Управление вводом-выводом, файловой системой и безопасностью в ОС Windows. Часть 2. Ввод-вывод и внешняя память

Лекция

Ревизия: 0.2

Последняя модификация: Vlad Kovtun

Дата последней модификации: 03.04.2015 15:17:00

История изменений

28.09.2010 – Версия 0.1. Первичный документ. Ковтун В.Ю.

13.09.2014 – Версия 0.2. Общая редакция. Ковтун В.Ю.

Содержание

Лекция 12. Управление вводом-выводом, файловой системой и безопасность в ОС Windows. Часть 2.

Вопросы

1. Компоненты подсистемы ввода-вывода.
2. Драйверы устройств.
3. Обработка ввода-вывода.
4. Диспетчер Plug & Play.
5. Диспетчер электропитания.
6. Драйверы дисков.
7. Управление томами.

Компоненты подсистемы ввода-вывода

Согласно целям, поставленным при разработке, подсистема ввода-вывода в Windows должна обеспечивать приложениям абстракцию устройств — как аппаратных (физических), так и программных (виртуальных или логических) — и при этом предоставлять следующую функциональность:

- стандартные средства безопасности и именования устройств для защиты разделяемых ресурсов;
- высокопроизводительный асинхронный пакетный ввод-вывод для поддержки масштабируемых приложений;
- сервисы для написания драйверов устройств на высокоуровневом языке и упрощения их переноса между разными аппаратными платформами;
- поддержку многоуровневой модели и расширяемости для добавления драйверов, модифицирующих поведение других драйверов или устройств без внесения изменений в них;
- динамическую загрузку и выгрузку драйверов устройств, чтобы драйверы можно было загружать по требованию и не расходовать системные ресурсы без необходимости;
- поддержку Plug and Play, благодаря которой система находит и устанавливает драйверы для нового оборудования, а затем выделяет им нужные аппаратные ресурсы;
- управление электропитанием, чтобы система и отдельные устройства могли переходить в состояния с низким энергопотреблением;
- поддержку множества устанавливаемых файловых систем, в том числе FAT, CDFS (файловую систему CD-ROM), UDF (Universal Disk Format) и NTFS;
- поддержку Windows Management Instrumentation (WMI) и средств диагностики, позволяющую управлять драйверами и вести мониторинг за ними через WMI-приложения и сценарии.

Для реализации этой функциональности подсистема ввода-вывода в Windows состоит из нескольких компонентов исполнительной системы и драйверов устройств (Рис. 1).

- Центральное место в этой подсистеме занимает диспетчер ввода-вывода; он подключает приложения и системные компоненты к виртуальным, логическим и физическим устройствам, а также определяет инфраструктуру, поддерживающую драйверы устройств.
- Драйвер устройства, как правило, предоставляет интерфейс ввода-вывода для устройств конкретного типа. Такие драйверы принимают от диспетчера ввода-вывода команды, предназначенные управляемым ими устройствам, и уведомляют диспетчер ввода-вывода о выполнении этих команд. Драйверы часто используют этот диспетчер для пересылки команд ввода-вывода другим драйверам, задействованным в реализации интерфейса того же устройства и участвующим в управлении им.
- Диспетчер PnP работает в тесном взаимодействии с диспетчером ввода-вывода и **драйвером шины** (bus driver) - одной из разновидностей драйверов устройств. Он управляет выделением аппаратных ресурсов, а также распознает устройства и реагирует на их подключение или отключение. Диспетчер PnP и драйверы шины отвечают за загрузку соответствующего драйвера при обнаружении нового устройства. Если устройство добавляется в систему, в которой нет нужного драйвера устройства, компоненты исполнительной системы, отвечающие за

поддержку PnP, вызывают сервисы установки устройств, поддерживаемые диспетчером PnP пользовательского режима.

- Диспетчер электропитания, также в тесном взаимодействии с диспетчером ввода-вывода, управляет системой и драйверами устройств при их переходе в различные состояния энергопотребления.
- Процедуры поддержки Windows Management Instrumentation (WMI) (Инструментарий управления Windows), образующие провайдер WDM (Windows Driver Model) WMI, позволяют драйверам устройств выступать в роли провайдеров, взаимодействуя со службой WMI пользовательского режима через провайдер WDM WMI.
- Реестр служит в качестве базы данных, в которой хранится описание основных устройств, подключенных к системе, а также параметры инициализации драйверов и конфигурационные настройки.

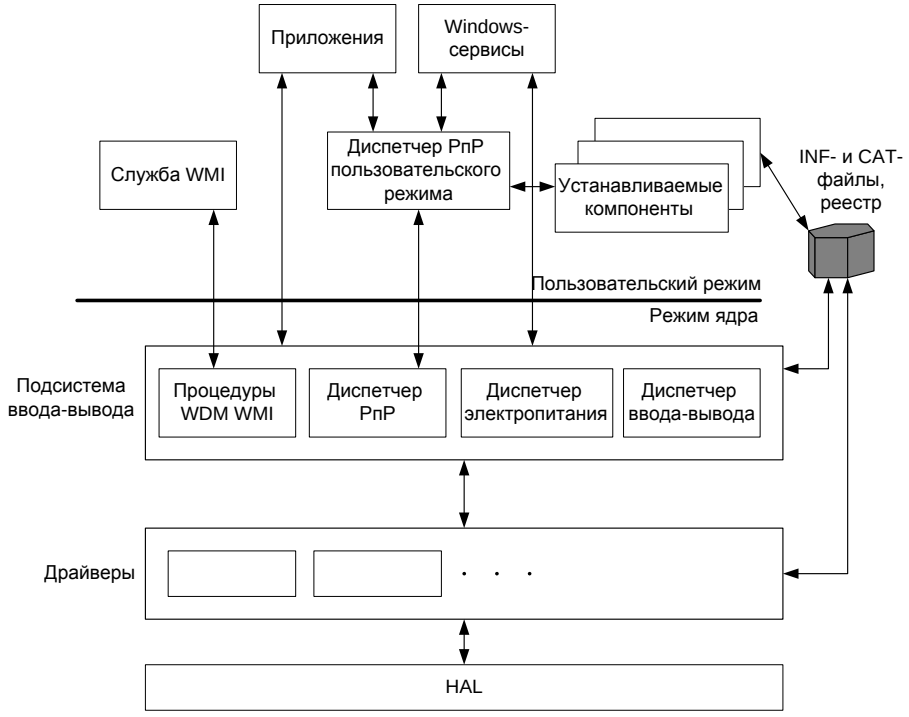

Рис. 1. Компоненты подсистемы ввода-вывода

- Для установки драйверов используются INF-файлы; они связывают конкретное аппаратное устройство с драйвером, который берет на себя ведущую роль в управлении этим устройством. Содержимое INF-файла состоит из инструкций, описывающих соответствующее устройство, исходное и целевое местонахождение файлов драйвера, изменения, которые нужно внести в реестр при установке драйвера, и информацию о зависимостях драйвера. В CAT-файлах хранятся цифровые подписи, которые удостоверяют файлы драйверов, прошедших испытания в лаборатории **Microsoft Windows Hardware Quality Lab** (WHQL).
- **Уровень абстрагирования от оборудования** (HAL) изолирует драйверы от специфических особенностей конкретных процессоров и контроллеров прерываний, поддерживая API, скрывающие межплатформенные различия. В сущности HAL является драйвером шины для тех устройств на материнской плате компьютера, которые не контролируются другими драйверами.

Диспетчер ввода-вывода

Диспетчер ввода-вывода (I/O manager) определяет модель доставки запросов на ввод-вывод драйверам устройств. Подсистема ввода-вывода управляется пакетами. Большинство запросов ввода-вывода представляется **пакетами запросов ввода-вывода** (I/O request packets, IRP), передаваемых от одного компонента подсистемы ввода-вывода другому. **Подсистема ввода-вывода** позволяет индивидуальному потоку приложения управлять сразу несколькими запросами на ввод-вывод. IRP — это структура данных, которая содержит информацию, полностью описывающую запрос ввода-вывода.

Функции диспетчера ввода-вывода:

- Создание пакетов-запроса ввода-вывода (IRP) и направление их соответствующему драйверу, а также перенаправление пакетов запроса ввода-вывода между драйверами.
- Удаление и освобождение пакетов запроса ввода-вывода после завершения операции ввода-вывода. Взаимодействие с диспетчером кэша и другими компонентами NT Executive.
- Взаимодействие с диспетчером виртуальной памяти для предоставления файловым системам функций ввода-вывода с записью данных в память.
- Мониторинг загруженных файловых систем и их вызов по требованию.
- Предоставление поддержки синхронного и асинхронного ввода-вывода. Асинхронный ввод-вывод особенно важен для приложений хранения данных. Например, приложение резервного копирования может использовать асинхронный ввод-вывод для размещения в очереди нескольких запросов, что позволяет полностью загрузить устройство записи на ленту.
- Управление буферами для операций ввода-вывода.

Типичная обработка ввода-вывода

Большинство операций ввода-вывода не требует участия всех компонентов подсистемы ввода-вывода. Как правило, запрос на ввод-вывод выдается приложением, выполняющим соответствующую операцию (например, чтение данных с устройства); такие операции обрабатываются диспетчером ввода-вывода, одним или несколькими драйверами устройств и HAL.

Как уже упоминалось, в Windows потоки выполняют операции ввода-вывода над виртуальными файлами. ОС абстрагирует все запросы на ввод-вывод, скрывая тот факт, что конечное устройство ввода-вывода может и не быть устройством с файловой структурой. Эти позволяет обобщить интерфейс между приложениями и устройствами. Приложения пользовательского режима (к какой бы подсистеме они ни относились — Windows или POSIX) вызывают документированные функции, которые в свою очередь обращаются к внутренним функциям подсистемы ввода-вывода для чтения/записи файла и для выполнения других операции-запросы, адресованные виртуальным файлам, диспетчер ввода-вывода динамически направляет соответствующим драйверам устройств. Базовая схема обработки запроса на ввод-вывод показана на Рис. 2.

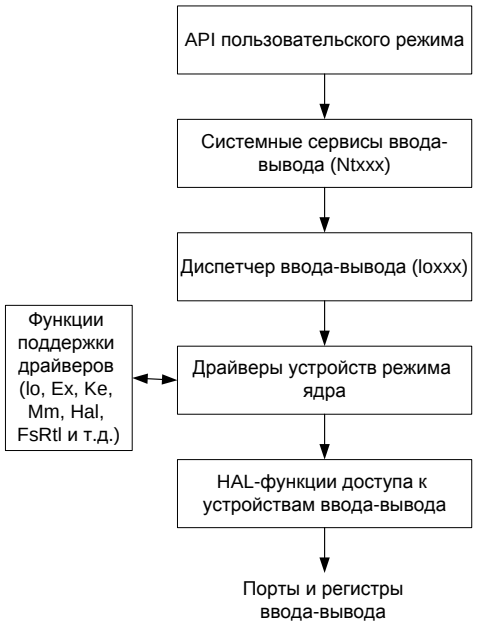

Рис. 2. Схема обработки типичного запроса на ввод-вывод

Драйверы устройств

Для интеграции с диспетчером ввода-вывода и другими компонентами подсистемы ввода-вывода драйвер устройства должен быть написан в соответствии с правилами, специфичными для управляемого им типа устройств и для его роли в управлении такими устройствами. Здесь мы познакомимся с типами драйверов устройств, поддерживаемых Windows, и исследуем внутреннюю структуру драйвера устройства.

Типы драйверов устройств

Драйверы могут работать в двух режимах: в пользовательском или в режиме ядра. Windows поддерживает несколько типов драйверов пользовательского режима:

- **Драйверы виртуальных устройств** (virtual device drivers, VDD). Используются для эмуляции 16-разрядных программ MS-DOS. Они перехватывают обращения таких программ к портам ввода-вывода и транслируют их в вызовы Windows-функций ввода-вывода, передаваемые реальным драйверам устройств. Поскольку Windows является полностью защищенной ОС, программы MS-DOS пользовательского режима не могут напрямую обращаться к аппаратным средствам — они должны делать это через драйверы устройств режима ядра.
- **Драйверы принтеров**. Драйверы подсистемы Windows, которые транслируют аппаратно-независимые запросы на графические операции в команды, специфичные для принтера. Далее эти команды обычно направляются драйверу режима ядра, например драйверу параллельного порта (Parport.sys) или драйверу порта принтера на USB-шине (Usb-print.sys).

Далее будут рассматриваться драйвера устройств, работающих в режиме ядра. Эти драйверы можно разбить на несколько основных категорий.

- **Драйверы файловой системы**. Принимают запросы на ввод-вывод и выполняют их, выдавая более специфические запросы драйверам устройств массовой памяти или сетевым драйверам.
- **PnP-драйверы**. Драйверы, работающие с оборудованием и интегрируемые с диспетчерами электропитания и PnP. В их число входят драйверы для устройств массовой памяти, видеоадаптеров, устройств ввода и сетевых адаптеров.
- Драйверы, не отвечающие спецификации Plug and Play — также называются **расширениями ядра**. Расширяют функциональность системы, предоставляя доступ из пользовательского режима к сервисам и драйверам режима ядра. Они не интегрируются с диспетчерами PnP и электропитания. К ним, в частности, относятся драйверы протоколов и сетевого API. Категория драйверов режима ядра подразделяется на группы в зависимости от модели, на которой они основаны, и их роли в обслуживании запросов к устройствам.

WDM-драйверы

Это драйверы устройств, отвечающие спецификации **Windows Driver Model** (WDM). WDM требует от драйверов поддержки управления электропитанием, Plug and Play и WMT. Большинство драйверов Plug and Play построены как раз на модели WDM. Эта модель реализована в Windows, Windows 98 и Windows Millennium Edition, поэтому WDM-драйверы этих ОС совместимы на уровне исходного кода, а во многих случаях и на уровне двоичного кода. Существует три типа WDM-драйверов.

- **Драйверы шин**. Управляют логическими или физическими шинами. Примеры шин — PCMCIA, PCI, USB, IEEE 1394, ISA. Драйвер шины отвечает за распознавание устройств, подключенных к управляемой им шине, оповещение о них диспетчера PnP и управление параметрами электропитания шины.
- **Функциональные драйверы**. Управляют конкретным типом устройств. Драйверы шин представляют устройства функциональным драйверам через диспетчер PnP. Функциональным считается драйвер, экспортирующий рабочий интерфейс устройства ОС.
- **Драйверы фильтров.** Занимающие более высокий логический уровень, чем функциональные драйверы, они дополняют функциональность или изменяют поведение устройства либо другого драйвера. Так, утилиту для перехвата ввода с клавиатуры **можно реализовать** в виде драйвера фильтра клавиатуры, расположенного на более высоком уровне, чем функциональный драйвер клавиатуры.

Многоуровневые драйверы

Кроме WDM-драйверов шин, функциональных драйверов и драйверов фильтров, оборудованием могут поддерживать следующие компоненты:

- **Драйверы классов устройств** (class drivers). Реализуют обработку ввода-вывода для конкретного класса устройств, например дисковых устройств, ленточных накопителей или приводов CD-ROM, где аппаратные интерфейсы стандартизированы и один драйвер может обслуживать аналогичные устройства от множества производителей.

- **Порт-драйверы** (port drivers). Обрабатывают запросы на ввод-вывод, специфичные для определенного типа порта ввода-вывода, например SCSI. Порт-драйверы реализуются как библиотеки функций режима ядра, а не как драйверы устройств.
- **Минипорт-драйверы** (miniport drivers). Преобразуют универсальные запросы ввода-вывода к порту конкретного типа в запросы, специфичные для адаптера конкретного типа, например для SCSI-адаптера. Минипорт-драйверы являются истинными драйверами устройств, которые импортируют функции, предоставляемые порт-драйвером.

Вот пример, который демонстрирует, как работают драйверы устройств. Драйвер файловой системы принимает запрос на запись данных в определенное место конкретного файла. Он преобразует его в запрос на запись определенного числа байтов по определенному «логическому» адресу на диске. После этого он передает этот запрос (через диспетчер ввода-вывода) простому драйверу диска. Последний в свою очередь преобразует запрос в физический адрес на диске (цилиндр/дорожка/сектор) и позиционирует головки дискового устройства для записи данных. Эта схема действий показана на Рис. 3.

Эта схема иллюстрирует разделение труда между двумя драйверами. Диспетчер ввода-вывода получает запрос на запись, в котором адрес записи относителен началу конкретного файла. Далее диспетчер ввода-вывода передает запрос драйверу файловой системы, который преобразует информацию запроса в адрес начала записи на диске и число байтов, которые нужно записать на диск. Драйвер файловой системы передает через диспетчер ввода-вывода запрос драйверу диска, который транслирует его в физический адрес на диске и переносит нужные данные.

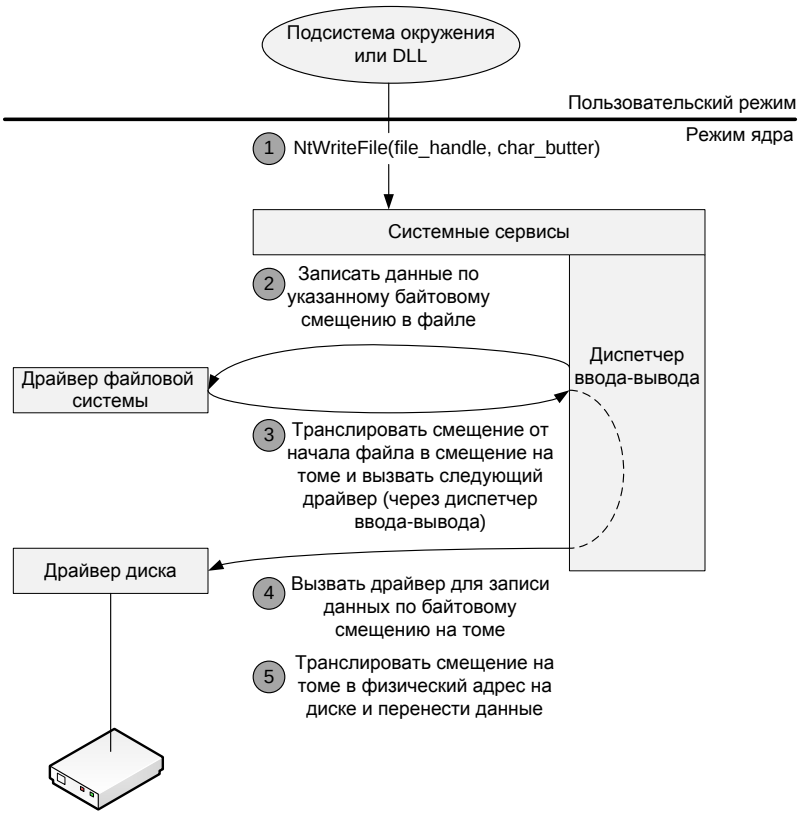

Рис. 3. Взаимодействие драйвера файловой системы и драйвера диска

Поскольку все драйверы — и устройств, и файловой системы — предоставляют ОС одинаковую инфраструктуру, в их иерархию легко добавить еще один драйвер, не изменяя существующие драйверы или подсистему ввода-вывода. Например, введя соответствующий драйвер, можно логически представить несколько дисков как один большой диск. Такой драйвер, кстати, имеется в Windows — он обеспечивает поддержку отказоустойчивых дисков. Драйвер диспетчера томов вполне логично размещается между драйверами файловой системы и дисков, как показано на рис. 4.

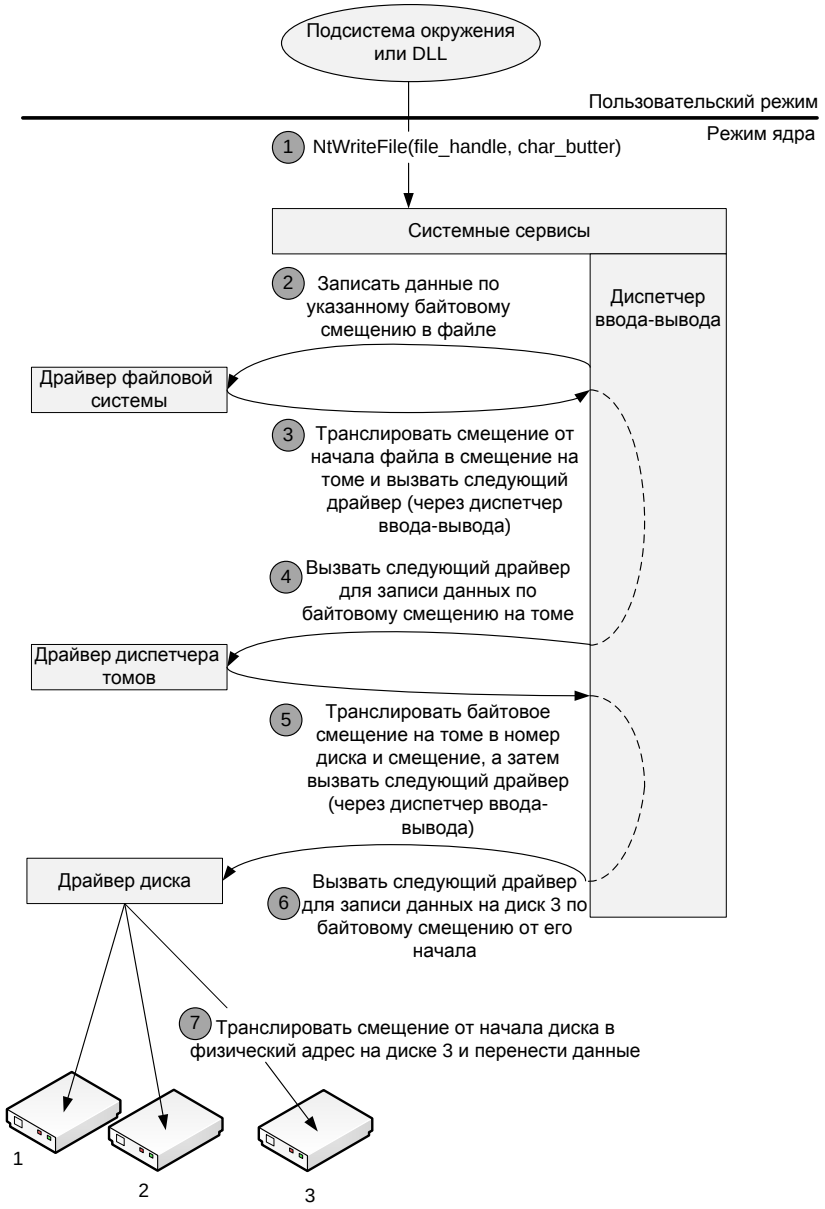

Рис. 4. Добавление промежуточного драйвера

Структура драйвера

Выполнением драйверов устройств управляет подсистема ввода-вывода. Драйвер устройства состоит из набора процедур, вызываемых на различных этапах обработки запроса ввода-вывода. Основные процедуры драйвера показаны на Рис. 5.

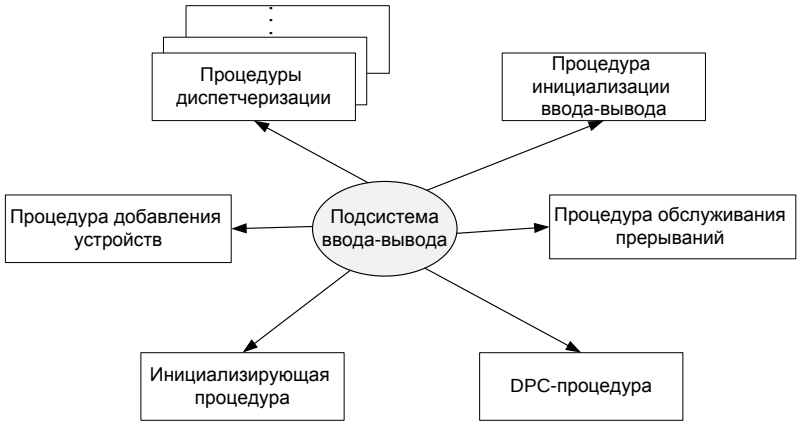

Рис. 5. Основные процедуры драйвера устройства

- **Инициализирующая процедура**. Диспетчер ввода-вывода выполняет инициализирующую процедуру драйвера (которая обычно называется **DriverEntry**) при загрузке этого драйвера в ОС. Данная процедура регистрирует остальные процедуры драйвера в диспетчере ввода-вывода, заполняя соответствующей информацией системные структуры данных, и выполняет необходимую глобальную инициализацию драйвера.
- **Процедура добавления устройства**. Такие процедуры реализуются в драйверах, поддерживающих Plug and Play. Через эту процедуру диспетчер PnP посылает драйверу уведомление при обнаружении устройства, за которое отвечает данный драйвер. Выполняя эту процедуру, драйвер обычно создает объект «устройство», представляющий аппаратное устройство.
- **Процедуры диспетчеризации.** Это основные функции, предоставляемые драйвером устройства, например для открытия, закрытия, чтения, записи и реализации других возможностей устройства, файловой системы или сети. Диспетчер ввода-вывода, вызванный для выполнения операции ввода-вывода, генерирует IRP и обращается к драйверу через одну из его процедур диспетчеризации.
- **Процедура инициации ввода-вывода**. С помощью этой процедуры драйвер может инициировать передачу данных, как на устройство, так и с него. Эта процедура определяется лишь в драйверах, использующих поддержку диспетчера ввода-вывода для помещения входящих запросов в очередь. Диспетчер ввода-вывода ставит в очередь IRP для драйвера, гарантируя одновременную обработку им только одного IRP. Большинство драйверов обрабатывают сразу несколько IRP, но создание очереди имеет смысл для некоторых драйверов, в частности для драйвера клавиатуры.
- **Процедура обслуживания прерываний** (ISR). Когда устройство генерирует прерывание, диспетчер прерываний ядра передает управление этой процедуре. В модели ввода-вывода Windows процедуры ISR работают на уровне **DIRQL** (Device IRQL), поэтому они выполняют минимум действий во избежание, слишком продолжительной блокировки прерываний более низкого уровня.
- **DPC-процедура обработки прерываний**. DPC-процедура выполняет основную часть обработки прерывания, оставшуюся после выполнения ISR. Она работает при более низком IRQL (уровня «PPC/dispatch») чем ISR, чтобы не блокировать без необходимости другие прерывания. DPC-процедура инициирует завершение текущей операции ввода-вывода и выполнение следующей операции ввода-вывода из очереди на данном устройстве. У многих драйверов устройств имеются процедуры, не показанные на Рис. 5.
- **Одна или несколько процедур завершения ввода-вывода.** У драйвера могут быть процедуры завершения ввода-вывода, уведомляющие его об окончании обработки IRP драйвером более низкого уровня.
- **Процедура отмены ввода-вывода.** Если операция ввода-вывода может быть отменена, драйвер определяет одну или более процедур отмены ввода-вывода. Получив IRP для запроса ввода-вывода, который может быть отменен, драйвер связывает с IRP процедуру отмены. Если поток, выдавший запрос на ввод-вывод, завершается до окончания обработки запроса или отменяет операцию (например, вызовом Windows-функции CancelIo), диспетчер ввода-вывода выполняет процедуру отмены, связанную с IRP (если таковая есть). Процедура отмены отвечает за выполнение любых действий, необходимых для освобождения всех ресурсов, выделенных при обработке IRP, а также за завершение IRP со статусом отмены.
- **Процедура выгрузки**. Эта процедура освобождает все системные ресурсы, задействованные драйвером, после чего диспетчер ввода-вывода может удалить их из памяти. При выполнении процедуры выгрузки обычно освобождаются ресурсы, выделенные процедурой инициализации. Драйвер может загружаться и выгружаться во время работы системы.
- **Процедура уведомления о завершении работы системы**. Эта процедура позволяет драйверу проводить очистку при завершении работы системы.
- **Процедуры регистрации ошибок**. При возникновении неожиданных ошибок (например, когда на диске появляется поврежденный блок), процедуры регистрации ошибок, принадлежащие драйверу, уведомляют о них диспетчер ввода-вывода. Последний записывает эту информацию в файл журнала ошибок.

Объекты «драйвер» и «устройство»

Когда поток открывает описатель объекта «файл» (этот процесс описывается в разделе «Обработка ввода-вывода» далее в этой главе), диспетчер ввода-вывода, исходя из

имени этого объекта, должен определить, к какому драйверу (или драйверам) нужно обратиться для обработки запроса. Более того, диспетчер ввода-вывода должен знать, где найти эту информацию, когда в следующий раз поток вновь воспользуется тем же описателем файла. Для этого предназначены следующие объекты.

- **Объект «драйвер»**, представляющий отдельный драйвер в системе, именно от этого объекта диспетчер ввода-вывода получает адрес процедуры диспетчеризации (точки входа) драйвера.
- **Объект «устройство»**, представляющий физическое или логическое устройство в системе и описывающий его характеристики, например границы выравнивания буферов и адреса очередей для приема IRP, поступающих на это устройство.

Диспетчер ввода-вывода создает объект «драйвер» при загрузке в систему соответствующего драйвера и вызывает его инициализирующую процедуру (например, `DriverEntry`), которая записывает в атрибуты объекта точки входа этого драйвера.

После загрузки драйвер может создавать объекты «устройство» для представления устройств или даже для формирования интерфейса драйвера (вызовом `IoCreateDevice` или `IoCreateDeviceSecure`). Однако большинство PnP-драйверов создают объекты «устройство» с помощью своих процедур добавления устройств, когда диспетчер PnP информирует их о присутствии управляемого ими устройства. С другой стороны, драйверы, не отвечающие спецификации Plug and Play, создают объекты «устройство» при вызове диспетчером ввода-вывода их инициализирующих процедур. Диспетчер ввода-вывода выгружает драйвер после удаления его последнего объекта «устройство», когда ссылок на устройство больше нет.

Создавая объект «устройство», драйвер может присвоить ему имя. Тогда этот объект помещается в пространство имен диспетчера объектов. Драйвер может определить имя этого объекта явно или позволить диспетчеру ввода-вывода сгенерировать его автоматически. По соглашению объекты «устройство» помещаются в каталог `\Device` пространства имен, недоступный приложениям через Windows API.

Чтобы сделать объект «устройство» доступным для приложений драйвер должен создать в каталоге `\Global??` (или в каталоге `\??` в Windows 2000) символьную ссылку на имя этого объекта в каталоге `\Device`. Драйверы, не поддерживающие Plug and Play, и драйверы файловой системы обычно создают символьную ссылку с общеизвестным именем (скажем, `\Device\Hardware2`). Поскольку общеизвестные имена не срабатывают в средах с динамически меняющимся составом оборудования, PnP-драйверы предоставляют один или несколько интерфейсов через функцию `IoRegisterDeviceInterface`, передавая ей GUID, определяющий тип предоставляемой функциональности. GUID являются 128-битными числами, которые можно генерировать с помощью утилиты `Guidgen`, входящей в состав DDK и Platform SDK. Диапазон чисел, который может быть представлен 128 битами, гарантирует, что каждый GUID, созданный этой утилитой, всегда будет глобально уникальным.

`IoRegisterDeviceInterface` определяет символьную ссылку, сопоставляемую с экземпляром объекта «устройство». Однако прежде чем диспетчер ввода-вывода действительно создаст ссылку, драйвер должен вызвать функцию `IoSetDeviceInterfaceState`, чтобы разрешить использование интерфейса этого устройства. Обычно драйвер делает это, когда диспетчер PnP посылает ему команду `start-device` для запуска устройства.

Приложение, которому нужно открыть объект «устройство», представленный GUID-идентификатором, может вызывать PnP-функции настройки, например `SetupDiEnumDeviceInterfaces` для перечисления интерфейсов, доступных по конкретному GUID, и получения имен символьных ссылок, с помощью которых может быть открыт объект «устройство». Чтобы получить дополнительную информацию (например, автоматически сгенерированное имя устройства), приложение вызывает функцию `SetupDiGetDeviceInterfaceDetail` для всех устройств, перечисленных `SetupDiEnumDeviceInterfaces`. Получив от `SetupDiGetDeviceInterfaceDetail` имя устройства, приложение обращается к Windows-функции `CreateFile`, чтобы открыть устройство и получить его описатель.

Как видно на Рис. 6, объект «устройство» ссылается на свой объект «драйвер», благодаря чему диспетчер ввода-вывода знает, из какого драйвера нужно вызвать процедуру при получении запроса ввода-вывода. С помощью объекта «устройство» он находит объект «драйвер», который представляет драйвер, обслуживающий

устройство. После этого он обращается к объекту «драйвер», используя номер функции из исходного запроса; каждый номер функции соответствует точке входа драйвера.

С объектом «драйвер» нередко сопоставляется несколько объектов «устройство». Список объектов «устройство» представляет физические и логические устройства, управляемые драйвером. Так, для каждого раздела жесткого диска имеется отдельный объект «устройство» с информацией, специфичной для данного раздела. Но для обращения ко всем разделам используется один и тот же драйвер жесткого диска. При выгрузке драйвера из системы диспетчер ввода-вывода с помощью очереди объектов «устройство» определяет устройства, на которые повлияет удаление драйвера.

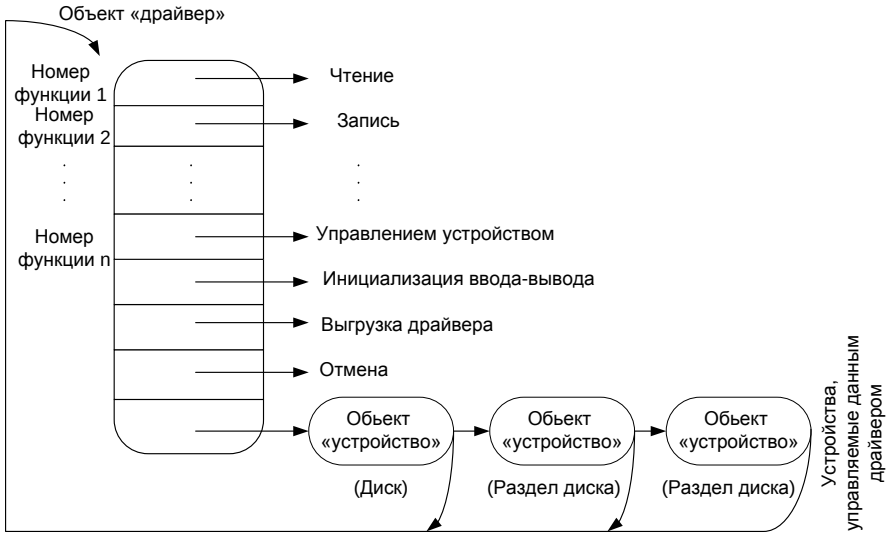

Рис. 6. Объект «драйвер»

Использование объектов для регистрации информации о драйверах означает, что диспетчеру ввода-вывода не нужно знать никаких деталей реализации драйверов. Он просто находит драйвер по указателю, тем самым позволяя легко загружать новые драйверы и обеспечивая их переносимость. Кроме того, представление устройств и драйверов разными объектами упрощает подсистеме ввода-вывода закрепление драйверов за дополнительными устройствами, которые появляются при изменении конфигурации системы.

Открытие устройств

Объекты «файл» являются структурами режима ядра, которые точно соответствуют определению объектов в Windows: это системные ресурсы, доступные для совместного использования двум или нескольким процессам, у них могут быть имена, их безопасность обеспечивается моделью защиты объектов, и они поддерживают синхронизацию.

Объекты «файл» — представление ресурсов в памяти, которое обеспечивает чтение и запись данных в эти ресурсы. Описание и размеры полей см. в определении структуры `FILE_OBJECT` в `Ntddk.h`.

Когда поток открывает файл или простое устройство, диспетчер ввода-вывода возвращает описатель объекта «файл». Рис. 7 иллюстрирует, что происходит при открытии файла.

В этом примере С-программа (1) вызывает из стандартной библиотеки функцию `fopen`, которая в свою очередь вызывает Windows-функцию `CreateFile` (2). Далее DLL, подсистемы Windows (в данном случае — `Kernel2.dll`) вызывает функцию `NtCreateFile` из Ntdll.dll (3). Эта функция в Ntdll.dll содержит соответствующие команды, вызывающие переход в режим ядра в диспетчер системных сервисов, который и обращается к настоящей функции `NtCreateFile` (4) из `Ntoskrnl.exe`.

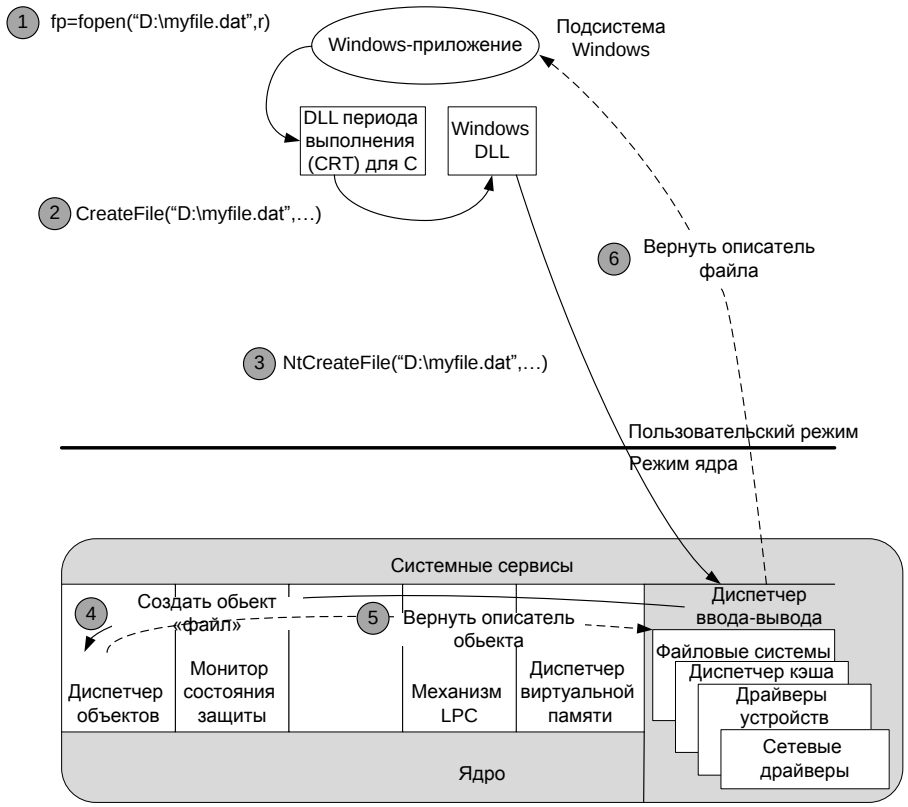

Рис. 7. Открытие объекта «файл»

ПРИМЕЧАНИЕ. Объекты «файл» представляют открытые экземпляры файлов, а не сами файлы. В отличие от UNIX-систем, где используются `vnode`, в Windows представление файла не определено — системные драйверы Windows определяют свои представления.

Как и другие объекты исполнительной системы, файлы защищаются дескрипторами защиты, которые содержат список управления доступом (ACL). Чтобы выяснить, позволяет ли ACL файла получить процессу доступ того типа, который был запрошен его потоком, диспетчер ввода-вывода обращается к системе защиты. Если да, диспетчер объектов (5,6) разрешает доступ и сопоставляет предоставленные права доступа с описателем файла, возвращаемым потоку. Если этому или другому потоку того же процесса понадобятся дополнительные операции с файлом, не указанные в исходном запросе, ему придется открыть новый описатель, по которому тоже будет проведена проверка прав доступа.

Поскольку объект «файл» — представление разделяемого ресурса в памяти, а не сам ресурс, он отличается от остальных объектов исполнительной системы. Объект «файл» содержит лишь данные, уникальные для описателя объекта, тогда как собственно файл — совместно используемые данные или текст. Всякий раз, когда поток открывает описатель файла, создается новый объект «файл» с новым набором атрибутов, специфичным для этого описателя.

Описатель файла уникален для процесса, но определяемый им физический ресурс — нет. Поэтому, как и при доступе к любому другому общему ресурсу, потоки должны синхронизировать свое обращение к совместно используемым файлам, каталогам и устройствам.

Имя открываемого файла включает имя объекта «устройство», который представляет устройство, содержащее файл. Например, `\Device\Floppy0\Myfile.dat` ссылается на файл `Myfile.dat` на дискете в дисководе A. Подстрока «`\Device\Floppy0`» является внутренним именем объекта «устройство», представляющего данный дисковод. При открытии файла `Myfile.dat` диспетчер ввода-вывода создает объект «файл», сохраняет в нем указатель на объект «устройство» `Floppy0` и возвращает описатель файла вызывающему потоку. Впоследствии, когда вызывающий поток воспользуется описателем файла, диспетчер ввода-вывода сможет обращаться непосредственно к объекту `Floppy0`.

Обработка ввода-вывода

Типы ввода-вывода

Синхронный и асинхронный ввод-вывод

Большинство операций ввода-вывода приложений являются синхронными, т.е. приложение ждет, когда устройство выполнит передачу данных и вернет код статуса по завершении операции ввода-вывода. После этого программа продолжает работу и немедленно использует полученные данные. В таком простейшем варианте Windows-функции `ReadFile` и `WriteFile` выполняются синхронно. Перед возвратом управления они должны завершить операцию ввода-вывода.

Асинхронный ввод-вывод позволяет приложению выдать запрос на ввод-вывод и продолжить выполнение, не дожидаясь передачи данных устройством. Этот тип ввода-вывода увеличивает эффективность работы приложения, позволяя заниматься другими задачами, пока выполняется операция ввода-вывода. Для использования асинхронного ввода-вывода вы должны указать при вызове `CreateFile` флаг `FILE_FLAG_OVERLAPPED`. Конечно, инициировав операцию асинхронного ввода-вывода, поток должен соблюдать осторожность и не обращаться к запрошенным данным до их получения от устройства. Следовательно, поток должен синхронизировать свое выполнение с завершением обработки запроса на ввод-вывод, отслеживая описатель синхронизирующего объекта (которым может быть событие, порт завершения ввода-вывода или сам объект «файл»), который по окончании ввода-вывода перейдет в свободное состояние.

Независимо от типа запроса операции ввода-вывода, инициированные драйвером в интересах приложения, выполняются асинхронно, т.е. после выдачи запроса драйвер устройства возвращает управление подсистеме ввода-вывода. А когда она вернет управление приложению, зависит от типа запроса. Схема управления при инициации операции чтения показана на Рис. 8. Ожидание зависит от состояния флага перекрытия в объекте «файл» и реализуется функцией `NtReadFile` в режиме ядра.

Проверить статус незавершенной операции асинхронного ввода-вывода вызовом Windows-функции `HasOverlappedloCompleted`. При использовании портов завершения ввода-вывода с той же целью можно вызывать `GetQueuedCompletionStatus`.

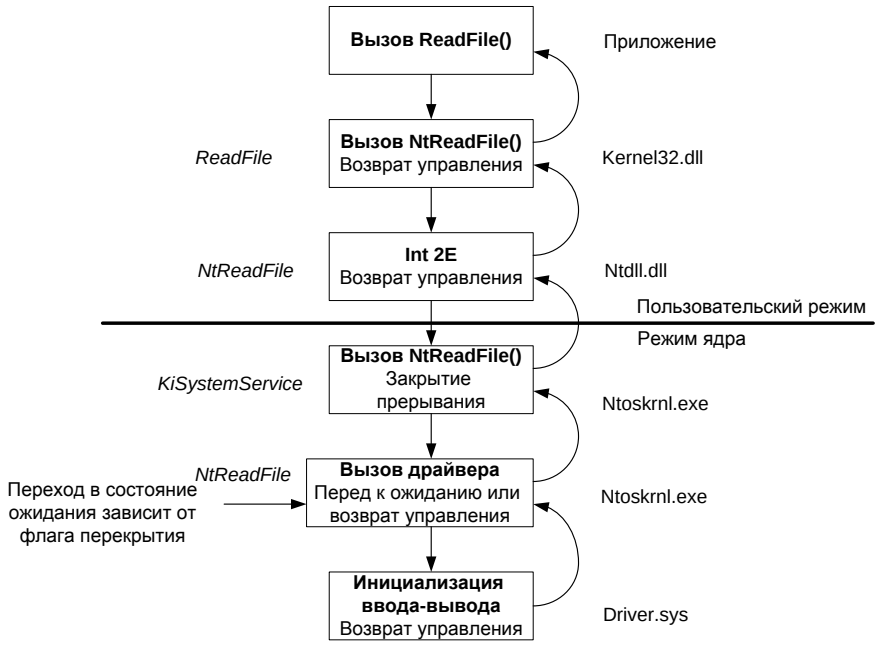

Рис. 8.Схема управления при операции ввода-вывода

Быстрый ввод-вывод

Быстрый ввод-вывод (fast I/O) — специальный механизм, который позволяет подсистеме ввода-вывода напрямую, не генерируя IRP, обращаться к драйверу файловой системы или диспетчеру кэша. Драйвер регистрирует свои точки входа для быстрого ввода-вывода, записывая их адреса в структуру, на которую ссылается указатель `PFAST_IO_DISPATCH` его объекта «драйвер».

Ввод-вывод в проецируемые файлы и кэширование файлов

Ввод-вывод в проецируемые файлы (mapped tile I/O) — важная функция подсистемы ввода-вывода, поддерживаемая ею совместно с диспетчером памяти. Термин «ввод-вывод в проецируемые файлы» относится к возможности интерпретировать файл на диске как часть виртуальной памяти процесса. Программа может обращаться к такому файлу как к большому массиву, не прибегая к буферизации или дисковому вводу-выводу. При доступе программы к памяти диспетчер памяти использует свой механизм подкачки для загрузки нужной страницы из дискового файла. Если программа изменяет какие-то данные в своем виртуальном адресном пространстве, диспетчер памяти записывает эти данные обратно в дисковый файл в ходе обычной операции подкачки страниц.

Ввод-вывод по механизму «scatter/gather»

Windows также поддерживает особый вид высокопроизводительного ввода-вывода с использованием механизма «scatter/gather»; он доступен через Windows-функции `ReadFileScatter` и `WriteFileGather`. Эти функции позволяют приложению в рамках одной операции считывать или записывать данные из нескольких буферов в виртуальной памяти в непрерывную область дискового файла, а не выдавать отдельный запрос ввода-вывода для каждого буфера. Чтобы задействовать такой ввод-вывод, вы должны открыть файл для некэшируемого асинхронного (перекрывающегося) ввода-вывода и выровнять пользовательские буферы по границам страниц. Более того, если ввод-вывод направлен на устройство массовой памяти, то передаваемые данные нужно выровнять по границам секторов устройства, а их объем должен быть кратен размеру сектора.

Пакеты запроса ввода-вывода

Пакет запроса ввода-вывода (I/O request packet, IRP) хранит информацию, нужную для обработки запроса на ввод-вывод. Когда поток вызывает сервис ввода-вывода, диспетчер ввода-вывода создает IRP для представления операции в процессе ее выполнения подсистемой ввода-вывода. По возможности диспетчер ввода-вывода выделяет память под IRP в одном из двух ассоциативных списков IRP, индивидуальных для каждого процессора и хранящихся в пуле неподкачиваемой памяти. Ассоциативный список малых IRP (small IRP look-aside list) хранит IRP с одним блоком стека (об этих блоках — чуть позже), а ассоциативный список больших IRP (large-IRP look-aside list) — IRP с несколькими блоками стека. По умолчанию в IRP второго списка содержится 8 блоков стека, но раз в минуту система варьирует это число на основании того, сколько блоков было запрошено. Если для IRP требуется больше блоков стека, чем имеется в ассоциативном списке больших IRP, диспетчер ввода-вывода выделяет память под IRP из пула неподкачиваемой памяти. После создания и инициализации TRP диспетчер ввода-вывода сохраняет в IRP указатель на объект «файл» вызывающего потока.

На Рис. 9 показан пример запроса ввода-вывода, демонстрирующий взаимосвязи между IRP и объектами «файл», «устройство» и «драйвер». Данный пример относится к запросу ввода-вывода, который адресован одноуровневому драйверу, но большинство операций ввода-вывода гораздо сложнее, так как в их выполнении участвует не один, а несколько многоуровневых драйверов. (Этот случай выходит за рамки лекции, с ним можно ознакомится в [3])

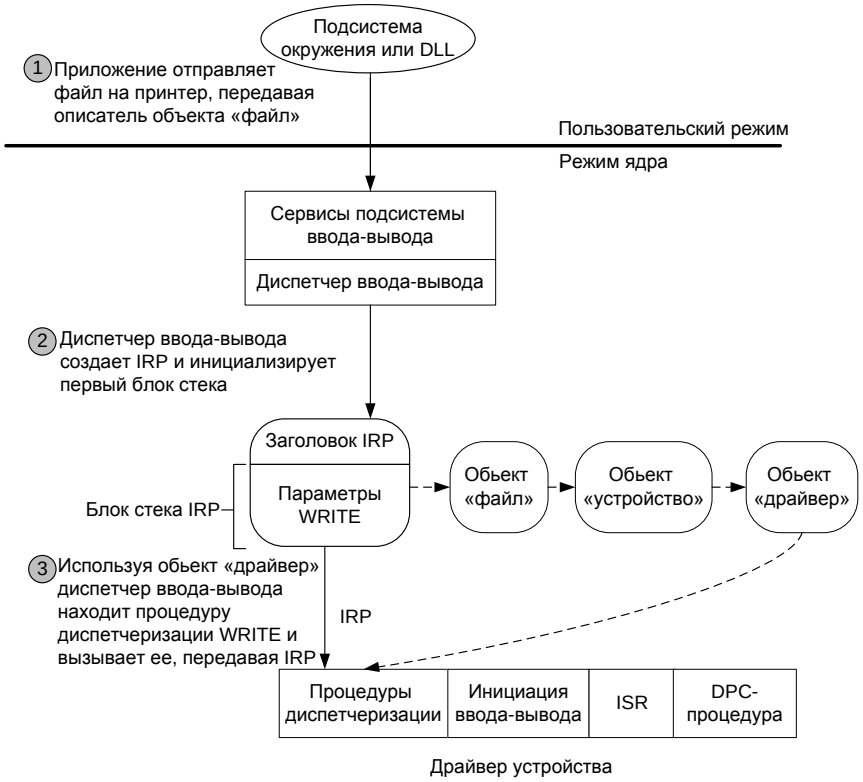

Рис. 9. Структуры данных, участвующие в обработке запроса на ввод-вывод, адресованного одноуровневому драйверу

Блок стека IRP

IRP состоит из двух частей: **фиксированного заголовка** (часто называемого телом IRP) и **одного или нескольких блоков стека**.

Фиксированная часть содержит такую информацию, как тип и размер запроса, указатель на буфер в случае буферизованного ввода-вывода, данные о состоянии, изменяющиеся по мере обработки запроса, а также сведения о том, является запрос синхронным или асинхронным. Блок стека IRP (IRP stack location) содержит номер функции (состоящий из основного и дополнительного номеров), параметры, специфичные для функции, и указатель на объект «файл» вызывающего потока. Основной номер функции (major function code) идентифицирует принадлежащую драйверу процедуру диспетчеризации, которую диспетчер ввода-вывода вызывает при передаче IRP драйверу. Необязательный дополнительный номер функции (minor function code) иногда используется как модификатор основного номера. В командах управления электропитанием и Plug and Play всегда указывается дополнительный номер функции. В большинстве драйверов процедуры диспетчеризации определены только для подмножества основных функций, т. е. функций, предназначенных для создания/открытия, записи, чтения, управления вводом-выводом на устройстве, управления электропитанием, операций Plug and Play, System (для WMI команд) и закрытия. Драйверы файловой системы определяют функции для всех (или почти всех) точек входа. Диспетчер ввода-вывода записывает в точки входа, не заполненные драйверами, указатели на свою функцию `IopInvalidDeviceRequest`. Эта функция возвращает вызывающему потоку код ошибки, который уведомляет о попытке обращения к функции, не поддерживаемой данным устройством.

Управление буфером IRP

Диспетчер ввода-вывода поддерживает три вида управления буферами.

- **Буферизованный ввод-вывод** (buffered I/O). Диспетчер ввода-вывода выделяет в пуде неподкачиваемой памяти буфер, равный по размеру буферу вызывающего потока. Создавая IRP при операциях записи, диспетчер ввода-вывода копирует данные из буфера вызывающего потока в выделенный буфер. Завершая обработку IRP при операциях чтения, диспетчер ввода-вывода копирует данные из выделенного буфера в пользовательский буфер и освобождает выделенный буфер.

- **Прямой ввод-вывод** (direct I/O). Создавая IRP, диспетчер ввода-вывода блокирует пользовательский буфер в памяти (делает его неподкачиваемым). Закончив работу с IRP, диспетчер ввода-вывода разблокирует буфер. Диспетчер хранит описание этой памяти в форме MDL (memory descriptor list). MDL указывает объем физической памяти, занятой буфером (подробнее о MDL см. Windows DDK). Устройствам, использующим DMA (прямой доступ к памяти), требуется лишь физическое описание буфера, поэтому таким устройствам достаточно MDL. (Устройства, поддерживающие DMA, передают данные в память компьютера напрямую, не используя процессор.) но, если драйверу нужен доступ к содержимому буфера, он может спроецировать его на системное адресное пространство.
- **Ввод-вывод без управления** (neither I/O). Диспетчер ввода-вывода не участвует в управлении буферами. Ответственность за управление ими возлагается на драйвер устройства.

При любом типе управления буферами диспетчер ввода-вывода помещает в IRP ссылки на буферы ввода и вывода. Тип управления буферами, реализуемого диспетчером ввода-вывода, зависит от типа управления, запрошенного драйвером для операций конкретного типа. Драйвер регистрирует нужный ему тип управления буферами для операций чтения и записи в объекте «устройство», который представляет устройство. Операции управления вводом-выводом на устройстве (выполняемые `NtDeviceloControlFile`) задаются определенными в драйвере управляющими кодами ввода-вывода. Управляющий код включает тип управления буферами со стороны диспетчера ввода-вывода при обработке IRP с данным кодом.

Запрос ввода-вывода к одноуровневому драйверу

Далее опишем процесс обработки запросов синхронного ввода-вывода к **одноуровневому драйверу режима ядра**. Такая обработка проходит в семь этапов:

1. Запрос на ввод-вывод передается через DLL подсистемы.

2. DLL подсистемы вызывает сервис `NtWriteFile` диспетчера ввода-вывода.

3. Диспетчер ввода-вывода создает IRP, описывающий запрос, и посылает его драйверу (в данном случае — драйверу устройства), вызывая свою функцию `IoCallDriver`.

4. Драйвер передает данные из IRP на устройство и инициирует операцию ввода-вывода.

5. Драйвер уведомляет о завершении ввода-вывода, генерируя прерывание.

6. Когда устройство завершает операцию и вызывает прерывание, драйвер устройства обслуживает прерывание.

7. Драйвер вызывает функцию `IoCompleteRequest` диспетчера ввода-вывода, чтобы уведомить его о завершении обработки IRP, и диспетчер ввода-вывода завершает данный запрос на ввод-вывод.

Эти семь этапов показаны на Рис. 10.

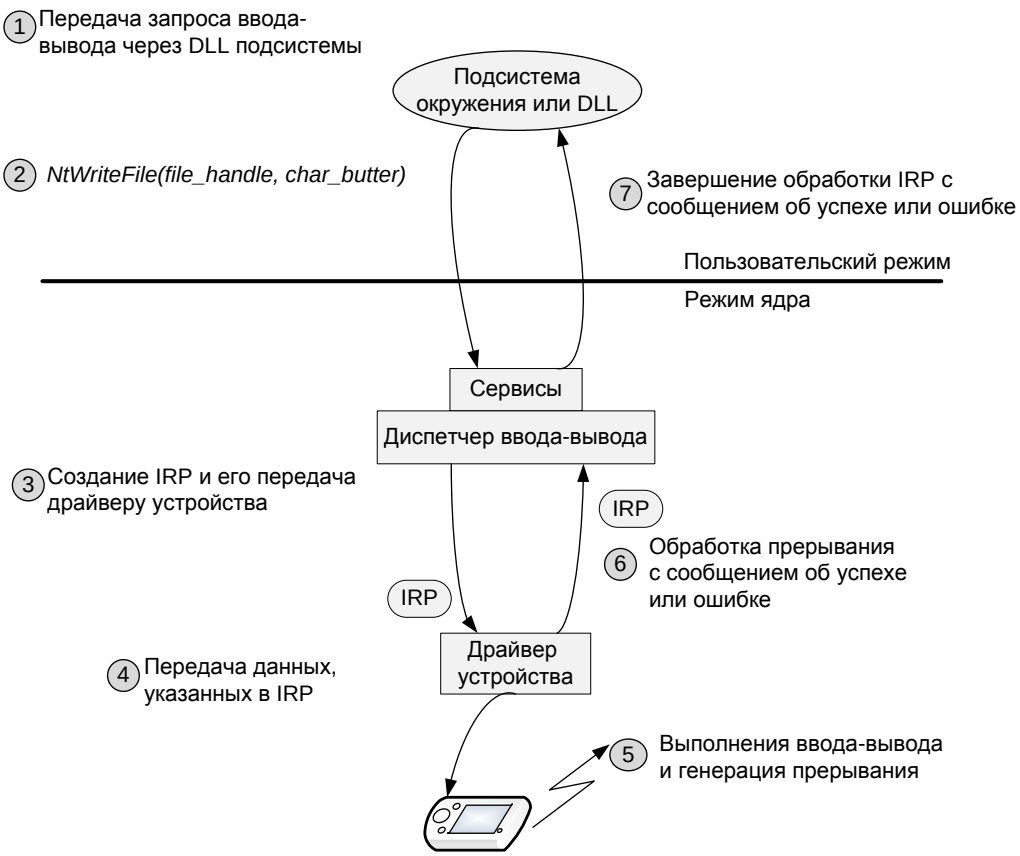

Рис. 10. Обработка синхронного запроса

Рассмотрим обслуживание прерывания и завершение ввода-вывода.

Обслуживание прерывания

Завершая передачу данных, устройство генерирует прерывание, после чего в дело вступают ядро Windows, диспетчер ввода-вывода и драйвер устройства. На Рис. 11 показана первая фаза этого процесса.

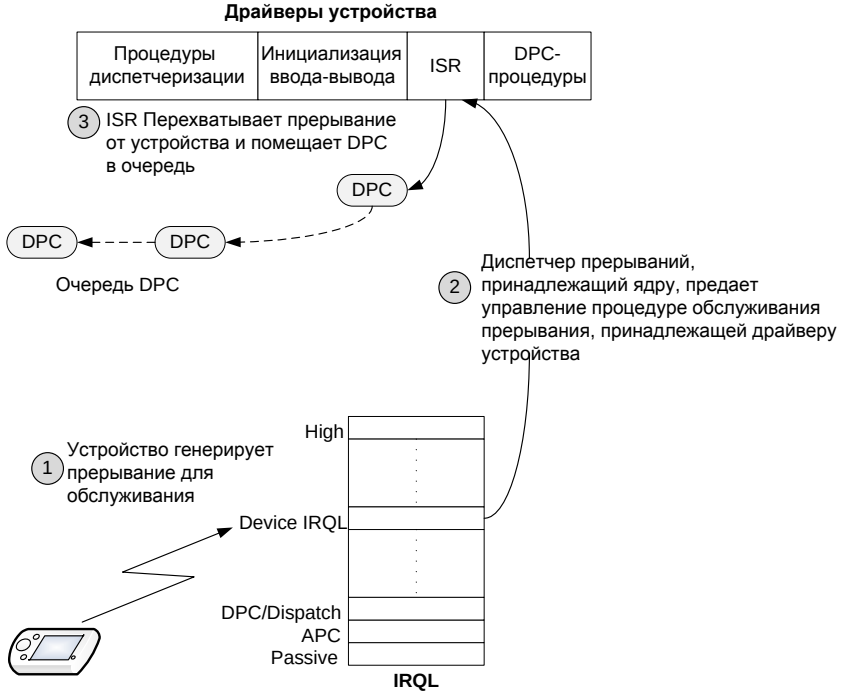

Рис. 11. Обслуживание прерывания от устройства (фаза 1)

Когда устройство генерирует прерывание, процессор передает управление обработчику ловушки ядра, который находит ISR для этого устройства по таблице диспетчеризации

прерываний. ISR в Windows обычно обрабатывают прерывания от устройств в два этапа. При первом вызове ISR, как правило, остается на уровне Device IRQL ровно столько времени, сколько нужно для того, чтобы сохранить состояние устройства и запретить дальнейшие прерывания от него. После этого ISR помещает DPC в очередь и, закрыв прерывание, завершается. Впоследствии, при вызове DPC-процедуры драйвер устройства заканчивает обработку прерывания, а затем вызывает диспетчер ввода-вывода для завершения ввода-вывода и удаления IRP. Этот драйвер также может начать выполнение следующего запроса ввода-вывода, ждущего в очереди устройства.

Преимущество выполнения большей части обработки прерываний от устройств через DPC в том, что это разрешает любые блокируемые прерывания с приоритетами от «Device IRQL» до «DPC/dispalch» — пока не началась обработка DPC, имеющего более низкий приоритет. А за счет этого удается более оперативно (чем это могло бы быть в ином случае) обслуживать прерывания среднего приоритета. Вторая фаза ввода-вывода (обработка DPC) показана на Рис. 12.

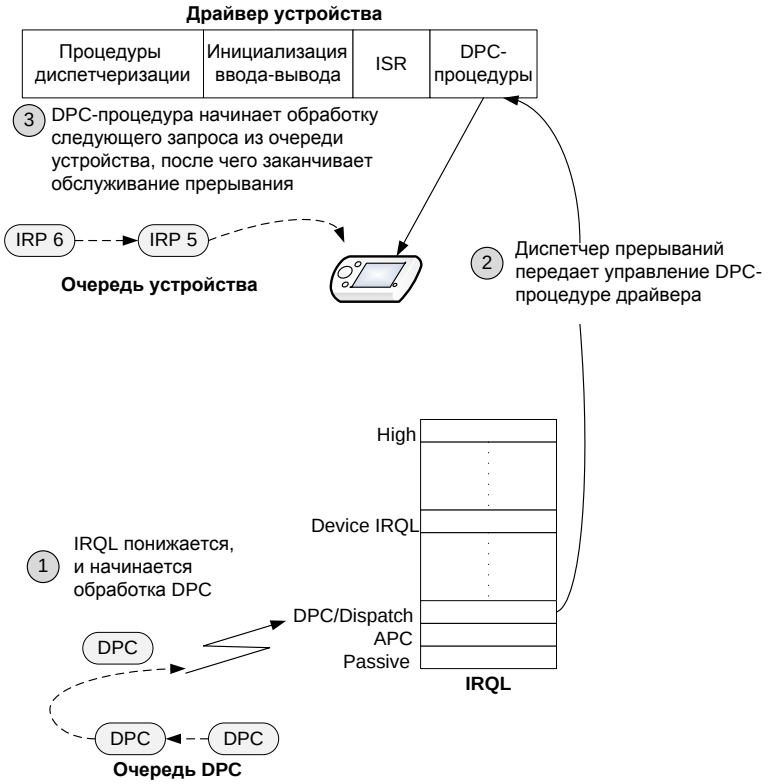

Рис. 12. Обслуживание прерывания от устройства (фаза 2)

Завершение обработки запроса на ввод-вывод

Третья стадия обработки ввода-вывода называется завершением ввода-вывода (I/O completion) и начинается с вызова драйвером функции `IoCompleteRequest` для уведомления диспетчера ввода-вывода о том, что обработка запроса, указанного в IRP (и принадлежащих ему блоках стека), закончена. Действия, выполняемые на этом этапе, различны для разных операций ввода-вывода.

В любом случае подсистема ввода-вывода должна копировать отдельные данные из системной памяти в виртуальное адресное пространство процесса, которому принадлежит вызывающий поток. Если IRP выполняется синхронно, это адресное пространство является текущим и доступно напрямую, но если IRP обрабатывается асинхронно, диспетчер ввода-вывода должен отложить завершение IRP до тех пор, пока у него не появится возможность обращаться к нужному адресному пространству. Чтобы получить доступ к виртуальному адресному пространству процесса, которому принадлежит вызывающий поток, диспетчер ввода-вывода должен передавать данные «в контексте вызывающего потока», т.е. при выполнении этого потока (иначе говоря, процесс этого потока должен быть текущим, а его адресное пространство — активным на CPU). Эту задачу диспетчер ввода-вывода решает, ставя в очередь данному потоку APC режима ядра (Рис. 13).

APC выполняется только в контексте определенного потока, а DPC—в контексте любого потока. Это означает, что DPC не затрагивает адресное пространство процесса

пользовательского режима. Вспомните также, что приоритет программного прерывания у DPC выше, чем у APC.

В следующий раз, когда поток начинает выполняться при низком IRQL, ему доставляется отложенный APC. Ядро передает управление APC-процедуре диспетчера ввода-вывода, которая копирует данные (если они есть) и код возврата в адресное пространство процесса вызывающего потока, освобождает IRP, представляющий данную операцию ввода-вывода, и переводит описатель файла (и любое событие или порт завершения ввода-вывода, если таковой объект предоставлен вызывающим потоком) в свободное состояние. Теперь ввод-вывод считается завершенным. Вызывающий поток или любые другие потоки, ждущие на описателе файла (или иного объекта), выходят из состояния ожидания и переходят в состояние готовности к выполнению. Вторая фаза завершения ввода-вывода показана на Рис. 14.

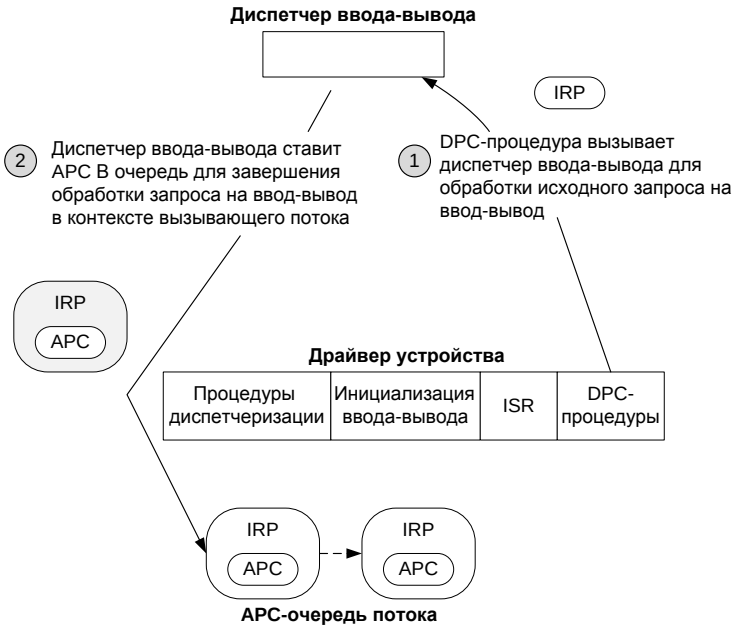

Рис. 13. Завершение обработки запроса на ввод-вывод (фаза 1)

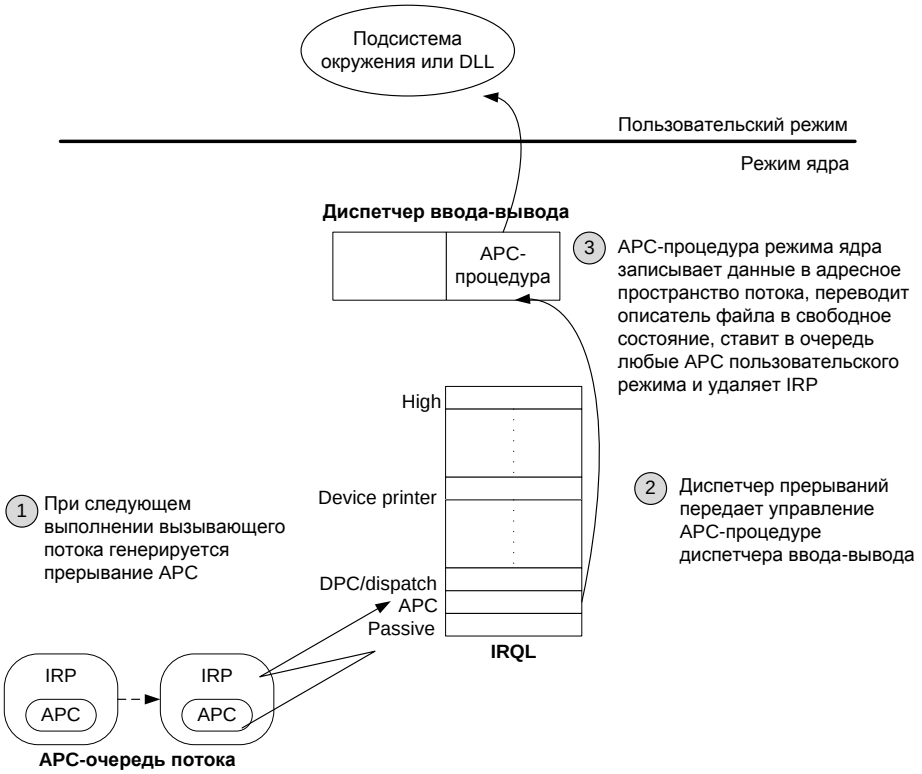

Рис. 14. Завершение обработки запроса на ввод-вывод (фаза 2)

Синхронизация

Драйверы должны синхронизировать свое обращение к глобальным данным и регистрам устройств в силу двух причин.

- Выполнение драйвера может быть прервано из-за вытеснения потоками с более высоким приоритетом, по истечении выделенного кванта времени CPU, а также из-за генерации прерывания.
- В multi-CPU системах Windows может выполнять код драйвера сразу на нескольких CPU.

Без синхронизации данные могут быть повреждены. Например, код драйвера устройства выполняется при IRQL, уровня «passive». Какая-то программа инициирует операцию ввода-вывода, в результате чего возникает аппаратное прерывание. Оно прерывает выполнение кода драйвера и активизирует его ISR. Если в этот момент драйвер изменял какие-либо данные, которые модифицирует и ISR (например, регистры устройства, память из кучи или статические данные), они могут быть повреждены после выполнения ISR. Эту проблему демонстрирует Рис. 15.

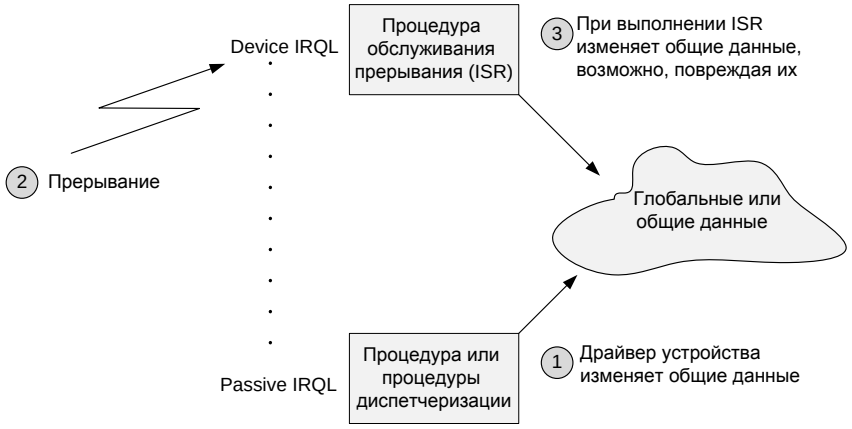

Рис. 15. Пример повреждения данных, используемых драйвером устройства, в отсутствие синхронизации

Во избежание такой ситуации драйвер, написанный для Windows, должен синхронизировать обращение к любым данным, которые он разделяет со своей ISR. Прежде чем обновлять общие данные, драйвер должен заблокировать все остальные потоки (или CPU, если система multi-CPU), чтобы запретить им доступ к тем же данным.

Ядро Windows предоставляет специальную синхронизирующую процедуру `KeSynchronizeExecution`, которую драйверы устройств должны вызывать при доступе к данным, разделяемым с ISR. Эта процедура не допускает выполнения ISR, пока драйвер обращается к общим данным. В однопроцессорных системах перед обновлением общих структур данных она повышает IRQL до уровня, сопоставленного с ISR. Но в multi-CPU системах эта методика не гарантирует полной блокировки, так как код драйвера может выполняться на двух и более CPU одновременно. Поэтому в multi-CPU системах применяется другой механизм — **спин-блокировка**. Драйвер также может использовать `KeAcquirelnterruptSpinLock` для прямого доступа к спин-блокировке объекта прерывания, хотя вариант синхронизации с ISR через `KeSynchronizeExecution` обычно работает быстрее.

Запрос ввода-вывода к многоуровневому драйверу

Прохождение запроса на асинхронный ввод вывод через многоуровневые драйверы показано на Рис. 16. Данный пример относится к диску, управляемому файловой системой.

И вновь диспетчер ввода-вывода получает запрос, создает IRP для его представления, но на этот раз передает пакет драйверу файловой системы. С этого момента драйвер файловой системы в основном и управляет операцией ввода-вывода. В зависимости от типа запроса файловая система посылает драйверу диска тот же IRP или генерирует дополнительные IRP и передает их этому драйверу по отдельности.

Файловая система, скорее всего будет повторно использовать IRP, если полученный запрос можно преобразовать в единый запрос к устройству.

Для поддержки использования несколькими драйверами IRP содержит набор блоков стека (не путать со стеком потока). Эти блоки данных — по одному на каждый вызываемый драйвер — хранят информацию, необходимую каждому драйверу для обработки своей части запроса (например, номер функции, параметры, сведения о контексте драйвера). Как показано на Рис. 16, по мере передачи IRP от одного драйвера другому заполняются дополнительные блоки стека. IRP можно считать аналогом стека в отношении добавления и удаления данных. Но TRP не сопоставляется ни с каким процессом, и его размер фиксирован. В самом начале операции ввода-вывода диспетчер ввода-вывода выделяет память для TRP в одном из ассоциативных списков IRP или в пуле неподкачиваемой памяти.

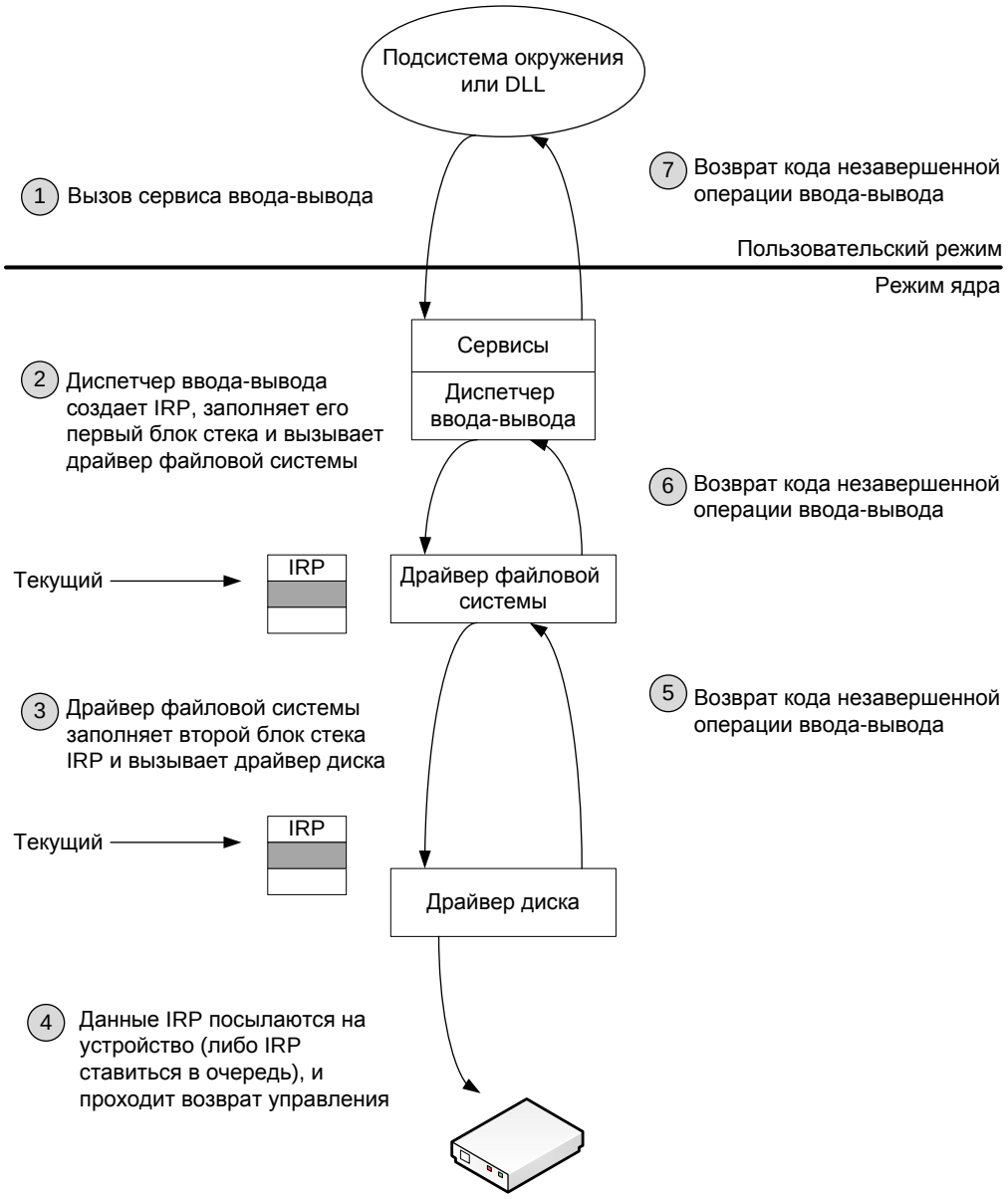

Рис. 16. Обработка асинхронного запроса к многоуровневым драйверам

После того как драйвер диска завершает передачу данных, диск генерирует прерывание, и ввод-вывод завершается (Рис. 17).

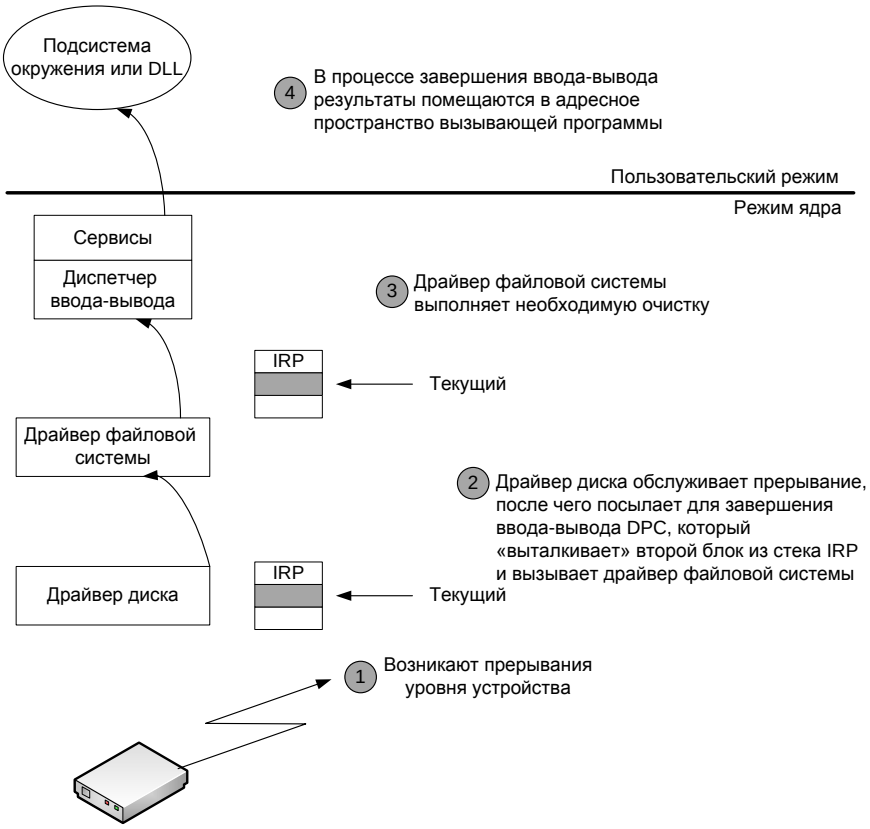

Рис. 17. Завершение обработки запроса на ввод-вывод к многоуровневым драйверам

В качестве альтернативы повторному использованию единственного IRP файловая система может создать группу сопоставленных IRP (associated IRPs), которые будут обрабатываться параллельно. Например, если данные, которые нужно считать из файла, разбросаны по всему диску, драйвер файловой системы может создать несколько IRP, каждый из которых инициирует чтение данных из отдельного сектора. Этот случай иллюстрирует Рис. 18.

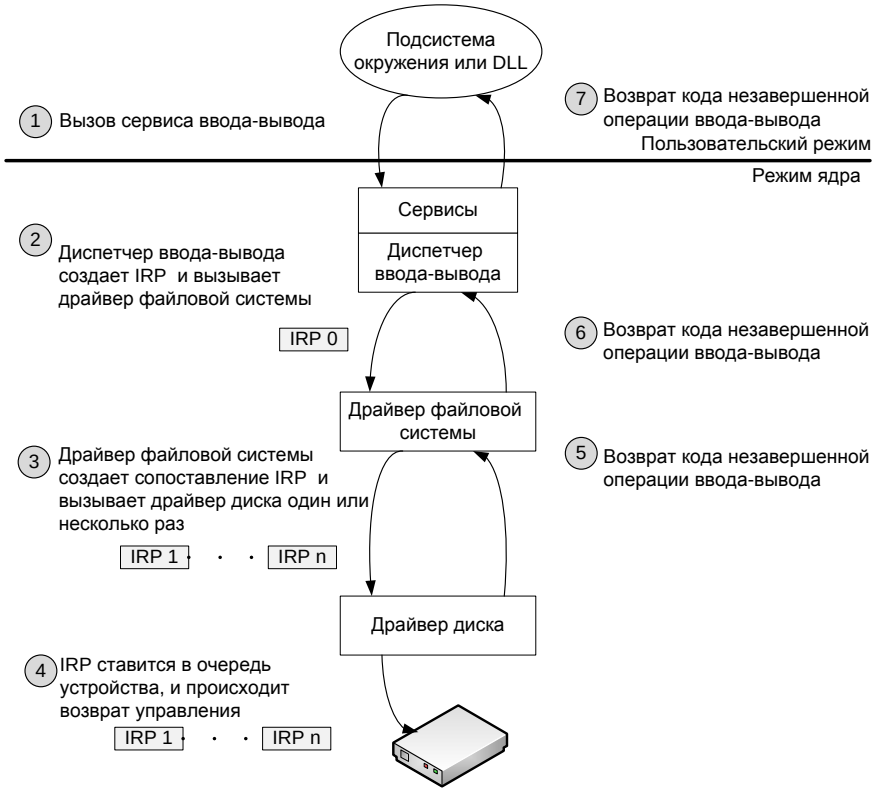

Рис. 18. Обработка с использованием сопоставленных IRP

Драйвер файловой системы передает сопоставленные IRP драйверу устройства, который ставит их в очередь устройства. Они обрабатываются по одному, а файловая система отслеживает возвращаемые данные. Когда выполнение всех сопоставленных IRP заканчивается, подсистема ввода-вывода завершает обработку исходного IRP и возвращает управление вызывающему потоку (Рис. 19).

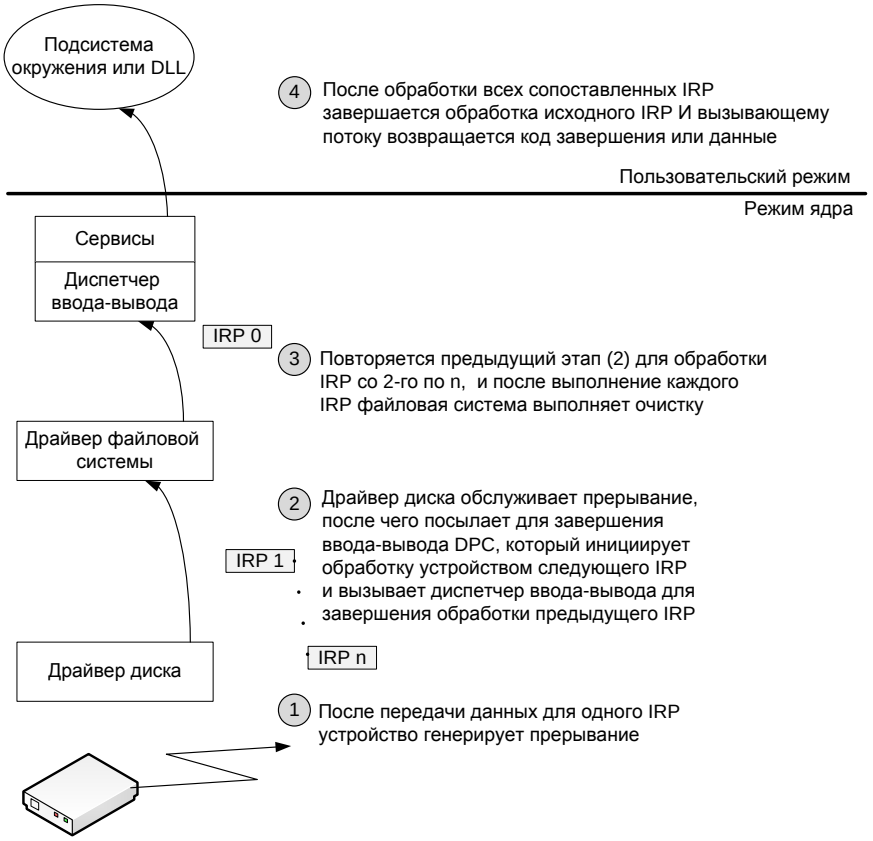

Рис. 19. Завершение обработки сопоставленных IRP

ПРИМЕЧАНИЕ. Все драйверы, управляющие дисковыми файловыми системами в Windows, являются частью как минимум трехуровневого стека драйверов: драйвер файловой системы находится на верхнем уровне, диспетчер томов — на среднем, а драйвер диска — на нижнем. Кроме того, между этими драйверами может размещаться любое число драйверов фильтров. Для ясности в предыдущем примере были показаны лишь драйверы файловой системы и диска.

Порты завершения ввода-вывода

Задача сервера — свести к минимуму число переключений контекста, что бы избежать излишнего блокирования потоков, и в то же время добиться максимального параллелизма в обработке за счет множества потоков. С этой точки зрения идеальна ситуация, при которой к каждому CPU подключен один активно обрабатывающий клиентские запросы поток, что позволяет обойтись без блокировки потоков, если на момент завершения обработки текущих запросов их ждут другие запросы. Однако такая оптимизация требует, чтобы у приложения была возможность активизировать другой поток, когда поток, обрабатывающий клиентский запрос, блокируется в ожидании ввода-вывода (например, для чтения файла в процессе обработки).

Объект IoCompletion

Приложения используют объект `IoCompletion` исполнительной системы, который экспортируется в Windows как **порт завершения** (completion port) — фокальная точка завершения ввода-вывода, сопоставляемая с множеством описателей файлов. Если какой-нибудь файл сопоставлен с портим завершения, то по окончании любой операции асинхронного ввода-вывода, связанной с этим файлом, в очередь порта завершения ставится **пакет завершения** (completion packet). Ожидание завершения любой из операций ввода-вывода в нескольких файлах может быть реализовано простым ожиданием соответствующего пакета завершения, который должен появиться

в очереди порта завершения. Windows API поддерживает аналогичную функциональность через `WaitForMultipleObjects`, но порты завершения дают одно большое преимущество: число потоков, активно обслуживающих клиентские запросы, контролируется самой системой.

Создавая порт завершения, приложение указывает максимальное число сопоставленных с портом потоков, которые могут быть активны. Как уже говорилось, в идеале на каждом CPU должно быть по одному активному потоку. Windows использует это значение для контроля числа активных потоков приложения. Если число активных потоков, сопоставленных с портом, равно заданному максимальному значению, выполнение потока, ждущего на порте завершения, запрещено. По завершении обработки текущего запроса один из активных потоков проверяет, имеется ли в очереди порта другой пакет. Если да, он просто извлекает этот пакет из очереди и переходит к обработке соответствующих данных; контекст, при этом, не переключается.

Использование портов завершения

Высокоуровневая схема работы порта завершения представлена на Рис. 20. Порт завершения создается вызовом Windows-функции `CreateloComplelionPort`. Потоки, блокированные на порте завершения, считаются сопоставленными с ним и пробуждаются по принципу FIFO («последним пришел — первым вышел»), т.е. следующий пакет достается потоку, заблокированному последним. Стеки потоков, блокируемых в течение длительного времени, могут быть выгружены в страничный файл. В итоге, если с портом сопоставлено больше потоков, чем нужно для обработки текущих заданий, система автоматически минимизирует объем памяти, занимаемой слишком долго блокируемыми потоками.

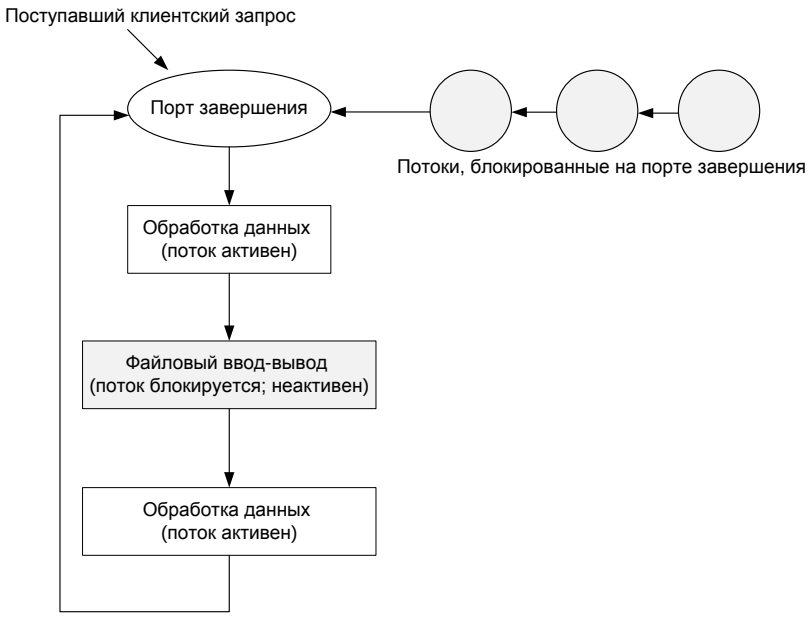

Рис. 20. Так работает порт завершения ввода-вывода

Microsoft рекомендует устанавливать максимальное число активных потоков на порте завершения примерно равным числу CPU в системе. Имейте в виду, что это значение может быть превышено. Допустим, вы задали, что максимальное значение должно быть равно 1. При поступлении клиентского запроса выделенный для его обработки поток становится активным. Поступает второй запрос, но второй поток не может продолжить его обработку, так как лимит уже достигнут. Затем первый поток блокируется в ожидании файлового ввода-вывода и становится неактивным. Тогда освобождается второй поток и, пока он активен, завершается файловый ввод-вывод для первого потока, в результате чего первый поток вновь активизируется. С этого момента и до блокировки одного из потоков число активных потоков превышает установленный лимит на 1.

Как работает порт завершения ввода-вывода

Windows-приложения создают порты завершения вызовом Windows-функции `CreateloCompletionPort` с указанием NULL вместо описателя порта завершения. Это приводит к выполнению системного сервиса `NtCreateloCompletion`. Объект `IoCompletion` исполнительной системы, построенный на основе синхронизующего объекта ядра, называется очередью. Таким образом, системный сервис создаст объект «порт завершения» и инициализирует объект «очередь» в памяти, выделенной для порта. (Указатель на порт ссылается и на объект «очередь», так как последний находится в начальной области памяти порта.) Максимальное число сопоставленных с портом потоков, которые могут быть активны, указывается в объекте «очередь» при его инициализации; это значение, которое было передано в `CreateIoCompletionPort`. Для инициализации объекта «очередь» порта завершения `NtCreateIoCompletion` вызывает функцию `KeInitializeQueue`.

Когда приложение обращается к `CreateloCompletionPort` для связывания описателя файла с портом, вызывается системный сервис `NtSetlnformationFile`, которому передается описатель этого файла. При этом класс информации для `NtSetInformationFile` устанавливается как `FileCompletionlnformation`, и, кроме того, эта функция принимает описатель порта завершения и параметр `CompletionKey`, ранее переданный в `CreateIoCompletionPort`. Функция `NtSetInformationFile` производит разыменование описателя файла для получения объекта «файл» и создает структуру данных контекста завершения.

Указатель на эту структуру `NtSetlnformationFile` помещает в поле `CompletianContext` объекта «файл». По завершении асинхронной операции ввода-вывода для объекта «файл» диспетчер ввода-вывода проверяет, отличается ли поле `CompletionContext` от `NULL`. Если да, он создает пакет завершения и ставит его в очередь порта завершения вызовом `KeInsertQueue`; при этом в качестве очереди, в которую помещается пакет, указывается порт. (Здесь объект «порт завершения» — синоним объекта «очередь».)

Когда серверный поток вызывает `GetQueuedCompletionStatus`, выполняется системный сервис `NtRemoveIoCompletion`. После проверки параметров и преобразования описателя порта завершения в указатель на порт `NtRemoveIoCompletion` вызывает `KeRemoveQueue`.

Как видите, `KeRemoveQueue` и `KelnsertQueue` — это базовые функции, обеспечивающие работу порта завершения. Они определяют, следует ли активизировать поток, ждущий пакет завершения ввода-вывода. Объект «очередь» поддерживает внутренний счетчик активных потоков и хранит такое значение, как максимальное число активных потоков. Если при вызове потоком `KeRemoveQueue` текущее число активных потоков равно максимуму или превышает его, данный поток будет включен (в порядке LIFO) в список потоков, ждущих пакет завершения. Список потоков отделен от объекта «очередь». В блоке управления потоком имеется поле для указателя на очередь, сопоставленную с объектом «очередь»; если это поле пустое, поток не связан с очередью.

Windows отслеживает потоки, ставшие неактивными из-за ожидания на каких-либо объектах, отличных от порта завершения, по указателю на очередь, присутствующему в блоке управления потоком. Процедуры планировщика, в результате выполнения которых поток может быть блокирован (`KeWaitForSingleObject`, `KeDelayExecutionThread` и т.д.), проверяют этот указатель. Если он не равен `NULL`, они вызывают функцию `KiActivateWaiterQueue`, которая уменьшает счетчик числа активных потоков, сопоставленных с очередью. Если конечное число меньше максимального и в очереди есть хотя бы один пакет завершения, первый поток из списка потоков очереди пробуждается и получает самый старый пакет. И напротив, всякий раз, когда после блокировки пробуждается поток, связанный с очередью, планировщик выполняет функцию `KiUnwaitThread`, увеличивающую счетчик числа активных потоков очереди.

Наконец, в результате вызова Windows-функции `PostQueuedCompletionStatus` выполняется системный сервис `NtSetIoCornpletion`, который просто вставляет с помощью `KelnsertQueue` специальный пакет в очередь порта завершения.

Порт завершения в действии показан на Рис. 21. Хотя к обработке пакетов завершения готовы два потока, максимум, равный 1, допускает активизацию только одного потока, связанного с портом завершения. Таким образом, на этом порте завершения блокируется два потока.

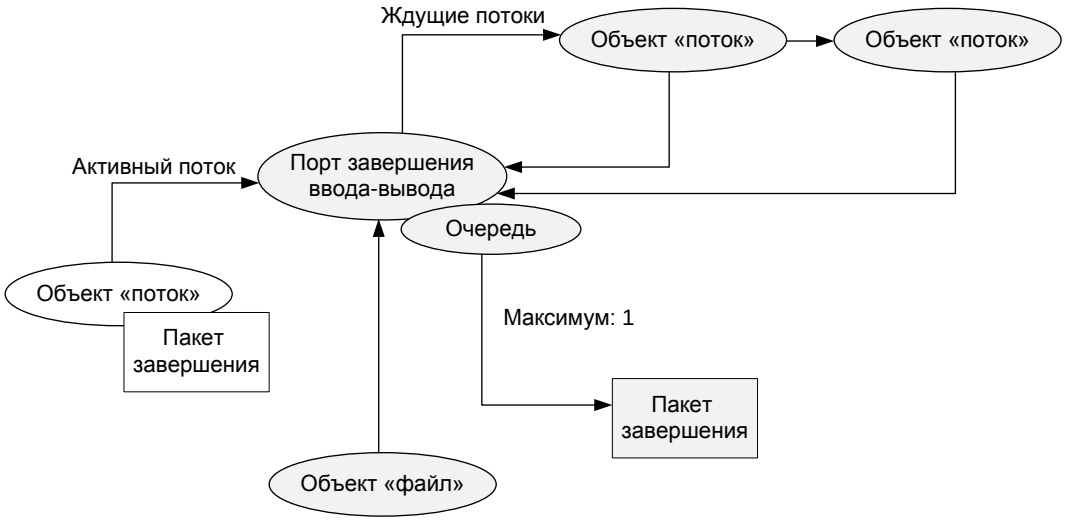

Рис. 21. Порт завершения ввода-вывода в действии

Диспетчер Plug & Play

Основной компонент, от которого зависит способность Windows к распознаванию изменений в аппаратной конфигурации. Благодаря этому пользователю не требуется знания тонкостей настройки устройств и системы при их установке и удалении.

Поддержка PnP требует взаимодействия на уровнях оборудования, драйверов устройств и ОС. Эта поддержка в Windows базируется на промышленных стандартах перечисления и идентификации устройств. Возможности реализуемые PnP:

- Диспетчер PnP автоматически **распознает установленные устройства**.
- **Выделяет аппаратные ресурсы**, собирая информацию о требованиях устройств к аппаратным ресурсам. В ходе арбитража ресурсов диспетчер PnP распределяет ресурсы между устройствами с учетом требований.
- **Загружает соответствующие драйверы**, на основе идентификационных данных устройства он определяет, установлен ли в системе драйвер, способный управлять этим устройством. Если да, диспетчер PnP указывает диспетчеру ввода-вывода загрузить его. Если подходящий драйвер не установлен, диспетчер PnP режима ядра взаимодействует с диспетчером PnP пользовательского режима, чтобы установить устройство. При этом он может попросить пользователя указать местонахождение нужных драйверов.
- Реализует механизмы, позволяющие приложениям и драйверам обнаруживать изменения в аппаратной конфигурации.

Уровень поддержки Plug and Play

Windows нацелена на полную поддержку Plug and Play, но конкретный уровень поддержки зависит от устройств, подключенных к системе, и установленных в ней драйверов. Уровень поддержки Plug and Play может быть снижен, если хотя бы один драйвер или устройство не отвечает стандарту Plug and Play. Более того, драйвер, не поддерживающий Plug and Play, может лишить систему возможности использовать другие устройства.

PnP-несовместимое устройство, например унаследованная звуковая плата с ISA-шиной, не поддерживает автоматическое определение. Из-за этого таким устройствам запрещены некоторые операции вроде «горячего» подключения или перехода в один из режимов сна. Если для такого устройства вручную установить PnP-совместимый драйвер, он сможет по крайней мере использовать ресурсы, которые диспетчер PnP будет выделять этому устройству.

Поддержка Plug and Play со стороны драйвера

Для поддержки Plug and Play в драйвере должна быть реализована процедура диспетчеризации Plug and Play, а также процедура добавления устройства. Однако драйверы шип должны поддерживать типы запросов Plug and Play, отличные от тех, которые поддерживаются функциональными драйверами и драйверами фильтров. Так, при перечислении устройств в процессе загрузки диспетчер PnP запрашивает у драйверов шин описание устройств, найденных ими на своих шинах. В это описание

входят данные, уникально идентифицирующие каждое устройство, а также требования устройств к аппаратным ресурсам. Диспетчер PnP принимает эту информацию и загружает функциональные драйверы или драйверы фильтров, установленные для обнаруженных устройств. Затем он вызывает процедуру добавления устройства каждого драйвера, установленного для каждого устройства.

Выполняя процедуру добавления устройства, функциональные драйверы и драйверы фильтров готовятся начать управление своими устройствами, но на самом деле пока еще не взаимодействуют с ними. Они ждут команду `start-device`, которую диспетчер PnP должен передать их процедурам диспетчеризации Plug and Play. До передачи этой команды диспетчер PnP выполняет арбитраж ресурсов, чтобы решить, какие ресурсы выделить тому или иному устройству. В команде `start-device` указываются назначенные ресурсы, определенные диспетчером PnP при арбитраже ресурсов. Получив команду `start-device`, драйвер может настроить свое устройство па использование указанных ресурсов. Если программа пытается открыть устройство, которое не готово к началу работы, она получает код ошибки, указывающий на отсутствие этого устройства.

После запуска устройства диспетчер PnP может посылать драйверу дополнительные PnP-команды, в том числе относящиеся к удалению устройства из системы или перераспределению ресурсов. Например, когда пользователь выбирает команду Unplug Or Eject Hardware (Отключение или извлечение аппаратного устройства), — и командует Windows извлечь устройство, диспетчер PnP посылает уведомление `query-remove` каждому приложению, зарегистрированному на получение PnP-уведомлений об этом устройстве. Как правило, приложения регистрируются на получение уведомлений через свои описатели устройства, которые они закрывают, получая уведомление `query-remove`. Если ни одно приложение не налагает вето на запрос `query-remove`, диспетчер PnP посылает команду `query-remove` драйверу, управляющему извлекаемым устройством. На этом этапе драйвер решает, что ему делать дальше: запретить удаление устройства или завершить все операции ввода-вывода на этом устройстве и прекратить дальнейший прием запросов на ввод-вывод, направляемых устройству. Если драйвер отвечает согласием на запрос об удалении и открытых описателей устройства больше нет, диспетчер PnP посылает драйверу команду `remove`, требующую от него прекратить обращение к устройству и освободить все ресурсы, выделенные им для данного устройства.

Когда диспетчеру PnP нужно перераспределить ресурсы для устройства, он сначала запрашивает драйвер, может ли тот временно приостановить операции на устройстве, и с этой целью посылает команду `query-stop`. Драйвер отвечает на этот запрос согласием, если нет риска потери или повреждения данных; в ином случае он отклоняет такой запрос. Как и в случае команды `query-remove`, драйвер, согласившись с запросом, заканчивает незавершенные операции ввода-вывода и больше не передает этому устройству запросы на ввод-вывод. (Новые запросы на ввод-вывод драйвер обычно ставит в очередь.) Далее диспетчер PnP посылает драйверу команду `stop`. На этом этапе диспетчер PnP может указать драйверу выделить устройству другие ресурсы, а потом послать команду `start-device`.

Команды Plug and Play вызывают переход устройства в строго определенные состояния, которые в упрощенной форме представлены на Рис. 22. (Некоторые состояния и команды Plug and Play на этой иллюстрации опущены. Кроме того, этот вариант относится к диаграмме состояний, реализуемой функциональными драйверами. Диаграмма состояний, реализуемых драйверами шин, гораздо сложнее.) Кстати, на рис. 22 показано одно из состояний, которое мы еще не обсудили, — устройство переходит в него после команды `surprise-remove` диспетчера PnP. Эта команда посылается при неожиданном удалении устройства из системы, например из-за его отказа или из-за извлечения устройства без применения соответствующей утилиты. Команда `surprise-remove` заставляет драйвер немедленно прекратить всякое взаимодействие с устройством, так как оно больше не подключено к системе, и отменить любые незавершенные запросы ввода-вывода.

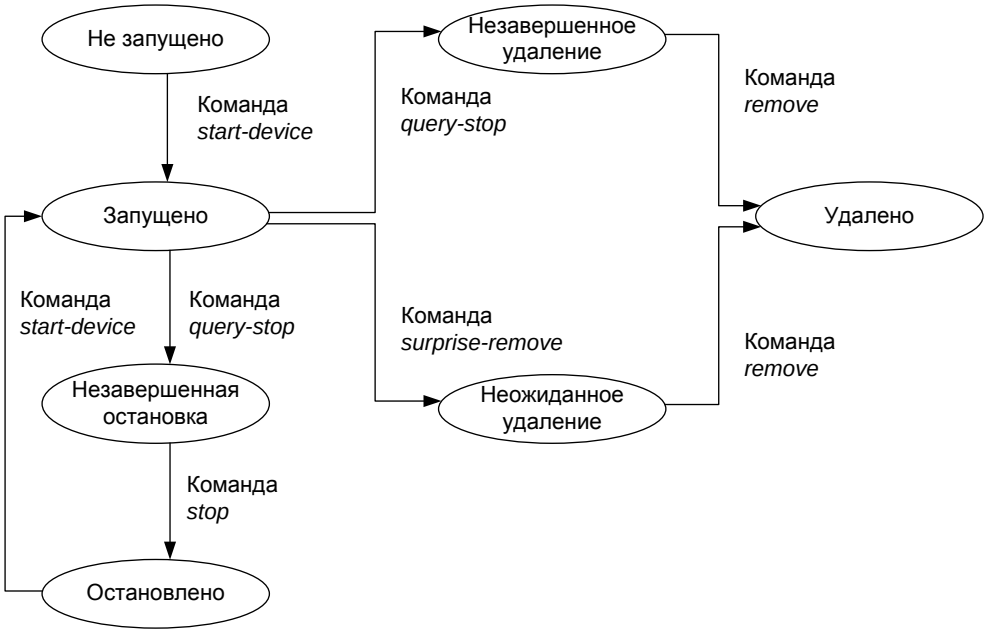

Рис. 22. Переходы PnP-состояний устройств

Загрузка, инициализация и установка драйвера

Параметр Start

У каждого драйвера и Windows-сервиса есть свой раздел в ветви реестра `Services` текущего набора параметров управления. В этот раздел входят параметры, указывающие тип образа, путь к файлу образа драйвера или сервиса и параметры, контролирующие порядок загрузки драйвера или сервиса. Между загрузкой Windows-сервисов и явной загрузкой драйверов есть два главных различия:

- только для драйверов устройств в параметре `Start` могут быть указаны значения 0 (запуск при загрузке системы) и 1 (запуск системой);
- драйверы устройств могут использовать параметры `Group` и `Tag` для контроля порядка своей загрузки при запуске системы, но в отличие от сервисов не могут определять параметры `DependOnGroup` или `DependOnService`.

Параметр `Start` драйвера, равный 0, означает, что этот драйвер загружается загрузчиком ОС. А если `Start` равен 1, драйвер загружается диспетчером ввода-вывода после инициализации компонентов исполнительной системы. Диспетчер ввода-вывода вызывает инициализирующие процедуры драйверов в том порядке, в каком драйверы загружались при запуске системы. Как и Windows-сервисы, драйверы используют параметр `Group` в своем разделе реестра, чтобы указать группу, к которой они принадлежат; порядок загрузки групп определяется параметром `HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder\List`.

Драйвер может еще больше детализировать порядок своей загрузки с помощью параметра `Tag`, который указывает конкретную позицию драйвера в группе. Диспетчер ввода-вывода сортирует драйверы в группе по значениям параметров `Tag`, определенных в разделал, реестра, соответствующих этим драйверам. Драйверы, не имеющие параметра `Tag`, перемещаются в конец списка драйверов группы. Можно предположить, что диспетчер ввода-вывода сначала инициализирует драйверы с меньшими значениями `Tag`, потом — с большими, но это не так. Приоритет значений параметров `Tag` в рамках группы определяется в `HKLM\SYSTEM\CurrentControlSet\Control\GroupOrderList`; этот раздел реестра дает Microsoft и разработчикам драйверов свободу в определении собственной системы целых чисел.

Вот правила, по которым драйверы устанавливают значение своего параметра `Start`.

- Драйверы, не поддерживающие Plug and Play, настраивают `Start` так, что-бы система загружала их на определенном этапе своего запуска.
- Драйверы, которые должны загружаться системным загрузчиком при запуске ОС, указывают в `Start` значение 0 (запуск при загрузке системы).

- Драйвер, который не требуется для загрузки системы и распознает устройство, не перечисляемое драйвером системной шины, указывает в `Start` значение, равное 1 (запуск системой). Пример — драйвер последовательного порта, информирующий диспетчер PnP о присутствии стандартных последовательных портов, которые были обнаружены программой `Setup` и зарегистрированы в реестре.
- Драйвер, не поддерживающий Plug and Play, или драйвер файловой системы, не обязательный для загрузки системы, устанавливает значение `Start` равным 2 (автозапуск). Пример - драйвер многосетевого UNC-nровайдера (Multiple UJNC Provider, MUP), поддерживающий UJNC-имена удаленных ресурсов (вроде `\\REMOTECOMPUTERNAME\SHARE`).
- PnP-драйверы, не нужные для загрузки системы (например, драйверы сетевых адаптеров), указывают значение `Start` равным 3 (запуск по требованию). Единственное предназначение параметра `Start` для PnP-драйверов и драйверов перечисляемых устройств — загрузка драйвера с помощью загрузчика ОС, если такой драйвер обязателен для успешного запуска системы.

Перечисление устройств

Диспетчер PnP начинает перечисление устройств с виртуального драйвера шины с именем Root, который представляет всю систему и выступает в роли драйвера шины для драйверов, не поддерживающих Plug and Play и для HAT.

HAL работает как драйвер шины, перечисляющий устройства, напрямую подключенные к материнской плате, и такие системные компоненты, как аккумуляторы. Определяя основную шину (обычно это PCI-шина) и устройства типа аккумуляторов и вентиляторов, HAL на самом деле полагается на описание оборудования, зафиксированное программой `Setup` в реестре еще при установке ОС.

Драйвер основной шины перечисляет устройства на этой шине, при этом он может найти другие шины, драйверы которых инициализируются диспетчером PnP. Эти драйверы в свою очередь могут обнаруживать другие устройства, включая вспомогательные шины. Такой рекурсивный процесс — перечисление, загрузка драйвера (если он еще не загружен), дальнейшее перечисление — продолжается до тех пор, пока не будут обнаружены и сконфигурированы все устройства в системе.

По мере поступления сообщений от драйверов шин об обнаруженных устройствах, диспетчер PnP формирует внутреннее дерево, называемое **деревом устройств** (device tree) и отражающее взаимосвязи между устройствами. Узлы этого дерева называются **узлами устройств** (device nodes, dev-nodes). Узел устройств содержит информацию об объектах «устройство», представляющих устройства, и другую PnP-информацию, которая записывается в узел диспетчером PnP. Упрощенный пример дерева устройств показан на Рис. 23. Эта система ACPI-совместима, и поэтому перечислителем основной шины является ACPI-совместимый HAL. К основной шине PCI в данной системе подключены шины USB, ISA и SCSI.

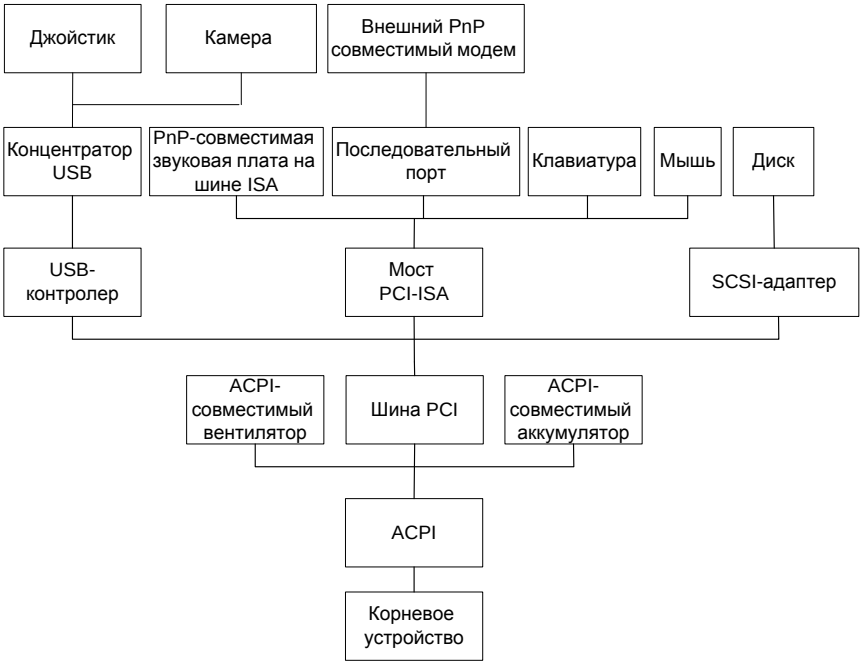

Рис. 23. Пример дерева устройств

С учетом перечисления устройств загрузка и инициализация драйверов происходит в следующем порядке.

1. Диспетчер ввода-вывода вызывает входную процедуру каждого драйвера, запускаемого при загрузке системы. Если у такого драйвера имеются дочерние устройства, диспетчер ввода-вывода перечисляет эти устройства, сообщая о них диспетчеру PnP. Дочерние устройства конфигурируются и запускаются, если их драйверы являются запускаемыми при загрузке системы. Если у устройства есть драйвер, не запускаемый при загрузке системы, диспетчер PnP создает для этого устройства узел, но не запускает устройство и не загружает его драйвер.

2. После инициализации драйверов, запускаемых при загрузке системы, диспетчер PnP проходит по дереву устройств, загружая драйверы для узлов устройств, не загруженных на первом этапе, и запускает их устройства. Запуская каждое устройство диспетчер PnP перечисляет его дочерние устройства (если таковые есть). Для этого он запускает соответствующие драйверы и при необходимости перечисляет их дочерние устройства. На данном этапе диспетчер PnP загружает драйверы для обнаруженных устройств независимо от значений параметров `Start` этих драйверов (кроме тех драйверов, параметр `Start` которых содержит значение «отключен»). В конце этого этапа драйверы всех PnP-устройств загружены и запущены, кроме драйверов не перечисляемых устройств и их дочерних устройств.

3. Диспетчер PnP загружает любые еще не загруженные драйверы, запускаемые системой. Эти драйверы определяют свои устройства, не перечисляемые обычным образом, и сообщают о них. После этого диспетчер PnP загружает драйверы для этих устройств.

4. Наконец, диспетчер управления сервисами (SCM) загружает автоматически запускаемые драйверы.

Дерево устройств используется диспетчерами PnP и электропитания в то время, когда они выдают устройствам IRP-пакеты, связанные с Plug and Play и управлением электропитанием. Как правило, поток TRP распространяется от верхней части узла устройств вниз, и иногда какой-либо драйвер в одном из узлов устройств создает новые IRP для передачи другим узлам. О потоках IRP-пакетов, связанных с Plug and Play и управлением электропитанием, мы поговорим позже.

Все устройства, обнаруженные после установки системы; регистрируются в подразделах раздела реестра `HKLM\SYSTEM\CurrentControlSet\Enum`. Этим подразделам присваиваются имена в виде <Перечислитель>\<№_устройства>\<№_экземпляра>, где перечислитель — драйвер шины, №_устройства — уникальный идентификатор устройств данного типа, а №_экземпляра — уникальный идентификатор данного экземпляра этого устройства-Узлы устройств.

Узел устройств, который включает минимум два объекта «устройство», показан на Рис. 24.

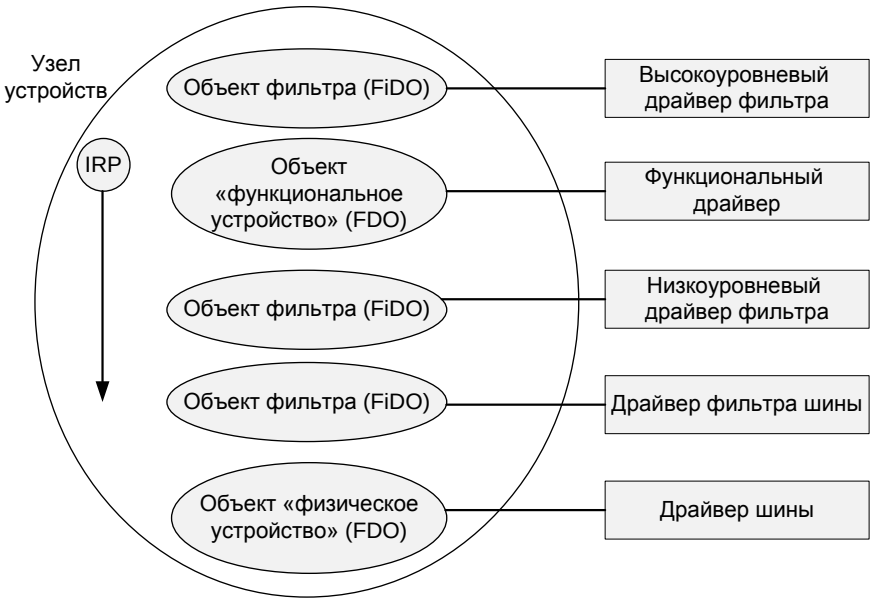

Рис. 24. Внутренняя структура узла устройств

- **Объект «физическое устройство»** (physical device object, PDO). Создается драйвером шины по заданию диспетчера PnP, когда драйвер шины, перечисляя устройства на своей шине, сообщает о наличии какого-либо устройства. PDO представляет физический интерфейс устройства.
- **Необязательные группы объектов-фильтров** (filter device objects, FiDO). Одна группа таких объектов размещается между PDO и FDO (создается драйверами фильтров шин), вторая — между FDO и первой группой FDO (создается низкоуровневыми драйверами фильтров), третья над FDO (создается высокоуровневыми драйверами фильтров).
- **Объект «функциональное устройство»** (functional device object, FDO). Создается функциональным драйвером, который загружается диспетчером PnP для управления обнаруженным устройством. FDO представляет логический интерфейс устройства. Функциональный драйвер может выступать и в роли драйвера шины, если к устройству, представленному FDO, подключены другие устройства. Этот драйвер часто создает интерфейс к PDO, соответствующему данному FDO, что позволяет приложениям и другим драйверам открывать устройство и взаимодействовать с ним. Иногда функциональные драйверы подразделяются на драйвер класса, порт-драйвер и минипорт-драйвер, совместно управляющие вводом-выводом для FDO.

Узлы устройств полагаются на функциональность диспетчера ввода-вывода; поток IRP идет по узлу устройств сверху вниз. Решение об окончании обработки IRP может быть принято на любом уровне узла устройств. Например, функциональный драйвер может обработать запрос на чтение, не пересылая IRP драйверу шины. IRP проходит весь путь сверху вниз и далее к узлу устройств, содержащему драйвер шины, только если функциональному драйверу нужна помощь драйвера шины.

Загрузка драйверов для узла устройств

Существует два важных вопроса:

- как диспетчер PnP определяет, какой функциональный драйвер нужно загрузить для данного устройства;
- как драйверы фильтров регистрируют свое присутствие, чтобы их можно было загружать при создании узла устройств?

Ответ на оба вопроса надо искать в реестре. Перечисляя устройства, драйвер шины сообщает диспетчеру PnP идентификаторы обнаруженных устройств. Эти идентификаторы специфичны для конкретной шины. Например, для шины USB такой идентификатор состоит из **идентификатора изготовителя** (vendor ID; VID) и **идентификатора продукта** (product ID, PID), назначенного устройству изготовителем (подробнее о форматах идентификаторов устройств см. в DDK). В совокупности эти идентификаторы образуют то, что в терминологии спецификации Plug and Play называется **идентификатором устройства** (device ID). Диспетчер PnP также запрашивает у драйвера шины **идентификатор экземпляра** (instance ID), который позволяет различать отдельные экземпляры одного и того же устройства. Идентификатор экземпляра может определять устройство относительно шины (например, USB-порт) или представлять глобально уникальный дескриптор (скажем, серийный номер устройства). Идентификаторы устройства и экземпляра дают **идентификатор экземпляра устройства** (device instance ID, DIID), используемый диспетчером PnP для поиска раздела устройства в ветви реестра `HKLM\SYSTEM\CurrentControlSet\Enum`. В эти разделы помещаются данные, характеризующие устройство, и получаемые из INF-файла параметры Service и ClassGUID, с помощью которых диспетчер PnP находит драйверы, нужные для данного устройства.

Параметр ClassGUID позволяет диспетчеру PnP найти раздел класса устройства в `HKLM\SYSTEM\CurrentControlSet\Control\Class`. Раздел, созданный для устройства в процессе перечисления, и раздел класса предоставляют диспетчеру PnP всю информацию, на основе которой он загружает драйверы, необходимые для узла данного устройства. Загрузка драйверов происходит в следующем порядке.

1. Любые низкоуровневые драйверы фильтров, указанные в параметре `LowerFilters` раздела, созданного для устройства в процессе перечисления.

2. Любые низкоуровневые драйверы фильтров, указанные в параметре `LowerFilters` раздела класса данного устройства.

3. Функциональный драйвер, заданный в параметре `Service` раздела, созданного для устройства в процессе перечисления. Значение этого параметра интерпретируется как имя раздела драйвера в `HKLM\SYSTEM\CurrentControlSet\Services`.

4. Любые высокоуровневые драйверы фильтров, указанные в параметре `UpperFilters` раздела, созданного для устройства в процессе перечисления.

5. Любые высокоуровневые драйверы фильтров, указанные в параметре `UpperFilters` раздела класса данного устройства.

Ссылки на драйверы всегда содержат имена их разделов в `HKLM\SYSTEM\CurrentControlSet\Services`.

Установка драйвера

Если диспетчер PnP встречает устройство, драйвер которого не установлен, он обращается к диспетчеру PnP пользовательского режима, и тот устанавливает нужный драйвер. Если устройство обнаруживается при загрузке системы, для него определяется узел устройств, но загрузка драйверов откладывается до запуска диспетчера PnP пользовательского режима (он реализован в `\Windows\System32\Newdev.dll` и выполняется как сервис в процессе `Services.exe`).

Компоненты, участвующие в установке драйвера, показаны на Рис. 25. Серые блоки на этом рисунке соответствуют компонентам, обычно предоставляемым системой, а остальные блоки — компонентам, предоставляемым установочными файлами. Сначала драйвер шины информирует диспетчер PnP о перечисленном устройстве, сообщая его DIID (1). Диспетчер PnP проверяет, определен ли в реестре подходящий функциональный драйвер. Если нет, он уведомляет диспетчер PnP пользовательского режима (2) о новом устройстве, сообщая его DIID. Диспетчер PnP пользовательского режима пытается автоматически установить драйверы для устройства. Если в процессе установки выводятся диалоговые окна, требующие внимания пользователя, а зарегистрированный в данный момент пользователь имеет привилегии администратора (3), то диспетчер PnP пользовательского режима запускает Rundll32.exe (хост-программу апплетов Control Panel) для выполнения мастера установки оборудования (`\Windows\System32\Newdev.dll`). Если зарегистрированный в данный момент пользователь не имеет привилегий администратора (или в системе нет пользователей), а установка устройства требует взаимодействия с пользователем, диспетчер PnP пользовательского режима откладывает установку до того момента, когда в систему войдет привилегированный пользователь. Для поиска INF-файлов, соответствующих драйверам, совместимым с обнаруженным устройством, мастер установки оборудования использует API-функции Setup и CfgMgr (диспетчера конфигурации). При этом пользователю может быть предложено вставить в один из дисководов носитель с нужными INF-файлами; кроме того, просматриваются INF-файлы в `\Windows\Driver Cache\i386\Driver.cab`, где содержатся драйверы, поставляемые с Windows.

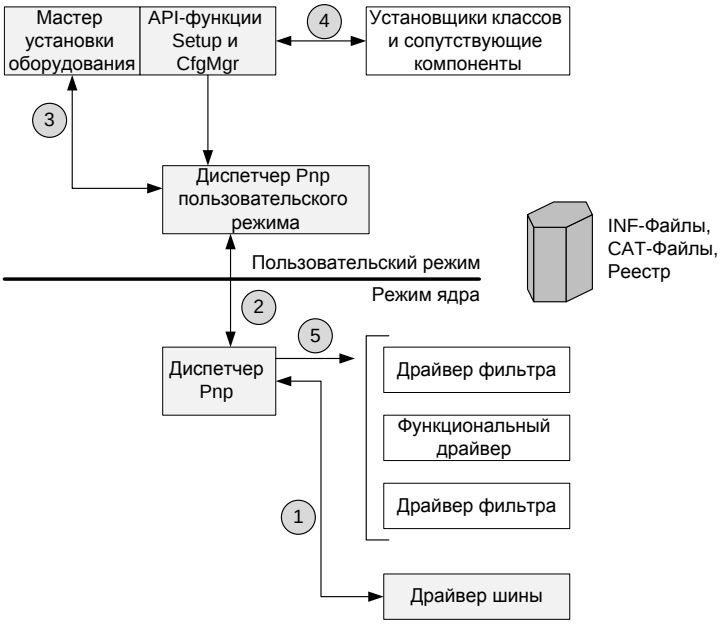

Рис. 25. Компоненты, участвующие в установке драйвера

Чтобы найти драйверы для нового устройства процесс установки получает от драйвера шины список **идентификаторов оборудования** (hardware ID) и **идентификаторов совместимых устройств** (compatible ID). Эти идентификаторы описывают все способы, предусмотренные в установочном файле драйвера (INF-файле) для идентификации устройства. Списки упорядочиваются так, чтобы наиболее специфические характеристики устройства описывались первыми. Если совпадения идентификаторов обнаруживаются в нескольких INF-файлах, предпочтение отдается наиболее полным совпадениям. Аналогичным образом предпочтение отдается INF-файлам с цифровой подписью, а среди них — более новым. Если найденный идентификатор соответствует идентификатору совместимого устройства, мастер установки оборудования может запросить носитель с обновленными драйверами для этого устройства.

INF-файл определяет местонахождение файлов функционального драйвера и содержит команды, которые вводят нужные данные в раздел перечисления и раздел класса драйвера. INF-файл может указать мастеру установки оборудования запустить DLL установщика класса или компонента, участвующего в установке устройства (4), — эти модули выполняют операции, специфичные для класса или устройства, например выводят диалоговые окна, позволяющие настраивать параметры устройства.

Перед установкой драйвера диспетчер PnP пользовательского режима проверяет системную политику проверки цифровых подписей в драйверах. Эта политика хранится в разделе реестра `HKLM\SOFTWARE\Microsoft\Driver Signing\Policy`, если администратор выбрал общесистемную политику, или в `HKCU\Software\Microsoft\Driver Signing\Policy`, если в системе применяются политики только по отношению к индивидуальным пользователям. Политика проверки цифровых подписей в драйверах настраивается через диалоговое окно Driver Signing Options (Параметры подписывания драйвера), доступное с вкладки Hardware (Оборудование) окна свойств системы.

Диспетчер электропитания

Как и PnP-функции Windows, управление электропитанием требует аппаратной поддержки. Она должна отвечать спецификации Advanced Configuration and Power Interface (ACPI). Согласно этой спецификации BIOS (Basic Input Output System) тоже должна соответствовать стандарту ACPI. Этим требованиям удовлетворяет большинство x8б-компьютеров, выпускавшихся с конца 1998 года.

Стандарт ACPI определяет различные уровни энергопотребления для системы и устройств. Шесть состояний для системы — от S0 (полностью активное, или рабочее, состояние) до S5 (полное отключение) — перечислены в таблице 1. Каждое из них характеризуется следующими параметрами.

- **Энергопотребление** (power consumption) Количество энергии, потребляемой компьютером.
- **Возобновление работы ПО** (software resumption) Состояние программного обеспечения при переходе компьютера в «более активное» состояние.
- **Аппаратная задержка** (hardware latency) Время, необходимое на то, чтобы вернуть компьютер в полностью активное состояние.

Таблица 1. Определения состояний системы с различным энергопотреблением

Состояние	Энергопотребление	Возобновление работы ПО	Аппаратная задержка
S0 (fully on) (полностью активное состояние)	Максимальное	-	Нет
S1 (sleeping) (состояние ожидания)	Менее S0, но более S2	Система возобновляет работу с той точки, где она была прервана (возвращается в состояние SO)	Менее 2 секунд

S2 (sleeping) (состояние ожидания)	Менее S1, но более S3	Система возобновляет работу с той точки, где она была прервана (возвращается в состояние S0)	От 2 секунд и более
S3 (sleeping) (состояние ожидания)	Менее S2, CPU отключен	Система возобновляет работу с той точки, где она была прервана (возвращается в состояние S0)	От 2 секунд и более
S4 (hibernating) (спящий режим)	Ток подается на кнопку включения электропитания и в контур пробуждения (wake circuitry)	Система перезапускается с помощью файла спящего режима (hibernate file) и возобновляет работу с той точки, где она была прервана переходом в спящий режим (возвращается в состояние S0)	Длительная, неопределенная
S5 (fully off) (полное отключение)	Ток подается на кнопку включения электропитания	Система загружается заново	Длительная, неопределенная

В состояния ожидания (S1-S4) компьютер кажется отключенным, так как потребляет меньше энергии. Но при этом он сохраняет и в памяти, и на диске всю информацию, необходимую для возврата в состояние S0. В состояниях S1-S3 для сохранения содержимого памяти нужно достаточное количество энергии, поскольку при переходе в S0 (при пробуждении компьютера пользователем или устройством) диспетчер электропитания возобновляет работу системы с той точки, где оно было прервано. Когда система переходит в состояние S4, диспетчер электропитания сохраняет содержимое памяти в сжатой форме в файле спящего режима (Hiberfil.sys), который помещается в корневой каталог системного тома. (Этот файл должен быть такого размера, чтобы в нем могло уместиться несжатое содержимое всей памяти; сжатие используется для того, чтобы свести к минимуму операции ввода-вывода на диске, а также ускорить переход в спящий режим и выход из него.) Сохранив содержимое памяти, диспетчер электропитания отключает компьютер. При последующем включении компьютера происходит обычный процесс загрузки — с тем исключением, что Ntldr проверяет наличие действительного образа памяти, сохраненного в файле спящего режима. Если в этом файле сохранены данные о состоянии системы, Ntldr считывает его содержимое в память и возобновляет выполнение с точки, зафиксированной в Hiberfil.sys,

Компьютер никогда не переходит напрямую между состояниями S1 и S4, для этого ему нужно сначала перейти в состояние S0. Как показано на Рис. 26, переход системы из состояний S1-S5 в состояние S0, называется пробуждением (waking), а переход из состояния S0 в состояния S1-S5 — переходом в сон (sleeping).

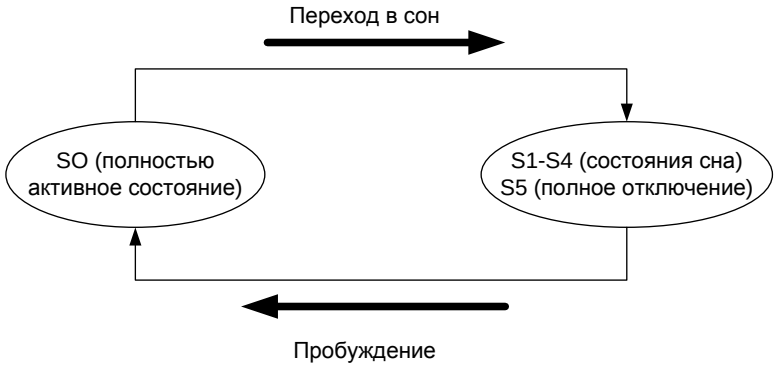

Рис. 26. Переходы системы между различными состояниями энергопотребления

Хотя система может пребывать в одном из шести состояний энергопотребления, ACPI определяет для устройств четыре состояния: D0-D3. В состоянии D0 устройство полностью включено, а в состоянии D3 полностью отключено. ACPI позволяет драйверам и устройствам самостоятельно определять состояния D1 и D2 с единственным условием, что устройство в состоянии D1 должно потреблять столько же

или меньше энергии, чем в состоянии D0, а в состоянии D2 — столько же или меньше, чем в состоянии D1.

Работа диспетчера электропитания

Политика управления электропитанием в Windows определяется диспетчером электропитания и драйверами устройств. Владельцем системной политики управления электропитанием является диспетчер электропитания. Это значит, что он принимает решение о том, в каком состоянии энергопотребления должна находиться система в текущий момент. При необходимости выключения либо перехода в ждущий или спящий режим диспетчер электропитания указывает устройствам, поддерживающим управление электропитанием, перейти в соответствующее состояние. Этот диспетчер принимает решение о переходе в другое состояние энергопотребления, исходя из:

- уровня активности системы;
- уровня заряда аккумуляторов;
- наличия запросов приложений на выключение компьютера или переход в ждущий/спящий режим;
- действий пользователя, например нажатия кнопки включения электропитания;
- параметров электропитания, заданных в Control Panel.

Часть информации, получаемой диспетчером PnP при перечислении устройств, связана с поддержкой устройствами функций управления электропитанием. Драйвер сообщает, поддерживает ли устройство состояния D1 и D2, а также какие задержки требуются ему для перехода из состояний D1-D3 в D0 (последняя часть данных необязательна). Чтобы диспетчеру было легче определять, когда систему следует переводить в другое состояние энергопотребления, драйверы шин также возвращают таблицу сопоставлений между системными состояниями (S0-S5) и состояниями, поддерживаемыми конкретным устройством. В этой таблице указывается состояние устройства с наименьшим энергопотреблением для каждого системного состояния.

Таблица 2. Пример таблицы сопоставлений системных состояний с состояниями устройств

Состояние системы	Состояние устройства
S0 (полностью активное)	D0 (полностью активное)
S1 (ждущий режим)	D2
S2 (ждущий режим)	D2
S3 (ждущий режим)	D2
S4 (спящий режим)	D3 (полное отключение)
S5 (полное отключение)	D3 (полное отключение)

Участие драйверов в управлении электропитанием

Диспетчер электропитания, принимая решение о переходе системы в другое состояние, посылает команды процедуре драйвера, отвечающей за диспетчеризацию электропитания. Управлять устройством могут несколько драйверов, но только один из них является владельцем политики управления электропитанием устройства. Этот драйвер определяет состояние устройства в зависимости от состояния энергопотребления системы. Например, при переходе системы из состояния S0 в состояние S1 драйвер может принять решение о переводе устройства из состояния D0 в состояние D1. Вместо того чтобы напрямую оповещать об этом другие драйверы, участвующие в управлении устройством, владелец политики управления электропитанием устройства делает это через диспетчер электропитания, вызывая функцию `PoRequestPowerIrp`.

Как драйвер управляет электропитанием устройства

Драйвер не только отвечает на команды диспетчера электропитания, связанные с изменением состояния системы, но и может сам управлять состоянием энергопотребления своих устройств. В некоторых случаях драйвер может снизить

энергопотребление управляемого им устройства, если оно неактивно в течение определенного времени. Драйвер может обнаруживать простаивающие устройства самостоятельно или через механизмы, предоставляемые диспетчером электропитания. Во втором случае устройство регистрируется в диспетчере электропитания вызовом функции `PoRegisterDeviceForIdleDetection`. Эта функция сообщает диспетчеру электропитания пороговые интервалы простоя устройства и указывает, в какое состояние следует переводить устройство, если оно простаивает. Драйвер задает два таймаута:

- первый — для энергосберегающей конфигурации,
- второй — для максимально производительной.

Вызвав `PoRegisterDeviceForIdleDetection`, драйвер должен уведомлять диспетчер электропитания об активности устройства через функцию `PoSetDeviceBusy`.

Управление внешней памятью

Термин **внешняя память** (storage) относится к носителям, применяемым в самых разнообразных устройствах, в том числе к магнитным лентам, оптическим дискам, гибким дискам, локальным жестким дискам и сети устройств хранения данных (storage area networks, SAN). Windows предоставляет специализированную поддержку для каждого класса носителей внешней памяти.

Базовая терминология

Чтобы полностью усвоить материал следует четко понимать базовую терминологию.

- **Диск** — физическое устройство внешней памяти, например жесткий диск, 3,5-дюймовая дискета или компакт-диск (CD-ROM).
- Диск делится на **секторы**, блоки фиксированного размера. Размер сектора определяется аппаратно. Например, размер сектора жесткого диска, как правило, составляет 512 байтов, а размер сектора CD-ROM — обычно 2048 байт.
- **Раздел** (partition) — набор непрерывных секторов на диске. Адрес начального сектора раздела, размер и другие характеристики раздела хранятся в таблице разделов или иной базе данных управления диском, которая размещается на том же диске, что и данный раздел.
- **Простой том** (simple volume) — объект, представляющий секторы одного раздела, которым драйверы файловых систем управляют как единым целым.
- **Составной том** (multipartition volume) — объект, представляющий секторы нескольких разделов, которыми драйверы файловых систем управляют как единым целым. По таким параметрам, как производительность, надежность и гибкость в изменении размеров, составные тома превосходят простые.

Драйверы дисков

Драйверы устройств, участвующие в управлении конкретным устройством внешней памяти (накопителем), обобщенно называются стеком **драйверов внешней памяти** (storage stack). На Рис. 27 показаны все типы драйверов, которые могут присутствовать в стеке. Опишем поведение драйверов устройств, расположенных в стеке ниже уровня файловой системы.

Рис. 27. Стек драйверов устройств внешней памяти в Windows

Ntldr

Первой частью процесса загрузки ОС Windows дирижирует Ntldr. С технический точки зрения Ntldr не является частью стека внешней памяти, он участвует в управлении ею, поскольку предоставляет поддержку для доступа к дисковым устройствам до того, как начнет работать подсистема ввода-вывода Windows. Он находится на системном томе и запускается кодом, размещенным в загрузочном секторе этого тома. Ntldr считывает с системного тома файл Boot.ini и предлагает пользователю выбрать вариант загрузки. Имена разделов в Boot.ini представлены в виде `multi(0)disk(0)rdisk(0)partition(1)`. Эти имена являются частью стандартной схемы именования разделов Advanced RISC Computing (ARC), используемой микрокодом Alpha и других RISC-процессоров. Ntldr транслирует имя выбранного пользователем элемента Boot.ini в имя загрузочного раздела и загружает в память системные файлы Windows (начиная с реестра, Ntoskrnl.exe и загрузочных драйверов). Во всех случаях Ntldr использует BIOS для чтения диска, содержащего системный том, но, иногда полагается на функции минипорт-драйвера диска для чтения с диска, где находится загрузочный том.

Драйвер класса дисков, порт- и минипорт-драйверы

При инициализации диспетчер ввода-вывода запускает драйверы жестких дисков. Драйверы устройств внешней памяти в Windows соответствуют архитектуре «класс-порт-минипорт». Согласно этой архитектуре, Microsoft предоставляет драйвер класса внешней памяти, который реализует функциональность, общую для всех устройств внешней памяти, и порт-драйвер, который поддерживает функциональность, общую для конкретной шины, например SCSI (Small Computer System Interface) или IDE (Integrated Device Electronics). А изготовители оборудования поставляют минипорт-драйверы, подключаемые к порт-драйверам и формирующие интерфейс между Windows и конкретными устройствами.

В архитектуре драйверов дисковой памяти только драйверы класса имеют стандартные интерфейсы драйверов устройств Windows. Минипорт-драйверы вместо интерфейса драйверов устройств используют интерфейс порт-драйверов, который просто реализует набор процедур, служащих интерфейсом между Windows и минипорт-драйверами.

Такой подход упрощает разработку минипорт-драйверов, поскольку Microsoft предоставляет порт-драйверы, специфичные для ОС, а также обеспечивает переносимость минипорт-драйверов на уровне двоичного кода между Windows 98, Windows Millennium Edition и Windows.

Windows включает драйвер класса дисков (`\Windows\System32\Drivers\Disksys`), реализующий стандартную функциональность дисков Windows также предоставляет разнообразные порт-драйверы дисков. Например, `Scsi-port.sys` — это порт-драйвер дисков, подключаемых к SCSI-шине, а `Atapi.sys` — порт-драйвер для систем на базе IDE. В Windows Server 2003 введен портдрайвер `Storport.sys`, заменяющий `Scsiport.sys`, `Storport.sys` был разработан для реализации функциональности высокопроизводительных аппаратных RAID-контроллеров и адаптеров Fibre Channel. Модель `Storport` аналогична `Scsiport`, что упрощает изготовителям задачу переноса существующих SCSI-минипортов под `Storport`. Минипорт-драйверы, создаваемые разработчиками для использования `Storport`; используют преимущества нескольких механизмов `Storport`, повышающих производительность, в частности поддержки параллельной инициации и завершения запросов на ввод-вывод в multi-CPU системах, более управляемой архитектуры очереди запросов на ввод-вывод и выполнения большей части кода при более низком уровне IRQL, чтобы свести к минимуму длительность маскирования аппаратных прерываний.

Драйверы `Scsiport.sys` и `Atapi.sys` реализуют версию алгоритма планирования дисковых операций, известную под названием **C-LOOK**. Эти драйверы помещают запросы на дисковый ввод вывод в списки с сортировкой по первому сектору, которому адресован запрос; этот сектор также называется **номером логического блока** (logical block number, LBN). С помощью функций `KelnsertByKeyDeviceQueue` и `KeRemoveByKeyDeviceQueue` (документированных в Windows DDK) они представляют запросы ввода-вывода как элементы (items) и используют начальный сектор запроса в качестве ключа, требуемого этими функциями. Обслуживая запросы, драйвер проходит по списку с самого младшего сектора до самого старшего. Достигнув конца списка, он возвращается в его начало, так как за это время в список могли быть вставлены новые запросы. Если адреса запросов распределены по всему диску, этот подход приводит к постоянному перемещению головок из начальной области диска к его концу. `Storport.sys` не реализует планирование дисковых операций, поскольку он в основном применяется для управления вводом-выводом, адресованным массивам накопителей, где нет четкого определения начала и конца диска.

Драйверы iSCSI

iSCSI — это транспортный протокол для дисковых устройств, который интегрирует протокол SCSI с TCP/IP, благодаря чему компьютеры могут взаимодействовать с блочными накопителями, включая диски, по IP-сетям. Архитектура **сети устройств хранения данных** (storage area networking, SAN) обычно базируется на сети Fibre Channel, но администраторы могут использовать iSCSI для создания сравнительно недорогих SAN на основе таких сетевых технологий, как гигабитная Ethernet, что позволяет обеспечить масштабируемость, защиту от катастроф, эффективное резервное копирование и защиту данных. В Windows поддержка iSCSI реализуется в виде Microsoft iSCSI Software Initiator, который можно скачать с сайта Microsoft и который работает в Windows 2000, Windows XP и Windows Server 2003.

Microsoft iSCSI Software Initiator включает несколько компонентов.

- **Initiator** (инициатор). Этот необязательный компонент, состоящий из порт-драйвера iSCSI (`\Windows\Systern32\Drivers\Iscsiprt.sys`) и мини-порт-драйвера (`\Windows\System32\Drivers\Msiscis.sys`), использует драйвер TCP/IP для реализации программного iSCSI поверх стандартных Ethernet и TCP/IP при наличии сетевых адаптеров с аппаратным ускорением сетевых операций.
- **Initiator Service** (служба инициатора). Эта служба, реализованная в `\Windows\System32\Iscsiexe.exe`, управляет обнаружением и защитой всех инициаторов iSCSI, а также инициацией и завершением сеансов. Функциональность обнаружения устройств iSCST реализована в `\Windows\System32\Iscsiurii.dll` и соответствует спецификации протокола Internet Storage Name Service (iSNS).
- **Управляющие приложения**. К ним относятся Iscsicli.exe (утилита командной строки для управления соединениями iSCSI-устройств и их защитой) и соответствующий апплет для Control Panel (Панель управления).

MPIO-драйверы

У большинства дисковых устройств только один путь (path) между ними и компьютером — набор адаптеров, кабелей и коммутаторов. В серверах, требующих высокого уровня готовности к работе, применяются решения с несколькими путями — между компьютером и диском существует более одного набора соединительного оборудования, чтобы при аварии одного пути система все равно могла бы обращаться к диску пи альтернативному пути. Однако без поддержки со стороны ОС или драйверов диск с двумя путями будет виден как два разных диска. Windows включает поддержку **ввода-вывода по нескольким путям** (multipath I/O, MPIO) для управления дисками с несколькими путями как одним диском; эта поддержка опирается на сторонние драйверы — модули, специфичные для конкретного устройства (device-specific modules, DSM). Эти модули берут на себя всю специфику управления путями, в том числе политику балансировки нагрузки, на основе которой выбирается путь передачи запросов ввода вывода, и механизмы обнаружения ошибок, уведомляющие Windows об аварии того или иного пути. Поддержка MPIO доступна для Windows 2000 Server, Advanced Server, Datacenter Server и Windows Server 2003 в виде Microsoft MPIO Driver Development Kit, который лицензируется поставщиками аппаратного и программного обеспечения.

В стеке драйверов внешней памяти Windows MPIO (Рис. 28) Multipath Disk Driver Replacement (`\Windows\Systern32\Drivers\Mpdev.sys`) заменяет функциональность стандартного драйвера класса `Disk.sys`. `Mpdev.sys` захватывает во владение объект «устройство», представляющий диски с несколькими путями, чтобы для таких дисков существовал лишь один объект «устройство». Кроме того, этот драйвер отвечает за поиск подходящего DSM для управления путями к устройству. Multipath Bus Driver (`\Windows\System32 \Drivers\Mpio.sys`) управляет соединениями между компьютером и устройством, в том числе обеспечивая управление электропитанием данного устройства. `Mpdcv.sys` уведомляет `Mpio.sys` о наличии устройств, которые тот должен контролировать. Наконец, Multipath Port Filter (`\Windows\System32\Drivers\Mpsfltr.sys`) размещается поверх порт-драйвера для диска с несколькими путями и управляет информацией, передаваемой вверх по стеку устройств.

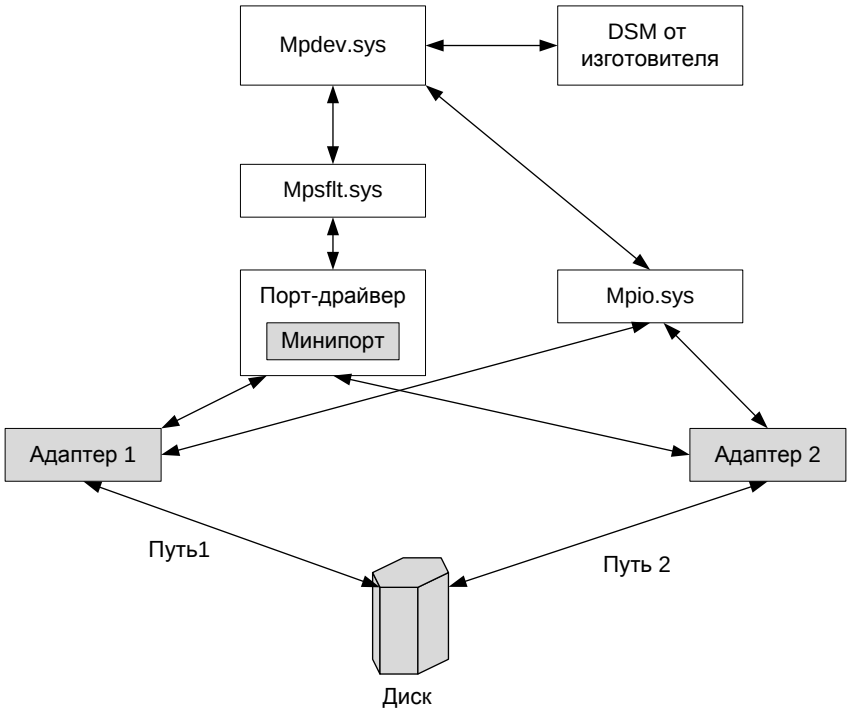

Рис. 28. Стек устройств внешней памяти Windows MPIO

Объекты «устройство» для дисков

Драйвер класса дисков создает объекты «устройство», представляющие диски и дисковые разделы. Имена таких объектов имеют вид `\Device\HarddiskX\DRX`, где X — номер диска. Для идентификации разделов и создания объектов «устройство», представляющих эти разделы, драйвер класса дисков в Windows 2000 использует функцию `IoReadPartitionTable` диспетчера ввода-вывода, а в Windows XP и

Windows Server 2003 — функцию `IoReadPartitionTableEx`. Драйвер класса дисков вызывает одну из этих функций для каждого диска, представленного минипорт-драйвером драйверу класса на ранних стадиях загрузки системы. А функция инициирует дисковый ввод-вывод на уровне секторов, поддерживаемый драйвером класса, порт- и минипорт-драйверами, для считывания MBR- или GPT-таблицы разделов и для формирования внутреннего представления жестких разделов диска. Драйвер класса дисков создает объекты «устройство», представляющие все главные разделы (в том числе логические диски внутри дополнительных разделов), которые этот драйвер получает от `IoReadPartitionTable` или `IoReadPartitionTableEx`. Вот пример имени объекта раздела:

```
\Device\Harddisk0\DP(1)0x7e000-0x7ff50c00+2
```

Это имя идентифицирует первый раздел первого диска системы. Два первых шестнадцатеричных числа (`0x7e000` и `0x7ff50c00`) определяют начало и длину раздела, а последнее число — внутренний идентификатор, назначенный драйвером класса.

Для совместимости с приложениями, использующими правила именования, принятые в Windows NT 4, драйвер класса дисков формирует для имен в формате Windows NT 4 символьные ссылки на объекты «устройство», созданные драйвером. Например, драйвер класса создает ссылки `\Device\Harddisk0\Partition0` на `\Device\Harddisk0\DR0` и `\Device\Harddisk0O\Partition1` на объект «устройство» первого раздела первого диска. В Windows драйвер класса создает такие же символьные ссылки, представляющие физические диски, созданные в системах под управлением Windows NT 4. Так, ссылка `\??\PhysicalDrive0` указывает на `\Device\Harddisk0\DR0`.

Windows API ничего не знает о пространстве имен диспетчера объектов. Windows резервирует два подкаталога пространства имен, один из которых — подкаталог `\Global??` (`\??` в Windows 2000). В этом подкаталоге объекты «устройство», включая диски, последовательные и параллельные порты, становятся доступными Windows-приложениям. Так как на самом деле объекты дисков находятся в других подкаталогах, для связывания имен в `\Global??` с объектами, расположенными в других каталогах пространства имен, Windows использует символьные ссылки. Диспетчер ввода вывода создает ссылку `\Global??\PhysicalDriveX` для каждого физического диска системы; такая ссылка указывает на `\Device\HarddiskX\Partition0` (где `X`—числа, начиная с 0). Windows-приложения, напрямую обращающиеся к секторам диска, открывают диск вызовом Windows-функции `CreateFile` и указывают в качестве параметра имя `\\.\PhysicalDriveX` (где `X` — номер диска). Прежде чем передать имя диспетчеру объектов, прикладной уровень Windows преобразует его в `\Global??\PhysicalDriveX`.

Домашнее задание

Самостоятельно следует ознакомиться с управлением томами [3]:

- Базовые диски.
- Динамические диски.
- Управление составными томами.
- Пространство имен томов.
- Операции ввода-вывода на томах.
- Служба виртуального диска.
- Служба теневого копирования тома.

Литература

1. Э. Таненбаум. Современные операционные системы. 2-ое изд. –СПб.: Питер, 2002. – 1040 с.

2. Э. Таненбаум, А. Вудхалл. Операционные системы: разработка и реализация. Классика CS. –СПб.: Питер, 2006. –576 с.

3. М. Руссинович, Д. Соломон. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Мастер-класс. / Пер. с англ. -4-е изд. –М.: Издательско-торговый дом «Русская редакция»; СПб.: Питер; 2005. -992 с.

4. Microsoft Development Network. URL: http://msdn.com

5. Набор спецификаций Microsoft по управлению питанием устройствами ОС. URL: www.microsoft.com/whdc/resources/respec/specs/pmref