Лекция 10:

**Управление устройствами**

**Подсистема ввода-вывода**

В компьютерных системах кроме процессора и оперативной памяти присутствует множество разнообразных устройств (*device*) – жесткие диски, приводы оптических дисков (CD, *DVD*, Blu-Ray *Disk*), устройства флеш-памяти, принтеры, сканеры, звуковые и видеокарты, модемы, сетевые карты и т. п.

*Операционная система* должна обеспечивать управление всеми этими устройствами, т. е. предоставлять способы обмена информацией между приложениями и устройствами.

Управление устройствами в *Windows* осуществляется подсистемой ввода вывода, включающей несколько компонентов (см. [рис.4.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16583) в лекции 4 "*Архитектура* *Windows*"):

* диспетчер ввода-вывода (I/O manager – Input/Output manager) – основной компонент; обеспечивает интерфейс между приложениями и устройствами;
* диспетчер PnP (Plug and Play manager) – компонент, реализующий принцип Plug and Play ("подключи и работай") – автоматическое распознавание и конфигурацию подключаемых к системе устройств;
* диспетчер электропитания (power manager) – обеспечивает поддержку различных режимов энергопотребления системы и устройств;
* драйверы устройств – программы, реализующие операции ввода-вывода для конкретного устройства; драйверы больше других компонентов системы "знают" о специфике своего устройства;
* HAL (Hardware Abstraction Layer) – уровень абстрагирования от аппаратных средств; скрывает от других компонентов особенности реализации конкретных процессоров, системных плат и контроллеров прерываний;
* реестр (registry) – используется как база данных для параметров устройств и драйверов.

Далее будут рассмотрены общая схема ввода-вывода, функции и структуры данных диспетчера ввода-вывода, представленные в WRK, а также пример выполнения *операции* чтения.

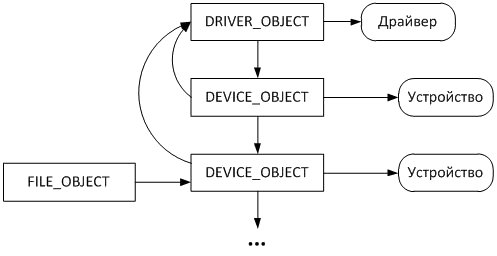
**Принцип управления устройствами**

Рассмотрим схематично принцип управления внешними устройствами, а затем перейдем к изучению соответствующих структур и функций WRK.

Для пользовательских приложений *операционная система* представляет устройства в виде файлов. Такое *представление* позволяет единообразно работать с разными устройствами, используя одинаковые функции, не задумываясь о деталях реализации доступа к устройствам.

*Файл* (file) – совокупность данных, имеющих имя и допускающих *операции* чтения-записи. Типичная последовательность работы с файлом: *открытие файла*, выполнение команд чтения-записи, *закрытие файла*.

При открытии файла создается файловый *объект* типа FILE\_OBJECT, который связан с объектом, представляющим конкретное устройство (DEVICE\_OBJECT). В объекте-устройстве содержится *информация* о драйвере, который управляет этим устройством. *Драйвер* в системе описывается объектом типа DRIVER\_OBJECT. Объекты DRIVER\_OBJECT создаются при загрузке в систему нового драйвера. Затем *объект* DRIVER\_OBJECT может создать несколько объектов DEVICE\_OBJECT – по количеству управляемых драйвером устройств ([рис.15.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16583?page=1#image.15.1)).



**Рис. 15.1.**Объекты для управления вводом-выводом

Как видно из [рис.15.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16583?page=1#image.15.1), в объекте DRIVER\_OBJECT содержится *указатель* на *список* объектов-устройств, а в каждом из этих объектов хранится *ссылка* на управляющий *драйвер*. Таким образом, имея информацию об объекте DRIVER\_OBJECT, можно найти все устройства, которыми он управляет и, наоборот, по объекту DEVICE\_OBJECT легко определяется *драйвер* устройства.

*Приложение*, которому необходимо произвести некоторую операцию с устройством (файлом), вызывает соответствующую WinAPI функцию (CreateFile, ReadFile, WriteFile и др.), которая, в свою *очередь*, обращается к функции диспетчера ввода-вывода.

Операция, которая запрашивается приложением, представляется в системе объектом типа *IRP* (*I/O* Request *Packet* – пакет запроса на *ввод/вывод*). В этом объекте хранится *информация* о типе *операции* ввода/вывода (создание, чтение, *запись* и т. п.), а также необходимые параметры для данной *операции*. Пакет *IRP* передается диспетчером ввода-вывода в *очередь* *IRP* потока, который запросил операцию ввода-вывода, после чего вызывается соответствующий *драйвер*, непосредственно выполняющий запрошенную операцию.

**Структуры данных для ввода-вывода**

*Драйвер* в системе описывается объектом типа DRIVER\_OBJECT (*файл* base\ntos\inc\io.h, строка 1603), имеющим следующие основные поля:

* Type – поле, определяющее тип структуры подсистемы ввода-вывода. Значения этого поля могут быть следующими – IO\_TYPE\_DRIVER, IO\_TYPE\_FILE, IO\_TYPE\_DEVICE, IO\_TYPE\_IRP и др. (см. файл base\ntos\inc\io.h, строка 25);
* Size – размер объекта в байтах;
* DeviceObject – ссылка на первый объект DEVICE\_OBJECT в списке устройств, управляемых данным драйвером (см. [рис.15.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16583?page=1#image.15.1)). Следующие устройства в списке можно определять по полю NextDevice объекта DEVICE\_OBJECT;
* Flags – флаги, определяющие тип драйвера (см. файл base\ntos\inc\io.h, строка 1530);
* DriverName – имя драйвера в системе;
* HardwareDatabase – путь в реестре к информации о драйвере;
* DriverStart, DriverSize, DriverSection – информация о расположении драйвера в памяти;
* DriverInit – адрес процедуры DriverEntry (точка входа в драйвер), отвечающей за инициализацию драйвера;
* DriverUnload – адрес процедуры выгрузки драйвера;
* MajorFunction – массив адресов процедур, каждая из которых отвечает за определенную операцию с устройством. Максимальное количество таких процедур равно константе IRP\_MJ\_MAXIMUM\_FUNCTION+ 1 = 2 8 (файл base\ntos\inc\io.h, строка 80), которая определяет также количество кодов IRP (см. далее).

Устройства представлены объектами типа DEVICE\_OBJECT, который включает следующие главные поля (*файл* base\ntos\inc\io.h, строка 1397):

* Type, Size – совпадают по назначению с полями типа DRIVER\_OBJECT;
* ReferenceCount – счетчик количества открытых дескрипторов для устройства. Позволяет отслеживать, используется кем-либо устройство или нет;
* DriverObject – ссылка на драйвер, который управляет устройством;
* NextDevice – указатель на следующее устройство в списке устройств для данного драйвера;
* Flags, Characteristics – поля, уточняющие характеристики устройства;
* DeviceType – тип устройства; возможные типы перечислены в файле public\sdk\inc\devioctl.h (строка 26);
* SecurityDescriptor – дескриптор безопасности, сопоставленный с устройством (см. лекцию 9 "Безопасность в Windows").

Пакеты запроса на ввод-*вывод* описываются типом *IRP* (*I/O* Request *Packet*), состоящим из двух частей – заголовка фиксированной длины (тело *IRP*) и одного или нескольких блоков стека. В заголовке описывается *информация*, общая для запроса. Каждый блок стека содержит данные об одной *операции* ввода-вывода.

Заголовок включает следующие основные поля:

* Type, Size – поля, по назначению аналогичные соответствующим полям типов DRIVER\_OBJECT и DEVICE\_OBJECT;
* IoStatus – статус операции при завершении;
* RequestorMode – режим, в котором работает поток, инициировавший операцию ввода-вывода, – пользовательский или режим ядра;
* StackCount – количество блоков стека;
* Tail.Overlay.Thread – указатель на структуру ETHREAD потока, запросившего операцию ввода-вывода;
* Tail.Overlay.CurrentStackLocation – указатель на блок стека (IRP Stack Location), который описывается структурой IO\_STACK\_LOCATION.

Структура блока стека IO\_STACK\_LOCATION описана в файле base\ntos\inc\io.h, строка 2303) и имеет следующие главные поля:

* MajorFunction – номер основной функции, определяющий запрошенную операцию ввода-вывода и совпадающий с номером функции драйвера в массиве MajorFunction (структура DRIVER\_OBJECT, см. выше), которую нужно вызвать для выполнения запрошенной операции. Как уже отмечалось, всего кодов 28 (IRP\_MJ\_MAXIMUM\_FUNCTION + 1), они описаны в файле base\ntos\inc\io.h (строки 51–79);
* DeviceObject – указатель на структуру DEVICE\_OBJECT, определяющую устройство для данной операции ввода-вывода;
* FileObject – указатель на структуру FILE\_OBJECT (файл base\ntos\inc\io.h, строка 1763), которая ассоциирована со структурой DEVICE\_OBJECT.

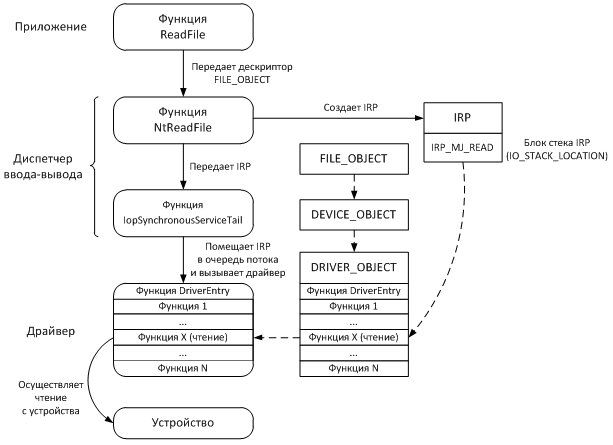
В следующем параграфе разобран пример *операции* чтения с использованием рассмотренных выше структур данных.

**Пример ввода-вывода**

Для ввода-вывода используются следующие основные функции:

* создание/открытие файла – IoCreateFile (файл base\ntos\io\iomgr\iosubs.c, строка 4795);
* чтение из файла – NtReadFile (файл base\ntos\io\iomgr\read.c, строка 90);
* запись в файл – NtWriteFile (файл base\ntos\io\iomgr\write.c, строка 87);
* закрытие файла – IopDeleteFile файл base\ntos\io\iomgr\objsup.c, строка 465).

Рассмотрим пример чтения с устройства, используя изученные структуры данных и функцию NtReadFile ([рис.15.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16583?page=1#image.15.2)).



**Рис. 15.2.**Последовательность операций и структуры данных при чтении с устройства

Предположим, некоторому приложению требуется прочитать данные с устройства, например, из файла на жестком диске. Предварительно *приложение* должно получить *дескриптор* объекта FILE\_OBJECT, например, при помощи WinAPI функции CreateFile.

Для чтения из файла *приложение* вызывает WinAPI-функцию ReadFile, которая обращается к функции диспетчера ввода-вывода NtReadFile и передает ей *дескриптор* объекта FILE\_OBJECT.

*Функция* NtReadFile определена в файле base\ntos\io\iomgr\read.c (строка 90) и выполняет две основные задачи – создает *объект* *IRP* (строка 517) и вызывает функцию IopSynchronousServiceTail (строка 725). При создании объекта *IRP* в блок стека заносится номер основной функции (Major *Function*), в случае *операции* чтения этот код равен константе IRP\_MJ\_READ (строка 558) и указывает на функцию чтения в массиве MajorFunction структуры DRIVER\_OBJECT.

*Функция* IopSynchronousServiceTail определена в файле base\ntos\io\iomgr\internal.c (строка 7458). Эта *функция* помещает переданный ей *объект* *IRP* в *очередь* потока (*функция* IopQueueThreadIrp, строка 7468). *Указатель* на *очередь* *IRP* потока хранится в *поле* IrpList структуры ETHREAD (*файл* base\ntos\inc\ps.h, строка 623). Кроме этого, *функция* IopQueueThreadIrp вызывает соответствующий *драйвер* (*функция* IoCallDriver, строка 7494).

*Драйвер* выполняет определенную кодом *IRP* функцию и возвращает статус *операции*.

**Резюме**

В лекции представлены компоненты подсистемы ввода вывода в *Windows*, рассмотрен принцип управления устройствами, а также реализация этого принципа на основе структур данных и функций *Windows* Research *Kernel*. Разобран пример ввода вывода для *операции* чтения из файла.

Лекция 11:

**Файловая система NTFS**

**Основные понятия**

*Файловая система* (*file system*) – способ организации данных в виде файлов на устройствах внешней памяти (жестких и оптических дисках, устройствах флеш-памяти и т. п.).

*Файловая система* должна обеспечивать:

1. безопасное и надежное хранение данных (т. е. защищенное от несанкционированного использования и различного рода сбоев и ошибок);
2. программный интерфейс доступа к файлам;
3. организацию файлов в виде иерархии каталогов.

*Windows* поддерживает несколько файловых систем для различных внешних устройств:

* NTFS – основная файловая система семейства Windows NT;
* FAT (File Allocation Table – таблица размещения файлов) – простая файловая система используемая Windows для устройств флеш памяти, а также для совместимости с другими операционными системами при установке на диски с множественной загрузкой. Основным элементом этой файловой системы является таблица размещения файлов FAT (по имени которой названа вся файловая система), необходимая для определения расположения файла на диске. Существует три варианта FAT, отличающихся разрядностью идентификаторов, указывающих размещение файлов: FAT12, FAT16 и FAT32;
* exFAT (Extended FAT – расширенная FAT) – развитие файловой системы FAT, использующее 64 разрядные идентификаторы. Применяется в основном для устройств флеш-памяти;
* CDFS (CD ROM File System) – файловая система для CD дисков, объединяющая форматы ISO 96601 и Joliet2;
* UDF (Universal Disk Format – универсальный формат дисков) – файловая система для CD и DVD дисков, разработанная для замены ISO 9660.

Для дальнейшего изложения необходимо знать следующие важные понятия: *диск*, раздел, простые и составные тома, сектор, *кластер*.

*Диск* (*disk*) – *устройство внешней памяти*, например, жесткий *диск* или оптический *диск* (CD, *DVD*, Blu ray).

Раздел (*partition*) – непрерывная часть жесткого диска. *Диск* может содержать несколько разделов.

Том (*volume*) или *логический* *диск* (logical *disk*) – область внешней памяти, с которой *операционная система* работает как с единым целым. Тома бывают простые и составные.

Простой том (simple *volume*) – том, состоящий из одного раздела.

Составной том (multipartition *volume*) – том, состоящий из нескольких разделов (необязательно на одном диске).

Понятия раздела и простого тома отличаются: во первых, *разделы* формируются, в основном, только на жестких дисках, а тома создаются и на других устройствах внешней памяти (например на оптических дисках и устройствах флеш памяти), во вторых, понятие "раздел" связано с физическим устройством, а понятие "том" – с логическим представлением внешней памяти.

Сектор (*sector*) – *блок данных* фиксированного размера на диске; наименьшая *единица* информации для диска. Типичный размер сектора для жестких дисков равен 512 байтам, для оптических дисков – 2048 *байт*. *Деление* диска на секторы происходит один раз при создании диска в процессе низкоуровневого форматирования и обычно не может быть изменено.

*Кластер* (*cluster*) – *логический* *блок данных* на диске, включающий один или несколько секторов. Количество секторов, составляющих *кластер*, обычно кратно степеням двойки. *Размер кластера* задается операционной системой в процессе высокоуровневого форматирования, которое может осуществляться многократно.

При записи на *диск* *файл* всегда будет занимать *целое число* кластеров. Например, *файл* размером 100 *байт* в файловой системе с размером кластера 4 КБ будет занимать ровно 4 КБ.

Выбор размера кластера связан со следующими соображениями. Малые кластеры позволяют сократить размер фактически неиспользуемого дискового пространства, возникающего за счет размещения файла в целом числе кластеров. Но при этом общее количество кластеров на диске увеличивается и размер служебных структур файловой системы, в которых хранится *информация* о файлах, возрастает.

**Возможности NTFS**

*Файловая система* *NTFS* (*New* *Technology* *File System*) разрабатывалась Microsoft в начале 1990 х гг. как основная *файловая система* для серверных версий операционных систем *Windows*. *NTFS* была представлена в 1993 году в операционной системе *Windows* NT 3.1.

В настоящее время *NTFS* рассматривается в качестве предпочтительной файловой системы как для серверных, так и для клиентских версий *Windows*.

В *NTFS* используются 64 разрядные идентификаторы кластеров, поэтому теоретически том *NTFS* может содержать 264 кластеров (16 ЭБ3 ). Однако текущие реализации в *Windows* поддерживают только 32 разрядную адресацию кластеров, что при размере кластера *максимум* 64 КБ (216 *байт*) позволяет *NTFS* тому достигать размера до 256 ТБ:

232 \* 216 байт = 248 байт = 28 \* 240 байт = 256 ТБ.

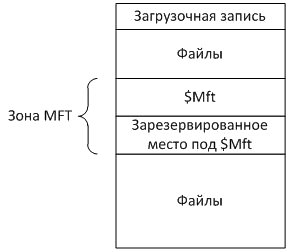
Для томов, больших 4 ГБ, при форматировании *Windows* предлагает *размер кластера* *по* умолчанию 4 КБ.

Перечислим некоторые возможности *NTFS* [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5), стр. 761]:

* восстанавливаемость (recoverability) – способность файловой системы возвращаться к работоспособному состоянию после возникновения сбоя. Реализуется такая возможность, во первых, за счет поддержки атомарных транзакций, во вторых, за счет избыточности хранения информации. Атомарная транзакция (atomic transaction) – операция с файловой системой, приводящая к её изменению, которая либо полностью успешно выполняется, либо не выполняется вообще (т. е. в случае сбоя во время атомарной транзакции все изменения откатываются). Избыточность используется при хранении важнейших данных файловой системы, критически необходимых для её корректной работы;
* безопасность (security) – защищенность файлов от несанкционированного доступа. Реализуется при помощи модели безопасности Windows, рассмотренной в лекции 9 "Безопасность в Windows";
* шифрование (encryption) – преобразование файла в зашифрованный код, который невозможно прочесть без ключа. Обычные механизмы безопасности, такие как назначение прав доступа пользователей к файлам, не обеспечивают полной защиты информации, например, в случае перемещения диска на другой компьютер. Администратор операционной системы всегда может получить доступ к файлам других пользователей, даже на томе NTFS. Поэтому в NTFS включена поддержка шифрующей файловой системы EFS (Encrypting File System), которая позволяет легко зашифровывать и расшифровывать файлы;
* поддержка RAID (Redundant Array of Inexpensive (Independent) Disks – массив недорогих (независимых) дисков с избыточностью) – возможность использования для хранения информации нескольких дисков; данные с одного диска автоматически копируются на другие, обеспечивая тем самым повышенную надежность;
* дисковые квоты для пользователей (Per-User Volume Quotas) – возможность выделения для каждого пользователя определенного пространства на диске (квоты); NTFS не позволяет пользователю записывать данные на диск сверх выделенной квоты.

**Структура NTFS**

Структура тома *NTFS* представлена на [рис.17.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=1#image.17.1).



**Рис. 17.1.**Структура NTFS тома

В начале тома находится загрузочная *запись* тома (*Volume* *Boot* *Record*), в которой содержится код загрузки *Windows*, *информация* о томе (в частности, тип файловой системы), адреса системных файлов ($*Mft* и $MftMirr – см. далее). Загрузочная *запись* занимает обычно 8 КБ (16 первых секторов).

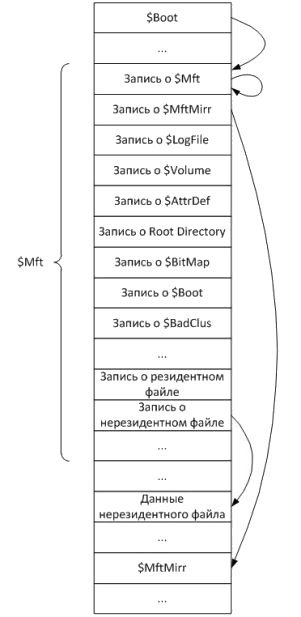
В определенной области тома (*адрес* начала этой области указывается в загрузочной записи) расположена основная системная структура *NTFS* – главная *таблица* файлов (Master File *Table*, *MFT*). В записях этой таблицы содержится вся *информация* о расположении файлов на томе, а небольшие файлы хранятся прямо в записях *MFT*.

Важной особенностью *NTFS* является то, что вся *информация*, как пользовательская, так и системная, хранится в виде файлов. Имена системных файлов начинаются со знака "$". Например, загрузочная *запись* тома содержится в файле $*Boot*, а главная *таблица* файлов – в файле $*Mft*. Такая организация информации позволяет единообразно работать как с пользовательскими, так и с системными данными на томе.

Поскольку *MFT* является важнейшей системной структурой, к которой при операциях с томом наиболее часто происходят обращения, выгодно хранить *файл* $*Mft* в непрерывной области логического диска, чтобы избежать его фрагментации (размещения в разных областях диска), и, следовательно, повысить скорость работы с ним. С этой целью при форматировании тома выделяется непрерывная область, называемая зоной *MFT* (*MFT* *Zone*). *По* мере увеличения главной таблицы файлов, *файл* $*Mft* расширяется, занимая зарезервированное *место* в зоне.

Остальное *место* на томе *NTFS* отводится под файлы – системные и пользовательские.

Рассмотрим более подробно структуру *MFT* ([рис.17.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=1#image.17.2)).

[](https://www.intuit.ru/EDI/19_02_19_1/1550528380-21796/tutorial/1143/objects/17/files/11-2.jpg)

[увеличить изображение](https://www.intuit.ru/EDI/19_02_19_1/1550528380-21796/tutorial/1143/objects/17/files/11-2.jpg)  
**Рис. 17.2.**Главная таблица файлов MFT

Главная *таблица* файлов *MFT* состоит из *множества* записей о файлах (файловых записей), расположенных на томе. Размер одной записи – 1 КБ (2 сектора). Самая первая *запись* в *MFT* – это *запись* о самом файле $*Mft*. Во второй записи содержится *информация* о файле $MftMirr – зеркальной копии *MFT*. В этом файле дублируются первые 4 записи таблицы *MFT*, в том числе *запись* о $*Mft*. В случае возникновения сбоя, если *MFT* окажется недоступной, *информация* о системных файлах будет считываться из $MftMirr (в загрузочной записи имеется *адрес* $MftMirr).

Перечислим следующие несколько записей в таблице *MFT* и кратко опишем назначение соответствующих системных файлов:

* $LogFile – файл журнала, в котором записывается информация о всех операциях, изменяющих структуру тома NTFS, например, создание файлов и каталогов. Файл журнала используется при восстановлении тома NTFS после сбоев;
* $Volume – файл информации о томе, в котором содержатся имя тома (Volume label), версия NTFS и набор флагов состояния тома, например, флаг (т. н. грязный бит, dirty bit), установка которого означает, что том был поврежден и требует восстановления при помощи системной утилиты Chkdsk;
* $AttrDef – таблица определения атрибутов (Attribute Definition Table), содержащая возможные на данном томе типы атрибутов файлов (см. далее);
* Root Directory (обозначается также обратным слешем "\") – файл с информацией о корневом каталоге тома. В нем хранятся ссылки на файлы и каталоги, содержащиеся в корневом каталоге;
* $BitMap – файл битовой карты (bitmap), каждый бит в этой карте соответствует кластеру на томе: если бит равен 1, кластер занят, иначе – свободен;
* $Boot – файл загрузочной записи тома;
* $BadClus – файл плохих кластеров (bad clusters), содержащий информацию обо всех кластерах, имеющих сбойные секторы (bad sectors).

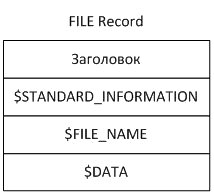
Кроме перечисленных, имеются и другие системные файлы *NTFS*, а в новых версиях появляются новые системные файлы.

Далее рассмотрим, что представляет собой *файл* в системе *NTFS*.

**Файлы NTFS**

Как уже обсуждалось, основная *информация* о файле содержится в файловой записи (File *Record*) размером 1 КБ таблицы *MFT*, а небольшие файлы целиком хранятся в файловой записи.

Файловая *запись* состоит из заголовка (*Header*) и набора атрибутов (*Attribute*). В заголовке содержится служебная *информация* о файловой записи, например, её тип и размер. Все данные, относящиеся непосредственно к файлу, хранятся в виде атрибутов. Названия атрибутов, так же как и системных файлов, начинаются с "$". Например, отдельными атрибутами являются *имя файла* ($FILE\_NAME), *информация* о его свойствах ($STANDARD\_INFORMATION), данные файла ($DATA). Типичная файловая *запись* представлена на [рис.17.3](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.3).



**Рис. 17.3.**Файловая запись

На диске файловая *запись* всегда расположена в начале сектора, первые байты файловой записи кодируют *слово* "FILE" (ASCII-коды: 46 49 4C 45). Конец записи определяется 4 байтовой последовательностью FF FF FF FF.

Физически *атрибут* файла хранится в виде потока байтов (*stream*) – простой последовательности байтов. Такое *представление* позволяет одинаковым образом работать с разнотипными атрибутами, а также добавлять нестандартные пользовательские атрибуты.

Каждый *атрибут* состоит из заголовка (*attribute* *header*), определяющего тип атрибута и его свойства, и тела (*attribute* *body*), содержащего основную информацию атрибута.

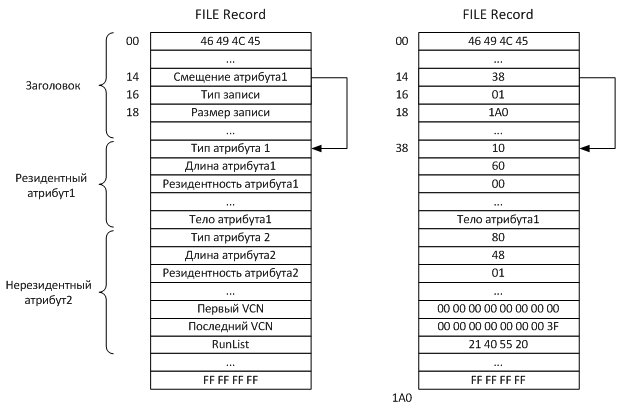
Более подробная структура файловой записи представлена на [рис.17.4](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.4).



**Рис. 17.4.**Структура файловой записи

*По* расположению относительно *MFT* атрибуты бывают резидентные и нерезидентные. Резидентные атрибуты (*resident* attributes) полностью помещаются в файловую *запись* *MFT*, нерезидентные атрибуты (nonresident attributes) хранятся вне *MFT*. Область, в которой расположен нерезидентный *атрибут*, называется группой (run). Поскольку нерезидентных атрибутов в файле может быть несколько, то и групп бывает тоже несколько. Множество групп файла называется списком групп (RunList). Файловая *запись* при наличии нерезидентных атрибутов содержит ссылку на расположение группы на диске (см. пример на [рис.17.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=1#image.17.2) "Главная *таблица* файлов *MFT*").

Некоторые поля заголовка файловой записи, а также резидентных и нерезидентных атрибутов представлены на [рис.17.5](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.5). На том же рисунке справа показан пример файловой записи с конкретными значениями рассматриваемых полей. Числа слева от полей записи обозначают шестнадцатеричное смещение поля от начала записи.



**Рис. 17.5.**Поля заголовка и атрибутов файловой записи

В начале файловой записи находится признак её начала – *слово* "FILE" (46 49 4C 45). *По* смещению 0x14 расположено двухбайтовое *поле*, в котором записано смещение первого атрибута относительно начала файловой записи. В примере в этом *поле* записано 38, т. е. первый *атрибут* расположен *по* смещению 38.

В следующем *поле* хранится тип файловой записи: *значение* 01 обозначает *файл*, 02 – каталог (*directory*). В примере файловая *запись* соответствует файлу (*значение* 01 *по* смещению 16).

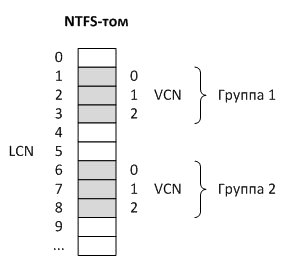
Ещё одно *поле* в заголовке содержит размер всей записи. В примере на [рис.17.5](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.5) в этом *поле* записано 1A0, т. е. размер записи составляет 416 *байт*.

Каждый *атрибут* имеет поля, указывающие тип, длину и резидентность атрибута. Все типы атрибутов имеют свои численные значения, например, атрибуту $FILE\_NAME соответствует *значение* 0x30, атрибуту $STANDARD\_INFORMATION – 0x10, атрибуту $DATA – 0x80.

Если *атрибут* резидентный, в *поле* резидентности записывается 0x00, иначе – 0x01. В случае нерезидентного атрибута предусмотрены поля для хранения номеров кластеров, в которых располагается *группа* или несколько групп, выделенных для размещения файла.

В примере на [рис.17.5](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.5) показаны два атрибута. Первый *атрибут* имеет тип $STANDARD\_INFORMATION (*значение* 10), *длина* атрибута 96 *байт* (6016 = 9610), *атрибут* является резидентным (00). У второго атрибута тип $*DATA* (80), *длина* – 72 байта (4816 = 7210), *атрибут* является нерезидентным (01).

Для обозначения кластеров используются два типа номеров: *LCN* и *VCN*. При помощи первого типа, *LCN* (Logical *Cluster* Number – *логический* номер кластера), нумеруются все кластеры на диске, от первого до последнего. *LCN* применяются, чтобы найти начальный *кластер* группы. Номера *VCN* (*Virtual* *Cluster* Number – виртуальный номер кластера) обозначают порядковый номер кластера внутри группы. Схема нумерации кластеров *LCN* *VCN* проиллюстрирована на [рис.17.6](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.6).



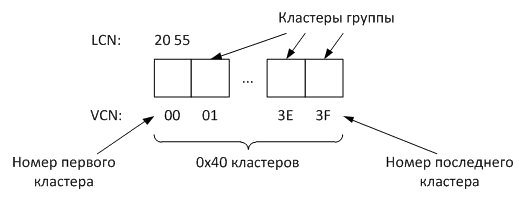
**Рис. 17.6.**Схема нумерации кластеров с использованием LCN VCN

В случае нерезидентных атрибутов в заголовке атрибута содержатся следующие поля: номер *VCN* первого кластера группы (обычно равен 0х00), номер *VCN* последнего кластера группы и *список* групп (RunList), описывающий расположение групп на диске.

Рассмотрим пример описания расположения групп, приведенный на [рис.17.5](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.5) (справа). В этом примере значения полей следующие:

* первый VCN = 0x00;
* последний VCN = 0x3F;
* список групп (RunList) = 0x21 40 55 20 00.

Расположение кластеров для данного примера приведено на [рис.17.7](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.7).



**Рис. 17.7.**Расположение кластеров группы для примера на рис. 17.5

В этом примере *значение* для списка групп

0x21 40 55 20 00

обозначает следующее:

* 0x21 – первый байт кодирует размер двух полей, которые за ним следуют:
  + младший полубайт обозначает размер поля (в байтах), в котором хранится длина группы в кластерах; в данном случае значение 1 указывает, что на длину группы отводится один байт;
  + старший полубайт обозначает размер поля (в байтах), в котором расположен номер LCN первого кластера группы; в данном случае значение 2 указывает на двухбайтовое поле;
* 0x40 – длина группы. Поскольку в первом байте размер поля длины группы определен в один байт, в качестве длины группы рассматриваем однобайтовое поле; в данном примере оно равно 0x40 (64 кластера);
* 0x2055 – LCN номер первого кластера. В первом байте размер поля номер первого кластера определен в два байта, поэтому в качестве LCN номера первого кластера рассматривается двухбайтовое поле, которое в примере равно 0x2055 (обратите внимание, байты на диске записываются в обратном порядке: сначала младшие – 55, затем старшие – 20);
* 0x00 – признак окончания описания списка групп.

Указанные обозначения проиллюстрированы на [рис.17.8](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.8).

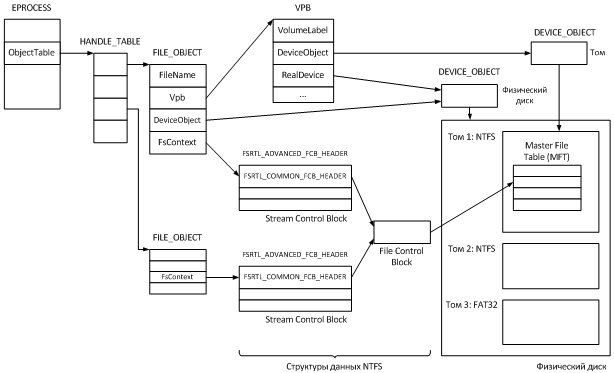


**Рис. 17.8.**Список группы

Отметим, что в рассмотренном примере нерезидентный *атрибут* содержится всего в одной группе; в общем случае групп может быть несколько.

**Структуры данных для управления файлами**

Рассмотрим структуры данных, задействованные при работе с файловой системой *NTFS* ([рис.17.9](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.9)).



**Рис. 17.9.**Структуры данных, связанные с NTFS

В структуре данных EPROCESS, описывающей процесс в *Windows* (см. лекцию 7 "Процессы и потоки"), имеется *поле* ObjectTable (*файл* base\ntos\inc\ps.h, строка 293), в котором содержится *указатель* на таблицу дескрипторов процесса типа HANDLE\_TABLE (*файл* base\ntos\inc\ex.h, строка 5179). В строках этой таблицы содержатся ссылки на ресурсы, открытые процессом, и в том числе, ссылки на объекты типа FILE\_OBJECT, которые упоминались в лекции 15 "Управление устройствами".

Структура FILE\_OBJECT (*файл* base\ntos\inc\io.h, строка 1763) содержит следующие основные поля:

* FileName – строковое имя файла;
* Vpb – указатель на структуру VPB, которая представляет том на устройстве внешней памяти;
* FsContext – указатель на блок управления потоком данных NTFS;
* DeviceObject – указатель на объект типа DEVICE\_OBJECT, связанный с физическим диском, на котором находится файл.

В структуре VPB (*Volume* *Parameter* *Block*) содержатся следующие поля (*файл* base\ntos\inc\io.h, строка 1288):

* VolumeLabelLength – размер тома в байтах;
* DeviceObject – указатель на объект типа DEVICE\_OBJECT, ассоциированный с данным томом (логическим диском);
* RealDevice – указатель на объект типа DEVICE\_OBJECT, ассоциированный с физическим устройством внешней памяти (физическим диском);
* SerialNumber – серийный номер тома;
* ReferenceCount – счетчик количества ссылок на структуру; если это поле не равно нулю, значит, структура кем-то используется;
* VolumeLabel – метка (строковое имя) тома. Максимальная длина метки определяется константой MAXIMUM\_VOLUME\_LABEL\_LENGTH, равной 32 двухбайтовым символам (см. файл base\ntos\inc\io.h, строка 1286).

На [рис.17.9](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.9) изображен физический *диск*, разбитый на три тома (логических диска) – два тома *NTFS* и один том *FAT32*. *Поле* DeviceObject структуры VPB указывает на *объект* DEVICE\_OBJECT, который ссылается на первый из томов *NTFS*. *Поле* RealDevice структуры VPB указывает на *объект* DEVICE\_OBJECT, связанный с физическим диском.

С каждым файлом *NTFS*, с которым *операционная система* в данный момент работает, связана *структура данных*, называемая блок управления файлом (File *Control* *Block*, *FCB*). В этой структуре хранится *указатель* на *запись* в таблице *MFT* для данного файла (см. [рис.17.9](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.9)).

Поскольку в файле существует в общем случае несколько потоков, для каждого потока создается своя *структура данных* – блок управления потоком (*Stream* *Control* *Block*, SCB), в котором содержится *ссылка* на *FCB*. *Поле* FsContext структуры FILE\_OBJECT указывает на SCB для открытого потока, таким образом, структура FILE\_OBJECT на самом деле связана с потоком, а не с файлом. Чтобы открыть другой *поток* в том же файле требуется создавать новый *объект* FILE\_OBJECT. На [рис.17.9](https://www.intuit.ru/studies/courses/10471/1078/lecture/16586?page=2#image.17.9) показана ситуация, когда процесс открыл два разных потока одного и того же файла. В WRK можно найти описание блока управления потоком SCB – структуру FSRTL\_ADVANCED\_FCB\_HEADER (*файл* base\ntos\inc\fsrtl.h, строка 120) вместе с её основным полем, представляющим структуру FSRTL\_COMMON\_FCB\_HEADER (тот же *файл*, строка 65).

**Резюме**

В этой лекции приведен обзор файловых систем, поддерживаемых *Windows*, и подробно рассматривается основная *файловая система* *Windows* – *NTFS*. Дается *определение* базовым понятиям – *диск*, раздел, том, сектор, *кластер*. Перечисляются возможности *NTFS*. Описывается структура *NTFS* тома, особое внимание уделяется главной таблице файлов *MFT*. Рассматриваются виды и структура файловых записей *MFT*. В заключение приводятся структуры данных *Windows* Research *Kernel*, связанные с файловой системой *NTFS*.