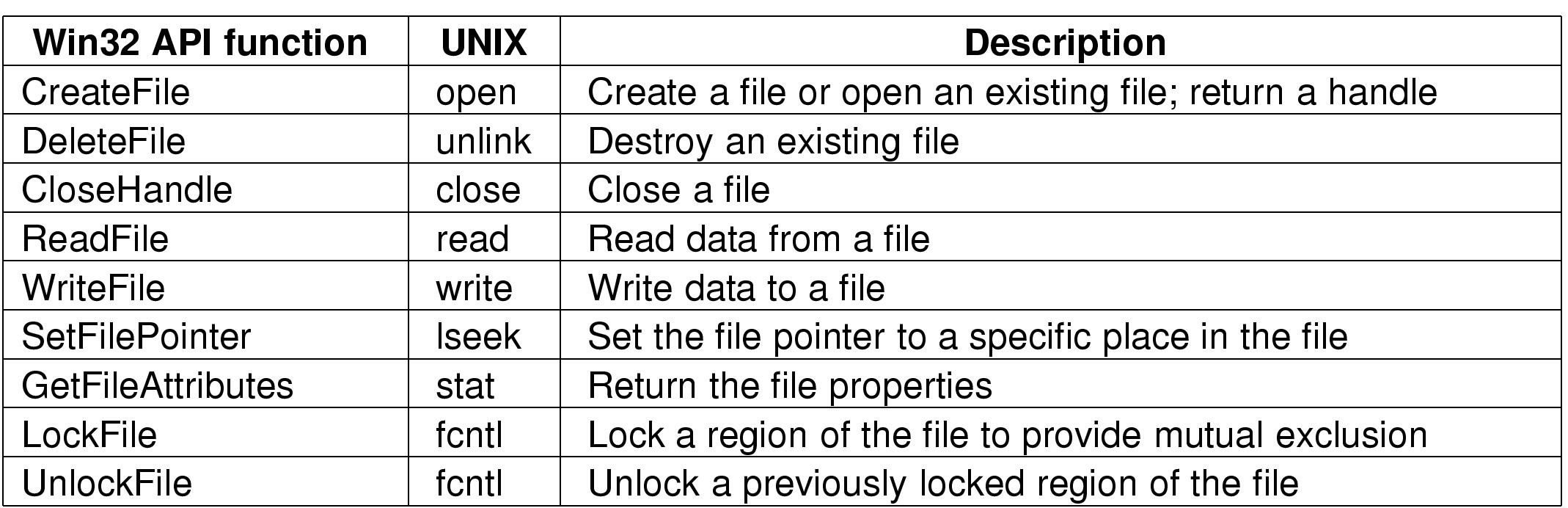
FAT\_2006 **Файловые системы**

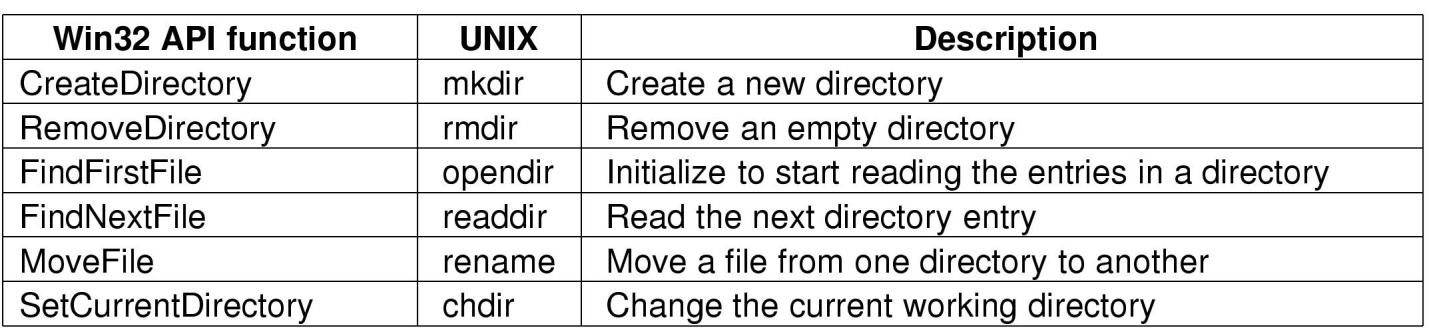
**Работа с файлами в Windows API**

File System API Calls in Windows 2000

****

* Principle Win32 API functions for file I/O
* Second column gives nearest UNIX equivalent

File System API Calls in Windows 2000

****

* Principle Win32 API functions for directory management
* Second column gives nearest UNIX equivalent, when one exists

Работа с томами

* Для выяснения того, какие логические диски существуют в системе, используется функция

**DWORD GetLogicalDrives( void )**

Каждый установленный бит возвращаемого значения соответствует существующему в системе логическому устройству. Например, если в системе существуют диски A:, C: и D:, то возвращаемое функцией значение равно 13(10).

* Функция

**DWORD GetLogicalDrivesStrings( DWORD cchBuffer, LPTSTR lpszBuffer)**

заполняет lpszBuffer информацией о корневом каталоге каждого логического диска в системе. В приведенном выше примере буфер будет заполнен символами

A:\<null>C:\<null>D:\<null><null>

параметр cchBuffer определяет длину буфера. Функция возвращает реальную длину буфера, необходимую для размещения всей информации.

* Для определения типа диска предназначена функция

**UINT GetDriveType( LPTSTR lpszRootPathName )**

В качестве параметра ей передается символическое имя корневого каталога (напр. **A:\**), а возвращаемое значение может быть одно из следующих:

|  |  |
| --- | --- |
| Идентификатор | Описание |
| 0 | Тип устройства определить нельзя |
| 1 | Корневой каталог не существует |
| DRIVE\_REMOVABLE | Гибкий диск |
| DRIVE\_FIXED | Жесткий диск |
| DRIVE\_REMOTE | Сетевой диск |
| DRIVE\_CDROM | Компакт диск |
| DRIVE\_RAMDISK | RAM диск |

* Для получения подробной информации о носителе используется функция **GetVolumeInformation**. Она заполняет параметры информацией об имени тома, названии файловой структуры, максимальной длине имени файла, дополнительных атрибутах тома, специфических для файловой структуры.
* Функция **GetDiskFreeSpace** сообщает информацию о размерах сектора и кластера и о наличии свободных кластеров.

Работа с каталогами и файлами

|  |  |
| --- | --- |
| Функция | Выполняемое действие |
| GetCurrentDirectory | Получение текущего каталога |
| SetCurrentDirectory | Смена текущего каталога |
| GetSystemDirectory | Получение системного каталога |
| GetWindowsDirectory | Получение основного каталога системы |
| CreateDirectory | Создание каталога |
| RemoveDirectory | Удаление каталога |
| CopyFile | Копирование файла |
| MoveFile  MoveFileEx | Перемещение или переименование файла |
| DeleteFile | Удаление файла |

Синхронная работа с файлами

HANDLE **CreateFile** (

LPCTSTR *lpFileName*, // pointer to name of the file

DWORD *dwDesiredAccess*, // access (read-write) mode

DWORD *dwShareMode*, // share mode

LPSECURITY\_ATTRIBUTES *lpSecurityAttributes*, // pointer to security // descriptor

DWORD *dwCreationDistribution*,// how to create

DWORD *dwFlagsAndAttributes*, // file attributes

HANDLE *hTemplateFile* // handle to file with attributes to copy

);

В случае удачи функция **CreateFile** возвращает описатель открытого файла как объекта ядра. Существенно, что в противном случае она возвращает не NULL, а INVALID\_HANDLE\_VALUE.

При синхронной работе с файлами прикладная программа, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).

В Win32 перед принятием решения об использовании операций синхронной работы с файлами настоятельно рекомендуется рассмотреть вопрос о возможности использования файлов, отображаемых в память.

* Параметр **lpFileName** определяет имя файла.
* Параметр **dwDesiredAccess** задает тип доступа к файлу. Можно определить флаги GENERIC\_READ и GENERIC\_WRITE а так же их комбинацию для разрешения чтения или записи в файл.
* Параметр **dwShareMode** определяет режим совместного использования файла различными процессами. Если этот параметр равен нулю, то никакой другой поток не сможет открыть этот же файл. Флаги FILE\_SHARE\_READ и FILE\_SHARE\_WRITE а так же их комбинация разрешают другим потокам осуществлять доступ к файлу для чтения или записи.
* Параметр **lpSecurityAttributes** указывает на структуру, описывающую защиту создаваемого объекта ядра. Ему может быть присвоено значение NULL.
* Параметр **dwCreationDistribution** определяет действия функции в зависимости от того, существует ли уже файл с указанным именем.
* Параметр **dwCreationDistribution** определяет действия функции в зависимости от того, существует ли уже файл с указанным именем.

CREATE\_NEW - Создает файл, если файл существует, то ошибка.

CREATE\_ALWAYS - Создает файл , если файл существует, то старый файл удаляется и новый создается.

OPEN\_EXISTING - Открывает существующий файл.

OPEN\_ALWAYS - Создает файл, если файл не существует, то создается новый файл.

TRUNCATE\_EXISTING - Открывает файл и урезает его до нулевой длины

* Параметр **dwFlagsAndAttributes** определяет атрибуты файла, если он создается и задает режим работы с файлом.

FILE\_ATTRIBUTE\_ARCHIVE, FILE\_ATTRIBUTE\_HIDDEN, FILE\_ATTRIBUTE\_NORMAL, FILE\_ATTRIBUTE\_READONLY, FILE\_ATTRIBUTE\_SYSTEM, FILE\_ATTRIBUTE\_TEMPORARY

Атрибуты файла могут комбинироваться за исключением FILE\_ATTRIBUTE\_NORMAL, который всегда используется в одиночестве.

Вместе с атрибутами могут комбинироваться и флаги, задающие режим работы с файлом.

FILE\_FLAG\_NO\_BUFFERING - Не осуществлять кэширование и опережающее чтение

FILE\_FLAG\_RANDOM\_ACCESS - Кэшировать как файл произвольного доступа

FILE\_FLAG\_SEQUENTIAL\_SCAN - Кэшировать как файл последовательного доступа

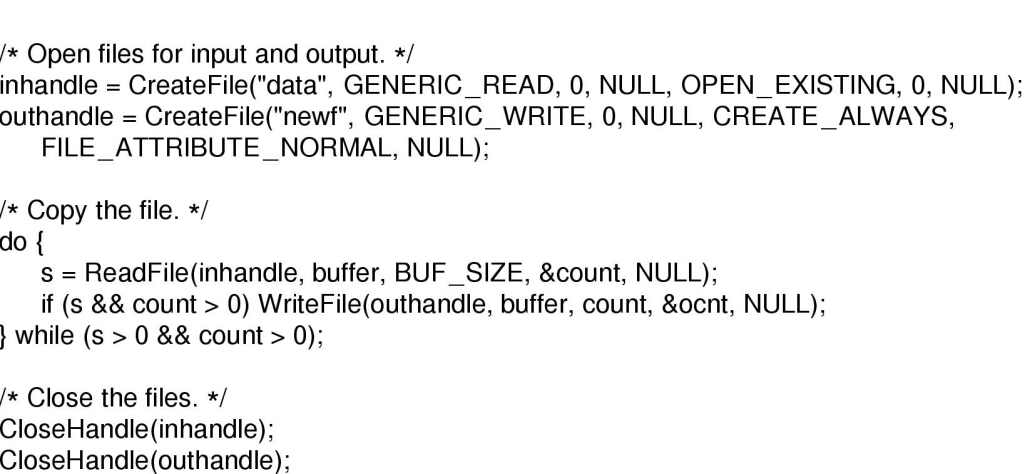
FILE\_FLAG\_WRITE\_TROUGH - Не буферизовать операцию записи. Производить запись на диск немедленно.

FILE\_FLAG\_DELETE\_ON\_CLOSE - Уничтожить файл при закрытии. Полезно комбинировать с атрибутом FILE\_ATTRIBUTE\_TEMPORARY.

FILE\_FLAG\_OVERLAPPED - Работа с файлом будет осуществляться асинхронно.

* Параметр **hTemplateFile** либо равен NULL, либо задает описатель открытого файла. Во втором случае параметр dwFlagsAndAttributes игнорируется, а указанный файл используется в качестве шаблона, т.е. используются его флаги и атрибуты.

File System API Calls in Windows 2000



A program fragment for copying a file using the Windows 2000 API functions

Асинхронная работа с файлами

BOOL **ReadFile**(

HANDLE hFile, // handle of file to read

LPVOID lpBuffer, // address of buffer that receives data

DWORD nNumberOfBytesToRead,// number of bytes to read

LPDWORD lpNumberOfBytesRead,// address of number of bytes read

LPOVERLAPPED lpOverlapped // address of structure needed for // overlapped I/O

);

BOOL **WriteFile**(

HANDLE hFile, // handle to file to write to

LPCVOID lpBuffer, // pointer to data to write to file

DWORD nNumberOfBytesToWrite, // number of bytes to write

LPDWORD lpNumberOfBytesRead,// pointer to number of bytes written

LPOVERLAPPED lpOverlapped // address of structure needed for //overlapped I/O

);

При асинхронной работе с файлами прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения а продолжает исполняться.

Windows’95 не позволяет организовать асинхронную работу с файлами, но тот же самый асинхронный механизм может использоваться при работе с последовательными портами, транспортерами и почтовыми ящиками.

Для организации асинхронной работы с файлами необходимо при вызове функции CreateFile установить флаг FILE\_FLAG\_OVERLAPPED в параметре dwFlagsAndAttributes. После этого функции ReadFile и WriteFile будут работать асинхронно, т.е. только запускать операции ввода вывода и не ожидать их завершения.

Параметры функции ReadFile имеют следующее предназначение:

* hFile – описатель объекта ядра “файл”, полученный в результате вызова функции CreateFile
* LpBuffer – адрес буфера, в который будет производиться чтение
* nNumberOfBytesToRead – количество байт, которые необходимо прочитать
* lpNumberOfBytesRead – адрес переменной, в которой будет размещено количество реально прочитанных байт. Существенно, что сразу после выполнения функции ReadFile, этот параметр не может быть установлен, так как операция чтения только началась.
* lpOverlapped – указатель на структуру OVERLAPPED, управляющую асинхронным вводом выводом.

Параметры функции WriteFile аналогичны параметрам функции ReadFile.

typedef struct \_OVERLAPPED {

DWORD *Internal*; //Используется операционной системой. //Хранит статус завершения операции.

DWORD *InternalHigh*; //Используется ОС. Хранит //количество переданных байт.

DWORD *Offset*; //Позиция в файле, начиная с которой //необходимо производить операцию

//чтения (записи).

DWORD *OffsetHigh*;//Количество байт для передачи.

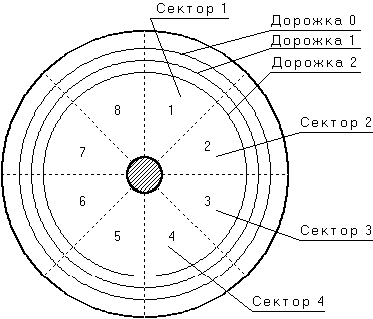
HANDLE *hEvent*; //Описатель события, которое произойдет //при завершении операции чтения //(записи).

} OVERLAPPED;

В отличие от синхронных операций, при организации асинхронного чтения (записи) необходимо явно указать позицию, начиная с которой производится операция. Это связано с тем, что текущей позиции не существует, так как несколько операций чтения и записи могут производиться одновременно с разных позиций в одном файле.

Схема организации асинхронных чтения и записи

* Перед запуском операции создается объект ядра “событие” и его описатель передается в функцию ReadFile (WriteFile) в качестве элемента hEvent параметра lpOverlapped. Программа, выполнив необходимые действия одновременно с операцией передачи данных, вызывает одну из функций ожидания (напр. WaitForSingleObject), передавая ей в качестве параметра описатель события. Выполнение программы при этом приостанавливается до завершения операции ввода-вывода.
* Событие не создается. В качестве ожидаемого объекта выступает сам файл. Его описатель передается в функцию WaitForSingleObject. Этот метод прост и корректен, но не позволяет производить параллельно несколько операций ввода-вывода с одним и тем же файлом.
* “Тревожный” асинхронный ввод-вывод. Схема построена на использовании функций ReadFileEx и WriteFileEx. В качестве дополнительного параметра в эти функции передается адрес функции завершения, которая будет вызываться всякий раз при завершении операции ввода-вывода. Существенно, что эти функции выполняются в том же самом потоке что и функции файлового ввода/вывода. Это значит, что поток, запустивший операции чтения записи должен обратиться к функции ожидания, чтобы разрешить системе вызвать функцию завершения.

**Файловые системы фирмы Microsoft**

Магнитный диск

МД представляет собой один или несколько объединенных дисков (блинов), по поверхностям (Sides) которых перемещаются головки (Heads).

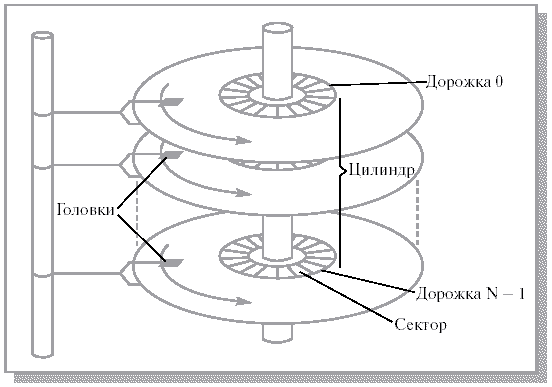
Головки "бегают" по круговым дорожкам (Tracks), каждая дорожка разделена на сектора (Sectors). Дорожки, равноудаленные от центра диска и образующие цилиндрическую поверхность, называют цилиндрами (Cilinders).

***Форматирование дисков.*** Для того чтобы на диске можно было хранить информацию, диск должен быть отформатирован, то есть должна быть создана физическая и логическая структура диска.

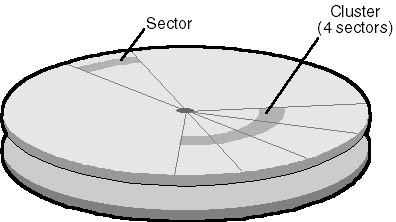
Форматирование физической структуры диска состоит в создании на диске концентрических дорожек, которые, в свою очередь, делятся на секторы. Для этого в процессе форматирования магнитная головка дисковода расставляет в определенных местах диска метки дорожек и секторов.

***Логическая структура дисков***. Логическая структура магнитного диска представляет собой совокупность секторов, каждый из которых имеет свой порядковый номер. Сектора нумеруются в линейной последовательности от первого сектора нулевой дорожки до последнего сектора последней дорожки.

Строение жесткого диска

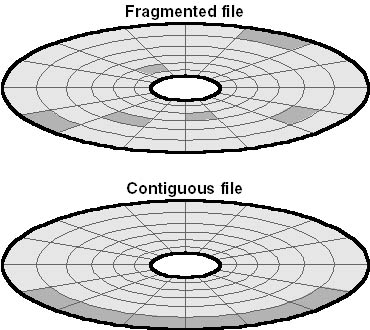
* Современный жесткий магнитный диск представляет собой набор круглых пластин, находящихся на одной оси и покрытых с одной или двух сторон специальным магнитным слоем
* Около каждой рабочей поверхности каждой пластины расположены магнитные головки для чтения и записи информации. Эти головки присоединены к специальному рычагу, который может перемещать весь блок головок над поверхностями пластин как единое целое. Поверхности пластин разделены на концентрические кольца, внутри которых, собственно, и может храниться информация. Набор концентрических колец на всех пластинах для одного положения головок (т. е. все кольца, равноудаленные от оси) образует цилиндр. Каждое кольцо внутри цилиндра получило название дорожки (по одной или две дорожки на каждую пластину). Все дорожки делятся на равное число секторов. Количество дорожек, цилиндров и секторов может варьироваться от одного жесткого диска к другому в достаточно широких пределах. Как правило, сектор является минимальным объемом информации, которая может быть прочитана с диска за один раз.

Кластеры

В ОС фирмы Windows основной единицей хранения информации является кластер – группа смежных секторов. Число секторов в кластере всегда равно степени двойки.

Увеличение размера кластера приводит к нерациональному использованию пространства МД для мелких файлов.

Фрагментация и дефрагментация

Файл, который занимает на диске более одного непрерывного участка, называется *фрагментированным.*

**Фрагментация диска** - это появление на диске множества свободных участков, разделенных занятыми участками.

**Дефрагментация диска** - это перемещение данных на разделе, после которого, кластеры содержащие части одного файла, размещаются последовательно.

Файловые системы фирмы Microsoft выделяют для записываемых на диск файлов некоторое количество кластеров, в зависимости от размера файла.

В процессе работы с диском при записи и удалении файлов разного размера на диске появятся свободные и занятые области разной длины.

Такой метод хранения файлов позволяет использовать всё имеющееся на диске свободное место, т.к. если длина записываемого файла больше, чем размеры непрерывных свободных участков, то файл просто расположится в нескольких несмежных участках.

Однако при использовании описанного выше метода файл становится фрагментированным - он как бы "размазан" по диску. К чему это может привести?

К тому, что для доступа к файлу необходимо перемещать магнитные головки от одного участка файла к другому. А на это требуется время, и весьма значительное.

Реально время чтения сильно фрагментированного файла по сравнению с файлом, занимающим непрерывную область на диске, может отличаться в несколько раз! Внешне это выглядит так, как будто все программы стали работать в несколько раз медленнее, при этом наблюдается интенсивное перемещение головок диска от одного участка файла к другому.

* Файловые системы фирмы Microsoft выделяют для записываемых на диск файлов некоторое количество кластеров, в зависимости от размера файла.
* В процессе работы с диском при записи и удалении файлов разного размера на диске появятся свободные и занятые области разной длины.
* Такой метод хранения файлов позволяет использовать всё имеющееся на диске свободное место, т.к. если длина записываемого файла больше, чем размеры непрерывных свободных участков, то файл просто расположится в нескольких несмежных участках.
* Реально время чтения сильно фрагментированного файла по сравнению с файлом, занимающим непрерывную область на диске, может отличаться в несколько раз! Внешне это выглядит так, как будто все программы стали работать в несколько раз медленнее, при этом наблюдается интенсивное перемещение головок диска от одного участка файла к другому.

Физические и логические диски

Основные причины разбиения физического диска на несколько логических:

* ограничения файловых систем на максимальный размер физического диска;
* повышение надежности файловой системы;
* поддержка нескольких ОС.

**Файловая система FAT для MS DOS**

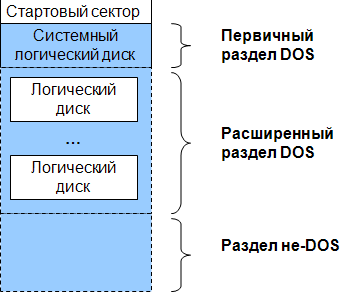
История FAT16

* 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 CP/M 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 вначале предназначена для работы с гибкими дисками размером менее 1 Мбайт и не предусматривала поддержки жестких дисков.

Структура FAT16

* Первый сектор жёсткого диска (сектор 1, дорожка 0) содержит так называемую главную загрузочную запись (Master Boot Record = MBR)*,* которая загружается в память под управлением BIOS и выполняется.
* В конце первого сектора HDD находится таблица разделов диска (Partition table). Эта таблица содержит до четырёх элементов, описывающих разделы диска.
* Разделы могут трех типов: первичными (1), расширенными (1), не-DOS разделами (2-3). Основной раздел может содержать код загрузки операционной системы. Расширенный раздел может быть дополнительно разбит на подразделы. Каждому разделу и подразделу операционная система ставит в соответствие логический диск и назначает свое имя (C:, D:, E: и т.д.)

Таблица разделов логического диска

Самый первый сектор НЖМД содержит *Главную корневую запись,* которая загружается в память под управлением BIOS и выполняется. Последняя часть этого сектора содержит таблицу разделов: 4-элементную таблицу с 16-байтовыми элементами. Этой таблицей манипулирует программа FDISK (или эквивалентная ей в иной операционной системе).

Во время загрузки BIOS загружает главную корневую запись и передает ей управление. Она считывает таблицу разделов, чтобы определить раздел, помеченный как активный. Затем в память считывается корректный загрузочный сектор и выполняется.

Таблица логических дисков имеет формат, аналогичный формату таблицы разделов диска, но содержит только два элемента. Один из них указывает на первый сектор логического диска, он имеет код системы, соответствующий типу FAT логического диска. Второй элемент может иметь код системы равный 5 или 0. Если он равен пяти, то элемент указывает на следующую таблицу логических дисков, если код системы равен нулю, то раздел не используется.

Расширенный раздел DOS является необязательным, такой раздел используется обычно для создания дополнительных логических дисков.

Структура логического диска FAT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Загрузочная запись | FAT | FAT (копия) | Корневой каталог | Область файлов |

* **Загрузочная запись** (первый сектор диска) – служит для загрузки ОС и организация хранения данных.
* **FAT** (File Allocation Table) – таблица размещения файлов.
* **Корневой каталог** – для FAT16 512 записей о файлах и каталогах, расположенных в корне файловой системы.

Структура раздела FAT изображена на слайде.

Файловая система FAT имеет древовидную структуру. В корневом каталоге располагаются 32-байтовые элементы, которые содержат информацию о файлах и других каталогах.

Корневой каталог занимает непрерывную область фиксированного размера. Размер корневого каталога задаётся при форматировании и определяет максимальное количество файлов и каталогов, которые могут быть в нём описаны.

Вслед за корневым каталогом на логическом диске находится область файлов и подкаталогов корневого каталога.

Область данных разбита на кластеры, причём нумерация кластеров начинается с двойки.

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 является степенью двойки и определяется размером тома при форматировании диска. Кластер представляет собой минимальное пространство, которое может занимать файл.

Кластеру с номером 2 соответствуют первые секторы области данных.

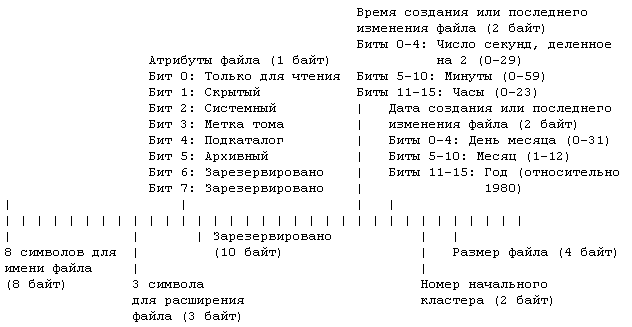
Любой каталог содержит 32-байтовые элементы — дескрипторы, описывающие файлы и другие каталоги.

Элемент каталога FAT16

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

* Размер элемента каталога – 32 байта

Элемент каталога FAT16



«Время изменения» и «Дата изменения» кодируются по следующим формулам:

Время = (часых2048)+(минутых32)+(секунды+2);   
Дата = ((год–1980)х512)+(месяц х 32)+день.

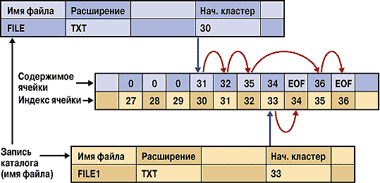
Функции FAT

* Хранение информации о размещении файлов на диске
* Хранение информации о свободном месте на диске
* Хранение информации о сбойных кластерах на диске

Организация данных в FAT

* FAT представляет собой таблицу, связывающую кластеры дискового пространства с файлами. В этой базе для каждого кластера предусматривается только один элемент.
* Первые два элемента содержат информацию о самой системе FAT. Третий и последующие элементы ставятся в соответствие кластерам дискового пространства, начиная с первого кластера, отведенного для файлов.
* Элементы FAT могут содержать несколько специальных значений, указывающих, что
  + кластер свободен, т.е. не использован ни одним файлом (для FAT16 это значение составляет 0000H);
  + кластер содержит один или несколько секторов с физическими дефектами и не должен использоваться (дл FAT16 это значение составляет FFF7H);
  + данный кластер - последний кластер файла (дл FAT16 это значение составляет FFF8 - FFFFH).
* Для любого используемого файлом, но не последнего кластера элемент FAT содержит номер следующего кластера, занятого файлом, из-за этого FAT называют файловой системой со связанным списком индексов.

Логическая организация данных

На слайде представлена схема работы и организации FAT, а также фрагментация, когда части файла разбросаны по всему диску.

Размеры разделов и кластеров FAT16 для Windows 95-2000

|  |  |
| --- | --- |
| **Размер раздела** | **Размер кластера** |
| 0 – 32 Мб | 512 б |
| 33 – 64 Мб | 1 Кб |
| 65 – 128 Мб | 2 Кб |
| 129 Мб – 256 Мб | 4 Кб |
| 257 Мб – 512 Мб | 8 Кб |
| 513 Мб – 1024 Мб | 16 Кб |
| 1 Гб – 2 Гб | 32 Кб |
| 2 Гб – 4 Гб (Win’2000) | 64 Кб |

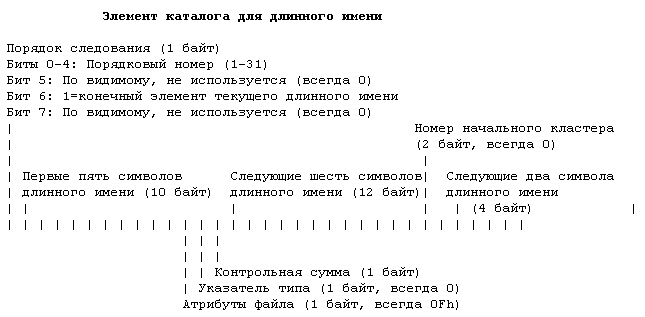
**Файловая система VFAT**

* ФС Virtual FAT появилась в первой версии Win’95.
* Во второй версии Win’95 пользователям была предложена уже FAT32.
* Virtual FAT поддерживала длинные имена файлов (LFN), но была 16-разрядной.
* Длинные имена (LFN) хранятся в специально отформатированных 32-байт записях, байт атрибутов у которых равен 0Fh.
* Поддерживает разделы до 4 Гб.

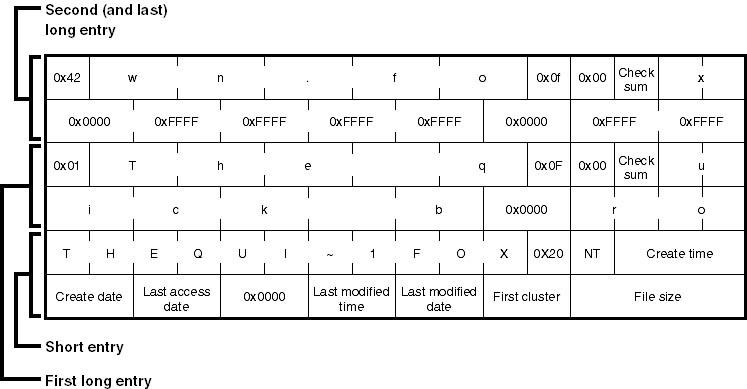
Long File Names

* FAT32 преодолела ограничение прежней системы наименования файлов "8.3". В VFAT имя файла может содержать до 255 символов. К счастью, FAT32 воспринимает файлы, которые уже существовали на диске, даже если эти файлы используются 16-битовыми приложениями, разработанными не для Windows 95. Для каждого имени файла VFAT создает *псевдоним,* соответствующий нотации "8.3". Например, у файла "Файл с длинным именем" будет псевдоним "файлсд~1 .doc" в FAT. Более того, VFAT тома будут совместимы с DOS и Windows 3.1. Например, дискету, которую создали в Windows 95 и на которую записали несколько файлов с длинными именами, можно прочесть в DOS или Windows 3.1.
* Кроме того, в этой файловой системе может быть несколько расширений, разделяемых точкой. Однако тип файла определяется по последнему расширению, а остальные рассматриваются как имя файла.
* Длинные имена (LFN) хранятся в специально отформатированных 32-байт записях, байт атрибутов у которых равен 0Fh. Для конкретного файла или подкаталога непосредственно перед его единственной записью каталога с его именем в формате 8.3 находится группа из одной или нескольких записей, представляющих длинное имя. Каждая такая запись содержит часть длинного имени файла не более 13 символов, и ОС составляет полное длинное имя из всех записей.
* Длинные имена файлов хранятся на диске в указанном формате и размещаются в одном или нескольких 32-байт элементах каталога перед элементами каталога для коротких имен. Символы, составляющие имя файла, представлены в кодах Unicode, т. е. на каждый из них по 2 байта.

Элемент каталога VFAT



Пример длинного имени



Проблемы длинных имен

* На первый взгляд использованный в FAT32 механизм длинных имен файлов позволяет сохранить преемственность с прикладными программами прошлого поколения и выглядит идеальным. Однако этот метод далек от совершенства:
  + Требуется больше дискового пространства
  + Б**ó**льшая фрагментация (на уровне каталогов)

**Файловая система FAT32**

* FAT32 это развитие файловой системы FAT(VFAT, FAT16).
* 32-разрядная адресация кластеров – максимальное число адресуемых кластеров – 4 294 377 472.
* Поддержка  больших разделов (более 4Gb), кроме этого уменьшен размер кластера на разделе.
* Поддержка длинных имен до 255 символов, причем нет ограничений на число и размер расширения.
* Корневой каталог, раньше имевший фиксированный размер и строго определенное место на диске, теперь можно свободно наращивать по мере необходимости подобно подкаталогу. Теперь не существует ограничений на число записей в корневом каталоге. Это особенно важно, поскольку под каждое длинное имя файла используется несколько записей каталога.

Имеет более высокую надежность: FAT32 способна перемещать корневой каталог, создает резервную копию загрузочного сектора и может работать с резервной копией FAT.

Система MS-DOS до версии 7.0 включительно и система Windows могли распознать 65 536 логических блоков на диске. Начиная с Windows 95 OSR2, появилась возможность использовать для нумерации логических элементов на диске 32-разрядные данные, а значит, число адресуемых элементов теоретически возросло до 4 294 967 295.

Зеркализация FAT

* Исторически сложилось так, что на всех FAT-дисках существуют 2 экземпляра таблицы FAT. Ecли при чтении исходного экземпляра возникает ошибка, файловая система пытается считать его резервную копию. На дисках с 12-и 16-разрядной FAT первая таблица FAT всегда является основной, и все изменения автоматически записываются в ее копию. Создание резервной копии второй таблицы FAT называется зеркализацией (mirroring).
* В FAT32 зеркализацию второй таблицы FAT можно отключить. Тогда операции чтения/записи ускоряются, а если первая FAT оказывается поврежденной, используется ее второй экземпляр (он становится основным). На FAT32-дисках таблица FAT может достигать огромных размеров, и отключение зеркализации способно заметно ускорить доступ к файлам.

Структура элемента каталога

* Чтобы обеспечить возможность работы с возросшим числом кластеров, в записи каталога для каждого файла должно выделяться 4 байт для начального кластера файла .
* 2 дополнительных байта (по сравнению с FAT16 и VFAT) выделяются среди зарезервированных 10 байт.

Сравнение FAT16 и FAT32

* Самое принципиальное отличие заключается в том, что FAT32 намного эффективнее расходует дисковое пространство. FAT32 использует дисковые кластеры меньшего размера по сравнению с предыдущими версиями, которые ограничивались 65 535 кластерами на том (соответственно с увеличением размера диска приходилось увеличивать и размер кластеров). Следовательно, даже для дисков размером до 8 Гбайт FAT32 может использовать 4-килобайтные кластеры. В результате по сравнению с дисками FAT16 экономится в среднем 10-15% дискового пространства.
* FAT32 также может перемещать корневой каталог и использовать резервную копию FAT вместо стандартной. Расширенная загрузочная запись FAT32 позволяет создавать копии критических структур данных; это повышает устойчивость дисков FAT32 к нарушениям структуры FAT по сравнению с предыдущими версиями. Корневой каталог в FAT32 представлен в виде обычной цепочки кластеров. Следовательно, корневой каталог может находиться в произвольном месте диска, что снимает действовавшее ранее ограничение на размер корневого каталога (512 элементов).

|  |  |
| --- | --- |
| **FAT 16** | |
| - | максимальный размер раздела – 2Гб (для Win2000 – 4Гб); |
| - | при размере раздела > 512 Мб неэкономно расходует место на диске (из-за большого размера кластера); |
| + | распознается большинством ОС, используемыми на ПК; |
| + | позволяет уплотнять диск программой сжатия данных Drivespace; |
| - | **имеет корневой каталог фиксированного размера (512 записей).** |

|  |  |
| --- | --- |
| **FAT 32** | |
| - | работает чуть медленнее, чем FAT16; |
| + | разделы до 2Тб; |
| + | большая эффективность использования места на диске; |
| - | нельзя уплотнить с помощью программы сжатия данных; |
| - | старые версии DOS и многие другие ОС не "видят" разделы с форматом FAT 32; |
| + | корневой каталог является обычным расширяемым каталогом. |