**МИНИСТЕРСТВО ПУТЕЙ СООБЩЕНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

Департамент кадров и учебных заведений

**САМАРСКИЙ ИНСТИТУТ ИНЖЕНЕРОВ ЖЕЛЕЗНОДОРОЖНОГО
ТРАНСПОРТА ИМ. М.Т.ЕЛИЗАРОВА**

Кафедра ИИС

Технологии доступа к данным в информационных системах

Часть II
Файловая система NTFS

САМАРА 2002

УДК 681.3
Методические указания **Технологии доступа к данным в информационных системах: Файловая система NTFS. Часть II. — Самара: СамИИТ, 2002. – 22 с.**

Утверждено на заседании кафедры ИСС 31.01.02 протокол № 6.
Печатается по решению редакционно-издательского совета института.

Материал данной работы составляет цикл работ, посвященных технологиям доступа к данным. Описываются основные файловые системы, используемые семейством операционных систем Windows. Излагаются логические структуры файловых систем и методы работы с ними.

Материал предназначен для студентов, изучающих информатику и решающих задачи доступа к данным. Рекомендуется в качестве учебника для использования в учебном процессе специальностей “Информационные технологии” и “Микроинформационные управляющие системы”.

Составитель: Павлов Андрей Юрьевич, доцент кафедры ИСС, к.т.н.

Рецензенты: Тихомиров А.А. – доцент кафедры ВТ СамГТУ, к.т.н.
Чертыковцев В.К. — проректор по информатизации СамИИТа, д.т.н., профессор.

Редактор: Шимина И.А.

Подписано в печать 19.03.02
Тираж 100 Заказ № 32

Введение

К недостаткам файловых систем FAT можно отнести следующее:

1. 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.
2. 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.
3. 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.
4. FAT не предотвращает 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.

Устранить перечисленные недостатки призвана новая операционная система NTFS (New Technology File System). Это восстанавливаемая, защищенная и надежная файловая система, поддерживающая диски и файлы большого объёма. В этой файловой системе обеспечиваются множественные потоки данных, имена в UNICODE, универсальное средство индексации, переназначение плохих кластеров и поддержка POSIX.

Структура NTFS

Как и другие структуры файловых систем, структура NTFS, располагается в одном из разделов жесткого диска, указанном в таблице разделов. Первые шестнадцать секторов в разделе NTFS распределены под загрузочную запись и код загрузки (дубликат сектора загрузочной записи находится в логическом центре диска). Содержимое загрузочной записи показано в таблице 1.

Дополнительные поля, содержащиеся в расширенном блоке параметров BIOS (таблица 2), позволяют в процессе загрузки находить главную таблицу файлов (MFT). (MFT в отличие от FAT не располагается по заранее определенному адресу на диске. Кроме того, в случае порчи секторов, распределенных под MFT, она может быть перемещена).

Загрузочная запись NTFS Таблица 1

Смещение	Длина	Пример значения	Содержание
(+0)	3	EB 5B 00	Инструкция JMP.
(+3)	8	NTFS	OEM идентификатор.
(+11)	25		Блок параметров BIOS.
(+36)	48		Расширенный блок параметров BIOS.
(+84)	426		Самозагружающийся код.
(+500)	2	55 AA	Маркер конца сектора.

Блок параметров BIOS и расширенный блок параметров BIOS в NTFS Таблица 2

(0)	2	Sect_siz	Количество байтов в одном секторе диска.
(+2)	1	Clustsiz	Количество секторов в одном кластере.
(+3)	2	Res_sect	Количество зарезервированных секторов.
(+5)	1	fat_cnt	Всегда 0.
(+6)	2	root_siz	Всегда 0.
(+8)	2	tot_sect	Общее количество секторов на носителе данных (в разделе). Не используется в NTFS.
(+10)	1	media	Байт-описатель среды носителя данных.
(+11)	2	fat_size	Всегда 0.
(+13)	2	sectors	Количество секторов на дорожке.
(+15)	2	heads	Количество сторон (головок).
(+17)	4	hidden_l	Количество скрытых секторов для раздела.
(+21)	4		Не используется в NTFS.
(+25)	4		Не используется в NTFS.
(+29)	8	Tot_secs	Общее количество секторов на логическом диске для раздела.
(+37)	8	MFTStart	Номер логического кластера начала MFT.
(+45)	8	MFTMirrStart	Номер логического кластера начала частичной копии MFT.
(+53)	4	ClusPerFile	Количество кластеров, распределяемых под файловую запись.
(+57)	4	ClusPerIndex	Количество кластеров, распределяемых под индексный блок. (Обычно 1).
(+61)	8	SerNumVol	Серийный номер тома.
(+69)	4	Checksum	Проверочная сумма.

Как и описанные выше файловые системы, NTFS память под файлы в разделе распределяет кластерами. NTFS поддерживает почти любые размеры кластеров — от 512 байт до 64 Кбайт, неким стандартом же считается кластер размером 4 Кбайт. Для задания физического местоположения в разделе, кластеры нумеруются от начала раздела до его конца. Номер кластера в этой последовательности соответствует логическому номеру кластера (logical cluster number, LCN).

Раздел NTFS условно делится на две части. Самый главный, служебный файл файловой системы NTFS $MFT не подлежит дефрагментации. И чтобы $MFT не фрагментировался при своем росте, первые 12% диска отводятся под так называемую MFT зону — пространство, в которое растет метафайл $MFT. Запись каких-либо данных в эту область невозможна. Остальные 88% диска представляют собой обычное пространство для хранения файлов.

Свободное место диска, однако, включает в себя все свободное место, в том числе и незаполненные куски MFT-зоны. 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 место для записи файлов. При освобождении места в обычной области MFT зона может снова расшириться. При этом не исключена ситуация, когда в этой зоне остались и обычные файлы: никакой аномалии тут нет. Что ж, система старалась оставить ее свободной, но ничего не получилось. В этом случае метафайл $MFT все-таки может фрагментироваться, хоть это нежелательно.

Каждый элемент файловой системы NTFS представляет собой файл. NTFS является объектно-ориентированной файловой системой, и файл в NTFS — это набор атрибутов, которыми она манипулирует. Такое представление обеспечивает большую гибкость для наращивания новых возможностей, некоторые из которых описаны ниже.

Главная таблица файлов. Основу структуры NTFS составляет главная таблица файлов (master file table, MFT), хранящаяся в файле $MFT и содержащая запись для каждого файла раздела (рис.1).

Первые 16 файлов, записи которых отмечены в MFT, носят служебный характер. Они называются метафайлами.

Первая запись описывает непосредственно главную файловую таблицу. За ней следует зеркальная запись MFT. Если первая запись MFT разрушена, то NTFS читает вторую запись для отыскания зеркального файла MFT, в котором хранятся копии первых шестнадцати строк MFT. Местоположения MFT и зеркального файла MFT записаны в секторе начальной загрузки.

Другие записи MFT содержат файлы журнала транзакций, тома, таблицы определения атрибутов, корневого каталога, битовой карты, загрузки, плохих кластеров.

Журнал транзакций, в котором регистрируются все операции, влияющие на структуру тома, включая создание файла и любые команды, изменяющие структуру каталогов, используется для восстановления тома NTFS после сбоя системы.

Файл тома содержит имя тома, версию NTFS и бит, который будучи установлен, сигнализирует, что содержимое тома повреждено и должно быть исправлено. Все это

содержится в атрибутах $VOLUME_NAME и $VOLUME_INFORMATION файловой записи для этого файла.

Таблица определения атрибутов задает типы атрибутов, поддерживаемые на томе, и указывает, можно ли их индексировать, восстанавливать операцией восстановления и т.д.

0	$MFT	MFT
1	$MFTMirr	Частичная копия MFT
2	$LogFile	Файл журнала транзакций
3	$Volume	Файл тома
4	$AttrDef	Таблица определения атрибутов
5	.	Корневой каталог
6	$BitMap	Файл битовой карты
7	$Boot	Загрузочный файл (Содержит загрузочную запись и код загрузки, описанные выше).
8	$BadClus	Файл плохих кластеров
9	$Quota	
10	$UpCase	Содержит таблицу трансляции символов Unicode верхнего регистра.
11 - 15		. . .
16		Пользовательские файлы и каталоги
. . .		. . .

Рис. 1. Главная таблица файлов

В корневом каталоге хранится список файлов и каталогов, содержащихся в корне дерева каталогов.

Файл битовой карты содержит схему распределения пространства на томе. Каждый бит данных этого файла представляет один кластер тома и указывает, свободен он или занят.

В загрузочном фале хранится код начального загрузчика операционной системы.

В файле плохих кластеров содержатся все точки повреждения тома.

Системный файл $QUOTA содержит спецификации квот пользователей. NTFS использует эту информацию для ограничения прав пользователя, имеющего квоту, выделенную ему администратором. Ограничивается доступный пользователю размер пространства на томе под хранение файлов.

Файл $UpCase содержит список всех Unicode символов в верхнем регистре.

Файловая ссылка. Файл на томе NTFS идентифицируется 64-разрядным значением, называемым файловой ссылкой. Файловая ссылка состоит из номера файла и номера последовательности.

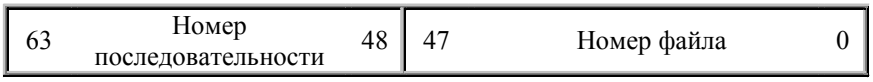

Рис.2. Файловая ссылка

Номер файла соответствует позиции его файловой записи в MFT минус 1.

Номер последовательности в файловой ссылки увеличивается всякий раз, когда данная позиция в MFT используется повторно. Этот номер должен совпадать со значением поля “Номер последовательности” файловой записи, на которую ссылается номер файла.

Файловая запись. Файловая запись, является элементом MFT и описывает файл, содержащийся в разделе NTFS. Если файл описывается несколькими файловыми записями, то первая из них называется **основной файловой записью**, а все другие — **дополнительными файловыми записями**. Дополнительная файловая запись используется в случае нехватки места под атрибуты файла из-за их большого количества или размера.

Файловая запись состоит из заголовка и последовательности атрибутов файла. Формат заголовка представлен в таблице 3.

Заголовок файловой записи Таблица 3

Смещение	Длина	Описание
0	4	Идентификатор файловой записи. Содержит 'FILE'.
4	2	Смещение поля «Счетчик обновления последовательности» относительно начала заголовка.
6	2	Количество элементов массива, содержащего значения обновления последовательности. S
16	2	Номер последовательности.
18	2	Счетчик жестких ссылок.
20	2	Смещение второй части файловой записи, последовательности атрибутов файла.
22	2	Флаг. 00 00 – файловая запись не используется; 0001 – файловая запись используется; 00 02 – файловая запись описывает директорию.
24	4	Реальный размер файловой записи.
28	4	Количество байт, выделенных под файловую запись.
32	8	Файловая ссылка на основную файловую запись. 0 – если это и есть файловая запись.
40	2	Идентификатор атрибута, имеющий максимальное значение, увеличенный на 1.
42	2	Поле «Счетчик обновления последовательности»
44	2*(S-1)	Массива обновления последовательности

Заголовок записи файла начинается с идентификатора. Он занимает первые четыре байта, в которых содержится слово FILE.

Второе поле используется для ссылки на поле “Счетчик обновления последовательности”, а в третьем содержится количество элементов (один элемент занимает два байта) массива обновления последовательности. Поле “Счетчик обновления последовательности” и массив значений обновления последовательности используются в технологии обновления последовательности, целью которой является обнаружение ошибок обновления последовательности секторов, составляющих файловую запись. Технология включает в себя следующее.

При сохранении файловой записи последовательно выполняются операции:

1. В оперативной памяти на 1 увеличивается значение поля “Счетчик обновления последовательности”.
2. Содержащееся в конце каждого сектора файловой записи, старое значение счетчика обновления последовательности заносится в массив значений обновления последовательности, и в конец каждого сектора заносится новое значение.
3. Копируем файловую запись из памяти на диск.

При считывании файловой записи последовательно выполняются операции:

1. Копируем с диска файловую запись в оперативную память.
2. В оперативной памяти проверяется корректность идентификатора записи.

3. Проверяется конец каждого сектора на равенство значению поля “Счетчик обновления последовательности”. В случае различия значений, означающего, что сектор на диске не был обновлен (например, из-за отключения питания при сохранении записи), в конец каждого сектора файловой записи заносится значение из массива значений обновления последовательности.

Одним из атрибутов файла является его имя. В Windows NT один и тот же файл может иметь несколько имен. В этом случае файл содержит несколько атрибутов имени, а в поле “Счетчик жестких ссылок” содержится их количество. Дополнительные имена файла являются жесткими ссылками.

За заголовком файла располагается последовательность атрибутов файла. Эта последовательность формируется в порядке возрастания значения типов атрибутов. Заканчивается последовательность **FF FF FF FF**.

Атрибут данных может иметь имя. В этом случае можно получить доступ к атрибуту из командной строки, используя нотацию **имя_файла : имя атрибута**.

Атрибут данных. Любой атрибут данных состоит из последовательности байт, называемых потоком и содержащих значение атрибута, и метаданных, используемых для доступа к потоку. Атрибут имеет **заголовок** и **содержание**.

Стандартная часть заголовка атрибута Таблица 4

Смещение	Длина	Описание
0	4	Тип. (Таблица 8)
4	4	Длина атрибута.
8	1	Флаг резидентности.
9	1	Длина имени. Содержит 00 если атрибут не имеет имени.
10	2	Смещение потока относительно начала атрибута.
12	2	Флаг компрессии. В NTFS позволяется осуществлять компрессию на уровне атрибутов.
14	2	Идентификатор.

Заголовок атрибута содержит постоянные поля (таблица 4), и поля зависящие от свойства резидентности атрибута.

Атрибут является резидентным, если его значение содержится в файловой записи. Для не резидентного атрибута значение содержится в отдельных областях предоставляемых файлу, а в содержании хранится отображение номеров виртуальных кластеров (VCN) в номера логических кластеров (LCN). VCN последовательно нумеруются, начиная с нуля, кластеры, принадлежащие данному атрибуту.

Для резидентного атрибута последующая часть заголовка показана в таблице 5.

Заключительная часть заголовка резидентного атрибута Таблица 5

Смещение	Длина	Описание
16	4	Длина потока
20	2	Смещение потока
22	2	Флаг индексации. Если установлен, то по этому атрибуту можно производить индексацию.

Для не резидентного атрибута последующая часть заголовка показана в таблице 6.

Содержание начинается с имени атрибута, за которым следует либо сам поток, либо отображение VCN в LCN.

Заключительная часть заголовка нерезидентного атрибута Таблица 6

Смещение	Длина	Описание
16	8	Начальный VCN
24	8	Последний VCN
32	2	Смещение области отображения VCN в LCN
34	2	Номер механизма компрессии.
40	8	Размер выделения под поток.
48	8	Реальный (некомпрессированный) размер потока.
56	8	Размер компрессированного потока.

Элемент таблицы отображения VCN в LCN Таблица 7

Смещение в полбайтах	**Длина**	**Описание**
0	1	F=размер поля смещения.
1	1	L=размер поля длины.
2	2*L	Длина (в кластерах) отрезка, выделяемого под поток атрибута.
2+2*L	2*F	Смещение относительно предыдущего LCN. Для первого элемента смещение относительно начала раздела.

Отображение VCN в LCN представляет собой таблицу, состоящую из элементов, формат которых приводится в таблице 7.

Такой формат используется для экономии места, так как использует переменную длину полей.

Список предопределенных атрибутов Таблица 8

Тип	Метка
10	$STANDART_INFORMATION
20	$ATRIBUTE_LIST
30	$FILE_NAME
50	$SECURITY_DESCRIPTOR
60	$VOLUME_NAME
70	$VOLUME_INFORMATION
80	$DATA
90	$INDEX_ROOT
A0	$INDEX_ALLOCATION
B0	$BITMAP
C0	$SYMBOLIC_LINC

Windows NT использует предопределенные атрибуты, список которых содержится в таблице 8.

Атрибут $ATTRIBUTE_LIST. Атрибут **$ATTRIBUTE_LIST** используется, если файл нуждается в дополнительных записях в MFT (рис.3). Его поток является списком записей, каждая из которых описывает файловый атрибут. Запись имеет формат, показанный в таблице 9.

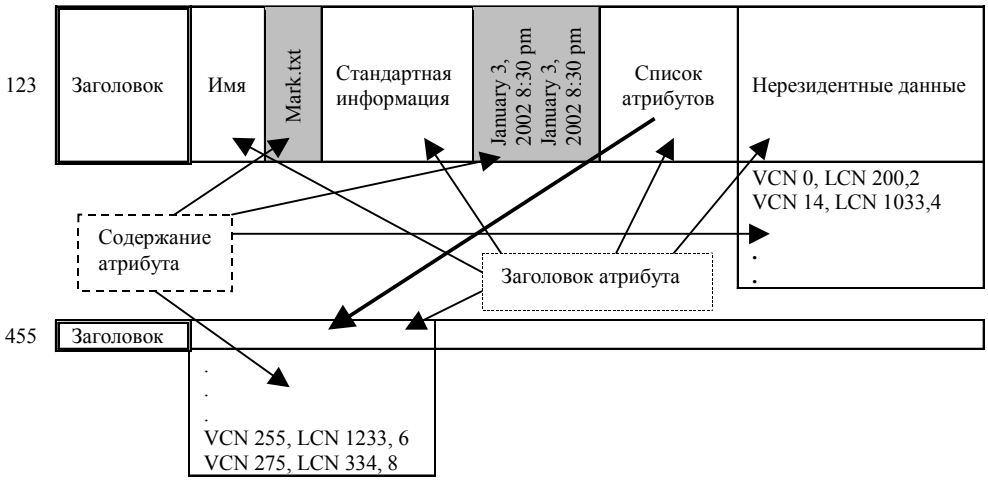

Рис.3. Основная и дополнительная файловые записи в MFT

Формат записи потока атрибута **$ATTRIBUTE_LIST** Таблица 9

Смещение от начала потока	Длина	Описание
0	4	Тип
4	2	Длина записи
6	1	L=Длина имени
8	8	Начальный VCN. 0 для резидентного атрибута, копия поля со смещением 16 из заголовка нерезидентного атрибута.
16	8	Файловая ссылка на файловую запись содержащую атрибут.
24	2	Идентификатор. Содержит копию поля со смещением 15 из заголовка.
26	2*L	Имя в Unicode

$STANDART_INFORMATION. Атрибут $STANDART_INFORMATION всегда является резидентным. В нем хранится информация о стандартных свойствах файла (MS DOS атрибуты файла) (таблица 10).

Список значений DOS-атрибутов файла расширен (таблица 11).

Формат потока атрибута $STANDART_INFORMATION Таблица 10

Смещения от начала потока	Длина	Описание
0	8	Время создания файла
8	8	Время последней модификации файла
16	8	Время последней модификации записи файла
24	8	Время последнего доступа
32	4	DOS-атрибуты файла
36	12	Всегда 0, не используется

DOS-атрибуты файла Таблица 11

Бит	Смысл
08 00	Сжатый
04 00	Символическая ссылка
00 20	Архивный
00 04	Системный
00 02	Скрытый
00 01	Только для чтения

$FILE_NAME. Атрибут $FILE_NAME также всегда является резидентным. Элементы потока атрибута приводятся в таблице 12.

Формат потока атрибута $FILE_NAME Таблица 12

Смещение	Длина	Описание
0	8	Файловая ссылка на базовую файловую запись директории, содержащей это имя.
8	32	Время модификации файла
40	8	Размер места, выделенного под файл.
48	8	Реальный размер файла
56	8	Флаг. Таблица 11.
64	1	L=Длина имени (в символах).
65	1	Тип имени (таблица 13).
66	2*L	Имя файла в Unicode. Максимальная длина имени 255 символов.

Типы имен в NTFS Таблица 13

Значение	Смысл
0	PPSIX тип: любые Unicode символы кроме 0 (нуля) и '/'. Имя зависит от регистра.
1	Win32 тип: символы "" '*' '/' ':' '<' '>' '?' '\' '|' запрещены. Имя не может закончится '.' или ' ' и не зависит от регистра.
2	DOS тип: Допускается только 8 битные символы отличные от "" '*' '+' ',' ':' ';' '<' '=' '>' '?' '\'. Имя должно записываться в формате: от 1 до 8 символов, разделитель '.', от 1 до 3 символов. Большой регистр.
3	Win32 и DOS типы: Обеспечивается Win32 имя уже принадлежащее к DOS типу.

Атрибут $SECURITY_DESCRIPTOR позволяет санкционировать доступ к файлу.

Поток атрибута $DATA содержит данные файла.

Поток атрибута $VOLUME_NAME содержит в Unicode имя тома.

Поток атрибута $VOLUME_INFORMATION содержит элементы, представленные в таблице 14.

Некоторые элементы потока $VOLUME_INFORMATION Таблица 14

Смещение	Длина	Описание
8	1	Major номер версии.
9	1	Minor номер версии.
10	1	Флаг проверки. Если установлен в 1, то Windows NT должна запускать `chkdsk для проверки диска.`

$BITMAP. Атрибут **$BITMAP** применяется для определения статуса кластеров раздела и для индексации. Поток атрибута **$BITMAP** содержит последовательность бит, каждый установленный в 1 бит, которой свидетельствует об использовании соответствующего кластера.

Атрибут $SYMBOLIC_LINC может содержать символьную ссылку.

Директория. С точки зрения пользователя, директория — это файл, хранящий внутри себя другие файлы.

Файловая запись для директории не имеет атрибута данных, вместо этого она имеет три атрибута: корень индекса $INDEX_ROOT, размещение индекса $INDEX_ALLOCATION и битовую карту (bit map).

Потоки первых двух атрибутов содержат индекс (список) имен файлов, или более точно, последовательность индексных элементов, хранящих атрибуты имени. Индексный элемент создается для атрибута имени каждого файла, расположенного в директории. Этот вид индексных элементов позволяет сортировать атрибуты имени файлов в лексикографическом порядке, заданном в $UpCase. Индексный элемент может указывать на подузлы, содержащие индексные элементы нижнего уровня.

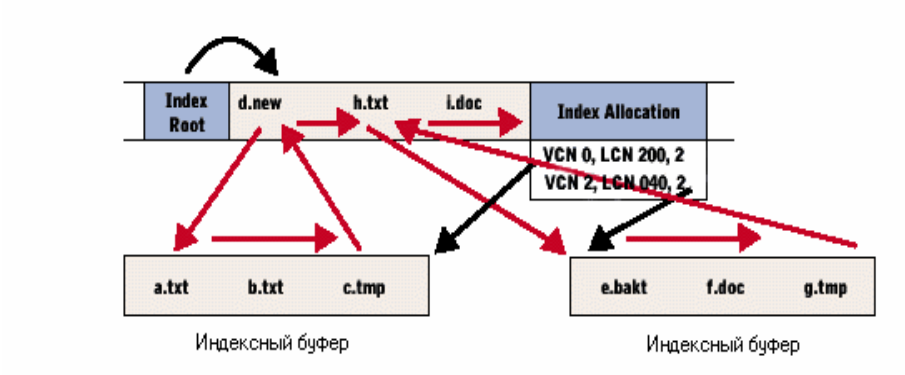

Рис.4. Реализация B+ дерева

Поток атрибута $INDEX_ROOT Таблица 15

Смещение	Длина	Описание
0	4	Тип индексируемого атрибута. Возможные значения соответствуют типам атрибутов, представленных в Таблице 9.
4	4	Всегда 00 00 00 01.
8	4	Размер индексных буферов.
12	4	Количество кластеров под индексный буфер.
16	4	Всегда 00 00 00 10.
20	4	Размер последовательности индексных элементов + 10.
24	4	
28	4	Флаг. Значение 0 соответствует ситуации, когда индекс полностью хранится в корне. Значение 1 устанавливается, если для хранения индекса требуется дополнительное место.
32		Последовательность индексных элементов.

Индекс имен хранится в виде B+ дерева (рис. 4), что минимизирует количество обращений к диску при поиске данного файла, особенно в больших каталогах. Первый уровень дерева сохраняется в потоке атрибута $INDEX_ROOT. Остальная часть дерева хранится в индексных буферах, содержащихся в потоке атрибута $INDEX_ALLOCATION.

$INDEX_ROOT. Поток $INDEX_ROOT имеет формат, представленный в таблице 15. Последовательность индексных элементов оканчивается индексным элементом, у которого установлен флаг последнего элемента. Элемент последовательности имеет формат, показанный в таблице 16.

Поток атрибута $INDEX_ALLOCATION является простой последовательностью индексных буферов. Индексный буфер состоит из двух частей: заголовка и содержания. Заголовок имеет формат, показанный в таблице 17.

В содержании хранится последовательность индексных буферов, подобная той, что хранится в атрибуте $INDEX_ROOT.

Индексный элемент Таблица 16

Смещение	Длина	Описание
Следующие поля используются только, когда не установлен флаг последнего элемента. Поле со смещением 12.		
0	8	Файловая ссылка.
8	2	L = длина индексного элемента.
10	2	M = длина потока. Копия поля длины из заголовка индексируемого атрибута.
12	1	Флаги. 01 - индексный элемент указывает на подузел; 02 – последний индексный элемент.
Следующее поле существует только, когда не установлен флаг последнего элемента.		
16	M	Поток. Копия потока индексируемого атрибута. Например, для директории в этом поле содержится поток атрибута имени файла.
Следующее поле существует только, когда установлен флаг подузла.		
L - 8	8	VCN подузла в атрибуте $INDEX_ALLOCATION

В битовой карте для индексирования каждый бит представляет VCN атрибута $INDEX_ALLOCATION, начиная с VCN 0. Бит установлен, если VCN используется индексным буфером.

Как уже было сказано, таблица определения атрибутов задает типы атрибутов, поддерживаемые на томе.

Поток данных файла $AttrDef представляет собой последовательность записей, каждая из которых определяет один файловый атрибут. Формат записи приведен в таблице 18.

Заголовок индексного буфера Таблица 17

Смещение	Длина	Описание
0	4	Идентификатор индексного буфера. Содержит ' INDX '.
4	2	Смещение поля «Счетчик обновления последовательности» относительно начала заголовка.
6	2	Количество элементов массива, содержащего значения обновления последовательности. S +1
16	8	VCN индексного буфера в атрибуте $INDEX_ALLOCATION
24	2	Смещение начала последовательности индексных элементов – 18
28	4	Смещение конца последовательности индексных элементов - 18
32	4	Смещение конца индексного буфера - 18
36	4	1, если узел не является последним (листом)
40	2	Поле «Счетчик обновления последовательности»
42	2*(S-1)	Массива обновления последовательности

Формат записи файлового атрибута Таблица 18

Смещение	Длина	Описание
0	128	Метка в Unicode
128	8	Тип
136	8	Флаги. Таблица 19.
144	8	Минимально выделяемый размер
152	8	Максимально выделяемый размер

Флаги записи файлового атрибута Таблица 19

Значение	**Смысл**
00000001	Может быть индексирован.
00000040	Нуждается в восстановлении во время фазы регенерации.
00000080	Может быть не резидентным.

Доступ к структуре NTFS

Файловая система NTFS может использоваться только операционными системами Windows NT, Windows 2000, Windows XP, в которых для доступа к секторам диска можно воспользоваться универсальными функциями CreateFile, ReadFile и WriteFile.

// Пример чтения имен файлов из MFT

```
#include <windows.h>
#include <stdio.h>
#include <winioctl.h>          // Windows NT IOCTL коды
#include <malloc.h>
```

```
#pragma pack(1)
typedef   struct   tagBPB_NTFS
{
     WORD  wBytesPerSec;     // Байтов в секторе
     BYTE  bSecPerClust;     // Секторов в кластере
     WORD  wResSectors;      // Зарезервированных секторов
     BYTE  bFATs;            // Не используется
     WORD  wRootDirEnts;     // Не используется
     WORD  wSectors;         // Не используется
     BYTE  bMedia;           // Описатель среды
     WORD  wFATsecs;         // Не используется
     WORD  wSecPerTrack;     // Количество секторов на дорожке
     WORD  wHeads;           // Количество головок
     LONG dwHiddenSect;      // Количество скрытых секторов
     WORD dwBigTotalSectors; // Не используется
     WORD dwBigTotalSectorsHigh; // Не используется
     WORD dwBigSectorsPerFat;  // Не используется
     WORD dwBigSectorsPerFatHigh; // Не используется
     INT64 TotalSectors;  //  Секторов на томе
     INT64 FirstMFT;  //  Сектор начала MFT
     INT64 MirrMFT;  //  Сектор копии MFT
     LONG ClustersPerFileRecord;   // Число кластеров под файловую запись
     LONG ClustersPerIndexBlock;  // Число кластеров под индексный блок
     INT64 VolumeSerialNumber;  // Серийный номер тома
     LONG Checksum;   //  Проверочная сумма
} BPB_NTFS, FAR * LPBPB_NTFS;

// Заголовок файловой записи
struct FileRecordHeader
{
     char ID[4];  // Идентификатор файловой записи
     WORD UpDateSequenceOffset; // Смещение счетчика обновления последовательности
     WORD CountItemUpDateSequence; // Количество элементов массива, содержащего значения
                                   // обновления последовательности
     BYTE Reserve[8]; // Зарезервировано
     WORD NumSecquence;  // Номер последовательности
     WORD HardLinkCount;  // Счетчик жестких ссылок
     WORD AttribBegOffset;  // Смещение второй части файловой записи, последовательности
                            // атрибутов файла
     WORD Flag;             // Флаг
     DWORD RealSize;  // Реальный размер файловой записи
     DWORD ByteAllocate;  // Количество байт, выделенных под файловую запись
     INT64 FileReference;  // Ссылка на основную запись
     WORD AttribMaxID;  // Идентификатор атрибута, имеющий максимальное значение,
                        // увеличенный на 1
     WORD UpDateSequenceNum;  // Массива обновления последовательности
};

struct StandartAttribHeader  // Заголовок атрибута
{
     DWORD Type;  // Тип атрибута
     DWORD Length;  // Длина атрибута
     BYTE FlagRez;  // Флаг резидентности
     BYTE LengthName;  // Длина имени
     WORD ContentOffset;  // Смещение потока
     WORD Flag;  // Флаг компрессии
```

```
    WORD ID;  // Идентификатор
};

struct FileNameAttrCont  // Содержание атрибута имени
{
        INT64 FileReference;  //  Файловая ссылка на директорию
        BYTE Time[32];  //  Время
        INT64 ByteAllocate;  //  Распределено байт
        INT64 RealSize;       //  Реальный размер
        INT64 Flag;  //  флаг
        BYTE LengthName;  //  Длина имени
        BYTE TypeName;  //  Тип имени
        char FileName[30];   //  Имя
};
#pragma pack()

byte buf[512];
byte buf_FileRec[1024];

int main(int argc, char **argv)
{
    HANDLE h;

    // открываем логический диск
    h = CreateFile("\\\\.\\C:", GENERIC_READ | GENERIC_WRITE,
      FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);

    // Проверяем успешно ли закончилась попытка открытия логического диска
    if (h != INVALID_HANDLE_VALUE)
    {
        DWORD dwBytesRead;  DWORD dwBytesWrite;  LONG BeginMFTSector;

        // Читаем первый сектор раздела
        ReadFile(h, buf, 512, &dwBytesRead, NULL);
        // Извлекаем из считанного сектора блок параметров BIOS
        BPB_NTFS sbpbntfs;
        memcpy(&sbpbntfs, buf + 11, 73);

        // Определяем сектор начала MFT
        BeginMFTSector = sbpbntfs.bSecPerClust * sbpbntfs.FirstMFT;

        // Позиционируемся на начало MFT
        DWORD dwPtrLow = SetFilePointer (h, 512*(BeginMFTSector+32), NULL, FILE_BEGIN);

        // Читаем блок из MFT
        ReadFile(h, buf_FileRec, 1024, &dwBytesRead, NULL);

        // Извлекаем из считанного заголовок файловой записи
        FileRecordHeader  fHeader;
        memcpy(&fHeader, buf_FileRec, sizeof(fHeader));

        DWORD OffsetFileRec = 0;
        while(strncmp(fHeader.ID, "FILE",4) == 0)
        {
            // Читаем из MFT файловую запись
            ReadFile(h, buf_FileRec, 1024, &dwBytesRead, NULL);
```

```
            // Извлекаем из считанного заголовок файловой записи
            FileRecordHeader fHeader;
            memcpy(&fHeader, buf_FileRec, sizeof(fHeader));

            // Извлекаем из считанного заголовок атрибута
            StandartAttribHeader stattrHeader;
            WORD attrOffset;
            attrOffset = fHeader.AttribBegOffset;
            memcpy(&stattrHeader, buf_FileRec + attrOffset, sizeof(StandartAttribHeader));

            // Пока не прочли все атрибуты
            while(stattrHeader.ID != fHeader.AttribMaxID - 1)
            {
                // Извлекаем из считанного заголовок очередного атрибута
                attrOffset = attrOffset + stattrHeader.Length;
                memcpy(&stattrHeader, buf_FileRec + attrOffset, sizeof(StandartAttribHeader));

                // Если последний атрибут, выходим из цикла
                if(stattrHeader.Type == 0xffffffff) break;

                switch(stattrHeader.Type)
                {
                    case 0x30:  // Если тип атрибута имя
                    {
                        FileNameAttrCont fNameAttr;

                        // Извлекаем из считаного содержание атрибута имя
                        memcpy(&fNameAttr , buf_FileRec + attrOffset + 24 +
                        stattrHeader.ContentOffset,  sizeof(FileNameAttrCont ));

                        //  Если длина имени  больше нуля, выводим имя на экран
                        if(fNameAttr. LengthName > 0)
                        {
                            char MultiByteStr[30]; int dj = 0;
                            for(int di = 0; di < fNameAttr.LengthName * 2; di = di+2)
                            {
                                MultiByteStr[dj] = fNameAttr.FileName[di]; dj++;
                            }
                            MultiByteStr[dj] = '\0';
                            printf("%s\n",MultiByteStr);
                        }
                        break;
                    }
                }
            }
            // Определяем смещение следующей файловой записи
            OffsetFileRec = OffsetFileRec + 1024;

            // Позиционируемся на файловую запись
            dwPtrLow = SetFilePointer (h, 512*(BeginMFTSector+32) + OffsetFileRec, NULL,
            FILE_BEGIN);
        }
        // Закрываем дескриптор
        CloseHandle(h);
}
printf("The End");
return 0;
```

}

Кроме того, в этих операционных системах можно напрямую обратиться к драйверу дискового устройства с помощью функции управления вводом/выводом, которая называется DeviceIoControl.

```
BOOL DeviceIoControl(
  HANDLE  hDevice,        // идентификатор устройства
  DWORD   dwIoControlCode, // код выполняемой операции
  LPVOID  lpInBuffer,     // буфер для входных данных
  DWORD   nInBufferSize,  // размер буфера lpInBuffer
  LPVOID  lpOutBuffer,    // буфер для выходных данных
  DWORD   nOutBufferSize, // размер буфера lpOutBuffer
  LPDWORD lpBytesReturned, // указатель на счетчик выведенных байт
  LPOVERLAPPED  lpOverlapped); // указатель на структуру OVERLAPPED.
```

Коды операций функции DeviceIoControl Таблица 20

Код операции	Описание
FSCTL_DISMOUNT_VOLUME	Размонтирование тома
FSCTL_GET_COMPRESSION	Определение состояния компрессии для каталога или файла
FSCTL_LOCK_VOLUME	Блокирование тома
FSCTL_SET_COMPRESSION	Установка состояния компрессии для каталога или файла
FSCTL_UNLOCK_VOLUME	Разблокирование тома
IOCTL_DISK_CHECK_VERIFY	Проверка замены носителя данных для устройства со сменным носителем
IOCTL_DISK_EJECT_MEDIA	Извлечение носителя данных из устройства с интерфейсом SCSI
IOCTL_DISK_FORMAT_TRACKS	Форматирование нескольких дорожек диска
IOCTL_DISK_GET_DRIVE_GEOMETRY	Получение информации о физической геометрии диска
IOCTL_DISK_GET_DRIVE_LAYOUT	Получение информации о всех разделах диска
IOCTL_DISK_GET_MEDIA_TYPES	Получение информации о среде, которую можно использовать для хранения данных в устройстве
IOCTL_DISK_GET_PARTITION_INFO	Получение информации о разделе диска
IOCTL_DISK_LOAD_MEDIA	Загрузка носителя данных в устройство
IOCTL_DISK_MEDIA_REMOVAL	Включение или отключение механизма извлечения носителя данных
IOCTL_DISK_PERFORMANCE	Получение информации о производительности устройства
IOCTL_DISK_REASSIGN_BLOCKS	Перевод блоков диска в область резервных блоков
IOCTL_DISK_SET_DRIVE_LAYOUT	Создание разделов на диске
IOCTL_DISK_SET_PARTITION_INFO	Установка типа разделов диска
IOCTL_DISK_VERIFY	Выполнение логического форматирования
IOCTL_SERIAL_LSRMST_INSERT	Разрешение или запрещение добавления информации о состоянии линии и модема в поток передаваемых данных

Через параметр hDevice вы должны передать идентификатор устройства, полученный от функции CreateFile. Для того, чтобы воспользоваться этой функцией для открывания устройства, вы должны указать имя устройства следующим образом (пример приведен для диска C:).

```
hDevice = CreateFile("\\\\.\\C:", 0, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
```

С помощью параметра dwIoControlCode можно задать один из кодов операции, указанных в таблице 20.

Через параметр lpInBuffer вы должны передать функции DeviceIoControl адрес управляющего блока, необходимого для выполнения операции. Формат этого блока зависит от кода выполняемой операции.

В буфер, адрес которого передается через параметр lpOutBuffer, будет записан результат выполнения операции. Формат этого буфера также зависит от кода операции.

При необходимости с помощью функции DeviceIoControl вы можете выполнять асинхронные операции, подготовив структуру типа OVERLAPPED и передав ее адрес через параметр lpOverlapped.

```
#include <windows.h>
#include <stdio.h>
#include <winioctl.h>                // Windows NT IOCTL коды

//  Регистры
typedef   struct   _DIOC_REGISTERS {
    DWORD reg_EBX;
    DWORD reg_EDX;
    DWORD reg_ECX;
    DWORD reg_EAX;
    DWORD reg_EDI;
    DWORD reg_ESI;
    DWORD reg_Flags;
  }DIOC_REGISTERS, *PDIOC_REGISTERS;

  // Intel x86 флаг состояния процессора
  #define CARRY_FLAG  0x1

//  Функция получения информации об устройстве
DWORD GetDriveFormFactor(int iDrive)
{
    HANDLE h;
    TCHAR tsz[8];
    DWORD dwRc;
    // Преобразуем введенный параметр к форматированной строке
    wsprintf(tsz, TEXT("\\\\.\\%c:"), TEXT('@') + iDrive);

    // Открываем требуемое устройство
    h = CreateFile(tsz, 0, FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0);
    // Если открытие успешно
    if (h != INVALID_HANDLE_VALUE)
    {
        DISK_GEOMETRY Geom[20];
        DWORD cb;
        // Если вызов функции с кодом, позволяющим получить информацию о среде, которую
                //можно использовать для хранения данных в устройстве, прошел успешно
        if (DeviceIoControl (h, IOCTL_DISK_GET_MEDIA_TYPES, 0, 0,
                    Geom, sizeof(Geom), &cb, 0)  && cb > 0)
        {
            // Можем выполнить требуемые действия с определенным носителем
            switch (Geom[0].MediaType)
            {
                case F5_1Pt2_512: // 5.25 1.2MB floppy
                case F5_360_512:  // 5.25 360K  floppy
                case F5_320_512:  // 5.25 320K  floppy
                case F5_320_1024: // 5.25 320K  floppy
                case F5_180_512:  // 5.25 180K  floppy
                case F5_160_512:  // 5.25 160K  floppy
```

```
                    dwRc = 525;
                    break;

                    case F3_1Pt44_512: // 3.5 1.44MB floppy
                    case F3_2Pt88_512: // 3.5 2.88MB floppy
                    case F3_20Pt8_512: // 3.5 20.8MB floppy
                    case F3_720_512:   // 3.5 720K   floppy
                    dwRc = 350;
                    break;

                    case RemovableMedia:
                    dwRc = 2;
                    break;

                    case FixedMedia:
                    dwRc = 1;
                    break;

                    default:
                    dwRc = 0;
                    break;
                }
            }
            else // Неудачное выполнение функции
                dwRc = 0;
            // Закрываем устройство
            CloseHandle(h);
        }
        else // Неудачное открытие устройство
            dwRc = 0;
        // Возвращаем полученное значение
        return dwRc;
    }

    int main(int argc, char **argv)
    {
        int iDrive;
        for (iDrive = 1; iDrive <= 26; ++iDrive)
        {
            DWORD dwFF = GetDriveFormFactor(iDrive);

            if (dwFF)
            printf("%c: = %d\n", TEXT('@') + iDrive, dwFF);
        }
        scanf("%c: = %d\n", TEXT('@') + iDrive);
        return 0;
    }
```

Заключение

NTFS на сегодняшний день используется наиболее распространенными на персональных компьютерах операционных системах. NTFS обеспечивает комбинацию эффективности, надежности и совместимости, отсутствующую у FAT. Она разработана для быстрого выполнения стандартных файловых операций типа чтения, записи и поиска, а также улучшенных операций типа восстановления файловой системы на очень больших жестких дисках.

Библиографический список

1. Барри Саймон. Файловая система FAT32 для Windows 95. PC Magazine, Апрель 8, 1997, стр.279.
2. Хелен Кастер. Основы Windows NT и NTFS/Пер. с англ. — М: Издательский отдел “Русская редакция” ТОО “Channel Trading Ltd.”, 1996.—440 с.
3. Ресурсы Windows NT: пер. с англ. — СПб.: BHV — Санкт-Петербург, 1996. — 720 с.
4. Фролов А.В., Фролов Г.В. Программирование для Windows NT. — М.: ДИАЛОГ-МИФИ, 1996. – 272 с. Т. 26.
5. Фролов А.В., Фролов Г.В. Программирование для Windows NT. — М.: ДИАЛОГ-МИФИ, 1997. – 271 с. Т. 27.
6. Нортон П. Персональный компьютер фирмы IBM и операционная система MS-DOS: Пер. с англ. — М.: Радио и связь, 1991. – 416 с.