 или Windows ХР, то использовать какую-либо файловую систему, отличную от NTFS - значит существенно ограничивать свое удобство и гибкость работы самой операционной системы. Windows 2000 и Windows ХР, составляет с NTFS как бы две части единого целого - множество полезных возможностей NT напрямую завязано на физическую и логическую структуру файловой системы, и использовать там FAT или FAT32 имеет смысл лишь для совместимости - если у вас стоит задача читать эти диски из каких-либо других систем.

Приложение 4. Файловая система ZFS

ZFS

ZFS (Zettabyte File System) — файловая система, разработанная компанией Sun Microsystems и обладающая такими характеристиками как возможность хранения больших объемов данных, управления томами и множеством других. Первоначально файловая система была разработана для ОС Solaris, но впоследствии перенесена на ряд других операционных систем, в том числе на FreeBSD и Mac OS X.

Файловая система ZFS распространяется с открытым кодом по opensource-лицензии CDDL (несовместимой с GPL).

Появление ZFS наделало много шума, и до сих пор эта файловая является предметом горячих обсуждений. Причин у такой популярности несколько, в первую очередь:

* Большие технические возможности файловой системы, в частности возможность хранения огромных объёмов данных и интеграция возможностей управления томами, квот, резервирования и контроля целостности данных в саму файловую систему.
* Факт уже выполненного её портирования на несколько платформ и продолжение портирования на другие на фоне запрета на портирование на Linux.

Файловая система ZFS позволяет адресовать ZFS = 128-bit = 3\* 10A26 [ЗЕ26] ТВ (на одну ФС)

Разговор о ZFS хотелось бы начать с нескольких слов о проблемах, присущих современным файловым системам (ФС), и причинах их возникновения.

Во-первых, большинство современных файловых систем не имеют средств защиты от повреждений данных. Если данные на пути от диска до оперативной памяти машины будут повреждены, ни одна из систем, за исключением ZFS, не сообщит вам об этом.

Во-вторых, существующие файловые системы «бесчеловечны», когда речь идет об администрировании. Прежде чем создать файловую систему, необходимо диски отформатировать, разбить на разделы, создать тома, выбрать размер блока, определить, насколько большой должна быть система, до каких размеров она может вырасти и сколько файлов может быть создано... После того как файловая система создана, нужно добавить ее в конфигурационные файлы, чтобы обеспечить автоматическое монтирование при загрузке или предоставлении доступа к ней удаленным клиентам. При этом необходимо все время помнить ее параметры и наложенные ограничения (размер блока, максимальный размер системы, количество файлов, каталогов и т. д.). Если руководствоваться максимальными возможностями конкретной файловой системы, то можно столкнуться с неприятными сюрпризами.

В третьих, существующие файловые системы часто не обеспечивают достаточной скорости, поскольку создавались во времена, когда еще не было многопроцессорных машин, диски были небольшими и их было немного, а пользователи просто не могли иметь миллионы файлов в каталогах. Архитектура старых файловых систем во многом и отражает эти ограничения. Время создания ФС на диске часто линейно зависит от ее емкости, размер блока фиксирован, стратегия блокировок не оптимизирована для одновременной работы нескольких потоков и т. д.

Цель создания ZFS — решить проблемы традиционных ФС. Управление данными должно быть простым и комфортным, доступ к данным быстрым, и, что главное, данные должны быть в безопасности.

Разработчики ZFS понимали, что проблемы существующих файловых систем решить невозможно и нужно начинать «с чистого листа», отбросив устаревшие на 20 лет принципы.

Три кита ZFS - основные принципы конструкции самой ZFS.

Первое — организация всего доступного дискового пространства в единый пул (пул накопителей). Основная идея — избавиться от управления отдельными дисками и томами и перейти к использованию концепций виртуальной памяти в проекции на дисковое пространство. Если нужно установить в сервер новые модули памяти, достаточно выключить его (при отсутствии средств динамической реконфигурации), установить новые модули и включить сервер снова. Не требуется никаких команд вроде dimmconfig, создания виртуальных «модулей памяти», разграничения прав... Все гораздо проще. Когда программе нужна память, она выдает запрос ОС. и та предоставляет ее. Неважно, где будет выделена эта память. Можно использовать такой же подход и для дискового пространства, ZFS работает именно так. Имеется пул накопителей, к нему добавляем имеющиеся диски (почти так же, как новые модули памяти). В результате появляются многочисленные новые возможности.

Второе — сквозной контроль целостности данных. Проверяется каждый блок данных, чтобы убедиться, что он такой, каким должен быть. 20 лет назад это считалось очень дорогостоящей (в смысле вычислительных ресурсов) операцией. Сегодня, когда скорость процессоров по сравнению со скоростью дисков возросла на порядки, оказалось, что это совсем недорого. Используемые в ZFS алгоритмы на современном процессоре класса Opteron позволяют вычислять контрольные суммы со скоростью 2—8 Гбайт/с. Это означает, что на большинстве современных машин можно пожертвовать небольшой долей процессорного времени для гарантирования целостности данных. Альтернатива — «неявные» повреждения данных, не проявляющиеся при чтении с диска.

Третье — транзакционность. Данные всегда хранятся в целостном и непротиворечивом состоянии. Существующие данные и метаданные на диске никогда не перезаписываются. В случае возникновения проблем всегда можно вернуться в предыдущее состояние. Все это позволяет снять практически все ограничения на порядок выполнения дисковых операций (что позволяет получить существенный выигрыш в производительности).

Масштабируемость и производительность

ZFS — первая в мире 128-битная ФС. Несмотря на кажущуюся огромность дискового пространства, которое позволяет адресовать 64-битная ФС, оказалось, что до момента реализации всех ее возможностей потребуется не слишком много времени. Уже сейчас есть заказчики, оперирующие петабайтными объемами данных (2 в степени 50). Для достижения предела потребуется всего лишь 14 удвоений, и, если существующие тенденции сохранятся, 65-й бит потребуется уже через 10-15 лет. Поэтому для ZFS было выбрано 128 бит. Достаточно ли этого? Наверное, да, во всяком случае, расчеты показывают, что предел размера ФС в ZFS превышает возможности накопителя, который теоретически можно создать на нашей планете, а энергии, которая потребуется только для того, чтобы установить в нужное состояние 2 в степени 128 бит, хватит, чтобы несколько раз вскипятить весь мировой океан.

Все метаданные в ZFS — полностью динамические. Нет никаких статических регионов, индексных дескрипторов (inode) и т. п., все подобные структуры выделяются динамически, гак что фактический предел размера и количества файлов, каталогов, файловых систем, мгновенных снимков, клонов и т. д. определяется только объемом доступной дисковой памяти. Соответственно отсутствуют и разные непонятные ограничения, вроде количества inode в одной группе цилиндров.

При проектировании ZFS были учтены уроки масштабирования Solaris, поэтому все операции в ZFS максимально параллельны. Можно параллельно считывать и писать один и тот же файл из разных потоков или процессов без нарушения семантики POSIX, иметь множество процессов, создающих и удаляющих файлы в одном каталоге. Все блокировки в ZFS полностью масштабируемы и очень быстры.

Многое, о чем мы говорили, приводит к высокой производительности. Во-первых, копирование при записи и возможность записывать любые данные куда угодно дают возможность превратить случайные операции записи в последовательные, а это шаг к повышению производительности. Динамический размер блока позволяет выбирать его в соответствии с параметрами нагрузки. Планировщик ввода-вывода в ZFS в совокупности с интегрированностью системы обеспечивает огромные возможности планирования, агрегирования и выполнения операций ввода-вывода. При этом учитываются взаимозависимости между операциями, приоритеты и крайние сроки выполнения.

Динамическое чередование помогает наиболее полно реализовать пропускную способность дисков. При статическом чередовании, предусмотренном в RAID-0, используется фиксированное число дисков. В ZFS можно добавить диски в пул, при этом все новые записываемые данные будут распределяться по всем имеющимся дискам, а старые данные, благодаря COW, тоже будут постепенно перераспределены по всем дискам.

Еще один механизм, обеспечивающий высокую производительность ZFS, — интеллектуальная предвыборка. Он не просто работает на уровне блоков, но и учитывает, какой процесс или поток обращается к этим блокам. Если у вас несколько процессов, последовательно читающих разные файлы в разных местах диска, то ZFS сможет это обнаружить и считывать данные, которые понадобятся каждому из процессов, заранее. Традиционные ФС в таких случаях, как правило, сдаются, поскольку считают получающиеся последовательности операций чтения случайными. Более того, ZFS умеет определять размер и шаг операций чтения, что делает ее подходящей для выполнения высокопроизводительных вычислений, обрабатывающих большие массивы данных, которые хранятся в файлах.

В ZFS предусмотрен эффективный механизм поддержки приложений, нуждающихся в гарантии того, что данные записаны на диск. Он называется ZFS Intent Log или ZIL и гарантирует сохранение данных на диске, не теряя преимуществ отложенного сохранения изменений в рамках групп транзакций. Вскоре этот механизм планируется усовершенствовать так, чтобы он позволил использовать для хранения ZIL отдельные быстродействующие устройства, например отдельные быстрые или твердотельные диски, а это даст существенный выигрыш в производительности.

Приложение 4. Журналируемые файловые системы для Linux

Стандартной файловой системой для Linux была Ext2. Эта файловая система была разработана Wayne Davidson в сотрудничестве с Stephen Tweedie и Theodore Ts'o. Это - улучшение предыдущей системы Ext, разработанной Remy Card. Ext2 основана на структуре i-node. I-node содержит информацию о файле и указатели на блоки данных, в которых расположен файл. Для повышения быстродействия операций ввода/вывода данные временно располагаются в оперативной памяти. Проблема возникает, если сбой происходит до того, как данные из кэша перепишутся на диск. Это вызывает несоответствие в файловой системе. Например, возникает ссылка на файл, еще не созданный на диске, или файлы были уже удалены, но их i-nodes и блоки данных остались на диске. Fsck (File system check - проверка файловой системы) - стандартная программа для устранения несоответствий. Единственный способ это сделать - просканировать весь диск, и проверить все зависимости между i-nodes, блоками данных и содержанием директорий. С увеличением объемов дисков эта процедура стала занимать огромное количество времени - серьезная проблема для серверов, которые должны работать постоянно.

Это и стало главной причиной внедрения в файловые системы технологии транзакций, взятой из баз данных, и технологии восстановления, что привело к появлению журналируемых файловых систем.

Журналируемая файловая система - это устойчивая к сбоям файловая система, в которой целостность данных гарантирована, потому что обновления meta-данных записываются в лог на диске перед любыми изменениями в структуре файловой системы. В случае сбоя журналируемая файловая система гарантирует восстановление всех потерянных данных. Самый распространенный подход - это метод журналирования или логгирования meta- данных файлов. Его суть в том, что информация о любом изменении записывается в зарезервированную область файловой системы, и только после этого совершается само изменение.

Самые ранние журналируемые файловые системы, созданные в середине 80-х, включали в себя Veritas (VxFS), Tolerant, и IBM JFS. С увеличением спроса на поддержку файловыми системами терабайтных данных, миллионов файлов и 64-битной архитектуры интерес к журналируемым FS для Linux возрос за последние годы.

Linux имеет четыре новых журналируемых FS - это ReiserFS от Namesys, XFS от SGI, JFS от IBM, и Ext3, разработанная Stephen Tweedie, учавствовавшим в создании Ext2.

Если ReiserFS - это полностью новая файловая система, написанная с нуля, то XFS, JFS и Ext3 появились из коммерческих продуктов или уже существующих файловых систем. XFS базирована на системе, разработанной SGI. JFS была спроектирована и разработана ГВМ для OS/2 Warp.

ReiserFS - единственная журналируемая FS, включенная в стандартное ядро Linux, а другие только собираются быть включенными в ядро версии 2.5. Но все равно - все они являются полнофункциональными, и выпущены, как официальные патчи к ядру Linux.

Ext3 - это расширение для Ext2. Она добавляет два независимых модуля - для транзакций и ведения логов. Ext3 уже близка к финальной версии, и она уже включена в дистрибутив RedHat 7.2.

ReiserFS

ReiserFS базируется на В+Tree в организации объектов файловой системы. Объекты файловой системы - это структуры, используемые для хранения информации о файле - время последнего изменения, права доступа, etc. (Другими словами - вся информация, содержащаяся в i-node директорий и файлов). ReiserFS называет эти объекты "stat data'"', "directory items" и "direct/indirect items". ReiserFS предоставляет только журналирование meta-данных. В случае незапланированной перезагрузки данные в блоках, используемых во время сбоя, могут быть повреждены, так что ReiserFS не гарантирует того, что после сбоя данные останутся неповрежденными.

Unformatted nodes - это логические блоки без определенного формата, используемые для хранения файловых данных, и direct items состоят из самих файловых данных. Блоки с самими данными могут быть различных размеров. Одна из специальных возможностей ReiserFS - это Tail Packing. Tail - это файл, размер которого меньше, чем логический блок, или какие-то части файлов, занимающие меньше, чем один блок. Для сохранения свободного места ReiserFS использует сжатие tail-файлов, и это позволяет примерно на 5% увеличить свободное место по сравнению с Ext2. ТаЛ-файлы могут быть интегрированы в В- Тгее, и поэтому быстродействие ReiserFS при работе с очень маленькими файлами намного выше.

Проблема в том, что использование этой технологии соединения tail-файлов вместе увеличивает фрагментацию данных. Даже больше, задача запаковывания tail-файлов отнимает время и ведет к потерям в быстродействии. Это - следствие сдвигов памяти, происходящих, когда кто-нибудь добавляет данные к файлу. В Namesys признали реальность этой проблемы и позволили администратору системы отключить tail packing с помощью опции notail.

ReiserFS использует фиксированный размер блоков (4 Kb), и это отрицательно влияет на быстродействие операций ввода-вывода при работе с большими файлами. Другая слабость ReiserFS - работа с сильно фрагментированными файлами производится значительно медленнее по сравнению с Ext2, так что Namesys работает над оптимизацией этого процесса.

XFS

1 мая 2001 года SGI выпустила первый релиз своей журналируемой файловой системы для Linux. Особенности XFS - поддержка больших дисков и очень высокая скорость ввода- вывода (на тестировании доходило до 7 Gb в секунду). XFS была разработана для операционной системы IRIX 5.3 SGI Unix, и первая версия этой файловой системы была представлена в декабре 1994 года. Цель этой файловой системы - поддержка очень больших файлов и высокая пропускная способность для проигрывания и записи видео в реальном времени.

Для увеличения масштабируемости файловой системы XFS обширно использует B+Trees. Они используются для пометки свободных расширителей (extents), индексации директорий и для контроля за динамически размещаемыми i-nodes, разбросанными по всей файловой системе. В дополнение ко всему этому XFS использует асинхронную схему логгирования с журналированием (для защиты meta-данных от сбоев после обновления), и позволяет быстро восстановить файловую систему.

XFS использует систему распределения места, основанную на расширителях, и имеет такие возможности, как delayed allocation, space рге-allocation и space coalescing on deletion. Данные располагаются с использованием самых больших расширителей из имеющихся, и это позволяет записывать файлы огромной длины. Чтобы сделать управление огромными количествами непрерывных фрагментов эффективным, XFS использует очень большие описания для расширителей. Каждое описание может включать в себя информацию о блоках файловой системы в количестве до 2 миллионов. Описание больших количеств блоков разгружает процессор от такой работы, как поиск записей в extent map. чтобы определить, являются ли блоки непрерывными. Теперь можно просто прочитать длину всего расширителя, который уже является непрерывным, а не смотреть на каждую запись, определяя, является ли она продолжением предыдущей записи.

XFS позволяет использовать различные размеры блоков - от 512 байт до 64 килобайт. Изменение размеров блоков может повлиять на фрагментацию. Файловые системы с большим количеством маленьких файлов обычно используют маленькие размеры блоков, чтобы избежать лишних трат свободного места в случаях, когда размер файла меньше, чем размер блока, а системы с большими файлами обычно делают противоположный выбор и используют большие размеры блоков, чтобы уменьшить фрагментацию файловой системы.

XFS - это комплексная разработка, основанная на IRIX, и она очень связана с IRIX, поэтому при портировании на Linux все было полностью переработано и написано с нуля. Результат - модуль pagebuf для Linux, обеспечивающий интерфейс между XFS и подсистемой виртуальной памяти, а так же между XFS и Linux block device layer.

XFS поддерживает ACL'bi (с помощью интеграции с сервером Samba), и transactional quotes. В Linux она поддерживает квотирование для группы, в отличии от IRIX квотирования для проекта, потому что именно так квотирование реализуется в файловых системах для Linux. (В Linux нет концепции, эквивалентной "проектам" в IRIX). Более необычные функции XFS из оригинальной версии для 1RIX, позволяющие оказывать специальные сервисы различным приложениям (например, обслуживание video в реальном времени) еще не были перенесены в Linux-версию.

Нормальный режим работы для XFS - это использование асинхронно ведущегося логгирования. Из-за этого XFS приобретает 2 вещи:

1. Множественные обновления могут быть произведены одной операцией записи в лог. Это увеличивает эффективность ведения лога.
2. Быстродействие изменения meta-данных обычно делается независимым от скорости устройства, на котором записаны данные. Вообще-то, эта независимость ограничена количеством буферизации, используемой для ведения лога, но все равно - это намного лучше, чем синхронные обновления, используемые в старых файловых системах.

XFS также имеет большое количество инструментов для dump'HHra, восстановления, наращивания, использования ACL'ob и дисковых квот, и т.д.

JFS

IBM в первый раз представила свою собственную журналируемую файловую систему для Unix в выпуске AJX версии 3.1. Сейчас они предлагают более новую файловую систему для AIX-систем, названную JFS2, которая доступна в AIX версии 5.0 и выше. Открытый вариант JFS сейчас отличается оттого, который сейчас распространяется с OS/2 Warp Server'oM.

JFS отличается высокой пропускной способностью и надежностью, удовлетворяющей требованиям серверов. В этой файловой системе используется адресная структура, базированная на расширениях (extent-based), и расположение блоков в кластерах, что позволяет создавать компактные, эффективные и масштабируемые структуры расположения файлов. Расширение - это несколько последовательно идущих блоков, относящихся к файлу, описываемых тремя параметрами - логическое удаление (offset), длина и физические параметры. Структура адресации - это В+Tree с описаниями расширений, расположенными в i-node.

JFS ведет логи, в которых записывается информация об операциях над метаданными. Формат лога устанавливается утилитой создания файловой системы.

Особенность логов JFS в том, что когда операция изменения метаданных файловой системы возвращает успешный результат, эффекты этой операции уже записаны, и они останутся в логах даже тогда, когда сбой происходит сразу же после выполнения операции.

JFS поддерживает блоки размера 512, 1024, 2048 и 4096 байт. Небольшая величина блока уменьшает фрагментацию, но может увеличивать длину пути, потому что перераспределение блоков может происходить более часто, чем если бы были использованы большие блоки. Поэтому стандартный размер - 4096 байт.

JFS динамически распределяет место для i-nodes, и освобождает место, когда они больше не

нужны. Эта файловая система поддерживает две различные организации директорий:

\*

* 1. Используется для маленьких директорий. Оглавление директории хранится в ее i-node.
  2. Для больших директорий. Каждая директория представляется, как В+Tree с ее именем. Это дает возможность быстрого поиска, вставки и удаления директорий.

JFS поддерживает и "разряженные" и "плотные" файлы. У первого типа файлов данные разбросаны по различным местам, и размер файла - это просто номер самого последнего записанного байта. А физическим расположением самих блоков занимается операция записи.

Ext3

Вообще-то, это Ext2 с файлом журнала. Ext3 - это всего лишь половина настоящей журналируемой системы, надстройка над Ext2. С одной стороны, она позволяет вести лог операций для более быстрого восстановления. Но эта файловая система унаследовала некоторые ограничения от Ext2 (например, она базируется на блоках и использует полный перебор при поиске файлов и директорий), и поэтому ее нельзя назвать чистой журналируемой файловой системой.

ext3 или 3-я расширенная файловая система — журналируемая файловая система, используемая в операционных системах на ядре Linux, является файловой системой по умолчанию во многих дистрибутивах. Основана на ФС ext2.

Основное отличие от ext2fs 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).

Указывается режим журналирования в строке параметров для программы mount, например: mount /dev/hda6 /mnt/disc -о data=<peжим>

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 размер 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[1 ].

Главные преимущества:

* Ext3 записывает изменения данных и метаданных. Поэтому, в отличии от предыдущих файловых систем, сохраняется и содержимое файлов. Уровень журналирования может контролироваться опцией команды mount.
* Разделы Ext3 не отличаются от Ext2 на уровне файловой структуры, поэтому можно портировать файлы со старой системы на новую, или делать backup на старую систему.

Ext3 резервирует одну из специальных Ext2 i-nodes для хранения логов, но сам журнал может быть на любой i-node в любой файловой системе. Возможно иметь несколько файловых систем Ext3 с совместным журналом.

Работа файла журналирования - записывать состояние системных метаданных во время процесса совершения транзакций. В журнал пишется состояние трех типов данных: метаданные, блоки описания и Header блоки.

Журналируемый блок метаданных всегда записывается полностью, даже если в файловой системе произошло очень маленькое изменение. Это делается относительно быстро, потому что журналируемые операции ввода/вывода могут быть объединены в большие кластеры и блоки могут быть записаны прямо из кэша с помощью использования структуры bufferhead.

Блоки описания описывают другие блоки метаданных, так что механизм восстановления может копировать метаданные назад в файловую систему. Они записываются перед изменением журнала метаданных.

Header блоки описывают заголовок, окончание журнала, и порядок номеров - для того, чтобы гарантировать упорядоченную запись во время восстановления.

Сравнение быстродействия и выводы:

Различные тесты показали, что XFS и ReiserFS имеют очень хорошее быстродействие по сравнению с многократно тестированной и оптимизированной Ext2. Ext3 работает

39

медленнее, но приближается к Ext2. (Ожидается, что ее быстродействие будет улучшаться). JF5' показала самые плохие результаты не тольког быстродействии, но и в стабильности.

XFS, ReiserFS и Ех13 доказали, что они - быстрые и надежные файловые системы. XFS имеет более высокое быстродействие в такой важной области, как работа с большими файлами. Сейчас ReiserFS полностью перечитывает дерево при записи каждого 4KB блока, и вставляет по одному указателю одновременно, и поэтому происходит повышение затрат на балансирование дерева.

При операциях с маленькими файлами, обычно между 100 и 10000 байт, ReiserFS показала лучшие результаты в том случае, когда эти файлы еще не находятся в кэше. А при чтении файлов из кэша разница почти незаметна для Ext2, Ext3, ReiserFS и XFS.

Среди всех файловых систем только ReiserFS была включена в ядро Linux, начиная с версии 2.4.1, и SuSE поддерживает ее уже более 2 лет. Ext3 собирается быть стандартной файловой системой для Red Hat. a XFS используется на больших серверах, особенно в Голливуде, потому что там большое влияние имеет SGI. A IBM нужно вложить много усилий в JFS. если они хотят сделать ее распространенной файловой системой, и альтернативой для перехода с А1Х и OS/2 на Linux.