Оглавление

[1. Понятие операционной системы. Перечень известных операционных систем. Взгляд на ОС с разных точек зрения. Эволюция ОС. Основные задачи ОС. Основные понятия, связанные с ОС. 2](#_Toc29310021)

[2. Основные подходы к организации архитектуры ОС. Классификация ОС. Роли пользователей ОС. 6](#_Toc29310022)

[3. Основные понятия ОС Windows (процессы, потоки, задания, виртуальная память). 9](#_Toc29310023)

[4. Общая организация ОС Windows: режим ядра и пользовательский режим, объекты и описатели, безопасность, реестр. 10](#_Toc29310024)

[5. Обзор архитектуры ОС Windows: компоненты архитектуры ОС и основные особенности. Подсистемы. 12](#_Toc29310025)

[6. ОС Windows: процессы (понятие процесса, структуры данных, переменные ядра, сопутствующие функции). 13](#_Toc29310026)

[7. Создание процессов в ОС Windows. 17](#_Toc29310027)

[8. Управление процессами в ОС Windows. 19](#_Toc29310028)

[9. ОС Windows: потоки (внутреннее устройство потоков, структуры данных, переменные ядра, сопутствующие функции, создание потоков). 20](#_Toc29310029)

[10. Планирование потоков: уровни приоритета процессов и потоков, состояния потоков. 24](#_Toc29310030)

[11. Кванты времени, выделяемые потокам, управление величиной кванта. 26](#_Toc29310031)

[12. Сценарии планирования выполнения потоков, переключение контекста, поток простоя. 27](#_Toc29310032)

[13. Особенности планирования потоков для многопроцессорных систем. 30](#_Toc29310033)

[14. Диспетчер памяти ОС Windows: компоненты диспетчера, синхронизация, конфигурирование. 31](#_Toc29310034)

[15. Диспетчер памяти ОС Windows: сервисы диспетчера памяти. Страницы, блокировка, гранулярность выделения памяти, резервирование памяти, разделяемая память и проецируемые файлы. 33](#_Toc29310035)

[16. Защита памяти ОС Windows. 35](#_Toc29310036)

[17. Системные пулы памяти. 37](#_Toc29310037)

[18. Структуры виртуального адресного пространства ОС Windows. 38](#_Toc29310038)

[19. Трансляция виртуальных адресов. Операции ввода-вывода, связанные с подкачкой страниц. Обработка ошибок. 39](#_Toc29310039)

[20. Страничные файлы. Дескрипторы виртуальных адресов. Объекты-разделы. 41](#_Toc29310040)

[21. Рабочие наборы. Управление рабочими наборами. 42](#_Toc29310041)

[22. Управление безопасностью ОС Windows: классы безопасности, компоненты системы защиты. 43](#_Toc29310042)

[23. Защита объектов в ОС Windows: проверка прав доступа. Идентификаторы защиты. Маркеры. Олицетворение. 43](#_Toc29310043)

[24. Управление безопасностью ОС Windows: дескрипторы защиты и управление доступом. Определение прав доступа. 43](#_Toc29310044)

[25. Права и привилегии учетных записей, суперпривелегии, аудит безопасности ОС Windows. Вход в систему, политика ограниченного использования программ. 43](#_Toc29310045)

[26. Управление вводом-выводом в ОС Windows: компоненты подсистемы ввода-вывода, диспетчер, типичная обработка ввода-вывода. 43](#_Toc29310046)

[27. Драйверы устройств в ОС Windows: типы драйверов, основные процедуры драйвера устройства. Объекты Windows: «драйвер» и «устройство». 43](#_Toc29310047)

[28. Обработка ввода-вывода: типы ввода-вывода (синхронный, асинхронный, быстрый, ввод-вывод в проецируемые файлы). Пакеты запроса ввода-вывода, блок стека IRP, управление буфером IRP. 43](#_Toc29310048)

[29. Процесс обработки запросов синхронного ввода-вывода к одноуровневому драйверу режима ядра. Запрос ввода-вывода к многоуровневому драйверу. Порты завершения ввода-вывода. 43](#_Toc29310049)

[30. Диспетчер Plug & Play. Загрузка, инициализация и установка драйвера. 43](#_Toc29310050)

[31. Диспетчер электропитания. 43](#_Toc29310051)

[32. Управление внешней памятью в ОС Windows: терминология, драйверы дисков, объекты «устройство» для дисков. 43](#_Toc29310052)

[33. Файловые системы ОС Windows (обзор): NTFS, FAT, прочие. 43](#_Toc29310053)

[34. Архитектура драйвера файловой системы. 43](#_Toc29310054)

[35. Цели разработки и особенности NTFS. Драйвер файловой системы NTFS. 43](#_Toc29310055)

[36. Структура NTFS на диске. 43](#_Toc29310056)

[37. Поддержка восстановления в NTFS. Механизм EFS. 43](#_Toc29310057)

[38. Функции синхронизации процессов при работе с Windows. 43](#_Toc29310058)

[39. Объекты синхронизации Windows (основные). 43](#_Toc29310059)

[40. Дополнительные объекты и механизмы синхронизации Windows 43](#_Toc29310060)

[41. Реализация прерываний, системных вызовов и исключений в ОС Windows 43](#_Toc29310061)

[42. Организация сетевого взаимодействия (Windows): Сокеты. Каналы. 43](#_Toc29310062)

[43. Архитектура UNIX (основная концепция, архитектура, ядро). 43](#_Toc29310063)

[44. UNIX: файловая система, управление процессами. 43](#_Toc29310064)

[45. Общая архитектура ОС Android. Особенности ядра. 43](#_Toc29310065)

[46. ОС Android: Java-машина Dalvik. Обзор Java-интерфейсов прикладного программиста. 43](#_Toc29310066)

1. Понятие операционной системы. Перечень известных операционных систем. Взгляд на ОС с разных точек зрения. Эволюция ОС. Основные задачи ОС. Основные понятия, связанные с ОС.

**Операционная система** — комплекс программ, обеспечивающий управление аппаратными средствами компьютера, организующий работу с файлами и выполнение прикладных программ, осуществляющий ввод и вывод данных.

**Операционная система (ОС)** - комплекс системных и управляющих программ, предназначенных для наиболее эффективного использования всех ресурсов вычислительной системы (ВС) (Вычислительная система - взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации) и удобства работы с ней.

**Перечень известных операционных систем**

**UNIX**

Операционная система UNIX была разработана группой сотрудников Bell Labs под руководством Денниса Ричи, Кена Томпсона и Брайана Кернигана в 1969 году.

**BSD**

В конце 1970-х годов сотрудники Калифорнийского университета в Беркли внесли ряд усовершенствований в исходные коды UNIX, включая работу с протоколами TCP/IP. Их разработка стала известна под именем BSD — Berkeley Systems Distribution. Она распространялась под лицензией, позволяющей дорабатывать и совершенствовать продукт и передавать результат третьим лицам, с исходными кодами или без них, при условии указания авторства кода, написанного в Беркли.

**Linux**

В начале 1990-х годов студент Хельсинкского университета Линус Торвальдс начал разработку ядра новой ОС для IBM-совместимых ПК, которое было названо Linux. В настоящее время GNU/Linux (совокупность различных дистрибутивов построенных на базе ядра Linux) стоит на втором месте по популярности среди ОС, используемых на рабочих столах пользователей (первое место принадлежит Microsoft Windows).

**Amiga OS**

Операционная система для персональных компьютеров семейства Amiga (процессор Motorola 68k), имеет атипичное микроядро называемое Exec.

DOS

В 1980 Тимом Патерсоном (Tim Paterson) из Seattle Computer Products (SCP) была создана QDOS (Quick and Dirty Operating System). QDOS, по большей части, была 16-разрядным клоном CP/M, но с новой файловой системой — FAT. QDOS была переименована в 86-DOS, поскольку разрабатывалась для работы на процессоре Intel 8086. Microsoft приобрела QDOS за $60 000 и продала её IBM уже как PC-DOS (MS-DOS).

1 августа 1984 IBM объявляет о выпуске нового поколения персональных компьютеров — IBM PC/AT.

**Free DOS**

FreeDOS — свободно-распространяемая функциональная копия известной операционной системы MS-DOS.

**FreeBSD**

FreeBSD Свободная операционная система семейства Unix, потомок AT&T Unix по линии BSD, создана Калифорниским Университетом Беркли

**Mac OS**

Mac OS - Unix подобная операционная система, разработанна компанией Apple Computers Co. в 1984 году, под названием "System 1". В 1997 году, была выпушена 8-я версия операционной системы, и операционная система была переиминована в Mac OS (Полное название: Macintosh Operating System).

**macOS 10**

macOS - десятый выпуск линейки Mac OS, разрабатываеммая и выпускаемая американской компанией Apple Computers Co. (Сегодня: Apple Inc.). В отличее от Mac OS 9, macOS X была разработана на ядре XNU, и имела код из FreeBSD. Начиная с версий OS X Yosemite, была введена функция Hand Off. С этой функцией можно было взаимодействовать с iOS и перехватывать данные приложения, и передать данные на iOS устройства c запушенным приложением.

**Microsoft Windows**

Microsoft Windows — это семейство операционных систем компании Microsoft. Была создана для компьютеров IBM с поддержкой MS-DOS

**IBM OS/2**

OS/2 — операционная система, разрабатывавшаяся компанией IBM (первоначально совместно с Microsoft, позже самостоятельно). В настоящее время работы над клиентскими версиями прекращены, в связи с широким распространением операционных систем семейства Windows NT. Серверные версии продолжают поддерживаться в связи с широким ореолом внедрения. Широко использовалась в США, в банковской и производственной сферах, а также в России, в банкоматах.

**ReactOS**

ReactOS — операционная система, один из проектов сообщества Open Source. В ходе разработки предполагается добиться полной совместимости с приложениями и драйверами Microsoft Windows(R) NT4. Это открытая операционная система, основанная на принципах архитектуры Windows NT® (такие продукты компании Microsoft, как Windows XP, Windows 7, Windows Server 2012 построены на архитектуре Windows NT). Система была разработана с нуля, и таким образом не основана на Linux и не имеет ничего общего с архитектурой UNIX.

**Plan 9**

Plan9 — Операционная система, разработанная в Bell Labs — колыбели UNIX и языка Си. Построена на идее использования файловых иерархий для представления любых ресурсов операционной системы и оборудования. Идеально подходит для построения распределенных систем.

**Inferno OS**

Inferno — продолжатель идей Plan9, отличительной особенностью которой является малые требования к ресурсам компьютера и возможность работы как поверх установленной ОС, так и самостоятельно. VitaNuova

**Menuet OS**

Menuet - самостоятельная операционная система, написанная на языке программирования ассемблер. 64-битная версия коммерческая и требуется оплата.

Колибри ОС

Колибри - ответвление от Menuet OS, в отличие от Menuetа, полностью бесплатная.

**iOS**

iOS - (до 24 июня 2010 года — iPhone OS) — мобильная операционная система для смартфонов, электронных планшетов, носимых проигрывателей и некоторых других устройств, основанная на macOS X, разрабатываемая и выпускаемая американской компанией Apple.

**ANDROID**

ANDROID - операционная система, основана на ядре Linux, предназначена для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, фитнесс-браслетов, игровых приставок, ноутбуков, нетбуков, смартбуков, Очков Google Glass[3], телевизоров[4] и других устройств (в 2015 году появилась поддержка автомобильных развлекательных систем[5] и бытовых роботов).

**Взгляд на ОС с разных точек зрения**

***Операционная система как виртуальная машина***

Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать. Это концептуально проще, чем заботиться о деталях перемещения головок дисков или организации работы мотора. Аналогичным образом, с помощью простых и ясных абстракций, скрываются от программиста все ненужные ему подробности организации прерываний, работы таймера, управления памятью и т.д. Более того, на современных вычислительных комплексах может быть создана иллюзия неограниченного размера операционной памяти и числа процессоров. Всем этим занимается операционная система.

**Операционная система как менеджер ресурсов**

Операционная система предназначена для управления всеми частями весьма сложной архитектуры компьютера. Представим для примера, что случится, если несколько программ, работающих на одном компьютере, будут пытаться одновременно осуществлять вывод на принтер. Мы получили бы неупорядоченную смесь строчек и страниц, выведенных различными программами. Операционная система предотвращает хаос такого рода за счет буферизации информации, предназначенной для печати, на диске и организации очереди на печать. Для многопользовательских компьютеров, необходимость управления ресурсами и их защиты еще более очевидна.

Следовательно, операционная система как менеджер ресурсов, осуществляет упорядоченное и контролируемое распределение процессоров, памяти и других ресурсов между различными программами, их использующими.

**Операционная система как защитник пользователей и программ**

Если вычислительная система допускает совместную работу нескольких пользователей, то возникает проблема организации их безопасной деятельности. Необходимо обеспечить сохранность информации на диске, чтобы никто не мог удалить или повредить чужие файлы. Нельзя разрешить программам одних пользователей произвольно вмешиваться в работу программ других пользователей. Нужно пресекать попытки несанкционированного использования вычислительной системы. Всю эту деятельность осуществляет операционная система как организатор безопасной работы пользователей и их программ.

**Операционная система как постоянно функционирующее ядро**

Наконец, можно дать и такое определение: операционная система — это программа, постоянно работающая на компьютере и взаимодействующая со всеми прикладными программами. Казалось бы, это абсолютно правильное определение, но во многих современных операционных системах постоянно работает на компьютере лишь часть операционной системы, которую принято называть ее ядром.

**Реализация защитных механизмов**. Программы не должны иметь самостоятельного доступа к распределению ресурсов, что приводит к появлению привилегированных и непривилегированных команд. Привилегированные команды, например, команды ввода-вывода, могут исполняться только операционной системой. Говорят, что она работает в привилегированном режиме. Переход управления от прикладной программы к ОС сопровождается контролируемой сменой режима. Во-вторых, это защита памяти, позволяющая изолировать конкурирующие пользовательские программы друг от друга, а ОС от программ пользователей.

Все современные процессоры поддерживают, как минимум, два режима: привилегированный режим (он же режим ядра, kernel mode) и непривилегированный (режим задачи, режим пользователя, user mode). Программы, работающие в режиме ядра, имеют полный, неограниченный доступ ко всем ресурсам компьютера: его командам, адресам, портам ввода/вывода и т.п. В режиме задачи возможности программы ограничены, она, в частности, не может выполнить некоторые специальные команды. Аппаратное разграничение возможностей является абсолютно необходимым условием реализации надежной защиты данных в многопользовательской системе. Отсюда вытекает и определение ядра как части ОС, работающей в режиме ядра. Все остальные программы, как системные утилиты, так и программы пользователей, работают в режиме пользователя и должны обращаться к ядру для выполнения многих системных действий.

**Обработка прерываний.** Внешние прерывания оповещают ОС о том, что произошло асинхронное событие, например, завершилась операция ввода-вывода. Внутренние прерывания (сейчас их принято называть исключительными ситуациями) возникают, когда выполнение программы привело к ситуации, требующей вмешательства ОС, например, деление на ноль или попытка нарушения защиты.

**Организация мультипрограммирования**

Интерфейс между прикладной программой и ОС был организован при помощи набора системных вызовов.

Организация очереди из заданий в памяти и выделение процессора одному из заданий потребовали планирования заданий.

Для переключения процессора с одного задания на другое возникла потребность в сохранении содержимого регистров и структур данных, необходимых для выполнения задания, иначе говоря, контекста, для обеспечения правильного продолжения вычислений.

Поскольку память является ограниченным ресурсом, оказались нужны стратегии управления памятью, то есть потребовалось упорядочить процессы размещения, замещения и выборки информации из памяти.

Так как программы могут пожелать произвести санкционированный обмен данными, стало необходимо их обеспечить средствами коммуникации.

Для корректного обмена данными необходимо предусмотреть координацию программами своих действий, т.е. средства синхронизации.

**Основные функции классических операционных систем:**

• Планирование заданий и использования процессора.

• Обеспечение программ средствами коммуникации и синхронизации.

• Управление памятью.

• Управление файловой системой.

• Управление вводом-выводом.

• Обеспечение безопасности

Каждая из приведенных функций обычно реализована в виде подсистемы, являющейся структурным компонентом ОС. В каждой конкретной операционной системе эти функции, конечно, реализовывались по-своему, в различном объеме. Они не были придуманы как составные части деятельности операционных систем изначально, а появились в процессе развития, по мере того, как вычислительные системы становились удобнее, эффективнее и безопаснее.

Дополнительные функции:

 параллельное или псевдопараллельное выполнение задач (многозадачность).

 взаимодействие между процессами: обмен данными, взаимная синхронизация.

 защита самой системы, а также пользовательских данных и программ от злонамеренных действий пользователей или приложений.

 разграничение прав доступа и многопользовательский режим работы (аутентификация, авторизация).

**Основные понятия, связанные с ОС**

• Системные вызовы

• Прерывания

• Исключительные ситуации

• Файлы

• Процессы, нити

В любой операционной системе поддерживается некоторый механизм, который позволяет пользовательским программам обращаться за услугами ядра ОС. Эти средства общения с ядром часто назывались экстракодами или системными макрокомандами. В ОС UNIX такие средства называются системными вызовами.

**Системные вызовы** (system calls) интерфейс между операционной системой и пользовательской программой. Они создают, удаляют и используют различные объекты, главные из которых процессы и файлы. Пользовательская программа запрашивает сервис у операционной системы, осуществляя системный вызов. Имеются библиотеки процедур, которые загружают машинные регистры определенными параметрами и осуществляют прерывание процессора, после чего управление передается обработчику данного вызова, входящему в ядро операционной системы. Цель таких библиотек сделать системный вызов похожим на обычный вызов подпрограммы.

Основное отличие состоит в том, что при системном вызове задача переходит в привилегированный режим или режим ядра (kernel mode). Поэтому системные вызовы иногда еще называют программными прерываниями в отличие от аппаратных прерываний, которые чаще называют просто прерываниями.

В этом режиме работает код ядра операционной системы, причем он исполняется в адресном пространстве и в контексте вызвавшей его задачи. Таким образом, ядро операционной системы имеет полный доступ к памяти пользовательской программы, и при системном вызове достаточно передать адреса одной или нескольких областей памяти с параметрами вызова и адреса одной или нескольких областей памяти для результатов вызова.

В большинстве операционных систем системный вызов осуществляется командой программного прерывания (INT). Таким образом, программное прерывание это синхронное событие.

**Прерывания**

Прерывание (hardware interrupt) событие, генерируемое внешним (по отношению к процессору) устройством. Посредством аппаратных прерываний аппаратура либо информирует центральный процессор о том, что возникло какое-либо событие, требующее немедленной реакции (например, пользователь нажал клавишу), либо сообщает о завершении асинхронной операции ввода-вывода (например, закончено чтение данных с диска в основную память).

Важный тип аппаратных прерываний - прерывания таймера, которые генерируются периодически через фиксированный промежуток времени. Прерывания таймера используются операционной системой при планировании процессов. Каждый тип аппаратных прерываний имеет собственный номер, однозначно определяющий источник прерывания.

Аппаратное прерывание — это асинхронное событие, то есть оно возникает вне зависимости от того, какой код исполняется процессором в данный момент. Обработка аппаратного прерывания не должна учитывать, какой процесс является текущим.

**Исключительные ситуации**

Исключительная ситуация (exception) событие, возникающее в результате попытки выполнения программой недопустимой команды, доступа к ресурсу при отсутствии достаточных привилегий или обращения к отсутствующей странице памяти.

Исключительные ситуации так же, как и системные вызовы, являются синхронными событиями, возникающими в контексте текущей задачи. Исключительные ситуации можно разделить на исправимые и неисправимые. К исправимым относятся такие исключительные ситуации, как отсутствие нужной информации в оперативной памяти. После устранения причины исправимой исключительной ситуации программа может продолжить выполнение.

Возникновение в процессе работы операционной системы исправимых исключительных ситуаций является нормальным явлением. Неисправимые исключительные ситуации обычно возникают в результате ошибок в программах. Обычно операционная система реагирует на такие ситуации завершением программы, вызвавшей исключительную ситуацию.

**Файлы**

Файлы предназначены для хранения информации на внешних носителях, то есть, принято, что информация, лежащая, например, на диске, должна находиться внутри файла. Обычно под файлом понимают часть пространства на носителе информации, имеющую имя.

Главная задача файловой системы (file system) скрыть особенности ввода-вывода и дать программисту простую абстрактную модель файлов, независимых от устройств. Для чтения, создания, удаления, записи, открытия и закрытия файлов также имеется обширная категория системных вызовов (create, delete, open, close, read, write ).

Пользователям хорошо знакомы такие понятия, связанные с организацией файловой системы, как каталог, текущий каталог, корневой каталог, путь, для манипулирования которыми в операционной системе имеются системные вызовы.

**Процессы и нити**

Проце́сс — программа, которая выполняется в текущий момент. Стандарт ISO 9000:2000 определяет процесс как совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие.

Компьютерная программа сама по себе — лишь пассивная последовательность инструкций. В то время как процесс — непосредственное выполнение этих инструкций.

Также, процессом называют выполняющуюся программу и все её элементы: адресное пространство, глобальные переменные, регистры, стек, открытые файлы и так далее.

Пото́к выполне́ния (тред; от англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

1. Основные подходы к организации архитектуры ОС. Классификация ОС. Роли пользователей ОС.

**Состав операционной системы:**

В общем случае в состав **ОС** входят следующие модули:

* Программный модуль, управляющий файловой системой.
* Командный процессор, выполняющий команды пользователя.
* Драйверы устройств.
* Программные модули, обеспечивающие графический пользовательский интерфейс.
* Сервисные программы.
* Справочная система.

**Драйвер устройства** (device driver) – специальная программа, обеспечивающая управление работой устройств и согласование информационного обмена с другими устройствами.  
  
**Командный процессор** (command processor) – специальная программа, которая запрашивает у пользователя команды и выполняет их (интерпретатор программ).

Интерпретатор  команд отвечает за загрузку приложений и управление информационным потоком между приложениями.

Для упрощения работы пользователя в состав современных ОС входят программные модули, обеспечивающие графический пользовательский интерфейс.  
Процесс работы компьютера в определенном смысле сводится к обмену файлами между устройствами. В ОС имеется программный модуль, управляющий файловой системой.

**Сервисные программы** позволяют обслуживать диски (проверять, сжимать, дефрагментировать и др.), выполнять операции с файлами (копирование, переименование и др.), работать в компьютерных сетях.

Для удобства пользователя в состав ОС входит **справочная система**, позволяющая оперативно получить необходимую информацию о функционировании как ОС в целом, так и о работе ее отдельных модулей.

*Состав модулей ОС, а также их количество зависит от семейства и вида ОС. Так, например, в ОС MS DOS отсутствует модуль, обеспечивающий графический пользовательский интерфейс.*

Наиболее общим подходом к структуризации **операционной системы** является разделение всех ее модулей на две группы:

1. **Ядро** – это модули, выполняющие основные функции ОС.
2. **Вспомогательные модули**, выполняющие вспомогательные функции ОС. Одним из определяющих свойств ядра является работа в **привилегированном** **режиме**.

Модули ядра выполняют следующие базовые функции ОС: Управление процессами, Управление системой прерываний, Управление памятью, управление устройствами ввода-вывода, Функции, решающие внутрисистемные задачи организации вычислительного процесса: переключение контекстов, загрузка/вы­грузка страниц, обработка прерываний. Эти функции недоступны для приложе­ний. Функции, служащие для поддержки приложений, создавая для них так называемую прикладную программную среду.

Приложения могут обращаться к ядру с запросами – **системными вызовами** – для выполнения тех или иных действий: для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т.д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования – API (**Application programming interface)**.  
**Вспомогательные модули** ОС выполняют вспомогательные функции ОС (полезные, но менее обязательные чем функции ядра).

**Примеры вспомогательных модулей:**

* Программа архивирования данных.
* Программа дефрагментации диска.
* Текстовый редактор.

Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур. Вспомогательные модули ОС подразделяются на следующие группы:

**утилиты** – программы, решающие задачи управления и сопровождения компьютерной системы: обслуживание дисков и файлов.

**системные обрабатывающие программы** – текстовые или графические редакторы, компиляторы, компоновщики, отладчики.

программы предоставления пользователю дополнительных услуг пользовательского интерфейса (калькулятор, игры).

библиотеки процедур различного назначения, упрощающие разработку при­ложений (библиотека математических функций, функций ввода-вывода).

Как и обычные приложения, для выполнения своих задач утилиты, обрабатывающие программы и библиотеки ОС, обращаются к функциям ядра посредством системных вызовов.  
Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями операционной системы, поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в оперативной памяти, то есть являются резидентными.

Вспомогательные модули обычно загружаются в оперативную память только на время выполнения своих функций, то есть являются транзитными. Такая организация ОС экономит оперативную память компьютера.

Примечание

*Разделение операционной системы на ядро и вспомогательные модули обеспечивает легкую расширяемость ОС. Чтобы добавить новую высокоуровневую функцию, достаточно разработать новое приложение, и при этом не требуется модифицировать основные функции, образующие ядро системы.*

Объектами ядра ОС являются:

* Процессы
* Файлы.
* События.
* Потоки
* Семафоры – объекты, позволяющие войти в заданный участок кода не более чем n потокам.
* Мьютексы – одноместные семафоры, служащие в программировании для синхронизации одновременно выполняющихся потоков.
* Файлы, проецируемые в память.

**Классификация ОС**

Существует несколько схем классификации операционных систем. Возможные критерии:

• Реализация многозадачности

• Поддержка многопользовательского режима.

• Многопроцессорная обработка

• Реализация режима реального времени

***По числу одновременно выполняемых задач*** операционные системы могут быть разделены на два класса:

• многозадачные (Unix, OS/2, Windows).

• однозадачные (например, MS-DOS) и

Многозадачная ОС, решая проблемы распределения ресурсов и конкуренции, полностью реализует мультипрограммный режим.

***Поддержка многопользовательского режима***

По числу одновременно работающих пользователей ОС можно разделить на:

• однопользовательские (MS-DOS, Windows 3.x);

• многопользовательские (Windows NT, Unix).

Наиболее существенно отличие заключается в наличии у многопользовательских систем механизмов защиты персональных данных каждого пользователя.

***Многопроцессорная обработка***

Многопроцессорные системы состоят из двух или более центральных процессоров, осуществляющих параллельное выполнение команд. Поддержка мультипроцессирования является важным свойством ОС и приводит к усложнению всех алгоритмов управления ресурсами. Многопроцессорная обработка реализована в таких ОС, как Linux, Solaris, Windows NT и в ряде других.

Многопроцессорные ОС разделяют на симметричные и асимметричные. В симметричных ОС на каждом процессоре функционирует одно и то же ядро и задача может быть выполнена на любом процессоре, то есть обработка полностью децентрализована.

В асимметричных ОС процессоры неравноправны. Обычно существует главный процессор (master) и подчиненные (slave), загрузку и характер работы которых определяет главный процессор.

***Системы реального времени***

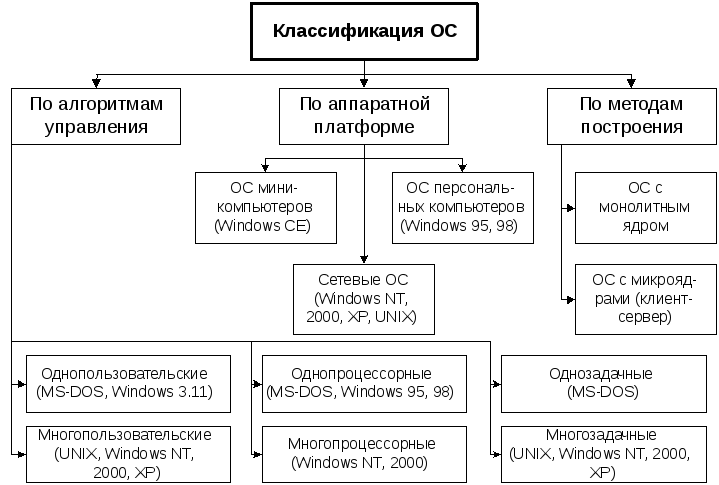
В разряд многозадачных ОС, наряду с пакетными системами и системами разделения времени, включаются также системы реального времени, не упоминавшиеся до сих пор.

Они используются для управления различными техническими объектами или технологическими процессами. Такие системы характеризуются предельно допустимым временем реакции на внешнее событие, в течение которого должна быть выполнена программа, управляющая объектом. Система должна обрабатывать поступающие данные быстрее, чем те могут поступать, причем от нескольких источников одновременно.

Столь жесткие ограничения сказываются на архитектуре систем реального времени, например, в них может отсутствовать виртуальная память, поддержка которой дает непредсказуемые задержки в выполнении программ.

Приведенная классификация ОС не является исчерпывающей.

Можно рассмотреть некоторые другие классификации, например, см. рис. 9.





1. Основные понятия ОС Windows (процессы, потоки, задания, виртуальная память).

**Программа** представляет собой статический набор команд, а **процесс** — это контейнер для набора ресурсов, используемых при выполнении экземпляра программы. На самом высоком уровне абстракции процесс в Windows включает следующее:

• **закрытое виртуальное адресное пространство** — диапазон адресов виртуальной памяти, которым может пользоваться процесс;

• **исполняемую программу** — исполняемый код и данные, проецируемые на виртуальное адресное пространство процесса;

• **список открытых описателей (handles) различных системных ресурсов** — семафоров, коммуникационных портов, файлов и других объектов, доступных всем потокам в данном процессе;

• **контекст защиты** (security context), называемый маркером доступа (асcess token) и идентифицирующий пользователя, группы безопасности и привилегии, сопоставленные с процессом;

• **уникальный идентификатор процесса** (во внутрисистемной терминологии называемый идентификатором клиента); минимум один поток.

**Поток (thread)** — некая сущность внутри процесса, получающая время CPU для выполнения. Без потока программа процесса не может выполняться. Поток включает следующие наиболее важные элементы:

* содержимое набора регистров CPU, отражающих состояние CPU;
* два стека, один из которых используется потоком при выполнении в режиме ядра, а другой — в пользовательском режиме;
* закрытую область памяти, называемую локальной памятью потока (thread-local storage, TLS) и используемую подсистемами, библиотеками исполняющих систем (run-time libraries) и DLL;
* уникальный идентификатор потока (во внутрисистемной терминологии также называемый идентификатором клиента: идентификаторы процессов и потоков генерируются из одного пространства имен и никогда не перекрываются);
* иногда потоки обладают своим контекстом защиты, который обычно используется многопоточными серверными приложениями, подменяющими контекст защиты обслуживаемых клиентов.

В Windows реализована система виртуальной памяти, основанная на **плоском (линейном) адресном пространстве**. Она создает каждому процессу иллюзию того, что у него есть собственное большое и закрытое адресное пространство. Виртуальная память дает логическое представление, не обязательно соответствующее структуре физической памяти. В период выполнения диспетчер памяти, используя аппаратную поддержку, транслирует, или **проецирует (maps)**, виртуальные адреса на физические, по которым реально хранятся данные. Управляя проецированием и защитой страниц памяти, ОС гарантирует, что ни один процесс не помешает другому и не сможет повредить данные самой ОС. На Рис. 1 показано, как три смежные страницы виртуальной памяти проецируются на три разрозненные страницы физической памяти.

Физическая память

Виртуальная память

1. Общая организация ОС Windows: режим ядра и пользовательский режим, объекты и описатели, безопасность, реестр.

**Режим ядра и пользовательский режим**

Для предотвращения доступа приложений к критически важным данным ОС и устранения риска их модификации Windows использует два режима доступа к CPU (даже если он поддерживает более двух режимов): пользовательский (user mode), ядра (kernel mode).

Код приложений работает в **пользовательском режиме**, тогда как код ОС (например, системные сервисы и драйверы устройств) — **в режиме ядра**. В режиме ядра предоставляется доступ ко всей системной памяти и разрешается выполнять любые машинные команды CPU. Предоставляя ОС более высокий уровень привилегий, чем прикладным программам, CPU позволяет разработчикам ОС реализовать такие архитектуры, которые не дали возможности сбойным приложениям нарушать стабильность работы всей системы.

Хотя каждый Windows-процесс имеет свою (закрытую) память, **код ОС и драйверы устройств, работающие в режиме ядра, делят единое виртуальное адресное пространство**. Каждая страница в виртуальной памяти помечается тэгом, определяющим, в каком режиме должен работать CPU для чтения и/или записи данной страницы. Страницы в системном пространстве доступны лишь в режиме ядра, а все страницы в пользовательском адресном пространстве — в пользовательском режиме. Страницы только для чтения (например, содержащие лишь исполняемый код) ни в каком режиме для записи недоступны.

**Объекты и описатели**

В ОС Windows **объект** — это единственный экземпляр периода выполнения (run-time instance) статически определенного типа объекта. Тип объекта состоит из общесистемного типа данных, функций, оперирующих экземплярами этого типа данных, и набора атрибутов.

**Атрибут объекта (object attribute)** — это поле данных в объекте, частично определяющее состояние этого объекта. Например, объект типа «процесс», имеет атрибуты, в число которых входят идентификатор процесса, базовый приоритет и указатель на объект маркера доступа. Методы объекта (средства для манипулирования объектами) обычно считывают или изменяют какие-либо атрибуты.

Самое главное различие между объектом и обычной структурой данных заключается в том, что внутренняя структура объекта скрыта. Чтобы получить данные из объекта или записать в него какую-то информацию, следует вызвать его сервис. Прямое чтение или изменение данных внутри объекта невозможно. Тем самым реализация объекта отделяется от кода, который просто использует его, а это позволяет менять реализацию объекта, не модифицируя остальной код.

Объекты очень удобны для поддержки четырех важных функций ОС:

1. присвоения понятных имен системным ресурсам;
2. разделения ресурсов и данных между процессами;
3. защиты ресурсов от несанкционированного доступа;
4. учета ссылок (благодаря этому система узнает, когда объект больше не используется, и автоматически уничтожает его).

**Безопасность**

Windows с самого начала разрабатывалась как защищенная система, удовлетворяющая требованиям различных правительственных и промышленных стандартов безопасности, например **спецификации Common Criteria for Information Technology Security Evaluation (CCITSE)**. Подтверждение правительством рейтинга безопасности ОС позволяет ей конкурировать в сферах, требующих повышенной защиты. Разумеется, многим из этих требований должна удовлетворять любая многопользовательская система.

Базовые возможности защиты в Windows таковы:

• избирательная защита любых разделяемых системных объектов (файлов, каталогов, процессов, потоков и т.д.),

• аудит безопасности (для учета пользователей и инициируемых ими операций),

• аутентификация паролей при входе и предотвращение доступа одного из пользователей к неинициализированным ресурсам, освобожденным другим пользователем.

Windows поддерживает два вида контроля доступа к объектам.

**Управление избирательным доступом** (discretionary access control) — является механизмом, который как раз и связывается большинством пользователей с защитой. Это метод, при котором владельцы объектов (например, файлов или принтеров) разрешают или запрещают доступ к ним для других пользователей. При входе пользователь получает набор удостоверений защиты (security credentials), или контекст защиты (security context). Когда он пытается обратиться к объекту, его контекст защиты сверяется со списком управления доступом (access control list, ACL) для данного объекта, чтобы определить, имеет ли он разрешение на выполнение запрошенной операции.

**Управление привилегированным доступом** (privileged access control) — необходим в тех случаях, когда управления избирательным доступом недостаточно. Данный метод гарантирует, что пользователь сможет обратиться к защищенным объектам, даже если их владелец недоступен. Например, если какой-то сотрудник увольняется из компании, администратору нужно получить доступ к файлам, которые могли быть доступны только бывшему сотруднику. В таких случаях Windows позволяет администратору стать владельцем этих файлов и при необходимости управлять правами доступа к ним.

**Реестр**

**Реестр** - системная база данных с информацией, необходимой для загрузки и конфигурирования системы; в ней содержатся общесистемные параметры, контролирующие работу Windows, база данных защиты и конфигурационные настройки, индивидуальные для каждого пользователя.

**Кроме того, реестр** — это окно, через которое можно заглянуть в переменные системные данные, чтобы, например, выяснить текущее состояние аппаратной части системы (какие драйверы устройств загружены, какие ресурсы они используют и т.д.) или значения счетчиков производительности Windows. Счетчики производительности, которые на самом деле в реестре не хранятся, доступны через функции реестра.

1. Обзор архитектуры ОС Windows: компоненты архитектуры ОС и основные особенности. Подсистемы.

Теперь обратимся к ключевым компонентам системы, составляющим ее архитектуру. Упрощенная версия этой архитектуры показана на Рис. 4 (упрощенная схема не отражает всех деталей архитектуры).



Рис. 4. Упрощенная архитектура Windows

Прямоугольники над разделительной линией соответствуют процессам пользовательского режима, а компоненты под ней — сервисам режима ядра. Известно, что потоки пользовательского режима выполняются в защищенных адресных пространствах процессов (хотя при выполнении в режиме ядра они получают доступ к системному пространству). Таким образом, процессы поддержки системы, сервисов, приложений и подсистем окружения имеют свое адресное пространство.

Существует четыре типа пользовательских процессов:

* **фиксированные процессы поддержки системы** (system support processes) — например, процесс обработки входа в систему и диспетчер сеансов, не являющиеся сервисами Windows (т. е. не запускаемые диспетчером управления сервисами);
* **процессы сервисов** (service processes) — носители Windows-сервисов вроде Task Scheduler и Spooler. Многие серверные приложения Windows, например Microsoft SQL Server и Microsoft Exchange Server, тоже включают компоненты, выполняемые как сервисы;
* **пользовательские приложения** (user applications) — бывают шести типов:
* 32-разрядной Windows,
* 64-разрядной Windows,
* 16-разрядной Windows 3.1,
* 16-разрядной MS-DOS,
* 32-разрядной POSIX,
* 32-разрядной OS/2;
* **подсистемы окружения** (environment subsystems) — реализованы как часть поддержки среды ОС, предоставляемой пользователям и программистам. Изначально Windows NT поставлялась с тремя подсистемами окружения: Windows, POSIX и OS/2. Последняя была изъята в Windows 2000. Что касается Windows XP, то в ней исходно поставляется только подсистема Windows, улучшенная подсистема POSIX доступна как часть бесплатного продукта Services for UNIX.

В Windows пользовательские приложения не могут вызывать родные сервисы ОС напрямую, вместо этого они работают с одной или несколькими DLL подсистем. Их назначение заключается в трансляции документированных функций в соответствующие внутренние (и обычно недокументированные) вызовы системных сервисов Windows.

Windows включает следующие компоненты режима ядра:

* **Исполнительная система** (executive) Windows, содержащая базовые сервисы ОС, которые обеспечивают управление памятью, процессами и потоками, защиту, ввод-вывод и взаимодействие между процессами.
* **Ядро** (kernel) Windows, содержащее низкоуровневые функции ОС, которые поддерживают, например, планирование потоков, диспетчеризацию прерываний и исключений, а также синхронизацию при использовании нескольких CPU. Оно также предоставляет набор процедур и базовых объектов, применяемых исполнительной системой для реализации структур более высокого уровня.
* **Драйверы устройств** (device drivers), в состав которых входят драйверы аппаратных устройств, транслирующие пользовательские вызовы функции ввода-вывода в запросы, специфичные для конкретного устройства, а также сетевые драйверы и драйверы файловых систем.
* **Уровень абстрагирования от оборудования** (hardware abstraction layer, HAL), изолирующий ядро, драйверы и исполнительную систему Windows от специфики оборудования на данной аппаратной платформе (например, от различий между материнскими платами).
* **Подсистема поддержки окон и графики** (windowing and graphics system), реализующая функции графического пользовательского интерфейса (GUI), более известные как Windows-функции модулей USER и GDI. Эти функции обеспечивают поддержку окон, элементов управления пользовательского интерфейса и отрисовку графики.

1. ОС Windows: процессы (понятие процесса, структуры данных, переменные ядра, сопутствующие функции).

**Процессы**

**Структуры данных**

Каждый процесс в Windows **представлен блоком процесса** (EPROCESS), создаваемым исполнительной системой. Кроме многочисленных атрибутов, относящихся к процессу, в блоке EPROCESS содержатся указатели на некоторые структуры данных. Так, у каждого процесса есть один или более потоков, представляемых **блоками потоков** (ЕТНREAD) исполнительной системы. Блок EPROCESS и связанные с ним структуры данных, за исключением **блока переменных окружения процесса** (process environment block, РЕВ) — существуют в системном пространстве. РЕВ находится в адресном пространстве процесса, так как содержит данные, модифицируемые кодом пользовательского режима.

Для каждого процесса, выполняющего Windows-программу, процесс подсистемы Windows (Csrss) поддерживает в дополнение к блоку EPROCESS параллельную структуру данных. Кроме того, часть подсистемы Windows, paботающая в режиме ядра (Win32k.sys), поддерживает структуру данных для каждого процесса, которая создается при первом вызове потоком любой функции USER или GDI, реализованной в режиме ядра.

Блок

переменных

окружения

процесса

Блок

переменных

окружения

потока

Блок потока

Блок

процесса

Таблица описателей

Блок процесса

Windows

Адресное пространство процесса

Системное адресное пространство

Сначала рассмотрим блок процесса. Ключевые поля EPROCESS показаны на Рис. 2.

Блок процесса ядра

(

или

PCB

)

Идентификатор родительского процесса

Идентификатор процесса

Статус завершения

Блок квот

Блок следующего процесса

Время создания и завершения

Информация

связанная с управлением памятью

Порт отладчика

Порт исключений

Имя образа исполняемого файла

Блок переменных окружения процесса

Карта устройств

Базовый адрес образа

Класс приоритета образа

Блок процесса

Windows

Объект

«

задание

»

Таблица описателей

Первичный маркер доступа

PsActiveProcessHead

EPROCESS

Рис. 2. Блок процесса (EPROCESS), создаваемый исполнительной системой

Таблица 1. Содержимое блока EPROCESS

|  |  |
| --- | --- |
| Элемент | Описание |
| Блок процесса ядра (KPROCESS) | Общий заголовок объекта диспетчера, указатель на каталог страниц процесса, список блоков потоков ядра (KTHREAD), принадлежащих процессу, базовый приоритет по умолчанию выделяемый квант времени CPU, маска привязки к CPU, а также суммарное время работы потоков в режиме ядра и пользовательском режиме |
| Идентификация процесса | Уникальный идентификатор процесса, идентификатор процесса-создателя, имя выполняемого образа, имя WindowStation-объекта для данного процесса |
| Блок квот | Лимиты на пулы подкачиваемой и неподкачиваемой памяти, а также на использование страничного файла плюс текущее и пиковое использование пулов подкачиваемой и неподкачиваемой памяти процесса. Некоторые процессы могут разделять эту структуру: все системные процессы ссылаются на один общесистемный блок квот по умолчанию, а все процессы интерактивного сеанса используют один блок квот, определяемый Winlogon |
| Дескрипторы виртуальных адресов (virtual address  descriptors) | Наборы структур данных, описывающих статус частей адресного пространства, которые существуют в процессе |
| Информация о  рабочем наборе | Указатель на список рабочего набора (структура MNTWSL); текущий, пиковый, минимальный и максимальный размеры рабочего набора; время последнего усечения (last trim time); счетчик числа ошибок страниц; приоритет памяти; некоторые специфические флаги; хронология ошибок страниц |
| Информация о  виртуальной памяти | Текущий и пиковый размер виртуальной памяти, использование страничного файла, элемент аппаратной таблицы страниц, указывающий на каталог страниц процесса |
| LPC-порт исключений | Канал межпроцессной связи, по которому диспетчер процессов посылает сообщение, если один из потоков этого процесса вызывает исключение |
| Отладочный LPC-порт | Канал межпроцессной связи, по которому диспетчер процессов посылает сообщение, если один из потоков этого процесса генерирует событие отладки |
| Маркер доступа  (ACCESS TOKEN) | Объект исполнительной системы, описывающий профиль защиты данного процесса |
| Таблица описателей | Адрес таблицы описателей, принадлежащей процессу |
| Карта устройств | Адрес каталога объектов для разрешения ссылок на устройства по именам (поддерживается несколько пользователей) |
| Блок переменный окружения процесса  (РЕВ) | Информация об образе исполняемого файла (базовый адрес, номера версий, список модулей), информация о куче процесса и TLS-пaмяти (указатели на кучи процесса начинаются с первого байта после РЕВ) |
| Блок процесса подсистемы Windows (W32PROCESS) | Дополнительная информация о процессе, необходимая той части подсистемы Windows, которая работает ь режиме ядра |

Переменные ядра

В Таблица 2 перечислено несколько важнейших глобальных переменных ядра, связанных с процессами. На эти переменные будем ссылаться по ходу изложения материала, в частности при описании этапов создания процесса.

Таблица 2. Переменные ядра, связанные с процессами

|  |  |  |
| --- | --- | --- |
| Переменная | Тип | Описание |
| PsActiveProcessHead | Заголовок очереди | Заголовок списка блоков процесса |
| PsIdleProcess | EPROCESS | Блок процесса Idle |
| PsInitialSystemProces s | Указатель на  EPROCESS | Указатель на блок начального системного процесса (с идентификатором 2), который содержит системные потоки |
| PspCreateProcessNotif yRoutine | Массив указателей | Указатели на процедуры (максимум 8), вызываемые при создании и удалении процесса |
| PspCreateProcess NotifyRoutineCount | DWORD | Счетчик зарегистрированных процедур уведомления о создании процесса |
| PspLoadlmageNotifyRo utine | Массив указателей | Указатели на процедуры, вызываемые при загрузке образа исполняемого файла |
| PspLoadlmageNotifyRo utineCount | DWORD | Счетчик зарегистрированных процедур уведомления о загрузке образа |
| PspCidTable | Указатель на  HANDLETABLE | Таблица описателей для клиентских идентификаторов процесса и потока |

**Сопутствующие функции**

В Таблица 4 приведена информация по некоторым Windows-функциям, связанным с процессами [4].

Таблица 4. Функции, связанные с процессами

|  |  |
| --- | --- |
| Функция | Описание |
| CreateProcess | Создает новый процесс и поток с использованием идентификации защиты вызывающего процесса |
| CreateProcessAsUser | Создает новый процесс и поток с указанным альтернативным маркером защиты |
| CreateProcessWithLog onW | Создает новый процесс и поток для выполнения под учетной записью, соответствующей указанным имени и паролю пользователя |
| CreateProccssWithTok enW | Создает новый процесс и поток с указанным альтернативным маркером защиты и поддерживает дополнительные возможности, например разрешает загрузку профиля пользователя |
| OpenProcess | Возвращает описатель указанного объекта «процесс» |
| ExitProcess | Завершает процесс с уведомлением всех подключенных DLL |
| TerminateProcess | Завершает процесс без уведомления подключенных DLL |
| FlushInstructionCache | Опустошает кэш команд указанного процесса |
| GetProcessTimes | Получает временные параметры процесса, описывающие, сколько времени процесс провел в режиме ядра и в пользовательском режиме |
| GetExitCodeProcess | Возвращает код завершения процесса, указывающий, как и почему завершился этот процесс |
| GetCommandLine | Возвращает указатель на командную строку, переданную текущему процессу |
| GetCurrentProcess | Возвращает псевдоописатель текущего процесса |
| GetCurrentProcessId | Возвращает идентификатор текущего процесса |
| GetProcessVersion | Возвращает старший и младший номера версии Windows, необходимой для запуска указанного процесса |
| GetStartupInfo | Возвращает содержимое структуры STARTUPINFO, заданное при вызове CreateProcess |
| GetEnvironmentString  s | Возвращает адрес блока переменных окружения |
| GetEnvironmentVariab  le | Возвращает значение указанной переменной окружения |
| GetProcessShutdownP  arameters и  SetProcessShutdownP  arameters | Определяет приоритет завершения и число попыток для текущего процесса |
| GetGuiResources | Возвращает число открытых описателей USER и GDI |

1. Создание процессов в ОС Windows.

**Что делает функция CreateProcess**

Как процессы появляются на свет и как они завершаются, выполнив задачи, для которых они предназначались?

Создание Windows-процесса осуществляется вызовом одной из таких функций, как

CreateProcess, CreateProcessAsUser, CreateProcessWtihTokenW или

CreateProcessWithLogonW, и проходит в несколько этапов с участием трех компонентов

ОС: Kernel32.dll (библиотеки клиентской части Windows), исполнительной системы и процесса подсистемы окружения Windows (Csrss). Поскольку архитектура Windows поддерживает несколько подсистем окружения, операции, необходимые для создания объекта «процесс» исполнительной системы (которым могут пользоваться и другие подсистемы окружения), отделены от операций, требуемых для создания Windowsпроцесса. Поэтому часть действий Windows-функции CreateProcess специфична для семантики, привносимой подсистемой Windows.

В приведенном ниже списке перечислены основные этапы создания процесса Windowsфункцией CreateProcess.

**ПРИМЕЧАНИЕ.** Многие этапы работы CreateProcess связаны с подготовкой виртуального адресного пространства процесса и поэтому требуют понимания массы структур и терминов, связанных с управлением памятью и описываемых в лекции далее.

1. Открывается файл образа (ЕХЕ), который будет выполняться в процессе.
2. Создается объект «процесс» исполнительной системы.
3. Создается первичный поток (стек, контекст и объект «поток» исполнительной системы).
4. Подсистема Windows уведомляется о создании нового процесса и потока.
5. Начинается выполнение первичного потока (если не указан флаг CREAТЕ\_SUSPENDED).
6. В контексте нового процесса и потока инициализируется адресное пространство (например, загружаются требуемые DLL) и начинается выполнение программы.

Общая схема создания процесса в Windows показана на Рис. 5. Прежде чем открыть исполняемый образ для выполнения, CreateProcess делает следующее.

При вызове CreateProcess класс приоритета указывается в параметре СrеаtionFlags, и, вызывая CreateProcess, вы можете задать сразу несколько классов приоритета. Windows выбирает самый низкий из них.

Когда для нового процесса не указывается класс приоритета, по умолчанию принимается Normal, если только класс приоритета процесса-создателя не равен Idle или Below Normal. В последнем случае новый процесс получает тот же класс приоритета, что и у родительского процесса.

Если для нового процесса указан класс приоритета Real-time, а создатель не имеет привилегии Increase Scheduling Priority, устанавливается класс приоритета High. Иначе говоря, функция CreateProcess завершается успешно, даже если у того, кто ее вызвал, недостаточно привилегий для создания процессов с классом приоритета Realtime, просто класс приоритета нового процесса будет ниже Real-time.

Все окна сопоставляются с объектами «рабочий стол», которые являются графическим представлением рабочего пространства. Если при вызове CreateProcess не указан конкретный объект «рабочий стол», новый процесс сопоставляется с текущим объектом «рабочий стол» процесса-создателя.

Начинается

выполнения

первичного потока

Уведомляется

подсистема

Windows

Создается объект

«

поток

»

Создается объект

«

процесс

»

Открывается

EXE

и создается

объект

«

раздел

»

Создается новый

процесс и поток

**Подсистема**

**Windows**

**Создание процесса**

**Новый процесс**

Завершается

инициализация

процесса

/

образа

Выполнение

начинается с

точки входа в

образ

Управление

возвращается

вызвавшей функции

Этап

1

Этап

2

Этап

3

Этап

4

Этап

5

Этап

6

1. Управление процессами в ОС Windows.

Процесс (process) представляет собой объект, обладающий собственным независимым виртуальным адресным пространством, в котором могут размещаться код и данные, защищенные от других процессов. В свою очередь, внутри каждого процесса могут независимо выполняться одна или несколько потоков (threads). Поток, выполняющийся внутри процесса, может сама создавать новые потоки и новые независимые процессы, а также управлять взаимодействием объектов между собой и их синхронизацией.

Создавая процессы и управляя ими, приложения могут организовывать параллельное выполнение нескольких задач, обеспечивающих обработку файлов, проведение вычислений или связь с другими системами в сети. Допускается даже использование нескольких процессоров с целью ускорения обработки данных.

1. ОС Windows: потоки (внутреннее устройство потоков, структуры данных, переменные ядра, сопутствующие функции, создание потоков).

Внутреннее устройство потоков

Там, где явно не сказано обратное, считайте, что весь материал этого раздела в равной мере относится как к обычным потокам пользовательского режима, так и к системным потокам режима ядра.

Структуры данных

На уровне ОС поток представляется блоком потока, принадлежащим исполнительной системе (ETHREAD). Структура этого блока показана на Рис. 7

Блок ETHREAD и все структуры данных, на которые он ссылается, существуют в системном адресном пространстве, кроме **блока переменных окружения потока** (thread environment block, TEB) — он размещается в адресном пространстве процесса.

Помимо этого, процесс подсистемы Windows (Csrss) поддерживает параллельную структуру для каждого потока, созданного в Windows-процессе. Часть подсистемы Windows, работающая в режиме ядра (Win32k.sys), также поддерживает для каждого потока, вызывавшего USER- или GDI-функцию, структуру W32THREАD, на которую указывает блок ETHREAD.

Время создание и завершение

Идентификатор процесса

Стартовый адрес потока

Информация, используемая при

олицетворении

Данные сообщений

LPC

Данные таймера

KTHREAD

Незавершенные вопросы

на ввод

-

вывод

Маркер доступа

EPROCESS

TEB

Рис. 7. Блок потока (ETHREAD) исполнительной системы

Поля блока потока ETHREAD, показанные на Рис. 7, в большинстве своем не требуют дополнительных пояснений. Первое поле - блок потока ядра (КТНREAD). За ним следуют идентификационные данные потока и процесса (включая указатель на процесс — владелец данного потока, что обеспечивает доступ к информации о его окружении), затем информация о защите в виде указателя на маркер доступа и сведения, необходимые для олицетворения (подмены одного процесса другим), а также поля, связанные с сообщениями LPC и незавершенными запросами на ввод-вывод.

Таблица 6. Ключевые поля блока потока ETHREAD

|  |  |
| --- | --- |
| Элемент | Описание |
| KTHREAD | Таблица 7 |
| Временные показатели потока | Время создания и завершения потока |
| Идентификационные данные процесса | Идентификатор процесса и указатель на блок EPROCESS процесса, которому принадлежит данный поток |
| Стартовый адрес | Адрес стартовой процедуры потока |
| Информация об  олицетворении | Маркер доступа и уровень олицетворения (если поток олицетворяет, или подменяет, клиент) |
| Информация LPC | Идентификатор и адрес сообщения, ожидаемого потоком |
| Информация, связанная с вводом-выводом | Список необработанных пакетов запросов на ввод-вывод (I/O request packets, IRP) |

Рассмотрим две ключевые структуры потока, упомянутым выше, — к блокам KTHREAD и ТЕВ. Первый содержит информацию, нужную ядру Windows для планирования потоков и их синхронизации с другими потоками. Схема блока KTHREAD показана на Рис. 8.

Заголовок диспетчера

Суммарное время в пользовательском

режиме

Суммарное время в режиме ядра

Информация

,

связанная с

планированием потока

Фрейм ловушки

Список незавершенных

APC

Блок таймера и блок ожидания

Список объектов

,

ожидаемых потоком

Информация

,

связанная со стоком ядра

Таблица системных сервисов

Массив

TLS

TEB

Информация

,

связанная

синхронизаций

Рис. 8. Схема блока потока

Ключевые поля блока KTHREAD кратко рассмотрены в Рис. 7.

Таблица 7. Ключевые поля блока KTHREAD

|  |  |
| --- | --- |
| Элемент | Описание |
| Заголовок диспетчера | Поскольку поток — это объект, на котором можно ждать, он запускается со стандартным заголовком объекта диспетчера ядра |
| Время выполнения | Суммарное время выполнения (в режиме ядра и в пользовательском режиме) |
| Указатель на информацию стека ядра | Базовый и верхний адреса стека ядра |
| Указатель на таблицу системных сервисов | В начале выполнения каждого потока указатель в данном поле ссылается на главную таблицу системных сервисов  KeServiceDescriptorTable, когда поток впервые вызывает GUIсервис Windows, текущей таблицей системных сервисов становится та, которая включает сервисы GDT и USER в Win32k.sys |
| Информация, связанная с планированием | Базовый и текущий приоритеты, значение кванта, маска привязки к CPUs, идеальный CPU, статус планирования, счетчики числа  замораживаний (freeze count) и приостановок (suspend count) |
| Блоки ожидания | В блоке потока содержится четыре встроенных блока ожидания, поэтому их не нужно создавать и инициализировать при каждом переходе потока в состояние ожидания какого-либо объекта (один блок ожидания принадлежит таймерам) |
| Информация об ожидании | Список объектов, ожидаемых потоком, причина ожидания и время перехода потока в состояние ожидания |
| Список мутантов | Список принадлежащих потоку объектов «мутант» |
| Очереди АРС | Список незавершенных АРС режима ядра и пользовательского режима, а также флаг оповещения (alertable flag) |
| Блок таймера | Встроенный блок таймера (а также соответствующий блок ожидания) |
| Список очередей | Указатель на сопоставленный с потоком объект очереди |
| Указатель на ТЕВ | Идентификатор потока, данные TEB, указатель на РЕВ, информация, связанная с GDI и OpenGL |

В отличие от других структур данных, описываемых в этом разделе, только блок ТЕВ, показанный на Рис. 9, присутствует в адресном пространстве процесса, а не системы. В ТЕВ хранится контекстная информация загрузчика образов и различных Windows DLL. Поскольку эти компоненты выполняются в пользовательском режиме, им нужны структуры данных, доступные для записи из этого режима. Вот почему ТЕВ размещается в адресном пространстве процесса, а не системы, где он был бы доступен для записи только из режима ядра. Адрес ТЕВ можно найти с помощью команды thread отладчика ядра.

Список исключений

База стека

Лимит стека

Идентификатор потока

Описатель активного

RPC

Значение

LastError

Счетчик критических секций

,

принадлежащих потоку

Идентификатор текущих

региональных стандартов в системе

Информация клиента

User

32

Данные

GDI

32

Данные

OpenGL

Массив

TLS

Блок информации о потоке

(

Т

ІВ

)

подсистемы

Информация о волокнах

РЕВ

Данные

Winsock

Рис. 9. Поля блока переменных окружения потоков

Переменные ядра

Как и в случае процессов, ряд переменных ядра Windows контролирует выполнение потоков. Список таких переменных, связанных с потоками, приводится в Таблица 8.

Таблица 8. Переменные ядра, относящиеся к потокам

|  |  |  |
| --- | --- | --- |
| Переменная | Тип | Описание |
| PspCreateThread NotifyRoutine | Массив указателей | Массив указателей на процедуры (максимум 8) вызываемых при создании и удалении потока |
| PspCreateThread | DWORD | Счетчик зарегистрированных процедур |
| NotifyRoutineCount |  | уведомления потока |
| PspCreateProcessNotifyRoutin | Массив указателей | Массив указателей на процедуры (максимум 8) вызываемых при создании  и удалении процесса |

**Рождение потока**

Жизненный цикл потока начинается при его создании программой. Запрос на его создание в конечном счете поступает исполнительной системе Windows, где диспетчер процессов выделяет память для объекта «поток» и вызывает ядро для инициализации блока потока ядра. Ниже перечислены основные этапы создания потока Windowsфункцией CreateThread (которая находится в Kernel32.dll).

1. CreateThread создает стек пользовательского режима в адресном пространстве процесса.
2. CreateThread инициализирует аппаратный контекст потока, специфичный для конкретной архитектуры CPU.
3. Для создания объекта «поток» исполнительной системы вызывается NtCreateThread. Он создается в приостановленном состоянии.
4. CreateThread уведомляет подсистему Windows о создании нового потока, и та выполняет некоторые подготовительные операции.
5. Вызвавшему коду возвращаются описатель и идентификатор потока

(сгенерированный на этапе 3).

1. Выполнение потока возобновляется, и ему может быть выделено время CPU, если только он не был создан с флагом CREATE\_SUSPENDED. Перед вызовом по пользовательскому стартовому адресу поток выполняет операции, описанные в разделе «Этап 3: создание первичного потока, его стека и контекста» ранее в этой главе.
2. Планирование потоков: уровни приоритета процессов и потоков, состояния потоков.

В Windows реализована подсистема **вытесняющего планирования** **на основе уровней приоритета**, в которой всегда выполняется поток с наибольшим приоритетом, готовый к выполнению. Однако выбор потока для выполнения может быть ограничен набором CPUs, на которых он может работать. Это явление называется **привязкой к CPUs** (processor affinity). По умолчанию поток выполняется на любом доступном CPU, но можно изменить привязку к CPUs через Windows-функции планирования, или заданием маски привязки в заголовке образа.

Код Windows, отвечающий за планирование, реализован в ядре. Поскольку этот код рассредоточен по ядру, единого модуля или процедуры с названием «планировщик» нет. Совокупность процедур, выполняющих эти обязанности, называется **диспетчерам ядра** (kernel's dispatcher). Диспетчеризация потоков может быть вызвана любым из следующих событий.

* **Поток готов к выполнению** — например, он только что создан или вышел из состояния ожидания.
* **Поток выходит из состояния Running** (выполняется), так как его квант истек или поток завершается либо переходит в состояние ожидания.
* **Приоритет потока изменяется** в результате вызова системного сервиса или самой Windows.
* **Изменяется привязка к CPUs**, из-за чего поток больше не может работать на CPU, на котором он выполнялся.

**Уровни приоритета**

Как показано на Рис. 10, в Windows предусмотрено 32 уровня приоритета — от 0 до 31. Эти значения группируются так:

шестнадцать уровней **реального времени** (16-31);

пятнадцать **варьируемых (динамических)** уровней (1—15);

один **системный уровень** (0), зарезервированный для потока обнуления страниц (zero page thread).



Уровни приоритета потока назначаются с учетом двух разных точек зрений Windows API и ядра Windows. Windows API сначала упорядочивает процессы по классам приоритета, назначенным при их создании [Real-time (реального времени), High (высокий), Above Normal (выше обычного), Normal (обычный), Below Normal (ниже обычного) и Idle (простаивающий)], а затем — по относительному приоритету индивидуальных потоков в рамках этих процессов [Time-critical (критичный по времени), Highest (наивысший), Above-normal (выше обычного), Normal (обычный), Below-normal (ниже обычного), Lowest (наименьший) и Idle (простаивающий)].

Базовый приоритет каждого потока в Windows API устанавливается, исходя из класса приоритета его процесса и относительного приоритета самого потока. Связь между приоритетами Windows API и внутренними приоритетами ядра Windows (в числовой форме) показана на Рис. 11.

Если у процесса только одно значение приоритета (базовое), то у каждого потока их два: текущее и базовое. Решения, связанные с планированием, принимаются на основе текущего приоритета. В определенных обстоятельствах система может на короткое время повышать приоритеты потоков в динамическом диапазоне (1-15). Windows никогда не изменяет приоритеты потоков в диапазоне реального времени (16-31), поэтому у таких потоков базовый приоритет идентичен текущему.



**Состояния потоков**

За время своего существования поток может находиться в нескольких состояниях. Перечислим основные состояния:

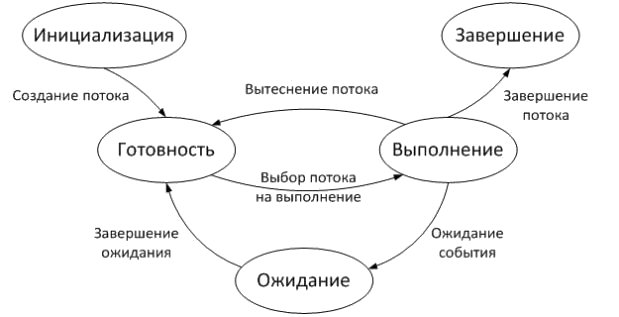
Готовность (Ready) – поток готов к выполнению и ждет своей очереди занять процессор.

Выполнение (Running) – поток выполняется на процессоре.

Ожидание (Waiting) – поток не может выполняться, поскольку ждет наступление некоторого события (например, завершения операции ввода-вывода или сообщения от другого потока)

Кроме основных существует ещё несколько состояний – Инициализация (Init), Завершение (Terminate), Состояние простоя (Standby), Переходное состояние (Transition), Состояние отложенной готовности (Deferred ready). Подробнее о них можно узнать в [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5); [2](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.2)].

На [рис.9.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=1#image.9.1) показаны основные состояния потока, возможные переходы между состояниями и условия переходов.



1. Кванты времени, выделяемые потокам, управление величиной кванта.

**Кванты**

В Windows имеется два базовых размера кванта – 2 интервала системного таймера и 12 интервалов. Если квант времени короткий, то потоки будут переключаться быстрее и "отзывчивость" (responsiveness) системы улучшится – это важное свойство для пользователя, поэтому в клиентских системах Windows по умолчанию используются короткие кванты. При этом производительность системы в целом снижается, поскольку потоки не будут успевать выполнять свои задачи в течение выделенного кванта, а частые переключения создадут высокие накладные расходы (служебные операции системы при смене потоков). Вследствие этого в серверных версиях Windows по умолчанию применяются длинные кванты.

Каждый процесс хранит величину кванта в поле QuantumReset структуры KPROCESS (файл base\ntos\inc\ke.h, строка 1029). Значение в этом поле равно количеству интервалов таймера, умноженному на 3. Например, для длинных квантов (12 интервалов) значение QuantumReset будет равно 36. Таким образом, при каждом срабатывании таймера (возникает прерывание) система уменьшает квант выполняющегося потока на 3 единицы.

Умножение на три введено для того чтобы можно было в разной степени уменьшать квант в двух различных ситуациях – срабатывании таймера (квант уменьшается на 3 единицы) и выходе из состояния ожидания (квант уменьшается на единицу). Уменьшение кванта при выходе потока из состояния ожидания применяется чтобы избежать ситуации бесконечно выполняющегося потока: если при каждом срабатывании таймера поток находится в состоянии ожидания, а при выходе из ожидания значение кванта не изменяется, то теоретически поток может выполняться бесконечно. Поэтому при выходе из состояния ожидания текущее значение его кванта уменьшается на единицу.

За изменение величины кванта отвечает функция KeSetQuantumProcess (файл base\ntos\ke\procobj.c, строка 1393).

Кроме длинных и коротких квантов в Windows реализовано динамическое увеличение размера кванта для потоков активного процесса (т.е. того процесса, окно которого в настоящий момент активно). За повышение кванта (и приоритета) отвечает функция PspComputeQuantumAndPriority (файл base\ntos\ps\psquery.c, строка 4415). Более подробную информацию о динамическом увеличении кванта см. [5, стр. 361].

1. Сценарии планирования выполнения потоков, переключение контекста, поток простоя.

**Сценарии планирования**

Известно, что вопрос «Какому потоку отдать CPU время?» Windows 2000 решает, исходя из приоритетов. Но как этот подход работает на практике? Следующие разделы иллюстрируют, как вытесняющая многозадачность, управляемая на основе приоритетов, действует на уровне потоков.

**Самостоятельное переключение**

Во-первых, поток может самостоятельно освободить CPU, перейдя в состояние ожидания на каком-либо объекте (например, событии, мьютексе, семафоре, порте завершения ввода-вывода, процессе, потоке, оконном сообщении и др.) путем вызова одной из многочисленных Windows-функций ожидания (скажем, WaitForSingleObject или WaitForMultipleObjects).

На Рис. 16 показано, как поток входит в состояние ожидания и как Windows выбирает новый поток для выполнения.

В состояние

ожидания

Ready

Running

Приоритет

14

15

16

17

18

19

20

Рис. 16. Самостоятельное переключение

На Рис. 16 поток (верхний блок) самостоятельно освобождает процессор, в результате чего к процессору подключается другой поток из очереди (отмеченный кольцом в колонке Running). Когда поток входит в состояние ожидания, квант не сбрасывается. Как уже говорилось, после успешного завершения ожидания квант потока уменьшается на одну единицу, что эквивалентно трети интервала таймера (исключение составляют потоки с приоритетом от 14 и выше, у которых после ожидания квант сбрасывается).

**Вытеснение**

В этом сценарии поток с более низким приоритетом вытесняется готовым к выполнению потоком с более высоким приоритетом. Такая ситуация может быть следствием двух обстоятельств:

1. завершилось ожидание потока с более высоким приоритетом (т.е. произошло событие, которого он ждал);
2. приоритет потока увеличился или уменьшился.

В любом из этих случаев Windows решает, продолжить выполнение текущего потока или вытеснить его потоком с более высоким приоритетом.

**ПРИМЕЧАНИЕ.** Потоки пользовательского режима могут вытеснять потоки режима ядра. То есть режим выполнения потока значения не имеет — определяющим фактором является его приоритет.

Когда поток вытесняется, он помещается в начало очереди готовых потоков соответствующего уровня приоритета. Эту ситуацию иллюстрирует Рис. 17.

**Ready**

**Running**

**Приоритет**

13

14

15

16

17

18

Из состояния ожидания

Рис. 17. Планирование потоков с вытеснением

На Рис. 17 поток с приоритетом 18 выходит из состояния ожидания и вновь захватывает процессор, вытесняя выполняемый в этот момент поток (с приоритетом 16) в очередь готовых потоков. Заметьте, что вытесненный поток помещается не в конец, а в начало очереди. После завершения вытеснившего потока вытесненный сможет отработать остаток своего кванта.

**Завершение кванта**

Когда поток израсходует свой квант времени CPU, Windows должна решить, следует ли понизить его приоритет и подключить к CPU другой поток.

Снизив приоритет потока, Windows ищет более подходящий для выполнения поток (таким потоком, например, будет любой из очереди готовых потоков с приоритетом выше нового приоритета текущего потока). Если Windows оставляет приоритет потока прежним и в очереди готовых потоков есть другие потоки с тем же приоритетом, она выбирает из очереди следующий поток с тем же приоритетом, а выполнявшийся до этого поток перемещает в хвост очереди (задавая ему новую величину кванта и переводя его из состояния Running в состояние Ready). Этот случай иллюстрирует Рис. 18. Если ни один поток с тем же приоритетом не готов к выполнению, текущему потоку выделяется еще один квант времени CPU.

**Ready**

**Running**

**Приоритет**

11

12

13

14

15

Рис. 18. Планирование потоков в момент завершения кванта текущего потока

**Завершение потока**

Завершаясь (после возврата из основной процедуры и вызова ExitThread или из-за уничтожения вызовом TerminateThread), поток переходит в состояние Terminated. Если в этот момент ни один описатель его объекта «поток» не открыт, поток удаляется из списка потоков процесса, и соответствующие структуры данных освобождаются.

**Переключение контекста**

Контекст потока и процедура его переключения зависят от архитектуры процессора. В типичном случае переключение контекста требует сохранения и восстановления следующих данных:

* указателя команд;
* указателей на стек ядра и пользовательский стек;
* указателя на адресное пространство, в котором выполняется поток (каталог таблиц страниц процесса).

Ядро сохраняет эту информацию, заталкивая ее в текущий стек ядра, обновляя указатель стека и сохраняя его в блоке KTHREAD потока. Далее указатель стека ядра устанавливается на стек ядра нового потока и загружается контекст этого потока. Если новый поток принадлежит другому процессу, в специальный регистр процессора загружается адрес его каталога таблиц страниц, в результате чего адресное пространство этого процесса становится доступным. При наличии отложенной АРС ядра запрашивается прерывание IRQL уровня 1. В ином случае управление передается загруженному для нового потока указателю команд, и выполнение этого потока возобновляется.

**Поток простоя**

Если нет ни одного потока, готового к выполнению на CPU, Windows подключает к данному CPU поток простоя (процесса Idle). Для каждого CPU создается свой поток простоя.

Разные утилиты для просмотра процессов в Windows по-разному называют процесс Idle. Диспетчер задач и Process Explorer обозначают его как «System Idle Process». Windows сообщает, что приоритет потока простоя равен 0. Но на самом деле у него вообще нет уровня приоритета, поскольку он выполняется лишь в отсутствие других потоков.

Холостой цикл, работающий при IRQL уровня «DPC/dispatch», просто запрашивает задания, например на доставку отложенных DPC или на поиск потоков, подлежащих диспетчеризации. Хотя последовательность работы потока простоя зависит от архитектуры, он все равно выполняет следующие действия.

Включает и отключает прерывания (тем самым давая возможность доставить отложенные прерывания).

Проверяет, нет ли у CPU незавершенных DPC. Если таковые есть, сбрасывает отложенное программное прерывание и доставляет эти DPC.

Проверяет, выбран ли какой-нибудь поток для выполнения на данном CPU, и, если да, организует его диспетчеризацию.

Вызывает из HAL процедуру обработки CPU в простое (если нужно выполнить какиелибо функции управления электропитанием).

В Windows Server 2003 поток простоя также проверяет наличие потоков, ожидающих выполнения на других CPU, но об этом пойдет речь в разделе по планированию потоков в многопроцессорных системах.

**Динамическое повышение приоритета**

Windows может динамически повышать значение текущего приоритета потока в одном из пяти случаев:

1. после завершения операций ввода-вывода;
2. при окончании ожидания на событии или семафоре исполнительной системы;
3. по окончании операции ожидания потоками активного процесса;
4. при пробуждении GUI-потоков из-за операций с окнами;
5. если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени.

Динамическое повышение приоритета предназначено для оптимизации общей пропускной способности и отзывчивости системы, а также для устранения потенциально «нечестных» сценариев планирования. Однако, как и любой другой алгоритм планирования, динамическое повышение приоритета — не панацея, и от него выигрывают не все приложения.

1. Особенности планирования потоков для многопроцессорных систем.

В однопроцессорной системе алгоритм планирования относительно прост: всегда выполняется поток с наивысшим приоритетом, готовый к выполнению. В многопроцессорной системе планирование усложняется, так как Windows пытается подключать поток к наиболее оптимальному для него CPU, учитывая предпочтительный и предыдущий CPU для этого потока, а также конфигурацию многопроцессорной системы. Поэтому, хотя Windows пытается подключать готовые к выполнению потоки с наивысшим приоритетом ко всем доступным CPUs, она гарантирует лишь то, что на одном из процессоров будет работать (единственный) поток с наивысшим приоритетом.

1. Диспетчер памяти ОС Windows: компоненты диспетчера, синхронизация, конфигурирование.

**Введение**

По умолчанию виртуальный размер процесса в 32-разрядной Windows — 2 Гб. Если образ помечен как поддерживающий большое адресное пространство и система загружается со специальным ключом (/3gb в файле boot.ini), 32-разрядный процесс может занимать до 3 Гб в 32-разрядной Windows и до 4 Гб в 64-разрядной. Размер виртуального адресного пространства процесса в 64-разрядной Windows составляет 7152 Гб на платформе IA64 и 8192 Гб на платформе хб4.

Максимальный объем физической памяти, поддерживаемый Windows, варьируется от 2 до 1024 Гб в зависимости от версии и редакции Windows. Так как виртуальное адресное пространство может быть больше или меньше объема физической памяти в компьютере, диспетчер управления памятью решает две главные задачи.

**Компоненты диспетчера памяти**

Диспетчер памяти является частью исполнительной системы Windows, содержится в файле Ntoskrnl.exe и включает следующие компоненты:

**Набор сервисов исполнительной системы** для выделения, освобождения и управления виртуальной памятью; большинство этих сервисов доступно через Windows API или интерфейсы драйверов устройств режима ядра.

**Обработчики ловушек трансляции недействительных адресов** (translation-notvalid) и нарушений доступа для разрешения аппаратно обнаруживаемых исключений, связанных с управлением памятью, а также загрузки в физическую память необходимых процессу страниц.

**Несколько ключевых компонентов**, работающих в контексте шести различных системных потоков режима ядра:

**Диспетчер рабочих наборов** (working set manager) с приоритетом 16. Диспетчер настройки баланса (системный поток, создаваемый ядром) вызывает его раз в секунду или при уменьшении объема свободной памяти ниже определенного порогового значения. Он реализует общие правила управления памятью, например усечение рабочего набора, старение и запись модифицированных страниц.

**Поток загрузки и выгрузки стеков** (process/stack swapper) с приоритетом 23. Выгружает (outswapping) и загружает (inswapping) стеки процесса и потока. При необходимости операций со страничным файлом этот поток пробуждается диспетчером рабочих наборов и кодом ядра, отвечающим за планирование.

**Подсистема записи модифицированных страниц** (modified page writer) с приоритетом 17. Записывает измененные страницы, зарегистрированные в списке модифицированных страниц, обратно в соответствующие страничные файлы. Этот поток пробуждается, когда возникает необходимость в уменьшении размера списка модифицированных страниц.

**Подсистема записи спроецированных страниц** (mapped page writer) с приоритетом 17. Записывает измененные страницы спроецированных файлов на диск. Пробуждается, когда нужно уменьшить размер списка модифицированных страниц или когда, страницы модифицированных файлов находятся в этом списке более 5 минут. Этот второй поток записи модифицированных страниц требуется потому, что он может генерировать ошибки страниц, в результате которых выдаются запросы на свободные страницы.

**Поток сегмента разыменования** (dereference segment thread) с приоритетом 18. Отвечает за уменьшение размеров системного кэша и изменение размеров страничного файла.

**Поток обнуления страниц** (zero page thread) с приоритетом 0. Заполняет нулями страницы, зарегистрированные в списке свободных страниц, (В некоторых случаях обнуление памяти выполняется более скоростной функцией MiZeroInParallel.).

**Внутренняя синхронизация**

Как и другие компоненты исполнительной системы Windows**, диспетчер памяти полностью** реентерабелен и поддерживает одновременное выполнение в multi-CPU системах, управляя тем, как потоки захватывают ресурсы. С этой целью диспетчер памяти контролирует доступ к собственным структурам данным, используя внутренние механизмы синхронизации, например спин-блокировку и ресурсы исполнительной системы.

Диспетчер памяти должен синхронизировать доступ к общесистемным ресурсам, как:

1. база данных **номеров фреймов страниц** (PFN) (контроль через спинблокировку),
2. объекты-разделы,
3. системный рабочий набор (контроль через спин-блокировку с заталкиванием указателя),
4. страничные файлы (контроль через объекты «мьютекс»).

В Windows XP и Windows Server 2003 ряд таких блокировок был либо удален, либо оптимизирован, что позволило резко снизить вероятность конкуренции.

К другим операциям, в которых больше не используется захват блокировок, относятся:

контроль квот на пулы подкачиваемой и неподкачиваемой памяти,

управление передачей страниц,

выделение и проецирование физической памяти через функции поддержки AWE (Address Windowing Extensions).

Эти изменения особенно важны в multi-CPU системах, где они позволили уменьшить частоту блокировки диспетчера памяти на период модификации со стороны другого CPU какой-либо глобальной структуры или вообще исключить такую блокировку.

**Конфигурирование диспетчера памяти**

Как и большинство компонентов Windows, диспетчер памяти старается автоматически оптимизировать работу систем различных масштабов и конфигураций при разных уровнях загруженности. Некоторые стандартные настройки можно изменить через параметры в разделе реестра HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemoryManagement, но, как правило, они оптимальны в большинстве случаев.

Многие пороговые значения и лимиты, от которых зависит политика принятия решений диспетчером памяти, вычисляются в период загрузки системы на основе доступной памяти и типа продукта (Windows 2000 Professional, Windows XP Professional и Windows XP Home Edition оптимизируется для интерактивного использования в качестве персональной системы, а системы Windows Server — для поддержки серверных приложений). Эти значения записываются в различные переменные ядра и впоследствии используются диспетчером памяти. Некоторые из них можно найти поиском в Ntoskrnl.exe глобальных переменных с именами, которые начинаются с Mm и содержат слово «maximum» или «minimum».

1. Диспетчер памяти ОС Windows: сервисы диспетчера памяти. Страницы, блокировка, гранулярность выделения памяти, резервирование памяти, разделяемая память и проецируемые файлы.

**Сервисы диспетчера памяти**

**Диспетчер памяти предоставляет набор системных сервисов** для выделения и освобождения виртуальной памяти, разделения памяти между процессами, проецирования файлов в память, сброса виртуальных страниц на диск, получения информации о диапазоне виртуальных страниц, изменения атрибутов защиты виртуальных страниц и блокировки в памяти.

Большинство этих сервисов предоставляется через Windows API. В него входят три группы прикладных функций управления памятью:

* для операций со страницами виртуальной памяти (Virtualxxx),
* **проецирования файлов в память** (CreateFileMapping, MapViemOfFile),
* **управления кучами** (Неарххх, а также функции из старых версий интерфейса — Localxxx и Globalxxx).

**Большие и малые страницы**

Виртуальное адресное пространство делится на единицы, называемые **страницами**. Это вызвано тем, что аппаратный блок управления памятью транслирует виртуальные адреса в физические по страницам. Поэтому страница — **наименьшая единица защиты на аппаратном уровне**. (Различные параметры защиты страниц описываются в разделе «Защита памяти» далее.) Страницы бывают двух размеров: малого и большого. Реальный размер зависит от аппаратной платформы (см. Таблица 1).

Таблица 1. Размер страницы

|  |  |  |
| --- | --- | --- |
| Архитектура | Размер малой страницы, кб | Размер большой страницы, Мб |
| x32 | 4 | 4 (в PAE системах 2) |
| x64 | 4 | 2 |
| IA64 | 8 | 16 |

**Блокировка памяти**

В целом, принятие решений о том, какие страницы следует оставить в физической памяти, лучше сохранить за диспетчером памяти. Однако в особых обстоятельствах можно подкорректировать работу диспетчера памяти. Существует два способа блокировки страниц в памяти.

Windows-приложения могут блокировать страницы в рабочем наборе своего процесса через функцию VirtualLock. Максимальное число страниц, которые процесс может блокировать, равно минимальному размеру его рабочего набора за вычетом восьми страниц. Следовательно, если процессу нужно блокировать большее число страниц, он может увеличить минимальный размер своего рабочего набора вызовом функции SetProcessWorkingSetSize.

Драйверы устройств могут вызывать функции режима ядра MmProbeAndLockPages,

MmLockPagableCodeSection и MmLockPagableSectionByHandle. Блокированные страницы остаются в памяти до снятия блокировки. Хотя число блокируемых страниц не ограничивается, драйвер не может блокировать их больше, чем это позволяет счетчик доступных резидентных страниц.

**Гранулярность выделения памяти**

Windows **выравнивает** начало каждого региона зарезервированного адресного пространства в соответствии с **гранулярностью выделения памяти** (allocation granularity). Это значение можно получить через Windows-функцию GetSystemInfo. В настоящее время оно равно 64 Кб. Такая величина выбрана из соображений поддержки будущих CPU с большим размером страниц памяти (до 64 Кб) или виртуально индексируемых кэшей (virtually indexed caches), требующих общесистемного выравнивания между физическими и виртуальными страницами (physical-to-virtual page alignment). Благодаря этому уменьшается риск возможных изменений, которые придется вносить в приложения, полагающиеся на определенную гранулярность выделения памяти. (Это ограничение не относится к коду Windows режима ядра — используемая им гранулярность выделения памяти равна одной странице.)

Windows также добивается, чтобы размер и базовый адрес зарезервированного региона адресного пространства всегда был кратен размеру страницы. Например, системы типа х86 используют страницы размером 4 Кб, и, если вы попытаетесь зарезервировать 18 Кб памяти, на самом деле будет зарезервировано 20 Кб. А если вы укажете базовый адрес 3 Кб для 18-килобайтного региона, то на самом деле будет зарезервировано 24 Кб.

**Разделяемая память и проецируемые файлы**

**Разделяемой** (shared memory) называется память, видимая более чем одному процессу или присутствующая в виртуальном адресном пространстве более чем одного процесса. Например, если два процесса используют одну и ту же DLL, есть смысл загрузить ее код в физическую память лишь один раз и сделать ее доступной всем процессам, проецирующим эту DLL (Рис. 1).

Виртуальная память процесса 1

Виртуальная память

процесса

2

Физическая память

Код

DLL

Рис. 1. Разделение памяти процессами

Каждый процесс поддерживает **закрытые области памяти** для хранения собственных данных, но программные инструкции и страницы немодифицируемых данных в принципе можно использовать совместно с другими процессами. Такой вид разделения реализуется автоматически, поскольку страницы кода в исполняемых образах проецируются с атрибутом «только для выполнения», а страницы, доступные для записи, — с атрибутом «копирование при записи» (copy-on-write).

Для реализации разделяемой памяти используются примитивы диспетчера памяти, объекты «раздел», которые в Windows XP называются **объектами «проекция файла»** (file mapping objects).

1. Защита памяти ОС Windows.

**Защита памяти**

Windows обеспечивает защиту памяти, предотвращая случайную или преднамеренную порчу пользовательскими процессами данных в адресном пространстве системы или других процессов. В Windows предусмотрено четыре основных способа защиты памяти.

**У каждого процесса имеется индивидуальное закрытое адресное пространство**, защищенное от доступа потоков других процессов, исключение составляют те случаи, когда процесс разделяет какие-либо страницы с другими процессами или когда у другого процесса есть права на доступ к объекту «процесс» для чтения и/или записи, что позволяет ему использовать функции ReadProcessMemory и WriteProcessMemory.

**Кроме косвенной защиты, обеспечиваемой трансляцией виртуальных адресов в физические**, все CPU, поддерживаемые Windows, предоставляют ту или иную форму аппаратной защиты памяти (например, доступ для чтения и записи, только для чтения и т.д.); конкретные механизмы такой защиты зависят от архитектуры CPU. Скажем, страницы кода в адресном пространстве процесса помечаются атрибутом «только для чтения», что защищает их от изменения пользовательскими потоками.

Атрибуты защиты памяти, определенные в Windows API, перечислены в Таблица 2 (см. также документацию на функции VirtualProtect, VirtualProtectEx, VirtualQuery и VirtualQueryEx).

Таблица 2. Атрибуты защиты памяти, определенные в Windows API

|  |  |
| --- | --- |
| Атрибут | Описание |
| PAGE\_NOACCESS | Любая попытка чтения, записи или выполнения кода на этой странице вызывает нарушение доступа |
| PAGE\_READONLY | Любая попытка записи (или выполнения кода на CPU без поддержки запрета на выполнение) на этой странице вызывает нарушение доступа, но чтение разрешено |
| PAGE\_READWRITE | Страница доступна для чтения и записи никакие действия не вызывают нарушения доступа |
| PAGE\_EXECUTE | Любая попытка записи на этой странице вызывает нарушение доступа, но выполнение кода (и чтения на всех существующих CPU разрешено |
| PAGE\_EXECUTEREAD | Любая попытка записи на этой странице вызывает нарушение доступа, но чтение и выполнение разрешено |
| PAGE\_EXECUTE\_READWRITE | Страница доступна для чтения, записи и выполнения — никакие действия не вызывают нарушения доступа |
| PAGE\_WRITECOPY | Любая попытка записи на этой странице заставляет систему выдать процессу закрытую копию данной страницы; при попытке выполнения кода на этой странице возникает нарушение доступа |
| PAGE\_EXECUTE\_WRITECOPY | Любая попытка записи на этой странице заставляет систему выдать процессу закрытую копию данной страницы. Чтение и выполнение кода на этой странице разрешено (в этом случае копия страницы не создается) |
| PAGE\_GUARD | Любая попытка чтения или записи сторожевой страницы вызывает исключение EXCEPTION\_GUARD\_PAGE, и она перестает быть сторожевой; этот атрибут можно комбинировать с любым другим атрибутом, кроме PAGE\_NOACCESS |
| PAGE\_NOCACHE | Используется некэшируемая физическая память, что, как правило, не рекомендуется делать. Имеет смысл для драйверов устройств, например, для проецирования буфера видеокадра без кэширования |
| PAGE\_WRITECOMBINE | Разрешает комбинированную запись на страницу памяти. В этом случае CPU может кэшировать запросы на запись в память для большей производительности. Например, при наличии нескольких запросов на запись по одному адресу возможна запись лишь по последнему запросу |

**Поток может менять атрибуты защиты виртуальных страниц раздела** (на уровне отдельных страниц), если это не противоречит разрешениям, указанным в ACL для данного раздела. Так, диспетчер памяти позволит потоку сменить атрибут страниц общего раздела «только для чтения» на «копирование при записи», но запретит его изменение на атрибут «для чтения и записи». Копирование при записи разрешается потому, что не влияет на другие процессы, тоже использующие эти данные.

1. Системные пулы памяти.

**Системные пулы памяти**

При инициализации системы диспетчер памяти создает два типа динамических пулов памяти, используемых компонентами режима ядра для выделения системной памяти.

**Пул неподкачиваемой памяти** (nonpaged pool). Состоит из диапазонов системных виртуальных адресов, которые всегда присутствуют в физической памяти и доступны в любой момент (при любом IRQL и из контекста любого процесса) без генерации ошибок страниц. Одна из причин существования такого пула — невозможность обработки ошибок страниц при IRQL уровня «DPC/dispatch» и выше.

**Пул подкачиваемой памяти** (paged pool). Регион виртуальной памяти в системном пространстве, содержимое которого система может выгружать в страничный файл и загружать из него. Драйверы, не требующие доступа к памяти при IRQL уровня «DPC/dispatch» и выше, могут использовать память из этого пула. Он доступен из контекста любого процесса. Оба пула находятся в системном адресном пространстве и проецируются на виртуальное адресное пространство любого процесса. Исполнительная система предоставляет функции для выделения и освобождения памяти в этих пулах (см. описание функций, чьи имена начинаются с ExAllocatePool, в Windows DDK).

**В single-CPU** системах создается три пула подкачиваемой памяти, а в multi-CPU — пять. Наличие нескольких подкачиваемых пулов уменьшает частоту блокировки системного кода при одновременных обращениях нескольких потоков к процедурам управления пулами. Начальный размер подкачиваемого и неподкачиваемого пулов зависит от объема физической памяти и может при необходимости расти до максимального значения, вычисляемого в период загрузки системы.

Таблица 3. Максимальные размеры пулов

|  |  |  |
| --- | --- | --- |
| Тип пула | Максимальный размер в 32-х  разрядных системах | Максимальный размер в 64х разрядных системах |
| Неподкачиваемый | 256 Мб (128 Мб, если был задан загрузочный параметр /3GB) | 128 Гб |
| Подкачиваемый | 491,875 Мб (в Windows 7000 и Windows ХР);  650 Мб (Windows Server 2003) | 128 Гб |

1. Структуры виртуального адресного пространства ОС Windows.

На виртуальное адресное пространство в Windows проецируются три основных вида данных:

• код и данные, принадлежащие процессу,

• код и данные, принадлежащие сеансу,

• общесистемные код и данные.

Каждому процессу выделяется собственное адресное пространство, недоступное другим процессам (если только у них нет разрешения на открытие процесса с правами доступа для чтения и записи). Потоки внутри процесса никогда не получают доступа к виртуальным адресам вне адресного пространства своего процесса, если только не проецируют данные на раздел общей памяти и/или не используют специальные функции, позволяющие обращаться к адресному пространству другого процесса.

**Структура виртуального адресного пространства**

Рассмотрим, из каких элементов состоит *виртуальное* *адресное пространство* процесса в 32 разрядных *Windows* ([рис.11.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.2)).

В пользовательском ВАП располагаются исполняемый образ процесса, динамически подключаемые библиотеки (*DLL*, *dynamic-link library*), *куча* процесса и стеки потоков.

При запуске программы создается процесс (см. лекцию 6 "Процессы и потоки"), при этом в *память* загружаются код и данные программы (исполняемый образ, *executable image*), а также необходимые программе динамически подключаемые библиотеки (*DLL*). Формируется *куча* (*heap*) – область, в которой процесс может выделять *память* динамическим структурам данных (т. е. структурам, размер которых заранее неизвестен, а определяется в ходе выполнения программы). *По* умолчанию размер кучи составляет 1 МБ, но при компиляции приложения или в ходе выполнения процесса может быть изменен. Кроме того, каждому потоку предоставляется *стек* (stack) для хранения локальных переменных и параметров функций, также *по* умолчанию размером 1 МБ.



**Рис. 11.2.**Структура виртуального адресного пространства

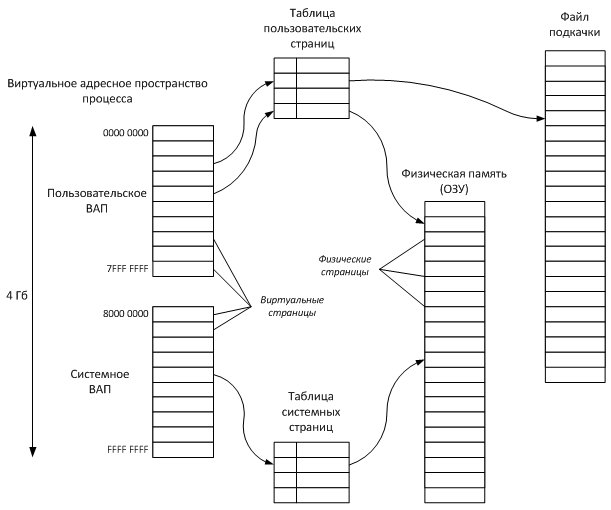
**В системном ВАП расположены:**

1. образы ядра (ntoskrnl.exe), исполнительной системы, HAL (hal.dll), драйверов устройств, требуемых при загрузке системы;
2. таблицы страниц процесса;
3. системный кэш;
4. пул подкачиваемой памяти (paged pool) – системная куча подкачиваемой памяти;
5. пул подкачиваемой памяти (nonpaged pool) – системная куча неподкачиваемой памяти;
6. другие элементы (см. [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5)]).

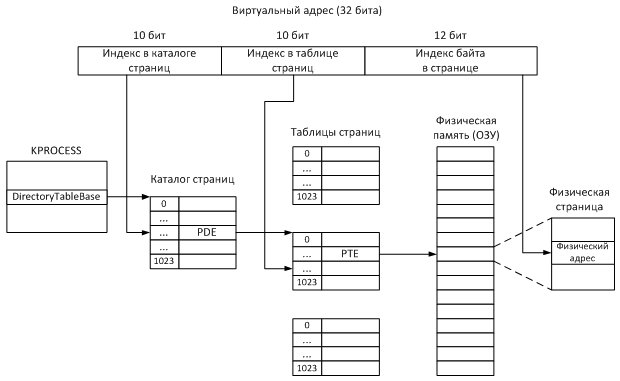
Переменные, в которых хранятся границы разделов в системном ВАП, приведены в [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5), стр. 442]. Вычисляются эти переменные в функции MmInitSystem (*файл* base\ntos\mm\mminit.c, строка 373), отвечающей за инициализацию подсистемы памяти. В файле base\ntos\mm\i386\mi386.h приведена структура ВАП и определены *константы*, связанные с управлением памятью (например, стартовый *адрес* системного кэша MM\_SYSTEM\_CACHE\_START, строка 199).

1. Трансляция виртуальных адресов. Операции ввода-вывода, связанные с подкачкой страниц. Обработка ошибок.

**Трансляция виртуального адреса** – это определение реального (физического) расположение ячейки памяти с данным виртуальным адресом, т. е. преобразование виртуального адреса в физический. Принцип трансляции показан на рис.11.1, здесь мы рассмотрим подробности трансляции и детали реализации в WRK.



Откуда процесс знает, где в памяти хранится каталог страниц? За это отвечает *поле* DirectoryTableBase структуры KPROCESS (*файл* base\ntos\inc\ke.h, строка 958, первый элемент массива). Схема трансляции адресов показана на [рис.11.3](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=2#image.11.3).



**Рис. 11.3.**Трансляция адресов

**Операции ввода-вывода, связанные с подкачкой страниц**

Такие операции ввода-вывода происходят в результате запроса на чтение страничного или проецируемого файла из-за ошибки страницы. Кроме того, поскольку в страничный файл могут помещаться и таблицы страниц, обработка ошибки страницы в случае таблицы страниц может повлечь за собой новые ошибки страниц.

Операции ввода-вывода, связанные с подкачкой, являются синхронными, т.е. поток ждет завершения подобной операции на каком-либо событии и она не может быть прервана вызовом асинхронной процедуры (АРС). Для идентификации ввода-вывода как связанною с подкачкой подсистема подкачки страниц (pager) вызывает функцию запроса ввода-вывода, указывая специальный модификатор. По завершении операции подсистема ввода-вывода освобождает событие. Это пробуждает подсистему подкачки страниц, и она продолжает свою работу.

В ходе операции ввода-вывода, связанной с подкачкой, поток, который вызвал ошибку страницы, не владеет критичными синхронизирующими объектами, используемыми при управлении памятью. Другие потоки того же процесса могут вызывать функции управления виртуальной памятью и обрабатывать ошибки страниц в ходе операции ввода-вывода, связанной с подкачкой. Однако подсистема подкачки страниц должна уметь выходить из некоторых ситуаций, которые могут возникать на момент завершения такой операции:

* другой поток в том же или другом процессе вызывает ошибку той же страницы, из-за чего происходит конфликт ошибок страницы (см. следующий раздел);
* страница удалена из виртуального адресного пространства и перепроецирована;
* сменился атрибут защиты страницы;
* ошибка относится к прототипному РТЕ, а страница, которая проецирует этот РТЕ, отсутствует в рабочем наборе.

Подсистема подкачки страниц выходит из таких ситуаций следующим образом. Перед запросом на операцию ввода-вывода, связанную с подкачкой, она сохраняет в стеке ядра потока статусную информацию, что позволяет после выполнения запроса распознать возникновение одной из перечисленных выше ситуаций и при необходимости отбросить ошибку страницы, не делая эту страницу действительной. Если команда, вызвавшая ошибку страницы, выдается повторно, вновь активизируется подсистема подкачки страниц, и РТЕ вычисляется заново.

**Обработка ошибок страниц**

Если битовый флаг Valid в РТЕ сброшен, это значит, что нужная страница по какой-либо причине сейчас недоступна процессу. Расскажем о типах недействительных РТЕ и о том, как разрешаются ссылки на такие РТЕ.

**ПРИМЕЧАНИЕ** Детально рассматриваются только РТЕ на 32-разрядной платформе х8б. РТЕ для 64-разрядных систем содержат аналогичную информацию.

При ссылке на недействительную страницу возникает ошибка страницы (page fault), и обработчик ловушки ядра перенаправляет ее обработчику MmAccessFault диспетчера памяти. Последняя функция, выполняемая в контексте вызвавшего ошибку потока, предпринимает попытку ее разрешения (если это возможно) или генерирует соответствующее исключение.

1. Страничные файлы. Дескрипторы виртуальных адресов. Объекты-разделы.

**Страничные файлы**

**Страничные файлы (page files)** предназначены для хранения модифицированных страниц, которые используются каким-то процессом, но должны быть выгружены из памяти на диск. Пространство в страничном файле резервируется, когда происходит начальная передача страниц, но реальные участки страничного файла не выбираются до тех пор, пока страницы не выгружаются на диск. Важно отметить, что система накладывает ограничение на число передаваемых закрытых страниц. Поэтому значение счетчика производительности Process: Page File Bytes на самом деле отражает суммарный объем закрытой памяти, переданной процессам. Соответствующие страницы могут находиться в страничном файле (частично или целиком) или, напротив, в физической памяти.

Диспетчер памяти отслеживает использование закрытой переданной памяти на глобальном уровне и по каждому процессу отдельно (в виде квоты страничного файла). И вновь эти данные отражают не размер использованного пространства в страничном файле, а объем переданной закрытой памяти. Соответствующие счетчики увеличиваются при передаче виртуальных адресов, требующих новых закрытых физических страниц.

**Дескрипторы виртуальных адресов**

Момент загрузки страниц в память диспетчер памяти определяет, используя алгоритм **подкачки по требованию** (demand-paging algorithm). Страница загружается с диска, если поток, обращаясь к ней, вызывает ошибку страницы. Подобно копированию при записи, подкачка по требованию является одной из форм отложенной оценки (lazy evaluation) выполнения операции только при ее абсолютной необходимости.

Диспетчер памяти использует отложенную оценку не только при загрузке страниц в память, но и при формировании таблиц, описывающих новые страницы. Например, когда поток передает память большой области виртуальной памяти с помощью VirtualAHoc*,* диспетчер памяти мог бы немедленно создать таблицы страниц, необходимые для доступа ко всему диапазону выделенной памяти.

При использовании алгоритма отложенной оценки выделение даже больших блоков памяти происходит очень быстро. Когда поток выделяет память, диспетчер памяти должен соответственно отреагировать. Для этого диспетчер памяти поддерживает набор структур данных, которые позволяют вести учет зарезервированных и свободных виртуальных адресов в адресном пространстве процесса. Эти структуры данных называются **дескрипторами виртуальных адресов** (virtual address descriptors, VAD). Для каждого процесса диспетчер памяти поддерживает свой набор VAD, описывающий состояние адресного пространства этого процесса. Для большей эффективности поиска VAD организованы в виде двоичного дерева с автоматической балансировкой. В Windows Server 2003 реализован **алгоритм дерева AVL** (это первые буквы фамилий его разработчиков — Adelson-Velskii и Landis), который обеспечивает более эффективную балансировку VAD-дерева, а это уменьшает среднее число операций сравнения при поиске VAD, соответствующего некоему виртуальному адресу.

**Объекты-разделы**

**Объект «раздел»** (section object), в подсистеме Windows **называемый объектом «проекция файла»** (file mapping object), представляет блок памяти, доступный двум и более процессам для совместного использования. Объект-раздел можно проецировать на страничный файл или другой файл на диске.

Исполнительная система использует разделы для загрузки исполняемых образов в память, а диспетчер кэша — для доступа к данным в кэшированном файле. Объекты «раздел» также позволяют проецировать файлы на адресные пространства процессов. При этом можно обращаться к файлу как к большому массиву, проецируя разные представления объекта-раздела и выполняя операции чтения-записи в памяти, а не в самом файле, — такие операции называются **вводам-выводам в проецируемые файлы** (mapped file I/O). Если программа обратится к недействительной странице (отсутствующей в физической памяти), возникнет ошибка страницы, и диспетчер памяти автоматически загрузит эту страницу в память из проецируемого файла. Если программа модифицирует страницу, диспетчер памяти сохранит изменения в файле в процессе обычных операций, связанных с подкачкой.

1. Рабочие наборы. Управление рабочими наборами.

**Рабочие наборы**

Сосредоточимся на виртуальной части Windows-процесса — таблицах страниц, РТЕ и VAD. Как известно, подмножество виртуальных страниц, резидентных в физической памяти, называется рабочим набором (working set). Существует три вида рабочих наборов:

• **процесса** — содержит страницы, на которые ссылаются его потоки;

• **системы** — содержит резидентное подмножество подкачиваемого системного кода (например, Ntoskrnl.exe и драйверов), пула подкачиваемой памяти и системного кэша;

• сеанса — в системах с включенной службой Terminal Services каждый сеанс получает свой рабочий набор. Он содержит резидентное подмножество специфичных для сеанса структур данных режима ядра, создаваемых частью подсистемы Windows, которая работает в режиме ядра (Win32k.sys), пула подкачиваемой памяти сеанса, представлений, проецируемых в сеансе, и других драйверов устройств, проецируемых на пространство сеанса.

Прежде чем детально рассматривать каждый тип рабочего набора, обсудим общие правила выбора страниц, загружаемых в память, и определения срока их пребывания в физической памяти.

**Управление рабочими наборами**

Все процессы начинают свой жизненный цикл с максимальным и минимальным размерами рабочего набора по умолчанию — 50 и 345 страниц соответственно, Хотя это мало что дает, эти значения по умолчанию можно изменить для конкретного процесса через Windows-функцию SetProcessWorkingSetSize, но для этого нужна привилегия Increase Scheduling Priority. Однако, если только не укажете процессу использовать жесткие лимиты на рабочий набор (новшество Windows Server 2003), эти лимиты игнорируются в том смысле, что диспетчер памяти разрешит процессу расширение за установленный максимум при наличии интенсивной подкачки страниц и достаточного объема свободной памяти (либо, напротив, уменьшит его рабочий набор ниже минимума при отсутствии подкачки страниц и при высокой потребности системы в физической памяти). Хотя в Windows 2000 степень расширения процесса за максимальную границу рабочего набора увязывалась с вероятностью его усечения, в Windows XP это решение принимается исключительно на основе того, к скольким страницам обращался процесс.

В Windows Server 2003 жесткие лимиты на размеры рабочего набора могут быть заданы вызовом функции SetProcessWorkingSetSizeEx с флагом QUOTA\_LIMITS\_HARDWS\_ENABLE. Этой функцией пользуется, например, Windows System Resource Manager (WSRM).

Максимальный размер рабочего набора не может превышать общесистемный максимум, вычисленный при инициализации системы и хранящийся в переменной ядра MmMaximwnWorkingSetSize. Это значение представляет собой число страниц, доступных на момент вычислений (суммарный размер списков обнуленных, свободных и простаивающих страниц), за вычетом 512 страниц. Однако существуют жесткие верхние лимиты на размеры рабочих наборов — они перечислены в Таблица 4.

Таблица 4. Верхний лимит на максимальные размеры рабочих наборов

|  |  |  |
| --- | --- | --- |
| Версия Windows | Максимальный рабочего набора | размер |
| х8б-версии Windows 2000, Windows XP, Windows XP SP1, Windows Server 2003 | 1984 Мб |  |
| х8б-версии Windows XP SP2, Windows Server 2003 SP1 | 2047,9 Мб |  |
| х8б-версии Windows, загружаемые с ключом /3GB | 3008 Мб | |
| IA64 | 7152 Гб | |
| хб4 | 8192 Гб | |

1. Управление безопасностью ОС Windows: классы безопасности, компоненты системы защиты.

**Классы безопасности**

**Trusted Computer System Evaluation Criteria**

Национальный центр компьютерной безопасности (National Computer Security Center, NCSC ) был создан в 1981 году в Агентстве национальной безопасности (NSA) Министерства обороны США. Одна из задач NCSC заключалась в определении рейтингов безопасности (см. Таблица 1), отражающих степень защищенности коммерческих ОС, сетевых компонентов и приложений. Эти рейтинги, детальное описание доступно , были определены в 1983 году и часто называются «Оранжевой книгой» («Orange Book»).

Таблица 1. Рейтинг безопасности TCSEC

|  |  |
| --- | --- |
| Рейтинг | Название |
| A1 | Verified Design |
| B3 | Security Domains |
| B2 | Structured Protection |
| B1 | Labeled Security Protection |
| C2 | Controlled Access protection |
| C1 | Discretionary Access Protection (obsolete) |
| D | Minimal Protection |

Стандарт TCSEC состоит из рейтингов «уровней доверия» («levels of trust» ratings), где более высокие уровни строятся на более низких, за счет последовательного ужесточения требований к безопасности и проверке. Ни одна ОС не соответствует уровню A1 (Verified Design). Хотя некоторым ОС присвоен один из уровней В, уровень С2 **считается достаточным и является высшим для ОС общего назначения**.

В июле 1995 года Windows NT 3.5 (Workstation и Server) с Service Pack 3 первой из всех версий Windows NT получила подтверждение об уровне безопасности С2. В марте 1999 года организация ITSEC, (Information Technology Security) правительства Великобритании присвоила Windows NT 4 с Service Pack 3 уровень ЕЗ, эквивалентный американскому уровню С2. Windows NT 4 с Service Pack 6а получила уровень С2 для сетевой и автономной конфигураций.

Основные требования к уровню безопасности С2 (они остались прежними) перечислены ниже.

**Механизм безопасной регистрации**, требующий уникальной идентификации пользователей. Доступ к компьютеру предоставляется лишь после аутентификации.

**Управление избирательным доступом**, позволяющее владельцу ресурса определять круг лиц, имеющих доступ к ресурсу, а также их права на операции с этим ресурсом. Владелец предоставляет пользователям и их группам различные права доступа.

**Аудит безопасности**, обеспечивающий возможность регистрации событий, связанных с защитой, а также любых попыток создания, удаления и обращения к системным ресурсам. При входе регистрируются идентификационные данные всех пользователей, что позволяет легко выявить любого, кто попытался выполнить несанкционированную операцию.

**Защита при повторном использовании объектов**, которая предотвращает просмотр одним из пользователей данных, удаленных из памяти другим, а также доступ к памяти, освобожденной предыдущим пользователем. Например, в некоторых ОС можно создать новый файл определенной длины, а затем просмотреть те данные, которые остались на диске и попали в область, отведенную под новый файл. Среди этих данных может оказаться конфиденциальная информация, хранившаяся в недавно удаленном файле другого пользователя. Так что защита при повторном использовании объектов устраняет потенциальную уязвимость в системе безопасности, заново инициализируя перед выделением новому пользователю все объекты, включая файлы и память. Windows также отвечает двум требованиям защиты уровня В.

**Функциональность пути доверительных отношений** (trusted path functionality), которая предотвращает перехват имен и паролей пользователей программами — троянскими конями. Эта функциональность реализована в Windows в виде входной сигнальной комбинации клавиш Ctrl+Alt+Del и не может быть перехвачена непривилегированными приложениями. Такая комбинация клавиш, известная как **SAS** (secure attention sequence), вызывает диалоговое окно входа в систему, обходя вызов его фальшивого эквивалента из троянского коня.

**Управление доверительными отношениями** (trusted facility management), которое требует поддержки набора ролей (различных типов учетных записей) для разных уровней работы в системе. Например, функции администрирования доступны только по одной группе учетных записей — Administrators (Администраторы).

Windows соответствует всем перечисленным требованиям.

### **Common Criteria**

В январе 1996 года США, Великобритания, Германия, Франция, Канада и Нидерланды опубликовали совместно разработанную спецификацию оценки безопасности Common Criteria for Information Technology Security Evaluation (CCITSE). Эта спецификация, чаще называемая Common Criteria (CC) , является международным стандартом оценки степени защищенности продуктов.

СС гибче уровней доверия TCSEC и по структуре ближе ITSEC, чем TCSEC. СС включает две концепции:

**профиля защиты** (Protection Profile, PP) — требования к безопасности разбиваются на группы, которые легко определять и сравнивать;

**объекта защиты** (Security Target, ST) — предоставляет набор требований к защите, которые могут быть подготовлены с помощью PP.

Windows 2000 оценивалась на соответствие требованиям Controlled Access PP, эквивалентным TCSEC C2, и на соответствие дополнительным требованиям Common Criteria в октябре 2002 года. Подробности можно найти. К значимым требованиям, не включенным в Controlled Access PP, но предъявляемым по условиям Windows 2000 Security Target, относятся:

**функции управления избирательным доступом** (Discretionary Access Control Functions), основанные на применении криптографии. Они peaлизуются файловой системой Encrypting File System и Data Protection API в Windows 2000;

**политика управления избирательным доступом** (Discretionary Access Control Policy) для дополнительных пользовательских объектов данных (User Data Objects), например объекты Desktop и WindowStation (реализуются подсистемой поддержки окон Windows 2000), а также объекты Active Directory (реализуются службой каталогов в Windows

2000);

**внутренняя репликация** (Internal Replication) для гарантированной синхронизации элементов данных, связанных с защитой, между физически раздельными частями системы Windows 2000 как распределенной ОС. Это требование реализуется службой каталогов Windows 2000 с применением модели репликации каталогов с несколькими хозяевами;

**утилизация ресурсов** (Resource Utilization) для физических пространств дисков. Это требование реализуется файловой системой NTFS в Windows 2000;

**блокировка интерактивного сеанса** (Interactive Session Locking) и путь **доверительных отношений** (Trusted Path) для первоначального входа пользователя (initial user logging on). Это требование реализуется компонентом Winlogon в Windows

2000;

**защита внутренней передачи данных** (Internal Data Transfer Protection), чтобы обезопасить данные от раскрытия и несанкционированной модификации при передаче между физически раздельными частями системы Windows 2000 как распределенной ОС. Это требование реализуется службой IPSEC в Windows 2000;

**систематическое устранение обнаруживаемых недостатков в системе защиты** (Systematic Flaw Remediation). Это требование реализуется Microsoft Security Response Center и Windows Sustained Engineering Team.

Так, Windows XP Embedded, Windows XP Professional и Windows Server 2003 уже прошли проверку на соответствие Common Criteria. Набор критериев расширен по сравнению с тем, который применялся к Windows 2000. Комитет Common Criteria также рассмотрел Windows XP и Windows Server 2003 (Standard, Enterprise и Datacenter Edition) для оценки технологий следующих типов :

* распределенной ОС;
* защиты конфиденциальных данных;
* управления сетью;
* службы каталогов;
* брандмауэра;
* VPN (Virtual Private Network);
* управления рабочим столом;

инфраструктуры открытого ключа (Public Key Infrastructure, PKI);  выдачи и управления сертификатами открытого ключа;  встраиваемой ОС.

**Компоненты системы защиты**

Ниже перечислены главные компоненты и базы данных, на основе которых реализуется защита в Windows.

**Монитор состояния защиты** (Security Reference Monitor, SRM).

**Компонент исполнительной системы** (\Windows\System32\Ntoskrnl.exe), отвечающий за определение структуры данных маркера доступа для представления контекста защиты, за проверку прав доступа к объектам, манипулирование привилегиями (правами пользователей) и генерацию сообщений аудита безопасности.

**Подсистема локальной аутентификации** (local security anthentication subsystem, LSASS). Процесс пользовательского режима, выполняющий образ

\Windows\System32\Lsass.exe, который отвечает за политику безопасности в локальной системе (например, круг пользователей, имеющих право на вход в систему, правила, связанные с паролями, привилегии, выдаваемые пользователям и их группам, параметры аудита безопасности системы), а также за аутентификацию пользователей и передачу сообщений аудита безопасности в Event Log. Основную часть этой функциональности реализует сервис локальной аутентификации Lsasrv

(\Windows\System32\Lsasrv.dll) — DLL-модуль, загружаемый Lsass.

**База данных политики LSASS**. База данных, содержащая параметры политики безопасности локальной системы. Она хранится в разделе реестра HKLM\SECURITY и включает следующую информацию: каким доменам доверена аутентификация попыток входа в систему, кто имеет права на доступ к системе и каким образом, кому предоставлены те или иные привилегии и какие виды аудита следует выполнять. База данных политики LSASS также хранит «секреты», которые включают в себя регистрационные данные, применяемые для входа в домены и при вызове Windowsсервисов.

**Диспетчер учетных записей безопасности** (Security Accounts Manager, SAM). Набор подпрограмм, отвечающих за поддержку базы данных, которая содержит имена пользователей и группы, определенные на локальной машине. Служба SAM, реализованная как \Windows\System32\Samsrv.dll, выполняется в процессе Lsass.

**База данных SAM**. База данных, которая в системах, отличных от контроллеров домена, содержит информацию о локальных пользователях и группах вместе с их паролями и другими атрибутами. На контроллерах домена SAM содержит определение и пароль учетной записи администратора, имеющего права на восстановление системы. Эта база данных хранится в разделе реестра HKLM\SAM.

**Active Directory**. Служба каталогов, содержащая базу данных со сведениями об объектах в домене. **Домен** — это совокупность компьютеров и сопоставленных с ними групп безопасности, которые управляются как единое целое. Active Directory хранит информацию об объектах домена, в том числе о пользователях, группах и компьютерах. Сведения о паролях и привилегиях пользователей домена и их групп содержатся в Active Directory и реплицируются на компьютеры, выполняющие роль контроллеров домена. Сервер Active Directory, реализованный как \Windows\System32\Ntdsa.dll, выполняется в процессе Lsass.

**Пакеты аутентификации**. DLL-модули, выполняемые в контексте процесса Lsass и клиентских процессов и реализующие политику аутентификации в Windows. DLL аутентификации отвечает за проверку пароля и имени пользователя, а также за возврат LSASS (в случае успешной проверки) детальной информации о правах пользователя, на основе которой LSASS генерирует **маркер** (token).

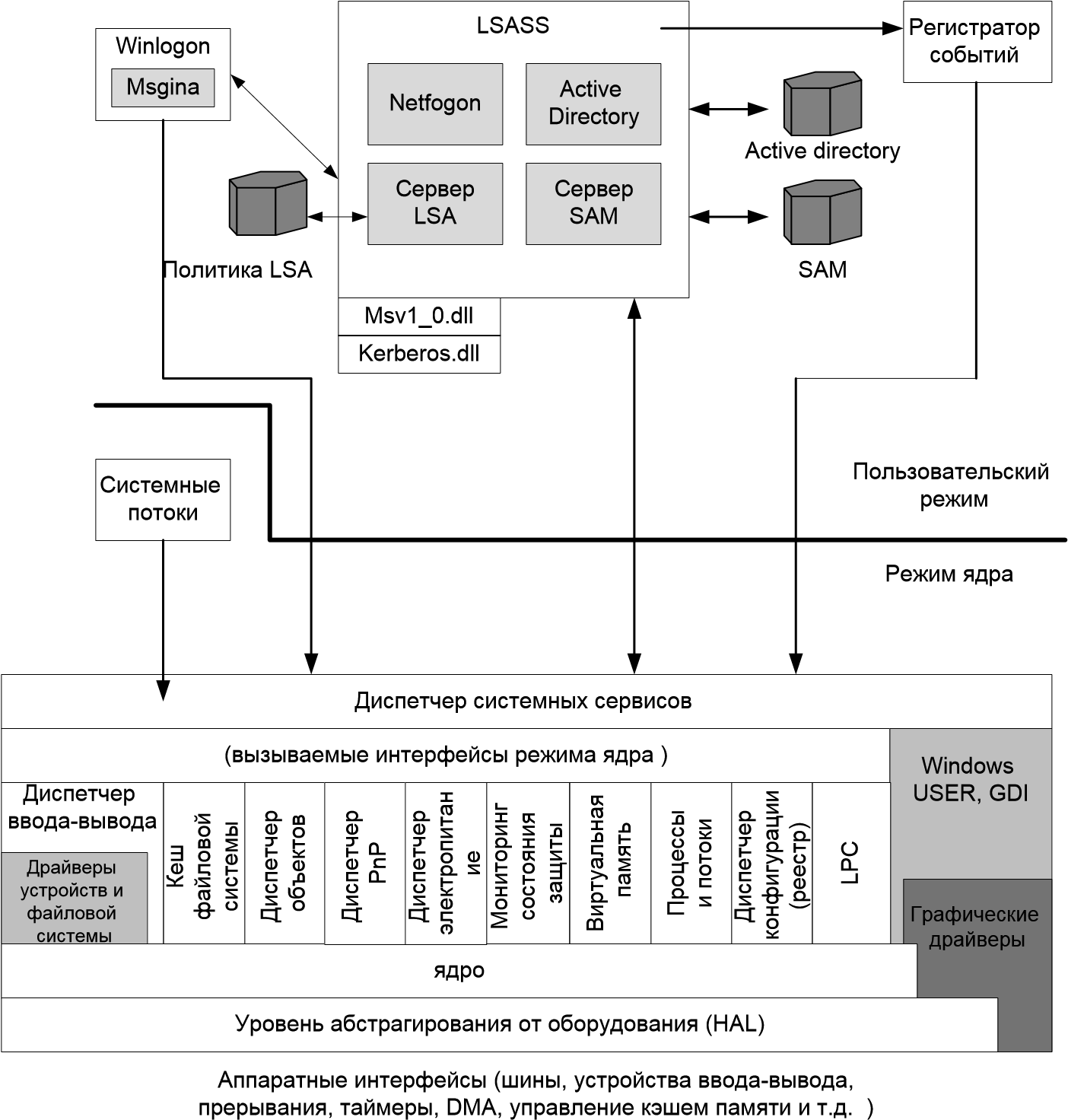
**Процесс входа (Winlogon)**. Процесс пользовательского режима

(\Windows\System32\Winlogon.exe), отвечающий за поддержку SAS и управление сеансами интерактивного входа в систему. Например, при регистрации пользователя Winlogon создаст оболочку — пользовательский интерфейс.

**GINA (Graphical Identification and Authentication)**. DLL пользовательского режима, выполняемая в процессе Winlogon и применяемая для получения пароля и имени пользователя или PIN-кода смарт-карты. Стандартная GINA хранится в \Windows\System32\Msgina.dll.

**Служба сетевого входа (Netlogon)**. Windows-сервис (\Windows\System32 \Netlogon.dll), устанавливающий защищенный канал с контроллером домена, по которому посылаются запросы, связанные с защитой, например для интерактивного входа (если контроллер домена работает под управлением Windows NT), или запросы на аутентификацию от LAN Manager либо NT LAN Manager (v1 и v2).

**Kernel Security Device Driver (KSecDD)**. Библиотека функций режима ядра, реализующая интерфейсы LPC (local procedure call), которые используются другими компонентами защиты режима ядра — в том числе **шифрующей файловой системой** (Encrypting File System, EFS) — для взаимодействия с LSASS в пользовательском режиме. KsecDD содержится в \Windows\System32\Drivers\Ksecdd.sys. На Рис. 1 показаны взаимосвязи между некоторыми из этих компонентов и базами данных, которыми они управляют.



1. Защита объектов в ОС Windows: проверка прав доступа. Идентификаторы защиты. Маркеры. Олицетворение.

**Проверка прав доступа**

Модель защиты Windows требует, чтобы поток заранее — еще до открытия объекта — указывал, какие операции он собирается выполнять над этим объектом. Система проверяет тип доступа, запрошенный потоком, и, если такой доступ ему разрешен, он получает описатель, позволяющий ему (и другим потокам того же процесса) выполнять операции над объектом. Диспетчер объектов регистрирует права доступа, предоставленные для данного описателя, в таблице описателей, принадлежащей процессу.

Одно из событий, заставляющее диспетчер объектов проверять права доступа, — **открытие процессом существующего объекта по имени.** При открытии объекта по имени диспетчер объектов ищет его в своем пространстве имен. Если этого объекта нет во вторичном пространстве имен (например, в пространстве имен реестра, принадлежащем диспетчеру конфигурации, или в пространстве имен файловой системы, принадлежащем драйверу файловой системы), диспетчер объектов вызывает внутреннюю функцию ObpCreateHandle. Как и подсказывает ее имя, она создает элемент в таблице описателей, который сопоставляется с объектом. Однако ObpCreateHandle вызывает функцию исполнительной системы ExCreateHandle и создаст описатель, только если другая функция диспетчера объектов, ObpIncrementHandleCount, сообщает, что поток имеет право на доступ к данному объекту. Правда, реальную проверку прав доступа осуществляет другая функция диспетчера объектов, ObpCheckObjectAccess, которая возвращает результаты проверки функции ObpIncrementHandleCount.

Другое событие, заставляющее диспетчер объектов выполнять проверку прав доступа, — **ссылка процесса на объект по существующему описателю**. Подобные ссылки часто делаются косвенно, например при манипуляциях с объектом через Windows API с передачей его описателя. Допустим, поток, открывающий файл, запрашивает доступ для чтения из файла. Если у потока есть соответствующие права, определяемые его контекстом защиты и параметрами защиты файла, диспетчер объектов создает описатель данного файла в таблице описателей, которая принадлежит процессу — владельцу этого потока. Информация о предоставленном процессу типе доступа сопоставляется с описателем и сохраняется диспетчером объектов.

**Идентификаторы защиты**

Для идентификации объектов, выполняющих в системе различные действия, Windows использует не имена (которые могут быть не уникальными), а **идентификаторы защиты** (security identifiers, SID). SID имеются у пользователей, локальных и доменных групп, локальных компьютеров, доменов и членов доменов. **SID** представляет собой числовое значение переменной длины, формируемое из номера версии структуры SID, 48-битного кода агента идентификатора и переменного количества 32-битных кодов **субагентов** и/или **относительных идентификаторов** (relative identifiers, RID). **Код агента идентификатора** (identifier authority value) определяет агент, выдавший SID. Таким агентом обычно является локальная система или домен под управлением Windows. Коды субагентов идентифицируют попечителей, уполномоченных агентом, который выдал SID, a RID — не более чем средство создания уникальных SID на основе **общего базового SID** (common-based SID). Поскольку длина SID довольно велика и Windows старается генерировать истинно случайные значения для каждого SID, вероятность появления двух одинаковых SID практически равна нулю.

В текстовой форме каждый SID начинается с префикса S, за которым следуют группы чисел, разделяемые дефисами, например:

S-1-5-21-1463437245-1224812800-863842198-1128

В этом SID номер версии равен 1, код агента идентификатора — 5 (центр безопасности Windows), далее идут коды четырех субагентов и один RID в конце (1128). Этот SID относится к домену, так что локальный компьютер этого домена получит SID с тем же номером версии и кодом агента идентификатора; кроме того, в нем будет столько же кодов субагентов.

Таблица 2. Некоторые общеизвестные SID

|  |  |  |
| --- | --- | --- |
| SID | Группа | Описание |
| S-1-1-0 | Everyone | Группа, включающая всех пользователей |
| S-1-2-0 | Local | Пользователи, которые регистрируются на терминалах, локально (физически) подключенных к системе |
| S-1-3-0 | Creator Owner ID | Идентификатор защиты, подлежащий замене идентификатором защиты пользователя, создавшего новый объект; этот SID применяется в наследуемых АСЕ |
| S-1-3-1 | Creator Group ID | Идентификатор защиты, подлежащий замене SID основной группы, в которую входит пользователь, создавший новый объект; этот SID применяется в наследуемых АСЕ |

**Маркеры**

Для идентификации контекста защиты процесса или потока SRM использует объект, называемый **маркером** (token), или **маркером доступа** (access token). В контекст защиты входит информация, описывающая привилегии, учетные записи и группы, сопоставленные с процессом или потоком. В процессе входа в систему (этот процесс рассматривается в конце главы) Winlogon создает начальный маркер, представляющий пользователя, который входит в систему, и сопоставляет его с начальным процессом (или процессами) — по умолчанию запускается Userinit.exe. Так как дочерние процессы по умолчанию наследуют копию маркера своего создателя, все процессы в сеансе данного пользователя выполняются с одним и тем же маркером. Можно сгенерировать маркер вызовом Windows-функции LogonUser и применить его для создания процесса, который будет выполняться в контексте защиты пользователя, зарегистрированного с помощью функции LogonUser, с этой целью вы должны передать полученный маркер Windows-функции CreateProcessAsUser. Функция CreateProcessWithLogon тоже создает маркер, создавая новый сеанс входа с начальным процессом. Именно так команда runas запускает процессы с **альтернативными маркерами**.

Механизмы защиты в Windows используют два элемента маркера, определяя, какие объекты доступны и какие операции можно выполнять. Первый элемент состоит из SID учетной записи пользователя и полей SID групп. Используя SID-идентификаторы, SRM определяет, можно ли предоставить запрошенный тип доступа к защищаемому объекту, например к файлу в NTFS.

Вторым элементом маркера, определяющим, что может делать поток или процесс, которому назначен данный маркер, является **список привилегий** — **прав, сопоставленных с маркером**. Примером привилегии может служить право процесса или потока, сопоставленного с маркером, на выключение компьютера. (Подробнее привилегии будут рассмотрены позже.) Поля основной группы маркера по умолчанию и **списка управления избирательным доступом** (discretionary access-control list, DACL) представляют собой атрибуты защиты, применяемые Windows к объектам, которые создаются процессом или потоком с использованием маркера. Включая в маркеры информацию о защите, Windows упрощает процессам и потокам создание объектов со стандартными атрибутами защиты, так как в этом случае им не требуется запрашивать информацию о защите при создании каждого объекта.

Маркер может быть **основным** (primary token) (идентифицирует контекст защиты процесса) и **олицетворяющим** (impersonation token) (применяется для временного заимствования потоком другого контекста защиты — обычно другого пользователя). Маркеры олицетворения сообщают уровень олицетворения, определяющий, какой тип олицетворения активен в маркере.

Остальные поля маркера служат для информационных нужд. Поле источника маркера содержит сведения (в текстовой форме) о создателе маркера. Оно позволяет различать такие источники, как диспетчер сеансов Windows, сетевой файл-сервер или RPCсервер. Идентификатор маркера представляет собой **локально уникальный идентификатор** (locally unique identifier, LUID), который SRM присваивает маркеру при его создании. Исполнительная система поддерживает свой LUID — счетчик, с помощью которого она назначает каждому маркеру уникальный числовой идентификатор.

Еще одна разновидность LUID - **идентификатор аутентификации** (authentication TD). Он назначается маркеру создателем при вызове функции LsaLogonUser. Если создатель не указывает LUID, то LSASS формирует LUID из LUID исполнительной системы. LSASS копирует идентификатор аутентификации для всех маркеров — потомков начального маркера. Используя этот идентификатор, программа может определить, принадлежит ли какой-то маркер тому же сеансу, что и остальные маркеры, анализируемые данной программой.

**Олицетворение**

**Олицетворение** (impersonation) — мощное средство, часто используемое в модели защиты Windows. Олицетворение также применяется в модели программирования «клиент-сервер». Например, серверное приложение может экспортировать ресурсы (файлы, принтеры или базы данных). Клиенты, которые хотят обратиться к этим ресурсам, посылают серверу запрос. Получив запрос, сервер должен убедиться, что у клиента есть разрешение на выполнение над ресурсом запрошенных операций. Так, если пользователь на удаленной машине пытается удалить файл с сетевого диска NTFS, сервер, экспортирующий этот сетевой ресурс, должен проверить, имеет ли пользователь право удалить данный файл. Казалось бы, в таком случае сервер должен запросить учетную запись пользователя и SID-идентификаторы группы, а также просканировать атрибуты защиты файла. Но этот процесс труден для программирования, подвержен ошибкам и не позволяет обеспечить поддержку новых функций защиты. Поэтому Windows в таких ситуациях предоставляет серверу сервисы олицетворения.

**Олицетворение позволяет** серверу уведомить SRM (Security Reference Monitor) о временном заимствовании профиля защиты клиента, запрашивающего ресурс. После этого сервер может обращаться к ресурсам от имени клиента, а SRM — проводить проверку его прав доступа. Обычно серверу доступен более широкий круг ресурсов, чем клиенту, и при олицетворении сервер может терять часть исходных прав доступа. Также вероятно и обратное: при олицетворении сервер может получить дополнительные права.

1. Управление безопасностью ОС Windows: дескрипторы защиты и управление доступом. Определение прав доступа.

**Дескрипторы защиты и управление доступом**

Маркеры, которые идентифицируют удостоверения пользователя, являются лишь частью выражения, описывающего защиту объектов. Другая его часть — информация о защите, сопоставленная с объектом и указывающая, кому и какие действия разрешено выполнять над объектом. Структура данных, хранящая эту информацию, называется **дескриптором защиты** (security diescriptor). Дескриптор защиты включает следующие атрибуты.

**Номер версии**. Версия модели защиты SRM, использованной для создания дескриптора.

**Флаги**. Необязательные модификаторы, определяющие поведение или характеристики дескриптора. Пример — флаг SE\_DACL\_PROTECTED, который запрещает наследование дескриптором параметров защиты от другого объекта.

**SID владельца**. Идентификатор защиты владельца.

**SID группы**. Идентификатор защиты основной группы для данного объекта (используется только POSIX).

**Список управления избирательным доступом** (discretionary асcess-control list, DACL) Указывает, кто может получать доступ к объек-ту и какие виды доступа.

**Системный список управления доступом** (system access-control list, SACL), указывает какие операции и каких пользователей должны регистрироваться в журнале аудита безопасности.

**Список управления доступом** (access-control list, ACL) состоит из заголовка и может содержать элементы (access-control entries, АСЕ). Существует два типа ACL: DACL и SACL. В DACL каждый АСЕ содержит SID и маску доступа (а также набор флагов), причем АСЕ могут быть четырех типов - «доступ разрешен» (access allowed), «доступ отклонен» (access denied), «разрешенный объект» (allowed-objecl) и «запрещенный объект» (denied-object). Первый тип АСЕ разрешает пользователю доступ к объекту, а второй — отказывает в предоставлении прав, указанных в маске доступа.

Разница между АСЕ типа «разрешенный объект» и «доступ разрешен», а также между АСЕ типа «запрещенный объект» и «доступ отклонен» заключается в том, что эти типы используются только в Active Directory. АСЕ этих типов имеют поле **глобально уникального идентификатора** (globally unique identifier, GUID), которое сообщает, что данный АСЕ применим только к определенным объектам или подобъектам (с GUIDидентификаторами). Кроме того, необязательный GUID указывает, что тип дочернего объекта наследует АСЕ при его (объекта) создании в контейнере Active Directory, к которому применен АСЕ. (GUID — это гарантированно уникальный 128-битный идентификатор.)

DACL

Разрешить Разрешить Разрешить

Объект Дескриптор пользователю группе TEAM1 любому доступ к

«файл» защиты USER1 доступ доступ для файлу для

для чтения чтения и записи выполнения

ACE ACE ACE

Рис. 2. Список управления избирательным доступом

**Определение прав доступа**

Для определения прав доступа к объекту используются два алгоритма:

* сравнивающий запрошенные права с максимально возможными для данного объекта и экспортируемый в пользовательский режим в виде Windows-функции

GetEffectiveRightsFromAcl;

* проверяющий наличие конкретных прав доступа и активизируемый через Windows-функцию AccessCheck или AccessCheckByType.

**Первый алгоритм** проверяет элементы DACL следующим образом.

1. В отсутствие DACL (DACL = null) объект является незащищенным, и система защиты предоставляет к нему полный доступ.
2. Если у вызывающего потока имеется привилегия на захват объекта во владение (takc-owncrship privilege), система защиты предоставляет владельцу право на доступ для записи (write-owner access) до анализа DACL (что такое привилегия захвата объекта во владение и право владельца на доступ для записи, мы поясним чуть позже).
3. Если вызывающий поток является владельцем объекта, ему предоставляются права управления чтением (read-control access) и доступа к DACL для записи (write-DACL access).
4. Из маски предоставленных прав доступа удаляется маска доступа каждого АСЕ типа «доступ отклонен», SID которого совпадает с SID маркера доступа вызывающего потока.
5. К маске предоставленных прав доступа добавляется маска доступа каждого АСЕ типа «доступ разрешен», SID которого совпадает с SID маркера доступа вызывающего потока (исключение составляют права доступа, в предоставлении которых уже отказано).

После анализа всех элементов DACL рассчитанная маска предоставленных прав доступа возвращается вызывающему потоку как максимальные права доступа. Эта маска отражает полный набор типов доступа, которые этот поток сможет успешно запрашивать при открытии данного объекта.

Все сказанное применимо лишь к той разновидности алгоритма, которая работает в режиме ядра. Его Windows-версия, реализованная функцией GetEffectiveRightsFromAcl, отличается отсутствием шага 2, а также тем, что вместо маркера доступа она рассматривает SID единственного пользователя или группы.

**Второй алгоритм** проверяет, можно ли удовлетворить конкретный запрос на доступ, исходя из маркера доступа вызывающего потока. У каждой Windows-функции открытия защищенных объектов есть параметр, указывающий желательную маску доступа — последний элемент выражения, описывающего защиту объектов. Чтобы определить, имеет ли вызывающий поток право на доступ к защищенному объекту, выполняются следующие операции.

1. В отсутствие DACL (DACL = null) объект является незащищенным, и система защиты предоставляет к нему запрошенный тип доступа.
2. Если у вызывающего потока имеется привилегия на захват объекта во владение, система защиты предоставляет владельцу право на доступ для записи, а затем анализирует DACL. Однако, если такой поток запросил только доступ владельца для записи, система защиты предоставляет этот тип доступа и не просматривает DACL.
3. Если вызывающий поток является владельцем объекта, ему предоставляются права управления чтением и доступа к DACL для записи. Если вызывающий поток запросил только эти права, система защиты предоставляет их без просмотра DACL.
4. Просматриваются все АСЕ в DACL — от первого к последнему. Обработка АСЕ выполняется при одном из следующих условий:
5. STD в АСЕ типа «доступ отклонен» совпадает с незаблокированным STD (SID могут быть незаблокированными и заблокированными) или SID с атрибутом проверки только на запрет в маркере доступа вызывающего потока;
6. SID в АСЕ типа «доступ разрешен» совпадает с незаблокированным SID в маркере доступа вызывающего потока, и этот SID не имеет атрибута проверки только на запрет;
7. Идет уже второй проход поиска в дескрипторе ограниченных SID, и SID в АСЕ совпадает с ограниченным SID в маркере доступа вызывающего потока.
8. В случае АСЕ типа «доступ разрешен» предоставляются запрошенные права из маски доступа АСЕ; проверка считается успешной, если предоставляются все запрошенные права. Доступ к объекту не предоставляется в случае АСЕ типа «доступ отклонен» и отказа в предоставлении какого-либо из запрошенных прав.
9. Если достигнут конец DACL и некоторые из запрошенных прав доступа еще не предоставлены, доступ к объекту запрещается.
10. Если все права доступа предоставлены, но в маркере доступа вызывающего потока имеется хотя бы один ограниченный SID, то система повторно сканирует DACL в поисках АСЕ, маски доступа которых соответствуют набору запрошенных прав доступа. При этом также идет поиск АСЕ, SID которых совпадает с любым из ограниченных SID вызывающего потока. Поток получает доступ к объекту, если запрошенные права доступа предоставлялись после каждого прохода по DACL.

Поведение обоих алгоритмов проверки прав доступа зависит от относительного расположения разрешающих и запрещающих АСЕ. Возьмем для примера объект с двумя АСЕ, первый из которых указывает, что определенному пользователю разрешен полный доступ к объекту, а второй отказывает в доступе. Если разрешающий АСЕ предшествует запрещающему, пользователь получит полный доступ к объекту. При другом порядке этих АСЕ пользователь вообще не получит доступа к объекту.

ACE

Пользователь

:

DaveC

Группа

1

:

Administrators

Группа

2

:

Writers

Запрос на открытие файла для записи

Версия

Флаги

SID

владельца

Основная группа

Указатель

DACL

Указатель

SACL

Отклонить

Writers

Чтение и запись

Разрешить

DaveC

Чтение и запись

Дескриптор защиты

ACE

Отклонен

Объект

Маркер доступа

Рис. 3. Пример проверки прав доступа

1. Права и привилегии учетных записей, суперпривелегии, аудит безопасности ОС Windows. Вход в систему, политика ограниченного использования программ.

**Права и привилегии**

Кроме операций с объектами система должна контролировать множество других действий пользователей, например, вход в систему, включение/выключение компьютера, изменение системного времени, *загрузка* драйверов и т.д.

Для управления такими действиями, не связанными с доступом к конкретным объектам, система использует два механизма – ***права***учетных записей и привилегии.

Право учетной записи (*account* *right*) – разрешение или запрет на определенный вид входа в систему.

Различают следующие виды входа:

* интерактивный (локальный) вход;
* вход из сети;
* вход через службу удаленных рабочих столов (ранее называлось – "через службу терминалов");
* вход в качестве службы;
* вход в качестве пакетного задания.

Проверка прав учетных записей осуществляется не в ядре, а в процессах Winlogon.exe и Lsass.exe.

***Привилегия*** (privilege) – разрешение или запрет определенных действий в системе, например, включение/выключение компьютера или *загрузка* драйверов. Привилегии хранятся в *поле* *Privileges* структуры маркера доступа *TOKEN* (см. выше).

**Суперпривилегии**

Несколько привилегий дают настолько широкие права, что пользователя, которому они назначаются, называют **«суперпользователем»** — он получает полный контроль над компьютером. Эти привилегии позволяют получать неавторизованный доступ к закрытым ресурсам и выполнять любые операции.

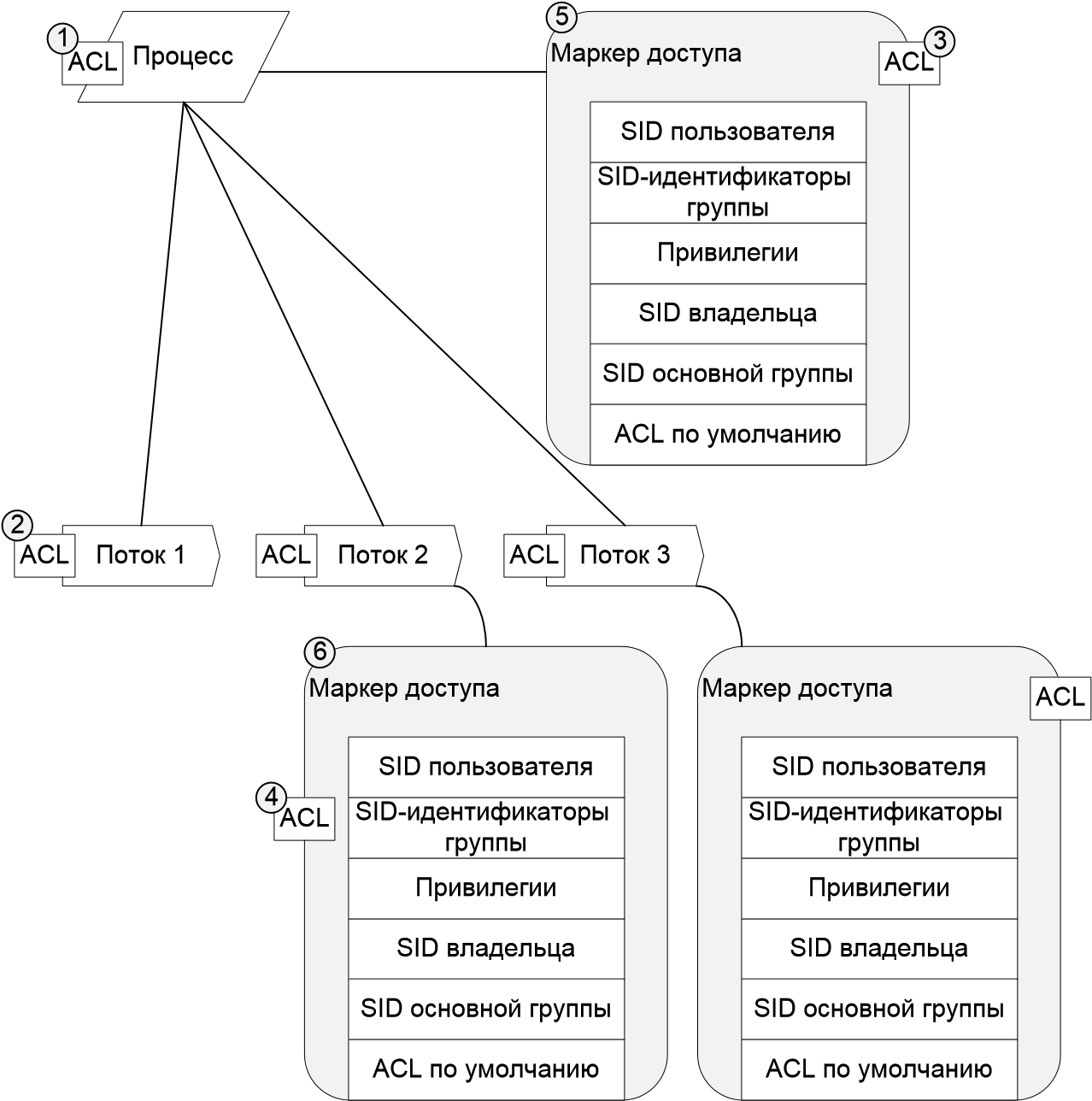


Рис. 6. Структуры защиты процессов и потоков

**Вход в систему**

При **интерактивном входе в систему** (в отличие от входа через сеть) происходит взаимодействие с процессами Winlogon, Lsass, одним или несколькими пакетами аутентификации, а также SAM или Active Directory. **Пакеты аутентификации** (authentication packages) — это DLL-модули, выполняющие проверки, связанные с аутентификацией. Пакетом аутентификации Windows для интерактивного входа в домен является Kerberos, a MSV1\_0 — аналогичным пакетом для интерактивного входа на локальные компьютеры, доменного входа в доверяемые домены под управлением версий Windows, предшествовавших Windows 2000, а также для входа в отсутствие контроллера домена.

**Winlogon** — доверяемый процесс, отвечающий за управление взаимодействием с пользователем в связи с защитой.

**Политики ограниченного использования программ**

Злонамеренный код вроде вирусов и червей создает все больше проблем. В Windows XP введен механизм **Software Restriction Policies** (Политики ограниченного использования программ), который позволяет администраторам контролировать образы и сценарии, выполняемые в их системах. Узел Software Restriction Policies в редакторе локальной политики безопасности служит интерфейсом управления для политик выполнения кода на компьютере, хотя возможны и политики, индивидуальные для пользователей; в последнем случае применяются доменные политики групп.

Узел Software Restriction Policies (Политики ограниченного использования программ) содержит несколько глобальных параметров.

Параметр **Enforcement** (Принудительный) определяет, как применяются политики ограничения — к библиотекам вроде DLL, только к пользователям или к пользователям и администраторам.

Параметр **Designated File Types** (Назначенные типы файлов) регистрирует расширения файлов, которые считаются исполняемыми.

Параметр **Trusted Publishers** (Доверенные издатели) контролирует, кто имеет право решать, каким издателям сертификатов можно доверять.

1. Управление вводом-выводом в ОС Windows: компоненты подсистемы ввода-вывода, диспетчер, типичная обработка ввода-вывода.

**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 WMIприложения 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 вводавывода, 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.

Приложения

Windows

-

сервисы

Служба

WMI

Диспетчер РпР

пользовательского

режима

Устанавливаемые

компоненты

Пользовательский режим

Режим ядра

Процедуры

WDM WMI

Диспетчер

РпР

Диспетчер

электропитания

Диспетчер

ввода

-

вывода

Подсистема

ввода

-

вывода

INF

-

и

CAT

-

файлы

,

реестр

Драйверы

HAL

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 САТ-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 вводавывода** (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 вводавывода 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.

API

пользовательского режима

Системные сервисы ввода

-

вывода

(

Ntxxx

)

Диспетчер ввода

-

вывода

(

loxxx

)

Драйверы устройств режима

ядра

HAL

-

функции доступа к

устройствам ввода

-

вывода

Функции

поддержки

драйверов

(

lo

,

Ex

,

Ke

,

Mm

,

Hal

,

FsRtl

и т

.

д

.)

Порты и регистры

ввода

-

вывода

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

1. Драйверы устройств в ОС Windows: типы драйверов, основные процедуры драйвера устройства. Объекты Windows: «драйвер» и «устройство».

**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 Windowsфункций 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.

**Основные процедуры драйвера устройства**

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.

Процедура

инициализации

ввода

-

вывода

Процедура обслуживания

прерываний

DPC

-

процедура

Инициализирующая

процедура

Процедура добавления

устройств

Подсистема

ввода

-

вывода

Процедуры

диспетчеризации

.

.

.

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 неблокировать без необходимости другие прерывания. DPC-процедура инициирует завершение текущей операции ввода-вывода и выполнение следующей операции вводавывода 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.

1. Обработка ввода-вывода: типы ввода-вывода (синхронный, асинхронный, быстрый, ввод-вывод в проецируемые файлы). Пакеты запроса ввода-вывода, блок стека IRP, управление буфером IRP.

**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 Windowsфункции 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 вводавывода 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.

**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.

**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 вводавывода 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.

1. Процесс обработки запросов синхронного ввода-вывода к одноуровневому драйверу режима ядра. Запрос ввода-вывода к многоуровневому драйверу. Порты завершения ввода-вывода.

**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:

Запрос на ввод-вывод передается через DLL подсистемы.

DLL подсистемы вызывает сервис NtWriteFile диспетчера ввода-вывода.

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

Драйвер передает данные из IRP на устройство и инициирует операцию вводавывода.

Драйвер уведомляет о завершении ввода-вывода, генерируя прерывание.

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

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.

Подсистема

окружения или

DLL

Пользовательский режим

Режим ядра

1

Передача запроса ввода

-

вывода через

DLL

подсистемы

*NtWriteFile*

*(*

*file*

*\_*

*handle*

*,*

*char*

*\_*

*butter*

*)*

2

7

Завершение обработки

IRP

с

сообщением об успехе или ошибке

Сервисы

Диспетчер ввода

-

вывода

Драйвер

устройства

IRP

6

Обработка прерывания

с сообщением об успехе

или ошибке

IRP

3

Создание

IRP

и его передача

драйверу устройства

4

Передача данных

,

указанных в

IRP

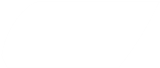
5

Выполнения ввода

-

вывода

и генерация прерывания



**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 вводавывода диспетчер ввода-вывода выделяет память для TRP в одном из ассоциативных списков IRP или в пуле неподкачиваемой памяти.

Подсистема окружения

или

DLL

1

Вызов сервиса ввода

-

вывода

Возврат кода незавершенной

операции ввода

-

вывода

7

Пользовательский режим

Режим ядра

Сервисы

Диспетчер

ввода

-

вывода

2

Диспетчер ввода

-

вывода

создает

IRP

,

заполняет его

первый блок стека и вызывает

драйвер файловой системы

Возврат кода незавершенной

операции ввода

-

вывода

6

Драйвер файловой

системы

Текущий

IRP

3

Драйвер файловой системы

заполняет второй блок стека

IRP

и вызывает драйвер диска

5

Возврат кода незавершенной

операции ввода

-

вывода

Текущий

IRP

Драйвер диска

4

Данные

IRP

посылаются на

устройство

(

либо

IRP

ставиться в очередь

)

,

и

проходит возврат управления

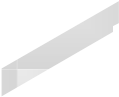
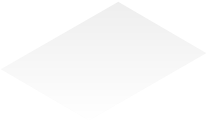


Рис. 16. Обработка асинхронного запроса к многоуровневым драйверам

**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.

Поступавший клиентский запрос

Порт завершения

Потоки

,

блокированные на порте завершения

Обработка данных

(

поток активен

)

Файловый ввод

-

вывод

(

поток блокируется

;

неактивен

)

Обработка данных

(

поток активен

)

1. Диспетчер Plug & Play. Загрузка, инициализация и установка драйвера.

**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.

**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 Windowsсервисов 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.

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-npoвайдера (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.

**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.

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

**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).

1. Диспетчер электропитания.

**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 х8б-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, неопределе нная |
| S5 (fully off) (полное отключение) | Ток подается на кнопку включения электропитания | Система загружается заново | Длительная, неопределе нная |

1. Управление внешней памятью в ОС Windows: терминология, драйверы дисков, объекты «устройство» для дисков.

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, 3,5дюймовая 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.

Приложение

|  |
| --- |
| Подсистема ввода-вывода |

Посылает запрос ввода-вывода файловой системе

|  |
| --- |
| Файловая система |

Накладывает файловую структуру на неформатированные тома

|  |
| --- |
| Снимок тома |

Управляет программными снимками

|  |
| --- |
| Диспетчер томов |

|  |
| --- |
| Диспетчер разделов |

Представляет тома (С:, D: ) пользователям; поддерживает базовые и динамические диски (RA|D)

Управляет разделами на дисках

|  |
| --- |
| Класс |

Управляет конкретным типом устройств, например дисками и ленточными накопителями

|  |  |  |
| --- | --- | --- |
| Порт |  | Минипорт |

Порт: управляет специфическим Минипорт: поставляется транспортным протоколом изготовителем, функционально связан

(Scsiport для SCSI, Storport для с соответствующим порт-драйвером,

RAID и FC и т.д.) управляет специфическими

|  |
| --- |
| Дисковая подсистема |

возможностями оборудования

Рис. 27. Стек драйверов устройств внешней памяти в Windows

**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 ХР и Windows Server 2003 — функцию IoReadPartitionTableEx. Драйвер класса дисков вызывает одну из этих функций для каждого диска, представленного минипортдрайвером 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 (0х7е000 и 0x7ff50c00) определяют начало и длину раздела, а последнее число — внутренний идентификатор, назначенный драйвером класса.

1. Файловые системы ОС Windows (обзор): NTFS, FAT, прочие.

### **Базовые понятия**

**Секторы** — аппаратно адресуемые блоки носителя. Размер секторов на жестких дисках в х86-системах почти всегда равен 512 байтам. Таким образом, если операционная система должна модифицировать 632-й байт диска, она записывает 512-байтовый блок данных во второй сектор диска.

**Форматы файловых систем** определяют принципы хранения данных на носителе и влияют на характеристики файловой системы. Например, файловая система, формат которой не допускает сопоставления прав доступа с файлами и каталогами, не поддерживает защиту. Формат файловой системы также может налагать ограничения на размеры файлов и емкости поддерживаемых устройств внешней памяти. Наконец, некоторые форматы файловых систем эффективно реализуют поддержку либо больших, либо малых файлов и дисков.

**Кластеры** — адресуемые блоки, используемые многими файловыми системами. Размер кластера всегда кратен размеру сектора (Рис. 1). Файловая система использует кластеры для более эффективного управления дисковым пространством: кластеры, размер которых превышает размер сектора, позволяют разбить диск на блоки меньшей длины — управлять такими блоками легче, чем секторами. Потенциальный недостаток кластеров большего размера — менее эффективное использование дискового пространства, или внутренняя фрагментация, которая возникает из-за того, что размеры файлов редко бывают кратны размеру кластера.

**Метаданные** — это данные, хранящиеся на томе и необходимые для поддержки управления файловой системой. Как правило, они недоступны приложениям. Метаданные включают, например, информацию, определяющую местонахождение файлов и каталогов на томе.

Кластер

(

2

сектора

)

Сектор

Рис. 1. Секторы и кластер на диске

Файловые системы

NTFS

FAT

12

,

FATl

6

и

FAT

32

UDF

CDFS

**NTFS**

**NTFS** — встроенная («родная») файловая система Windows. NTFS использует 64разрядные номера кластеров. Это позволяет NTFS адресовать тома размером до 16 экзабайт (16 миллиардов Гб). Однако Windows ограничивает размеры томов NTFS до значений, при которых возможна адресация 32-разрядными кластерами, т. е. до 128 Тб (с использованием кластеров по 64 Кб).

Таблица 3. Размеры кластеров на томах NTFS

|  |  |
| --- | --- |
| Размер тома | Размер кластера |
| 512 Мб и меньше | 512 байтов |
| 513-1024 Мб | 1 Кб |
| 1025-2048 Мб | 2 Кб |
| Более 2048 Мб | 4 Кб |

NTFS поддерживает ряд дополнительных возможностей — защиту файлов и каталогов, дисковые квоты, сжатие файлов, символьные ссылки на основе каталогов и шифрование. Одно из важнейших свойств NTFS — восстанавливаемость. При неожиданной остановке системы целостность метаданных тома FAT может быть утрачена, что вызовет повреждение структуры каталогов и значительного объема данных. NTFS ведет журнал изменений метаданных путем протоколирования транзакций, поэтому целостность структур файловой системы может быть восстановлена без потери информации о структуре файлов или каталогов.

**FAT12, FAT16 и FAT32**

Windows поддерживает файловую систему FAT по трем причинам:

* для возможности обновления операционной системы с прежних версий Windows до современных,
* для совместимости с другими операционными системами при многовариантной загрузке,
* как формат гибких дисков. Драйвер файловой системы FAT в Windows реализован в \Windows\System32\Drivers\ Fastfat.sys.

B название каждого формата FAT входит число, которое указывает разрядность, применяемую для идентификации кластеров на диске. 12-разрядный идентификатор кластеров в FAT12 ограничивает размер дискового раздела 212 (4096) кластерами. B Windows используются кластеры размером от 512 байтов до 8 Кб, так что размер тома FAT12 ограничен 32 Мб.

FAT16 — за счет 16-разрядных идентификаторов кластеров — может адресовать до 216 (65 536) кластеров. B Windows размер кластера FAT16 варьируется от 512 байтов до 64 Кб, поэтому размер FAT16-томa ограничен 4 Гб. Размер кластеров, используемых Windows, зависит от размера тома (Таблица 1). Если вы форматируете том размером менее 16 Мб для FAT с помощью команды format или оснастки Disk Management (Увправление дисками), Windows вместо FAT16 использует FAT12.

Таблица 1. Размеры кластеров в FAT16 в Windows по умолчанию

|  |  |
| --- | --- |
| Размер тома (Мб) | Размер кластера |
| 0-32 | 512 байтов |
| 33-64 | 1 Кб |
| 65-128 | 2 Кб |
| 129-256 | 4 Кб |
| 257-511 | 8 Кб |
| 512-1023 | 16 Кб |
| 1024-2047 | 32 Кб |
| 2048-4095 | 64 Кб |

FAT32 — более новая файловая система на основе формата FAT; она поддерживается Windows 95 OSR2, Windows 98 и Windows Millennium Edition. FAT32 использует 32разрядные идентификаторы кластеров, но при этом резервирует старшие 4 бита, так что эффективный размер идентификатора кластера составляет 28 бит. Поскольку максимальный размер кластеров FAT32 равен 32 Кб, теоретически FAT32 может работать с 8-терабайтными томами. Windows ограничивает размер новых томов FAT32 до 32 Гб, хотя поддерживает существующие тома FAT32 большего размера (созданные в других операционных системах). Большее число кластеров, поддерживаемое FAT32, позволяет ей управлять дисками более эффективно, чем FATl6. FAT32 может использовать 512-байтовые кластеры для томов размером до 128 Мб. Размеры кластеров на томах FAT32 по умолчанию показаны в Таблица 2. Таблица 2. Размер кластеров на томах FAT32 по умолчанию

|  |  |
| --- | --- |
| Размер раздела | Размер кластера |
| От 32 Мб до 7 Гб | 4 |
| 8-16 Гб | 8 |
| 16-32 Гб | 16 |
| 32 Гб | 32 |

1. Архитектура драйвера файловой системы.

## **Архитектура драйвера файловой системы**

Драйвер файловой системы (file system driver, FSD) управляет форматом файловой системы. Хотя FSD выполняются в режиме ядра, у них есть целый ряд особенностей по сравнению со стандартными драйверами режима ядра. Возможно, самой важной особенностью является то, что они должны регистрироваться у диспетчера вводавывода и более интенсивно взаимодействовать с ним. Кроме того, для большей производительности FSD обычно полагаются на сервисы диспетчера кэша. Таким образом, FSD используют более широкий набор функций, экспортируемых Ntoskrnl, чем стандартные драйверы. Если для создания стандартных драйверов режима ядра требуется Windows DDK, то для создания драйверов файловых систем понадобится Windows Installable File System (IFS) Kit. B Windows два типа драйверов файловых систем:

* локальные FSD, управляющие дисковыми томами, подключенными непосредственно к компьютеру;
* сетевые FSD, позволяющие обращаться к дисковым томам, подключенным к удаленным компьютерам.

### **Как работает файловая система**

Система и приложения могут обращаться к файлам двумя способами: напрямую (через функции ввода-вывода вроде ReadFile и WriteFile) и косвенно, путем чтения или записи части своего адресного пространства, где находится раздел проецируемого файла. Упрощенная схема на Рис. 9 иллюстрирует компоненты, участвующие в работе файловой системы, и способы их взаимодействия. Как видите, есть несколько путей вызова FSD:

* из пользовательского или системного потока, выполняющего явную операцию файлового ввода-вывода;
* из подсистем записи модифицированных и спроецированных страниц, принадлежащих диспетчеру памяти;
* неявно из подсистемы отложенной записи, принадлежащей диспетчеру кэша;  неявно из потока опережающего чтения, принадлежащего диспетчеру кэша;  из обработчика ошибок страниц, принадлежащего диспетчеру памяти.

1. Цели разработки и особенности NTFS. Драйвер файловой системы NTFS.

### **Требования к файловой системе класса «high end»**

C самого начала разработка NTFS велась с учетом требований, предъявляемых к файловой системе корпоративного класса. Чтобы свести к минимуму потери данных в случае неожиданного выхода системы из строя или ее краха, файловая система должна гарантировать целостность своих метаданных. Для защиты конфиденциальных данных от несанкционированного доступа файловая система должна быть построена на интегрированной модели защиты. Наконец, она должна поддерживать защиту пользовательских данных за счет программной избыточности данных в качестве недорогой альтернативы аппаратным решениям. Здесь вы узнаете, как эти возможности реализованы в NTFS.

**Восстанавливаемость**

B соответствии с требованиями к надежности хранения данных и доступа к ним NTFS обеспечивает восстановление файловой системы на основе концепции атомарной транзакции (atomic transaction). **Атомарные транзакции** — это метод обработки изменений в базе данных, при котором сбои в работе системы не нарушают корректности или целостности базы данных. Суть атомарных транзакций заключается в том, что некоторые операции над базой данных, называемые **транзакциями**, выполняются по принципу «все или ничего». Отдельные изменения на диске, составляющие транзакцию, выполняются **атомарно**: в ходе транзакции на диск должны быть внесены все требуемые изменения. Если транзакция прервана аварией системы, часть изменений, уже внесенных к этому моменту, нужно отменить. Такая отмена называется **откатом** (roll back). После отката база данных возвращается в исходное согласованное состояние, в котором она была до начала транзакции.

**Защита**

Защита в NTFS построена на модели объектов Windows. Файлы и каталоги защищены от доступа пользователей, не имеющих соответствующих прав. Открытый файл реализуется в виде объекта «файл» с дескриптором защиты, хранящимся на диске как часть файла. Прежде чем процесс сможет открыть описатель какого-либо объекта, в том числе объекта «файл», система защиты Windows должна убедиться, что у этого процесса есть соответствующие полномочия. Дескриптор защиты в сочетании с требованием регистрации пользователя при входе в систему гарантирует, что ни один процесс не получит доступа к файлу без разрешения системного администратора или владельца файла.

**Избыточность данных и отказоустойчивость**

Восстанавливаемость NTFS действительно гарантирует, что файловая система тома останется доступной, но не дает гарантии полного восстановления пользовательских файлов. Последнее возможно за счет поддержки избыточности данных.

Избыточность данных для пользовательских файлов реализуется через многоуровневую модель драйверов Windows, которая поддерживает отказоустойчивые диски. При записи данных на диск NTFS взаимодействует с диспетчером томов, а тот — с драйвером жесткого диска. Диспетчер томов может зеркалировать, или дублировать, данные одного диска на другом и таким образом позволяет при необходимости использовать данные с избыточной копии. Поддержка таких функций обычно называется RAID уровня 1. Диспетчеры томов также могут записывать данные в чередующиеся области (stripes) на три и более дисков, используя один диск для хранения информации о четности. Если данные на одном диске потеряны или стали недоступными, драйвер может реконструировать содержимое диска с помощью логической операции XOR. Такая поддержка называется RAID уровня 5.

**Драйвер файловой системы NTFS**

NTFS и другие файловые системы представляют собой загружаемые драйверы устройств режима ядра. Они неявно вызываются приложениями, использующими Windows или другие API ввода-вывода (например, POSIX). Как показано на Рис. 11, подсистемы окружения вызывают системные сервисы, которые в свою очередь находят соответствующие загруженные драйверы и вызывают их.

**Пользовательский режим**

**Режим ядра**

Подсистема

окружения или

DLL

Системные сервисы

Windows

Диспетчер

объектов

Монитор

состояния

защиты

**. . .**

Механизм

LPC

Диспетчер

виртуаль

-

ной памяти

Ядро

Исполнительная

система

Windows

Драйвер

NTFS

Диспетчер

томов

Драйвер

диска

**Диспетчер ввода**

**-**

**вывода**

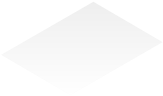


Рис. 11. Компоненты подсистемы ввода-вывода в Windows

Драйверы передают друг другу запросы ввода-вывода, вызывая диспетчер вводавывода исполнительной системы. Использование диспетчера ввода-вывода в качестве промежуточного звена обеспечивает независимость каждого драйвера, что позволяет загружать и выгружать его без последствий для других драйверов. Кроме того, драйвер NTFS взаимодействует с тремя другими компонентами исполнительной системы (Рис. 12), тесно связанными с файловыми системами.

**Сервис файла**

**журнала**

**Диспетчер кэша**

**Диспетчер**

**виртуальной памяти**

Драйвер

NTFS

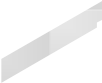
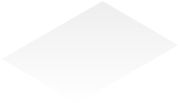
Диспетчер

томов

**Диспетчер ввода**

**-**

**вывода**



Драйвер

диска

Сброс файла

журнала на

диск

Загрузка

данных с

диска в

память

Запись в

кэш

Чтение

/

запись файла

Регистрация транзакции

Обращение к

спроецированному файлу или

сброс кэша

Чтение

/

запись

зеркального или

чередующего тома

Чтение

/

запись

диска

Рис. 12. NTFS и связные с ней компоненты

**Сервис файла журнала** (log file service, LFS) является частью NTFS и предоставляет функции для поддержки журнала изменений на диске. Файл журнала LFS используется при восстановлении тома NTFS в случае аварии системы (подробнее о LFS см. раздел «Сервис файла журнала» далее в этой главе).

**Диспетчер кэша** — компонент исполнительной системы, предоставляющий общесистемные сервисы кэширования для NTFS и драйверов других файловых систем, в том числе сетевых (т. е. для серверов и редиректоров). Все файловые системы, реализованные в Windows, получают доступ к кэшированным файлам, проецируя их на системное адресное пространство, а затем считывая соответствующие участки виртуальной памяти. C этой целью диспетчер кэша предоставляет диспетчеру памяти специализированный интерфейс файловых систем. Когда программа пытается обратиться к какой-либо части файла, не загруженной в кэш (промах кэша), диспетчер памяти вызывает NTFS для обращения к драйверу диска и загрузки нужных файловых данных. Диспетчер кэша оптимизирует дисковый ввод-вывод, вызывая диспетчер памяти (для сброса на диск содержимого кэша в фоновом режиме) из потоков подсистемы отложенной записи.

1. Структура NTFS на диске.

### **Тома**

Структура NTFS начинается с тома. Том (volume) соответствует логическому разделу на диске и создается при форматировании диска или его части под NTFS. Оснастка Disk Management (Управление дисками) консоли MMC также позволяет создать том RAID, охватывающий несколько дисков.

Ha диске может быть один или несколько томов. NTFS обрабатывает каждый том независимо от других. Три примера конфигурации 150-мегабайтного жесткого диска показаны на Рис. 13.

C:

**NTFS**

**-**

**том**

C

:

(

150

Мб

)

D

:

**NTFS**

**-**

**том**

**2**

**NTFS**

**-**

**том**

**1**

C

:

(

60

Мб

)

D

:

(

90

Мб

)

**NTFS**

**-**

**том**

**FAT**

**-**

**том**

(75 Мб)

(75 Мб)

Рис. 13. Примеры конфигурации диска

Том состоит из набора файлов и свободного пространства, оставшегося в данном разделе диска. B FAT том также содержит области, специально отформатированные для использования файловой системой. Ho в томе NTFS все данные файловой системы вроде битовых карт, каталогов и даже начального загрузочного кода хранятся как обычные файлы.

**Кластеры**

**Размер кластера** на томе NTFS, или **кластерный множитель** (cluster factor), устанавливается при форматировании тома командой format или в оснастке Disk Management (Управление дисками). Размер кластера по умолчанию определяется размером тома, но всегда содержит целое число физических секторов с дискретностью N2 (т. е. 1 сектор, 2 сектора, 4 сектора, 8 секторов и т. д.). Кластерный множитель выражается числом байтов в кластере, например 512 байт, 1 Кб или 2 Кб.

Внутренне NTFS работает только с кластерами. NTFS использует кластер как единицу выделения пространства для поддержания независимости от размера физического сектора. Это позволяет NTFS эффективно работать с очень большими дисками, используя кластеры большего размера, и поддерживать нестандартные диски с размером секторов, отличным от 512 байтов. Применение больших кластеров на больших томах уменьшает фрагментацию и ускоряет выделение свободного пространства за счет небольшого проигрыша в эффективности использования дискового пространства. Команда format или оснастка Disk Management выбирает кластерный множитель в зависимости от размера тома, но это значение можно изменить.

**Главная таблица файлов**

B NTFS все данные, хранящиеся на томе, содержатся в файлах. Это относится и к структурам данных, используемым для поиска и выборки файлов, к начальному загрузочному коду и битовой карте, в которой регистрируется состояние пространства всего тома (метаданные NTFS). Хранение всех видов данных в файлах позволяет файловой системе легко находить и поддерживать данные, а каждый файл может быть защищен дескриптором защиты. Кроме того, при появлении плохих секторов на диске, NTFS может переместить файлы метаданных.

**Имена файлов**

NTFS и FAT допускают длину каждого имени файла в пути до 255 символов. Эти имена могут содержать Unicode-символы, точки и пробелы. Однако длина имен файлов в FAT, встроенной в MS-DOS, ограничена 8 символами (не-Unicode), за которыми следует расширение из трех символов, отделенное точкой. Рис. 15 иллюстрирует различные пространства имен файлов, поддерживаемые Windows, и показывает, как они перекрываются.

Подсистема

POSIX

Подсистема

Windows

Клиенты

MS

-

DOS

и

Windows

“TrailingDots

.

“

“SameNameDifferentCase”

“samenamedifferentcase”

“TrailingSpacees ”

“LongFileName”

“UnicodeName

.

Ф

∆∏

Λ

”

“File Name With

.

Dots”

“File Name

2

.

With

.

Dots”

“Name With Embedded Spaces”

“ BeginningDot”

“EIGHTCHR

.

123

”

“CASEBLND

.

TYP”

Рис. 15. Пространства имен файлов, поддерживаемые Windows

1. Поддержка восстановления в NTFS. Механизм EFS.

Поддержка восстановления в NTFS гарантирует, что в случае отказа электропитания или аварии системы ни одна операция файловой системы (транзакция) не останется незавершенной; при этом структура дискового тома будет сохранена. NTFS включает утилиту Chkdsk, которая позволяет устранять последствия катастрофических повреждений диска, вызванных аппаратными ошибками ввода-вывода (например, из-за аварийных секторов на диске, электрических аномалий или сбоев в работе диска) либо ошибками в программном обеспечении. Наличие средств восстановления NTFS уменьшает потребность в использовании Chkdsk.

**Восстановление**

NTFS автоматически выполняет восстановление диска при первом обращении к нему какой-либо программы после загрузки системы. (Если восстановление не требуется, весь процесс тривиален.) При восстановлении используются две таблицы, которые NTFS поддерживает в памяти.

* **Таблица транзакций** (transaction table) — предназначена для отслеживания начатых, но еще не зафиксированных транзакций. B процессе восстановления результаты подопераций этих транзакций должны быть удалены с диска.
* **Таблица измененных страниц** (dirty page table) — в нее записывается информация о том, какие страницы кэша содержат изменения структуры файловой системы, еще не записанные на диск. Эти данные в процессе восстановления должны быть сброшены на диск.

**Механизм EFS**

**EFS** (Encrypting File System) использует средства поддержки шифрования. При первом шифровании файла EFS назначает учетной записи пользователя, шифрующего этот файл, криптографическую пару — закрытый и открытый ключи. Пользователи могут шифровать файлы с помощью Windows Explorer; для этого нужно открыть диалоговое окно Properties (Свойства) применительно к нужному файлу, щелкнуть кнопку Advanced (Другие) и установить флажок Encrypt Contents To Secure Data (Шифровать содержимое для защиты данных), как показано на Рис. 29. Пользователи также могут шифровать файлы с помощью утилиты командной строки cipher. Windows автоматически шифрует файлы в каталогах, помеченных зашифрованными. При шифровании файла EFS генерирует случайное число, называемое шифровальным ключом файла (file encryption key, FEK). EFS использует FEK для шифрования содержимого файла по более стойкому варианту DES (Data Encryption Standard) — DESX (в Windows 2000), а также no DESX, 3DES (Triple-DES) или AES (Advanced Encryption Standard) в Windows XP (Service Pack 1 и выше) и Windows Server 2003. EFS сохраняет FEK вместе с самим файлом, но FEK шифруется по алгоритму RSA-шифрования на основе открытого ключа. После выполнения EFS этих действий файл защищен: другие пользователи не смогут расшифровать данные без расшифрованного FEK файла, a FEK они не смогут расшифровать без закрытого ключа пользователя — владельца файла



1. Функции синхронизации процессов при работе с Windows.

***Функции синхронизации***

Функции синхронизации делятся на две основные категории – это функции, ожидающие единственного объекта и функции, ожидающие одного из нескольких объектов

**function** WaitForSingleObject(  
 hHandle: THandle; // идентификатор объекта  
 dwMilliseconds: DWORD // период ожидания  
): DWORD; **stdcall**;

Функция ожидает перехода объекта hHandle в сигнальное состояние в течении dwMilliseconds миллисекунд. Если в качестве параметра dwMilliseconds передать значение INFINITE, функция будет ждать в течение неограниченного времени. Если dwMilliseconds равен 0, то функция проверяет состояние объекта и немедленно возвращает управление.

Функция возвращает одно из следующих значений:

|  |  |
| --- | --- |
| WAIT\_ABANDONED | Поток, владевший объектом, завершился, не переведя объект в сигнальное состояние. |
| WAIT\_OBJECT\_0 | Объект перешел в сигнальное состояние |
| WAIT\_TIMEOUT | Истек срок ожидания. Обычно в этом случае генерируется ошибка, либо функция вызывается в цикле до получения другого результата |
| WAIT\_FAILED | Произошла ошибка, например неверное значение hHandle. Более подробную информацию можно получить, вызвав GetLastError |

**Функции, ожидающие нескольких объектов**

Иногда требуется задержать выполнение потока до срабатывания одного или всех сразу из группы объектов. Для решения подобной задачи служат следующие функции:

**type**  
 TWOHandleArray = **array**[0..MAXIMUM\_WAIT\_OBJECTS - 1] **of** THandle;  
 PWOHandleArray = ^TWOHandleArray;  
  
**function** WaitForMultipleObjects(  
 nCount: DWORD; // Задает количество объектов  
 lpHandles: PWOHandleArray; // Адрес массива объектов  
 bWaitAll: BOOL; // Задает, требуется ожидание всех  
 // объектов или любого  
 dwMilliseconds: DWORD // Период ожидания  
): DWORD; **stdcall**;

Функция возвращает одно из следующих значений:

|  |  |
| --- | --- |
| Число в диапазоне от WAIT\_OBJECT\_0 до WAIT\_OBJECT\_0 + nCount – 1 | Если bWaitAll равно TRUE, то это число означает, что все объекты перешли в сигнальное состояние.  Если FALSE – то, вычтя из возвращенного значения WAIT\_OBJECT\_0, мы получим индекс объекта в массиве lpHandles. |
| Число в диапазоне от WAIT\_ABANDONED\_0 до WAIT\_ABANDONED\_0 + nCount – 1 | Если bWaitAll равно TRUE – это означает, что все перешли в сигнальное состояние, но хотя бы один из владевших ими потоков завершился, не сделав объект сигнальным.  Если FALSE – то, вычтя из возвращенного значения WAIT\_ABANDONED\_0, мы получим индекс объекта в массиве lpHandles, поток, владевший которым, завершился, не сделав его сигнальным. |
| WAIT\_TIMEOUT | Истек период ожидания |
| WAIT\_FAILED | Произошла ошибка |

1. Объекты синхронизации Windows (основные).

Объектами синхронизации называются объекты Windows, идентификаторы которых могут использоваться в функциях синхронизации. Они делятся на две группы – объекты, использующиеся только для синхронизации и объекты, которые используются в других целях, но могут вызывать срабатывание функций ожидания. К первой группе относятся:

**Event (событие)**

Позволяет известить один или несколько ожидающих потоков о наступлении события. Event бывает

|  |  |
| --- | --- |
| Отключаемый вручную | Будучи установленным в сигнальное состояние, остается в нем до тех пор, пока не будет переключен явным вызовом функции ResetEvent |
| Автоматически отключаемый | Автоматически переключается в несигнальное состояние операционной системой, когда один из ожидающих его потоков завершается. |

Для создания объекта используется функция:

**function** CreateEvent(  
 lpEventAttributes: PSecurityAttributes; // Адрес структуры  
 // TSecurityAttributes  
 bManualReset, // Задает, будет Event переключаемым   
 // вручную (TRUE) или автоматически (FALSE)  
 bInitialState: BOOL; // Задает начальное состояние. Если TRUE -   
 // объект в сигнальном состоянии  
 lpName: PChar // Имя или **NIL**, если имя не требуется  
): THandle; **stdcall**; // Возвращает идентификатор созданного  
 // объекта

Если известно, что Event уже создан, для получения доступа к нему можно вместо CreateEvent воспользоваться функцией:

**function** OpenEvent(  
 dwDesiredAccess: DWORD; // Задает права доступа к объекту  
 bInheritHandle: BOOL; // Задает, может ли объект наследоваться   
 // дочерними процессами  
 lpName: PChar // Имя объекта  
): THandle; **stdcall**;

Функция возвращает идентификатор объекта, либо 0, в случае ошибки. Параметр dwDesiredAccess может принимать одно из следующих значений:

|  |  |
| --- | --- |
| EVENT\_ALL\_ACCESS | Приложение получает полный доступ к объекту |
| EVENT\_MODIFY\_STATE | Приложение может изменять состояние объекта функциями SetEvent и ResetEvent |
| SYNCHRONIZE | Только для Windows NT – приложение может использовать объект только в функциях ожидания |

После получения идентификатора можно приступать к его использованию. Для этого имеются следующие функции:

**function** SetEvent(hEvent: THandle): BOOL; **stdcall**;

Устанавливает объект в сигнальное состояние

**function** ResetEvent(hEvent: THandle): BOOL; **stdcall**;

Сбрасывает объект, устанавливая его в несигнальное состояние

**function** PulseEvent(hEvent: THandle): BOOL; **stdcall**

**Mutex (Mutually Exclusive)**

Мутекс – это объект синхронизации, который находится в сигнальном состоянии только тогда, когда он не принадлежит ни одному из процессов. Как только хотя бы один процесс запрашивает владение мутексом, он переходит в несигнальное состояние и остается в нем до тех пор, пока не будет освобожден владельцем. Такое поведение позволяет использовать мутексы для синхронизации совместного доступа нескольких процессов к разделяемому ресурсу. Для создания мутекса используется функция:

**function** CreateMutex(  
 lpMutexAttributes: PSecurityAttributes; // Адрес структуры   
 // TSecurityAttributes  
 bInitialOwner: BOOL; // Задает, будет ли процесс владеть  
 // мутексом сразу после создания  
 lpName: PChar // Имя мутекса  
): THandle; **stdcall**;

Если мутекс уже существует, приложение может получить его идентификатор функцией

**function** OpenMutex(  
 dwDesiredAccess: DWORD; // Задает права доступа к объекту  
 bInheritHandle: BOOL; // Задает, может ли объект наследоваться   
 // дочерними процессами  
 lpName: PChar // Имя объекта  
): THandle; **stdcall**;

Параметр dwDesiredAccess может принимать одно из следующих значений

|  |  |
| --- | --- |
| MUTEX\_ALL\_ACCESS | Приложение получает полный доступ к объекту |
| SYNCHRONIZE | Только для Windows NT – приложение может использовать объект только в функциях ожидания и функции ReleaseMutex |

**Semaphore (семафор)**

Семафор представляет собой счетчик, содержащий целое число в диапазоне от 0 до заданной при его создании максимальной величины. Счетчик уменьшается каждый раз, когда поток успешно завершает функцию ожидания, использующую семафор и увеличивается вызовом функции ReleaseSemaphore. При достижении семафором значения 0 он переходит в несигнальное состояние, при любых других значениях счетчика – его состояние сигнальное. Такое поведение позволяет использовать семафор в качестве ограничителя доступа к ресурсу, поддерживающему заранее заданное количество подключений.

Для создания семафора служит функция:

**function** CreateSemaphore(  
 lpSemaphoreAttributes: PSecurityAttributes; // Адрес структуры   
 // TSecurityAttributes  
 lInitialCount, // Начальное значение счетчика  
 lMaximumCount: Longint; // Максимальное значение счетчика  
 lpName: PChar // Имя объекта  
): THandle; **stdcall**;

Идентификатор ранее созданного семафора может быть, также, получен функцией:

**function** OpenSemaphore(  
 dwDesiredAccess: DWORD; // Задает права доступа к объекту  
 bInheritHandle: BOOL; // Задает, может ли объект наследоваться   
 // дочерними процессами  
 lpName: PChar // Имя объекта  
): THandle; **stdcall**;

Параметр dwDesiredAccess может принимать одно из следующих значений:

|  |  |
| --- | --- |
| SEMAPHORE\_ALL\_ACCESS | Поток получает все права на семафор |
| SEMAPHORE\_MODIFY\_STATE | Поток может увеличивать счетчик семафора функцией ReleaseSemaphore |
| SYNCHRONIZE | Только Windows NT – поток может использовать семафор в функциях ожидания |

Для увеличения счетчика семафора используется функция:

**function** ReleaseSemaphore(  
 hSemaphore: THandle; // Идентификатор семафора  
 lReleaseCount: Longint; // Счетчик будет увеличен на эту величину  
 lpPreviousCount: Pointer // Адрес 32-битной переменной,   
 // принимающей предыдущее значение  
 // счетчика  
): BOOL; **stdcall**;

**Waitable timer (таймер ожидания)**

Таймер ожидания отсутствует в Windows 95 и для его использования необходима Windows 98 или Windows NT 4.0 и выше.

Таймер ожидания переходит в сигнальное состояние по завершении заданного интервала времени. Для его создания используется функция:

**function** CreateWaitableTimer(  
 lpTimerAttributes: PSecurityAttributes; // Адрес структуры   
 // TSecurityAttributes  
 bManualReset: BOOL; // Задает, будет ли таймер переходить в   
 // сигнальное состояние по завершении функции  
 // ожидания  
 lpTimerName: PChar // Имя объекта  
): THandle; **stdcall**;

Идентификатор уже существующего таймера можно получить функцией:

**function** OpenWaitableTimer(  
 dwDesiredAccess: DWORD; // Задает права доступа к объекту  
 bInheritHandle: BOOL; // Задает, может ли объект наследоваться   
 // дочерними процессами  
 lpTimerName: PChar // Имя объекта  
): THandle; **stdcall**;

Параметр dwDesiredAccess может принимать следующие значения:

|  |  |
| --- | --- |
| TIMER\_ALL\_ACCESS | Разрешает полный доступ к объекту |
| TIMER\_MODIFY\_STATE | Разрешает изменять состояние таймера функциями SetWaitableTimer и CancelWaitableTimer |
| SYNCHRONIZE | Только Windows NT – разрешает использовать таймер в функциях ожидания |

1. Дополнительные объекты и механизмы синхронизации Windows

**Дополнительные объекты синхронизации**

Некоторые объекты Win32 API не предназначены исключительно для целей синхронизации, однако могут использоваться с функциями синхронизации. Такими объектами являются:

Сообщение об изменении папки (change notification)

Windows позволяет организовать слежение за изменениями объектов файловой системы. Для этого служит функция

**function** FindFirstChangeNotification(  
 lpPathName: PChar; // Путь к папке, изменения в которой нас   
 // интересуют  
 bWatchSubtree: BOOL; // Задает необходимость слежения за   
 // изменениями во вложенных папках  
 dwNotifyFilter: DWORD // Фильтр событий  
): THandle; **stdcall**;

Параметр dwNotifyFilter – это битовая маска из одного или нескольких следующих значений:

|  |  |
| --- | --- |
| FILE\_NOTIFY\_CHANGE\_FILE\_NAME | |
|  | Слежение ведется за любым изменением имени файла, в т.ч. созданием и удалением файлов |
| FILE\_NOTIFY\_CHANGE\_DIR\_NAME | |
|  | Слежение ведется за любым изменением имени папки, в т.ч. созданием и удалением папок |
| FILE\_NOTIFY\_CHANGE\_ATTRIBUTES | |
|  | Слежение ведется за любым изменением аттрибутов |
| FILE\_NOTIFY\_CHANGE\_SIZE | |
|  | Слежение ведется за изменением размера файлов. Изменение размера происходит при записи в файл. Функция ожидания срабатывает только после успешного сброса дискового кэша |
| FILE\_NOTIFY\_CHANGE\_LAST\_WRITE | |
|  | Слежение за изменением времени последней записи в файл, т.е., фактически, за любой записью в файл. Функция ожидания срабатывает только после успешного сброса дискового кэша. |
| FILE\_NOTIFY\_CHANGE\_SECURITY | |
|  | Слежение за любыми изменениями дескрипторов защиты |

Идентификатор, возвращенный этой функцией, может использоваться в любой функции ожидания. Он переходит в сигнальное состояние, когда в папке происходят запрошенные для слежения изменения. Продолжить слежение можно, используя функцию:

**function** FindNextChangeNotification(  
 hChangeHandle: THandle  
): BOOL; **stdcall**;

По завершении работы, идентификатор должен быть закрыт при помощи функции:

**function** FindCloseChangeNotification(  
 hChangeHandle: THandle  
): BOOL; **stdcall**;

**Задание (Job)**

Job – это новый механизм Windows 2000, позволяющий объединить группу процессов в одно задание и манипулировать ими одновременно. Идентификатор задания находится в сигнальном состоянии, если все процессы, ассоциированные с ним завершились по причине истечения лимита времени на выполнение задания.

**Процесс (Process)**

Идентификатор процесса, полученный при помощи функции CreateProcess, переходит в сигнальное состояние по завершении процесса. Это позволяет организовать ожидание завершения процесса, например, при запуске из приложения внешней программы.

**var**  
 PI: TProcessInformation;  
 SI: TStartupInfo;  
  
...  
  
 FillChar(SI, SizeOf(SI), 0);  
 SI.cb := SizeOf(SI);  
 Win32Check(CreateProcess(**NIL**, 'COMMAND.COM', **NIL**,  
 **NIL**, FALSE, 0, **NIL**, **NIL**, SI, PI));  
 // Задерживаем исполнение программы до завершения процесса  
 WaitForSingleObject(PI.hProcess, INFINITE);  
 CloseHandle(PI.hProcess);  
 CloseHandle(PI.hThread);

Следует понимать, что в этом случае вызывающий процесс будет заморожен полностью и не сможет обрабатывать сообщения. Поэтому, если дочерний процесс может выполняться в течение длительного времени, лучше использовать более корректный вариант ожидания, описанный в разделе, посвященном функции MsgWaitForMultipleObjects.

**Поток (thread)**

Идентификатор потока находится в несигнальном состоянии до тех пор, пока поток выполняется. По его завершении идентификатор переходит в сигнальное состояние. Это позволяет легко узнать, завершился ли поток, либо при помощи функции, ожидающей нескольких объектов, организовать ожидание завершения одного из, либо всех интересующих потоков.

**Дополнительные механизмы синхронизации**

**Критические секции**

Критические секции – это механизм, предназначенный для синхронизации потоков внутри одного процесса. Как и мутекс, критическая секция может в один момент времени принадлежать только одному потоку, однако, она предоставляет более быстрый и эффективный механизм, чем мутексы. Перед использованием критической секции необходимо инициализировать её функцией:

**procedure** InitializeCriticalSection(  
 **var** lpCriticalSection: TRTLCriticalSection  
); **stdcall**;

После создания объекта поток, перед доступом к защищаемому ресурсу должен вызвать функцию:

**procedure** EnterCriticalSection(  
 **var** lpCriticalSection: TRTLCriticalSection  
); **stdcall**;

Если в этот момент ни один из потоков в процессе не владеет объектом, то поток становится владельцем критической секции и продолжает выполнение. Если секция уже захвачена другим потоком то выполнение потока, вызвавшего функцию приостанавливается до её освобождения.

Поток, владеющий критической секцией, может повторно вызывать функцию EnterCriticalSection без блокирования своего исполнения. По завершению работы с защищаемым ресурсом поток должен вызвать функцию:

**procedure** LeaveCriticalSection(  
 **var** lpCriticalSection: TRTLCriticalSection  
); **stdcall**;

Эта функция освобождает объект независимо от количества предыдущих вызовов потоком функции EnterCriticalSection. Если имеются другие потоки, ожидающие освобождения секции, один из них становится её владельцем и продолжает исполнение. Если поток завершился, не освободив критическую секцию, её состояние становится неопределенным, что может вызвать блокировку работы программы.

Имеется возможность попытаться захватить объект без замораживания потока. Для этого служит функция:

**function** TryEnterCriticalSection(  
 **var** lpCriticalSection: TRTLCriticalSection  
): BOOL; **stdcall**;

Она проверяет, захвачена секция ли в момент её вызова. Если да – функция возвращает FALSE, в противном случае – захватывает секцию и возвращает TRUE.

По завершении работы с критической секцией, она должна быть уничтожена вызовом функции:

**procedure** DeleteCriticalSection(  
 **var** lpCriticalSection: TRTLCriticalSection  
); **stdcall**;

1. Реализация прерываний, системных вызовов и исключений в ОС Windows

***Системные вызовы (system calls)*** - механизм, позволяющий пользовательским программам обращаться к услугам ядра ОС, то есть это *интерфейс* между операционной системой и пользовательской программой. Концептуально *системный вызов* похож на обычный вызов подпрограммы. Основное отличие состоит в том, что при системном вызове выполнение программы осуществляется в привилегированном режиме или режиме ядра. Поэтому системные вызовы иногда еще называют *программными прерываниями*, в отличие от аппаратных прерываний, которые чаще называют просто прерываниями. В большинстве операционных систем *системный вызов* является результатом выполнения команды *программного прерывания* (INT). Таким образом, *системный вызов* — это *синхронное* событие.

***Прерывание (hardware interrupt)*** — это событие, генерируемое внешним (по отношению к процессору) устройством. Посредством аппаратных прерываний аппаратура либо информирует центральный *процессор* о том, что произошло событие, требующее немедленной реакции (например, *пользователь* нажал клавишу), либо сообщает о завершении *операции* ввода вывода (например, закончено *чтение данных* с диска в основную *память*). Каждый тип аппаратных прерываний имеет собственный номер, однозначно определяющий источник прерывания. *Аппаратное прерывание* — это *асинхронное* событие, то есть оно возникает вне зависимости от того, какой код исполняется процессором в данный момент. Обработка аппаратного прерывания не должна учитывать, какой процесс или *поток* является текущим.

***Исключительная ситуация (exception)*** - событие, возникающее в результате попытки выполнения программой команды, которая по каким-то причинам не может быть выполнена до конца. Примерами таких команд могут быть попытки доступа к ресурсу при отсутствии достаточных привилегий или обращение к отсутствующей странице памяти. Исключительные ситуации, как и системные вызовы, являются синхронными событиями, возникающими в контексте текущей задачи. Исключительные ситуации можно разделить на исправимые и неисправимые. К исправимым относятся такие исключительные ситуации, как отсутствие нужной информации в оперативной памяти. После устранения причины исправимой исключительной ситуации *программа* может выполняться дальше. Возникновение в процессе работы операционной системы исправимых исключительных ситуаций считается нормальным явлением. Неисправимые исключительные ситуации чаще всего возникают в результате ошибок в программах (например, *деление* на ноль). Обычно в таких случаях *операционная система* реагирует завершением программы, вызвавшей исключительную ситуацию.

**Реализация прерываний, системных вызовов и исключений в ОС Windows**

Рассмотрим реализацию основных механизмов операционной системы в ОС *Windows*. Следует отметить, что терминология корпорации Microsoft несколько отличается от общепринятой. Например, системные вызовы называются *системными сервисами*, а под программным прерыванием (см. прерывания *DPC* и *APC*) понимается выполнение специфичных функций ядра, требующих прерывания работы текущего процесса.

**Ловушки**

Общим для реализации рассматриваемых основных механизмов является *необходимость сохранения состояния текущего потока с его последующим восстановлением*. Для этого в ОС Windows используется механизм ловушек (trap). В случае возникновения требующего обработки события (прерывания, исключения или вызова *системного сервиса*) процессор переходит в привилегированный режим и передает управление обработчику ловушек, входящему в состав ядра. Обработчик ловушек создает в стеке ядра (о стеке ядра см. ["Реализация процессов и потоков"](https://www.intuit.ru/studies/courses/1089/217/lecture/5593)) прерываемого потока фрейм ловушки, содержащий часть контекста потока для последующего восстановления его состояния, и в свою очередь передает управление определенной части ОС, отвечающей за первичную обработку произошедшего события.

В типичном случае сохраняются и впоследствии восстанавливаются:

* программный счетчик;
* регистр состояния процессора;
* содержимое остальных регистров процессора;
* указатели на стек ядра и пользовательский стек;
* указатели на адресное пространство, в котором выполняется поток (каталог таблиц страниц процесса).

Эта информация специфицирована в структуре CONTEXT (файл winnt.h), и может быть получена пользователем с помощью функции GetThreadContext.

1. Организация сетевого взаимодействия (Windows): Сокеты. Каналы.

**Понятие сокета**

**Сокет** – это объект ОС, через который можно передавать и принимать данные от процессов независимо от того, выполняется она на одном или разных компьютерных сетях. Сокет, также как и файл, характеризуется номером дескриптора и к нему применены файловые операции. Но в отличие от файла сокет существует лишь до момента, пока на него ссылается хотя бы один из процессов. Сокет передается по наследству и с ним могут быть связаны один или несколько процессов. Сокеты, имеющие одинаковые схемы адресации и семейство протоколов, группируются в домены.

**Поточный сокет** обеспечивает двухсторонний, последовательный, надежный, и недублированный поток данных без определенных границ. Перед началом обмена данными через такой сокет процессы должны установить канал связи. Тип сокета - SOCK\_STREAM, в домене Интернета использует протокол TCP.

**Датаграммный сокет** не гарантирует надежную передачу и получение данных в той последовательности, в какой они посылались, но этот способ передачи более быстрый, поскольку для него не требуется сложные установочные операции. Этот тип сокетов не требует предварительной установки соединения, адреса отправителя и получателя передаются с каждым сообщением. Тип сокета - SOCK\_DGRAM, в домене Интернета использует протокол UDP.

**Сокет последовательных пакетов**: можно организовать передачу сообщений фиксированный длинны с предварительным установлением соединением. Тип сокета - SOCK\_SEQPACKET. Для этого типа сокета не существует специального протокола.

**Простой сокет** - обеспечивает доступ к основным протоколам связи. Они используется, когда нужно создать новый протокол связи или получить доступ к параметрам существующего протокола.

**Открытие сокета.**

Открытие сокета - это операция, выполняющая построение дескриптора сокета. В этом дескрипторе хранится информация о типе coкета, его текущем состоянии и используемом протоколе

**Понятие канала**

**Программный канал** - средство коммуникации процессов, которое можно рассматривать как программно реализованную линию связи для передачи данных между двумя или более процессами. Каналы реализованы в виде снабженных специальным механизмом управления буферов ввода-вывода, которые трактуются как псевдо-файлы. Формальное сходство с файлами позволяет использовать одинаковый аппарат для доступа к данным в канале и файле. Такой подход соответствует базовой концепции унификации ввода-вывода в OS UNIX. При этом каналы имеют специфические особенности, которые позволяют исключить недостатки организации межпроцессного обмена через файлы. Основные особенности каналов перечислены ниже.

Каналы принято использовать следующим образом. Один процесс направляет (записывает) данные в канал, другой процесс получает (читает) данные из канала

(Рис. 1).

Процесс

1

Процесс

2

Канал

Write

Read

Рис. 1. Одноканальная схема обмена 2-х процессов

OS UNIX поддерживает 2 типа каналов: **неименованные** (или обычные) и **именованные** (или FIFO-файлы). Неименованные каналы поддерживают обмен данными только между процессами-родственниками, например, между предком и потомком или между потомками одного предка. Именованные каналы могут быть использованы для коммуникации любых, необязательно родственных, процессов. Каналы указанных типов создаются и открываются по-разному, но используются одинаково.

**Неименованные каналы**

**Неименованный канал** создается и открывается СВ pipe, которому в качестве аргументов передается адрес массива 2-х целых чисел. При успешном завершении СВ pipe заполняет адресуемый массив канальными дескрипторами чтения и записи. Эту инициализацию обычно выполняет процесс-предок, когда предполагается использовать канальный механизм для обмена с потомками или между потомками.

**Именованный канал**

В отличие от обычного канала, каждому именованному каналу должен соответствовать оригинальный по маршрутному имени канальный файл, который может быть расположен в произвольном каталоге файловой системы OS UNIX. Для создания канального файла в прикладной программе можно использовать системные вызовы mkfifo или mknod. СВ mkfifo ориентирован исключительно на создание FIFO-файлов. СВ mknod может быть использован для создания новых файлов любых типов. Аналогично, в командном режиме именованный канал может быть создан специальной командой /etc/mkfifo или универсальной командой /etc/mknod.

1. Архитектура UNIX (основная концепция, архитектура, ядро).

**Задачи UNIX**

ОС UNIX представляет собой интерактивную систему, разработанную для одновременной поддержки нескольких процессов и нескольких пользователей. Она была разработана программистами и для программистов, чтобы использовать ее в окружении, в котором большинство пользователей являются относительно опытными и занимаются разработкой проектов ПО. В ОС UNIX есть достаточное количество средств, позволяющих программистам работать вместе и управлять совместным использованием общей информации. Очевидно, что модель группы опытных программистов, совместно работающих над созданием передового ПО, существенно отличается от модели одиночного начинающего пользователя, сидящего за персональным компьютером в текстовом процессоре, и это отличие отражено в ОС UNIX от начала до конца.

Чего хотят от ОС хорошие программисты:

1. Большинство хотело бы, чтобы их система была простой, элегантной и непротиворечивой. Например, на нижнем уровне файл должен представлять собой просто набор байтов. Наличие различных классов файлов для последовательного и произвольного доступа, доступа по ключу, удаленного доступа и т. д.
2. Мощь и гибкость. Это означает, что в системе должно быть небольшое количество базовых элементов, которые можно комбинировать бесконечным числом способов, чтобы приспособить их для конкретного приложения. Одно из основных правил ОС UNIX заключается в том, что каждая программа должна выполнять всего одну функцию, но делать это хорошо. Таким образом, компиляторы не занимаются созданием листингов, так как другие программы могут лучше справиться с этой задачей.

**Архитектура ОС UNIX**

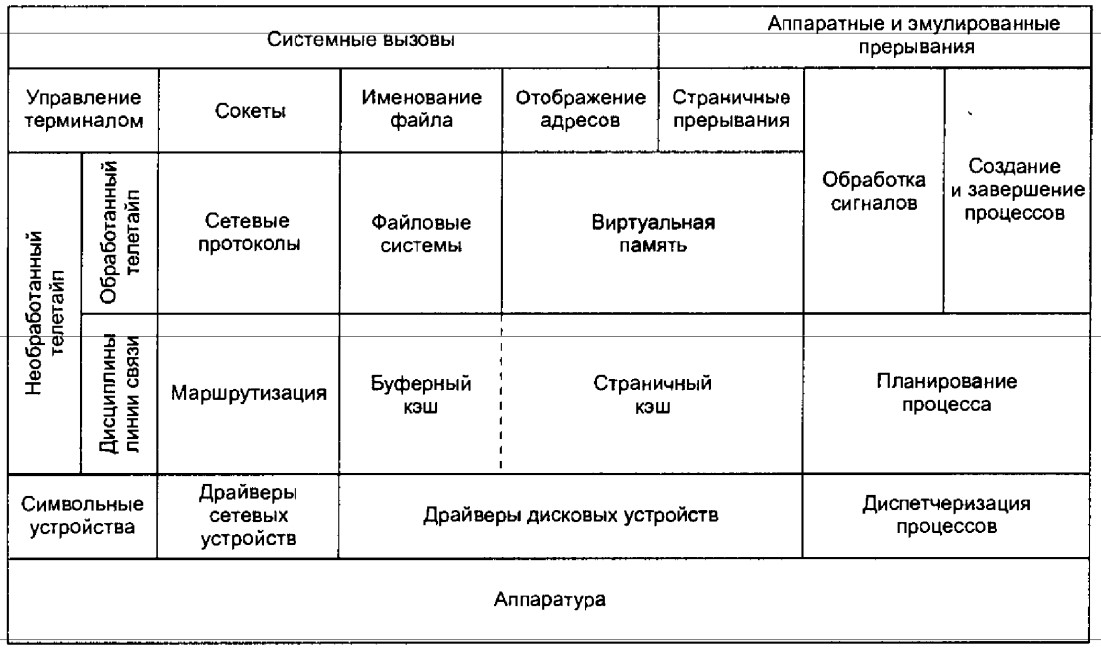
ОС UNIX можно рассматривать в виде пирамиды (рис. 1). У основания пирамиды располагается аппаратное обеспечение, состоящее из CPU, памяти, дисков, терминалов и других устройств. На голом «железе» работает ОС UNIX. Ее функция заключается в управлении аппаратным обеспечением и предоставлении всем программам интерфейса системных вызовов. Эти системные вызовы позволяют программам создавать процессы, файлы и прочие ресурсы, а также управлять ими.



Рис. 1. Уровни ОС UNIX

**Структура ядра**

На рис. 1 была показана общая структура системы UNIX. Рассмотрим ядро системы. Обзор структуры ядра системы UNIX представляет собой довольно непростое дело, так как существует множество различных версий этой системы. Однако, хотя диаграмма на рис. 2 описывает UNIX 4.4BSD, она также применима ко многим другим версиям, возможно, с небольшими изменениями в тех или иных местах.



1. UNIX: файловая система, управление процессами.

**Файловая система UNIX**

Прежде всего пользователь видит в любой ОС, включая систему UNIX, файловую систему. В следующих разделах рассмотрим основные идеи файловой системы UNIX, системные вызовы и детали реализации файловой системы. Некоторые принципы файловой системы UNIX были взяты у ОС MULTICS, в то время как многие другие позаимствованы в MS-DOS, Windows и других системах, хотя имеются и уникальные для UNIX идеи. Устройство файловой системы UNIX особенно интересно тем, что оно отчетливо иллюстрирует принцип разумной достаточности и красоту простых решений. При минимальном механизме и сильно ограниченном количестве системных вызовов ОС UNIX тем не менее предоставляет мощную и элегантную файловую систему.

Основные понятия

**Файл в системе UNIX** — последовательность байтов произвольной длины (от 0 до некоторого максимума), содержащая произвольную информацию. Не делается принципиального различия между текстовыми (ASCII) файлами, двоичными файлами и любыми другими файлами. Значение битов в файле целиком определяется владельцем файла. Системе это безразлично. Изначально размер имен файлов был ограничен 14 символами, но в системе Berkeley UNIX этот предел был расширен до 255 символов, что впоследствии было принято в System V, а также в большинстве других версий. В именах файлов разрешается использовать все ASCII-символы, кроме символа NULL, поэтому допустимы даже имена файлов, состоящие, например, из трех символов возврата каретки (хотя такое имя и не слишком удобно в использовании).

1. Общая архитектура ОС Android. Особенности ядра.

1. ОС Android: Java-машина Dalvik. Обзор Java-интерфейсов прикладного программиста.