Тема № 6: «Файловая система NTFS. Часть I.»

ОГЛАВЛЕНИЕ

I Категория данных содержимого 5

I.I Битовые поля 5

I.2 B-дерево 6

I.3 Целочисленный тип от Microsoft 9

I.4 Маркеры целостности секторов I0

I.5 Структура записей главной файловой таблицы II

I.6 Атрибуты I6

I.6.I Общее описание I6

I.6.2 Нерезидентные атрибуты I9

I.6.3 Типы атрибутов 23

Файловая система NTFS (англ. New Technology File System) была впервые представленна компанией Microsoft в 1993 году[1]. В настоящее время Micrososft определяет ее как основную файловую систему в последних версиях Windows и Windows Server, предоставляющую полный набор возможностей, включая дескрипторы безопасности, шифрование, дисковые квоты и расширенные метаданные[2].

NTFS разработана на основе файловой системы HPFS (от англ. High Performance File System -- высокопроизводительная файловая система), создававшейся Microsoft совместно с IBM для операционной системы OS/2. Но, получив такие несомненно полезные новшества, как квотирование, журналируемость, разграничение доступа и аудит, в значительной степени утратила присущую прародительнице (HPFS) весьма высокую производительность файловых операций[3].

С течением времени, файловоая система притерпивала ряд изменений, применяющихся в новых версиях ОС.

— 1.2 - Windows NT;

— 3.0 - Windows 2000;

— 3.1 - Windows XP и выше.

Целями создания файловой системы стали:

1. Надежность — свойство файловой системы сохранять во времени способность выполнять требуемые функции в заданных режимах и условиях применения,технического обслуживания, хранения и транспортирования[4]. Оно достигается за счет использования:

   — журналирования;

   — контрольных точек / транзакций.

[1]Разработка системы началась в 1991 году Брайном Андрю, Девидой Гейблом, Гари Кимурой и Томом Миллером.

[2]https://docs.microsoft.com/ru-ru/windows-server/storage/file-server/ntfs-overview

[3]https://ru.wikipedia.org/wiki/NTFS

[4]ГОСТ 27.002-2015 Надежность в технике. Термины и определения

2. Безопасность — свойство, позволяющее при выполнении всех условий, определенных в описании файловой системы, обеспечить конфиденциальность информации, выявить факт ее искажения, а также определить авторов, вносивших изменения:

   — журналирования;

   — списки управления доступом (ACL);

   — шифрование.

3. Поддержка томов большого объема.

Не сложно заметить, что свойство «надежность» с некоторым допущением может считаться свойством, обеспечивающим доступность информации[5].

Основные принципы файловой системы NTFS:

— все данные (системные и пользовательский) хранятся в виде файлов;

— все данные инкапсулируются[6] в объекты фиксированной структуры.

Как и в случае с FAT, для изложения материала будет рассматриваться образ носителя информации. Для его создания можно воспользоваться одним из двух способов:

I. Использовать штатные средства ОС Windows для форматирования какого-нибудь носителя малого размера.

2. Создать образ носителя альтернативными средствами.

Использование первого подхода позволяет получить наиболее актуальную версию образа, однако требует осуществления ряда манипуляций. Дело в том, что ОС Windows может не позволить осуществить форматирования flash носителя («флешки») малого размера под файловую систему NTFS. Это связано с тем, что «флешку» как правило извлекают из компьютера как только исчезнет окно

[5]Доступность информации не может быть обеспечена полностью. Например, при физическом уничтожении цифрового носителя, механизмы файловой системы будут бессильны.

[6]упаковываются

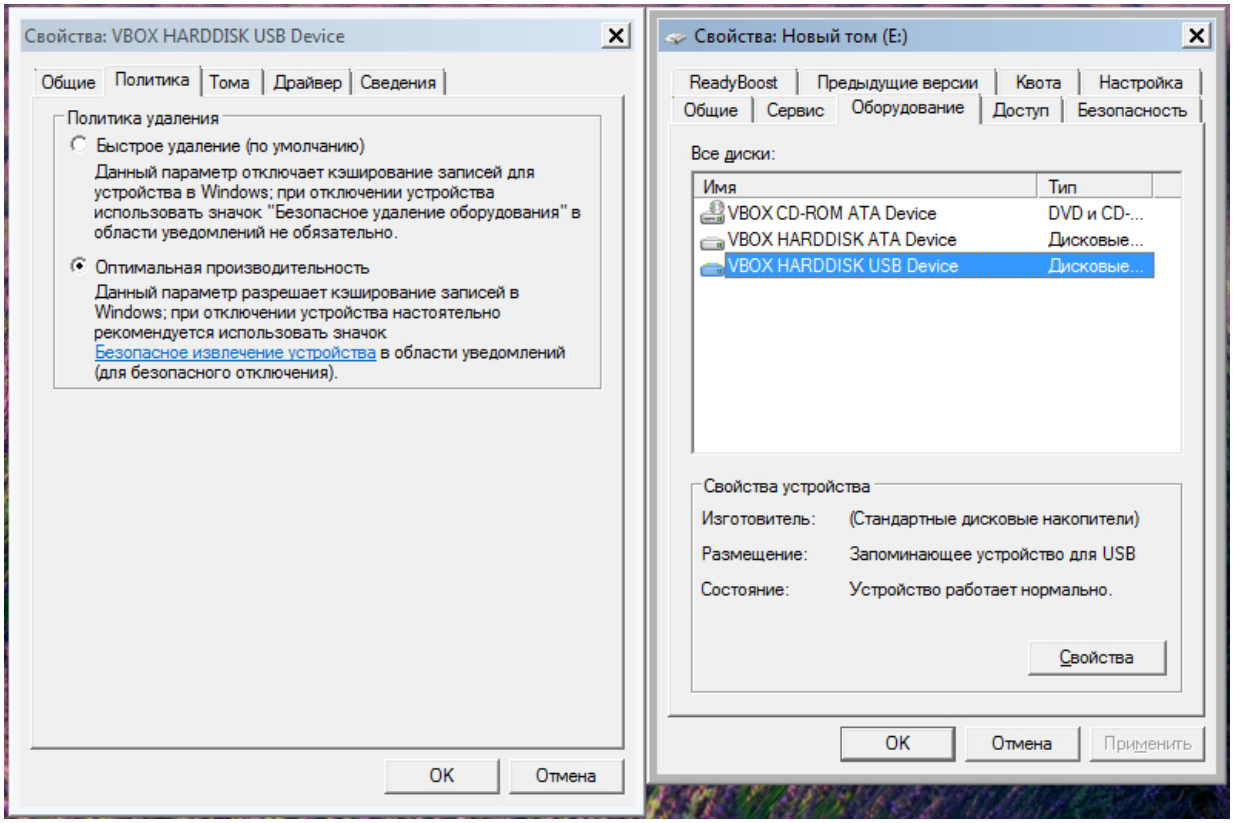

Рис I. Настройка дискового носителя для форматирования под файловую систему NTFS.

копирования, либо как только перестанет моргать лампочка на ней. По этой причине Windows рассматривает «флашки» как носители, оптимизированные для быстрого удаления, и форматирует их только под файловую систему FAT.

Для того, чтобы Windows позволила произвести форматирования под файловую систему NTFS, в свойствах запоминающего устройства следует изменить «политику удаления» на «Оптимальная производительность»(рис. I).

В качестве альтернативного способа создания образа можно предложить утилиту mkfs.ntfs, доступную в ОС Linux[7]. В отличии от mkfs.fat, указанная утилита не позволяет создавать сразу образ, а форматирует уже существующей.

Для создания образа, с использованием данной утилиты следует выполнить следующие команды.

```
dd if=/dev/zero of=image.img bs=1M count=32
```

[7]Утилита входит в пакет ntfs-3g

```
mkfs.ntfs -F image.img
```

Поле создания образа, можно приступать к рассмотрению самой файловой системы.

I. Категория данных содержимого

I.I. Битовые поля

Как уже было сказано, файловая система NTFS рассчитана на тома больших объемов. Также она позволяет сохранять файлы как больших, так и малых размеров. Все это приводит к необходимости выделения и освобождения ячеек памяти различного типа и размера.

В случае, когда ячейка занята и ее необходимо освободить, то процесс поиска этой ячейки в пространстве тома не вызывает затруднений. Все становится сложнее, когда требует выделить новую ячейку. В этом случае требуется вначале найти свободную. а уже после этого занять ее.

Для файловой системы FAT допускались два варианта поиска свободных ячеек:

— перебор;

— сохранение адреса последней ячейки в FSInfo.

Первый вариант был долгим, а второй приводил к тому, что данные на носителе размещались все дальше от начала носителя. Кроме того, FSInfo предполагает фиксированное число контролируемых ячеек. Как будет видно далее из лекции, в файловой системе NTFS задача поиска свободных ячеек будет возникать для объектов, содержащих переменное число ячеек.

Таким образом, в файловой системы NTFS для поиска свободных ячеек было принято решение использовать перебор. Для ускорение данного процесса и решили использовать «битовые поля».

Идея битовых полей заключается в том, чтобы для некоторого объекта, который разделен на ячейки, выделить область памяти, размер которой в битах не меньше числа этих ячеек. В случае, когда ячейка с некоторым индексом i является свободной и может быть выделена для

хранения данных, i-й бит этой области устанавливается в 0. Таким образом, для поиска свободной ячейки требуется найти в выделенной области памяти нулевой бит. Номер этого бита будет соответствовать номеру свободной ячейки.

В качестве поясняющего примера можно привести вахту в РТУ, где выдаются ключи. Если ключ от аудитории лежит в ячейке, то аудитория свободна, иначе – нет. Таким образом можно найти свободную аудиторию без необходимости проверки каждой аудитории. Безусловно, если эта ситуация предполагает, что от аудитории существует единственный ключ.

I.2. B-дерево

При реализации хранилища информации необходима реализация операции добавления объекта в хранилище, его удаление и поиск (обращение к нему). В процессе функционирования большинство хранилищ предполагает, что превалирующее число обращений к нему будет связано именно с поиском. По этой причине для хранения данных используются подходы максимизацию скорости поиска за счет снижения скорости добавления и удаления.

Рассмотрим пример. В пустое хранилище целых чисел в произвольном порядке добавляются значения от 0 до 10. Наиболее тривиальным подходом для хранения этих чисел является массив. При этом, для обеспечения высокой скорости этот массив должен быть упорядочен. Таким образом при добавлении элемента в хранилище закладываются накладные расходы на упорядочивание массива.

Для поиска элемента может использоваться бинарный поиск, принцип которого «разделяй и властвуй»[8]. На рис. 2 приведен пример поиска числа 6 в упорядоченном массиве.

При очевидной простоте реализации, использования массивов имеет достаточно жесткие ограничения в том случае, когда речь заходит о практической реализации. Основное ограничение

[8]Искомое значение сравнивается с центральным элементом массива и в зависимости от результата аналогичный процес повторяется либо в левой, либо в правой части массива. Таким образов на каждом шаге исходный массив уменьшается вдвое.

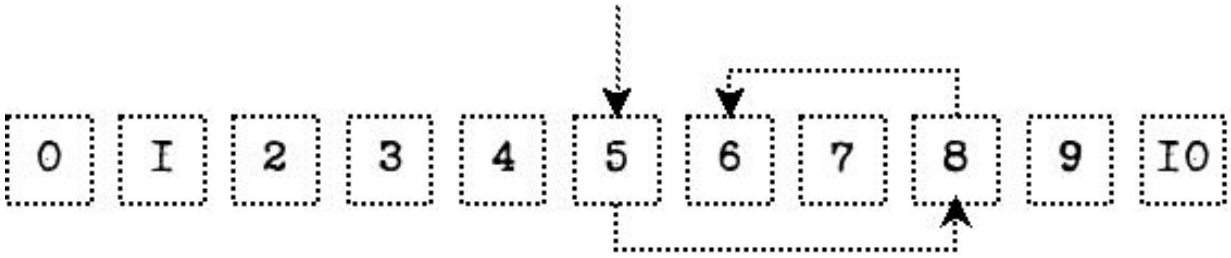

Рис 2. Процесс бинарного поиска числа «6» в упорядоченном массиве.

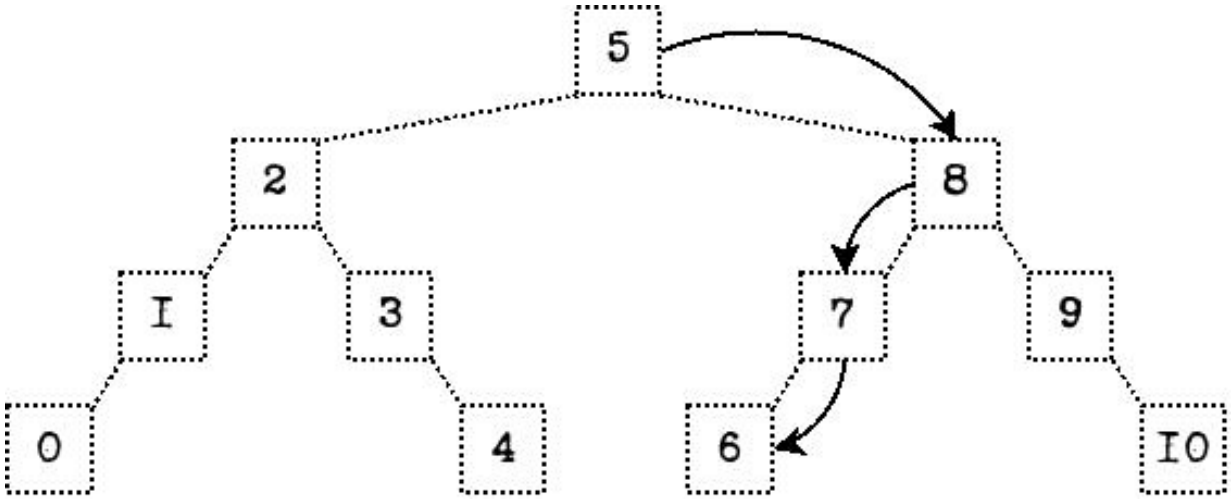

Рис 3. Процесс поиска числа «6» в бинарном дереве.

заключается в необходимости выделения непрерывного блока памяти, в котором будут размещаться элементы массива. По этой причине, при реализации хранилищ, в которые планируется добавление большого числа элементов в процессе его использования, применяются структуры использующие для связи ссылки. Для рассмотренного выше примера наиболее близкой заменой являются бинарные деревья[9] (рис. 3).

И казалось бы все хорошо. Теоретики ликуют!!! Пишут программки, которые подтверждают эффективность при размещении в памяти элементов дерева... Но вот маленькая проблемка: как только бы начнем писать эти элементы на диск, то элемент меньше сектора (а чаще всего меньше кластера) мы не запишем, так как это минимальная единица данных, которой оперируют носители информации. Т.е. элементы придется группировать.... при этом таким образом, чтобы обеспечить наиболее эффективное считывание потом... и вот тут на сцену выходят B-деревья. Их идеология предполагает, что

[9]В бинарном дереве каждый узел имеет не более двух потомков.

в одной вершине могут быть собраны несколько элементов, при этом если вершина содержит n элементов, то она должна иметь n+I дочернюю вершину, либо не иметь их вовсе. Кроме того, B-деревья удовлетворяют следующим свойством.

I. В случае, если вершина имеет дочерние вершины, то их число не должно быть меньше некоторого параметра t, который также называется «минимальная степень вершины».

2. Минимальная степень корневой вершины — I.

3. Максимальное число дочерних вершины для всех вершин составляет $2 \cdot t$.

4. Все ключи должны быть упорядочены в рамках своего узла.

Кроме того, для B-дерева справедливо следующее. Если родительская вершина содержит ключи K_I, K_2 ($K_I < K_2$), то первая дочерняя вершина содержит ключи из множества $\{K : K < K_I\}$, вторая — из множества $\{K : K_I \leq K < K_2\}$, а третья — из множества $\{K : K_2 < K\}$. При этом ключи могут быть из любого множества, на котором задана операция сравнения.

Теперь перейдем от теории к практике..... от математики к программированию. Вершину B-дерева можно представить как структуру, имеющую 2 поля:

— статический массив размера $2 \cdot t - I$ для хранения ключей;

— статический массив размера $2 \cdot t$ для хранения указателей на дочерние вершины.

При этом не имеет значение сколько в вершине будет хранить ключей в каждый конкретный момент времени, память выделяется сразу по максимуму. При этом, создание каждой новой вершины происходит лишь в том случае, когда существует минимальное число ключей которое можно в ней разместить.

Таким образом все дерево представляет из себя набор упорядоченных массивов малого размера (относительно общего объема данных). В том случае, когда необходимо осуществить поиск

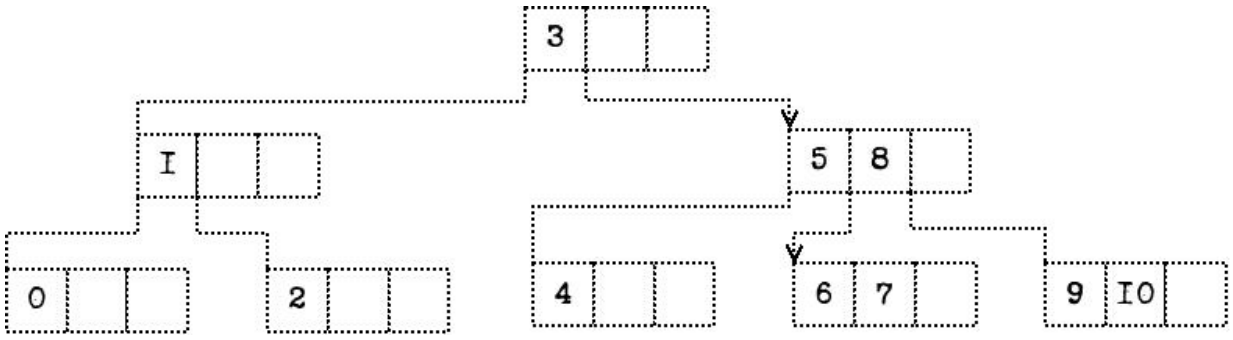

Рис 4. Процесс поиска числа «6» в B-дереве.

некоторого элемента, производится последовательно считывания таких массивов и определение следующего массива для поиска. На рис. 4 представлено B-дерево с «минимальной степенью вершины» t=2 и процесс поиска в нем.

I.3. Целочисленный тип от Microsoft

Чтение документации по технологиям компании Microsoft показывает, что большинство структур имеют кучу полей, которые либо зарезервированный для будущих поколений, либо должны иметь некоторое фиксированные значения. И вот в этом царстве расточительного использования памяти появляется странная «вещица»...

Компания Microsotf придумала способ позволяющий оптимизировать использование целочисленных значений, превращая знаковую переменную в беззнаковую по следующему правилу:

$$F(i)=\begin{cases} i, & \text{если } i \geq 0 \\ 2^i, & \text{если } i < 0 \end{cases}$$

Кроме оригинальности вычисления значений, меняется и смысл (размерность) этого значения. В случае положительного значения мы получаем число кластеров, а в случае отрицательного — вычисляем число байт. Т.е значение 5 — это 5 кластеров, а значение -2 — 4 байта..... Добро пожаловать в мир зазеркалья, где не все то, чем кажется))))

I.4. Маркеры целостности секторов

Одним из механизмов контроля целостности данных, записываемых на диск в NTFS, является механизм, основанный на использовании маркеров. Механизм применяется для контроля служебных объектов, размер которых кратен размеру сектора.

Тут следует сделать небольшое отступление. Рассматриваемый механизм в реальности не позволяет обнаружить искажения, которые произошли с данными в процессе хранения... точнее так в 510 из 512 случаем не позволяет. Реальная задача механизма — контроль фактической записи объекта на цифровой носителей в полном объеме.

Для достижения указанной цели в рамках объекта выделяется хранилище, называемое массивом последовательности обновлений (англ. – Update Sequence Array). В книге товарища Кэрри это массив также называется массивом маркеров и, учитывая те элементы, которые в нем хранятся, я считаю что это более корректное именование. Итак, число элементов массива на один превышает число секторов, занимаемых объектом.

Первым элементом массива является порядковый номер обновления (Update Sequence Number) — двухбайтовое значение, циклически увеличиваемое на единицу каждый раз, когда осуществляется запись объекта на носитель. Остальные элементы формируются из последних двух байт каждого сектора. Перед непосредственной записью сектора на цифровой носитель последние два байта сектора заносятся в соответствующую ячейку массива, а на их место записывается порядковый номер обновления (рис. 5).

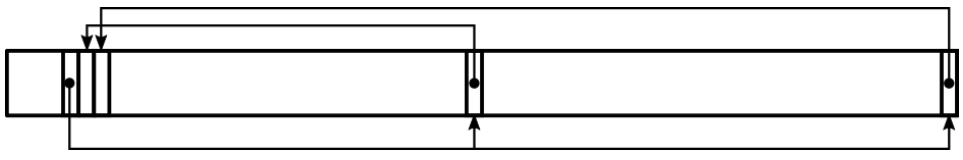

Рис 5. Процесс формирования массива маркеров.

При считывании объекта с носителя осуществляется проверка наличия в последний двух байтах корректного значения счетчика и если оно корректно, то из массива возвращаются на место все значения,

записанные туда ранее. Если же значения в хвостах секторов не соответствуют ожидаемым, то объект признается поврежденным.

I.5. Структура записей главной файловой таблицы

Все информация о файлах в рамках файловой системы NTFS сохраняется в специальной таблице, получившей названия Master File Table (далее — MFT).

Таблица MFT состоит из полей фиксированного размера, задаваемого в загрузочном секторе тома. Следует отметить, что как и в случае с размером сектора, размер записи зафиксирован и равен I024 байта. Сами записи не несут в себе информационной составляющей с точки зрения файловой системы, а используются для хранение более мелких объектов, называемых атрибутами. О них будет поведано в следующем разделе. Для описания одного файла может потребоваться хранение большого числа атрибутов, по этой причине для сохранения одного файла могут использоваться несколько записей таблицы.

Условно запись таблицы MFT может быть разделена на три части (рис. 6).

I. Заголовок.

2. Набор атрибутов. Заканчивается маркером конца записи — OxFFFFFFFF.

3. Свободное пространство.

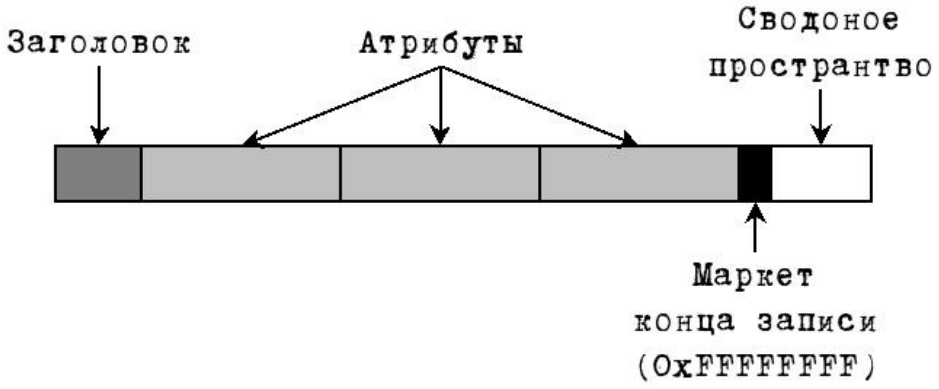

Рис 6. Общая структура записи таблицы MFT.

Ниже в таблице приведена структура заголовка записи таблицы MFT.

Смещение	Размер	Описание
0x00	4	Сигнатура
0x04	2	Смещение массива маркеров
0x06	2	Количество элементов массива маркеров
0x08	8	Номер записи в журнале транзакций (LSN)
0xI0	2	Порядковый номер
0xI2	2	Счетчик ссылок
0xI4	2	Смещение первого атрибута
0xI6	2	Флаги: 0xI - запись используется 0x2 - запись описывает каталог
0xI8	4	Используемый размер записи MFT
0xIC	4	Выделенный размер записи MFT
0x20	8	Адрес базовой записи MFT
0x28	2	Идентификатор для нового атрибута
0x2A	2	Зарезервировано
0x2C	4	Номер файловой записи

Последние две записи появились начиная в версии файловой системы 3.I.

Сигнатура записи определяет ее текущее состояние. Товарищь Кэрриэ выделяет два возможных значения этого поля, которые определяют является ли запись корректной, либо содержит ошибки. Однако, анализ исходных кодов драйвера ntfs ядра Linux[IO], а также исходные коды проекта ntfs-3g показали, что значения сигнатур немного больше. Напоминаю, что Microsoft запрещает исследования своих технологий, по этой причине мы опираемся на исследования,

[IO]Файл fs/ntfs/layout.h

проведенные другими.... И так, сигнатура записи может принимать одно из следующих значений.

Значения		Описание
HEX	ASCII	
0x454c4946	FILE	Обычная запись MFT
0x58444e49	INDX	Индексный массив
0x454c4f48	HOLE	Смысл поля чуть позже допишу....))))
0x444b4843	CHKD	Запись модифицирована утилитой chkdsk
0x444I4I42	BAAD	Запись битая

Смещение (задается от начала записи) и размер массива маркеров относятся к механизму, описанному в разделе I.4.

Поле порядковый номер[II] показывает какое количество раз данная запись выделялась для описание какого-либо файла. Понимаю что название не совсем адекватно отображает суть... но придумать иного не смог, а везде используется формулировка sequence number.((

Счетчик ссылок определяет число каталогов, содержащих записи для данной структура MFT. Он увеличивается на единицу для каждой жесткой ссылки, созданной для файл.

Используемый размер записи MFT определяет объем занимаемый данными записи. Следует отметить, что размер чаще всего делают кратным 8 байтам.

Выделенный размер записи MFT определяет полный размер записи. Данное поля одинаково для всех записей таблицы.

Тут следует отметить, что при уменьшении объема информации, размещаемой в файловой записи, «хвосты» ранее записанной туда информации никуда не деваются и могут быть без труда прочитаны. Это безусловно создает предпосылки для реализации угрозы конфиденциальности данных с одной стороны и помочь при проведении расследования инцидента с другой.

[II]Не путать с порядковым номером обновления маркеров.

Как говорилось ранее, для одного файла может использоваться несколько записей в таблице MFT. В этом случае выделяется основная (базовая) запись, на которую ссылаются все остальные записи. Для ссылки на запись как раз и используется поле адрес базовой записи MFT. Для базовой записи данное поле равно нулю.

Поскольку запись таблицы MFT может содержать несколько атрибутов, которые в том числе могут быть похожи как две капли воды, то не исключен вариант, при котором их можно будут просто перепутать. Для того, чтобы исключить это в рамках одной файловой записи все атрибуты получают уникальные идентификаторы. Поле идентификатор для нового атрибута содержит такой идентификатор для атрибута, который может быть понадобится добавить. Формирование идентификатора осуществляется итеративно. Это означает, что идентификатор следующего атрибута будет на единицу больше идентификатора последнего добавленного атрибута. Нумерация атрибутов начинается с нуля.

Номер файловой записи это порядковый номер (если угодно - индекс) файловой записи в таблице MFT.

На рис.7 приведен пример разбора первой записи таблицы MFT, размещенной на тестовом цифровом носителе. Следует отменить, что запись трижды была перезаписана, об этом свидетельствует значение в последних байтах секторов. При этом размер данных записи уменьшался, о чем свидетельствуют ненулевые значения в свободном пространстве записи.

Рис 7. Пример структуры файловой записи, размещённой на тестовом образе.

I.6. Атрибуты

I.6.I Общее описание

Атрибут это структура данных переменного размера, используемых для хранения данных, причем как служебных, так и непосредственно данных файлов. В некоторых источниках содержимое атрибута называют потокам (stream).

Атрибут условно разделяется на три составляющие: заголовок, имя и данные. При этом имя является опциональной составляющей атрибута. В разговорной речи имя атрибута ассоциируют с его данными (стримом). Если имя атрибута не задано, что говорят что это безымянный поток, в противном случае — именованный.

Из-за ограничений на максимальный размер данных, который позволяет хранить в себе атрибут (ограничения станут ясны чуть дальше), в записи MFT для хранения большего объема данных могут выделяться несколько атрибутов.

Кроме того, данные атрибутов могут храниться как вместе с заголовком, так и отдельно от него в области данных файловой системы. Атрибуты, у которых данные хранятся вместе с заголовком или данные которого физически отсутствуют на цифровом носителе (об этом позже), называются «резидентным», в противном случае — «нерезидентным».

Чтобы не путаться сильно, дальнейшее рамматрение продолжим относительно резидентных атрибутов. Нерезедентные атрибуты рассматрим отдельно, указав основные отличия от резитентных. Ниже приведена структура резидентного атрибута.

Смещение	Размер	Описание
0x00	4	Идентификатор типа атрибута
0x04	4	Длина атрибуты
0x08	I	Флаг нерезидентности (0 - атрибут резидентный)
0x09	I	Длина имени (0 - атрибут безымянный)
0x0A	2	Смещение имени

0x0C	2	Флаги
0x0E	2	Идентификатор атрибута
0x10	4	Длина содержимого
0x14	2	Смещение содержимого
0x16	1	Флаги резидентного атрибута
0x17	1	Зарезервированно

Идентификатор типа атрибута определяет семантику содержимого атрибута. В файловой системы существует набор зарезервированных типов. Кроме того существует возможность определения собственных типов атрибутов за счет их добавления в специальный файл (об этом позже).

В отличии от индентификатора типа атрибута индентификатор самго атрибута определет уникальный номер атрубута в рамках объекта, который его содержит.

За счет флагов могут быть определены дополнительные характеристики для области данных атрибутов.

Так значение 0x4000 означает, что данные атрибута хранятся в зашифрованном виде. Следует отметить, что процесс шифрования данных атрибута является нетривиальной задачей, которая обычно решаются студентами на занятиях... но и об этом тоже позже.

Значение 0x8000 говорит о том, что атрибут является разреженным. Это означает, что данные атрибута полностью состоят из нулей. В этом случае смысла хранить данные на цифровом носителе нет. Таким образом когда требует сохранить в атрибут неимоверного размера, но состоящий из нулей, на цифровом носителе создается заголовок атрибута, в котором указывается реальный размер данных и флаг разряженности.... все! Данные сохранены.

Последнее значение, которое будет рассмотрено — 0x1. Если атрибут сохраняется с указанным флагом, что к нему применяется процесс сжатия. В рамках этого процесса данные атрибута делятся на блоки, называемые блоками сжатия. К каждому блоку применяется одно из трех действий.

1. Если блок полностью состоит из нулей, то блок сохраняется в разреженном виде.

2. Если к блоку не применимо первое действие и при использовании алгоритма сжатия размер блока уменьшается, то блок записывается на диск в сжатом виде.

3. Если к блоку не применимо первое действие и при использовании алгоритма сжатия размер блока уменьшается незначительно, либо наоборот увеличивается[12], то блок записывается на диск в исходном виде.

Следует отметить, что сжатие применяется исключительно к нерезидентным атрибутам.

Флаги резидентного атрибута. О существовании данного поля в рекомендованной книге не указано, однако оно упоминается в исходных кодах ядра Linux. Там же в коде присутствует описание единственного флага (0x1), говорящего о том, что атрибут присутствует в индексе. Если флаг установлен, то при удалении, либо модификации атрибута, система должна определить в каком индексе учтен данный атрибут и перестроить индексное дерево.

Рассмотрим на примере структуру атрибута, хранящегося в файловой записи на рис. 7. На рис. 8 рассматривается исключительно часть файловой записи, отведенная под атрибуты.

Как видно на рисунке, файловая запись содержит 4 атрибута. При этом 2 из них резидентные, а 2 — нерезидентные. Кроме того, вызывает интерес значения поля идентификатор атрибута. В представленном примере они имеют значения: 0, 1, 3, 5. Одновременно следует вспомнить, что идентификатор нового атрибута, согласно рис. 7. Это означает, что в процессе использования тома из файловой записи были удалены 2 атрибута. Именно по этой причине используемый размер записи уменьшился и появился «хвост».

[12]Увеличение блока возможно по той причине, что к каждому блоку сжатия добавляется заголовок, содержащий служебные данные, необходимые для восстановления данных.

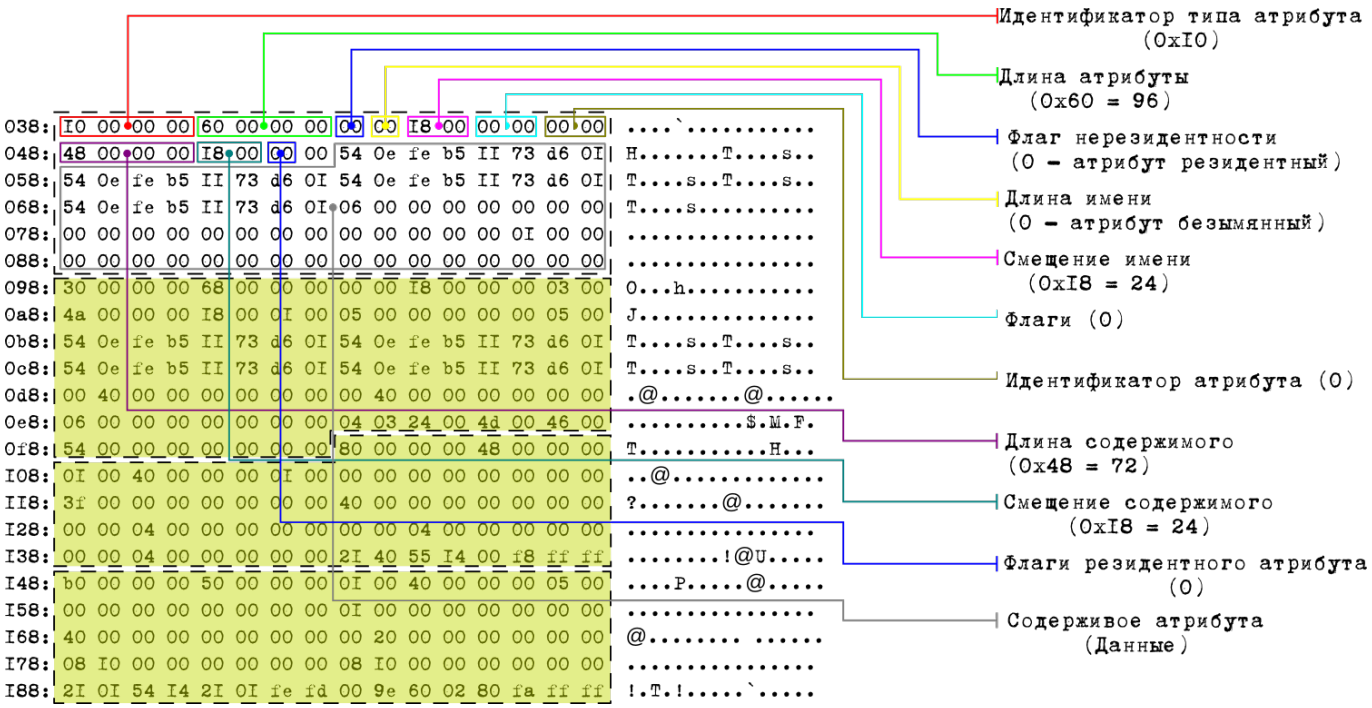

Рис 8. Пример структуры резидентного атрибута.

I.6.2 Нерезидентные атрибуты

Первые 16 байт заголовка нерезидентного атрибута полностью совпадают с заголовком резидентного, а дальше начинаются различия... Дабы не утруждать и без того утомленного непонятно чем студента бегать от странице к странице, ниже приведен полный формат заголовка нерезедентного атрибута (а не оставшиеся поля с отсылкой к прочитанному ранее).

Смещение	Размер	Описание
0x00	4	Идентификатор типа атрибута
0x04	4	Длина атрибуты
0x08	1	Флаг нерезидентности (1 – нерезидентный)
0x09	1	Длина имени (0 – атрибут безымянный)
0x0A	2	Смещение имени
0x0C	2	Флаги
0x0E	2	Идентификатор атрибута
0x10	8	Номер начального виртуального кластера

0xI8	8	Конечные номер виртуального кластера
0x20	2	Смещение списка серий
0x22	2	Размер блока сжатия
0x24	4	Не используется
0x28	8	Выделенный размер содержимого атрибута
0x30	8	Фактический размер атрибута
0x38	8	Инициализированный размер атрибута
0x40	8	Размер атрибута после сжатия (поле присутствует только у сжатых атрибутов)

Как было отмечено в разделе I.6.I для хранения данных могут использоваться несколько атрибутов. В этом случае, весь объем данных делятся на блоки размером в кластер и распределяются по атрибутам. Такие блоки называются виртуальными кластерами и нумеруются с нуля от начала данных. Для того, чтобы определить какие блоки размещаются в конкретном атрибуте и используются поля, хранящие начальный и конечный номер виртуального кластера.

Для определения объема дискового пространства, требуемого для хранения данных используются поля выделенный размер содержимого атрибута и размер атрибута после сжатия. Оба поля хранят выравненный до размера кластера размер данных. При этом, если атрибут сжатый, то фактический объем дискового пространства, занимаемого атрибутом, определяется по второму полю. Первое же поле всегда определяет размер для несжатых данных.

Поле фактический размер определяет реальный размер данных несжатого атрибута в байтах. Поле инициализированный размер, на моей памяти, ни разу не отличалось от фактического размера. Кстати, также приписка присутствует и в исходниках ядра Linux. Учитывая, что драйвер NTFS перед записью данных их накапливает перед записью на диск, на мой взгляд, данное поле используется для фиксации фактически записанных данных. Т.е. если до завершения записи данных

извлечь носитель, что возможно эти поля будут различаться.... но не факт....

На рис.9 разобрана структура последнего атрибута рассматриваемой ранее записи таблицы MFT. Не сложно заметить, что данный экземпляр атрибута описывает 0 и 1 виртуальные кластера данных, общий размер которых составляет 8192 байта. Забегая вперед скажу, что размер кластера на тестовом носителе составляет 4096 байта.

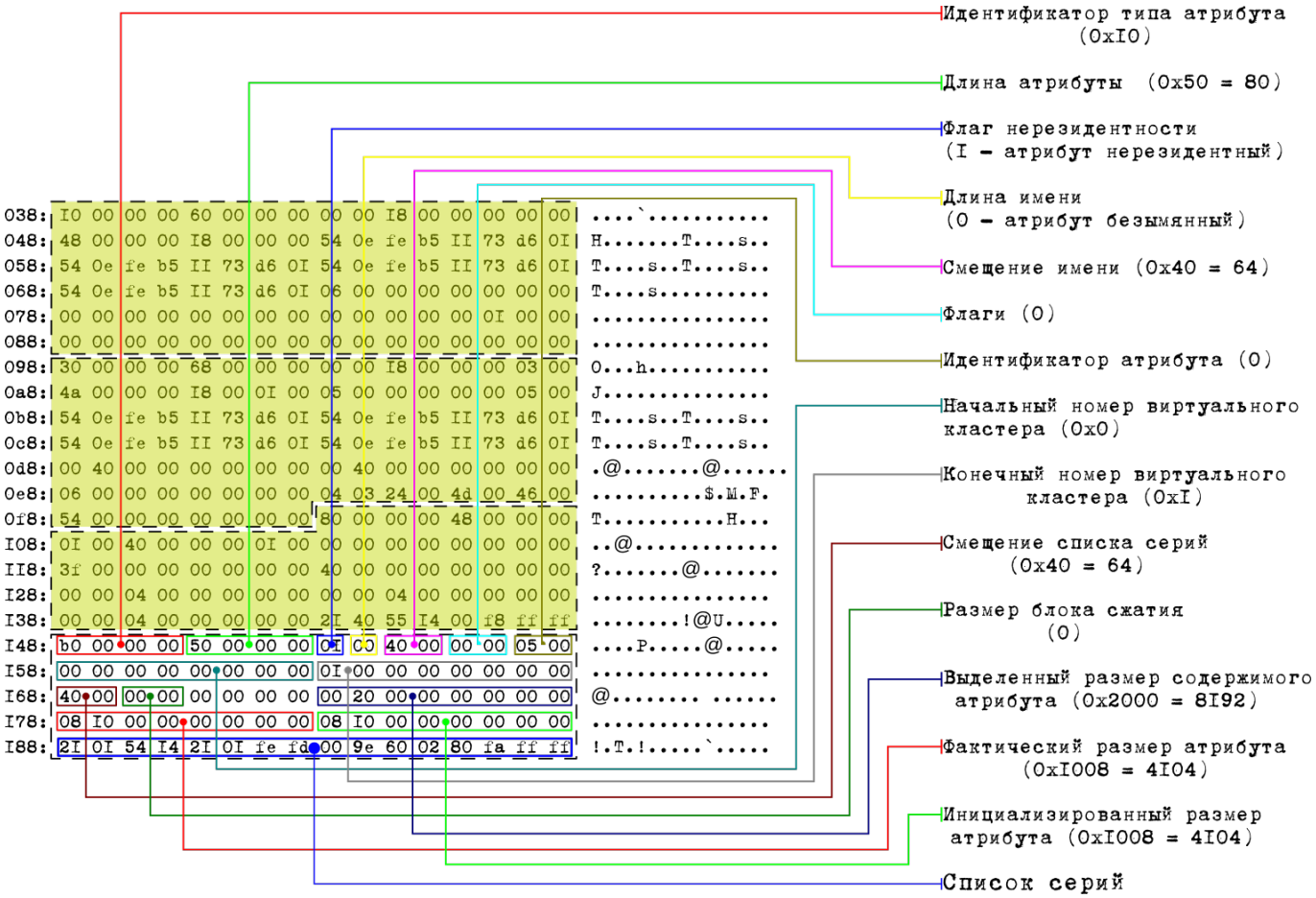

Рис 9. Пример структуры нерезидентного атрибута.

А теперь самое интересное... Список серий....

Список серий — это набор из элементов переменной длинны, но общей структуры, последовательно размещенные друг за другом. Элементы предназначены для описания последовательностей (серий) кластеров, в которых располагаются данные атрибута.

Трактовка каждого элемента определяется его первым байтом, который делиться на два полубайта, рассматриваемых отдельно. Возможны следующие варианты трактовок.

Старший	Младший	Описание
Не ноль	Не ноль	Описание серии, размещенной на диске
Ноль	Не ноль	Описание разреженной серии кластеров
Ноль	Ноль	Список серий завершен
Не ноль	Ноль	Список серий поврежден

Следует заменить, что нулевого значения первого байта в списке серий может и не встретится. В этом случае, список серии ограничивается размером атрибута.

При описании серии, размещенной на диске, дополнительно рассматриваются два поля, размер которых в байтах задается рассматриваемыми полубайтами. Размер первого поля задает младший полубайт, а само поле отвечает за размер серии, т.е. сколько кластеров в нее входит. Размер второго поля определяет старший полубайт, а само поле определяет смещение списка серий относительно начала предыдущей серии, либо от начала тома, если это первая серия в списке. Значения обоих полей являются знаковыми.

При описании разреженной серии кластеров, дополнительно рассматривается одно поле, размер которого задается младшим полубайтом. Данное поле определяет размер серии. Сама серия, как следует из описания разреженного атрибута, на диск не сохраняется. Теперь, получив новые и полезные знания, предлагаю разобраться со списком серий в атрибуте из нашего примера (рис. I0).

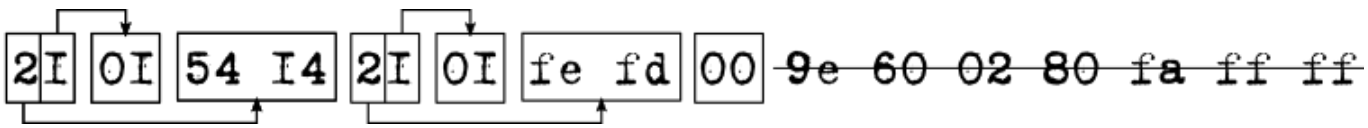

Рис I0. Пример разбора списка серий атрибута.

Как не сложно заметить для нахождении первой серии требуется чтения двух полей размером I и 2 байта. Из этих полей мы узнаем, что

первый блок данных размером I кластер располагается по смещению 5204 (0xI454) кластера от начала раздела. Для определения местоположения второй серии вновь требует чтение двух полей того же размера. Из них мы узнаем, что серия размером I кластер находится по смещению -5I4 ((signed short)0xfdfe) от начала предыдущей серии, либо 4690 (5204 - 5I4) кластера от начала раздела.

Далее нам встречается нулевой байт, что свидетельствует об окончании списка серий. Оставшиеся байты попали в атрибут случайно в процессе расширения его размера до некоторого кратного значения...

I.6.3 Типы атрибутов

Как было сказано ранее в файловой системе NTFS предусмотрен набор стандартных типов атрибутов. Далее я опишу те атрибуты, которые нам пригодится при изложении материала.

Начнем с самого простого! Идентификатор типа атрибута 0x80 — DATA. Этот атрибут предназначен для непосредственного хранения данных файла. Несмотря на простоту своего содержимого (иные атрибуты имеют сложную структуру), именно атрибут DATA неявно упоминается при разговоре про альтернативные потоки (stream). Чуть позже мы на примере разберем эту возможность, предоставляемую файловой системой. А сейчас продолжим разговор о типах атрибутов...

Идентификатор типа атрибута 0xI0 — STANDARD_INFORMATION. Данные атрибута содержат общую информацию о файловой записи и имеют следующую структуру.

Смещение	Размер	Описание
0x00	8	Время начала использования файловой записи Может ассоциироваться с созданием файла
0x08	8	Время модификации атрибута DATA (о нем дальше) Для файла это время модификации содержимого.
0xI0	8	Время модификации файловой записи
0xI8	8	Время последнего обращения.
0x20	4	Атрибуты (свойства) файла

0x24	4	Номер последний версии файла. (0 - если версия нет)
0x28	4	Версия данного файла.
0x2C	4	Идентификатор класса (даже не разбирался с этим)
0x30	4	Идентификатор владельца (пользователя).
0x34	4	Идентификатор безопасности.
0x38	8	Значение дисковой квоты для всех потоков файла Если ограничений нет, то значение поля - 0
0x40	8	Идентификатор последней записи в журнале транзакций

Следует отметить, актуальность поля время последнего обращений — спорный вопрос. Например, поле может не обновляется если цифровой носитель доступен только для чтения.

Сами временные метки показывают сколько сотен наносекунд прошло с I января I60I года.... При этом время храниться в UTC. Это означает, что созданный в Москве файл будет иметь временную метку на 3 часа меньше реальной. На рис.9 можно заметить, что значение всех временных меток равно 0xId673IIb5fe0e54, что соответствует I5 августа 2020, время I4:38:I5.89725.

Наиболее часто встречаются следующие значения атрибутов (свойств) файлов:

— 0x0I — файл доступен только для чтения;

— 0x02 — файл является скрытым;

— 0x04 — файл является системным;

— 0xI0 — директория;

В нашем примере запись описывает скрытый системный файл.

Что касается версий файлов, то вероятней всего речь идет о механизме транзакций, позволяющем скрывать изменения в файле от всех процессов до момента завершения транзакции.

Идентификатор типа атрибута 0x20 — ATTRIBUTE_LIST. Данные атрибута представляют из себя упорядоченное множество элементов, описывающих все атрибуты, которые существуют у файла (за исключением себя самого). Данный атрибут создается в том случае, когда для описания одного файла выделяется более одной записи в таблице MFT. В этом случае в основной записи (базовой) создается данный атрибут. Все элемента, которые в нем содержатся имеют различный размер (но обязательно кратный 8 байтам) и следуют друг за другом. Вместе с тем все элементы имеют следующую структуру.

Смещение	Размер	Описание
0x00	4	Идентификатор типа атрибута
0x04	2	Длина элемента
0x06	I	N - длина имени (0 - атрибут безымянны
0x07	I	Смещение имени
0x08	8	Номер начального виртуального кластера
0xI0	8	Идентификатор записи MFT, содержащий атрибут
0xI8	2	Идентификатор атрибута.
0xIA	N	Имя атрибута (если есть)

Сортировка элементов осуществляется последовательно по следующим полям:

I. Идентификатор типа атрибута.

2. Имя атрибута (если оно есть).

3. Идентификатор атрибута.

Идентификатор типа атрибута 0x30 — FILE_NAME. Данные атрибута содержат информацию о файле, который описывается данной записью. На первый взгляд рассматриваемый атрибут похож на атрибут STANDARD_INFORMATION, но это не совсем так. Данные атрибута FILE_NAME имеют следующую структуру.

Смещение	Размер	Описание
0x00	8	Ссылка на запись MFT, описывающую родительский каталог.
0x08	8	Время создания файла.
0xI0	8	Время модификации содержимого файла
0xI8	8	Время модификации файловой записи
0x20	8	Время последнего обращения.
0x28	8	Размер дискового пространства, выделенный под безымянный атрибут DATA, кратный размеру кластера.
0x30	8	Актуальный размер безымянного атрибута DATA.
0x38	4	Атрибуты (свойства) файла.
0x3C	4	Поле зависит от того, есть ли у файла расширенные атрибуты. Ниже поясню.
0x40	I	N = длина имени файла
0x4I	I	Пространство имен.

Теперь разберемся что же за странное поле расположено по смещению 0x3C. Содержимое данного поля зависит от двух факторов. Во-первых, содержит ли файл расширенные атрибуты (extended attributes). Если содержит, что поле I6 битный размер обрасти памяти, необходимый для хранения данных атрибутов.

Если расширенных атрибутов нет, то есть еще один вариант: запись используется неким не стандартным образом. В этом случае, поле содержит 32 битный идентификатор способа использования. В этом случае поле именуется как «reparse point tag», что в некоторых источниках именуется как «тег точки повторной обработки». На мой взгляд не сильно информативное название. Суть этого тега заключается в том, чтобы указать системе какой программой, отличной от драйвера файловой системы, либо какой конкретной подсистемой этого драйвера, следует обрабатывать данный файл. Сама формат поля выглядит следующим образом

Биты		Описание
Смещение	Размер	
00	I6	Значение тега
I6	I2	Зарезервированные биты
28	I	Нулевой бит!
29	I	Бит альтернативного наименования
30	I	Бит задержки
3I	I	Тег разработан компанией Microsoft

Как можно заметить описания 32 бита, компания Microsoft допускает разработку сторонними разработчиками собственных тегов. При их использовании этот бит должен быть равен 0. Что касается тегов от Microsoft, то они позволяют определять файл (директорию) как жесткую или мягкую ссылку, либо как точку монтирования цифрового носителя. Кому интересно подробнее почитать можно обратиться к следующей странице в сети Интернет http://docs.microsoft.com/ru-ru/windows/win32/fileio/reparse-point-tags.

Теперь обратимся к «биту задержки». Его описание, перейдя по вышеуказанной ссылке, вы не найдете. Для Microsoft это тоже просто зарезервированный бит. Его описание я нашел здесь:http://hex.pp.ua/reparse-point-custom.php. В соответствие с данным описание, этот бит говорит о том, что данные этого файла размещаются на весьма медленном носителе, т.е. при чтении и записи данных следует ожидать долгого отклика, а не рубить сессию по таймеру!

Что касается альтернативного именование, то тут все проще. Если бит установлен, то имя файла, указанное в атрибуте игнорируется, а вместо него будет кем-то выдано другое.

Теперь вернемся непосредственно к атрибуту FILE_NAME и рассмотрим поле «пространство имен». Дело в том, что для повышения совместимости, в файловой системе предусмотрена возможность различной интерпретации имения файла. Выделяются следующие пространства имен.

— 0x00 — POSIX. При чтении имен следует учитывать регистр букв. Допустимо использования любых UNICODE символов за исключением: '\0' и '/'. Кроме того, не рекомендуется использовать: '"', '<', '>', '\';

— 0x01 — WIN32. При чтении имен регистр букв не учитывается. Допустимо использования любых UNICODE символов за исключением: '\0', '"', '*', '/', ':', '<', '>', '?', '\', и '|';

— 0x02 — DOS. Все символы должны быть в верхнем регистре. Формат имени 8+3. Допускается использование только ascii-символ, код которых больше 0x20 (пробел) кроме: '"', '*', '+', ',', '/', ':', ';', '<', '=', '>', '?' и '\'.

Идентификатор типа атрибута 0x90 — INDEX_ROOT. Данные атрибута характерен для записей таблицы MFT, описывающих директории и отвечает за хранение корневого элемента B-дерева. Элементами дерева являются объекты, описывающие файлы и(или) поддиректории, размещенные в данной директории. В том случае, когда число объектов, хранящихся в каталоге, достаточно мало, чтобы быть размещенным в корневом, INDEX_ROOT используется самостоятельно. В противном случае, для директории создаются дополнительные атрибуты описанные ниже (INDEX_ALLOCATION и BITMAP).

Данные атрибута INDEX_ROOT имеют следующую структуру.

Смещение	Размер	Описание
0x00	4	Тип элементов, хранимых в индексе. Соответствует типам атрибутов.
0x04	4	Правила сортировки.
0x08	4	Размер каждого элемента индекса в байтах
0x0C	1	Размер каждого элемента, записанный в целочисленном типе Microsoft.
0x0D	3	Зарезервировано.
0x10	4	Смещение до корневого узла индексного дерева

0xI4	4	Выделенное пространство под узел
0xI8	4	Использованный размер выделенного пространства
0xIC	I	Флаг наличия иных узлов дереве
0xID	3	Зарезервировано.

Начну с очередной «непонятки», а именно наличию двух полей с размером элемента индекса. Если честно, то я не нашел (а может плохо искал) причину по которой этих полей два. Не сложно заметить, что первое поле позволяет описать размер элемента не превышающий $2^{32}-1$ байт. Второе поле позволяет задать размер не превышающий либо $2^{7}-I$ кластеров, либо $2^{2^{7}-I}$ байт.... В общем, первое поле позволяет точно задать, а второе много....)))) Но вот зачем так заморачиваться????

Под правилами сортировки понимаются не направление (убывание, возрастание), а правила рассмотрения элементов. Чтобы много времени не тратить на объяснения рассмотрим пример. Пусть дана следующая последовательность байт: 0xd0, 0x9I, 0xd0, 0x90. Если при сортировке рассматривать эту последовательность, как последовательность байт, то в результате можно получить либо последовательность 0x90, 0x9I, 0xd0, 0xd0, либо 0xd0,0xd0, 0x90, 0x9I... в зависимости от направления. Если же рассмотреть данную последовательность как набор символов в кодировке UTF-8, то исходная последовательность будет выглядеть следующим образом 0x9Id0, 0x90d0. Таким образом, в результатом сортировки будет является либо исходная последовательность, либо последовательность 0x90d0, 0x9Id0. Надеюсь хоть немного стало понятнее))))

Последнее поле, на которое следует обратить внимание это флаг наличия иных элементов элементов индекса. Это поле говорит о том существуют ли иныке узлы дерева, либо все элементы индекса были успешно размещены в корневом узле.

Так из чего же состоит узел дерева? Ответ достаточно прост — из индекстых элементов. Ниже приветена структыра элемента индекса.

Смещение	Размер	Описание

0x00	8	Сылка на двнне (см. дальше).
0x08	2	Размер элемента индекса.
0x0A	2	Размер ключа индекса.
0x0C	2	Флаг наличее потомков
0x0E	2	Зарезервировано.
0x10	*	Ключ.

Пояснять структуру начнем с последнего поля. До 3 версии NTFS индексы предназначался для работы исключительно с атрибутами FILE_NAME в качестве ключа. Позднее данный список был расширен. По этой причине размер ключа стал плавающим. При этом в исходных кодах ядра Linux определяются 7 вариантов ключей, а Microsoft вообще не вносит ключ в качетве элемента структуры, определяющей элемент индекса.

Ссылка на данные определяет исходя из назначения индекса. В тех случаях, когда индекс предназначен для указания на конкретные записи таблицы MFT (например, в директориях), данное поле содержин номер соответствующей запси. Если же двые должны храниться непосредвенно в теле элемента, то поле будет иметь следующую структуру.

Смещение	Размер	Описание
0x00	2	Смещение дынные.
0x02	2	Размер данных.
0x04	4	Зарезервировано.

Ну как-то так! Поехали дальше...

Идентификатор типа атрибута 0xA0 — INDEX_ALLOCATION. Данный атрибут содержит информацию об узлах B-дерева, обо всех, кроме корневого.

Как и в случаи с INDEX_ROOT, данные атрибута делятся на общий заголовок и набор непосредственных записей об элементах узла. Ниже приведена структура общего заголовка.

Смещение	Размер	Описание
0x00	4	Сигнатура "INDX"
0x04	2	Смещение массива маркеров
0x06	2	Количество элементов массива маркеров
0x08	8	Номер записи в журнале транзакций (LSN)
0x10	8	Номер начального виртуального кластера
0x18	4	Использованный размер выделенного пространства
0x1C	1	Типа индекса (Флаг).
0x1D	3	Зарезервировано.

Из перечня полей легко увидеть, что атрибут рассчитан на то, чтобы быт нерезидентным. На это указывает наличие массива маркеров. Кроме того, поле с номером виртуального кластера указывает на то, что для хранения всех элементов дерева используется более одного атрибута INDEX_ALLOCATION.

Тип индекса определяет указывают ли элементы индекса на иные узлы индекса, либо непосредвенно на элементы индекса.

Идентификатор типа атрибута 0xb0 — BITMAP. Ну тут вААААще все просто — данные этого атрибута - битовые поля.)))

Единственный нюанс заключается в том, что бы определить за что они отвечают.)))