


Основы построения файловых систем



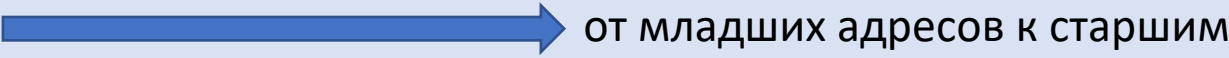
Устройство NTFS в целом



 от младших адресов к старшим

Устройство NTFS в целом

Boot Parameters Block	Master File Table	Free Space	More metadata	Free Space
-----------------------	-------------------	------------	---------------	------------



Boot Parameters Block – аналог superblock в ext2. Он указывает, где на диске отыскать начало MFT.


MFT – аналог inode table в ext2. Каждая запись в MFT описывает один файл*.

- More metadata – это область, хранящая
- MFT Mirror (копию первых четырёх записей в MFT),
 - Журнал.

** Иногда файл может занимать несколько записей в MFT (обсудим потом).*

Устройство NTFS в целом

Boot Parameters Block	Master File Table	Free Space	More metadata	Free Space
-----------------------	-------------------	------------	---------------	------------

 от младших адресов к старшим

Boot Parameters Block – аналог superblock в ext2. Он указывает, где на диске отыскать начало MFT.

MFT – аналог inode table в ext2. Каждая запись в MFT описывает один файл*.

- More metadata – это область, хранящая
- MFT Mirror (копию первых четырёх записей в MFT),
 - Журнал.

Можно ли увеличить размер MFT, если надо увеличить раздел с NTFS?

Где аналог block bitmap?

** Иногда файл может занимать несколько записей в MFT (обсудим потом).*

Boot Parameters Block (src/linux/fs/ntfs3/ntfs.h)

```
struct NTFS_BOOT {
    u8 jump_code[3];
    u8 system_id[8]; // "NTFS"

    u8 bytes_per_sector[2];

    u8 sectors_per_clusters;
    u8 unused1[7];
    u8 media_type;
    u8 unused2[2];
    __le16 sct_per_track;
    __le16 heads;
    __le32 hidden_sectors;
    u8 unused3[4];
    u8 bios_drive_num;
    u8 unused4;
    u8 signature_ex;

    u8 unused5;
    __le64 sectors_per_volume;
    __le64 mft_clst;
    __le64 mft2_clst;
    s8 mft_record_size;
    u8 unused6[3];
    s8 indx_record_size;
    u8 unused7[3];
    __le64 serial_num;
    __le32 check_sum;

    u8 boot_code[0x200 - 0x50 - 2 - 4];
    u8 boot_magic[2];
};
```

BPB содержит:

- Magic numbers для проверки того, что раздел действительно является разделом с NTFS,

Boot Parameters Block (src/linux/fs/ntfs3/ntfs.h)

```

struct NTFS_BOOT {
    u8 jump_code[3];
    u8 system_id[8]; // "NTFS"

    u8 bytes_per_sector[2];

    u8 sectors_per_clusters;
    u8 unused1[7];
    u8 media_type;
    u8 unused2[2];
    __le16 sct_per_track;
    __le16 heads;
    __le32 hidden_sectors;
    u8 unused3[4];
    u8 bios_drive_num;
    u8 unused4;
    u8 signature_ex;

    u8 unused5;
    __le64 sectors_per_volume;
    __le64 mft_clst;
    __le64 mft2_clst;
    s8 mft_record_size;
    u8 unused6[3];
    s8 indx_record_size;
    u8 unused7[3];
    __le64 serial_num;
    __le32 check_sum;

    u8 boot_code[0x200 - 0x50 - 2 - 4];
    u8 boot_magic[2];
};
    
```

BPB содержит:

- Magic numbers для проверки того, что раздел действительно является разделом с NTFS,
- Информацию о размере раздела и основных структур ФС,

Boot Parameters Block (src/linux/fs/ntfs3/ntfs.h)

```
struct NTFS_BOOT {
    u8 jump_code[3];
    u8 system_id[8]; // "NTFS"
    u8 bytes_per_sector[2];
    u8 sectors_per_clusters;
    u8 unused1[7];
    u8 media_type;
    u8 unused2[2];
    __le16 sct_per_track;
    __le16 heads;
    __le32 hidden_sectors;
    u8 unused3[4];
    u8 bios_drive_num;
    u8 unused4;
    u8 signature_ex;
    u8 unused5;
    __le64 sectors_per_volume;
    __le64 mft_clst;
    __le64 mft2_clst;
    s8 mft_record_size;
    u8 unused6[3];
    s8 indx_record_size;
    u8 unused7[3];
    __le64 serial_num;
    __le32 check_sum;
    u8 boot_code[0x200 - 0x50 - 2 - 4];
    u8 boot_magic[2];
};
```

BPB содержит:

- Magic numbers для проверки того, что раздел действительно является разделом с NTFS,
- Информацию о размере раздела и основных структур ФС,
- Указатели на MFT и MFT Mirror.

Boot Parameters Block (src/linux/fs/ntfs3/ntfs.h)

```
struct NTFS_BOOT {
    u8 jump_code[3];
    u8 system_id[8]; // "NTFS"

    u8 bytes_per_sector[2];

    u8 sectors_per_clusters;
    u8 unused1[7];
    u8 media_type;
    u8 unused2[2];
    __le16 sct_per_track;
    __le16 heads;
    __le32 hidden_sectors;
    u8 unused3[4];
    u8 bios_drive_num;
    u8 unused4;
    u8 signature_ex;

    u8 unused5;
    __le64 sectors_per_volume;
    __le64 mft_clst;
    __le64 mft2_clst;
    s8 mft_record_size;
    u8 unused6[3];
    s8 indx_record_size;
    u8 unused7[3];
    __le64 serial_num;
    __le32 check_sum;

    u8 boot_code[0x200 - 0x50 - 2 - 4];
    u8 boot_magic[2];
};
```


Как узнать

- версию ФС,
- размер MFT

?

Устройство MFT Entry

MFT Header	Attribute 0	Attribute 1	Attribute 2	...
------------	-------------	-------------	-------------	-----

 от младших адресов к старшим

В NTFS файл – это что-то вроде записи в таблице в БД. Атрибуты соответствуют колонкам таблицы.

Устройство MFT Entry

MFT Entry Header	Attribute 0	Attribute 1	Attribute 2	...
------------------	-------------	-------------	-------------	-----

 от младших адресов к старшим

В NTFS файл – это что-то вроде записи в таблице в БД. Атрибуты соответствуют колонкам таблицы.

Атрибуты регулярных файлов:

- \$STANDARD_INFORMATION (даты создания/модификации, флаги “системный”, “сжатый”, etc.)
- \$NAME,
- \$SECURITY_DESCRIPTOR,
- Данные.

Атрибуты файлов (src/linux/fs/ntfs3/ntfs.h)

ATTR header	Имя атрибута	Значение атрибута или runlist
-------------	--------------	-------------------------------

<pre>struct ATTRIB { enum ATTR_TYPE type; __le32 size; u8 non_res; u8 name_len; __le16 name_off; __le16 flags; __le16 id; union { struct ATTR_RESIDENT res; struct ATTR_NONRESIDENT nres; }; };</pre>	<pre>struct ATTR_RESIDENT { __le32 data_size; __le16 data_off; u8 flags; u8 res; };</pre>	<pre>struct ATTR_NONRESIDENT { __le64 svcn; __le64 evcn; __le16 run_off; u8 c_unit; u8 res1[5]; __le64 alloc_size; __le64 data_size; __le64 valid_size; __le64 total_size; };</pre>
	Короткие атрибуты, значения которых умещаются в MFT Entry. Обязательно резидентны Standard Information и имена файла.	Runlist – список екстентов, которые составляют значение атрибута.

Runlists

Runlist описывает, как VCN, Virtual Cluster Number (номера кластеров внутри значения атрибута), сопоставляются LCN, Logical Cluster Number (номерам кластеров на диске).

Runlist для несжатого файла представляет собой массив записей переменной длины (размеры указаны в элементах из 4 битов):

F	L	length	delta LCN	F	L	length	delta LCN	...
1	1	2*L	2*F	1	1	2*L	2*F	

Номера VCN, которые описывает i-й элемент runlist, вычисляются неявно: элементы runlist описывают примыкающие друг к другу части файла.

Номер LCN в runlist[i] получается как последний LCN, использованный в runlist[i-1], плюс delta LCN из runlist[i].

Runlists

Runlist описывает, как VCN, Virtual Cluster Number (номера кластеров внутри значения атрибута), сопоставляются LCN, Logical Cluster Number (номерам кластеров на диске).

Runlist для несжатого файла представляет собой массив записей переменной длины (размеры указаны в элементах из 4 битов):

F	L	length	delta LCN	F	L	length	delta LCN	...
1	1	2*L	2*F	1	1	2*L	2*F	

Номера VCN, которые описывает i-й элемент runlist, вычисляются неявно: элементы runlist описывают примыкающие друг к другу части файла.

Номер LCN в runlist[i] получается как последний LCN, использованный в runlist[i-1], плюс delta LCN из runlist[i].

Диапазону VCN могут не соответствовать никакие LCN: получается sparse file.

	F, L	Length	Delta LCN	
runlist[0]	4, 1	128	$2^{31} - 123$	Екстент начинается в кластере $2^{31} - 123$
runlist[1]	0, 1	64	(empty)	Sparse extent
runlist[2]	2, 1	128	2^{15}	Екстент начинается в кластере $2^{31} - 123 + 2^{15}$

Атрибуты регулярных файлов

- \$STANDARD_INFORMATION,
- \$FILE_NAME,
- \$OBJECT_ID,
- \$SECURITY_DESCRIPTOR,
- \$DATA,
- \$EA_INFORMATION,
- \$EA.

Атрибуты регулярных файлов

- \$STANDARD_INFORMATION,
- \$FILE_NAME,
- \$OBJECT_ID,
- \$SECURITY_DESCRIPTOR,
- \$DATA,
- \$EA_INFORMATION,
- \$EA.

```
struct ATTR_STD_INFO5 {  
    __le64 cr_time;           // 0x00: File creation file.  
    __le64 m_time;            // 0x08: File modification time.  
    __le64 c_time;            // 0x10: Last time any attribute was modified.  
    __le64 a_time;            // 0x18: File last access time.  
    enum FILE_ATTRIBUTE fa;    // 0x20: Standard DOS attributes & more.  
    __le32 max_ver_num;        // 0x24: Maximum Number of Versions.  
    __le32 ver_num;            // 0x28: Version Number.  
    __le32 class_id;  
  
    // Win2k and later  
  
    __le32 owner_id;           // 0x30: Owner Id of the user owning the file.  
    __le32 security_id;        // 0x34: The Security Id is a key in the  
                                // $SII Index and $SDS.  
    __le64 quota_charge;       // 0x38:  
    __le64 usn;                // 0x40: Last Update Sequence Number of the  
                                // file. This is a direct index into the file  
                                // $UsnJrnl. If zero, the USN Journal is  
                                // disabled.  
};
```

Атрибуты регулярных файлов

- \$STANDARD_INFORMATION,
 - **\$FILE_NAME**,
 - \$OBJECT_ID,
 - \$SECURITY_DESCRIPTOR,
 - \$DATA,
 - \$EA_INFORMATION,
 - \$EA.
- ```
struct ATTR_FILE_NAME {
 struct MFT_REF home; // 0x00: MFT record for directory.
 struct NTFS_DUP_INFO dup; // 0x08:
 u8 name_len; // 0x40: File name length in words.
 u8 type; // 0x41: File name type.
 __le16 name[]; // 0x42: File name.
};
```



# Атрибуты регулярных файлов

- \$STANDARD\_INFORMATION,
- **\$FILE\_NAME,**
- \$OBJECT\_ID,
- \$SECURITY\_DESCRIPTOR,
- \$DATA,
- \$EA\_INFORMATION,
- \$EA.

```
struct ATTR_FILE_NAME {
 struct MFT_REF home; // 0x00: MFT record for directory.
 struct NTFS_DUP_INFO dup; // 0x08:
 u8 name_len; // 0x40: File name length in words.
 u8 type; // 0x41: File name type.
 __le16 name[]; // 0x42: File name.
};
```

Тип файловых имён, по существу, задаёт пространство имён:

- FILE\_NAME\_POSIX,
- FILE\_NAME\_UNICODE,
- FILE\_NAME\_DOS.

У файлов на NTFS есть как минимум 2 имени: UNICODE-имя и DOS-имя, сокращённое до 8.3.

Файлы могут иметь дополнительные имена, если на них есть hardlinks.

# Атрибуты регулярных файлов

- \$STANDARD\_INFORMATION,
  - \$FILE\_NAME,
  - **\$OBJECT\_ID,**
  - \$SECURITY\_DESCRIPTOR,
  - \$DATA,
  - \$EA\_INFORMATION,
  - \$EA.
- ```
struct OBJECT_ID {  
    struct GUID ObjId;  
    struct GUID BirthVolumeId;  
    struct GUID BirthObjectId;  
    struct GUID DomainId;  
};
```

Уникальные идентификаторы файлов были добавлены для того, чтобы ярлыки на рабочем столе не ломались при переименовании или перемещении файлов.

Каталоги в NTFS

В NTFS каталог – это B-дерево со списком файлов, индексированных по некоторому их атрибуту.

Каталог – это файл, состоящий из следующих атрибутов:

- **\$STANDARD_INFORMATION,**
- **\$FILE_NAME,**
- **\$SECURITY_DESCRIPTOR,**
- **\$INDEX_ROOT,**
- **\$INDEX_ALLOCATION,**
- **\$BITMAP.**

Если индекс построен по атрибуту **\$NAME**, то получаем каталог в смысле ext2.

Индекс по атрибуту **\$ID** – это список файлов, которым присвоен уникальный идентификатор. Идентификатор присваивается файлу при создании на него ярлыка. Тогда файл можно переименовывать, а ярлыки продолжают на них ссылаться.

Файлы с большим количеством атрибутов.

Как быть с файлами, состоящими из большого числа экстенгов (фрагментированными или дырявыми)? Их runlist может не поместиться в одну MFT Entry.

Файлы с большим количеством атрибутов.

Как быть с файлами, состоящими из большого числа экстенгов (фрагментированными или дырявыми)? Их runlist может не поместиться в одну MFT Entry.

1. Файл может описываться многими MFT Entry. Файл может иметь атрибут \$ATTRIBUTE_LIST (возможно, нерезидентный!), значение которого – это массив из

```
struct ATTR_LIST_ENTRY {  
    enum ATTR_TYPE type;           // 0x00: The type of attribute.  
    __le16 size;                   // 0x04: The size of this record.  
    u8 name_len;                   // 0x06: The length of attribute name.  
    u8 name_off;                   // 0x07: The offset to attribute name.  
    __le64 vcn;                    // 0x08: Starting VCN of this attribute.  
    struct MFT_REF ref;            // 0x10: MFT record number with attribute.  
    __le16 id;                     // 0x18: struct ATTRIB ID.  
    __le16 name[];  
};
```

Элементы массива перечисляют атрибуты файла и указывают, в каких MFT Entry выделено место для их хранения.

Файлы с большим количеством атрибутов.

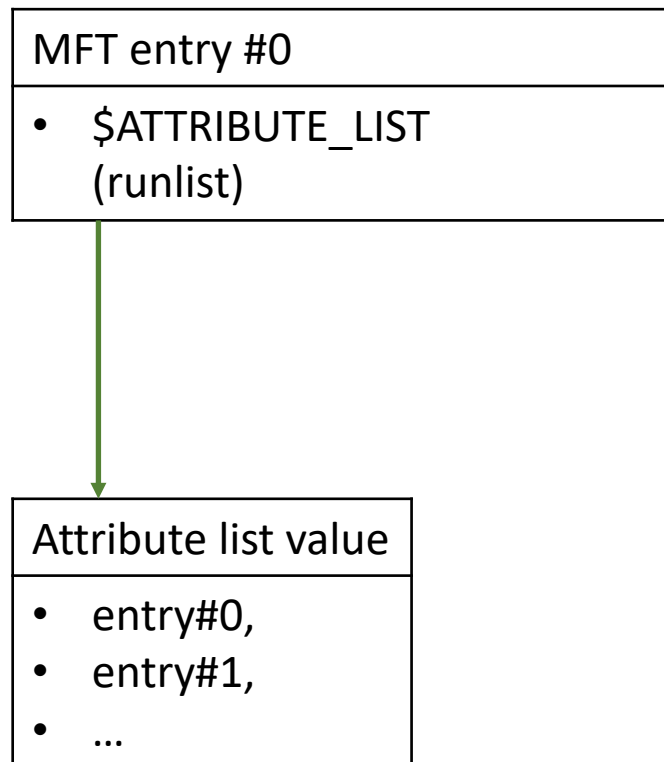
Как быть с файлами, состоящими из большого числа экстенгов (фрагментированными или дырявыми)? Их runlist может не поместиться в одну MFT Entry.

- 1. Файл может описываться многими MFT Entry.
- 2. Атрибут файла может повторяться. Тогда его значение получается как конкатенация значений повторяющихся атрибутов.

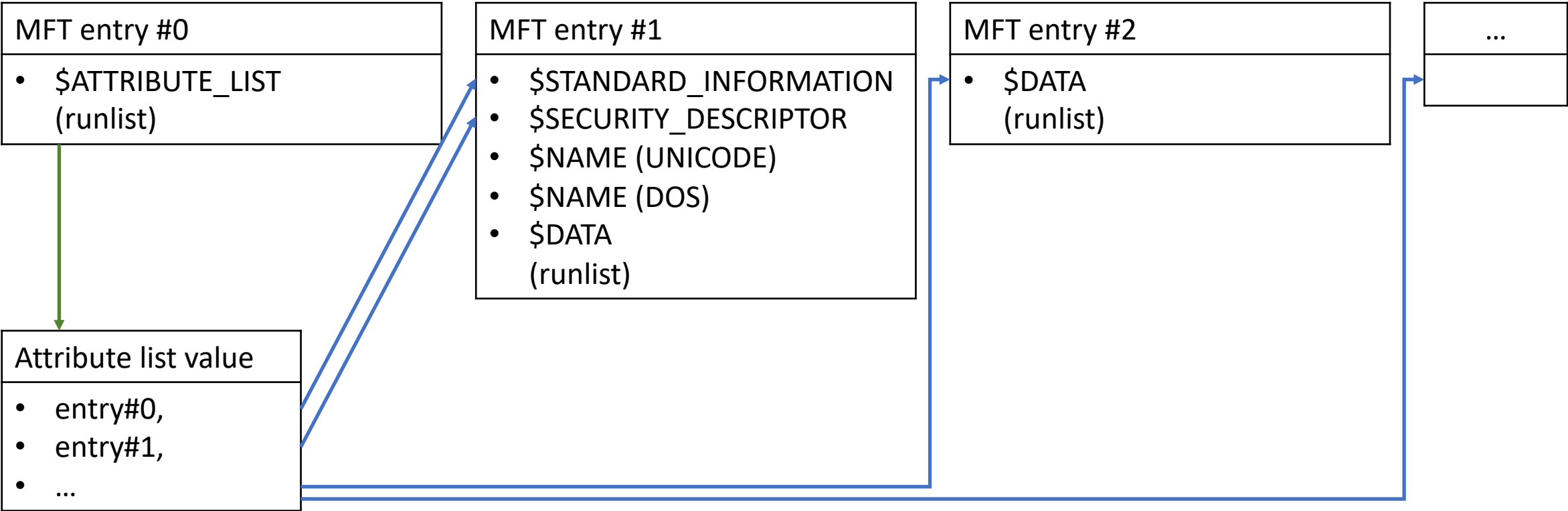
```
struct ATTR_NONRESIDENT {  
    __le64 svcn;           // 0x10: Starting VCN of this segment.  
    __le64 evcn;           // 0x18: End VCN of this segment.  
    __le16 run_off;  
    u8 c_unit;  
    u8 res1[5];  
    __le64 alloc_size;  
    __le64 data_size;  
    __le64 valid_size;  
    __le64 total_size;  
};
```

Runlist нерезидентного атрибута начинает отсчитывать VCN от SVCN атрибута (Start VCN). Это позволяет склеивать runlists из повторяющихся атрибутов.

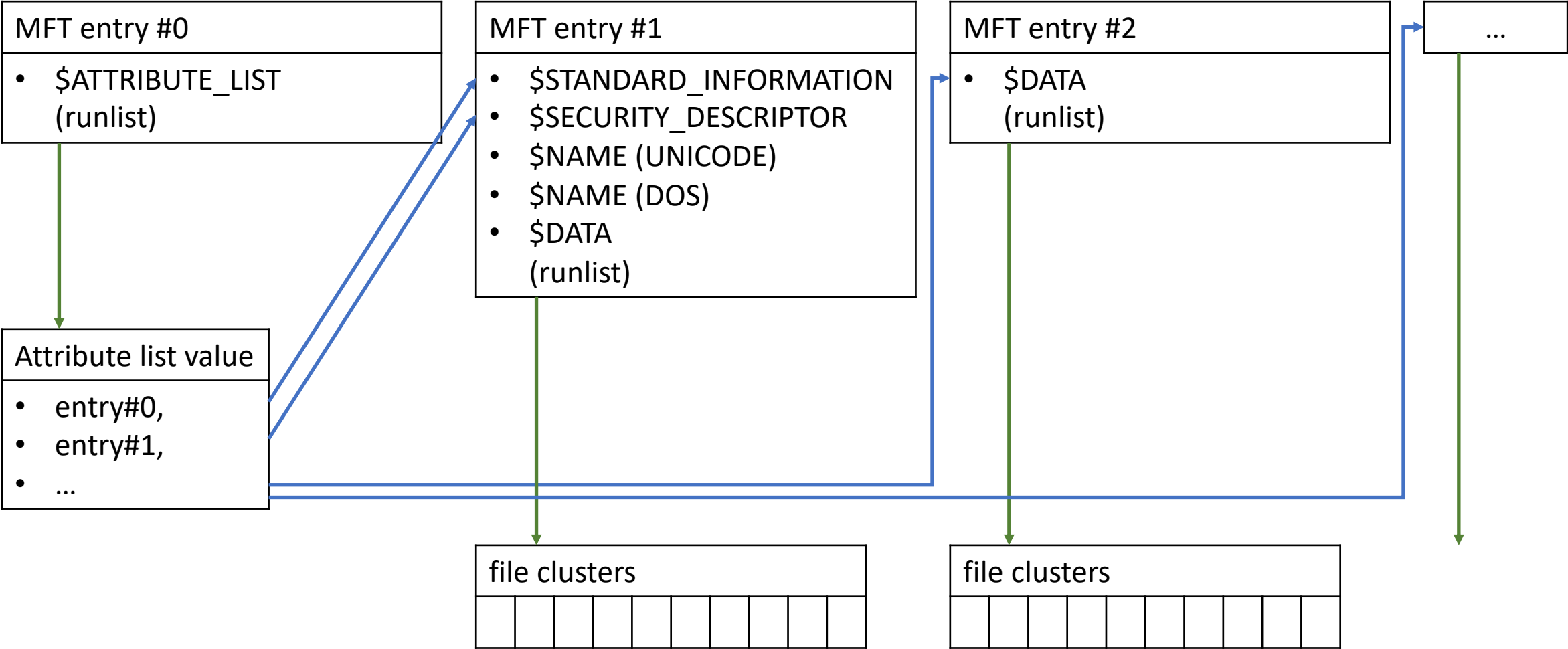
Файлы с большим количеством атрибутов: пример



Файлы с большим количеством атрибутов: пример



Файлы с большим количеством атрибутов: пример



Файлы в NTFS: отличия от POSIX

1. Имена файлов для разных пространств имён.
2. Имена файлов в пространстве имён “Unicode” имеют **case-preserving** семантику, т.е. имена файлов сохраняются так, как их передали приложения, но поиск файла при открытии делается без учёта строчных/прописных букв.
3. Каталоги могут группировать файлы по произвольному атрибуту, допускающему сравнение.
4. Множественные потоки данных:
 - `OpenFile(“file.txt”)` открывает неименованный `$DATA`,
 - `OpenFile(“file.txt:alt”)` открывает `$DATA` с именем “alt”.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- **\$MFT**,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Описывает область на диске, которую занимает MFT. Нужно для того, чтобы менять размер MFT при увеличении ФС или при заполнении её пользовательскими данными.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- **\$Volume**,
- \$AttrDef,
- .
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Содержит атрибуты \$VOLUME_NAME и \$VOLUME_INFORMATION: имя раздела и версию NTFS.

```
struct VOLUME_INFO {  
    __le64 res1;    // 0x00  
    u8 major_ver;   // 0x08: NTFS major version number  
    u8 minor_ver;   // 0x09: NTFS minor version number  
    __le16 flags;   // 0x0A: Volume flags, see VOLUME_FLAG_XXX  
};
```

```
#define VOLUME_FLAG_DIRTY                cpu_to_le16(0x0001)  
#define VOLUME_FLAG_RESIZE_LOG_FILE     cpu_to_le16(0x0002)
```

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- . <---
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Корневой каталог раздела.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- **\$Bitmap**,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Битовая маска занятых и свободных кластеров. Аналог block bitmap в ext2.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- \$Bitmap,
- **\$Boot**,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

Описывает область на диске с BPB.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- **\$UpCase**,
- \$Extend,
- \$ObjId,
- \$Reparse,
- \$UsnJournal.

NTFS – это **case-preserving** ФС, т.е. имена файлов сохраняются так, как их передали приложения, но поиск файла при открытии делается без учёта строчных/прописных букв.

Это проблема, т.к. правила преобразования регистра букв зависят не только от языка и региона, но и от версии Unicode. NTFS-разделы хранят правила преобразования символов, которые были актуальны при создании раздела.

Системные файлы в NTFS

Первые 16 файлов в NTFS-разделе – это системные файлы, содержащие метаданные самой ФС:

- \$MFT,
- \$MFTMirr,
- \$LogFile,
- \$Volume,
- \$AttrDef,
- .
- \$Bitmap,
- \$Boot,
- \$BadClus,
- \$Quota,
- \$Secure,
- \$UpCase,
- \$Extend,
- \$ObjId,
- **\$Reparse**,
- \$UsnJournal.

Список точек монтирования. В отличие от Linux или BSD, точки монтирования существуют не в runtime, а постоянно сохраняются на диске.

Особые типы reparse points:

- Junction point (аналог mount или bind-mount, >= win2k),
- Символические ссылки (>= vista),
- OneDrive.

Дополнительное чтение

- <https://dubeyko.com/development/FileSystems/NTFS/ntfsdoc.pdf>
- `/src/linux/fs/ntfs3/*`

Домашнее задание

Разберитесь с API библиотеки ntfs-3g и напишите программу, которая по образу диска строит список занятых секторов на нём, притом сопоставляет каждому сектору файл, который его использует.

1. Создайте раздел с NTFS,
2. Раздайте его по iSCSI,
3. Подключите этот раздел только на чтение в VM с Windows,
4. Запишите iSCSI-сессию с помощью tcpdump,
5. Прочтите несколько файлов с этого раздела,
6. Используя список из задачи 1, по дампу iSCSI-сессии восстановите список файлов, которые читала VM.