С. И. Макаренко
А. А. Ковальский
С. А. Краснов

ПРИНЦИПЫ ПОСТРОЕНИЯ И ФУНКЦИОНИРОВАНИЯ АППАРАТНО-ПРОГРАММНЫХ СРЕДСТВ ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ

Часть 2
Сетевые операционные системы и принципы обеспечения информационной безопасности в сетях

Учебное пособие

2020

Макаренко С.И., Ковальский А.А., Краснов С.А.

ПРИНЦИПЫ ПОСТРОЕНИЯ И ФУНКЦИОНИРОВАНИЯ АППАРАТНО-ПРОГРАММНЫХ СРЕДСТВ ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ

Часть 2
Сетевые операционные системы и принципы обеспечения информационной безопасности в сетях

Учебное пособие

Наукоемкие технологии
Санкт-Петербург
2020

УДК 004.75
ББК 32.973.26-018.2
М15

Рецензенты:

***Буйневич Михаил Викторович**, доктор технических наук, профессор, профессор кафедры безопасности информационных технологий Санкт-Петербургского государственного университета телекоммуникаций имени проф. М.А. Бонч-Бруевича.*

***Семенов Сергей Сергеевич**, доктор технических наук, доцент, профессор кафедры «Технического обеспечения связи и автоматизации» Военной академии связи имени Маршала Советского Союза С.М. Буденного.*

***Хомоненко Анатолий Дмитриевич**, доктор технических наук, профессор, заведующий кафедрой «Информационные и вычислительные системы» Петербургского государственного университета путей сообщения Императора Александра I.*

М15 Макаренко С. И., Ковальский А. А., Краснов С. А. Принципы построения и функционирования аппаратно-программных средств телекоммуникационных систем: учебное пособие. Часть 2: Сетевые операционные системы и принципы обеспечения информационной безопасности в сетях. – СПб.: Наукоемкие технологии, 2020. – 357 с.

ISBN 978-5-6044429-8-2

Учебное пособие написано по опыту проведения авторами дисциплин «Основы построения инфокоммуникационных систем и сетей», «Многоканальные системы связи специального назначения», «Эксплуатация инфокоммуникационных систем специального назначения», «Администрирование спутниковых сетей связи космических комплексов», «Защита инфокоммуникационных систем специального назначения», «Информационная безопасность», «Защита информации», «Автоматизированные системы специального назначения», «Аудит безопасности критической информационной инфраструктуры», «Научно-исследовательская работа», «Технология разработки информационных систем в защищенном исполнении», преподаваемых по программам специалитета по специальностям «Инфокоммуникационные технологии и системы специальной связи», «Математическое и программное обеспечение» и «Компьютерная безопасность» в Военно-космической академии имени А.Ф. Можайского и в Санкт-Петербургском государственном электротехническом университете «ЛЭТИ» имени В.И. Ульянова (Ленина). Учебное пособие учитывает требования государственного образовательного стандарта, и содержательно соответствует отдельным темам, изучаемым в рамках вышеуказанных дисциплин.

В первом разделе пособия подробно рассматриваются сетевые операционные систем, основы их построения и функционирования. Второй раздел пособия посвящен обеспечению информационной безопасности в телекоммуникационных сетях.

ISBN 978-5-6044429-8-2

Оглавление

Предисловие

Это учебное пособие написано с целью систематизации вопросов построения и функционирования операционных систем (ОС) и прикладного программного обеспечения (ПО) в телекоммуникационных сетях (ТКС), а также вопросов обеспечения их информационной безопасности (ИБ). Основное внимание уделяется современным направлениям развития сетевых ОС и принципам ИБ реализованным в современных ТКС специального и общего назначения.

Учебное пособие продолжает работу «Принципы построения и функционирования аппаратно-программных средств телекоммуникационных систем. Часть 1: Принципы функционирования аппаратных средств телекоммуникационных и вычислительных систем» [1] и является второй частью этого издания.

Учебное пособие написано по опыту проведения авторами дисциплин «Основы построения инфокоммуникационных систем и сетей», «Многоканальные системы связи специального назначения», «Эксплуатация инфокоммуникационных систем специального назначения», «Администрирование спутниковых сетей связи космических комплексов», «Защита инфокоммуникационных систем специального назначения», «Информационная безопасность», «Аудит безопасности критической информационной инфраструктуры», «Научно-исследовательская работа», «Технология разработки информационных систем в защищенном исполнении», преподаваемых по программам специалитета по специальностям «Инфокоммуникационные технологии и системы специальной связи», «Математическое и программное обеспечение», «Компьютерная безопасность» в Военно-космической академии имени А.Ф. Можайского и в Санкт-Петербургском государственном электротехническом университете «ЛЭТИ» имени В.И. Ульянова (Ленина). Учебное пособие учитывает требования государственного образовательного стандарта, и содержательно соответствует отдельным темам, изучаемым в рамках вышеуказанных дисциплинах.

В основу учебного пособия положено развитие и дополнение ранее изданных учебных работ авторов [2, 3], с учетом материала изданий [4-6, 33-99]. Следует отметить, что этом пособии нашли частичное отражение отдельные результаты научно-исследовательской работы авторов по исследованию качества функционирования ОС и ПО узлов ТКС в условиях нарушения требований ИБ нарушителями, а также влияния других дестабилизирующих факторов [7-32].

В первом разделе пособия подробно рассматриваются основы построения и функционирования сетевых ОС. Примеры функционирования рассмотрены на ОС, широко используемых в узловом оборудовании современных ТКС: Windows Server; UNIX, Linux, а также ОС реального времени управления специальными комплексами QNX. В основу материал этого раздела положено обобщение работ [1, 2, 5, 4, 9, 28-42, 75-77].

Во втором разделе пособия подробно рассматриваются основы обеспечения ИБ в ТКС. Рассмотрены типовые атаки и информационно-технические воздействия (ИТВ) характерные для современных ТКС, а также такие стандартные средства сетевой защиты как межсетевые экраны (firewall), частные виртуальные сети (VPN), системы обнаружения вторжений (IDS), системы управления безопасностью электронной коммуникации (на примере e-mail). В основу материала этого раздела положено обобщение работ [3, 6-8, 10-27, 43-73, 78-99].

Авторы выражает благодарность рецензентам за кропотливый труд по поиску ошибок и неточностей, а также ценные замечания, которые помогли сделать материал пособия лучше и доступнее.

Предложения и замечания по учебному пособию авторы просят направлять С.И. Макаренко на email: mak-serg@yandex.ru.

РАЗДЕЛ 1. СЕТЕВЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

1. Архитектура операционной системы

Операционная система (ОС) – от англ. operating system (OS) – комплекс взаимосвязанных программ, предназначенных для управления ресурсами вычислительной системы, управления другими программами и организации взаимодействия с пользователем.

В логической структуре типичной вычислительной системы операционная система занимает промежуточное положение между аппаратной частью вычислительной системы, с одной стороны, и прикладными программами пользователя, с другой. Разработчикам программного обеспечения (ОС) операционная система позволяет абстрагироваться от деталей реализации и функционирования аппаратных средств, предоставляя необходимый набор функций для программирования и функционирования приложений.

1.1. Ядро и вспомогательные модули операционной системы

Любая сложная система должна иметь понятную и рациональную структуру, то есть разделяться на части – модули, имеющие вполне законченное функциональное назначение с четко оговоренными правилами взаимодействия. Ясное понимание роли каждого отдельного модуля существенно упрощает работу по модификации и развитию системы. Напротив, сложную систему без хорошей структуры чаще проще разработать заново, чем модернизировать.

Функциональная сложность операционной системы неизбежно приводит к сложности ее архитектуры.

Архитектура ОС – структурная организация ОС на основе различных программных модулей.

Обычно в состав ОС входят:

- исполняемые и объектные модули стандартных для этой ОС форматов;
- библиотеки разных типов;
- модули исходного текста программ;
- программные модули специального формата (например, загрузчик ОС, драйверы ввода-вывода);
- конфигурационные файлы;
- файлы документации и модули справочной системы и т. д.

Большинство современных операционных систем представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой ар-

хитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС.

Общим подходом к структуризации ОС является разделение ее модулей на две группы:

- *ядро* – компоненты, выполняющие основные функции ОС;
- *модули* – компоненты, выполняющие вспомогательные функции ОС.

Модули ядра выполняют такие базовые функции ОС, как управление процессами, памятью, устройствами ввода-вывода и т. п.

Ядро составляет основу ОС, без него ОС является полностью неработоспособной и не сможет выполнить ни одну из своих функций.

В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса, такие как:

- переключение контекстов;
- загрузка/выгрузка станиц;
- обработка прерываний;
- поддержка приложений, создание для них так называемой прикладной программной среды.

Приложения могут обращаться к ядру с запросами – системными вызовами – для выполнения тех или иных действий, например, для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т.д.

Интерфейс прикладного программирования (API – application programming interface) – функции ядра, которые могут вызываться приложениями.

Скорость выполнения функций ядра 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.

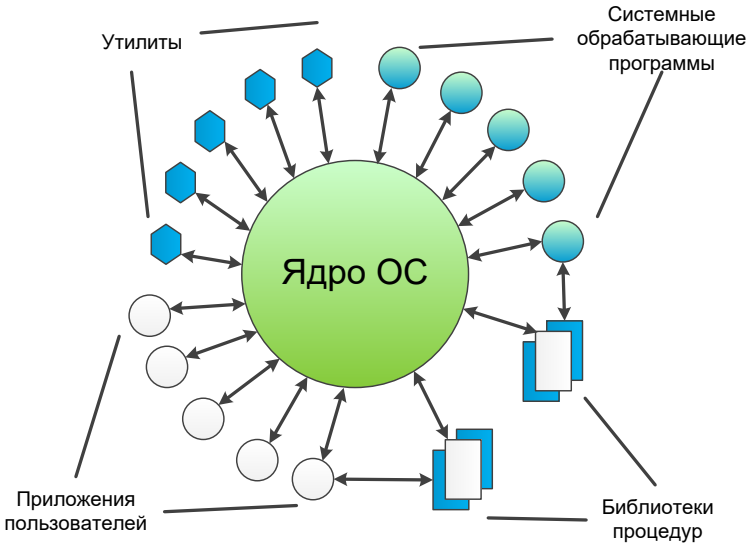


Рис. 1.1. Взаимодействие между ядром и вспомогательными модулями ОС

Вспомогательные модули ОС подразделяются на следующие группы:

- *утилиты* – программы, решающие отдельные задачи управления и сопровождения компьютерной системы, такие, например, как программы сжатия дисков, архивирования данных на магнитную ленту;
- *системные обрабатывающие программы* – текстовые или графические редакторы, компиляторы, компоновщики, отладчики;
- *программы предоставления пользователю дополнительных услуг* – специальный вариант пользовательского интерфейса, калькулятор и даже игры;
- *библиотеки процедур различного назначения* – библиотеки, упрощающие разработку приложений, например, библиотека математических функций, функций ввода-вывода и т. д.

Разделение операционной системы на ядро и модули-приложения обеспечивает легкую расширяемость ОС. Чтобы добавить новую высокоуровневую функцию, достаточно разработать новое приложение, и при этом не требуется модифицировать ответственные функции, образующие ядро системы. Однако внесение изменений в функции ядра может оказаться гораздо сложнее, и сложность эта зависит от структурной организации самого ядра и может потребовать полной перекомпиляции модулей ядра.

1.2. Ядро в привилегированном режиме

Для надежного управления ходом выполнения приложений операционная система должна иметь по отношению к приложениям определенные привилегии. Иначе некорректно работающее приложение может вмешаться в работу ОС и, например, разрушить часть ее кодов. Операционная система должна обладать исключительными полномочиями также для того, чтобы играть роль арбитра в споре приложений за ресурсы компьютера в мультипрограммном режиме. Ни одно приложение не должно иметь возможности без ведома ОС получать дополнительную область памяти, занимать процессор дольше разрешенного операционной системой периода времени, непосредственно управлять совместно используемыми внешними устройствами.

Привилегии для модулей ядра операционной системы обеспечивают специальные средства аппаратной поддержки.

Аппаратура должна поддерживать как минимум два режима работы:

- *пользовательский режим* (user mode);
- *привилегированный режим*, который также называют *режимом ядра* (kernel mode), или *режимом супервизора* (supervisor mode).

Подразумевается, что операционная система или некоторые ее части работают в привилегированном режиме, а приложения – в пользовательском режиме.

Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро становится той частью ОС, которая работает в привилегированном режиме (рис. 1.2). Иногда это свойство – работа в привилегированном режиме – служит основным определением понятия «ядро».

Приложения ставятся в подчиненное положение за счет запрета выполнения в пользовательском режиме следующих критичных команд:

- переключение процессора с задачи на задачу;
- управление устройствами ввода-вывода;
- управления доступом к механизмам распределения и защиты памяти.

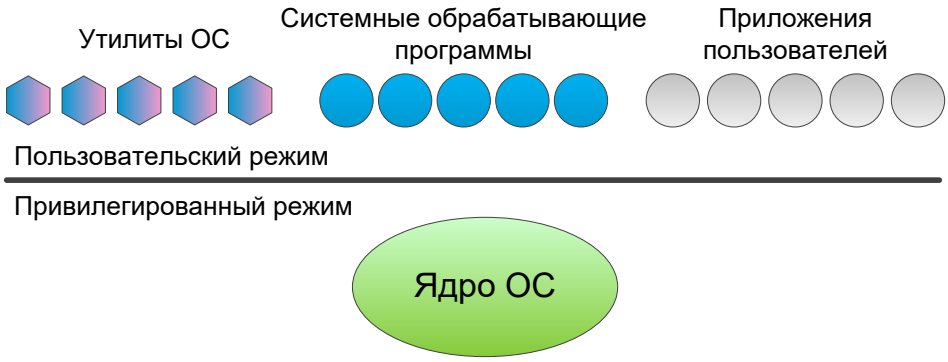


Рис. 1.2. Архитектура операционной системы с ядром в привилегированном режиме

Выполнение некоторых инструкций в пользовательском режиме запрещается безусловно (очевидно, что к таким инструкциям относится инструкция перехода в привилегированный режим), тогда как другие запрещается выполнять только при *определенных условиях.*

Повышение устойчивости операционной системы, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления выполнения системных вызовов.

Системный вызов ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению – обратно (рис. 1.3).

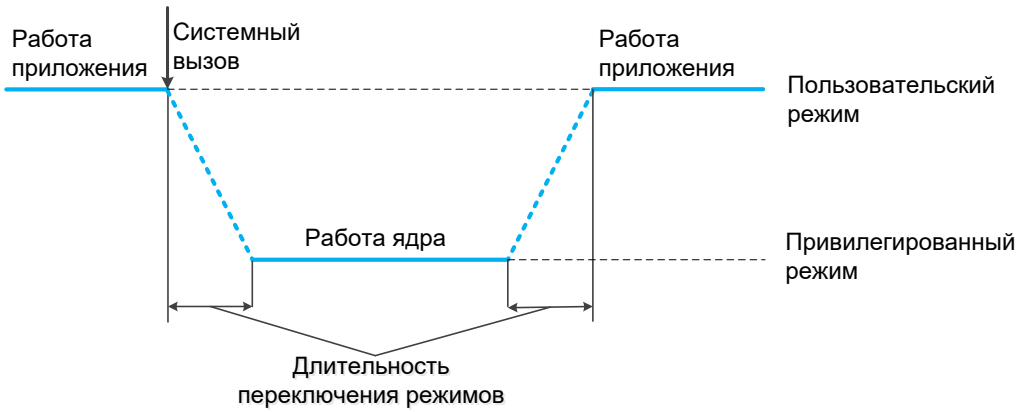


Рис. 1.3. Смена режимов при выполнении системного вызова к привилегированному ядру

Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической. Ее используют многие популярные операционные системы, в том числе многочисленные версии UNIX, Linux, и с определенными модификациями ОС семейства Windows.

В некоторых случаях разработчики ОС отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, в специализированной ОС NetWare используется привилегированный режим как для работы ядра, так и для работы своих специфических приложений – загружаемых модулей NLM (рис. 1.4).

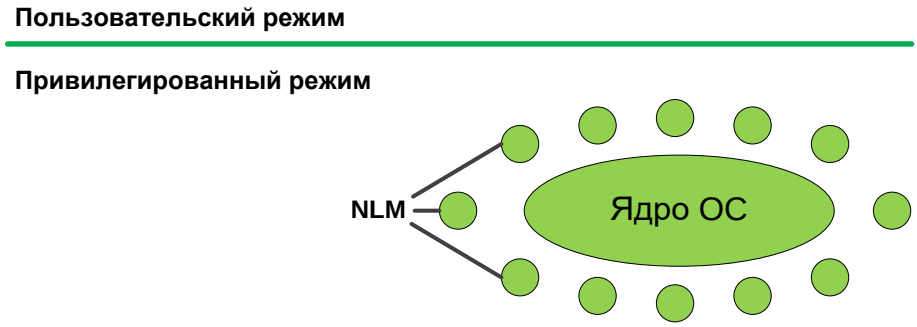


Рис. 1.4. Упрощенная архитектура ОС NetWare

При таком построении ОС обращения приложений к ядру выполняются быстрее, так как нет переключения режимов, однако при этом отсутствует надежная аппаратная защита памяти, занимаемой модулями ОС, от некорректно работающего приложения. Потенциальное снижение надежности ОС NetWare, компенсируется за счет тщательной отладки каждого приложения.

1.3. Многослойная структура операционной системы

Вычислительную систему, работающую под управлением ОС, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев (рис. 1.5):

- нижний слой – аппаратура;
- промежуточный слой – ядро ОС;
- верхний слой – утилиты, программы и приложения.

При такой организации ОС приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра.

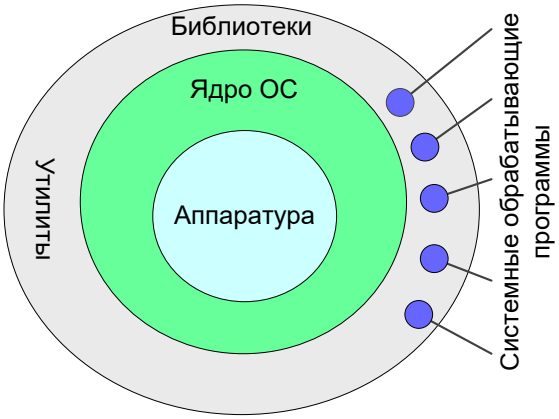


Рис. 1.5. Трехслойная схема вычислительной системы

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе и программных. В соответствии с ним система состоит из иерархии слоев. Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс (рис. 1.6).

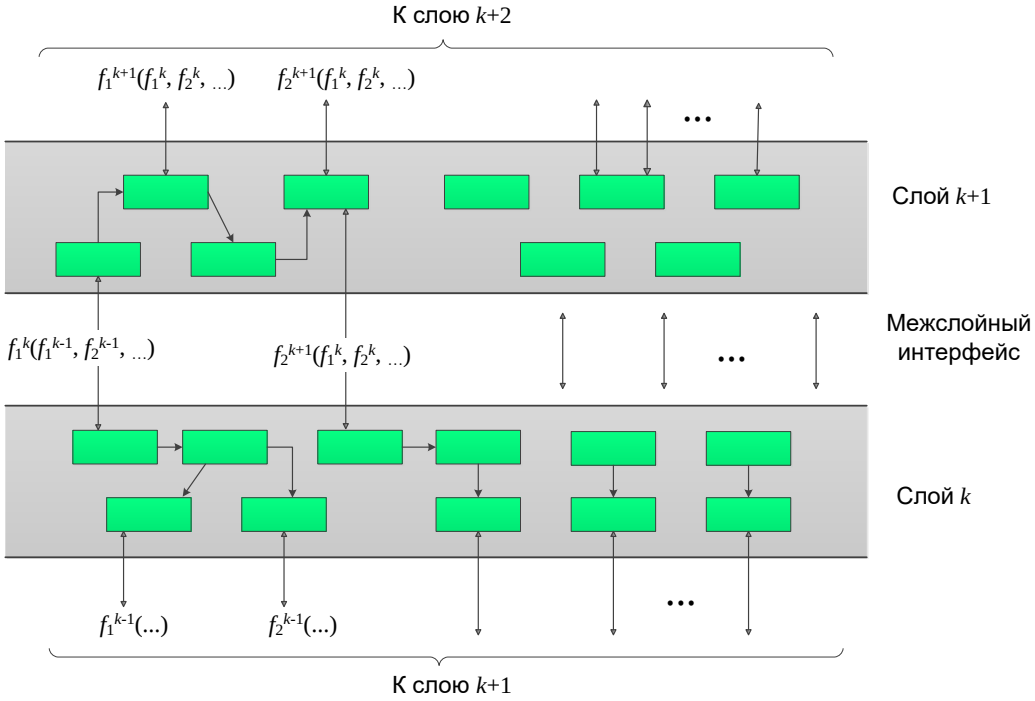


Рис. 1.6. Концепция многослойного взаимодействия

На основе функций нижележащего слоя следующий (вверх по иерархии) слой строит свои функции – более сложные и более важные, которые, в свою очередь, оказываются примитивами для создания еще более мощных функций вышележащего слоя. Строгие правила касаются только взаимодействия между слоями системы, а между модулями внутри слоя связи могут быть произвольными. Отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться к другому модулю своего

слоя, либо обратиться за помощью к нижележащему слою через межслойный интерфейс.

Ядро может состоять из следующих слоев (рис. 1.7):

Средства аппаратной поддержки ОС. Часть функций ОС может выполняться и аппаратными средствами. К операционной системе относят, естественно, не все аппаратные устройства компьютера, а только средства аппаратной поддержки ОС, то есть те, которые прямо участвуют в организации вычислительных процессов:

- средства поддержки привилегированного режима;
- систему прерываний;
- средства переключения контекстов процессов;
- средства защиты областей памяти и т. п.

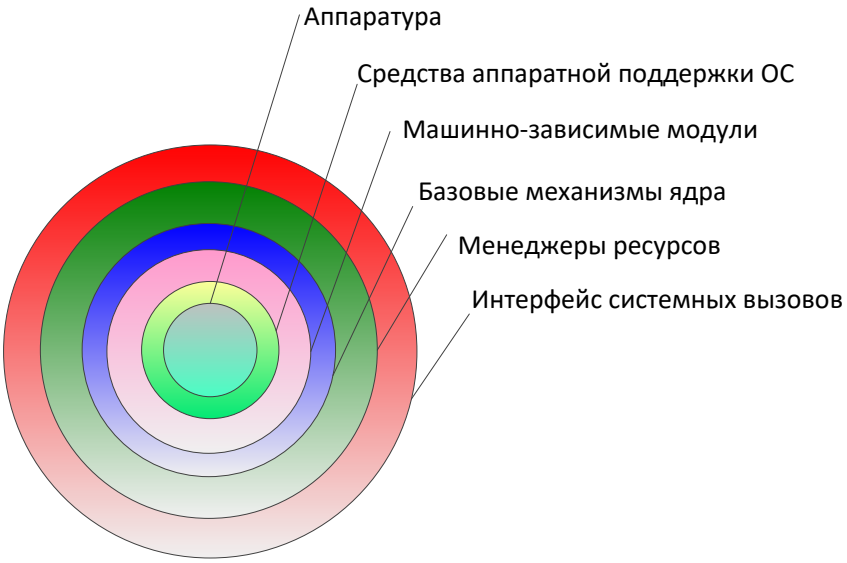


Рис. 1.7. Многослойная структура ядра ОС

Машинно-зависимые компоненты ОС. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеальном случае этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых этой ОС.

Базовые механизмы ядра. Этот слой выполняет наиболее примитивные операции ядра, такие как:

- программное переключение контекстов процессов;
- диспетчеризацию прерываний;
- перемещение страниц из памяти на диск и обратно и т. п.

Модули этого слоя не принимают решений о распределении ресурсов – они только отрабатывают принятые «наверху» решения, что и дает повод называть их исполнительными механизмами для модулей верхних слоев.

Менеджеры ресурсов. Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на этом слое работают менеджеры (называемые также диспетчерами):

- процессов;
- ввода-вывода;
- файловой системы;
- оперативной памяти.

Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений. Для исполнения принятых решений менеджер обращается к нижележащему слою. Внутри слоя менеджеров существуют тесные взаимные связи, отражающие тот факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам – процессору, области памяти, возможно, к определенному файлу или устройству ввода-вывода.

Интерфейс системных вызовов (*API*). Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс операционной системы. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Для осуществления комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов.

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними может быть и иным.

1.4. Аппаратная зависимость и переносимость операционной системы

1.4.1. Аппаратная зависимость операционной системы

Многие операционные системы успешно работают на различных аппаратных платформах без существенных изменений в своем составе. Во многом это объясняется тем, что, несмотря на различия в деталях, средства аппаратной поддержки ОС большинства компьютеров приобрели сегодня много типовых черт, а именно эти средства в первую очередь влияют на работу компонентов операционной системы. В ОС можно выделить достаточно компактный слой машинно-зависимых компонентов ядра и сделать остальные слои ОС общими для разных аппаратных платформ.

Четкой границы между программной и аппаратной реализацией функций ОС не существует – решение о том, какие функции ОС будут выполняться программно, а какие аппаратно, принимается разработчиками аппаратного и программного обеспечения компьютера.

Современные аппаратные платформы имеют некоторый типичный набор средств аппаратной поддержки ОС, в который входят следующие компоненты:

- средства поддержки привилегированного режима;
- средства трансляции адресов;
- средства переключения процессов;
- система прерываний;
- системный таймер;
- средства защиты областей памяти.

Средства поддержки привилегированного режима обычно основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора. Этот регистр содержит признаки, определяющие режимы работы процессора, в том числе и признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды. Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро – пользователь) или четыре (например: ядро – супервизор – выполнение – пользователь). В обязанности средств поддержки привилегированного режима входит выполнение проверки допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности.

Средства трансляции адресов выполняют операции преобразования виртуальных адресов, которые содержатся в кодах процесса, в адреса физической памяти. Таблицы, предназначенные при трансляции адресов, обычно имеют большой объем, поэтому для их хранения используются области оперативной памяти, а аппаратура процессора содержит только указатели на эти области. Средства трансляции адресов используют данные указатели для доступа к элементам таблиц и аппаратного выполнения алгоритма преобразования адреса, что значительно ускоряет процедуру трансляции по сравнению с ее чисто программной реализацией.

Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным. Содержимое контекста обычно включает содержимое всех регистров общего назначения процессора, регистра флагов операций (то есть флагов нуля, переноса, переполнения и т. п.), а также тех системных регистров и указателей, которые связаны с отдельным процессом, а не операционной системой, например, указателя на таблицу трансляции адресов процесса. Для хранения контекстов приостановленных процессов обычно используются области оперативной памяти, которые поддерживаются указателями процессора.

Переключение контекста выполняется по определенным командам процессора, например, по команде перехода на новую задачу. Такая команда вызывает автоматическую загрузку данных из сохраненного контек-

ста в регистры процессора, после чего процесс продолжается с прерванного ранее места.

Система прерываний нужна для того, чтобы оповестить процессор о возникновении в вычислительной системе некоторого непредсказуемого события или события, которое не синхронизировано с циклом работы процессора и позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, быстро переходить с одной программы на другую. Примерами таких событий могут служить:

- завершение операции ввода-вывода внешним устройством (например, запись блока данных контроллером диска);
- некорректное завершение арифметической операции (например, переполнение регистра);
- истечение интервала астрономического времени.

При возникновении условий прерывания его источник (контроллер внешнего устройства, таймер, арифметический блок процессора и т. п.) выставляет определенный электрический сигнал. Этот сигнал прерывает выполнение процессором последовательности команд, задаваемой исполняемым кодом, и вызывает автоматический переход на заранее определенную процедуру, называемую процедурой обработки прерываний. После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода.

Системный таймер, необходим операционной системе для выдержки интервалов времени. Для этого в регистр таймера программно загружается значение требуемого интервала в условных единицах, из которого затем автоматически с определенной частотой начинает вычитаться по единице. Частота «тиков» таймера, как правило, тесно связана с частотой тактового генератора процессора. При достижении нулевого значения счетчика таймер инициирует прерывание, которое обрабатывается процедурой операционной системы. Прерывания от системного таймера используются ОС в первую очередь для слежения за тем, как отдельные процессы расходуют время процессора.

Средства защиты областей памяти обеспечивают на аппаратном уровне проверку возможности программного кода осуществлять с данными определенной области памяти такие операции, как чтение, запись или выполнение (при передачах управления). Если аппаратура компьютера поддерживает механизм трансляции адресов, то средства защиты областей памяти встраиваются в этот механизм. Функции аппаратуры по защите памяти обычно состоят в сравнении уровней привилегий текущего кода процессора и сегмента памяти, к которому производится обращение.

1.4.2. Переносимость операционной системы

Опыт разработки операционных систем показывает: ядро можно спроектировать таким образом, что только часть модулей будут машинно-

зависимыми, а остальные не будут зависеть от особенностей аппаратной платформы. В хорошо структурированном ядре машинно-зависимые модули локализованы и образуют программный слой, естественно примыкающий к слою аппаратуры. Такая локализация машинно-зависимых модулей существенно упрощает перенос операционной системы на другую аппаратную платформу.

Переносимая (*portable*) или *мобильная ОС* – система код которой может быть сравнительно легко перенесен с аппаратной платформы одного типа на аппаратную платформу другого типа.

Для обеспечения свойства мобильности ОС, разработчики должны следовать следующим правилам.

- Большая часть кода должна быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить систему. Такими языками являются стандартизованные языки высокого уровня. Большинство переносимых ОС пишется на высокоуровневом языке, который имеет много особенностей, полезных для разработки кодов операционной системы, и компиляторы которого широко доступны. Ассемблер используется только для тех непереносимых частей системы, которые должны непосредственно взаимодействовать с аппаратурой (например, обработчик прерываний), или для частей, которые требуют максимальной скорости (например, целочисленная арифметика повышенной точности).
- Объем машинно-зависимых частей кода, которые непосредственно взаимодействуют с аппаратными средствами, должен быть по возможности минимизирован. Следует всячески избегать прямого манипулирования регистрами и другими аппаратными средствами процессора. Разработчики ОС должны также исключить возможность использования по умолчанию стандартных конфигураций аппаратуры или их характеристик. Аппаратно-зависимые параметры можно «спрятать» в программно-задаваемые данные абстрактного типа. Для осуществления всех необходимых действий по управлению аппаратурой, представленной этими параметрами, должен быть написан набор аппаратно-зависимых функций.
- Аппаратно-зависимый код должен быть надежно изолирован в нескольких модулях, а не быть распределен по всей системе. Изоляции подлежат все части ОС, которые отражают специфику как процессора, так и аппаратной платформы в целом. Низкоуровневые компоненты ОС, имеющие доступ к процессорно-зависимым структурам данных и регистрам, должны быть оформлены в виде компактных модулей, которые могут быть заменены аналогичными модулями для других процессоров.

В идеале слой машинно-зависимых компонентов ядра полностью экранирует остальную часть ОС от конкретных деталей аппаратной платформы, которую поддерживает данная ОС (рис. 1.8).

Для реализации свойства переносимости происходит подмена реальной аппаратуры некой унифицированной виртуальной машиной, одинаковой для всех вариантов аппаратной платформы. Все слои операционной системы, которые лежат выше слоя машинно-зависимых компонентов, управляют именно этой виртуальной аппаратурой. Таким образом, у разработчиков появляется возможность создавать один вариант машинно-независимой части ОС (включая компоненты ядра, утилиты, системные обрабатывающие программы) для всего набора поддерживаемых платформ.

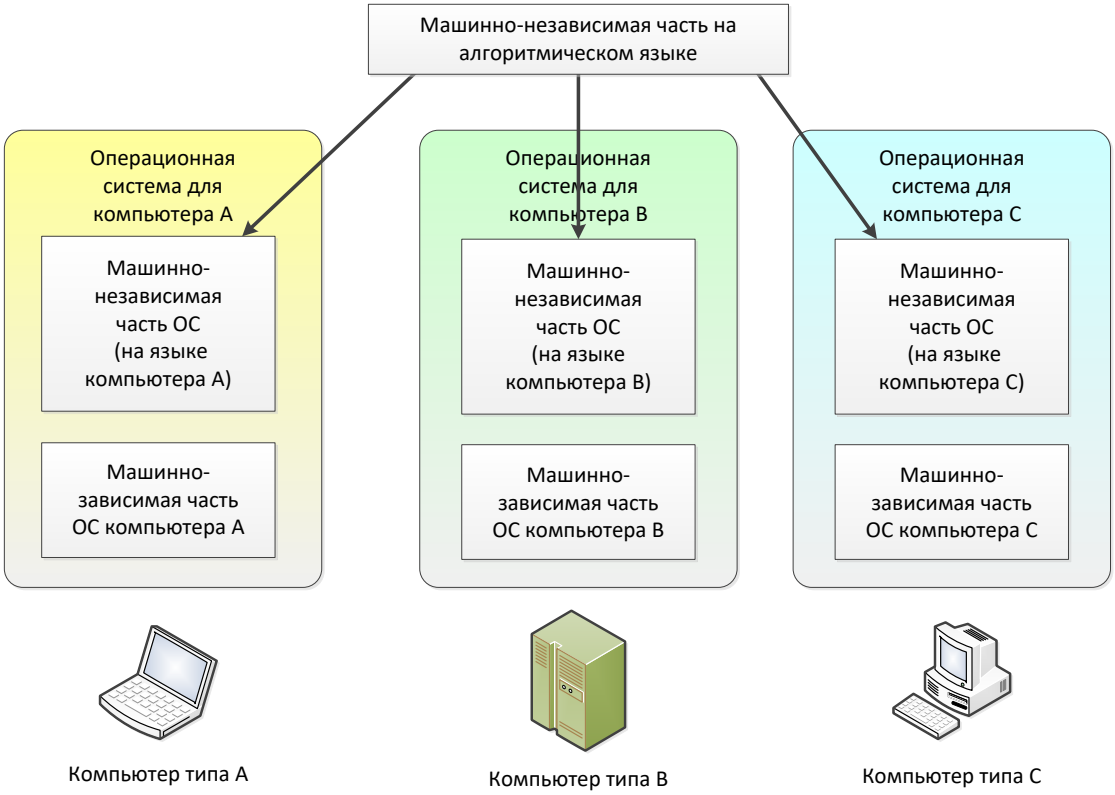


Рис. 1.8. Перенос операционной системы на разные аппаратные платформы

1.5. Микроядерная архитектура

Микроядерная архитектура является альтернативой классическому способу построения операционной системы.

Под *классической архитектурой* обычно понимается рассмотренная выше структурная организация ОС, в соответствии с которой все основные функции операционной системы, составляющие многослойное ядро, выполняются в привилегированном режиме. При этом некоторые вспомогательные функции ОС оформляются в виде приложений и выполняются в пользовательском режиме наряду с обычными пользовательскими программами (становясь системными утилитами или обрабатывающими программами). Каждое приложение пользовательского режима работает в собственном адресном пространстве и защищено тем самым от какого-либо

вмешательства других приложений. Код ядра, выполняемый в привилегированном режиме, имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций.

Суть *микроядерной архитектуры* состоит в том, что в привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. При этом микроядро защищено как от остальных частей ОС, так и от приложений (рис. 1.9).

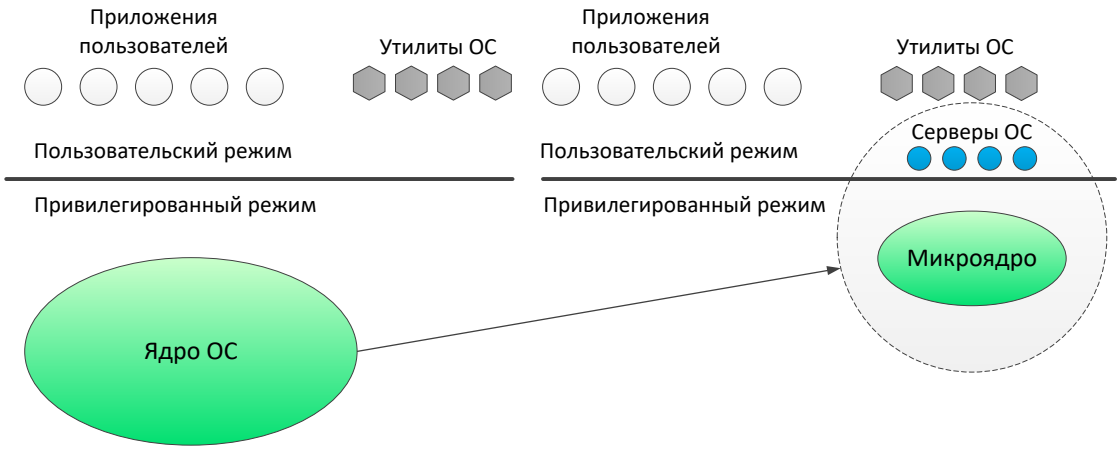


Рис. 1.9. Перенос основного объема функций ядра в пользовательское пространство

Набор функций микроядра обычно соответствует функциям слоя базовых механизмов обычного ядра. Такие функции операционной системы трудно, если не невозможно, выполнить в пространстве пользователя.

При этом в составе микроядра обычно остаются:

- машинно-зависимые модули;
- модули, выполняющие базовые (но не все!) функции ядра по управлению процессами;
- обработка прерываний;
- модули управления виртуальной памятью;
- модули управления пересылкой сообщений и управлению устройствами ввода-вывода;
- модули, связанные с загрузкой или чтением регистров устройств.

Все остальные более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме.

В общем случае многие менеджеры ресурсов, являющиеся неотъемлемыми частями обычного ядра – файловая система (ФС), подсистемы управления виртуальной памятью и процессами, менеджер безопасности и т. п., – становятся «периферийными» модулями, работающими в пользовательском режиме.

Схематично механизм обращения к функциям ОС, оформленным в виде серверов, выглядит следующим образом (рис. 1.10).

Клиент, которым может быть либо прикладная программа, либо другой компонент ОС, запрашивает выполнение некоторой функции у соответствующего сервера, посылая ему сообщение. Непосредственная передача сообщений между приложениями невозможна, так как их адресные пространства изолированы друг от друга. Микроядро, выполняющееся в привилегированном режиме, имеет доступ к адресным пространствам каждого из этих приложений и поэтому может работать в качестве посредника. Микроядро сначала передает сообщение, содержащее имя и параметры вызываемой процедуры нужному серверу, затем сервер выполняет запрошенную операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения. Таким образом, работа микроядерной операционной системы соответствует известной модели клиент-сервер, в которой роль транспортных средств выполняет микроядро.

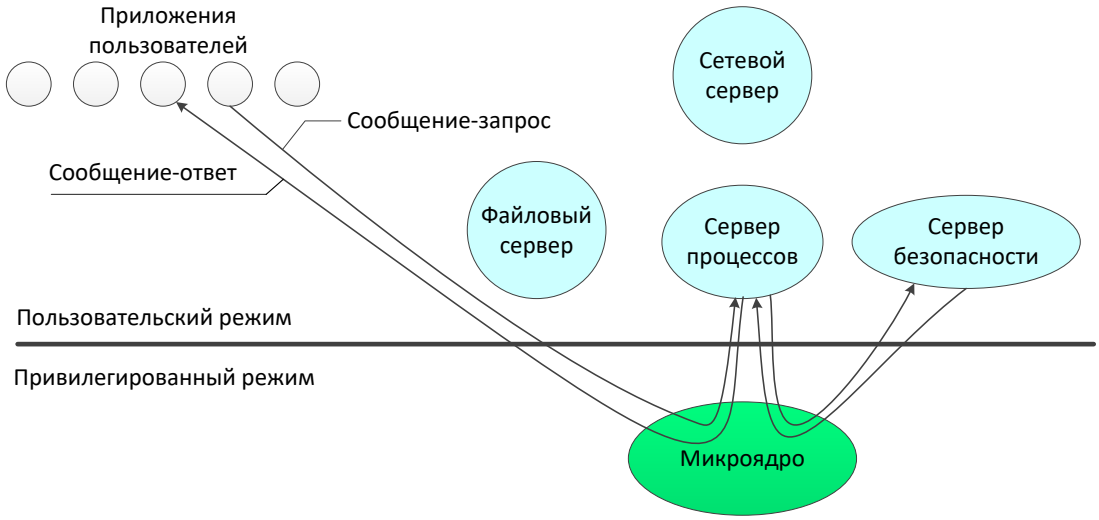


Рис. 1.10. Реализация системного вызова
в микроядерной архитектуре

Операционные системы, основанные на концепции микроядра, в высокой степени удовлетворяют большинству требований, предъявляемых к современным ОС, обладая следующими достоинствами.

- *Высокой степенью переносимости* – это обусловлена тем, что весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.
- *Расширяемостью.* В традиционных системах даже при наличии многослойной структуры нелегко удалить один слой и поменять его на другой по причине множественности и размытости интерфейсов между слоями. Добавление новых функций и изменение существующих требует хорошего знания операционной системы и больших затрат времени. В то же время ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС. Добавление новой подси-

стемы требует разработки нового приложения, что никак не затрагивает целостность микроядра.

- *Надежностью.* Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти и таким образом защищен от других серверов операционной системы, что не наблюдается в традиционной ОС, где все модули ядра могут влиять друг на друга. И если отдельный сервер терпит крах, то он может быть перезапущен без останова или повреждения остальных серверов ОС. Другим потенциальным источником повышения надежности ОС является уменьшенный объем кода микроядра по сравнению с традиционным ядром – это снижает вероятность появления ошибок программирования.
- *Поддержкой распределенных приложений.* Модель с микроядром хорошо подходит для поддержки распределенных вычислений, так как использует механизмы, аналогичные сетевым: взаимодействие клиентов и серверов путем обмена сообщениями. Серверы микроядерной ОС могут работать как на одном, так и на разных компьютерах. Переход к распределенной обработке требует минимальных изменений в работе операционной системы – просто локальный транспорт заменяется на сетевой.

Основным недостатком микроядерной архитектуры является невысокая производительность. При классической организации ОС (рис. 1.11, а) выполнение системного вызова сопровождается двумя переключениями режимов, а при микроядерной организации (рис. 1.11, б) – четырьмя.

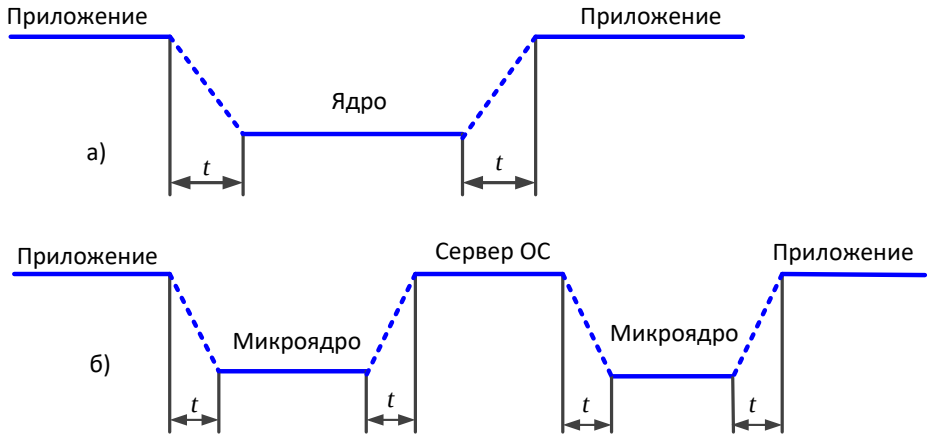


Рис. 1.11. Смена режимов при выполнении системного вызова

Таким образом, ОС на основе микроядра при прочих равных условиях всегда будет менее производительной, чем ОС с классическим ядром. Именно по этой причине микроядерный подход не получил такого широкого распространения, которое ему предрекали.

Классическим примером микроядерной ОС является QNX.

1.6. Совместимость и множественные прикладные среды

В то время как многие архитектурные особенности операционных систем непосредственно касаются только системных программистов, концепция множественных прикладных сред непосредственно связана с нуждами конечных пользователей – возможностью операционной системы выполнять приложения, написанные для других операционных систем.

Совместимость ОС – свойство выполнять приложения, написанные для других операционных систем.

Рассматривают следующие уровни совместимости.

- *Совместимость на уровне исходных текстов* – требует наличия соответствующего компилятора в составе ОС, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция имеющихся исходных текстов в новый исполняемый модуль.
- *Двоичная совместимость* – возможность использовать один и тот же программный продукт, поставляемый в виде двоичного исполняемого кода, в различных операционных средах и на различных машинах.

Достаточными условиями двоичной совместимости является соблюдения следующих условий:

- вызовы функций API, которые содержит приложение, должны поддерживаться этой ОС;
- внутренняя структура исполняемого файла приложения должна соответствовать структуре исполняемых файлов этой ОС.

Гораздо сложнее достичь двоичной совместимости операционным системам, предназначенным для выполнения на процессорах, имеющих разные архитектуры. Помимо соблюдения приведенных выше условий необходимо организовать эмуляцию обработки двоичного кода.

Пусть, например, требуется выполнить программу для ПК-совместимого компьютера на компьютере ARM. Компьютер ARM построен на основе RISC-процессора, а ПК – на основе процессора x86. Процессор ARM имеет архитектуру (систему команд, состав регистров и т. п.), отличную от архитектуры процессора x86, поэтому ему непонятен двоичный код программы, содержащей инструкции этого процессора. Для того чтобы компьютер ARM смог интерпретировать машинные инструкции, которые ему изначально непонятны, на нем должно быть установлено специальное программное обеспечение – *эмулятор*.

Создание эмулятора полноценной прикладной среды, полностью совместимой со средой другой операционной системы, является достаточно сложной задачей, тесно связанной со структурой операционной системы.

Существуют различные варианты построения множественных прикладных сред, отличающиеся как особенностями архитектурных решений,

так и функциональными возможностями, обеспечивающими различную степень переносимости приложений:

1. Реализация множественных прикладных сред основанной на стандартной многоуровневой структуре ОС. На рис. 1.12 операционная система ОС1 поддерживает кроме своих «родных» приложений приложения операционных систем ОС2 и ОС3. Для этого в ее составе имеются специальные приложения – прикладные программные среды, – которые транслируют интерфейсы «чужих» операционных систем API ОС2 и API ОС3 в интерфейс своей «родной» операционной системы – API ОС1.

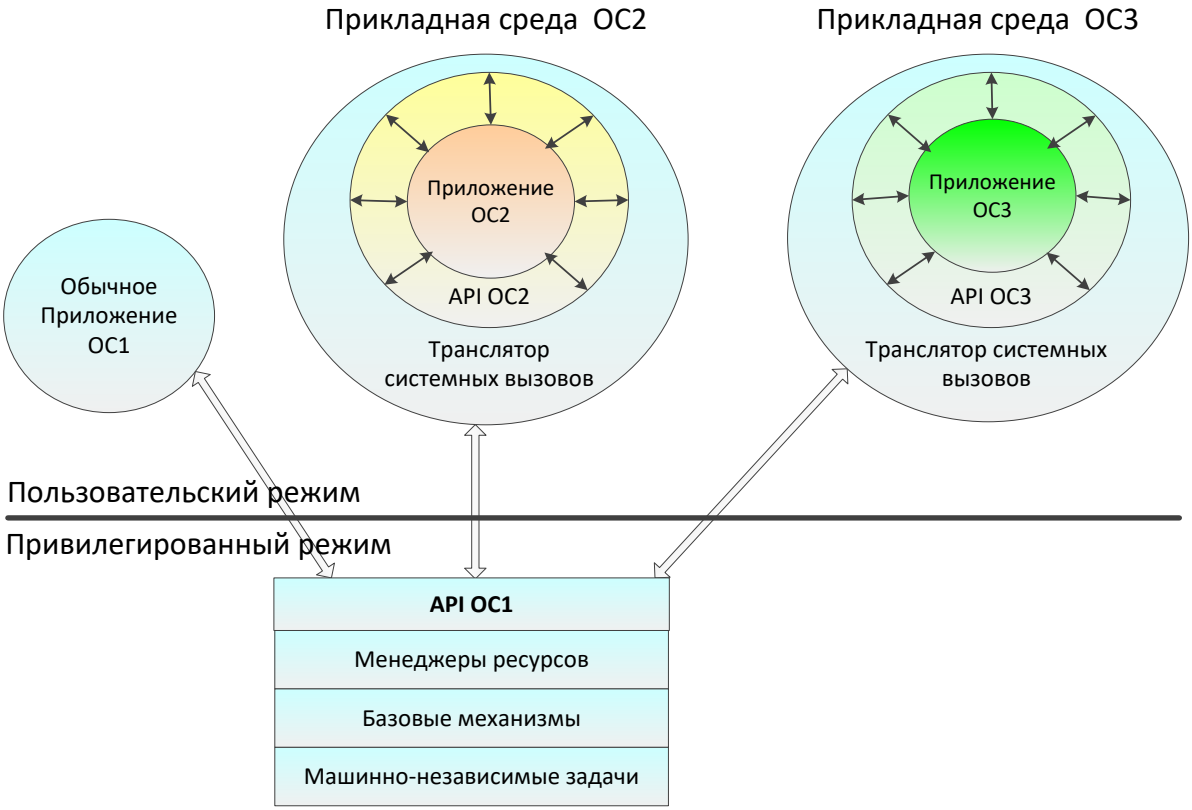


Рис. 1.12. Прикладные программные среды, транслирующие системные вызовы

2. Реализация множественных прикладных сред операционная система через несколько равноправных прикладных программных интерфейсов. В приведенном на рис. 1.13 примере операционная система поддерживает приложения, написанные для ОС1, ОС2 и ОС3. Для этого непосредственно в пространстве ядра системы размещены прикладные программные интерфейсы всех этих ОС: API ОС1, API ОС2 и API ОС3.

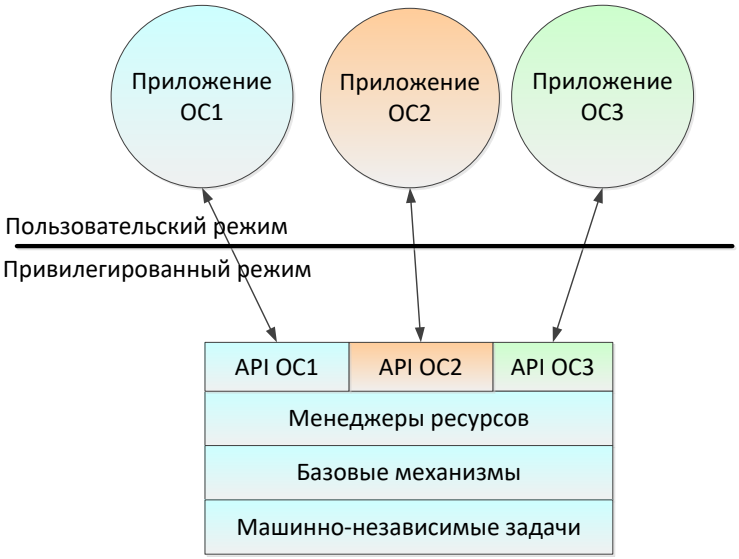


Рис. 1.13. Реализация совместимости на основе нескольких равноправных API

3. Построение множественных прикладных сред по микроядерному подходу. В соответствии с микроядерной архитектурой все функции ОС реализуются микроядром и серверами пользовательского режима. Важно, что каждая прикладная среда оформляется в виде отдельного сервера пользовательского режима и не включает базовых механизмов (рис. 1.14). Приложения, используя API, обращаются с системными вызовами к соответствующей прикладной среде через микроядро. Прикладная среда обрабатывает запрос, выполняет его (возможно, обращаясь для этого за помощью к базовым функциям микроядра) и отсылает приложению результат.

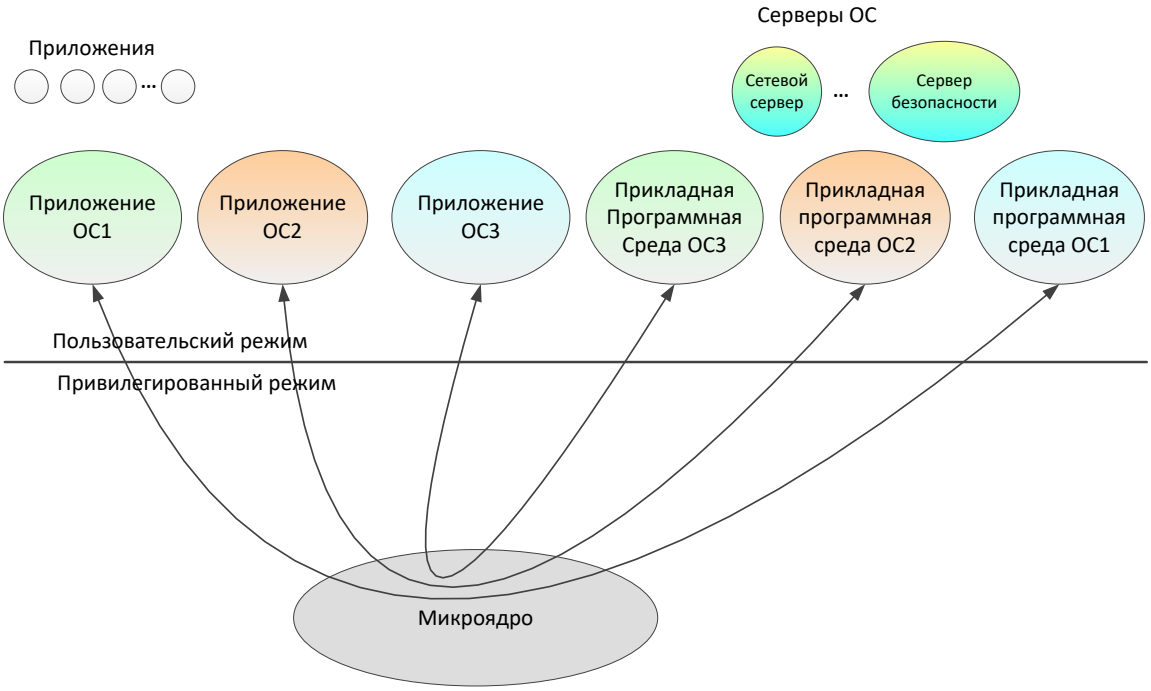


Рис. 1.14. Микроядерный подход к реализации множественных прикладных сред

2. Управление процессами

2.1. Понятие процесса и потока

Важнейшей функцией операционной системы является организация рационального использования всех ее аппаратных и информационных ресурсов. К основным ресурсам могут быть отнесены процессоры, память, внешние устройства, данные и программы. Располагающая одними и теми же аппаратными ресурсами, но управляемая различными ОС, вычислительная система может работать с разной степенью эффективности. Хотя и в однопрограммной ОС необходимо решать задачи управления ресурсами (например, распределение памяти между приложением и ОС), главные сложности возникают в мультипрограммных ОС, в которых за ресурсы конкурируют сразу несколько приложений. Именно поэтому большая часть всех проблем, рассматриваемых в этом материале, относится к мультипрограммным системам.

Мультипрограммирование, или *многозадачность* (multitasking), – это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ. Эти программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, данные. Мультипрограммирование призвано повысить эффективность использования вычислительной системы.

Одной из основных подсистем мультипрограммной ОС, непосредственно влияющей на функционирование вычислительной машины, является *подсистема управления процессами и потоками,* которая занимается их созданием и уничтожением, поддерживает взаимодействие между ними, а также распределяет процессорное время между несколькими одновременно существующими в системе процессами и потоками. Подсистема управления процессами и потоками ответственна за обеспечение процессов необходимыми ресурсами. ОС поддерживает в памяти специальные информационные структуры, в которые записывает, какие ресурсы выделены каждому процессу. Она может назначить процессу ресурсы в единоличное пользование или в совместное пользование с другими процессами. Некоторые из ресурсов выделяются процессу при его создании, а некоторые – динамически по запросам во время выполнения. Ресурсы могут быть приписаны процессу на все время его жизни или только на определенный период.

В настоящее время в большинстве операционных систем определены два типа единиц работы. Более крупная единица работы, обычно носящая название процесса, или задачи, требует для своего выполнения нескольких более мелких работ, для обозначения которых используют термины «поток», или «нить».

Процесс (*задача*) – заявка к ОС на потребление ресурсов вычислительной системы необходимых для функционирования приложения.

Многопоточная обработка (*multithreading*) – механизм распараллеливания вычислений, учитывающий тесные связи между отдельными ветвями вычислений одного и того же приложения.

Понятию «*поток*» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные.

Мультипрограммирование более эффективно на уровне потоков, а не процессов. Задача, оформленная в виде нескольких потоков в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной системе) выполнения ее отдельных частей. Например, если электронная таблица была разработана с учетом возможностей многопоточной обработки, то пользователь может запросить пересчет своего рабочего листа и одновременно продолжать заполнять таблицу.

2.2. Управление процессами и потоками

Создание процесса включает загрузку кодов и данных исполняемой программы этого процесса с диска в оперативную память. Для этого ОС должна обнаружить местоположение такой программы на диске, перераспределить оперативную память и выделить память исполняемой программе нового процесса. Затем необходимо считать программу в выделенные для нее участки памяти и, возможно, изменить параметры программы в зависимости от размещения в памяти. Создание описателя процесса знаменует собой появление в системе еще одного претендента на вычислительные ресурсы. Начиная с этого момента при распределении ресурсов ОС должна принимать во внимание потребности нового процесса.

В многопоточной системе при создании процесса ОС создает для каждого процесса как минимум один поток выполнения. При создании потока так же, как и при создании процесса, операционная система генерирует специальную информационную структуру – описатель потока, который содержит идентификатор потока, данные о правах доступа и приоритете, о состоянии потока и другую информацию.

Создание процесса – создание описателя процесса, в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе, необходимые операционной системе для управления им. В число таких сведений могут входить, например, иденти-

фикатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности процесса (приоритет и права доступа) и т. п.

На протяжении существования процесса выполнение его потоков может быть многократно прервано и продолжено. (В системе, не поддерживающей потоки, все сказанное ниже о планировании и диспетчеризации относится к процессу в целом.) Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации.

2.2.1. Планирование

Планирование – работа по определению того, в какой момент необходимо прервать выполнение текущего активного потока и какому потоку предоставить возможность выполняться.

Планирование, по существу, включает в себя решение двух задач:

1. определение момента времени для смены текущего активного потока;
2. выбор для выполнения потока из очереди готовых потоков.

Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании могут приниматься во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращений к вводу-выводу и другие факторы.

Существует множество различных алгоритмов планирования потоков, по-своему решающих каждую из приведенных выше задач. Алгоритмы планирования могут преследовать различные цели и обеспечивать разное качество мультипрограммирования.

В настоящее время основными наиболее общими используемыми алгоритмами планирования процессов являются динамическое и статистическое планирование.

Динамическое планирование, когда решения принимаются во время работы системы на основе анализа текущей ситуации. ОС работает в условиях неопределенности – потоки и процессы появляются в случайные моменты времени и также непредсказуемо завершаются. Динамические планировщики могут гибко приспосабливаться к изменяющейся ситуации и не используют никаких предположений о мультипрограммной смеси.

Динамическое планирование делится на:

- вытесняющие алгоритмы планирования;
- невытесняющие алгоритмы планирования.

Статическое планирование – используется в специализированных системах, в которых весь набор одновременно выполняемых задач определен заранее, например, в системах реального времени. В этом случае планировщик принимает решения о планировании не во время работы системы, а заранее.

Соотношение между динамическим и статическим планировщиками аналогично соотношению между диспетчером железной дороги, который

пропускает поезда строго по предварительно составленному расписанию, и регулировщиком на перекрестке автомобильных дорог, не оснащенном светофорами, который решает, какую машину остановить, а какую пропустить, в зависимости от ситуации на перекрестке.

2.2.2. Диспетчеризация

Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой.

Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем чтобы впоследствии использовать эту информацию для последующего возобновления выполнения этого потока.

Контекст содержит:

- состояние аппаратуры компьютера в момент прерывания потока (значение счетчика команд, содержимое регистров общего назначения, режим работы процессора, флаги, маски прерываний и другие параметры);
- параметры операционной среды (ссылки на открытые файлы, данные о незавершенных операциях ввода-вывода, коды ошибок, выполняемых этим потоком системных вызовов и т. д.).

Диспетчеризация сводится к следующему:

- сохранение контекста текущего потока, который требуется сменить;
- загрузка контекста нового потока, выбранного в результате планирования;
- запуск нового потока на выполнение.

Поскольку операция переключения контекстов существенно влияет на производительность вычислительной системы, программные модули ОС выполняют диспетчеризацию потоков совместно с аппаратными средствами процессора.

2.2.3. Состояния потока

ОС выполняет планирование потоков, принимая во внимание их состояние. В мультипрограммной системе поток может находиться в одном из трех основных состояний:

1. *выполнение* – активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
2. *ожидание* – пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса);

3. *готовность* – также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

В течение своей жизни каждый поток переходит из одного состояния в другое в соответствии с алгоритмом планирования потоков, принятым в конкретной операционной системе. Типичный граф, соответствующий поведению процесса в системе приведен на рис. 2.1.

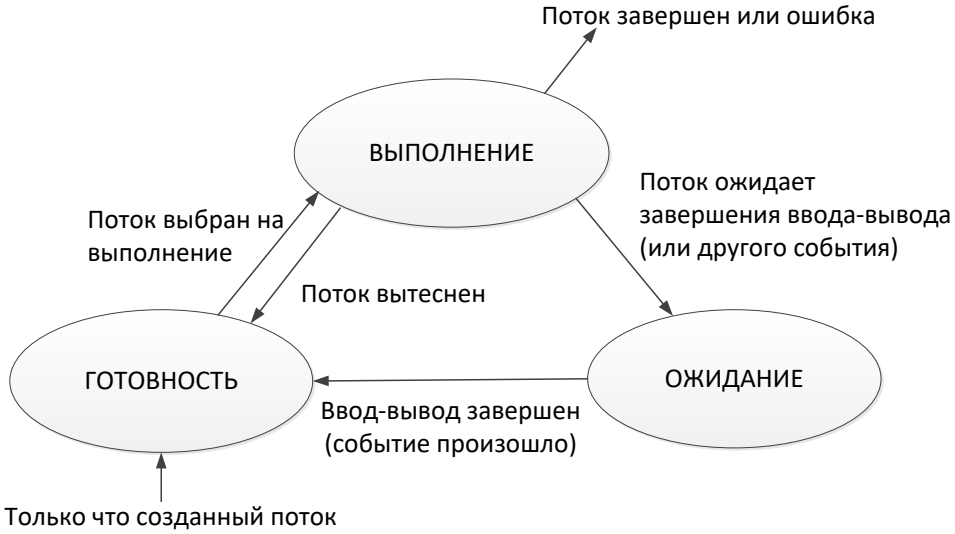


Рис. 2.1. Граф состояний потока в многозадачной среде

Только что созданный поток находится в состоянии готовности, он готов к выполнению и. стоит в очереди к процессору. Когда в результате планирования подсистема управления потоками принимает решение об активизации этого потока, он переходит в состояние выполнения и находится в нем до тех пор, пока либо он сам освободит процессор (перейдя в состояние ожидания какого-нибудь события), либо будет принудительно «вытеснен» из процессора (например, вследствие исчерпания кванта процессорного времени, отведенного этому потоку). В последнем случае поток возвращается в состояние готовности. В это же состояние поток переходит из состояния ожидания, после того как ожидаемое событие произойдет.

2.3. Алгоритмы планирования процессов

2.3.1. Вытесняющие и не вытесняющие алгоритмы планирования

С самых общих позиций множество алгоритмов планирования можно разделить на два класса:

1. *Вытесняющие* (*preemptive*) *алгоритмы* – это такие способы планирования потоков, в которых решение о переключении процес-

сора с выполнения одного потока на выполнение другого потока принимается операционной системой, а не активной задачей. При вытесняющем мультипрограммировании функции планирования потоков целиком сосредоточены в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться одновременно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активного потока, запоминает его контекст, выбирает из очереди готовых потоков следующий, запускает новый поток на выполнение, загружая его контекст.

2. *Невытесняющие* (*non-preemptive*) *алгоритмы* основаны на том, что активному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению поток. При невытесняющем мультипрограммировании механизм планирования распределен между операционной системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения очередного цикла своего выполнения и только затем передает управление ОС с помощью какого-либо системного вызова. ОС формирует очереди потоков и выбирает в соответствии с некоторым правилом (например, с учетом приоритетов) следующий поток на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков приложений.

Основным различием между вытесняющими и не вытесняющими алгоритмами является степень централизации механизма планирования потоков.

Почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows, Linux), реализованы вытесняющие алгоритмы планирования потоков (процессов).

2.3.2. Концепция квантования

В основе многих вытесняющих алгоритмов планирования лежит *концепция квантования.* В соответствии с ней каждому потоку поочередно для выполнения предоставляется ограниченный непрерывный период процессорного времени – *квант.*

Смена активного потока происходит, если:

- поток завершился и покинул систему;
- произошла ошибка;
- поток перешел в состояние ожидания;
- исчерпан квант процессорного времени, отведенный этому потоку.

Поток, который исчерпал свой квант, переводится в состояние готовности и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый поток из очереди готовых. Граф состояний потока, изображенный на рис. 2.1, соответствует алгоритму планирования, основанному на квантовании.

Кванты, выделяемые потокам, могут быть одинаковыми для всех потоков или различными.

2.3.3. Приоритетные алгоритмы планирования

Другой важной концепцией, лежащей в основе многих вытесняющих алгоритмов планирования, является приоритетное обслуживание. Приоритетное обслуживание предполагает наличие у потоков некоторой изначально известной характеристики – приоритета, на основании которой определяется порядок их выполнения.

Приоритет – это число, характеризующее степень привилегированности потока при использовании ресурсов вычислительной машины, в частности процессорного времени: чем выше приоритет, тем выше привилегии, тем меньше времени будет проводить поток в очередях.

Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что приоритет потока тем выше, чем больше (в арифметическом смысле) число, обозначающее приоритет. В других системах, наоборот, чем меньше число, тем выше приоритет.

В большинстве операционных систем, поддерживающих потоки, приоритет потока непосредственно связан с приоритетом процесса, в рамках которого выполняется этот поток. Приоритет процесса назначается операционной системой при его создании. Значение приоритета включается в описатель процесса и используется при назначении приоритета потокам этого процесса.

При назначении приоритета вновь созданному процессу ОС учитывает, является этот процесс системным или прикладным, каков статус пользователя, запустившего процесс, и др.

Во многих ОС предусматривается возможность изменения приоритетов в течение жизни потока. Изменение приоритета могут происходить по инициативе самого потока, когда он обращается с соответствующим вызовом к операционной системе, или по инициативе пользователя, когда он выполняет соответствующую команду. Кроме того, ОС сама может изменять приоритеты потоков в зависимости от ситуации, складывающейся в системе. В последнем случае приоритеты называются динамическими в отличие от неизменяемых, фиксированных, приоритетов.

От того, какие приоритеты назначены потокам, существенно зависит эффективность работы всей вычислительной системы. В современных ОС во избежание разбалансировки системы, которая может возникнуть при

неправильном назначении приоритетов, возможности пользователей влиять на приоритеты процессов и потоков стараются ограничивать.

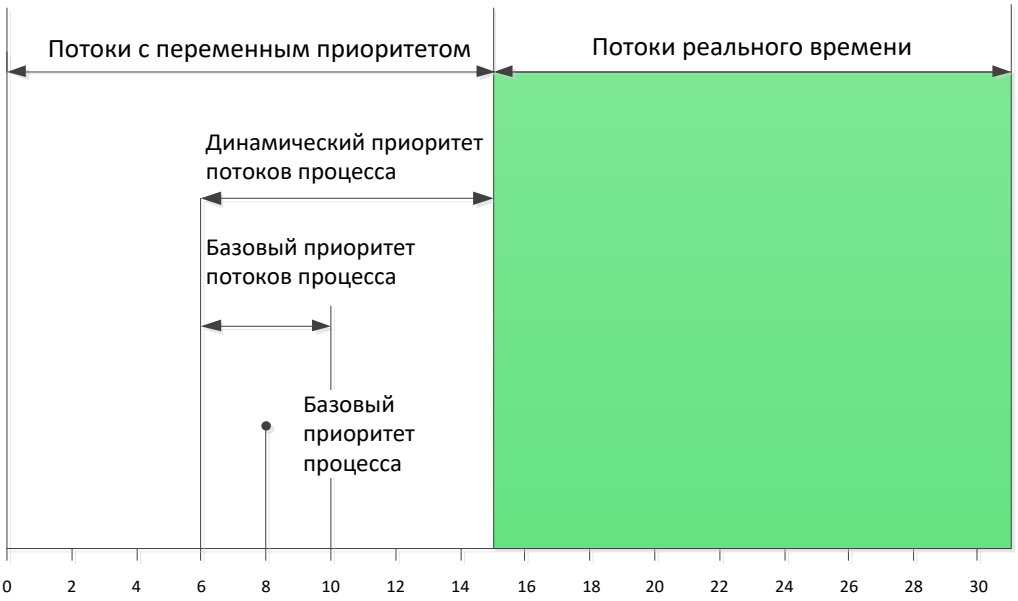


Рис. 2.2. Схема назначения приоритетов в ОС семейства Windows

Существуют две разновидности *приоритетного планирования*:

- *обслуживание с относительными приоритетами.* В системах с относительными приоритетами активный поток выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ожидания (или же произойдет ошибка, или поток завершится);
- *обслуживание с абсолютными приоритетами.* В системах с абсолютными приоритетами выполнение активного потока прерывается кроме указанных выше причин, еще при одном условии: если в очереди готовых потоков появился поток, приоритет которого выше приоритета активного потока.

2.3.4. Смешанные алгоритмы планирования

Во многих операционных системах алгоритмы планирования построены с использованием как концепции квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора потока из очереди готовых определяется приоритетами потоков. Именно так реализовано планирование в системах Windows, в которой квантование сочетается с динамическими абсолютными приоритетами. На выполнение выбирается готовый поток с наивысшим приоритетом. Ему выделяется квант времени. Если во время выполнения в очереди готовых появляется поток с более высоким приоритетом, то он вытесняет выполняемый поток. Вытесненный поток возвращается в очередь готовых, причем он становится впереди всех остальных потоков, имеющих такой же приоритет.

Процессы реального времени также используют стратегию фиксированных приоритетов, но пользователь может их изменять. Так как при наличии готовых к выполнению процессов реального времени другие процессы не рассматриваются, то процессы реального времени надо тщательно проектировать, чтобы они не захватывали процессор на слишком долгое время. Характеристики планирования процессов реального времени включают две величины: уровень глобального приоритета и квант времени. Для каждого уровня приоритета по умолчанию имеется своя величина кванта времени. Процессу разрешается захватывать процессор на указанный квант времени, а по его истечении планировщик снимает процесс с выполнения.

2.4. Синхронизация процессов и потоков

Существует достаточно обширный класс средств операционной системы, с помощью которых обеспечивается взаимная синхронизация процессов и потоков.

Потребность в синхронизации потоков возникает только в мультипрограммной операционной системе и связана с совместным использованием аппаратных и информационных ресурс об вычислительной системы. Синхронизация необходима для исключения гонок и тупиков при обмене данными между потоками, разделении данных, при доступе к процессору и устройствам ввода-вывода.

Рассмотрим, например (рис. 2.3), задачу ведения базы данных клиентов некоторого предприятия. Каждому клиенту отводится отдельная запись в базе данных, в которой среди прочих полей имеются поля Заказ и Оплата. Программа, ведущая базу данных, оформлена как единый процесс, имеющий несколько потоков, в том числе поток *A*, который заносит в базу данных информацию о заказах, поступивших от клиентов, и поток *B*, который фиксирует в базе данных сведения об оплате клиентами выставленных счетов. Оба эти потока совместно работают над общим файлом базы данных, используя однотипные алгоритмы, включающие три шага.

1. Считать из файла базы данных в буфер запись о клиенте с заданным идентификатором.
2. Внести новое значение в поле Заказ (для потока *A*) или Оплата (для потока *B*).
3. Вернуть модифицированную запись в файл базы данных.

Обозначим соответствующие шаги для потока *A* как A_1, A_2 и A_3, а для потока *B* как B_1, B_2 и B_3. Предположим, что в некоторый момент поток *A* обновляет поле Заказ записи о клиенте *N*. Для этого он считывает эту запись в свой буфер (шаг A_1), модифицирует значение поля Заказ (шаг A_2), но внести запись в базу данных (шаг *A*3) не успевает, так как его выполнение прерывается, например, вследствие завершения кванта времени.

Предположим также, что потоку В также потребовалось внести сведения об оплате относительно того же клиента *N*. Когда подходит очередь

потока *B*, он успевает считать запись в свой буфер (шаг B_1) и выполнить обновление поля Оплата (шаг B_2), а затем прерывается. Заметим, что в буфере у потока *B* находится запись о клиенте *N*, в которой поле Заказ имеет прежнее, не измененное значение.

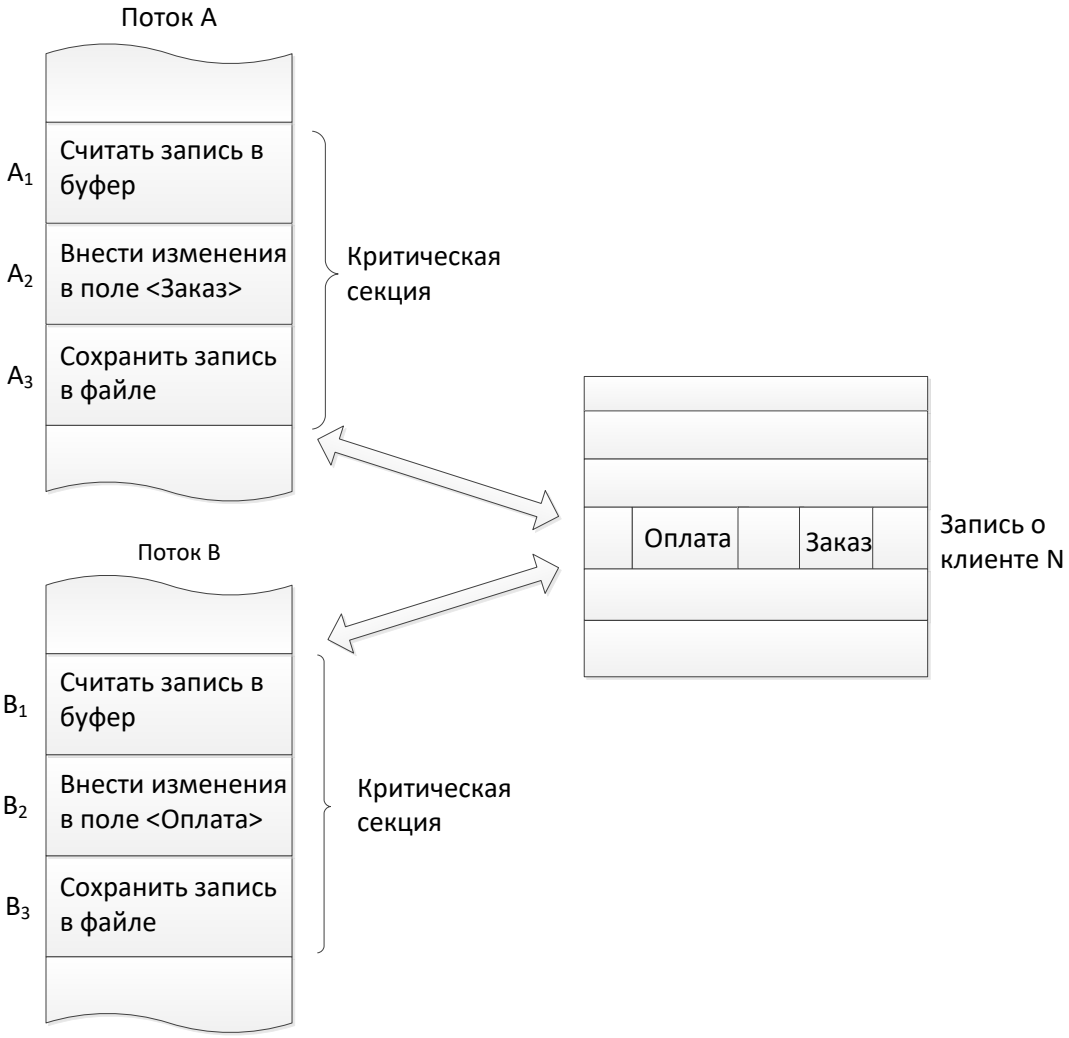


Рис. 2.3. Возникновение гонок при доступе к разделяемым данным

Когда в очередной раз управление будет передано потоку *A*, то он, продолжая свою работу, запишет запись о клиенте *N* с модифицированным полем Заказ в базу данных (шаг A_3). После прерывания потока *A* и активизации потока *B* последний запишет в базу данных поверх только что обновленной записи о клиенте *N* свой вариант записи, в которой обновлено значение поля Оплата. Таким образом, в базе данных будут зафиксированы сведения о том, что клиент *N* произвел оплату, но информация о его заказе окажется потерянной (рис. 2.4, а).

Сложность проблемы синхронизации кроется в нерегулярности возникающих ситуаций. Так, в предыдущем примере можно представить и другое развитие событий: могла быть потеряна информация не о заказе, а об оплате (рис. 2.4, б) или, напротив, все исправления были успешно внесены (рис. 2.4, в).

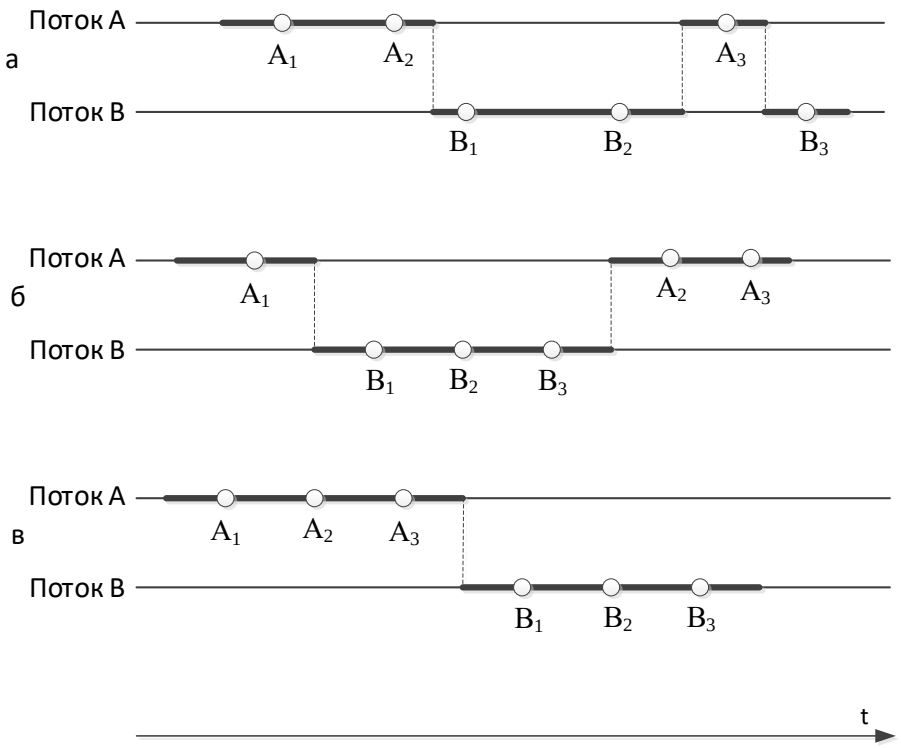


Рис. 2.4. Влияние относительных скоростей потоков на результат решения задачи

Все определяется взаимными скоростями потоков и моментами их прерывания. Поэтому отладка взаимодействующих потоков является сложной задачей.

Гонка – когда два или более потоков обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей потоков (ситуации, рассмотренные выше).

Тупики – состояние системы, когда два или более процессов взаимно блокируют друг друга.

Рассмотрим пример тупика. Пусть двум процессам, выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например, принтер и диск. На рис. 2.5 а показаны фрагменты соответствующих программ. И пусть после того, как процесс *A* занял принтер (установил блокирующую переменную), он был прерван. Управление получил процесс *B*, который сначала занял диск, но при выполнении следующей команды был заблокирован, так как принтер оказался уже занятым процессом *A*. Управление снова получил процесс *A*, который в соответствии со своей программой сделал попытку занять диск и был заблокирован: диск уже распределен процессу *B*. В таком положении процессы *A* и *B* могут находиться сколь угодно долго.

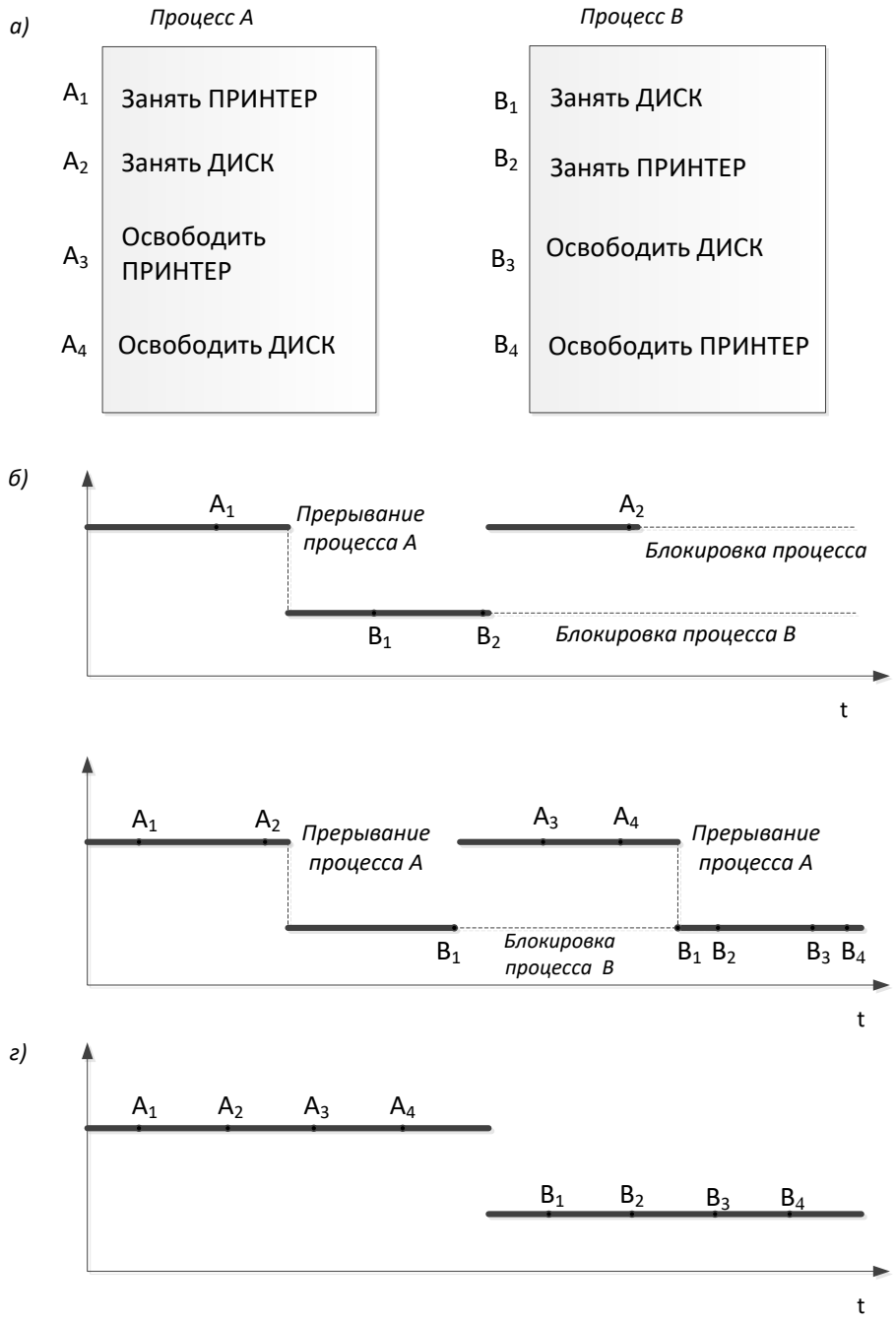


Рис. 2.5. а) фрагменты программ *A* и *B*, разделяющих принтер и диск; б) взаимная блокировка (клинч); в) очередь к разделяемому диску; г) независимое использование ресурсов.

2.4.1. Критическая секция

Критическая секция – это часть программы, результат выполнения которой может непредсказуемо меняться, если переменные, относящиеся к этой части программы, изменяются другими потоками в то время, когда выполнение этой части еще не завершено.

Критическая секция всегда определяется по отношению к определенным критическим данным, при несогласованном изменении которых

могут возникнуть нежелательные эффекты. В предыдущем примере такими критическими данными являлись записи файла базы данных. Во всех потоках, работающих с критическими данными, должна быть определена критическая секция.

Чтобы исключить эффект гонок по отношению к критическим данным, необходимо обеспечить, чтобы в каждый момент времени в критической секции, связанной с этими данными, находился только один поток. При этом неважно, находится этот поток в активном или в приостановленном состоянии. Этот прием называют взаимным исключением. Операционная система использует разные способы реализации взаимного исключения. Некоторые способы пригодны для взаимного исключения при вхождении в критическую секцию только потоков одного процесса, в то время как другие могут обеспечить взаимное исключение и для потоков разных процессов.

2.4.2. Блокирующие переменные

Для синхронизации потоков одного процесса прикладной программист может использовать *глобальные блокирующие переменные.* С этими переменными, к которым все потоки процесса имеют прямой доступ, программист работает, не обращаясь к системным вызовам ОС.

Каждому набору критических данных ставится в соответствие двоичная переменная, которой поток присваивает значение 0, когда он входит в критическую секцию, и значение 1, когда он ее покидает.

На рис. 2.6 показан фрагмент алгоритма потока, использующего для реализации взаимного исключения доступа к критическим данным D блокирующую переменную $F(D)$. Перед входом в критическую секцию поток проверяет, не работает ли уже какой-нибудь поток с данными D. Если переменная $F(D)$ установлена в 0, то данные заняты и проверка циклически повторяется. Если же данные свободны ($F(D)=1$), то значение переменной $F(D)$ устанавливается в 0 и поток входит в критическую секцию. После того как поток выполнит все действия с данными D, значение переменной $F(D)$ снова устанавливается равным 1.

Блокирующие переменные могут использоваться не только при доступе к разделяемым данным, но и при доступе к разделяемым ресурсам любого вида.

Если все потоки написаны с учетом вышеописанных соглашений, то взаимное исключение гарантируется. При этом потоки могут быть прерваны операционной системой в любой момент и в любом месте, в том числе в критической секции.

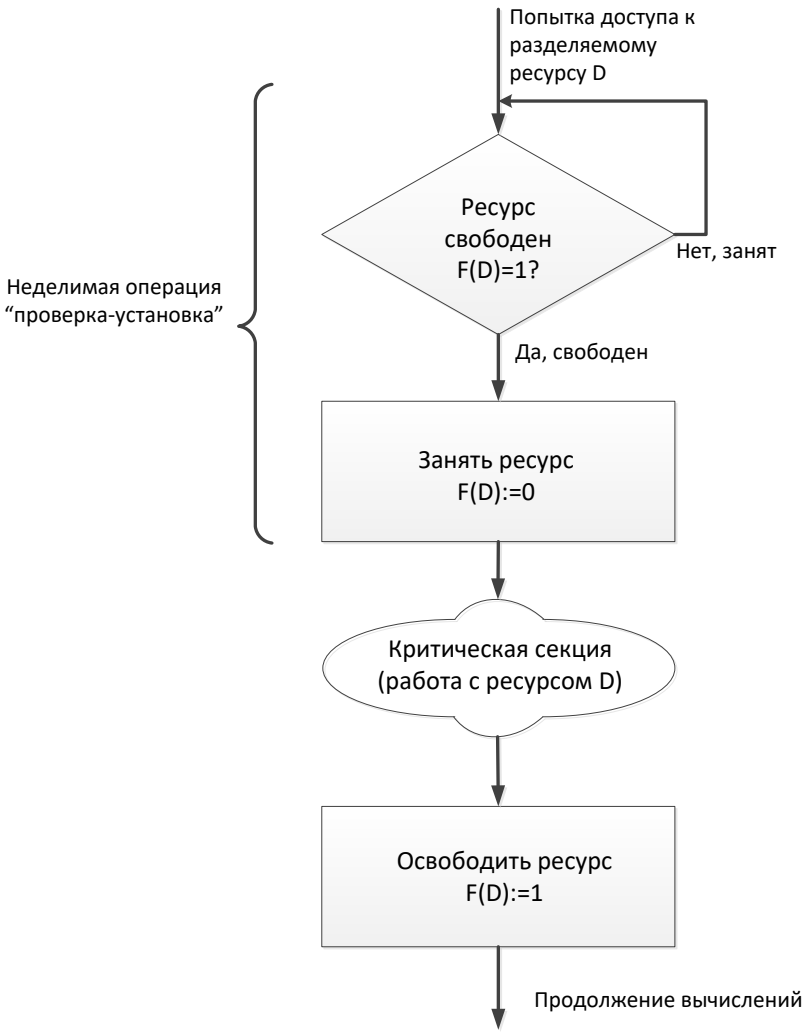


Рис. 2.6. Реализация критических секций с использованием блокирующих переменных

2.4.3. Семафоры

Обобщением блокирующих переменных являются так называемые *семафоры Дейкстры*. Вместо двоичных переменных Дейкстра (Dijkstra) предложил использовать переменные, которые могут принимать целые неотрицательные значения. Такие переменные, используемые для синхронизации вычислительных процессов, получили название *семафоров.*

Для работы с семафорами вводятся два примитива *POST* и *WAIT*, традиционно обозначаемых *P* и *V*. Пусть переменная S представляет собой семафор. Тогда действия *V*(*S*) и *P*(*S*) определяются следующим образом.

– *V*(*S*): переменная *S* увеличивается на 1 единым действием. Выборка, наращивание и запоминание не могут быть прерваны. К переменной S нет доступа другим потокам во время выполнения этой операции.

– *P*(*S*): уменьшение *S* на 1, если это возможно. Если *S*=0 и невозможно уменьшить *S*, оставаясь в области целых неотрицательных значений, то в этом случае поток, вызывающий операцию *P*, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также являются неделимой операцией.

Никакие прерывания во время выполнения примитивов *V* и *P* недопустимы.

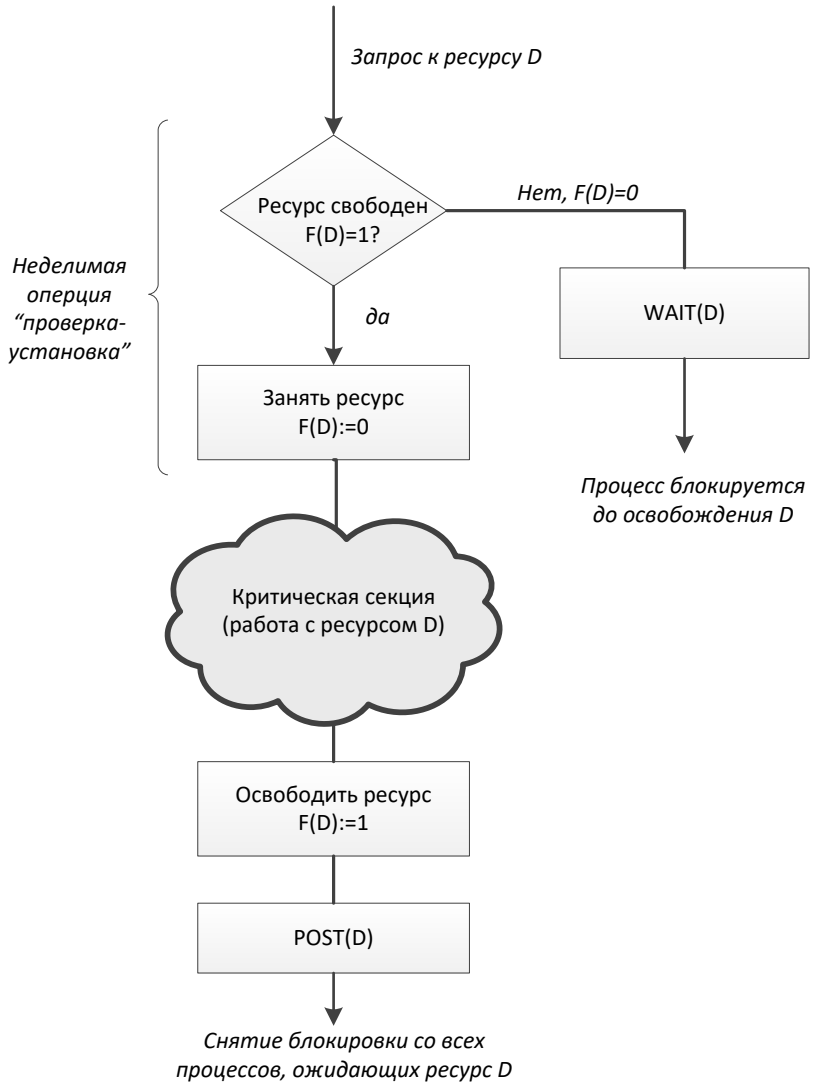


Рис. 2.7. Реализация критической секции с использованием системных функций *WAIT*(*D*) и *POST*(*D*)

В частном случае, когда семафор *S* может принимать только значения 0 и 1, он превращается в блокирующую переменную, которую по этой причине часто называют двоичным семафором. Операция *P* заключает в себе потенциальную возможность перехода потока, который ее выполняет, в состояние ожидания, в то время как операция *V* может при некоторых обстоятельствах активизировать другой поток, приостановленный операцией *P*.

Рассмотрим использование семафоров на классическом примере взаимодействия двух процессов, выполняющихся в режиме мультипрограммирования, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула. Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. Процесс «писатель» должен приостанавливаться, когда все буфера оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, процесс «читатель» приостанавливается, когда все буферы пусты, и активизируется при появлении хотя бы одной записи.

3. Управление памятью

3.1. Иерархия памяти

Память вычислительной машины представляет собой иерархию запоминающих устройств (ЗУ), отличающихся средним временем доступа к данным, объемом и стоимостью хранения одного бита (рис. 3.1)

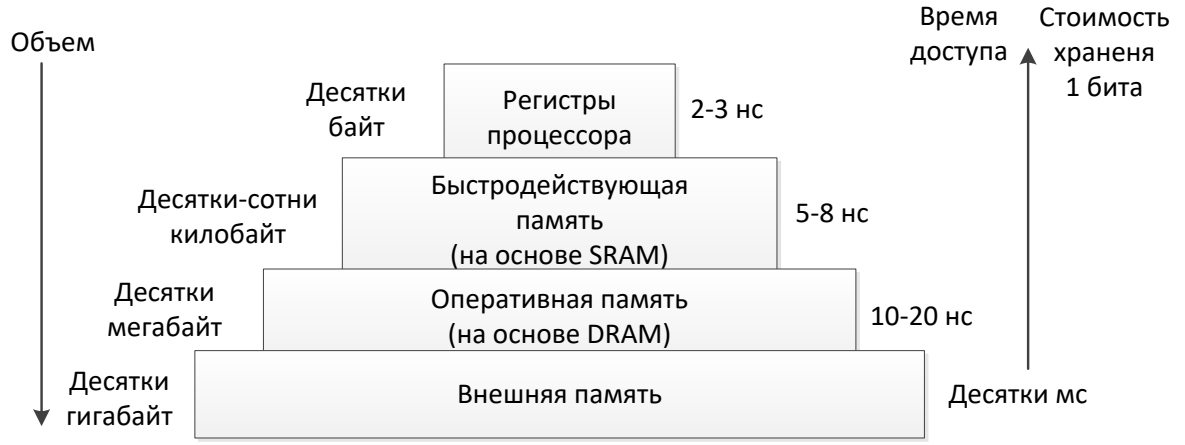


Рис. 3.1. Иерархия запоминающих устройств

Фундаментом этой пирамиды запоминающих устройств служит *внешняя память,* как правило, представляемая *жестким диском*. Она имеет большой объем (десятки и сотни гигабайт), но скорость доступа к данным является невысокой. Время доступа к диску измеряется миллисекундами.

На следующем уровне располагается более быстродействующая (время доступа[1] равно примерно 10-20 нс) и менее объемная (от десятков мегабайт до нескольких гигабайт) *оперативная память*, реализуемая на относительно медленной динамической памяти DRAM.

Для хранения данных, к которым необходимо обеспечить быстрый доступ, используются компактные быстродействующие запоминающие устройства на основе *статической памяти SRAM* (*Кэш*), объем которых составляет от нескольких десятков до нескольких сотен килобайт, а время доступа к данным обычно не превышает 8 нс.

Кэш-память, или просто *кэш* (*cache*) – это способ совместного функционирования двух типов запоминающих устройств, который за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ позволяет, с одной стороны, уменьшить среднее время доступа к данным, а с другой стороны, экономить более дорогую быстродействующую память.

Верхушку в этой пирамиде составляют *внутренние регистры процессора,* которые также могут быть использованы для промежуточного хранения данных. Общий объем регистров составляет несколько десятков

байт, а время доступа определяется быстродействием процессора и равно в настоящее время примерно 2-3 нс.

Все перечисленные характеристики ЗУ быстро изменяются по мере совершенствования вычислительной аппаратуры. В данном случае важны не абсолютные значения времени доступа или объема памяти, а их соотношение для разных типов запоминающих устройств.

Таким образом, можно констатировать печальную закономерность – чем больше объем устройства, тем менее быстродействующим оно является. Более того, стоимость хранения данных в расчете на один бит также увеличивается с ростом быстродействия устройств. Однако пользователю хотелось бы иметь и недорогую, и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

3.2. Управление памятью

Под *памятью* (memory) как правило подразумевается оперативная память компьютера. В отличие от памяти жесткого диска, которую называют внешней памятью (storage), оперативной памяти для сохранения информации требуется постоянное электропитание.

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Особая роль памяти объясняется тем, что процессор может выполнять инструкции протравы только в том случае, если они находятся в памяти. Память распределяется как между модулями прикладных программ, так и между модулями самой операционной системы.

Функциями ОС по управлению памятью в мультипрограммной системе являются:

- отслеживание свободной и занятой памяти;
- выделение памяти процессам и освобождение памяти по завершении процессов;
- вытеснение кодов и данных процессов из оперативной памяти на диск (полное или частичное), когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;
- настройка адресов программы на конкретную область физической памяти.

Помимо первоначального выделения памяти процессам при их создании ОС должна также заниматься *динамическим распределением памяти*, то есть выполнять запросы приложений на выделение им дополнительной памяти во время выполнения. После того как приложение перестает нуждаться в дополнительной памяти, оно может возвратить ее системе. Выделение памяти случайной длины в случайные моменты времени из общего пула памяти приводит к фрагментации и, вследствие этого, к неэффективному ее использованию. *Дефрагментация памяти* тоже является функцией операционной системы.

3.3. Типы адресации

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются *символьные имена* (метки), *виртуальные адреса* и *физические адреса* (рис. 3.2):

- *символьные имена* присваивает пользователь при написании программы на алгоритмическом языке или ассемблере;
- *виртуальные адреса*, называемые иногда математическими, или логическими адресами, вырабатывает транслятор, переводящий программу на машинный язык. Поскольку во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что начальным адресом программы будет нулевой адрес;
- *физические адреса* соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды.

Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством.*

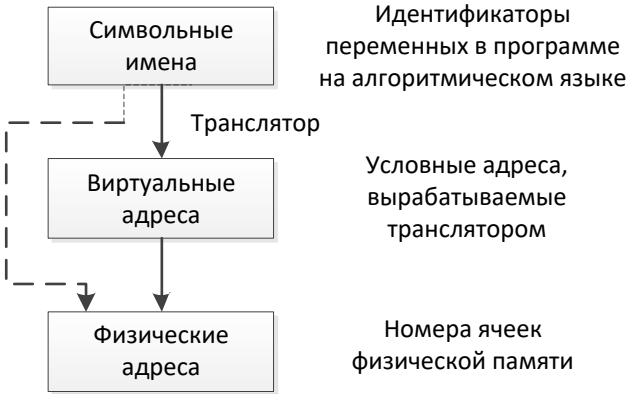


Рис. 3.2. Типы адресов

Диапазон возможных адресов виртуального пространства у всех процессов является одним и тем же. Например, при использовании 32-разрядных виртуальных адресов этот диапазон задается границами 00000000_{16} и $FFFFFFFF_{16}$. Тем не менее каждый процесс имеет собственное виртуальное адресное пространство – транслятор присваивает виртуальные адреса переменным и кодам каждой программе независимо (рис. 3.3). Совпадение виртуальных адресов переменных и команд различных процессов не приводит к конфликтам, так как в том случае, когда эти переменные одновременно присутствуют в памяти, операционная система отображает их на разные физические адреса.

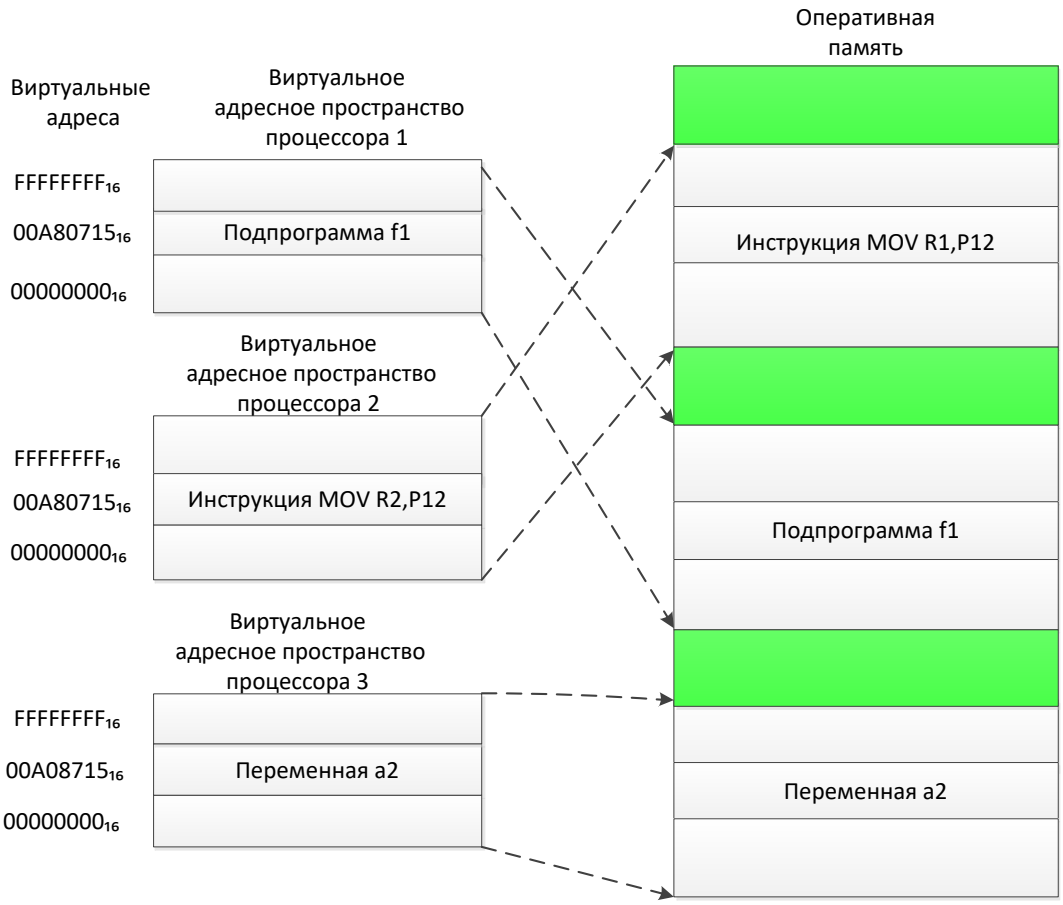
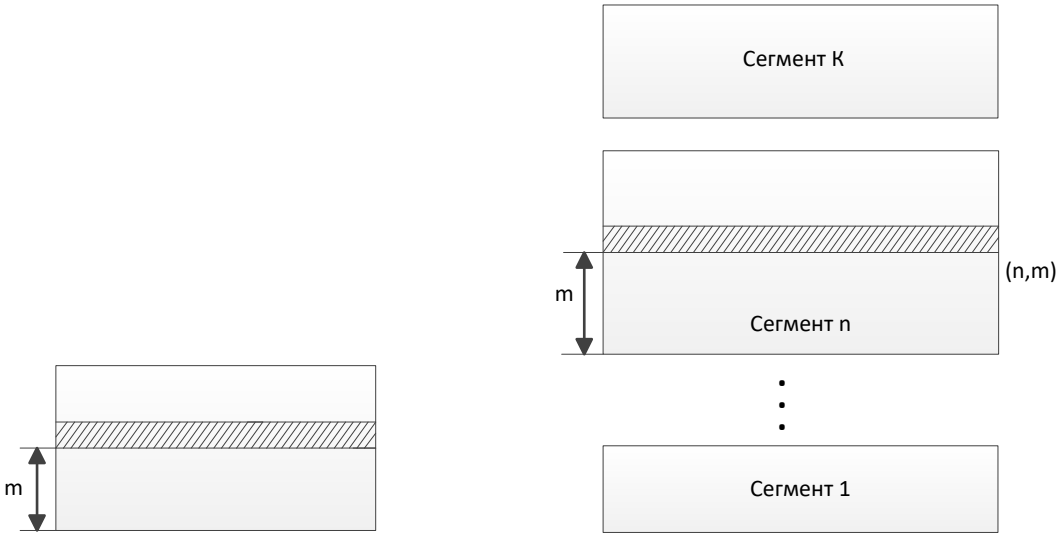


Рис. 3.3. Виртуальные адресные пространства нескольких программ

В разных операционных системах используются разные способы структуризации виртуального адресного пространства:

- в одних ОС виртуальное адресное пространство процесса подобно физической памяти представлено в виде непрерывной линейной последовательности виртуальных адресов. Такую структуру адресного пространства называют также *плоской* (*flat*). При этом виртуальным адресом является единственное число, представляющее собой смещение, относительно начала (обычно это значение 000...000) виртуального адресного пространства (рис. 3.4а). Адрес такого типа называют *линейным виртуальным адресом*;
- в других ОС виртуальное адресное пространство делится на части, называемые *сегментами* (или секциями, или областями, или другими терминами). В этом случае помимо линейного адреса может быть использован виртуальный адрес, представляющий собой пору чисел (*u, m*), где *n* определяет сегмент, а *m* – смещение внутри сегмента (рис. 3.4б).



а. б.

Рис. 3.4. Типы виртуальных адресных пространств:
плоское (а), сегментированное (б)

Существуют и более сложные способы структуризации виртуального адресного пространства, когда виртуальный адрес образуется тремя или даже более числами.

Задачей операционной системы является отображение индивидуальных виртуальных адресных пространств всех одновременно выполняющихся процессов на общую физическую память.

Существуют два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические.

1. Замена виртуальных адресов на физические выполняется один раз для каждого процесса во время начальной загрузки программы в память. Специальная системная программа – перемещающий загрузчик – на основании имеющихся у нее исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставленной транслятором об адресно-зависимых элементах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.
2. Программа загружается в память в неизмененном виде в виртуальных адресах, то есть операнды инструкций и адреса переходов имеют те значения, которые выработал транслятор. В наиболее простом случае, когда виртуальная и физическая память процесса представляют собой единые непрерывные области адресов, операционная система выполняет преобразование виртуальных адресов в физические по следующей схеме.

Во втором случае при загрузке операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразо-

вание виртуального адреса в физический. Схема такого преобразования показана на рис. 3.5.

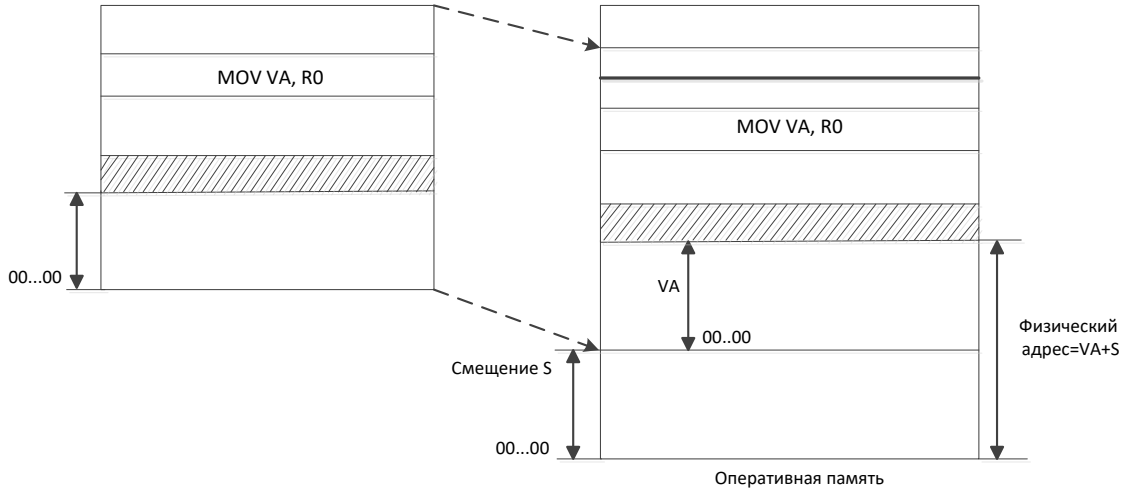


Рис. 3.5. Схема динамического преобразования адресов

Пусть, например, некоторая программа, работающая под управлением этой ОС, загружена в физическую память начиная с физического адреса *S*. ОС запоминает значение начального смещения *S* и во время выполнения программы помещает его в специальный регистр процессора. При обращении к памяти виртуальные адреса этой программы преобразуются в физические путем прибавления к ним смещения *S*. Например, при выполнении инструкции пересылки данных, находящихся по адресу *VA*, виртуальный адрес *VA* заменяется физическим адресом *VA+S*.

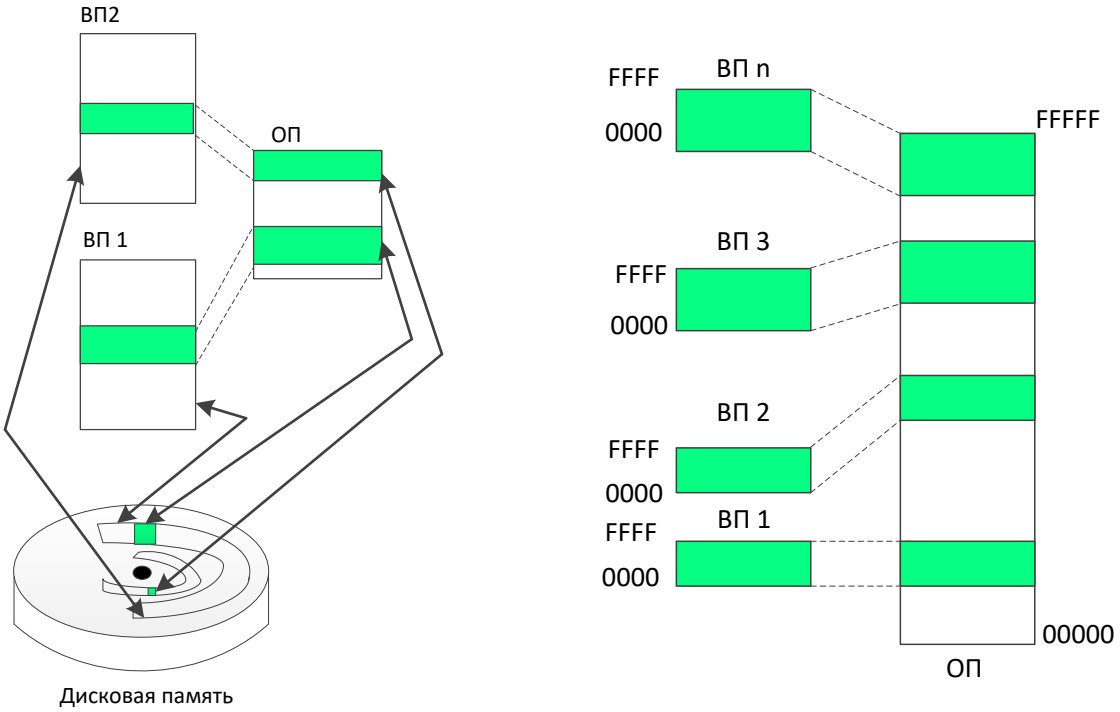
Такой способ является более гибким: в то время как перемещающий загрузчик жестко привязывает программу к первоначально выделенному ей участку памяти, динамическое преобразование виртуальных адресов позволяет перемещать программный код процесса в течение всего периода его выполнения.

3.4. Виртуальная память и свопинг

Сегодня для машин универсального назначения типична ситуация, когда объем виртуального адресного пространства превышает доступный объем оперативной памяти. В таком случае операционная система для хранения данных виртуального адресного пространства процесса, не помещающихся в оперативную память, использует внешнюю память, которая в современных компьютерах представлена жесткими дисками (рис. 3.6, а). Именно на этом принципе основана *виртуальная память* – наиболее совершенный механизм, используемый в операционных системах для управления памятью.

Однако соотношение объемов виртуальной и физической памяти может быть и обратным. Так, в мини-компьютерах 80-х годов разрядности поля адреса нередко не хватало для того, чтобы охватить всю имеющуюся

оперативную память. Несколько процессов могло быть загружено в память одновременно и целиком (рис. 3.6, б).



а. б.

Рис. 3.6. Соотношение объемов виртуального адресного пространства и физической памяти: виртуальное адресное пространство превосходит объем физической памяти (а), виртуальное адресное пространство меньше объема физической памяти (б)

Необходимо подчеркнуть, что виртуальное адресное пространство и виртуальная память – это различные механизмы, которые не обязательно реализуются в операционной системе одновременно. Можно представить себе ОС, в которой поддерживаются виртуальные адресные пространства для процессов, но отсутствует механизм виртуальной памяти. Это возможно только в том случае, если размер виртуального адресного пространства каждого процесса меньше объема физической памяти.

Подмена (виртуализация) оперативной памяти дисковой памятью позволяет повысить объем оперативной памяти компьютера поскольку суммарный объем памяти, занимаемой процессами, может существенно превосходить имеющийся объем оперативной памяти.

Виртуальным называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. В данном случае в распоряжение прикладного программиста предоставляется виртуальная оперативная память, размер которой намного превосходит всю имеющуюся в системе реальную оперативную память.

Виртуализация оперативной памяти осуществляется совокупностью программных модулей ОС и аппаратных схем процессора и включает решение следующих задач:

- размещение данных в запоминающих устройствах разного типа, например, часть кодов программы – в оперативной памяти, а часть – на диске;
- выбор образов процессов или их частей для перемещения из оперативной памяти на диск и обратно;
- перемещение по мере необходимости данных между памятью и диском; преобразование виртуальных адресов в физические.

Виртуализация памяти может быть осуществлена на основе двух различных подходов:

- *свопинг* (*swapping*) – образы процессов выгружаются на диск и возвращаются в оперативную память целиком;
- *виртуальная память* (*virtual memory*) – между оперативной памятью и диском перемещаются части (сегменты, страницы и т.п.) образов процессов.

Свопинг представляет собой частный случай виртуальной памяти и, следовательно, более простой в реализации способ совместного использования оперативной памяти и диска. Однако подкачке свойственна избыточность: когда ОС решает активизировать процесс, для его выполнения, как правило, не требуется загружать в оперативную память все его сегменты полностью – достаточно загрузить небольшую часть кодового сегмента с подлежащей выполнению инструкцией и частью сегментов данных, с которыми работает эта инструкция, а также отвести место под сегмент стека.

Для временного хранения сегментов и страниц на диске отводится либо специальная область, либо специальный файл, которые во многих ОС по традиции продолжают называть областью, или файлом свопинга, хотя перемещение информации между оперативной памятью и диском осуществляется уже не в форме полного замещения одного процесса другим, а частями. Другое популярное название этой области – *страничный файл* (*page file*, или *paging file*). Текущий размер страничного файла является важным параметром, оказывающим влияние на возможности операционной системы: чем больше страничный файл, тем больше приложений может одновременно выполнять ОС (при фиксированном размере оперативной памяти).

3.5. Алгоритмы управления памятью

Рассмотрим наиболее общие подходы к распределению памяти, которые были характерны для разных периодов развития операционных систем. Некоторые из них сохранили актуальность и широко используются в современных ОС, другие же представляют в основном только познавательный интерес, хотя их и сегодня можно встретить в специализирован-

ных системах. На рис. 3.7 все алгоритмы распределения памяти разделены на два класса:

- алгоритмы, в которых используется перемещение сегментов процессов между оперативной памятью и диском;
- алгоритмы, в которых внешняя память не привлекается.

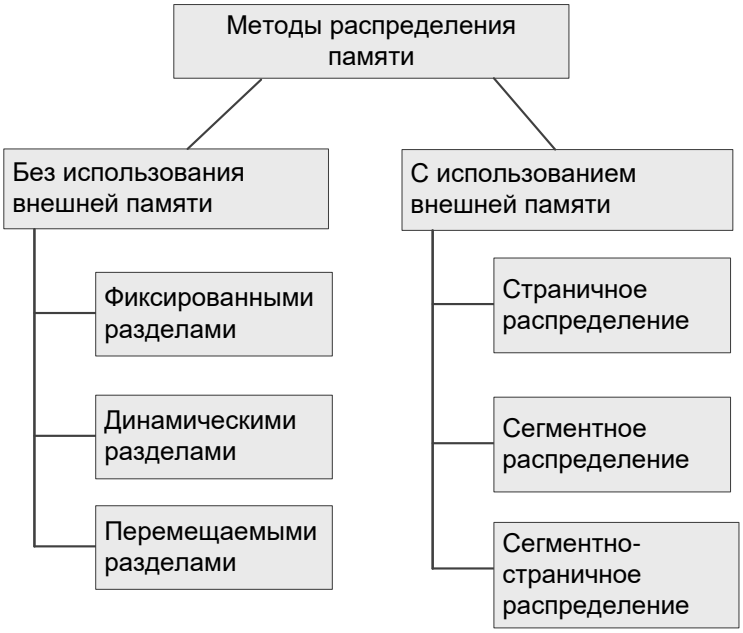


Рис. 3.7. Классификация методов распределения памяти

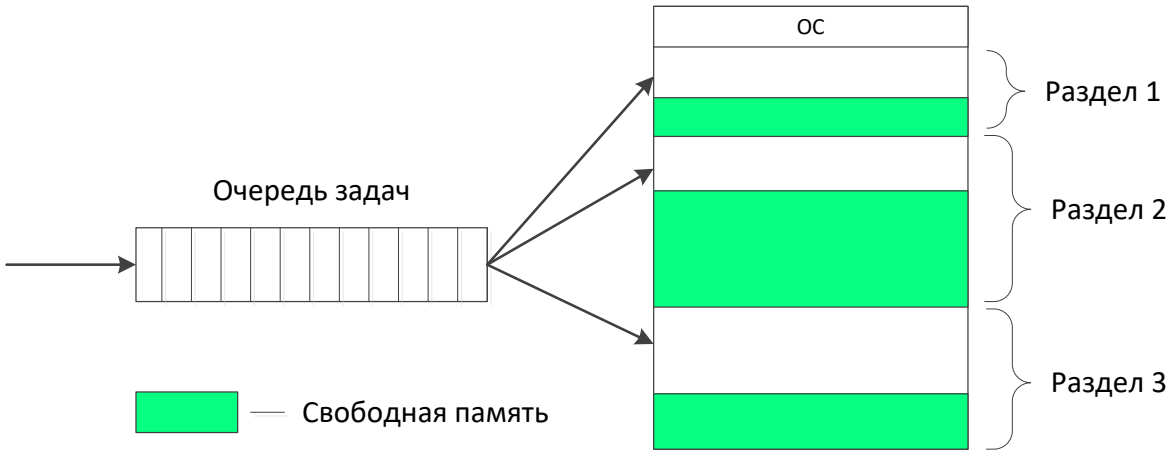
3.5.1. Алгоритмы управления памятью без использования механизма виртуальной памяти

3.5.1.1. Распределение памяти фиксированными разделами

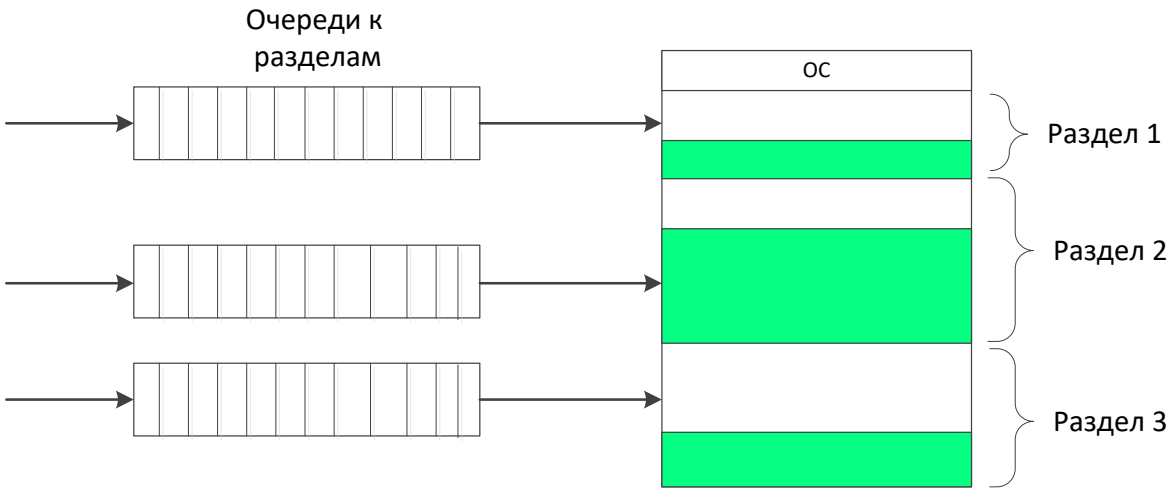
Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых разделами. Такое разбиение может быть выполнено вручную оператором во время старта системы или во время ее установки. После этого границы разделов не изменяются.

Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь (рис. 3.8, а), либо в очередь к некоторому разделу (рис. 3.8, б).

При очевидном преимуществе – простоте реализации, этот метод имеет существенный недостаток – жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов. Независимо от размера программы она будет занимать весь раздел. С другой стороны, разбиение памяти на разделы не позволяет выполнять процессы, программы которых не помещаются ни в один из разделов, но для которых было бы достаточно памяти нескольких разделов. Такой способ управления памятью применялся в ранних мультипрограммных ОС.



а.



б.

Рис. 3.8. Распределение памяти фиксированными разделами: с общей очередью (а), с отдельными очередями (б)

3.5.1.2. Распределение памяти динамическими разделами

В этом случае память машины не делится заранее на разделы. Сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память (если достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для него не создается). После завершения процесса память освобождается, и на это место может быть загружен другой процесс. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рис. 3.9 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так, в момент t_0 в памяти находится только ОС, а к моменту t_1 память разделена между 5 процессами, причем процесс П4, завершаясь, покидает память. На

освободившееся от процесса П4 место загружается процесс П6, поступивший в момент t_3.

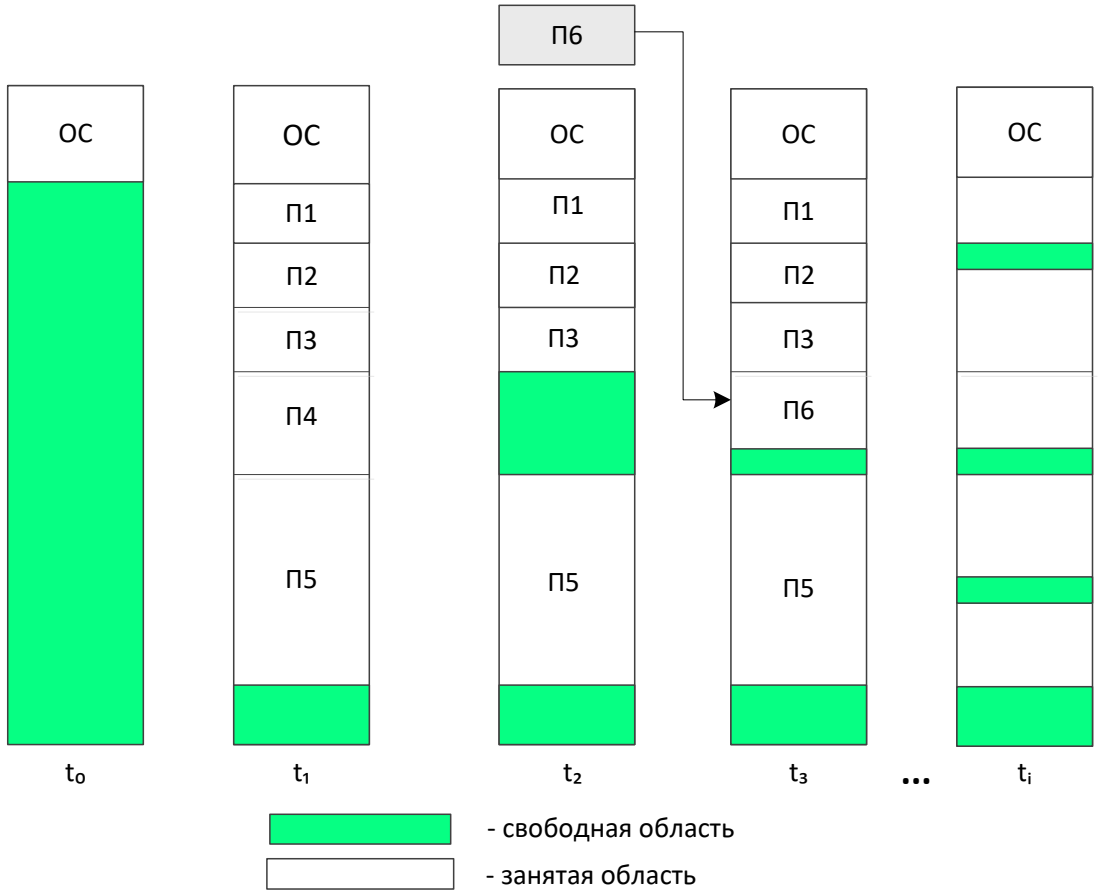


Рис. 3.9. Распределение памяти динамическими разделами

По сравнению с методом распределения памяти фиксированными разделами этот метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – фрагментация памяти.

Фрагментация – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Распределение памяти динамическими разделами лежит в основе подсистем управления памятью многих мультипрограммных операционных систем.

3.5.1.3. Перемещаемые разделы

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала единую свободную область (рис 3.10). ОС должна еще время от времени копировать содержимое разделов из од-

ного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется сжатием.

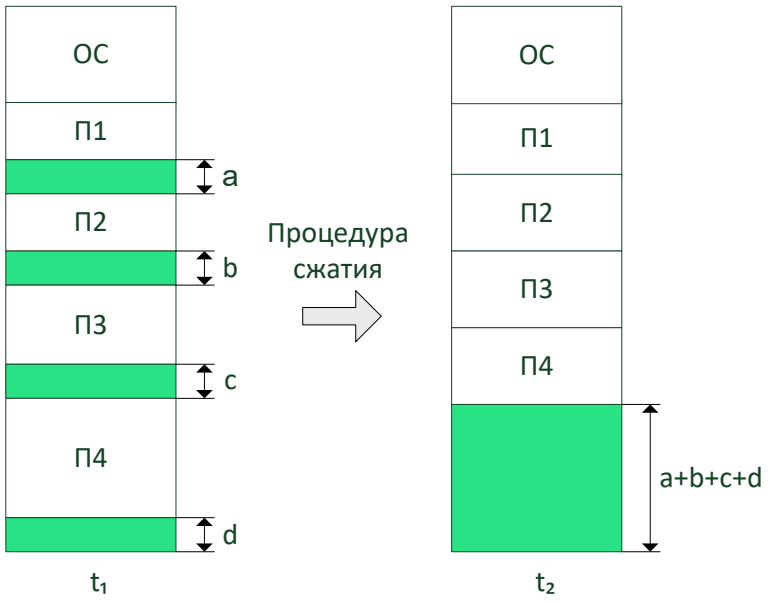


Рис. 3.10. Распределение памяти перемещаемыми разделами

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества этого метода.

Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то в данном случае невозможно выполнить настройку адресов с помощью перемещающего загрузчика. Здесь более подходящим оказывается динамическое преобразование адресов.

3.5.2. Алгоритмы управления памятью с использованием виртуальной памяти

Ключевой проблемой виртуальной памяти, возникающей в результате многократного изменения местоположения в оперативной памяти образов процессов или их частей, является преобразование виртуальных адресов в физические. Решение этой проблемы, в свою очередь, зависит от того, какой способ структуризации виртуального адресного пространства принят в системе управления памятью.

В настоящее время все множество реализаций виртуальной памяти может быть представлено тремя классами.

1. *Страничная виртуальная память* организует перемещение данных между памятью и диском страницами – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера.
2. *Сегментная виртуальная память* предусматривает перемещение данных сегментами – частями виртуального адресного простран-

ства произвольного размера, полученными с учетом смыслового значения данных.

3. *Сегментно-страничная виртуальная* память использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных здесь является страница. Этот способ управления памятью объединяет в себе элементы обоих предыдущих подходов.

3.5.2.1. Страничное распределение

На рис. 3.11 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для конкретной системы размера, называемые *виртуальными страницами* (*virtual pages*). В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

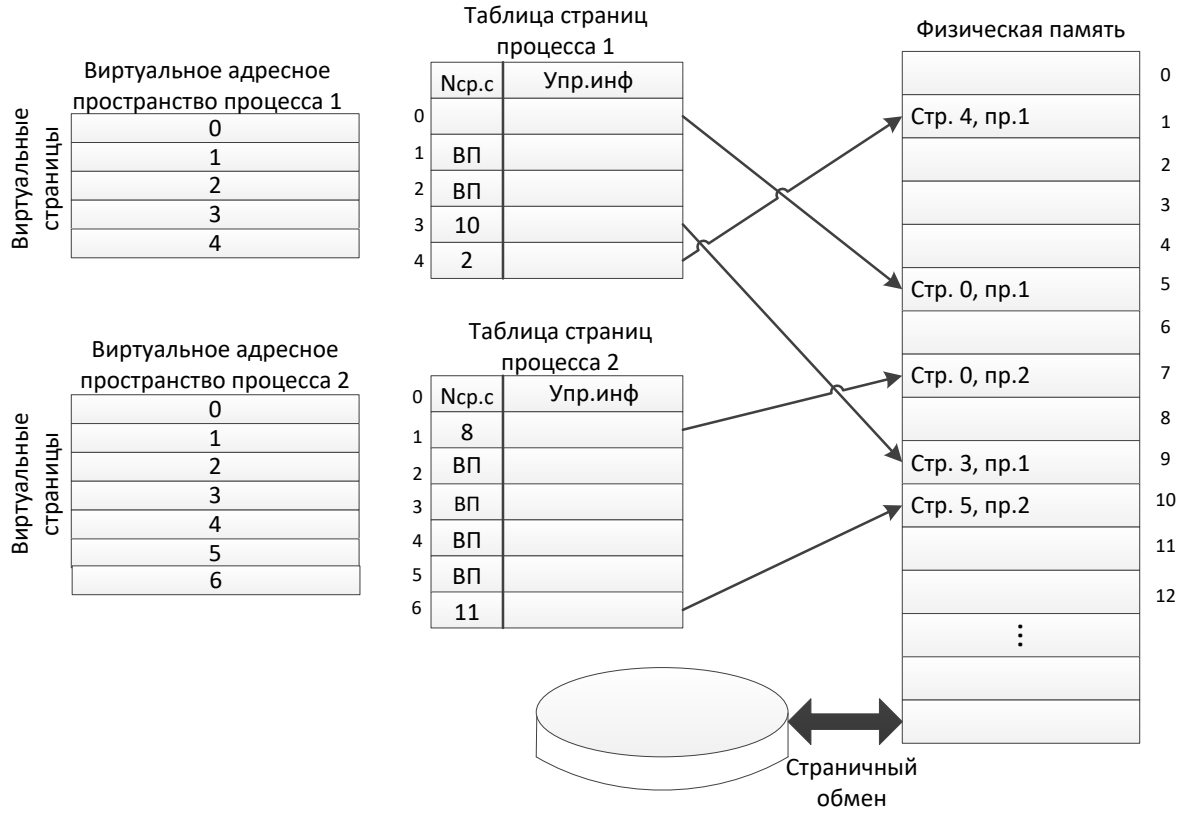


Рис. 3.11. Страничное распределение памяти

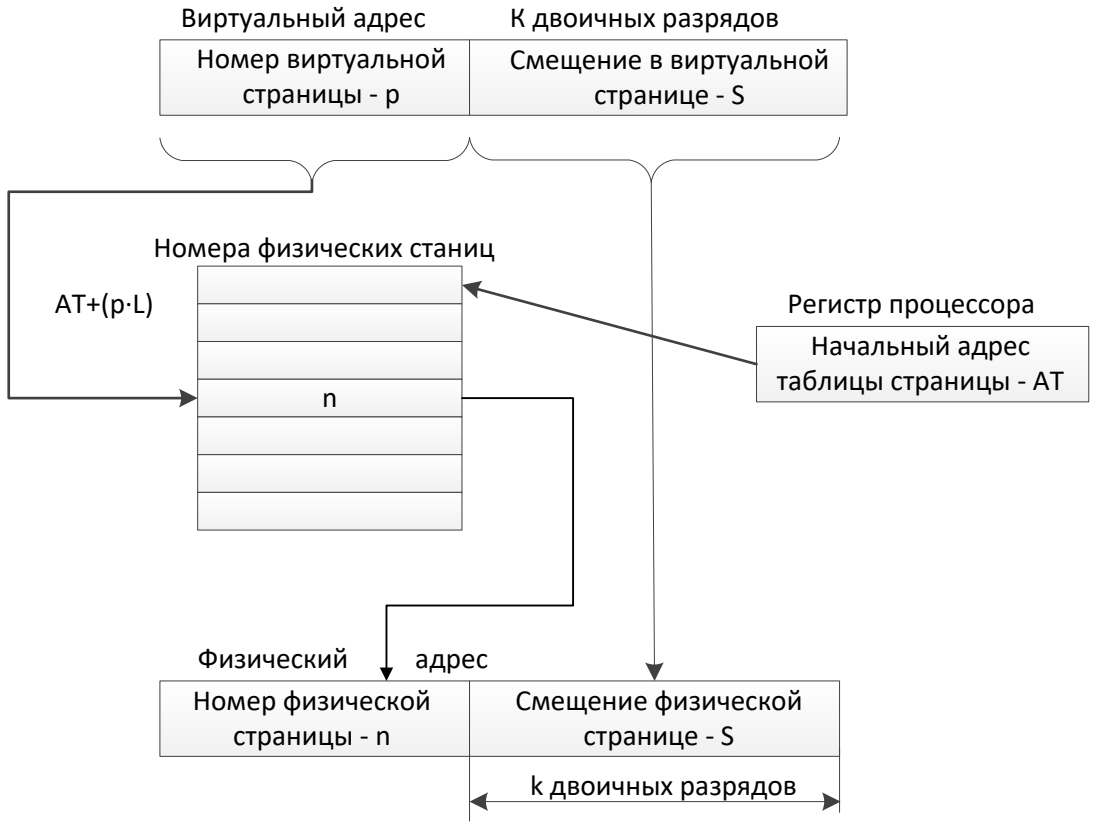


Рис. 3.12. Схема преобразования виртуального адреса в физический

Вся оперативная память машины также делится на части такого же размера, называемые *физическими страницами* (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить *механизм* преобразования адресов. Организация перемещение данных между памятью и диском ведется *страницами* – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера.

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. Для каждого процесса операционная система создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический, то есть виртуальный адрес заменяется указанным в записи таблицы физическим адресом. Если же нужная виртуальная страница в этот момент

выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу (для этого операционная система должна помнить положение вытесненной страницы в страничном файле диска) и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то на основании принятой в системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную память, обнуляется ее бит присутствия и анализируется ее признак модификации. Если выталкиваемая страница за время последнего пребывания в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой виртуальной страницы, и никакой записи на диск не производится. Физическая страница объявляется свободной. Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, с тем чтобы невозможно было использовать содержимое выгруженной страницы.

3.5.2.2. Сегментное распределение

При страничной организации виртуальное адресное пространство процесса делится на равные части механически, без учета смыслового значения данных. Такой подход не позволяет обеспечить дифференцированный доступ к разным частям программы, а это свойство могло бы быть очень полезным во многих случаях. Кроме того, разбиение виртуального адресного пространства на «осмысленные» части делает возможным совместное использование фрагментов программ разными процессами. При отображении в физическую память сегменты, содержащие коды подпрограммы из обоих виртуальных пространств, проецируются на одну и ту же область физической памяти. Таким образом оба процесса получат доступ к одной и той же копии подпрограммы (рис. 3.13).

Итак, виртуальное адресное пространство процесса делится на части – сегменты, размер которых определяется с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями.

Максимальный размер сегмента определяется разрядностью виртуального адреса. Сегменты не упорядочиваются друг относительно друга, так что общего для сегментов линейного виртуального адреса не суще-

ствует, виртуальный адрес задается парой чисел: номером сегмента и линейным виртуальным адресом внутри сегмента (рис. 3.14).

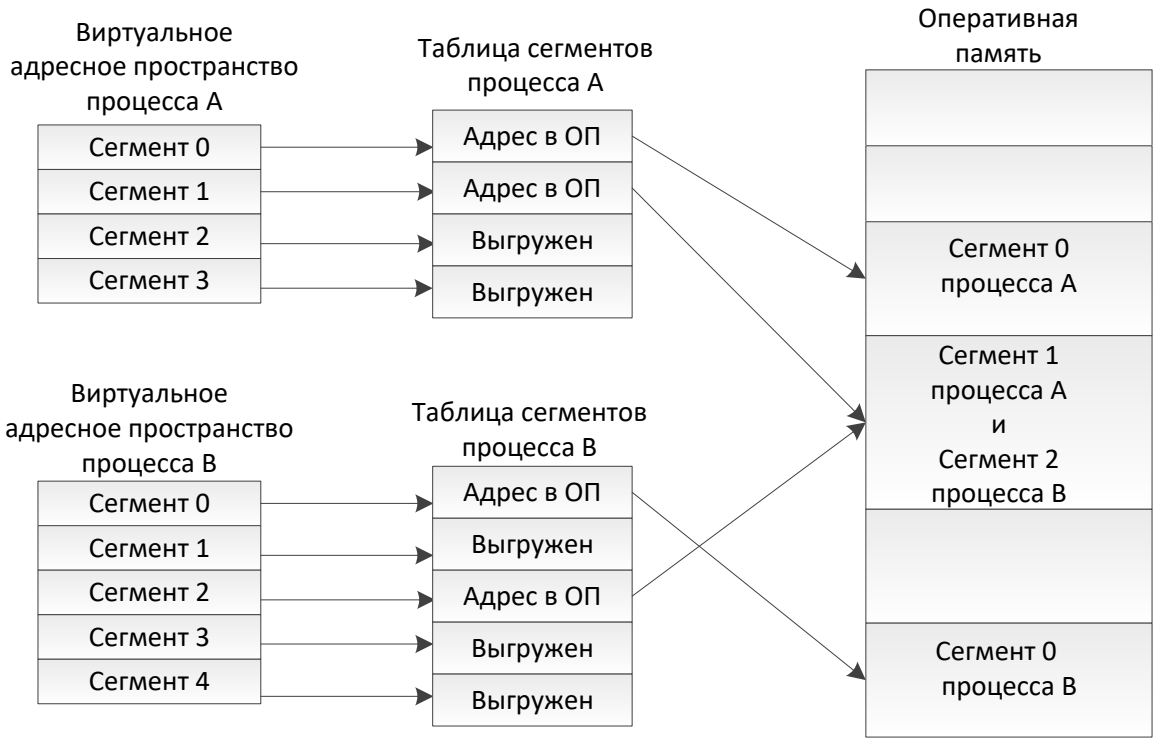


Рис. 3.13. Распределение памяти сегментами

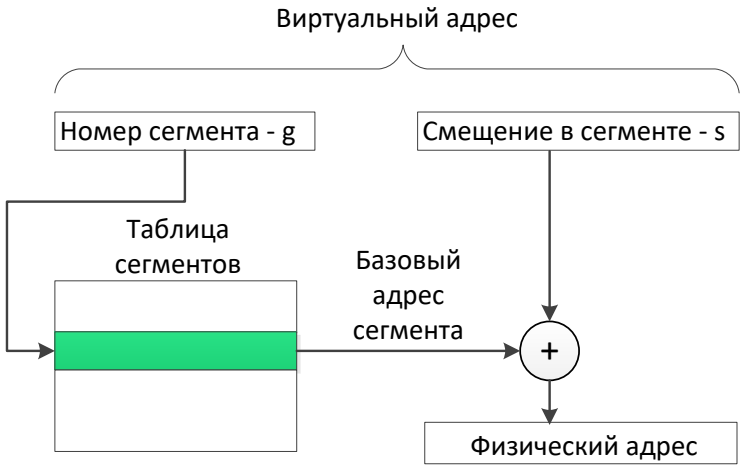


Рис. 3.14. Преобразование виртуального адреса при сегментной организации памяти

При загрузке процесса в оперативную память помещается только часть его сегментов, полная копия виртуального адресного пространства находится в дисковой памяти. Для каждого загружаемого сегмента операционная система подыскивает непрерывный участок свободной памяти достаточного размера. Смежные в виртуальной памяти сегменты одного процесса могут занимать в оперативной памяти несмежные участки. Если во время выполнения процесса происходит обращение по виртуальному адресу, относящемуся к сегменту, который в этот момент отсутствует в памяти,

то происходит прерывание. ОС приостанавливает активный процесс, запускает на выполнение следующий процесс из очереди, а параллельно организует загрузку нужного сегмента с диска. При отсутствии в памяти места, необходимого для загрузки сегмента, операционная система выбирает сегмент на выгрузку, при этом она использует критерии, аналогичные рассмотренным выше критериям выбора страниц при страничном способе управления памятью.

Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который этот сегмент загружается в единственном экземпляре.

Как видно, сегментное распределение памяти имеет очень много общего со страничным распределением. Механизмы преобразования адресов этих двух способов управления памятью тоже весьма схожи, однако в них имеются и существенные отличия, которые являются следствием того, что сегменты в отличие от страниц имеют произвольный размер. Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g – номер сегмента, а s – смещение в сегменте. Физический адрес получается путем сложения базового адреса сегмента, который определяется по номеру сегмента g из таблицы сегментов и смещения s (рис. 3.14).

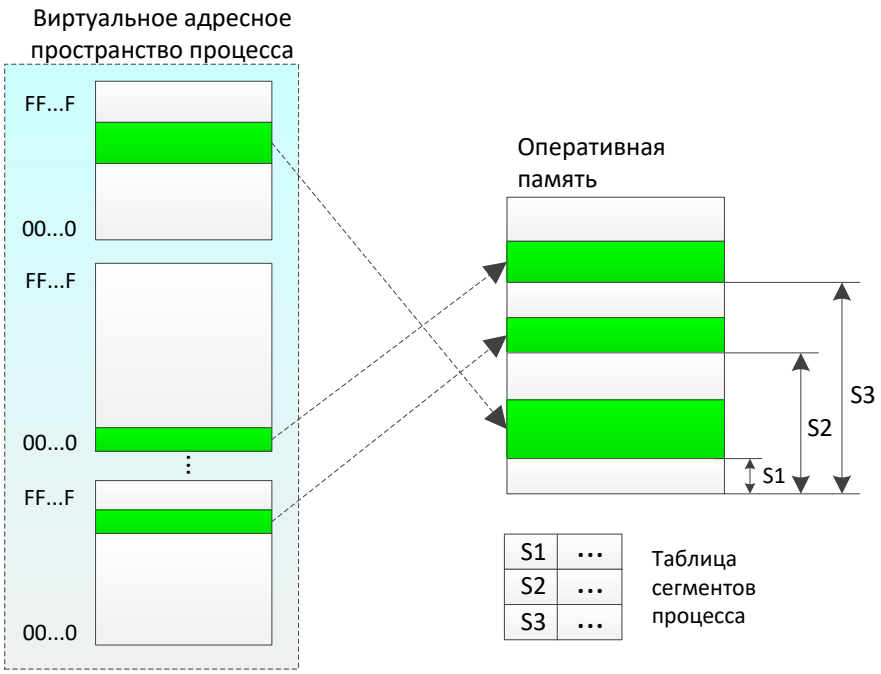
Главный недостаток сегментного распределения – это фрагментация, которая возникает из-за непредсказуемости размеров сегментов. В процессе работы системы в памяти образуются небольшие участки свободной памяти, в которые не может быть загружен ни один сегмент. Суммарный объем, занимаемый фрагментами, может составить существенную часть общей памяти системы, приводя к ее неэффективному использованию.

3.5.2.3. Сегментно-страничное распределение

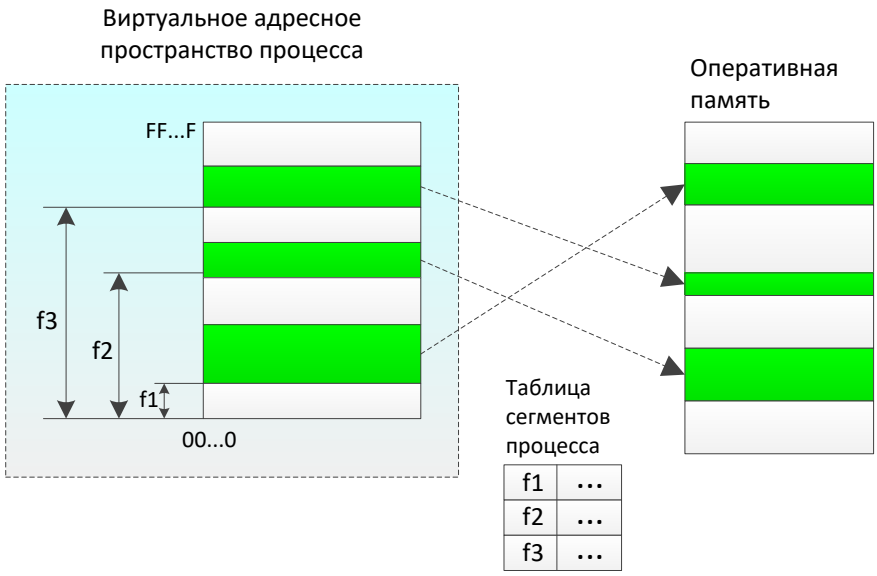
Метод *сегментно-страничного распределения* памяти представляет собой комбинацию страничного и сегментного механизмов управления памятью и направлен на реализацию достоинств обоих подходов.

Также, как и при сегментной организации памяти, виртуальное адресное пространство процесса разделено на сегменты. Это позволяет определять разные права доступа к разным частям кодов и данных программы. Перемещение данных между памятью и диском осуществляется не сегментами, а страницами. Для этого каждый виртуальный сегмент и физическая память делятся на страницы равного размера, что позволяет более эффективно использовать память, сократив до минимума фрагментацию.

В большинстве современных реализаций сегментно-страничной организации памяти в отличие от набора виртуальных диапазонов адресов при сегментной организации памяти (рис. 3.15а) все виртуальные сегменты образуют одно непрерывное линейное виртуальное адресное пространство (рис. 3.15б).



а.



б.

Рис. 3.15. Два способа сегментации при сегментно-страничной организации памяти

Координаты байта в виртуальном адресном пространстве при сегментно-страничной организации можно задать двумя способами:

- во-первых, линейным виртуальным адресом, который равен сдвигу этого байта относительно границы общего линейного виртуального пространства;
- во-вторых, парой чисел, одно из которых является номером сегмента, а другое – смещением относительно начала сегмента. При этом в отличие от сегментной модели, для однозначного задания виртуального адреса вторым способом необходимо каким-то об-

разом указать также начальный виртуальный адрес сегмента с этим номером.

Системы виртуальной памяти ОС с сегментно-страничной организацией используют второй способ, так как он позволяет непосредственно определить принадлежность адреса некоторому сегменту и проверить права доступа процесса к нему.

Для каждого процесса операционная система создает отдельную таблицу сегментов, в которой содержатся описатели (дескрипторы) всех сегментов процесса. Описание сегмента включает назначенные ему права доступа и другие характеристики, подобные тем, которые содержатся в дескрипторах сегментов при сегментной организации памяти. Однако имеется и принципиальное отличие. В поле базового адреса указывается не начальный физический адрес сегмента, отведенный ему в результате загрузки в оперативную память, а начальный линейный виртуальный адрес сегмента в пространстве виртуальных адресов (на рис. 3.15 базовые физические адреса обозначены S_1, S_2, S_3, а базовые виртуальные адреса – f_1, f_2, f_3).

Наличие базового виртуального адреса сегмента в дескрипторе позволяет однозначно преобразовать адрес, заданный в виде пары (номер сегмента, смещение в сегменте), в линейный виртуальный адрес байта, который затем преобразуется в физический адрес страничным механизмом.

Преобразование виртуального адреса в физический происходит в два этапа (рис. 3.16).

1. На первом этапе работает механизм сегментации. Исходный виртуальный адрес, заданный в виде пары (номер сегмента, смещение), преобразуется в линейный виртуальный адрес. Для этого на основании базового адреса таблицы сегментов и номера сегмента вычисляется адрес дескриптора сегмента. Анализируются поля дескриптора и выполняется проверка возможности выполнения заданной операции. Если доступ к сегменту разрешен, то вычисляется линейный виртуальный адрес путем сложения базового адреса сегмента, извлеченного из дескриптора, и смещения, заданного в исходном виртуальном адресе.
2. На втором этапе работает страничный механизм. Полученный линейный виртуальный адрес преобразуется в искомый физический адрес. В результате преобразования линейный виртуальный адрес представляется в том виде, в котором он используется при страничной организации памяти, а именно в виде пары (номер страницы, смещение в странице). Благодаря тому, что размер страницы выбран равным степени двойки, эта задача решается простым отделением некоторого количества младших двоичных разрядов. При этом в старших разрядах содержится номер виртуальной страницы, а в младших – смещение искомого элемента относительно начала страницы.

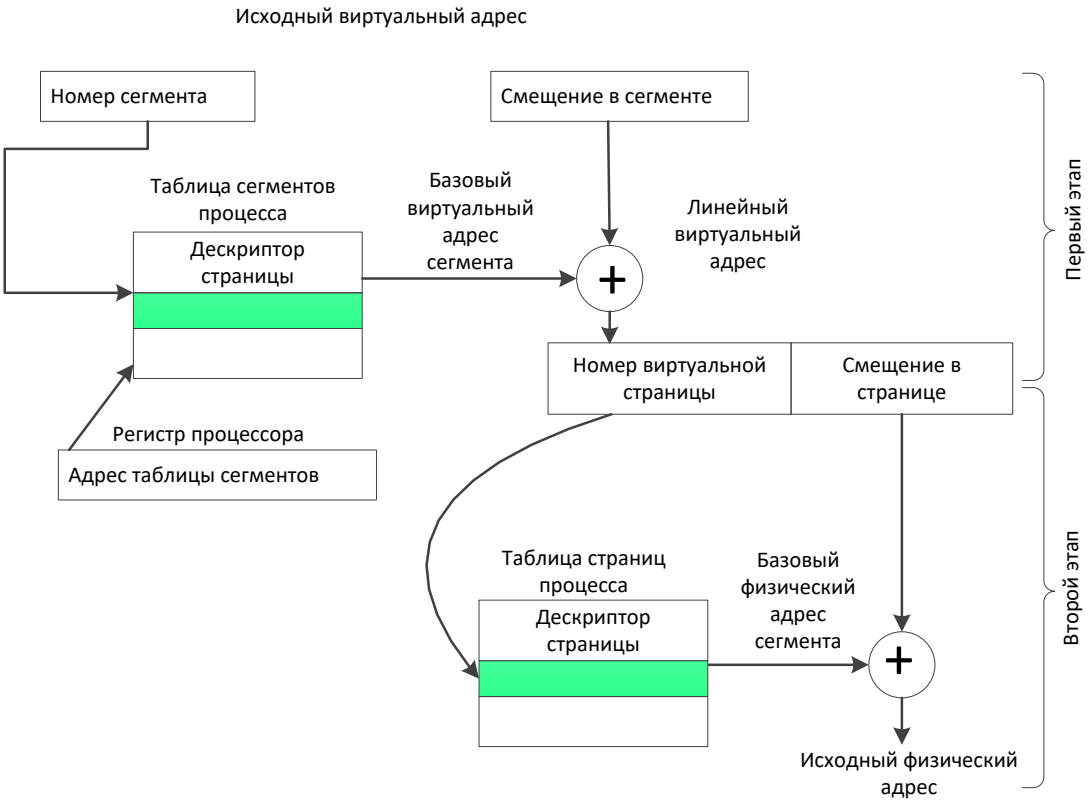


Рис. 3.16. Преобразование виртуального адреса в физический при сегментно -страничной организации памяти

4. Прерывания

4.1. Понятие прерывания

Система прерываний переводит процессор на выполнение потока команд, отличного от того, который выполнялся до сих пор, с последующим возвратом к исходному коду. Механизм прерываний очень похож на механизм выполнения процедур. Это на самом деле так, хотя между этими механизмами имеется важное отличие. Переключение по прерыванию отличается от переключения, которое происходит по команде безусловного или условного перехода, предусмотренной программистом в потоке команд приложения. Переход по команде происходит в заранее определенных программистом точках программы в зависимости от исходных данных, обрабатываемых программой. Прерывание же происходит в произвольной точке потока команд программы, которую программист не может прогнозировать. Прерывание возникает либо в зависимости от внешних по отношению к процессу выполнения программы событий, либо при появлении непредвиденных аварийных ситуаций в процессе выполнения этой программы. Сходство же прерываний с процедурами состоит в том, что в обоих случаях выполняется некоторая подпрограмма, обрабатывающая специальную ситуацию, а затем продолжается выполнение основной ветви программы.

В зависимости от источника, вызывающего прерывание, последние делятся на три больших класса:

1. *внешние прерывания* могут возникать в результате действий пользователя или оператора за терминалом, или же в результате поступления сигналов от аппаратных устройств – сигналов завершения операций ввода-вывода, вырабатываемых контроллерами внешних устройств компьютера. Внешние прерывания называют также *аппаратными*, отражая тот факт, что прерывание возникает вследствие подачи некоторой аппаратурой электрического сигнала, который передается на специальный вход прерывания процессора. Этот класс прерываний является асинхронным по отношению к потоку инструкций прерываемой программы;
2. *внутренние прерывания,* называемые также *исключениями* (exeption), происходят синхронно выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы. Примерами исключений являются деление на нуль, ошибки защиты памяти, обращения по несуществующему адресу, попытка выполнить привилегированную инструкцию в пользовательском режиме и т. п.;
3. *программные прерывания* отличаются от предыдущих двух классов тем, что они по своей сути не являются «истинными» прерываниями и возникают при выполнении особой команды процессо-

ра, выполнение которой имитирует прерывание, то есть переход на новую последовательность инструкций. Причины использования программных прерываний вместо обычных инструкций вызова процедур будут изложены ниже, после рассмотрения механизма прерываний.

Прерываниям приписывается приоритет, с помощью которого они ранжируются по степени важности и срочности.

Прерывания обычно обрабатываются модулями операционной системы, так как действия, выполняемые по прерыванию, относятся к управлению разделяемыми ресурсами вычислительной системы – принтером, диском, таймером, процессором и т. п. Процедуры, вызываемые по прерываниям, обычно называют *обработчиками прерываний*, или *процедурами обслуживания прерываний*:

- *аппаратные прерывания* – обрабатываются драйверами соответствующих внешних устройств;
- *исключения* – обрабатываются специальными модулями ядра;
- *программные прерывания* – обрабатываются процедурами ОС, обслуживающими системные вызовы.

Кроме этих модулей в операционной системе может находиться так называемый диспетчер прерываний, который координирует работу отдельных обработчиков прерываний.

4.2. Механизм прерываний

Механизм прерываний поддерживается аппаратными средствами компьютера и программными средствами операционной системы.

Существуют два основных способа, выполнения прерывания, причем в обоих способах процессору предоставляется информация об уровне приоритета прерывания на шине подключения внешних устройств:

1. *векторный* (*vectored*). В случае векторных прерываний в процессор передается также информация о начальном адресе программы обработки возникшего прерывания – обработчика прерываний. Устройствам, которые используют векторные прерывания, назначается вектор прерываний, представляющий собой электрический сигнал, выставляемый на соответствующие шины процессора и несущий в себе информацию об определенном, закрепленном за этим устройством номере, который идентифицирует соответствующий обработчик прерываний;
2. *опрашиваемый* (*polled*). При использовании опрашиваемых прерываний процессор получает от запросившего прерывание устройства только информацию об уровне приоритета прерывания. С каждым уровнем прерываний связано несколько устройств и соответственно несколько программ – обработчиков прерываний. При возникновении прерывания процессор определяет, какое устройство запросило прерывание путем опроса обработчиков

прерываний для этого уровня приоритета, пока один из обработчиков не подтвердит, что прерывание пришло от обслуживаемого им устройства. Если же с каждым уровнем прерываний связано только одно устройство, то определение нужной программы обработки прерывания происходит немедленно, как и при векторном прерывании.

Механизм прерываний аппаратной платформы может сочетать векторный и опрашиваемый типы прерываний. Контроллеры периферийных устройств выставляют на шину не вектор, а сигнал запроса прерывания определенного уровня *IRQ*. Например, вектор прерываний в процессоре Intel x86 поставляет контроллер прерываний, который отображает поступающий от шины сигнал *IRQ* на определенный номер вектора. Вектор прерываний, передаваемый в процессор, представляет собой целое число в диапазоне от 0 до 255, указывающее на одну из 256 программ обработки прерываний, адреса которых хранятся в таблице обработчиков прерываний. В том случае, когда к каждой линии *IRQ* подключается только одно устройство, процедура обработки прерываний работает так, как если бы система прерываний была чисто векторной, то есть процедура не выполняет никаких дополнительных опросов для выяснения того, какое именно устройство запросило прерывание. Однако, при совместном использовании одного уровня *IRQ* несколькими устройствами, программа обработки прерываний должна работать в соответствии со схемой опрашиваемых прерываний, то есть дополнительно выполнить опрос всех устройств, подключенных к этому уровню *IRQ*.

Механизм прерываний чаще всего поддерживает приоритезацию и маскирование прерываний.

Приоритезация означает, что все источники прерываний делятся на классы и каждому классу назначается свой уровень приоритета запроса на прерывание. Приоритеты могут обслуживаться как относительные и абсолютные.

Маскирование – при обслуживании некоторого запроса все запросы с равным или более низким приоритетом маскируются, то есть не обслуживаются. Схема маскирования предполагает возможность временного маскирования (приостановки) прерываний любого класса независимо от уровня приоритета.

Обобщенно последовательность действий аппаратных и программных средств по обработке прерывания можно описать следующими этапами.

1. При возникновении сигнала (для аппаратных прерываний) или условия (для внутренних прерываний) прерывания происходит первичное аппаратное распознавание типа прерывания. В зависимости от поступившей в процессор информации (уровень прерывания, вектор прерывания или тип условия внутреннего прерывания) происходит автоматический вызов процедуры обработки прерывания, адрес которой находится в специ-

альной таблице операционной системы, размещаемой либо в регистрах процессора, либо в определенном месте оперативной памяти.

2. Автоматически сохраняется некоторая часть контекста прерванного потока, которая позволит ядру возобновить исполнение потока процесса после обработки прерывания. В это подмножество обычно включаются значения счетчика команд, слова состояния машины, хранящего признаки основных режимов работы процессора, а также нескольких регистров общего назначения, которые требуются программе обработки прерывания. Может быть сохранен и полный контекст процесса, если ОС обслуживает это прерывание со сменой процесса.

3. Одновременно с загрузкой адреса процедуры обработки прерываний в счетчик команд может автоматически выполняться загрузка нового значения слова состояния машины, которое определяет режимы работы процессора при обработке прерывания, в том числе работу в привилегированном режиме. Прерывания практически во всех мультипрограммных ОС обрабатываются в привилегированном режиме модулями ядра, так как при этом обычно нужно выполнить ряд критических операций, от которых зависит жизнеспособность системы, – управлять внешними устройствами, перепланировать потоки и т. п.

4. Временно запрещаются прерывания этого типа, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры. Многие процессоры автоматически устанавливают признак запрета прерываний в начале цикла обработки прерывания, в противном случае это делает программа обработки прерываний.

5. После того как прерывание обработано ядром ОС, прерванный контекст восстанавливается, и работа потока возобновляется с прерванного места.

4.3. Функции централизованного диспетчера прерываний

Прерывания выполняют очень полезную для вычислительной системы функцию – они позволяют реагировать на асинхронные, но отношению к вычислительному процессу события. Трудности обработки прерываний связаны с непредвиденными переходами управления от одной процедуры к другой, возникающими в результате прерываний от контроллеров внешних устройств. А также возникновение в непредвиденные моменты времени исключений, связанных с ошибками во время выполнения инструкций.

Операционная система должна упорядочивать выполнения системных процедур, вызываемых по прерываниям во времени так же, как планировщик упорядочивает многочисленные пользовательские потоки. Кроме того, сам планировщик потоков является системной процедурой, вызываемой по прерываниям. Поэтому правильное планирование процедур, вызываемых по прерываниям, является необходимым условием правильного планирования пользовательских потоков.

Для упорядочения работы обработчиков прерываний в операционных системах применяется тот же механизм, что и для упорядочения работы пользовательских процессов – механизм приоритетов. В операционной системе выделяется программный модуль, который занимается диспетчеризацией обработчиков прерываний, называемый диспетчером прерываний.

При возникновении прерывания диспетчер прерываний вызывается первым. Он запрещает ненадолго все прерывания, а затем выясняет причину прерывания. После этого диспетчер сравнивает назначенный источнику прерывания приоритет с текущим приоритетом потока команд, выполняемого процессором. В этот момент времени процессор уже может выполнять инструкции другого обработчика прерываний, также имеющего некоторый приоритет. Если приоритет нового запроса выше текущего, то выполнение текущего обработчика приостанавливается, и он помещается в соответствующую очередь обработчиков прерываний. В противном случае в очередь помещается обработчик нового запроса.

4.4. Процедуры обработки прерываний, вызванные из текущего процесса

Важной особенностью процедур, выполняемых по запросам прерываний, является то, что они выполняют работу, чаще всего никак не связанную с текущим процессом. Процедуры обработки прерываний работают с ресурсами, которые были выделены им при инициализации соответствующего драйвера или инициализации самой операционной системы. Эти ресурсы принадлежат операционной системе, а не конкретному процессу. В частности, память выделяется драйверам из системной области. Поэтому обычно говорят, что процедуры обработки прерываний работают вне контекста процесса. Поскольку все подобные процедуры являются частью операционной системы, ответственность за соблюдение этих ограничений несет системный программист. Заставить свои модули выполнять эти ограничения ОС не может.

Диспетчеризация прерываний является важной функцией ОС, и эта функция реализована практически во всех мультипрограммных операционных системах.

В общем случае в операционной системе реализуется двухуровневый механизм планирования работ:

1. верхний уровень планирования выполняется диспетчером прерываний, который распределяет процессорное время между потоком поступающих запросов на прерывания различных типов – внешних, внутренних и программных;
2. оставшееся процессорное время распределяется другим диспетчером – диспетчером потоков, на основании дисциплин квантования.

4.5. Системные вызовы

Системный вызов позволяет приложению обратиться к операционной системе с запросом на выполнение того или иного действия, оформленного как процедура (или набор процедур) кодового сегмента ОС. Для прикладного программиста ОС выглядит как некая библиотека, предоставляющая некоторый набор полезных функций, с помощью которых можно упростить прикладную программу или выполнить действия, запрещенные в пользовательском режиме, например, обмен данными с устройством ввода-вывода.

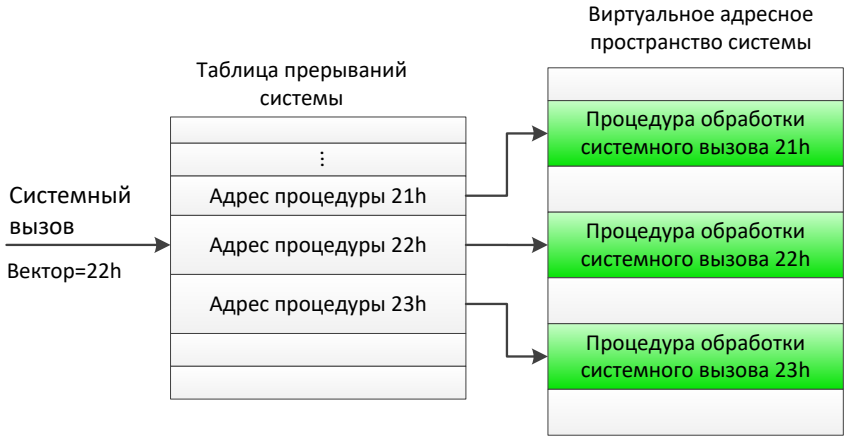
Реализация системных вызовов должна удовлетворять следующим требованиям:

- обеспечивать переключение в привилегированный режим;
- обладать высокой скоростью вызова процедур ОС;
- обеспечивать по возможности единообразное обращение к системным вызовам для всех аппаратных платформ, на которых работает ОС;
- допускать легкое расширение набора системных вызовов;
- обеспечивать контроль со стороны ОС за корректным использованием системных вызовов.

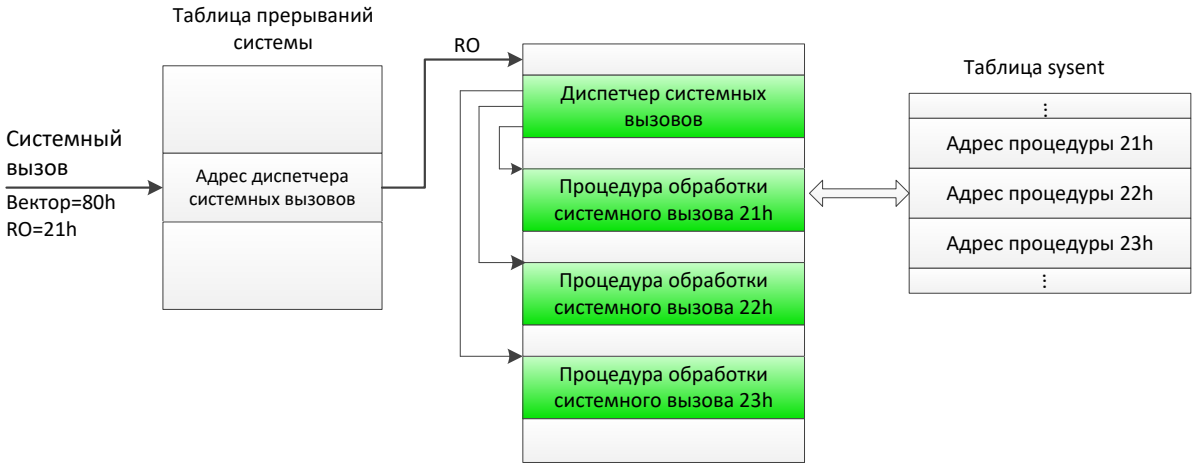
Первое требование для большинства аппаратных платформ может быть выполнено только с помощью механизма программных прерываний. Поэтому будем считать, что остальные требования нужно обеспечить именно для такой реализации системных вызовов. Как это обычно бывает, некоторые из этих требований взаимно противоречивы.

Для обеспечения высокой скорости полезно использовать векторные свойства системы программных прерываний, имеющиеся во многих процессорах, то есть закрепить за каждым системным вызовом определенное значение вектора прерывания. Приложение при таком способе вызова непосредственно указывает в аргументе запроса значение вектора, после чего управление немедленно передается требуемой процедуре операционной системы (рис. 4.1а) – то есть используется децентрализованный способ передачи управления.

В большинстве ОС системные вызовы обслуживаются по централизованной схеме, основанной на существовании диспетчера системных вызовов (рис. 4.14б). При любом системном вызове приложение выполняет программное прерывание с определенным и единственным номером вектора. Способ передачи зависит от реализации, например, номер можно поместить в определенный регистр общего назначения процессора или передать через стек (в этом случае после прерывания и перехода в привилегированный режим их нужно будет скопировать в системный стек из пользовательского, это действие в некоторых процессорах автоматизировано). Также некоторым способом передаются аргументы системного вызова, они могут как помещаться в регистры общего назначения, так и передаваться через стек или массив, находящийся в оперативной памяти.



а.



б.

Рис. 4.1. Децентрализованная (а) и централизованная (б) схемы обработки системных вызовов

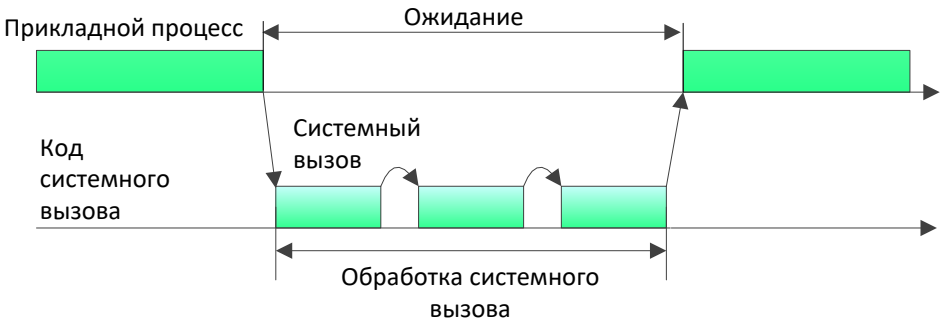
Процедура реализации системного вызова извлекает из системного стека аргументы и выполняет заданное действие. Это действие может быть весьма простым, например, чтение значения системных часов, так что системный вызов оформляется в виде одной функции. Более сложные системные вызовы, такие как чтение из файла или выделение процессу дополнительного сегмента памяти, требуют обращения системного вызова к нескольким внутренним процедурам ядра ОС, принадлежащим к различным подсистемам, таким как подсистема ввода-вывода или управления памятью.

После завершения работы системного вызова управление возвращается диспетчеру, при этом он получает также код завершения этого вызова. Диспетчер восстанавливает регистры процессора, помещает в определенный регистр код возврата и выполняет инструкцию возврата из прерывания, которая восстанавливает непривилегированный режим работы процессора.

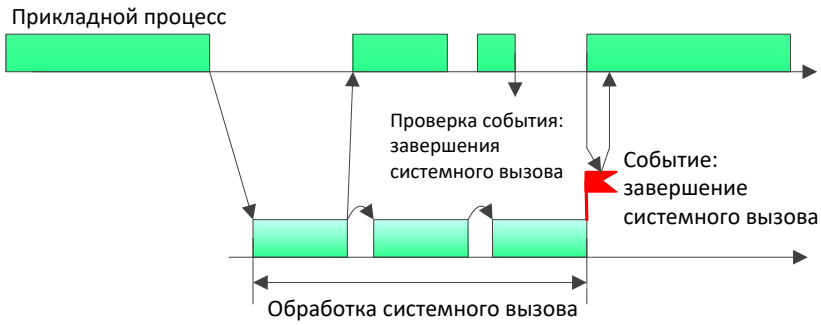
Операционная система может выполнять системные вызовы в *синхронном* или *асинхронном режимах.*

Синхронный системный вызов означает, что процесс, сделавший такой вызов, приостанавливается (переводится планировщиком ОС в состояние ожидания) до тех пор, пока системный вызов не выполнит всю требующуюся от него работу (рис. 4.2а). После этого планировщик переводит процесс в состояние готовности и при очередном выполнении процесс гарантированно может воспользоваться результатами завершившегося к этому времени системного вызова. Синхронные вызовы называются также блокирующими, так как вызвавший системное действие процесс блокируется до его завершения.

Асинхронный системный вызов не приводит к переводу процесса в режим ожидания после выполнения некоторых начальных системных действий, например, запуска операции вывода-вывода, управление возвращается прикладному процессу (рис. 4.2б).



а.



б.

Рис. 4.2. Синхронные (а) и асинхронные (б) системные вызовы

Большинство системных вызовов в операционных системах являются синхронными, так как этот режим избавляет приложение от работы по выяснению момента появления результата вызова. Асинхронные системные вызовы применяются в ОС на основе микроядерного подхода, так как при этом в пользовательском режиме работает часть ОС, которым необходимо иметь полную свободу в организации своей работы, а такую свободу дает только асинхронный режим обслуживания вызовов микроядром.

5. Управление вводом-выводом

5.1. Организация взаимодействия операционной системы с устройствами ввода-вывода

Каждое устройство ввода-вывода вычислительной системы (диск, принтер, терминал и т. п.) снабжено специализированным блоком управления, называемым контроллером. Контроллер взаимодействует с драйвером – *системным программным модулем*, предназначенным для управления этим устройством. Контроллер периодически принимает от драйвера выводимую на устройство информацию, а также команды управления, которые говорят о том, что с этой информацией нужно сделать (например, вывести в виде текста в определенную область терминала или записать в определенный сектор диска).

Основными компонентами подсистемы ввода-вывода являются:

- драйверы, управляющие внешними устройствами;
- файловая система.

Достоинством подсистемы ввода-вывода любой универсальной ОС является наличие разнообразного набора драйверов для наиболее популярных периферийных устройств.

Драйвер взаимодействует, с одной стороны, с модулями ядра ОС (модулями подсистемы ввода-вывода, модулями системных вызовов, модулями подсистем управления процессами и памятью и т. д.), а с другой стороны – с контроллерами внешних устройств. Поэтому существуют два типа интерфейсов:

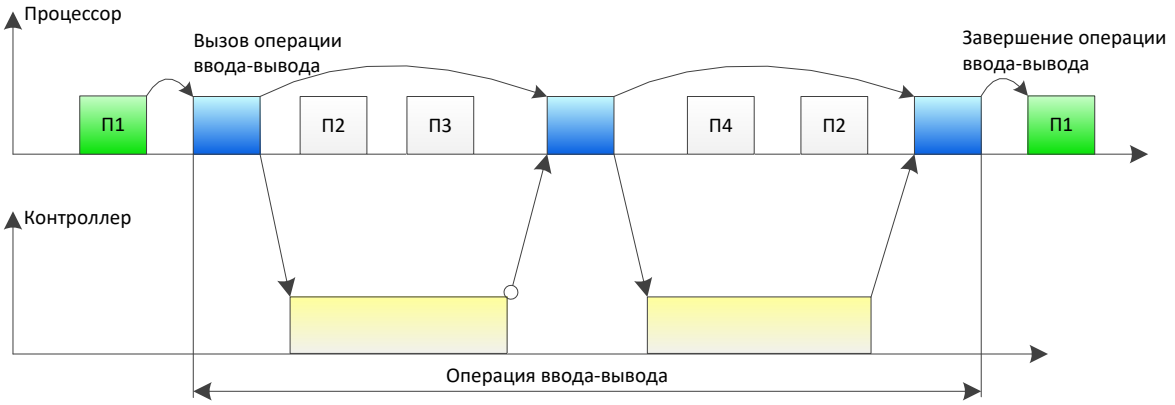
- интерфейс «драйвер-ядро» (DKI – Driver Kernel Interface);
- интерфейс «драйвер-устройство» (DDF – Driver Device Interface).

Подсистема ввода-вывода (Input – Output Subsystem) мультипрограммной ОС при обмене данными с внешними устройствами компьютера решает следующие задачи:

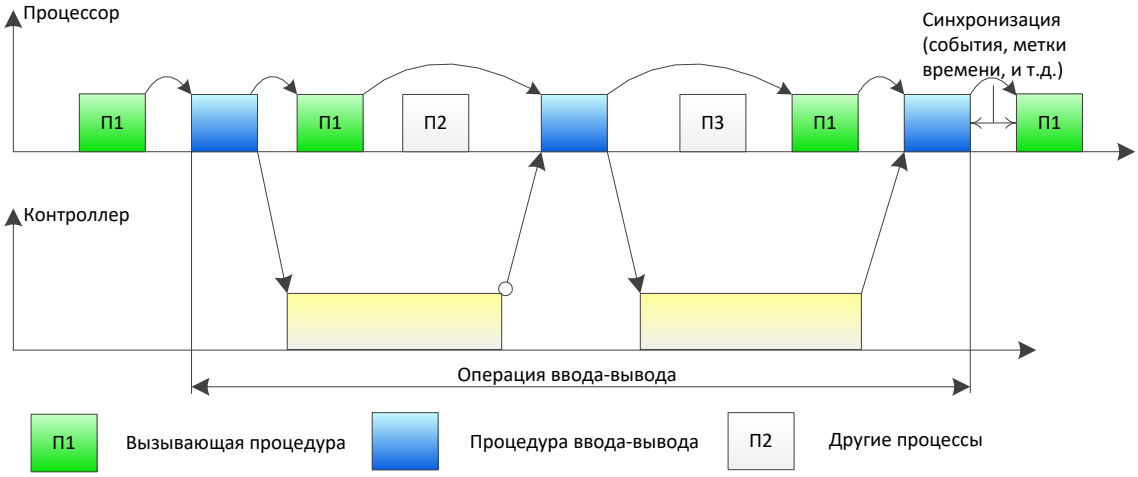
- организация параллельной работы устройств ввода-вывода и процессора;
- согласование скоростей обмена и кэширование данных;
- разделение устройств и данных между процессами;
- обеспечение удобного логического интерфейса между устройствами и остальной частью системы;
- поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера;
- динамическая загрузка и выгрузка драйверов;
- поддержка нескольких файловых систем;
- поддержка синхронных и асинхронных операций ввода-вывода.

Операция ввода-вывода может выполняться по отношению к программному модулю, запросившему операцию:

- *в синхронном режиме* – программный модуль приостанавливает свою работу до тех пор, пока операция ввода-вывода не будет завершена (рис. 7.1а);
- *в асинхронном режиме* – программный модуль продолжает выполняться в мультипрограммном режиме одновременно с операцией ввода-вывода (рис. 7.1б).



а) Синхронное выполнение операции ввода-вывода



б) Асинхронное выполнение операции ввода-вывода

Рис. 5.1. Два режима выполнения операций ввода-вывода

5.2. Многослойная модель подсистемы ввода-вывода

Многослойное построение программного обеспечения, характерно при построении подсистемы ввода-вывода. При этом нижние слои подсистемы ввода-вывода должны включать индивидуальные драйверы, написанные для конкретных физических устройств, а верхние слои должны обобщать процедуры управления этими устройствами, предоставляя об-

щий интерфейс для групп устройств, обладающих некоторыми общими характеристиками.

В самом общем виде программное обеспечение ввода-вывода можно разделить на четыре слоя (рис. 5.2):

1. обработка прерываний;
2. драйверы устройств;
3. независимый от устройств слой операционной системы;
4. пользовательский слой программного обеспечения.

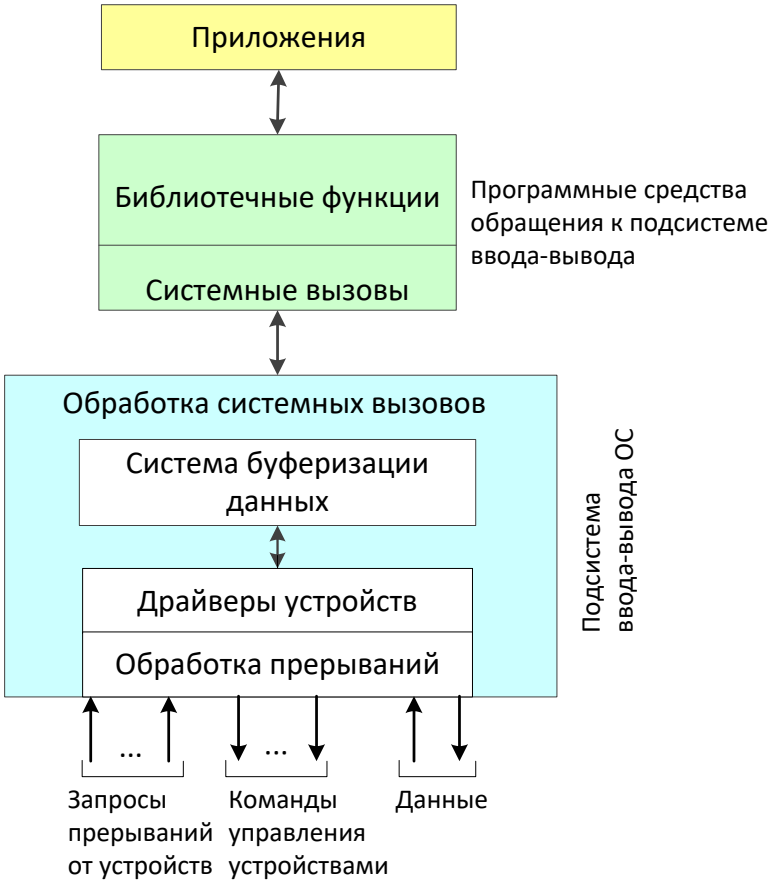


Рис. 5.2. Многоуровневая организация подсистемы ввода-вывода

В более частном виде структура подсистемы ввода-вывода, характерная для современных ОС представлена на рис. 5.3.

Большая часть программного обеспечения ввода-вывода является независимой от устройств. Точная граница между драйверами и независимыми от устройств программами определяется системой, так как некоторые функции, которые могли бы быть реализованы независимым способом, в действительности выполнены в виде драйверов для повышения эффективности функционирования.

Рис. 5.3. Структура подсистемы ввода-вывода современной ОС

Функциями для независимого от устройств слоя являются:
- обеспечение общего интерфейса к драйверам устройств;
- именование устройств;
- защита устройств;
- обеспечение независимого размера блока;
- буферизация;
- распределение памяти на блок-ориентированных устройствах;
- распределение и освобождение выделенных устройств;
- уведомление об ошибках.

5.3. Менеджеры ввода-вывода

В подсистеме ввода-вывода наряду с модулями, отражающими специфику внешних устройств и образующими вертикальные подсистемы, существуют модули универсального назначения.

Менеджер ввода-вывода – модуль ОС, организующий согласованную работу всех остальных компонентов подсистемы ввода-вывода, взаимодействие с пользовательскими процессами и другими подсистемами ОС. Причем функции управления устройствами, распределены по всем уровням, образуя оболочку.

Верхний слой менеджера составляют *системные вызовы ввода-вывода*, которые принимают от пользовательских процессов запросы на ввод-вывод и переадресуют их отвечающим за определенный класс устройств модулям и драйверам, а также возвращают процессам результа-

ты операций ввода-вывода. Таким образом этот слой поддерживает пользовательский интерфейс ввода-вывода, создавая для прикладных программистов максимум удобств по манипулированию внешними устройствами и расположенными на них данными.

Нижний слой менеджера реализует непосредственное взаимодействие с контроллерами внешних устройств, экранируя драйверы от особенностей аппаратной платформы компьютера – шины ввода-вывода, системы прерываний и т. п. Этот слой принимает от драйверов запросы на обмен данными с регистрами контроллеров в некоторой обобщенной форме с использованием независимых от шины ввода-вывода адресации и формата, а затем преобразует эти запросы в зависящий от аппаратной платформы формат.

5.4. Драйверы устройств

Под *драйвером* понимается программный модуль, который обладает следующими свойствами и функциями:

- входит в состав ядра операционной системы, работая в привилегированном режиме;
- непосредственно управляет внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;
- обрабатывает прерывания от контроллера устройства;
- предоставляет прикладному программисту удобный логический интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных;
- взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться, и т. п.

В ОС только драйвер устройства «знает» о конкретных особенностях какого-либо устройства.

Порядок функционирования драйвера устройства:

1. драйвер устройства принимает запрос от программного слоя и решает, как его выполнить. Если драйвер был свободен во время поступления запроса, то он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди уже имеющихся запросов, и он будет выполнен, когда наступит его очередь;
2. преобразование запроса ввода-вывода из абстрактной формы в конкретную. Для дискового драйвера это означает преобразова-

ние номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром;

3. передача команд контроллеру и принятие решения должен ли драйвер блокировать ли себя до окончания заданной операции или нет. Если операция занимает значительное время, как при печати некоторого блока данных, то драйвер блокируется до тех пор, пока операция не завершится, и обработчик прерывания не разблокирует его. Если команда ввода-вывода выполняется быстро (например, прокрутка экрана), то драйвер ожидает ее завершения без блокирования;
4. возвращение управления программе (вызвавшей драйвер) с результатом операции ввода-вывода.

5.5. Типовые базовые системы ввода и вывода

5.5.1. BIOS

Базовая система ввода и вывода или BIOS (Basic Input / Output System) – это программное обеспечение низкого уровня, которое создает «прослойку» между аппаратными средствами компьютера и ОС. Микросхема BIOS встроена в каждую материнскую плату ПК и не зависит от ОС. Доступ к ее настройкам можно получить без установленной ОС, где можно изменить различные параметры оборудования: частоту процессора и памяти, рабочее напряжение, задержку порядок использования загрузочных устройств и т.д. Это позволяет выполнить тонкую настройку компьютера и получить максимальную производительность. Работу в BIOS можно вести с помощью псевдографического интерфейса и клавиатуры.

BIOS запускается при включении компьютера и выполняет следующие функции:

- самотестирование при включении питания;
- обнаружение и инициализация видеокарты;
- отображение стартового экрана BIOS;
- осуществление быстрой проверки памяти (RAM);
- конфигурация устройств plug and play;
- определение загрузочного устройства.

Как только BIOS определил загрузочное устройство, он считывает первый дисковый сектор этого устройства в память. Первый сектор диска – это главная загрузочная запись (MBR) размером обычно 512 байт, содержащий следующие данные:

- загрузчик первой стадии (446 байт);
- таблицу разделов диска (16 байт на раздел × 4 раздела);
- подпись (2 байта).

На этом этапе MBR сканирует таблицу разделов и загружает в оперативную память загрузочный сектор – Volume Boot Record (VBR).

VBR обычно содержит начальный загрузчик программ – Initial Program Loader (IPL), этот код инициирует процесс загрузки. Начальный загрузчик программ включает в себя вторую стадию загрузчика, который затем загружает ОС. На ОС Windows 7, начальный загрузчик программ сначала загружает другую программу: диспетчер загрузки bootmgr, которая затем загружает операционную систему.

Для ОС на ядре Linux используется загрузчик GRUB (Grand Unified Bootloader). Процесс загрузки похож на описанный выше, но загрузчики на первой и второй стадии называются, соответственно, GRUB Stage 1 и GRUB Stage 2.

5.5.2. UEFI

UEFI (Unified Extensible Firmware Interface – унифицированный интерфейс расширяемой прошивки) представляет собой интерфейс между ОС и аппаратной частью, созданный для замены устаревшего BIOS. Основное назначение UEFI – инициализация оборудования при включении системы и передача управления загрузчику ОС и предоставление операционной системе низкоуровневых функций для управления аппаратной частью системы.

UEFI имеет ряд преимуществ перед BIOS:

- графический интерфейс с поддержкой клавиатуры и мыши;
- адресация памяти. UEFI позволяет адресовать до 8192 экзабайт RAM-памяти жесткого диска и до 16 экзабайт оперативной памяти;
- более быстрая загрузка, достигаемая за счет параллельной инициализации отдельных компонентов системы;
- загрузка драйверов и последующая передача их в распоряжение ОС;
- протокол Secure Boot, позволяющий защитить загрузчик ОС и саму ОС;
- поддержка GPT (GUID Partition Table) – нового способа разметки дисков, который позволяет, в отличие от MBR, диски размером более 2 Тб и неограниченное количество разделов;
- модульная архитектура и поддержка сервисов;
- встроенный менеджер загрузки.

UEFI не привязан изначально к 16-битным процессорам с архитектурой x86, как BIOS. Помимо архитектурной независимости, новый стандарт обеспечивает более быструю инициализацию системы, расширенные настройки и функциональность, а также повышенную безопасность (BIOS, в отличие от UEFI, не поддерживает безопасный режим начальной загрузки (Secure Boot) и исполнение кода с цифровой подписью).

6. Файловая система

6.1. Организация файловой системы

Одной из основных задач операционной системы является предоставление удобств пользователю при работе с данными, хранящимися на дисках. Для этого ОС подменяет физическую структуру хранящихся данных некоторой удобной для пользователя логической моделью. Логическая модель файловой системы материализуется в виде дерева каталогов, выводимого на экран такими утилитами, например, как Windows Explorer, в символьных составных именах файлов, в командах работы с файлами. Базовым элементом этой модели является файл, который так же, как и файловая система в целом, может характеризоваться как логической, так и физической структурой.

Файл – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Файлы хранятся в памяти, на зависящей от энергопитания, обычно – на магнитных дисках.

Основные цели использования файла перечислены ниже.

- Долговременное и надежное хранение информации. Долговременность достигается за счет использования запоминающих устройств, не зависящих от питания, а высокая надежность определяется средствами защиты доступа к файлам и общей организацией программного кода ОС, при которой сбои аппаратуры чаще всего не разрушают информацию, хранящуюся в файлах.
- Совместное использование информации. Файлы обеспечивают естественный и легкий способ разделения информации между приложениями и пользователями за счет наличия понятного человеку символьного имени и постоянства хранимой информации и расположения файла.

Файловая система (ФС) – это часть операционной системы, включающая:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих различные операции над файлами, такие как создание, уничтожение, чтение, запись, именование и поиск файлов.

Файловая система распределяет дисковую память, поддерживает именование файлов, отображает имена файлов в соответствующие адреса во внешней памяти, обеспечивает доступ к данным, поддерживает разделение, защиту и восстановление файлов. Таким образом, ФС играет роль промежуточного слоя, экранирующего все сложности физической органи-

зации долговременного хранилища данных, и создающего для программ более простую логическую модель этого хранилища, а также предоставляя им набор удобных в использовании команд для манипулирования файлами.

Основные функции в такой ФС нацелены на решение следующих задач:

- именование файлов;
- программный интерфейс для приложений;
- отображения логической модели файловой системы на физическую организацию хранилища данных;
- устойчивость файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
- защита файлов одного пользователя от несанкционированного доступа другого пользователя.

6.2. Типы файлов

Файлы бывают нескольких типов:

1 обычные файлы:
  1.1 текстовые;
  1.2 двоичные;
2 специальные файлы:
  2.1 блок-ориентированные;
  2.2 байт-ориентированные;
3 файлы-каталоги.

Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т. п. Текстовые файлы можно прочитать на экране и распечатать на принтере.

Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов – их собственные исполняемые файлы.

Специальные файлы – это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же, как и устройства ввода-вывода, делятся на:

- блок-ориентированные;
- байт-ориентированные.

Каталог – это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны – это файл, содержащий системную информа-

цию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

- информация о разрешенном доступе;
- пароль для доступа к файлу;
- владелец файла;
- создатель файла;
- признак «только для чтения»;
- признак «скрытый файл»;
- признак «системный файл»;
- признак «архивный файл»;
- признак «двоичный/символьный»;
- признак «временный» (удалить после завершения процесса);
- признак блокировки;
- длина записи;
- указатель на ключевое поле в записи;
- длина ключа;
- времена создания, последнего доступа и последнего изменения;
- текущий размер файла;
- максимальный размер файла.

Параметры прав доступа в самом общем случае могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки – всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рис. 6.1).

В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории:

- владельца файла;
- членов его группы;
- всех остальных.

Имена файлов

Имена пользователей	modem.txt	win.exe	class.dbf	unix.ppt
kira	читать	выполнять	—	выполнять
genya	читать	выполнять	—	выполнять читать
nataly	читать	—	—	выполнять читать
victor	читать писать	—	создать	—

Рис. 6.1. Матрица прав доступа

Различают два основных подхода к определению прав доступа:

1. *избирательный доступ*, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
2. *мандатный подход*, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

6.3. Иерархическая структура файловой системы

Пользователи обращаются к файлам по символьным именам. Однако способности человеческой памяти ограничивают количество имен объектов, к которым пользователь может обращаться по имени. Иерархическая организация пространства имен позволяет значительно расширить эти границы. Именно поэтому большинство файловых систем имеет иерархическую структуру, в которой уровни создаются за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рис. 6.2).

Граф, описывающий иерархию каталогов, может быть деревом или сетью. Каталоги образуют *дерево*, если файлу разрешено входить только в один каталог (рис. 6.2б), и *сеть* – если файл может входить сразу в несколько каталогов (рис. 6.2в).

Например, в MS-DOS и Windows каталоги образуют древовидную структуру, а в UNIX – сетевую. В древовидной структуре каждый файл является листом. Каталог самого верхнего уровня называется корневым каталогом, или корнем (root).

Все файлы имеют символьные имена. В иерархически организованных файловых системах обычно используются три типа имен файлов:

- простые;
- составные;
- относительные (NFS).

Простое, или короткое, символьное имя идентифицирует файл в пределах одного каталога. Простые имена присваивают файлам пользователи при этом они должны учитывать ограничения ОС как на номенклатуру символов, так и на длину имени. Так, в популярной файловой системе FAT длина имен ограничивались схемой 8+3 (8 символов – собственно имя, 3 символа – расширение имени).

Современные файловые системы, а также усовершенствованные варианты уже существовавших файловых систем, как правило, поддерживают длинные простые символьные имена файлов. Например, в файловых системах NTFS и FAT32, входящих в состав операционной системы Windows, имя файла может содержать до 255 символов.

В иерархических файловых системах разным файлам разрешено иметь одинаковые простые символьные имена при условии, что они принадлежат разным каталогам. То есть здесь работает схема «много файлов –

одно простое имя». Для однозначной идентификации файла в таких системах используется так называемое полное имя.

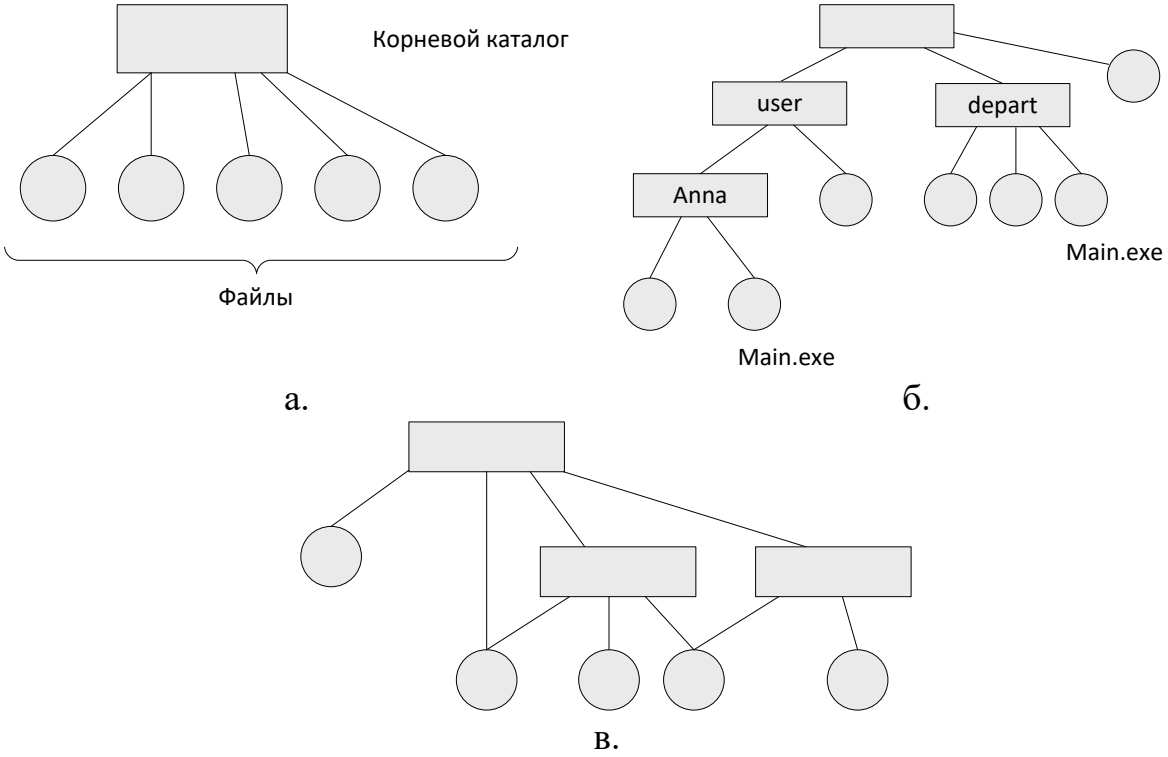


Рис. 6.2. Иерархия файловых систем

Полное имя файла представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до этого файла. Таким образом, полное имя является составным, в котором простые имена отделены друг от друга принятым в ОС разделителем. Часто в качестве разделителя используется прямой или обратный слеш, при этом принято не указывать имя корневого каталога. На рис. 6.2б два файла имеют простое имя main.exe, однако их составные имена /depart/main.exe и /user/anna/main.exe различаются.

В древовидной файловой системе между файлом и его полным именем имеется взаимно однозначное соответствие «один файл – одно полное имя».

В файловых системах, имеющих сетевую структуру, файл может входить в несколько каталогов, а значит, иметь несколько полных имен; здесь справедливо соответствие «один файл – много полных имен». В обоих случаях файл однозначно идентифицируется полным именем.

6.4. Понятие о монтировании

Монтирование – операция, в результате которой пользователю предоставляется возможность объединять файловые системы, находящиеся на разных устройствах, в единую файловую систему, описываемую единым деревом каталогов.

Рассмотрим, как осуществляется эта операция на примере ОС UNIX. Среди всех имеющихся в системе логических дисковых устройств операционная система выделяет одно устройство, называемое системным. Пусть имеются две файловые системы, расположенные на разных логических дисках (рис. 6.3), причем один, из дисков является системным.

Файловая система, расположенная на системном диске, назначается корневой. Для связи иерархий файлов в корневой файловой системе выбирается некоторый существующий каталог, в этом примере – каталог *man*. После выполнения монтирования выбранный каталог *man* становится корневым каталогом второй файловой системы. Через этот каталог монтируемая файловая система подсоединяется как поддерево к общему дереву (рис. 6.4).

После монтирования общей файловой системы для пользователя нет логической разницы между корневой и смонтированной файловыми системами, в частности именование файлов производится так же, как если бы она с самого начала была единой.

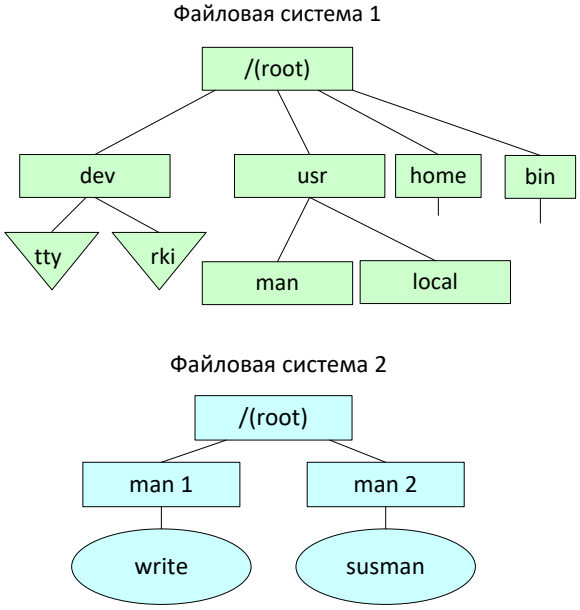


Рис. 6.3. Две файловые системы до монтирования

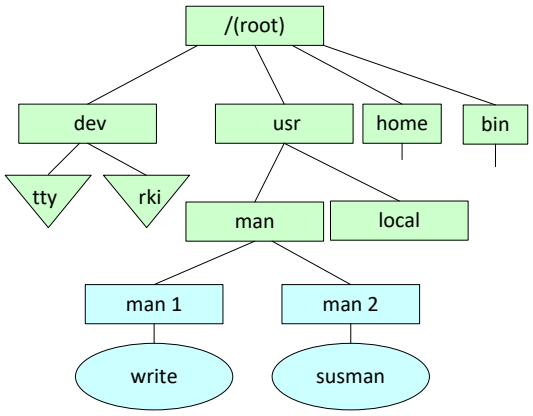


Рис. 6.4. Общая файловая система после монтирования

6.5. Физическая организация файловой системы

Представление пользователя о файловой системе как об иерархически организованном множестве информационных объектов имеет мало общего с порядком хранения файлов на диске. Файл, имеющий образ цельного, непрерывающегося набора байт, на самом деле очень часто разбросан «кусочками» по всему диску, причем это разбиение никак не связано с логической структурой файла, например, его отдельная логическая запись может быть расположена в несмежных секторах диска. Логически объединенные файлы из одного каталога совсем не обязаны соседствовать

на диске. Принципы размещения файлов, каталогов и системной информации на реальном устройстве описываются физической организацией файловой системы.

Основным типом устройства, которое используется в современных вычислительных системах для хранения файлов, являются *дисковые накопители.* Жесткий диск состоит из одной или нескольких стеклянных или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом. Таким образом, диск в общем случае состоит из пакета *пластин* (рис. 6.5).

На каждой стороне каждой пластины размечены тонкие концентрические кольца – *дорожки* (traks), на которых хранятся данные. Количество дорожек зависит от типа диска. Когда диск вращается, элемент, называемый *головкой*, считывает двоичные данные с магнитной дорожки или записывает их на магнитную дорожку.

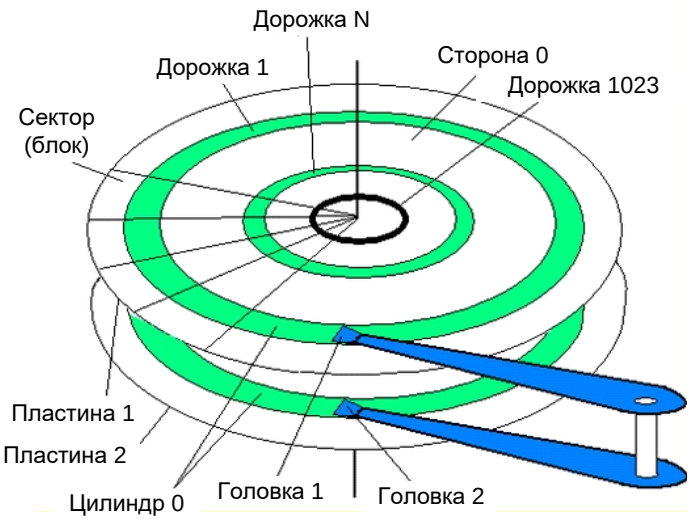


Рис. 6.5. Схема устройства жесткого диска

Головка позиционируются над заданной дорожкой. Головки перемещаются над поверхностью диска дискретными шагами, каждый шаг соответствует сдвигу на одну дорожку. В дисках имеется по головке на каждую дорожку. Обычно все головки закреплены на едином перемещающем механизме и двигаются синхронно. Поэтому, когда головка фиксируется на заданной дорожке одной поверхности, все остальные головки останавливаются над дорожками с такими же номерами.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин пакета называется *цилиндром* (cylinder).

Каждая дорожка разбивается на фрагменты, называемые *секторами* (sectors), или *блоками* (blocks), так что все дорожки имеют равное число секторов, в которые можно максимально записать одно и то же число байт.

Сектор – наименьшая адресуемая единица обмена данными дискового устройства с оперативной памятью. Для того чтобы контроллер мог

найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора:

- номер цилиндра;
- номер поверхности;
- номер сектора.

Дорожки и секторы создаются в результате выполнения процедуры физического, или низкоуровневого, форматирования диска, предшествующей использованию диска.

Разметку диска под конкретный тип файловой системы выполняют процедуры высокоуровневого, или логического, форматирования. При высокоуровневом форматировании определяется размер кластера и на диск записывается информация, необходимая для работы файловой системы, в том числе:

- информация о доступном и неиспользуемом пространстве;
- о границах областей, отведенных под файлы и каталоги;
- информация о поврежденных областях;
- загрузчик операционной системы – небольшая программа, которая начинает процесс инициализации операционной системы после включения питания или рестарта компьютера.

Прежде чем форматировать диск под определенную файловую систему, он может быть разбит на разделы.

Раздел – это непрерывная часть физического диска, которую операционная система представляет пользователю как логическое устройство (используются также названия логический диск и логический раздел)[1]. Логическое устройство функционирует так, как если бы это был отдельный физический диск. Именно с логическими устройствами работает пользователь, обращаясь к ним по символьным именам, используя, например, обозначения *A*:, *B*:, *C*:. На каждом логическом устройстве может создаваться только одна файловая система.

На разных логических устройствах одного и того же физического диска могут располагаться файловые системы разного типа. На рис. 6.6 показан пример диска, разбитого на три раздела, в которых установлены две файловых системы NTFS (разделы *C*: и *E*:) и одна файловая система FAT (раздел *D*:).

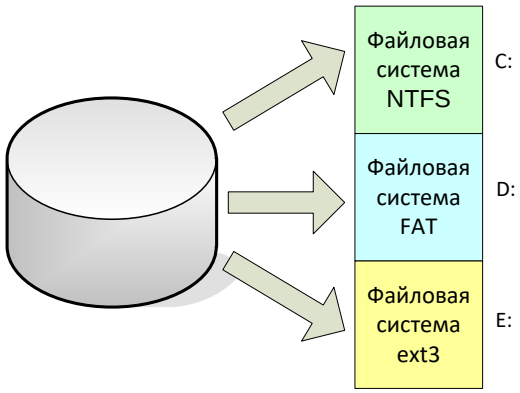


Рис. 6.6. Разбиение диска на разделы

6.6. Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рис. 6.7), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Рис. 6.7. Общая модель файловой системы

Задачей *символьного уровня* является определение по символьному имени файла его уникального имени. В файловых системах, в которых один и тот же файл может иметь несколько символьных имен, то на этом уровне просматривается цепочка каталогов для определения уникального имени файла.

На следующем, *базовом уровне* по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу.

Следующим этапом реализации запроса к файлу является *проверка прав доступа* к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к этому файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.

На *логическом уровне* определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая запись. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт.

На *физическом уровне* файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня – смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока.

После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

6.7. Понятие о журналируемых файловых системах

Журналируемые файловые системы – это класс файловых систем, характерная черта которых – ведение журнала, хранящего список изменений, в той или иной степени помогающего сохранить целостность файловой системы.

Причиной отсутствия целостности в файловой системе может быть некорректное размонтирование, сбой в момент обновления данных, а также ошибки (отсутствие целостности) в файлах данных. Сюда же можно включить ошибки в метаданных файловой системы, которые могут привести к потерям файлов и другим серьезным проблемам.

Для минимизации проблем, связанных с целостностью, журналируемая файловая система хранит список изменений, которые она будет проводить с файловой системой перед фактической записью изменений. Эти записи хранятся в отдельной части файловой системы, называемой «*журналом*». Как только изменения файловой системы безопасно внесены в журнал, журналируемая файловая система применяет эти изменения к файлам или метаданным, а затем удаляет эти записи из журнала. Записи журнала организованы в наборы связанных изменений файловой системы, что очень похоже на то, как изменения, добавляемые в базу данных организованы в транзакции.

Наличие журнала повышает вероятность сохранения целостности файловой системы, потому что записи в лог-файл ведутся до проведения фактических изменений, и эти записи хранятся до тех пор, пока они не будут целиком и безопасно применены. В случае сбоя в компьютере программа монтирования может гарантировать целостность журналируемой файловой системы простой проверкой «журнала» на наличие ожидаемых,

но не произведенных изменений и последующей записью их в файловую систему. Таким образом при наличии журнала в большинстве случаев системе не нужно проводить проверку целостности файловой системы, а это означает, что компьютер будет доступен для работы практически сразу после перезагрузки. Соответственно, шансы потери данных в связи с проблемами в файловой системе значительно снижаются.

6.8. Физическая организация и адресация в файле

Важным компонентом физической организации файловой системы является *физическая организация файла*, то есть способ размещения файла на диске.

Основными критериями эффективности физической организации файлов являются:

- скорость доступа к данным;
- объем адресной информации файла;
- степень фрагментированности дискового пространства;
- максимально возможный размер файла.

Непрерывное размещение – простейший вариант физической организации (рис. 6.8а), при котором файлу предоставляется последовательность кластеров диска, образующих непрерывный участок дисковой памяти. Основным достоинством этого метода является высокая скорость доступа, так как затраты на поиск и считывание кластеров файла минимальны. Также минимален объем адресной информации – достаточно хранить только номер первого кластера и объем файла. Данная физическая организация максимально возможный размер файла не ограничивает. Однако этот вариант имеет существенные недостатки, которые затрудняют его применимость на практике, несмотря на всю его логическую простоту. Серьезной проблемой является фрагментация. Поэтому на практике используются методы, в которых файл размещается в нескольких, в общем случае несмежных областях диска.

Размещение файла в виде связанного списка кластеров дисковой памяти (рис. 6.8б) в этом способе в начале каждого кластера содержится указатель на следующий кластер. В этом случае адресная информация минимальна: расположение файла может быть задано одним числом – номером первого кластера. В отличие от предыдущего способа каждый кластер может быть присоединен к цепочке кластеров какого-либо файла, следовательно, фрагментация на уровне кластеров отсутствует. Файл может изменять свой размер во время своего существования, наращивая число кластеров. Недостатком является сложность реализации доступа к произвольно заданному месту файла – чтобы прочитать пятый по порядку кластер файла, необходимо последовательно прочитать четыре первых кластера, прослеживая цепочку номеров кластеров.

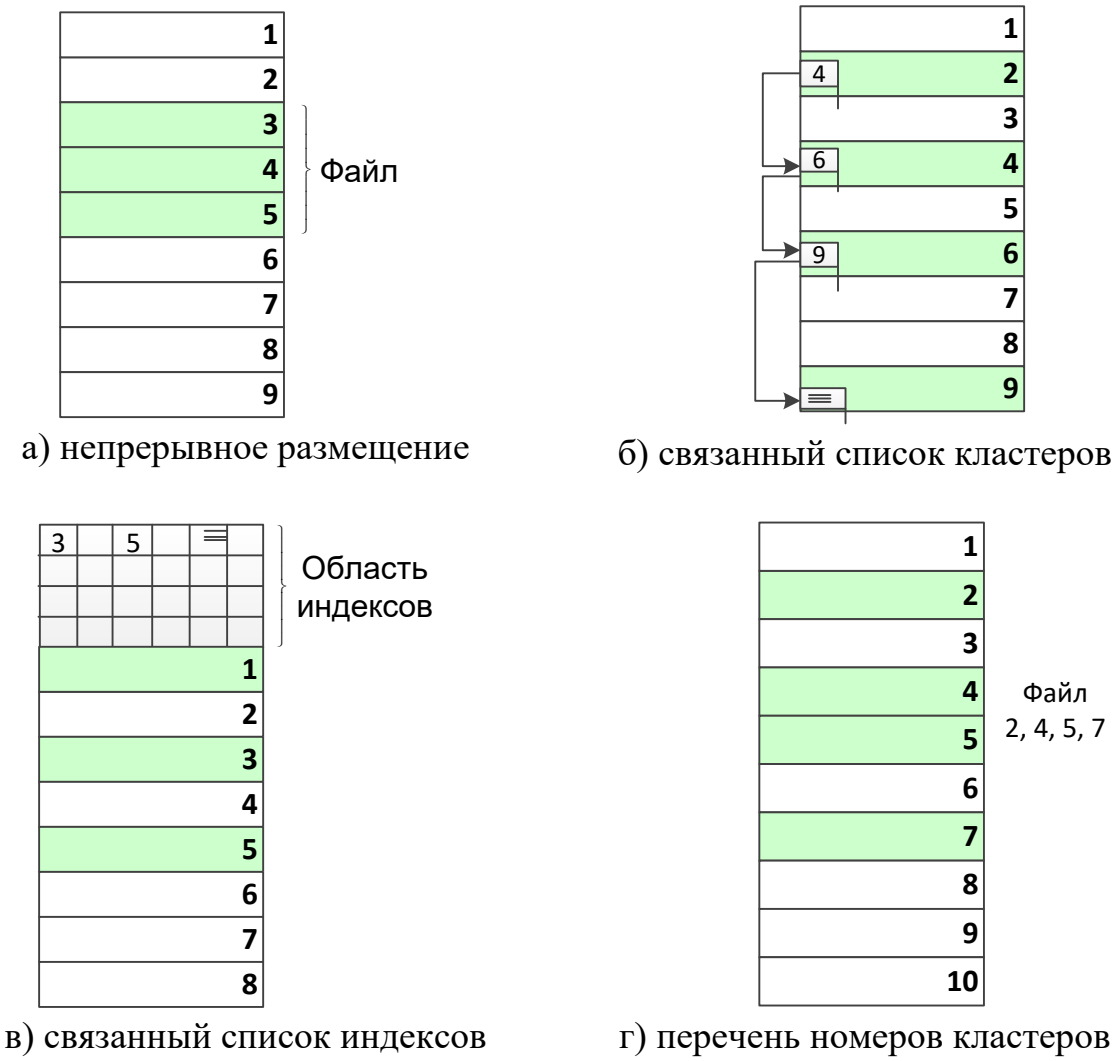


Рис. 6.8. Физическая организация файла

Использование связанного списка индексов (рис. 6.8в) – применяется, в файловой системе FAT. Этот способ является некоторой модификацией предыдущего. Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связывается некоторый элемент – *индекс.* Индексы располагаются в отдельной области диска – в MS-DOS это таблица FAT (File Allocation Table), занимающая один кластер. Когда память свободна, все индексы имеют нулевое значение. Если некоторый кластер *N* назначен некоторому файлу, то индекс этого кластера становится равным либо номеру *M* следующего кластера этого файла, либо принимает специальное значение, являющееся признаком того, что этот кластер является для файла последним. Индекс же предыдущего кластера файла принимает значение *N*, указывая на вновь назначенный кластер.

При такой физической организации сохраняются все достоинства предыдущего способа: минимальность адресной информации, отсутствие фрагментации, отсутствие проблем при изменении размера.

Перечисление номеров кластеров, занимаемых файлом (рис. 6.8г), при этом данный перечень и служит адресом файла. Недостаток этого способа очевиден: длина адреса зависит от размера файла и для большого файла может составить значительную величину. Достоинством же является высокая скорость доступа к произвольному кластеру файла, так как здесь применяется прямая адресация, которая исключает просмотр цепочки указателей при поиске адреса произвольного кластера файла. Фрагментация на уровне кластеров в этом способе также отсутствует.

7. Особенности построения современных файловых систем

7.1. Файловая система FAT

7.1.1. Структура файлов FAT

Логический раздел, содержащий файловую систему FAT (File Allocation Table), состоит из следующих областей.

- *Загрузочный сектор* содержит программу начальной загрузки операционной системы. Вид этой программы зависит от типа операционной системы, которая будет загружаться из этого раздела.
- *Основная копия FAT* содержит информацию о размещении файлов и каталогов на диске.
- *Резервная копия FAT.*
- *Корневой каталог* занимает фиксированную область размером в 32 сектора (16 Кбайт), что позволяет хранить 512 записей о файлах и каталогах, так как каждая запись каталога состоит из 32 байт.
- *Область данных* предназначена для размещения всех файлов и всех каталогов, кроме корневого каталога.

Файловая система FAT поддерживает всего два типа файлов:

- обычный файл;
- каталог.

Файловая система распределяет память только из области данных, причем использует в качестве минимальной единицы дискового пространства кластер.

Таблица FAT (как основная копия, так и резервная) состоит из массива индексных указателей, количество которых равно количеству кластеров области данных. Между кластерами и индексными указателями имеется взаимно однозначное соответствие – нулевой указатель соответствует нулевому кластеру и т. д.

Индексный указатель может принимать следующие значения, характеризующие состояние связанного с ним кластера:

- кластер свободен (не используется);
- кластер используется файлом и не является последним кластером файла; в этом случае индексный указатель содержит номер следующего кластера файла;
- последний кластер файла;
- дефектный кластер;
- резервный кластер.

Таблица FAT является общей для всех файлов раздела. В исходном состоянии (после форматирования) все кластеры раздела свободны и все

индексные указатели (кроме тех, которые соответствуют резервным и дефектным блокам) принимают значение «кластер свободен».

При размещении файла ОС просматривает FAT, начиная с начала, и ищет первый свободный индексный указатель. После его обнаружения в поле записи каталога «номер первого кластера» фиксируется номер этого указателя. В кластер с этим номером записываются данные файла, он становится первым кластером файла. Если файл умещается в одном кластере, то в указатель, соответствующий этому кластеру, заносится специальное значение «последний кластер файла». Если размер файла больше одного кластера, то ОС продолжает просмотр FAT и ищет следующий указатель на свободный кластер. После его обнаружения в предыдущий указатель заносится номер этого кластера, который теперь становится следующим кластером файла (рис. 7.1). Процесс повторяется до тех пор, пока не будут размещены все данные файла. Таким образом, создается связный список всех кластеров файла.

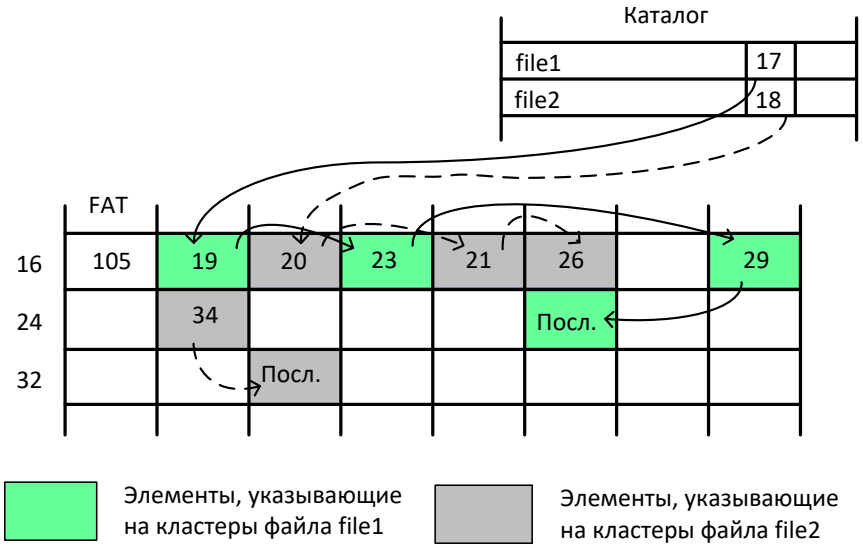


Рис. 7.1. Списки указателей файлов в FAT

Размер таблицы FAT и разрядность используемых в ней индексных указателей определяется количеством кластеров в области данных. Для уменьшения потерь из-за фрагментации желательно кластеры делать небольшими, а для сокращения объема адресной информации и повышения скорости обмена наоборот – чем больше, тем лучше. При форматировании диска под файловую систему FAT обычно выбирается компромиссное решение и размеры кластеров выбираются из диапазона от 1 до 128 секторов, или от 512 байт до 64 Кбайт.

Очевидно, что разрядность индексного указателя должна быть такой, чтобы в нем можно было задать максимальный номер кластера для диска определенного объема. Существует несколько разновидностей FAT, отличающихся разрядностью индексных указателей, которая и используется в качестве условного обозначения:

- FAT16 – используются 16-разрядные указатели, что позволяет поддерживать до 65 536 кластеров в области данных диска;

- FAT32 – использует 32-разрядные указатели, поддерживая для более чем 4 млрд кластеров.

Таблица FAT при фиксированной разрядности индексных указателей имеет переменный размер, зависящий от объема области данных диска.

При удалении файла из файловой системы FAT в первый байт соответствующей записи каталога заносится специальный признак, свидетельствующий о том, что эта запись свободна, а во все индексные указатели файла заносится признак «кластер свободен». Остальные данные в записи каталога, в том числе номер первого кластера файла, остаются нетронутыми, что оставляет шансы для восстановления ошибочно удаленного файла.

Резервная копия FAT всегда синхронизируется с основной копией при любых операциях с файлами, поэтому резервную копию нельзя использовать для отмены ошибочных действий пользователя, выглядевших с точки зрения системы вполне корректными. Резервная копия может быть полезна только в том случае, когда секторы основной памяти оказываются физически поврежденными и не читаются.

Используемый в FAT метод хранения адресной информации о файлах не отличается большой надежностью – при разрыве списка индексных указателей в одном месте, например, из-за сбоя в работе программного кода ОС по причине внешних электромагнитных помех, теряется информация обо всех последующих кластерах файла.

Файловые системы FAT12 и FAT16 получили большое распространение благодаря их применению в операционных системах MS-DOS и Windows 3.x – самых массовых операционных системах первого десятилетия эры персональных компьютеров. По этой причине эти файловые системы поддерживаются сегодня и другими ОС, такими как UNIX, Linux, Windows. Однако из-за постоянно растущих объемов жестких дисков, а также возрастающих требований к надежности, эти файловые системы практически вытеснены как системой FAT32, впервые появившейся в Windows 95, так и файловыми системами других типов.

7.1.2. Ограничения FAT в Windows

Штатными средствами Windows невозможно создать разделы FAT32 более 32 ГБ, однако, с такими разделами возможно работать, если они были предварительно созданы в других ОС. Причина этого заключается в том, что, по мнению компании Microsoft, при увеличении размера тома FAT32 выше 32 ГБ резко падает производительность, и что более подходящее решение – использование файловой системы NTFS. Но поскольку NTFS нецелесообразно использовать на флеш-накопителях, то была разработана специальная файловая система exFAT, снимающая ряд ограничений.

Максимально возможный размер файла для тома FAT32 – ~ 4 ГБ – 4 294 967 295 байт (232-1 – 4 294 967 295 байт) – это весьма важный фактор для смены файловой системы. FAT32 не поддерживает установку раз-

решений на доступ к файлам и папкам и некоторые другие функции современных файловых систем. Все эти причины привели к тому, что сейчас наблюдается тенденция отказа от FAT32 в пользу более продвинутых файловых систем, таких как NTFS, Ext2/Ext3.

7.1.3. Файловая система exFAT

exFAT (от англ. Extended FAT – «расширенная FAT») – проприетарная файловая система компании Microsoft, предназначенная главным образом для флэш-накопителей.

Основными преимуществами exFAT перед предыдущими версиями FAT являются:

- уменьшение количества перезаписей одного и того же сектора, что очень важно для флеш-накопителей, у которых ячейки памяти необратимо изнашиваются после определённого количества операций записи. Это была основная причина разработки ExFAT;
- теоретический лимит на размер файла 264 байт (16 экзабайт);
- максимальный размер кластера увеличен до 225 байт (32 Мбайта);
- улучшение распределения свободного места за счёт введения битмапы свободного места, что может уменьшать фрагментацию диска;
- отсутствие лимита на количество файлов в одной директории;
- введена поддержка списка прав доступа;
- поддержка транзакций (опциональная возможность, должна поддерживаться устройством).

7.2. Файловая система NTFS

Файловая система NTFS (New Technology File System) была разработана в качестве основной файловой системы для ОС Windows 2000 в начале 1990-х годов с учетом опыта разработки файловых систем FAT и HPFS (ранее – основная файловая система для OS/2), а также других существовавших в то время файловых систем.

Основными отличительными свойствами NTFS являются:

- поддержка больших файлов (до 16 терабайт) и больших дисков (объемом до 255 терабайт);
- восстанавливаемость после сбоев и отказов программ и аппаратуры управления дисками;
- высокая скорость операций, в том числе и для больших дисков;
- низкий уровень фрагментации, в том числе и для больших дисков;
- гибкая структура, допускающая развитие за счет добавления новых типов записей и атрибутов файлов с сохранением совместимости с предыдущими версиями ФС;
- устойчивость к отказам дисковых накопителей;
- поддержка длинных символьных имен;
- контроль доступа к каталогам и отдельным файлам.

7.2.1. Структура тома NTFS

В отличие от разделов FAT все пространство тома (том – раздел диска в терминологии Windows) NTFS представляет собой либо файл, либо часть файла.

Основой структуры тома NTFS является *главная таблица файлов* MFT (Master File Table), которая содержит по крайней мере одну запись для каждого файла тома, включая одну запись для самой себя. Каждая запись MFT имеет фиксированную длину, зависящую от объема диска – 1,2 или 4 Кбайт. Для большинства дисков, используемых сегодня, размер записи MFT равен 2 Кбайт, который мы далее будет считать размером записи по умолчанию.

Все файлы на томе NTFS идентифицируются номером файла, который определяется позицией файла в MFT. Весь том NTFS состоит из последовательности кластеров.

Порядковый номер кластера в томе NTFS называется *логическим номером кластера* LCN (Logical Cluster Number).

Файл NTFS также состоит из последовательности кластеров, при этом порядковый номер кластера внутри файла называется *виртуальным номером кластера* VCN (Virtual Cluster Number).

Базовая единица распределения дискового пространства для файловой системы NTFS – непрерывная область кластеров, называемая *отрезком*. В качестве адреса отрезка NTFS использует логический номер его первого кластера, а также количество кластеров в отрезке *k*, то есть пара (LCN, *k*). Таким образом, часть файла, помещенная в отрезок и начинающаяся с виртуального кластера VCN, характеризуется адресом, состоящим из трех чисел: (VCN, LCN, *k*).

Для хранения номера кластера в NTFS используются 64-разрядные указатели, что дает возможность поддерживать тома и файлы размером до 2^{64} кластеров. При размере кластера в 4 Кбайт это позволяет использовать тома и файлы, состоящие из 64 млрд килобайт.

Структура тома NTFS показана на рис. 7.2. Загрузочный блок тома NTFS располагается в начале тома, а его копия – в середине тома. Загрузочный блок содержит стандартный блок параметров BIOS, количество блоков в томе, а также начальный логический номер кластера основной копии MFT и зеркальную копию MFT.

Рис. 7.2. Структура тома NTFS

Далее располагается первый отрезок MFT, содержащий 16 стандартных, создаваемых при форматировании записей о системных файлах NTFS. Назначение этих файлов описано в приведенной ниже таблице MFT.

В NTFS файл целиком размещается в записи таблицы MFT, если это позволяет сделать его размер. В том же случае, когда размер файла больше размера записи MFT, в запись помещаются только некоторые атрибуты файла, а остальная часть файла размещается в отдельном отрезке тома (или нескольких отрезках). Часть файла, размещаемая в записи MFT, называется резидентной частью, а остальные части – нерезидентными. Адресная информация об отрезках, содержащих нерезидентные части файла, размещается в атрибутах резидентной части.

Таблица 7.1 – Записи о системных файлах NTFS в MFT

Номер записи	Системный файл	Имя файла	Назначение файла
0	Главная таблица файлов	$Mft	Содержит полный список файлов тома NTFS
1	Копия главной таблицы файлов	SMftMirr	Зеркальная копия первых трех записей MFT
2	Файл журнала	SLogFile	Список транзакций, который используется для восстановления файловой системы после сбоев
3	Том	SVolume	Имя тома, версия NTFS и другая информация о томе
4	Таблица определения атрибутов	SAttrDef	Таблица имен, номеров и описаний атрибутов
5	Индекс корневого каталога	$.	Корневой каталог
6	Битовая карта кластеров	SBitmap	Разметка использованных кластеров тома
7	Загрузочный сектор раздела	SBoot	Адрес загрузочного сектора раздела
8	Файл плохих кластеров	SBadClus	Файл, содержащий список всех обнаруженных на томе плохих кластеров
9	Таблица квот	SQuota	Квоты используемого пространства на диске для каждого пользователя
10	Таблица преобразования регистра символов	SUpcase	Используется для преобразования регистра символов для кодировки Unicode
11-15	Зарезервированы для будущего использования		

7.2.2. Структура файлов NTFS

Каждый файл и каталог на томе NTFS состоит из *набора атрибутов.* Важно отметить, что имя файла и его данные также рассматриваются как атрибуты файла, то есть в трактовке NTFS кроме атрибутов у файла нет никаких других компонентов.

Каждый атрибут файла NTFS состоит из полей: тип атрибута, длина атрибута, значение атрибута и, возможно, имя атрибута. Имеется системный набор атрибутов, определяемых структурой тома NTFS. Системные атрибуты имеют фиксированные имена и коды их типа, а также определенный формат.

Файлы NTFS в зависимости от способа размещения делятся на:

- небольшие;
- большие;

- очень большие;
- сверхбольшие.

Небольшие файлы (small) – файлы небольшого размера, целиком располагаться внутри одной записи MFT, имеющей, например, размер 2 Кбайт.

Небольшие файлы NTFS состоят по крайней мере из следующих атрибутов (рис. 7.3):

- стандартная информация (SI – Standard Information);
- имя файла (FN – File Name);
- данные (Data);
- дескриптор безопасности (SD – Security Descriptor).

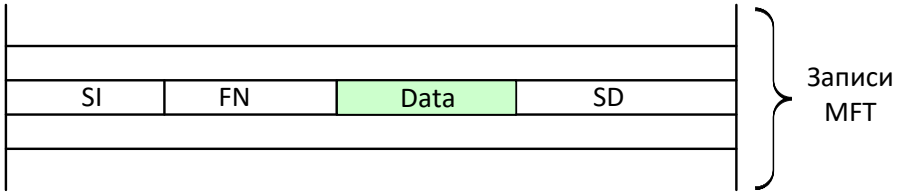


Рис. 7.3. Небольшой файл NTFS

Большие файлы (large). Если данные файла не помещаются в одну запись МЕТ, то этот факт отражается в заголовке атрибута Data, который содержит признак того, что этот атрибут является нерезидентным, то есть находится в отрезках вне таблицы MFT. В этом случае атрибут Data содержит адресную информацию (LCN, VCN, k) каждого отрезка данных (рис. 7.4).

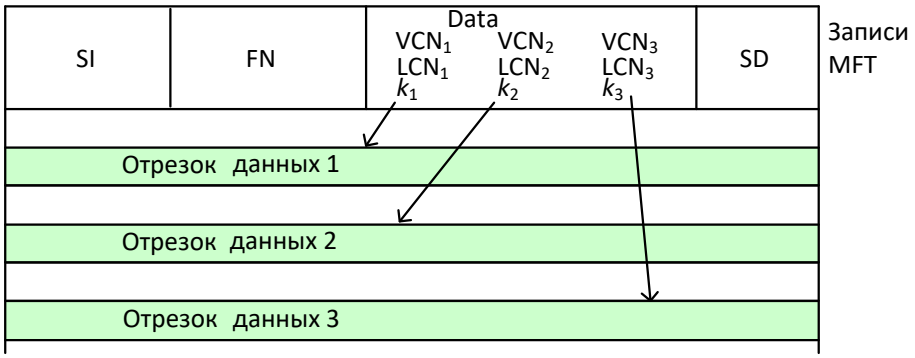


Рис. 7.4. Большой файл

Очень большие файлы (huge). Если файл настолько велик, что его атрибут данных, хранящий адреса нерезидентных отрезков данных, не помещается в одной записи, то этот атрибут помещается в другую запись MFT, а ссылка на такой атрибут помещается в основную запись файла (рис. 7.5). Эта ссылка содержится в атрибуте Attribute List. Сам атрибут данных по-прежнему содержит адреса нерезидентных отрезков данных.

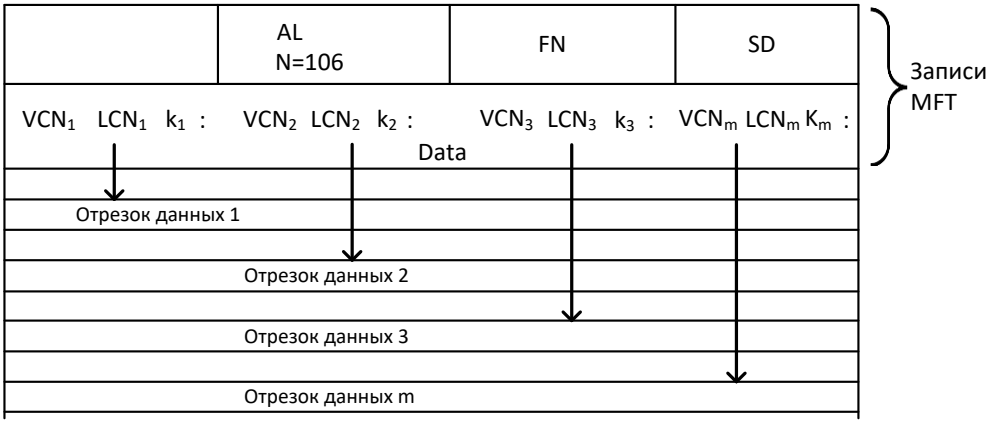


Рис. 7.5. Очень большой файл

Сверхбольшие файлы (Extremely Huge). Для сверхбольших файлов в атрибуте Attribute List можно указать несколько атрибутов, расположенных в дополнительных записях MFT (рис. 7.6). Кроме того, можно использовать двойную косвенную адресацию, когда нерезидентный атрибут будет ссылаться на другие нерезидентные атрибуты, поэтому в NTFS не может быть атрибутов слишком большой для системы длины.

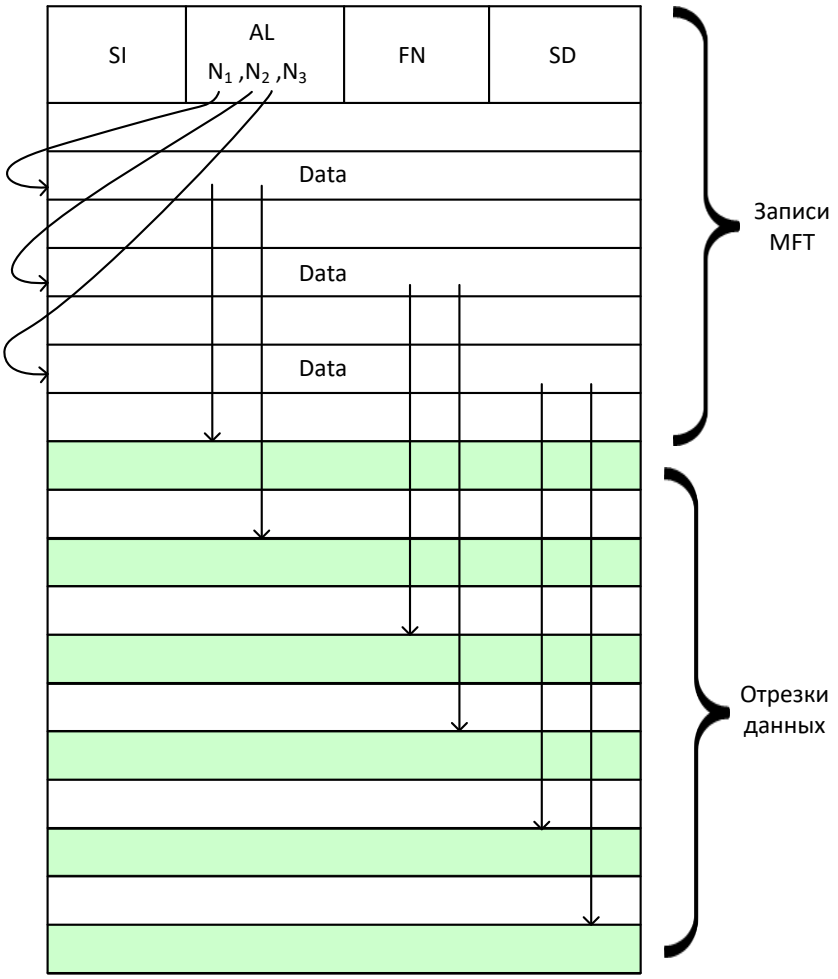
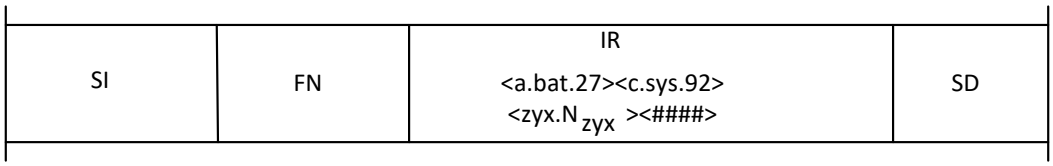


Рис. 7.6. Сверхбольшой файл

7.2.3. Каталоги NTFS

Каталог NTFS представляет собой один вход в таблицу MFT, который содержит атрибут Index Root. Индекс содержит список файлов, входящих в каталог. Индексы позволяют сортировать файлы для ускорения поиска, основанного на значении определенного атрибута. Обычно в файловых системах файлы сортируются по имени. NTFS позволяет использовать для сортировки любой атрибут.



###- признак конца файлов

Рис. 7.7 Небольшой каталог

Имеются две формы хранения списка файлов:

- *небольшие каталоги* (Small Indexes). Если количество файлов в каталоге невелико, то список файлов может быть резидентным в записи в MFT, являющейся каталогом (рис. 7.7). Для резидентного хранения списка используется единственный атрибут – Index Root. Список файлов содержит значения атрибутов файла. По умолчанию – это имя файла, а также номер записи MTF, содержащей начальную запись файла;
- *большие каталоги* (Large Indexes). По мере того как каталог растет, список файлов может потребовать нерезидентной формы хранения. Однако начальная часть списка всегда остается резидентной в корневой записи каталога в таблице MFT (рис. 7.8). Имена файлов резидентной части списка файлов являются узлами так называемого B-дерева (двоичного дерева). Остальные части списка файлов размещаются вне MFT. Для их поиска используется специальный атрибут *Index Allocation*, представляющий собой адреса отрезков, хранящих остальные части списка файлов каталога.

Поиск в каталоге уникального имени файла, которым в NTFS является номер основной записи о файле в MFT, по его символьному имени происходит следующим образом. Сначала искомое символьное имя сравнивается с именем первого узла в резидентной части индекса. Если искомое имя меньше, то это означает, что его нужно искать в первой нерезидентной группе, для чего из атрибута *Index Allocation* извлекается адрес отрезка (VCN_j, LCN_j, K_j), хранящего имена файлов первой группы. Среди имен этой группы поиск осуществляется прямым перебором имен и сравнением до полного совпадения всех символов искомого имени с хранящимся в каталоге именем. При совпадении из каталога извлекается номер основной записи о файле в MFT и остальные характеристики файла берутся уже оттуда.

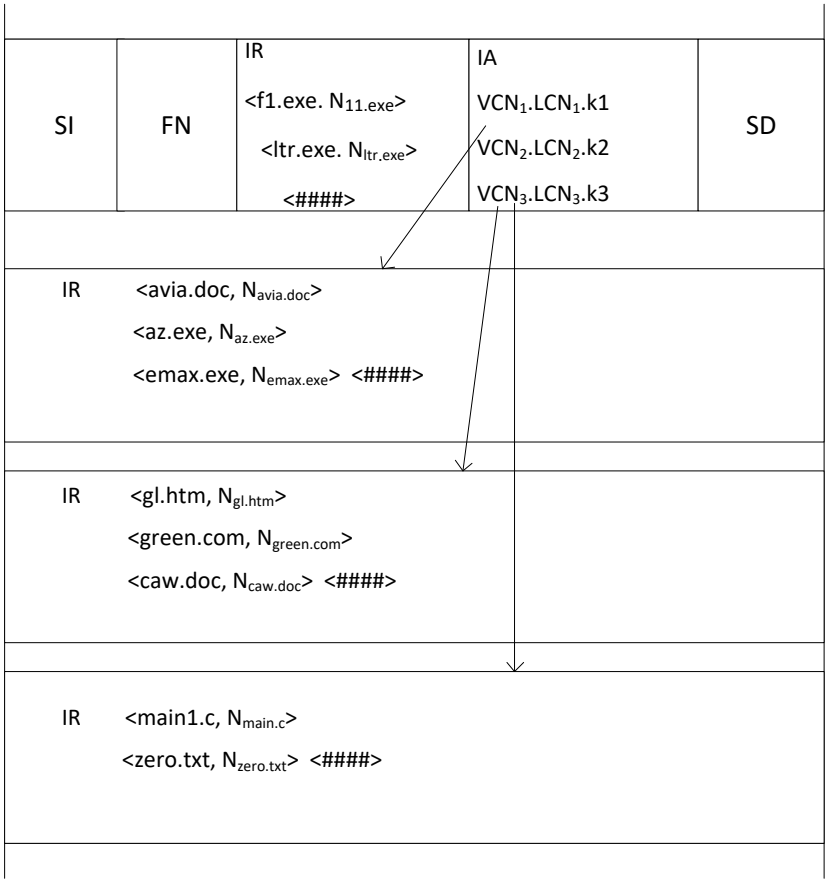


Рис. 7.8. Большой каталог

7.3. Файловая система ext 2/3/4

На заре своего развития Linux использовала файловую систему Minix. Эта файловая система была довольно стабильна, но была 16 разрядной и как следствие имела жесткое ограничение в 64 Мегабайта на раздел. Также присутствовало ограничение имени файла: оно составляло 14 символов. Эти и не только ограничения повлекли появление в апреле 1992 г. «расширенной файловой системы» (Extended File System), решавшей две главные проблемы Minix. Новая файловая система расширила ограничения на размер файла до 2 гигабайт и установила предельную длину имени файла в 255 символов. Но она все равно имела проблемы: не было поддержки раздельного доступа, временных меток модификации данных. Решением всех проблем стала новая файловая система, разработанная в январе 1993 г. В ext2 были сразу реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов.

7.3.1. Логическая организация файловой системы ext2

Граф, описывающий иерархию каталогов, файловой системы ext2 представляет собой *сеть*, это достигается тем, что один файл может входить сразу в несколько каталогов.

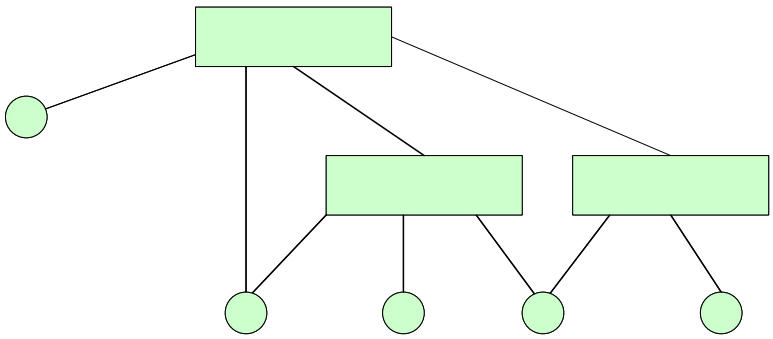


Рис. 7.9. Сетевая иерархия каталогов файловой системы ext2

Все типы файлов имеют символьные имена. Ограничения на простое имя состоят в том, что его длина не должна превышать 255 символов, а также в имени не должны присутствовать символ NUL и «/». Ограничения на символ NUL связаны с представлением строк на языке Си, а на символ «/» с тем, что он используются как разделительный символ между каталогами.

Полное имя представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до конкретного файла. В файловой системе ext2 файл может входить в несколько каталогов, а значит, иметь несколько полных имен; здесь справедливо соответствие «один файл – много полных имен». В любом случае полное имя однозначно определяет файл.

Атрибуты файлов хранятся не в каталогах, как это сделано в ряде простых файловых систем, а в специальных таблицах. В результате каталог имеет очень простую структуру, состоящую всего из двух частей:

- номера индексного дескриптора;
- имени файла.

В ФС ext2 существует один корневой каталог, который называется «/» («слэш»). Пользователь Linux всегда работает с единым деревом каталогов, даже если разные данные расположены на разных носителях: нескольких жестких или сетевых дисках, съемных дисках, CD-ROM и т. п.

Положение любого каталога в дереве каталогов точно и однозначно описывается при помощи полного пути. Полный путь всегда начинается от корневого каталога и состоит из перечисления всех вершин, встретившихся при движении по ребрам дерева до искомого каталога включительно. Названия соседних вершин разделяются символом «/». В корневом каталоге находятся только подкаталоги со стандартными именами, соответствующими стандарту FHS (Filesystem Hierarchy Standard – стандартная структура файловых систем).

Приведем краткое описание подкаталогов корневого каталога:

/bin – название происходит от слова «binaries» («двоичные», «исполняемые»). В этом каталоге находятся исполняемые файлы самых необходимых утилит. Сюда попадают такие программы, которые могут понадобиться системному адми-

нистратору или другим пользователям для устранения неполадок в системе или при восстановлении после сбоя;

/boot – «Boot» – загрузка системы. В этом каталоге находятся файлы, необходимые для самого первого этапа загрузки ядра, и, обычно, само ядро. Пользователю практически никогда не требуется непосредственно работать с этими файлами;

/dev – в этом каталоге находятся файлы особого типа, предназначенные для обращения к различным системным ресурсам и устройствам (англ. «devices» «устройства», отсюда и сокращенное название каталога). Например, файлы /dev/ttyN соответствуют виртуальным консолям, где N – номер виртуальной консоли;

/etc – каталог для системных конфигурационных файлов. Здесь хранится информация о специфических настройках системы: информация о зарегистрированных пользователях, доступных ресурсах, настройках различных программ;

/home – здесь расположены каталоги, принадлежащие пользователям системы домашние каталоги, отсюда и название «home». Преимущество: серьезное повреждение системы или необходимость обновления не затронет пользовательских файлов;

/lib – название этого каталога сокращение от «librarie» (англ. «библиотек»). Это собрания стандартных функций, например, библиотеки, необходимые для работы наиболее важных системных утилит (размещенных в /bin и /sbin);

/mnt – каталог для монтирования (от англ. «mount») временного подключения файловых систем, например, на съемных носителях (CD-ROM и др.);

/proc – в этом каталоге все файлы «виртуальны» они располагаются не на диске, а в оперативной памяти. В этих файлах содержится информация о программах (процессах), выполняемых в конкретный момент времени в системе;

/root – домашний каталог администратора системы – пользователя root. Размещен отдельно от домашних каталогов остальных пользователей, так должен присутствовать в любой ситуации;

/sbin – каталог для важнейших системных утилит (название каталога сокращение от «system binarie»): в дополнение к утилитам /bin здесь находятся программы, необходимые для загрузки, резервного копирования, восстановления системы. Полномочия на исполнение этих программ есть только у системного администратора;

/tmp – этот каталог предназначен для временных файлов: в таких файлах программы хранят необходимые для работы проме-

жуточные данные. После завершения работы программы временные файлы теряют смысл и должны быть удалены. Обычно каталог /tmp очищается при каждой загрузке системы;

/usr – каталог, содержащий динамически компонуемые программы, файлы пользователей и программы, устанавливаемые вручную. Он должен содержать только не изменяющиеся программами данные (то есть /usr в режиме эксплуатации может быть смонтирован в режиме «только для чтения» без ущерба для функциональности);

/var – название сокращение от «variable» («переменные» данные). Здесь размещаются данные, которые создаются в процессе работы разными программами и предназначены для передачи другим программам и системам (очереди печати, электронной почты и др.) или для сведения системного администратора. В отличие от каталога /tmp сюда попадают данные, которые могут понадобиться после того, как создавшая их программа завершила работу.

Индексный узел каждого файла содержит информацию, используемую ядром для поддержки политик контроля доступа. В файловой системе ext2, индексные узлы включают два поля, связанные с безопасностью:

- поле файловых разрешений (file permission);
- файловых атрибутов (file attribute).

Файловые разрешения определяют права на чтение (r), запись (w) и исполнение (x) для трех категорий пользователей (для каталогов право на исполнение и право на чтение всегда идут вместе и означают одно и то же):

- владелец файла (owner) – пользователь, создавший файл;
- группа пользователей (group), которые иметь доступ к файлу (обычно, группа, к которой принадлежит пользователь, создавший файл);
- остальные пользователи (others).

Файловые атрибуты контролируют возможности модификации данных. Например, файловый атрибут только добавление (append-only) означает, что пользователи могут добавлять данные к файлу, но не могут модифицировать данные, которые уже в нем присутствуют. Файловая система ext2 позволяет расширять перечень файловых атрибутов для поддержки других функций безопасности.

7.3.2. Структурная организация файловой системы ext2

Как и в любой другой файловой системе UNIX, в составе ext2 можно выделить следующие составляющие:

- блоки и группы блоков;
- индексный дескриптор;

- суперблок.

Всё пространство раздела диска разбивается на блоки фиксированного размера, кратные размеру сектора – 1024, 2048 и 4096 байт. Размер блока указывается при создании файловой системы на разделе диска. Меньший размер блока позволяет экономить место на жестком диске, но также ограничивает максимальный размер файловой системы. Все блоки имеют *порядковые номера*. С целью уменьшения фрагментации и количества перемещений головок жесткого диска при чтении больших массивов данных блоки объединяются в группы блоков.

Базовым понятием файловой системы является индексный дескриптор (информационный узел), Information Node, или Inode. Это специальная структура, которая содержит информацию об атрибутах и физическом расположении файла.

Рис. 7.9. Обобщенная структурная схема ФС ext2

Суперблок – основной элемент файловой системы ext2. Он содержит общую информацию о файловой системе:

- общее число блоков и индексных дескрипторов в файловой системе;
- число свободных блоков и индексных дескрипторов в файловой системе;
- размер блока файловой системы;
- количество блоков и индексных дескрипторов в группе;
- размер индексного дескриптора;
- идентификатор файловой системы.

От целостности суперблока напрямую зависит работоспособность файловой системы. Операционная система создает несколько резервных копий суперблока для возможности его восстановления в случае повреждения.

Описание группы блоков, представляет собой *массив*, содержащий общую информацию обо всех блоках раздела.

Битовая карта блоков – это структура, каждый бит которой показывает, отведен ли соответствующий ему блок какому-либо файлу. Если бит

равен 1, то блок занят. Аналогичную функцию выполняет битовая карта индексных дескрипторов, показывая какие именно индексные дескрипторы заняты, а какие нет.

Все оставшееся место, обозначенное как данные и отводится для хранения файлов.

7.3.3. Система адресации данных в файловой системе ext2

Система адресации данных – это одна из самых существенных составных частей файловой системы. Именно система адресации позволяет находить нужный файл среди множества как пустых, так и занятых блоков на диске.

Файловая система ext2 использует следующую схему адресации блоков файла. Для хранения адреса файла выделено 15 полей, каждое из которых состоит из 4 байт. Если размер файла меньше или равен 12 блоков, то номера этих кластеров непосредственно перечисляются в первых двенадцати полях адреса. Если размер файла превышает 12 блоков, то следующее 13-е поле содержит адрес кластера, в котором могут быть расположены номера следующих блоков файла.

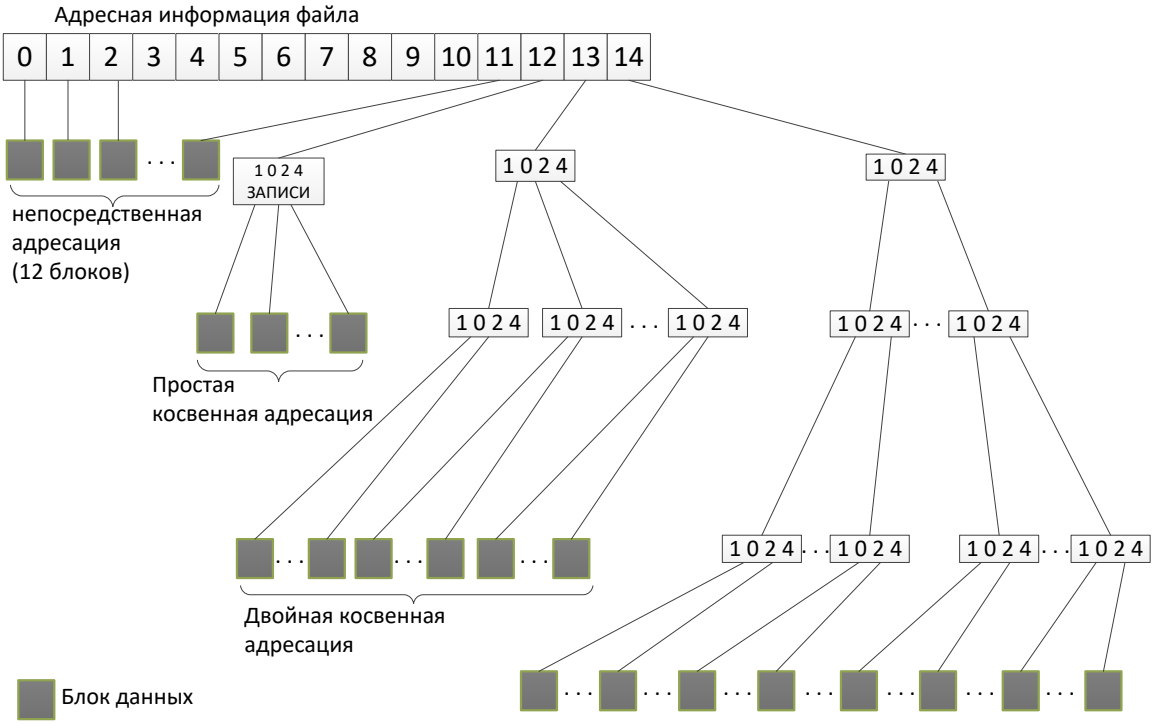


Рис. 7.10. Система адресации ФС ext2

Таким образом, 13-й элемент адреса используется для косвенной адресации. При максимальном размере блока равном 4096 байт, 13-й элемент, может содержать до 1024 номеров следующих кластеров данных файла. Если размер файла превышает 12+1024 блоков, то используется 14-е поле, в котором находится номер блока, содержащего 1024 номеров бло-

ков, каждый из которых хранят 1024 номеров блоков данных файла. Здесь применяется уже двойная косвенная адресация. И наконец, если файл включает более 12+1024+1048576 = 1049612 блоков, то используется последнее 15-е поле для тройной косвенной адресации.

Таким образом, описанная выше система адресации, позволяет при максимальном размере блока 4 Кбайт иметь файлы размера до 2 терабайт.

7.3.4. Особенности файловой системы ext3

Ext3 (или 3-я расширенная файловая система) – файловая система, основанная на ext2, включающая в себя элементы журналирования. То есть в ней предусмотрена запись специальных данных в «*журнал*», что позволяет восстановить файловую систему при сбоях в работе компьютера.

Стандартом предусмотрено три режима журналирования:

- *writeback*: в журнал записываются только метаданные файловой системы, то есть информация о её изменении. Не может гарантировать целостности данных, однако позволяет обнаружить ошибки ФС, что заметно сокращает время проверки по сравнению с ext2;
- *ordered*: то же, что и *writeback*, но запись данных в файл производится гарантированно до записи информации о изменении этого файла. Немного снижает производительность, также не может гарантировать целостности данных (хотя и увеличивает вероятность их сохранности при дописывании в конец существующего файла);
- *journal*: полное журналирование как метаданных ФС, так и пользовательских данных. Самый медленный, но и самый безопасный режим; может гарантировать целостность данных при хранении журнала на отдельном разделе (а лучше – на отдельном жёстком диске).

Файловая система ext3 может поддерживать файлы размером до 1 ТБ. С Linux-ядром 2.4 объем файловой системы ограничен максимальным размер блочного устройства, что составляет 2 терабайта. В Linux 2.6 (для 32-разрядных процессоров) максимальный размер блочных устройств составляет 16 ТБайт, однако ext3 поддерживает только до 4 ТБайт (линк).

7.3.5. Особенности файловой системы ext4

Файловая система ext4 – журналируемая файловая система, предлагаемая для использования по умолчанию в основных дистрибутивах ОС Linux и в ОС Android и отличающаяся высокой производительностью, надежностью и масштабируемостью. Блоки данных ext4 по умолчанию имеют размер 4 Кб, но также поддерживаются блоки размером 1, 2 и 64 Кб. Размер физического адреса блока составляет 48 бит, логического (для адресации блока в рамках файла) – 32 бита.

При использовании блоков данных размером 4 Кб файловая система ext4 позволяет создавать тома до одного эксбибайта (260 байт), а макси-

мально допустимый размер файла составляет 16 тебибайт (244 байт). В отличие от ext3, в ext4 число каталогов не ограничено.

В ext4 пространство под файлы выделяется не отдельными блоками, а экстентами (extent) цепочками (до 128 Мбайт) последовательно идущих блоков. Для адресации экстента используются только указатель на его начало и длина в блоках, что позволяет сократить количество хранимых метаданных. Поскольку экстенты представляют способ непрерывного размещения данных, их использование позволяет повысить производительность и снизить фрагментацию диска.

Объем индексного дескриптора ext4 составляет 60 байт, поэтому индексный узел может хранить заголовок и четыре указателя экстентов, по 12 байтов каждый. Файлы объемом до 512 Мб адресуются непосредственно из индексного узла. Для эффективного представления больших, в том числе фрагментированных, файлов строятся деревья экстентов (рис. 7.12).

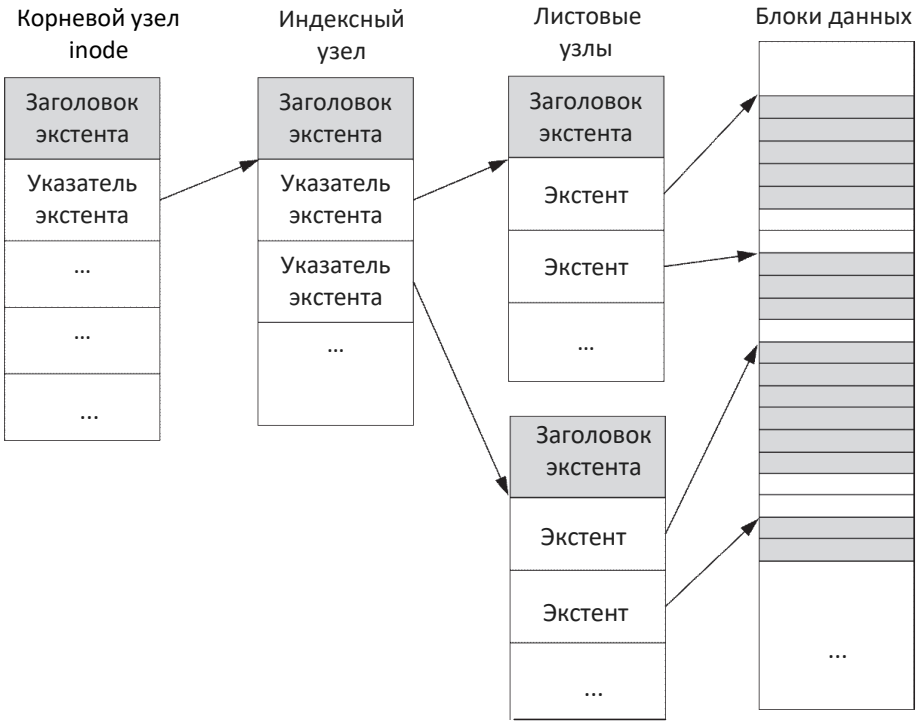


Рис. 7.12. Дерево экстентов

В дереве экстентов все узлы делятся на 3 категории:

- корневые;
- индексные;
- листовые.

Корневой индексный дескриптор может содержать ссылки на другие индексные узлы, каждый из которых может указывать на концевой узел (то есть непосредственно на экстенты).

Ext4 производит выделение блоков группами, по возможности производится попытка разместить файл в непрерывной группе блоков. Это снижает фрагментацию, требует гораздо меньшего количества системных вызовов и, как следствие, ускоряет работу ФС. Файловая система содержит

и другие средства для снижения фрагментации хранимых данных, в частности, механизм фоновой дефрагментации.

Кроме того, используется механизм отложенного распределения блоков, когда выделение дискового пространства по возможности откладывается до момента фактической записи файла из кэша на диск. Это позволяет избежать записи временных файлов, существующих лишь в течение короткого периода времени в кэш-памяти, но снижает устойчивость ФС к внезапным сбоям, например, при отключении питания.

Раздел файловой системы ext4 сходен с разделами ext2 и ext3. Он состоит из основной загрузочной записи MBR и нескольких групп блоков. Размер группы блоков ограничен 128 Мб. Первая группа блоков используется для хранения служебной информации (рис. 7.13).

Рис. 7.13. Структура раздела ext4

Суперблок хранит различную информацию о файловой системе: количество блоков, количество счетчиков inode, поддерживаемые функции, информацию об обслуживании и многое другое. По сравнению с ext3 размер суперблока увеличен с 32 до 64 бит. Каждая группа блоков содержит копию суперблока.

Дескриптор группы может рассматриваться как метаинформация для этой группы. Он содержит ссылки на начальные адреса битовой карты блоков (Block Bitmap), битовой карты индексных дескрипторов (Inode Bitmap), таблицы индексных дескрипторов (Inode Table), блоков данных и т. д. Кроме того, он содержит количество используемых блоков и индексных дескрипторов и количество каталогов в группе.

Ext4 позволяет использовать механизм «гибких групп» (Flex Group) за счет активации флага FLEX_BG. Гибкие группы служат для объединения нескольких обычных групп блоков в одну логическую группу. Битовые карты и индексная таблица располагаются в начале гибкой группы (рис. 7.14), что позволяет преодолеть жесткие ограничения на размер группы блоков – в этом случае он может достигать 8 Гб вместо 128 МБ (размер кратен 128 Мб). Кроме того, использование таких виртуальных групп позволяет сгруппировать метаинформацию на диске, уменьшить время поиска и упростить выделение экстентов под файлы.

В гибких группах добавлена специальная структура для управления свободным дисковым пространством, имеющая динамически изменяемый

размер и служащая для указания свободных непрерывных участков в рамках всей гибкой группы.

На рис. 7.14 использованы следующие сокращения для обозначения структур данных:

КС – копия суперблока;

ДГ – дескриптор группы;

БР – блоки расширения дескриптора группы;

БКБ – битовая карта блоков гибкой группы;

БКИ – битовая карта индексных дескрипторов гибкой группы;

ТИ – таблица индексных дескрипторов гибкой группы.

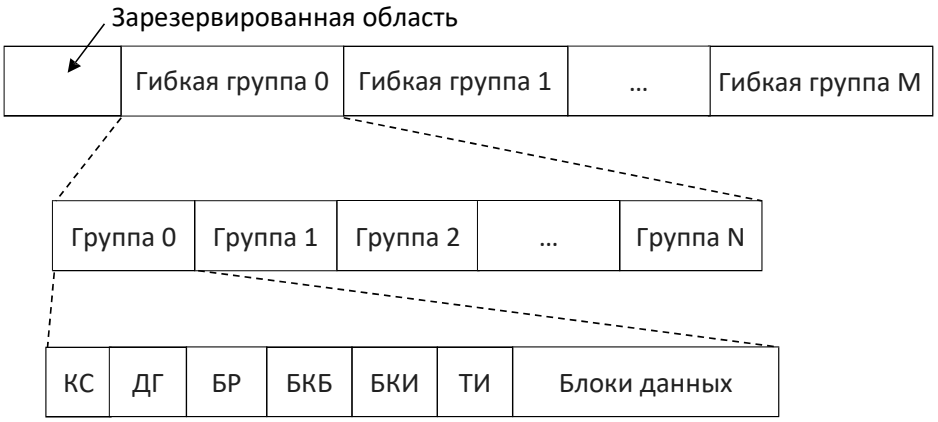


Рис. 7.14. Структура гибких групп

Платой за высокую производительность файловой системы и низкую степень фрагментации данных служит довольно большой объем дискового пространства, зарезервированного для хранения служебной информации.

Например, при создании файловой системы ext4 в Linux Ubuntu порядка 1,8% объема раздела занято служебными структурами ФС (заголовки групп блоков, бинарные карты для учета свободного места, индексные дескрипторы, основной и множество резервных копий суперблока). Кроме того, дополнительно резервируется 5% от объема раздела для нужд учетной записи суперпользователя (root) и системных служб (daemons). По другим оценкам, служебная информация может занимать до 10% от объема раздела ext4 в ОС Linux.

Журналирование позволяет обеспечить целостность файловой системы за счет протоколирования всех происходящих на диске изменений, однако при этом оно влечет незначительное снижение производительности. При необходимости журнал можно отключить. В зависимости от нужд пользователя ext4 может работать в разных режимах журналирования:

- в режиме обратной записи (в журнал заносятся только метаданные);
- в режиме упорядочивания (метаданные заносятся в журнал, но только после записи самих данных);

- в журнальном режиме (в журнал заносятся как данные, так и метаданные), последний способ обеспечивает наибольшие гарантии обеспечения целостности, но является и самым медленным.

Для проверки целостности данных журнала используются контрольные суммы. С 2012 г. контрольные суммы были добавлены ко всем основным структурам данных ext4. Кроме журнала, контрольные суммы рассчитываются для экстентов, суперблока, дескриптора группы блоков, индексных дескрипторов и добавляются к хранящейся в них метаинформации. Кроме того, ext4 поддерживает шифрование на уровне файлов и каталогов.

7.4. Виртуальная файловая система VFS

Для пользователей различных платформ в ОС Linux была встроена поддержка различных ФС с помощью промежуточного уровня *виртуальной файловой системы* VFS (Virtual File System). Виртуальная ФС позволяет скрыть от пользователя детали реализации доступа к файлам, предоставляя возможность просмотра всех файлов и папок системы из общего дерева каталогов. Все запросы, связанные с файлами, направляются на уровень VFS, предоставляющий интерфейс для доступа к данным любой доступной файловой системы. VFS определяет физическую ФС, к которой необходимо обратиться, и вызывает соответствующие драйверы этой ФС для выполнения требуемых операций.

Уровень виртуальной файловой системы определяет объекты, используемые для локализации данных и получения доступа к ним. Один из таких объектов, называемый *индексным узлом* (i-узел, index node, inode), описывает местоположение каждого файла, каталога или ссылки в рамках любой доступной ФС. Объекты inode в VFS не содержат имен представляемых ими файлов. Идентификация индексных узлов выполняется с помощью кортежа (записи), содержащего номер объекта inode (который является уникальным в пределах этой ФС) и номера, идентифицирующего саму ФС, к которой относится этот индексный узел. VFS позволяет связать несколько имен файлов с одним индексным узлом inode. Таким образом пользователи могут создавать жесткие ссылки множество имен файлов ставится в соответствие одному объекту inode файловой системы.

В виртуальной ФС каждый файл представлен с помощью файлового дескриптора, содержащего информацию об объекте inode, к которому производится обращение, и о позиции в пределах файла, к которой производится обращение.

В VFS для связи файловых дескрипторов с индексными узлами, используются элементы каталога (directory entry, dentry). Объект dentry содержит имя файла либо каталога, представленного объектом inode. Файловый дескриптор ссылается на объект dentry, который, в свою очередь, указывает на соответствующий объект inode.

Дерево каталогов Linux может состоять из одной или нескольких ФС, каждая из которых состоит из дерева объектов inode. Во время монти-

рования (mount) файловой системы, ее содержимое привязывается к определенной части основного дерева каталогов. Это позволяет процессам получать доступ к данным, хранящимся в различных ФС при помощи общего дерева каталогов.

Для ускорения доступа к файлам и папкам в виртуальной ФС существует два кэша: кэш элементов каталогов (directory entry cache, dcache) и кэш индексных узлов (inode cache). В этих кэшах хранится информация о недавно использовавшихся элементах дерева каталогов. Записи в кэше представляют объекты любой доступной смонтированной ФС.

7.5. Твердотельные накопители и их файловые системы

В настоящее время получили распространение твердотельные накопители SSD (Solid-State Drive), использующие полупроводниковые технологии флэш-памяти (flash). Эти устройства имеют следующие основные достоинства:

- высокая скорость чтения и записи, которая значительно превосходит скорость работы обычных жестких дисков (HDD) и может достигать 600 Мб/с;
- устойчивость к вибрации и ударам: механическая надежность устройства обусловлена отсутствием в нем каких-либо движущихся частей;
- низкое энергопотребление (до четырех раз меньше по сравнению с HDD);
- бесшумная работа, низкое тепловыделение;
- относительно небольшой вес и компактность.

Недостатки SSD:

- чувствительность к перепадам напряжения (утраченную в таком случае информацию восстановить невозможно);
- ограниченный рабочий ресурс (определяется конечным числом циклов перезаписи ячеек);
- высокая (относительно HDD) цена.

Существует два основных типа флэш-памяти, имеющие свои особенности физической организации, что определяет их свойства и области использования (табл. 7.2). Основу элементной базы флэш-памяти обоих типов составляют транзисторы с плавающим затвором.

Таблица 7.2 – Характеристики основных типов архитектуры флэш-памяти

Характеристики флэш-памяти	NOR	NAND
Высокоскоростной доступ	+	+
Режим постраничного доступа	–	+
Режим случайного доступа	+	–
Типичное использование	хранение программного кода	хранение информации

Архитектура NOR (Not OR – логическое Не-ИЛИ), предложена компанией Intel в 1988 г. NOR представляет собой двумерную матрицу проводников, в которой на пересечении строк и столбцов установлено по одной ячейке.

Достоинство: произвольный доступ к ячейкам памяти: возможна запись и считывание данных в определенном месте без необходимости обращаться к памяти последовательно (адресовать произвольную ячейку и обращаться к отдельному байту или слову (2 байта) данных).

Недостатки:

- процесс записи и стирания данных выполняется относительно медленно;
- большая площадь чипа.

Основное назначение этого типа памяти – хранение исполняемого программного кода, хранения файлов ОС в сотовых телефонах и планшетах, а также для хранения BIOS в компьютерах.

Архитектура NAND (Not AND – логическое Не-И) от компании Toshiba представляет трехмерную матрицу, но вместо одного транзистора на каждом пересечении строк и столбцов установлена серия из последовательно включенных ячеек.

Достоинства:

- плотность записи данных для NAND-устройств существенно превосходит возможности NOR;
- высокая скорость последовательного чтения и записи;
- возможность считывать и записывать информацию постранично (страницы размером от 0,5 до 8 Кб, как правило, 4 Кб).

Недостатки:

- не может обращаться к конкретному байту, как NOR;
- если на страницу уже была записана информация, ее невозможно перезаписать – необходимо сначала стереть целый блок страниц, после чего становится возможной повторная запись.

Развитием этой технологии стала 3D NAND, представляющая собой многослойную организацию чипов.

Компанией Intel совместно с компанией Micron разработана принципиально новая технология 3D XPoint – резистивная технология, основанная на изменении сопротивления ячеек, лежащих на пересечениях трехмерной матрицы проводников.

Достоинства 3D XPoint:

- плотность упаковки ячеек 3D XPoint выше в 8-10 раз по сравнению с NAND;
- индивидуальный доступ к каждой ячейке;
- не требуется очистка ячеек перед перезаписью;
- превосходство в скорости и долговечности (износостойкости);
- неограниченный срок хранения данных в пассивном режиме.

SSD-диск состоит из микросхем памяти и микроконтроллера, называемого FTL (Flash Translation Layer) контроллером, который выполняет задачи управления памятью и оптимизации ее использования за счет минимизации числа P/E (Program/Erase) циклов записи, стирания и повторной записи, а также балансировки нагрузки на блоки памяти.

FTL контроллер позволяет представить флэш-память на верхних уровнях как обычный диск. Это позволяет использовать для SSD накопителей традиционные файловые системы (в том числе, FAT, NTFS, ext*, которые будут рассмотрены отдельно), однако специфика физической организации флэш-памяти при этом не учитывается. Например, любое изменение в FAT32 вызывает перезапись одних и тех же секторов, в которых расположены файловые таблицы, что повышает риск отказа этих секторов и потери доступа к данным флэш-памяти. Этого недостатка лишена файловая система exFAT, которая разрабатывалась специально для USB Flash и карт памяти большого объема. Более отказоустойчивой будет работа и с файловой системой NTFS, использующей устойчивую к сбоям MFT таблицу.

Возможна внешняя балансировка нагрузки на флэш-память за счет использования специализированных файловых систем. Общая архитектура работы ОС с флэш-накопителями представлена на рис. 7.15.

Рис. 7.15. Общая архитектура файловых систем флэш-накопителей

На верхнем уровне взаимодействие приложений с SSD-накопителями осуществляется через виртуальную файловую систему VFS. Ниже

идет уровень специализированных файловых систем, таких как JFFS, JFFS2, YAFFS, F2FS и др. Далее располагается уровень флэш-преобразования (FTL), отвечающий за управление накопителем, включая выделение блоков под данные, преобразование адресов, динамическую оптимизацию износа и сборку мусора. В некоторых SSD-устройствах часть функций FTL-уровня может быть реализована аппаратно.

Сборка мусора заключается в очистке блоков, содержащих неиспользуемую информацию. Актуальные данные из такого блока переносятся в новый блок, а старый блок целиком стирается и становится доступным к повторному использованию. Обычно этот процесс проходит в фоновом режиме, но может запускаться при необходимости в случае нехватки свободного места.

Алгоритмы оптимизации износа обеспечивают динамическую оптимизацию, ограничивая число P/E циклов за счет более равномерного использования блоков, а также статическую, путем периодического переноса «старых» данных в новые блоки.

Журналируемая файловая система JFFS (Journaling Flash File System) является одной из первых разработок, учитывающих принципы организации флэш-памяти. Изначально она разрабатывалась шведской фирмой Axis Communications для собственных сетевых устройств и была ориентирована на повышение эффективности NOR памяти. При работе с флэш-памятью JFFS опирается на циклический журнал блоков. Данные, предназначенные к записи, заносятся в блоки из хвостовой части журнала, а головные блоки в это время подготавливаются для повторного использования. Свободное место находится в диапазоне между хвостом и головой журнала. Когда его становится слишком мало, производится сборка мусора, в ходе которой блоки с актуальными данными перемещаются в хвост, а освободившееся место стирается (рис. 7.16).

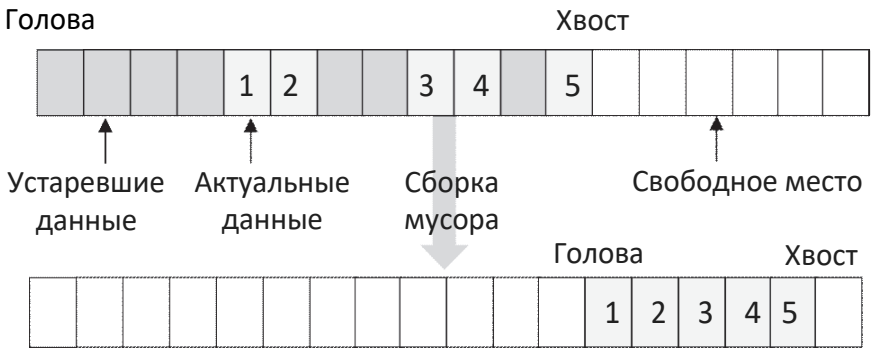


Рис. 7.16. Организация циклического журнала файловой системы JFFS

Оптимизационный алгоритм JFFS приводил к слишком частому стиранию блоков, что ускоряло износ устройства. Во второй версии системы JFFS2 (Journaling Flash File System 2), ориентированной на NAND память, было решено отказаться от циклического журнала, алгоритм сборки мусора был существенно изменен. Поддержка файловой системы JFFS2 вклю-

чена в дистрибутивы Linux, однако в настоящее время она имеет ограниченное применение, в основном в маршрутизаторах, сетевых хранилищах NAS (Network Attached Storage) и других сетевых устройствах.

Файловая система Btrfs (b-tree file system) разработана при поддержке компании Oracle специально для RAID и SSD-накопителей.

Достоинства Btrfs:

- для хранения больших файлов использует экстенты;
- поддерживается технология CoW (Copy-on-Write, копирование при записи): при изменении файла, данные не перезаписываются целиком, на диск записывается лишь модифицированная часть, а затем обновляются метаданные (снижаются требования к дисковому пространству, сокращается число операций перезаписи);
- доступно изменение размера файловой системы и разделов «на лету»;
- позволяет создавать RAID-массивы на нескольких физических или виртуальных дисках;
- автоматический поиск и исправление сбоев файловой системы;
- прозрачное сжатие данных;
- дискреционное управление доступом и другие функции.

Недостатки:

- значительная фрагментация при большом количестве перезаписей случайных фрагментов файлов;
- нельзя создавать файл подкачки.

Технология CoW позволяет снизить износ флэш накопителей, однако способствует фрагментации данных, что может быть критично для магнитных дисков HDD. Поэтому, при необходимости, функция CoW в btrfs может быть полностью отключена.

Btrfs позволяет создавать мгновенные снимки (writeable snapshots) ФС, которые позволяют произвести откат к любому предыдущему состоянию. Кроме того, btrfs ряд поддерживаемых ею функций оптимизируют хранение больших данных. Поэтому в настоящее время файловая система btrfs используется, в основном, в качестве серверной, где ее преимущества очевидны, например, на NAS-серверах.

Известен еще ряд файловых систем, разработанных специально для SSD и включенных в дистрибутивы Linux, но также получивших ограниченное распространение:

YAFFS (Yet Another Flash File System, альтернативная файловая система для флэш-накопителей), разработанная с учетом особенностей NAND памяти (вытеснена ext4).

F2FS (Flash Friendly File System) спроектирована специально для флэш-памяти с целью повышения производительности чтения/записи и защиты от износа. В настоящее время поддержка F2FS официально включена в Linux, однако на практике она показывает практически равную производительность с ext4.

7.6. Сравнительный анализ файловых систем

В настоящее время все популярные ОС поддерживают максимальное количество файловых систем. Сравнительный анализ распространенных файловых систем приведен в таблице 7.3.

Таблица 7.3 – Сравнительный анализ файловых систем

Параметры	Тип файловой системы				
	FAT 32	NTFS	ext 2	ext 3	Reiser FS
Создатель	Microsoft	Microsoft	Rémy Card	Stephen Tweedie	Namesys
Дата представления	1996	1993	1993	1999	2001
«Родная» ОС	Windows 95	Windows NT	GNU/ Linux	GNU/ Linux	GNU/ Linux
Допустимые символы в названиях	Любые символы, кроме NUL * *	Любые символы, кроме NUL, « / \ * ? < > | :	Любые символы, кроме NUL, / *	Любые символы, кроме NUL, / *	Любые символы, кроме NUL, / *
Максимальная длина пути файла	Нет ограничений	32 767 симв., каждый элемент пути – до 255 симв.	Нет ограничений	Нет ограничений	Нет ограничений
Максимальный размер файла	4 ГВ	16 ЕВ	16 ГВ – 2 ТВ	16 ГВ – 2 ТВ	8 ТВ
Максимальный размер тома	512 МВ – 8 ТВ	16 ЕВ	2 ТВ – 32 ТВ	2 ТВ – 32 ТВ	16 ТВ
Запись владельца файла	Нет	Да	Да	Да	Да
Создание временных меток	Да	Да	Нет	Нет	Нет
Временные метки доступа/чтения	Да	Да	Да	Да	Да
Контрольные суммы/ECC	Нет	Нет	Нет	Нет	Нет
Жёсткие ссылки	Нет	Да	Да	Да	Да
Мягкие ссылки	Нет	Да	Да	Да	Да
Журналирование блоков	Нет	Нет	Нет	Да	Да
Журналирование метаданных	Нет	Да	Нет	Да	Да
Чувствительно к регистру	Нет	Частично	Да	Да	Да
Лог. изменений файлов	Нет	Да	Нет	Нет	Нет
Прозрачная компрессия	Нет	Да	Нет	Нет	Да

8. Особенности построения сетевых операционных систем

Объединение компьютеров в сеть предоставляет возможность программам, работающим на отдельных компьютерах, оперативно взаимодействовать и сообща решать задачи пользователей. Связь между некоторыми программами может быть настолько тесной, что их удобно рассматривать в качестве частей одного приложения, которое называют в этом случае распределенным, или сетевым.

Распределенные приложения обладают рядом потенциальных преимуществ по сравнению с локальными. Среди этих преимуществ – более высокая производительность, отказоустойчивость, масштабируемость и приближение к пользователю.

8.1. Модели сетевых служб и распределенных приложений

Типичным является сетевое приложение, состоящее из двух частей. Одна часть приложения работает на компьютере, хранящем базу данных большого объема, а вторая – на компьютере пользователя, который хочет видеть на экране некоторые статистические характеристики данных, хранящихся в базе. Первая часть приложения выполняет поиск в базе записей, отвечающих определенным критериям, а вторая занимается статистической обработкой этих данных, представлением их в графической форме на экране, а также поддерживает диалог с пользователем, принимая от него новые запросы на вычисление тех или иных статистических характеристик. Можно представить себе случаи, когда приложение распределено и между большим числом компьютеров.

Распределенным в сетях может быть не только прикладное, но и системное программное обеспечение – компоненты операционных систем. Как и в случае локальных служб, программы, которые выполняют некоторые общие и часто встречающиеся в распределенных системах функции, обычно становятся частями операционных систем и называются сетевыми службами.

Целесообразно выделить три основных параметра организации работы приложений в сети:

1. способ разделения приложения на части, выполняющиеся на разных компьютерах сети;
2. выделение специализированных серверов в сети, на которых выполняются некоторые общие для всех приложений функции;
3. способ взаимодействия между частями приложений, работающих на разных компьютерах.

8.1.1. Способы разделения приложений на части

Существуют и типовые модели распределенных приложений. В следующей достаточно детальной модели предлагается разделить приложение на шесть функциональных частей:

1. средства представления данных на экране, например, средства графического пользовательского интерфейса;
2. логика представления данных на экране описывает правила и возможные сценарии взаимодействия пользователя с приложением: выбор из системы меню, выбор элемента из списка и т. п.;
3. прикладная логика – набор правил для принятия решений, вычислительные процедуры и операции;
4. логика данных – операции с данными, хранящимися в некоторой базе, которые нужно выполнить для реализации прикладной логики;
5. внутренние операции базы данных – действия СУБД, вызываемые в ответ на выполнение запросов логики данных, такие как поиск записи по определенным признакам;
6. файловые операции – стандартные операции над файлами и файловой системой, которые обычно являются функциями операционной системы.

На основе этой модели можно построить несколько схем распределения частей приложения между компьютерами сети.

8.1.2. Двухзвенные схемы разделения приложений

Наиболее распространенной является двухзвенная схема, распределяющая приложение между двумя компьютерами. Перечисленные выше типовые функциональные части приложения можно разделить между двумя компьютерами различными способами.

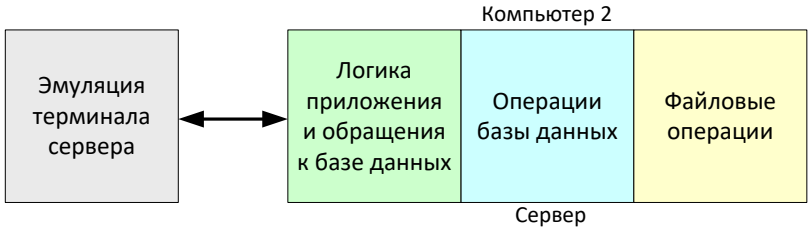
Рассмотрим типовые реализации двухзвенной схемы.

В централизованной схеме (рис. 8.1а) компьютер пользователя работает как терминал, выполняющий лишь функции представления данных, тогда как все остальные функции передаются центральному компьютеру. Ресурсы компьютера пользователя используются в этой схеме в незначительной степени, загруженными оказываются только графические средства подсистемы ввода-вывода ОС, отображающие на экране окна и другие графические примитивы по командам центрального компьютера, а также сетевые средства ОС, принимающие из сети команды центрального компьютера и возвращающие данные о нажатии клавиш и координатах мыши.

В схеме «файловый сервер» (рис. 8.1б) на клиентской машине выполняются все части приложения, кроме файловых операций. При этом в сети имеется компьютер, который играет роль файлового сервера, представляя собой централизованное хранилище данных, находящихся в разделяемом доступе. Распределенное приложение в этой схеме мало отличается от полностью локального приложения. Единственным отличием являет-

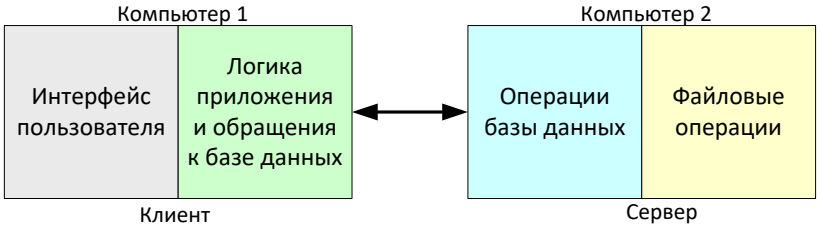
ся обращение к удаленным файлам вместо локальных. Такая схема обладает хорошей масштабируемостью, так как дополнительные пользователи и приложения добавляют лишь незначительную нагрузку на центральный узел – файловый сервер. Однако эта архитектура имеет и свои недостатки:

– во многих случаях резко возрастает сетевая нагрузка (например, многочисленные запросы к базе данных могут приводить к загрузке всей базы данных в клиентскую машину для последующего локального поиска нужных записей), что приводит к увеличению времени реакции приложения;
– компьютер клиента должен обладать высокой вычислительной мощностью, чтобы справляться с представлением данных, логикой приложения, логикой данных и поддержкой операций базы данных.



а.

б.



в.

Рис. 8.1. Варианты распределений частей приложения по двухзвенной схеме централизованная (а); «файловый сервер» (б); равномерное распределение функций (в)

Другая часто используется *схема, в которой на серверный компьютер возлагаются функции проведения внутренних операций базы данных и файловых операций* (рис. 8.1в). Клиентский компьютер при этом выполняет все функции, специфические для этого приложения, а сервер – функции,

реализация которых не зависит от специфики приложения, из-за чего эти функции могут быть оформлены в виде сетевых служб.

8.1.3. Трехзвенные схемы разделения приложений

Трехзвенная архитектура позволяет еще лучше сбалансировать нагрузку на различные компьютеры в сети, а также способствует дальнейшей специализации серверов и средств разработки распределенных приложений.

Примером трехзвенной архитектуры может служить такая организация приложения, при которой на клиентской машине выполняются средства представления и логика представления, а также поддерживается программный интерфейс для вызова частей приложения второго звена – *промежуточного сервера* (рис. 8.2).

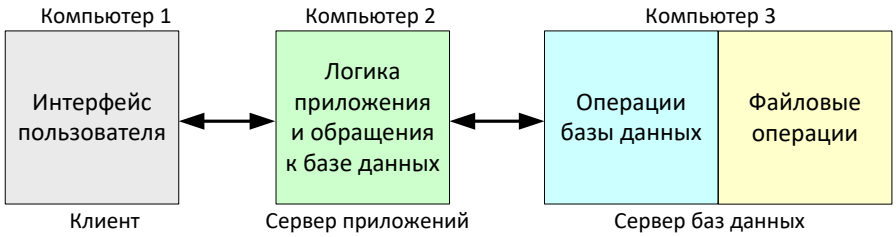


Рис. 8.2. Трехзвенная схема распределения частей приложения

Промежуточный сервер называют в этом варианте *сервером приложений,* так как на нем выполняются прикладная логика и логика обработки данных, представляющих собой наиболее специфические и важные части большинства приложений. Слой логики обработки данных вызывает внутренние операции базы данных, которые реализуются третьим звеном схемы – *сервером баз данных.*

Сервер баз данных, как и в двухзвенной модели, выполняет функции двух последних слоев – операции внутри базы данных и файловые операции.

Централизованная реализация логики приложения решает проблему недостаточной вычислительной мощности клиентских компьютеров для сложных приложений, а также упрощает администрирование и сопровождение. В том случае, когда сервер приложений сам становится узким местом, в сети можно применить несколько серверов приложений, распределив каким-то образом запросы пользователей между ними. Упрощается и разработка крупных приложений, так как в этом случае четко разделяются платформы и инструменты для реализации интерфейса и прикладной логики, что позволяет с наибольшей эффективностью реализовывать их силами специалистов узкого профиля.

Трехзвенные схемы часто применяются для централизованной реализации в сети некоторых общих для распределенных приложений функций, отличных от файлового сервиса и управления базами данных. Про-

граммные модули, выполняющие такие функции, относят к классу *middleware* – то есть промежуточному слою, располагающемуся между индивидуальной для каждого приложения логикой и сервером баз данных.

Сервер приложений должен базироваться на мощной аппаратной платформе (мультипроцессорные системы, специализированные кластерные архитектуры). ОС сервера приложений должна обеспечивать высокую производительность вычислений, а значит, поддерживать многопоточную обработку, вытесняющую многозадачность, мультипроцессирование, виртуальную память и наиболее популярные прикладные среды.

8.2. Механизмы передачи сообщений в распределенных системах

Единственным по-настоящему важным отличием распределенных систем от централизованных является способ взаимодействия между процессами.

Принципиально межпроцессное взаимодействие может осуществляться одним из двух способов:

1. с помощью совместного использования одних и тех же данных (разделяемая память);
2. путем передачи друг другу данных в виде сообщений.

В централизованных системах связь между процессами, как правило, предполагает наличие разделяемой памяти. В этом случае один процесс пишет в разделяемый буфер, а другой читает из него. Взаимодействие и в этом случае происходит за счет непосредственно доступной обоим участникам области памяти.

В распределенных системах не существует памяти, непосредственно доступной процессам, работающим на разных компьютерах, поэтому взаимодействие процессов (как находящихся в пользовательской фазе, так и в системной, то есть выполняющих код операционной системы) может осуществляться только путем передачи сообщений через сеть. В сообщениях переносятся запросы от клиентов некоторой службы к соответствующим серверам – например, запрос на просмотр содержимого определенного каталога файловой системы, расположенной на сетевом сервере. Сервер возвращает *ответ* – набор имен файлов и подкаталогов, входящих в этот каталог, также помещая его в сообщение и отправляя его по сети клиенту.

Сообщение – это блок информации, отформатированный процессом-отправителем таким образом, чтобы он был понятен процессу-получателю. Сообщение состоит из заголовка, обычно фиксированной длины, и набора данных определенного типа переменной длины.

В любой сетевой ОС имеется *подсистема передачи сообщений*, называемая также *транспортной подсистемой*, которая обеспечивает набор средств для организации взаимодействия процессов по сети. Назначение этой системы – экранировать детали сложных сетевых протоколов

от программиста. Подсистема позволяет процессам взаимодействовать посредством достаточно простых примитивов.

Транспортная подсистема сетевой ОС имеет обычно сложную структуру, отражающую структуру семиуровневой модели взаимодействия открытых систем (OSI – Open System Interconnection). Представление сложной задачи сетевого взаимодействия компьютеров в виде иерархии нескольких частных задач позволяет организовать это взаимодействие максимально гибким образом. В то же время каждый уровень модели OSI экранирует особенности лежащих под ним уровней от вышележащих уровней, что делает средства взаимодействия компьютеров все более универсальными по мере продвижения вверх по уровням. Таким образом, в процесс выполнения примитивов *send* и *receive* вовлекаются средства всех нижележащих коммуникационных протоколов (рис. 8.3).

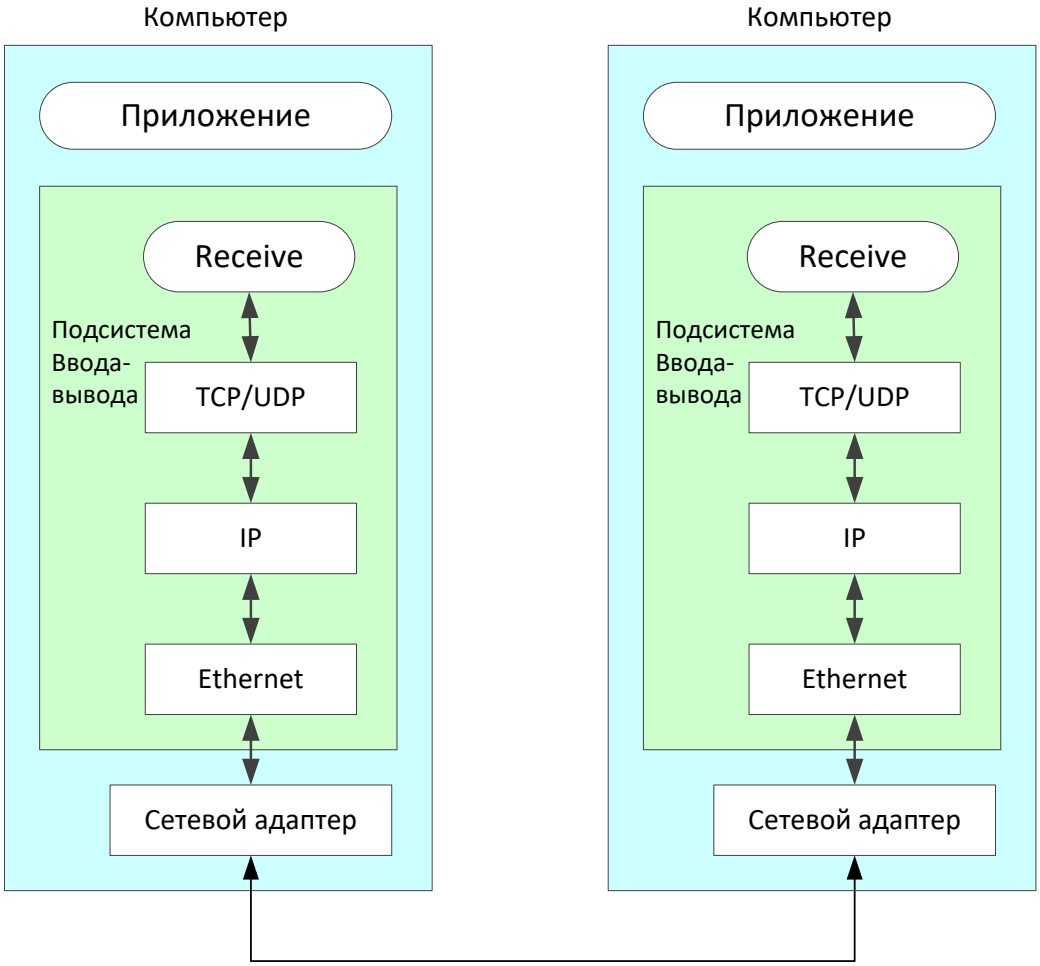


Рис. 8.3. Примитивы обмена сообщениями и транспортные средства подсистемы ввода-вывода

Несмотря на концептуальную простоту примитивов *send* и *receive*, существуют различные варианты их реализации, от правильного выбора которых зависит эффективность работы сети. В частности, эффективность зависит от способа задания адреса получателя. Не менее важны при реализации примитивов передачи сообщений ответы и на другие вопросы. В се-

ти всегда имеется один получатель или их может быть несколько? Требуется ли гарантированная доставка сообщений? Должен ли отправитель дождаться ответа на свое сообщение, прежде чем продолжать свою работу? Как отправитель, получатель и подсистема передачи сообщений должны реагировать на отказы узла или коммуникационного канала во время взаимодействия? Что нужно делать, если приемник не готов принять сообщение, нужно ли отбрасывать сообщение или сохранять его в буфере? А если сохранять, то как быть, если буфер уже заполнен? Разрешено ли приемнику изменять порядок обработки сообщений в соответствии с их важностью? Ответы на подобные вопросы составляют семантику конкретного протокола передачи сообщений.

8.3. Синхронизация в распределенных системах

Центральным вопросом взаимодействия процессов в сети является способ их синхронизации, который полностью определяется используемыми в операционной системе коммуникационными примитивами.

В этом отношении коммуникационные примитивы делятся на

- блокирующие (синхронные);
- неблокирующие (асинхронные);

причем смысл этих терминов, в целом, соответствует смыслу аналогичных терминов, применяемых при описании системных вызовов и операций ввода- В отличие от локальных системных вызовов при выполнении коммуникационных примитивов завершение запрошенной операции в общем случае зависит не только от некоторой работы локальной ОС, но и от работы удаленной ОС.

8.4. Вызов удаленных процедур

Еще одним удобным механизмом, облегчающим взаимодействие операционных систем и приложений по сети, является *механизм вызова удаленных процедур* (RPC – Remote Procedure Call). Этот механизм представляет собой надстройку над системой обмена сообщениями ОС, поэтому в ряде случаев он позволяет более удобно и прозрачно организовать взаимодействие программ по сети, однако его полезность не универсальна.

Идея вызова удаленных процедур состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений.

Наибольшая эффективность RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Характерными чертами вызова локальных процедур являются:

- *асимметричность* – одна из взаимодействующих сторон является инициатором взаимодействия;
- *синхронность* – выполнение вызывающей процедуры блокируется с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Поскольку вызывающая и вызываемая процедуры выполняются на разных машинах, то они имеют разные адресные пространства и это создает проблемы при передаче параметров и результатов, особенно если машины и их операционные системы не идентичны. Так как RPC не может рассчитывать на разделяемую память, это означает, что параметры RPC не должны содержать указателей на ячейки памяти и что значения параметров должны как-то копироваться с одного компьютера на другой.

Следующим отличием RPC от локального вызова является то, что он обязательно использует нижележащую систему обмена сообщениями, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах. Выполнение вызывающей программы и вызываемой локальной процедуры в одной машине реализуется в рамках единого процесса. Но в реализации RPC участвуют как минимум два процесса – по одному в каждой машине.

Идея, положенная в основу RPC, состоит в том, чтобы вызов удаленной процедуры по возможности выглядел так же, как и вызов локальной процедуры. Другими словами, необходимо сделать механизм RPC прозрачным для программиста: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

Механизм RPC достигает прозрачности следующим образом. Когда вызываемая процедура действительно является удаленной, в библиотеку процедур вместо локальной реализации оригинального кода процедуры помещается другая версия процедуры, называемая *клиентским стабом* (stub – заглушка). На удаленный компьютер, который выполняет роль сервера процедур, помещается оригинальный код вызываемой процедуры, а также еще один стаб, называемый *серверным стабом* (рис. 8.4).

Назначение клиентского и серверного стабов – организовать передачу параметров вызываемой процедуры и возврат значения процедуры через сеть, при этом код оригинальной процедуры, помещенной на сервер, должен быть полностью сохранен. Стабы используют для передачи данных через сеть средства подсистемы обмена сообщениями, то есть существующие в ОС примитивы *send* и *receive*. Подобно оригинальной процедуре, клиентский стаб вызывается путем обычной передачи параметров, однако затем вместо выполнения системного вызова, работающего с локальным ресурсом, происходит формирование сообщения, содержащего имя вызываемой процедуры и ее параметры

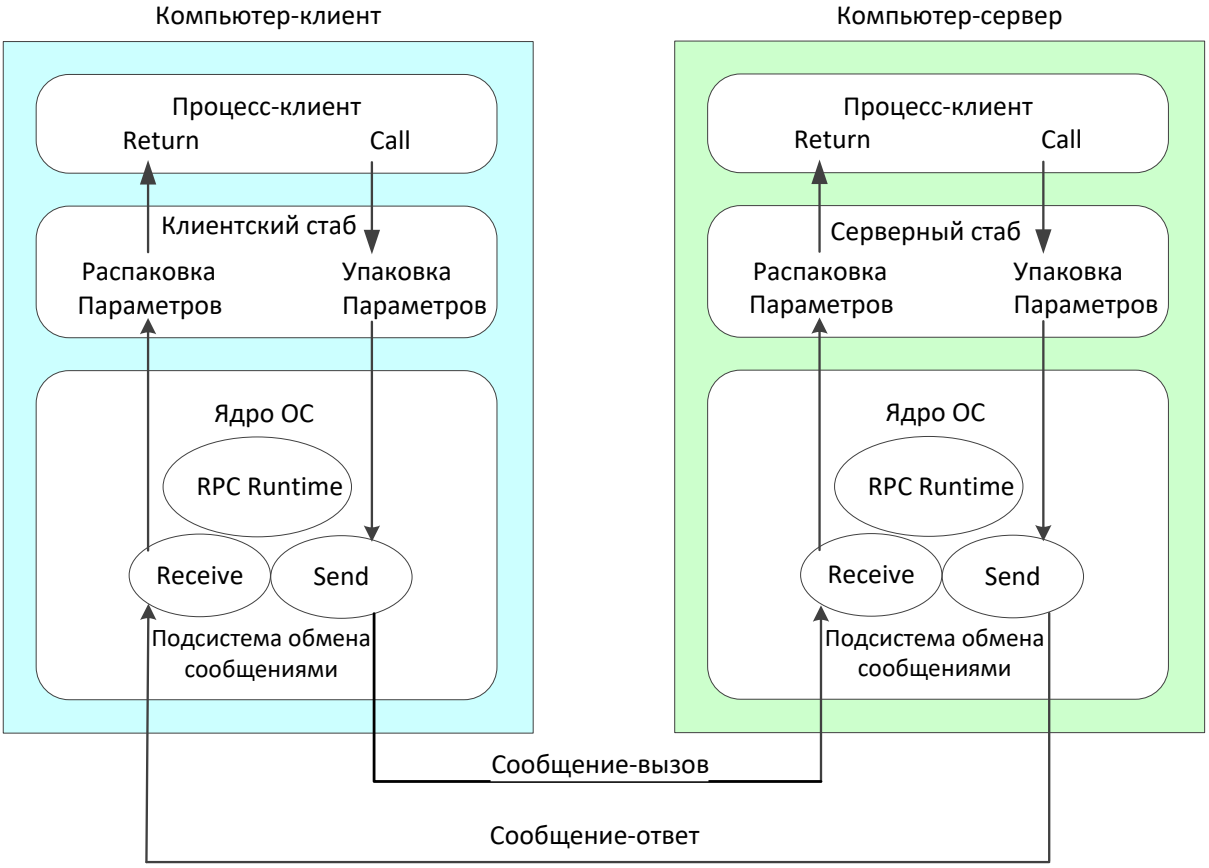


Рис. 8.4. Выполнение удаленного вызова процедуры

9. Сетевые файловые системы

9.1. Модель сетевой файловой системы

Ключевым компонентом любой распределенной системы является файловая система, которая также является в этом случае распределенной. В распределенной системе функцией файловой системы является хранение программ и данных и предоставление доступа к ним по мере необходимости. Распределенная файловая система поддерживается одним или более компьютерами, хранящими файлы. Эти компьютеры, которые позволяют пользователям сети получать доступ к своим файлам, обычно называют файловыми серверами. Файловые серверы отрабатывают запросы на чтение или запись файлов, поступающие от других компьютеров сети, которые в этом случае являются клиентами файловой службы. Каждый посланный запрос проверяется и выполняется, а ответ отсылается обратно. Файловые серверы обычно содержат иерархические файловые системы, каждая из которых имеет корневой каталог и каталоги более низких уровней. Во многих сетевых файловых системах клиентский компьютер может подсоединять и монтировать эти файловые системы к своим локальным файловым системам, обеспечивая пользователю удобный доступ к удаленным каталогам и файлам. При этом данные монтируемых файловых систем физически никуда не перемещаются, оставаясь на серверах.

Сетевая файловая система (ФС) в общем случае включает следующие элементы (рис. 9.1):

- локальная файловая система;
- интерфейс локальной файловой системы;
- сервер сетевой файловой системы;
- клиент сетевой файловой системы;
- интерфейс сетевой файловой системы;
- протокол клиент-сервер сетевой файловой системы.

Клиенты сетевой ФС – это программы, которые работают на компьютерах, подключенных к сети. Эти программы обслуживают запросы приложений на доступ к файлам, хранящимся на удаленном компьютере. В качестве таких приложений часто выступают графические или символьные оболочки ОС, такие как Windows Explorer или UNIX shell, а также любые другие пользовательские программы.

Клиент сетевой ФС передает по сети запросы другому программному компоненту – серверу сетевой ФС, работающему на удаленном компьютере. Сервер, получив запрос, может выполнить его либо самостоятельно, либо, передать запрос локальной файловой системе для отработки. После получения ответа от локальной файловой системы сервер передает его по сети клиенту, а тот, в свою очередь, – приложению, обратившемуся с запросом.

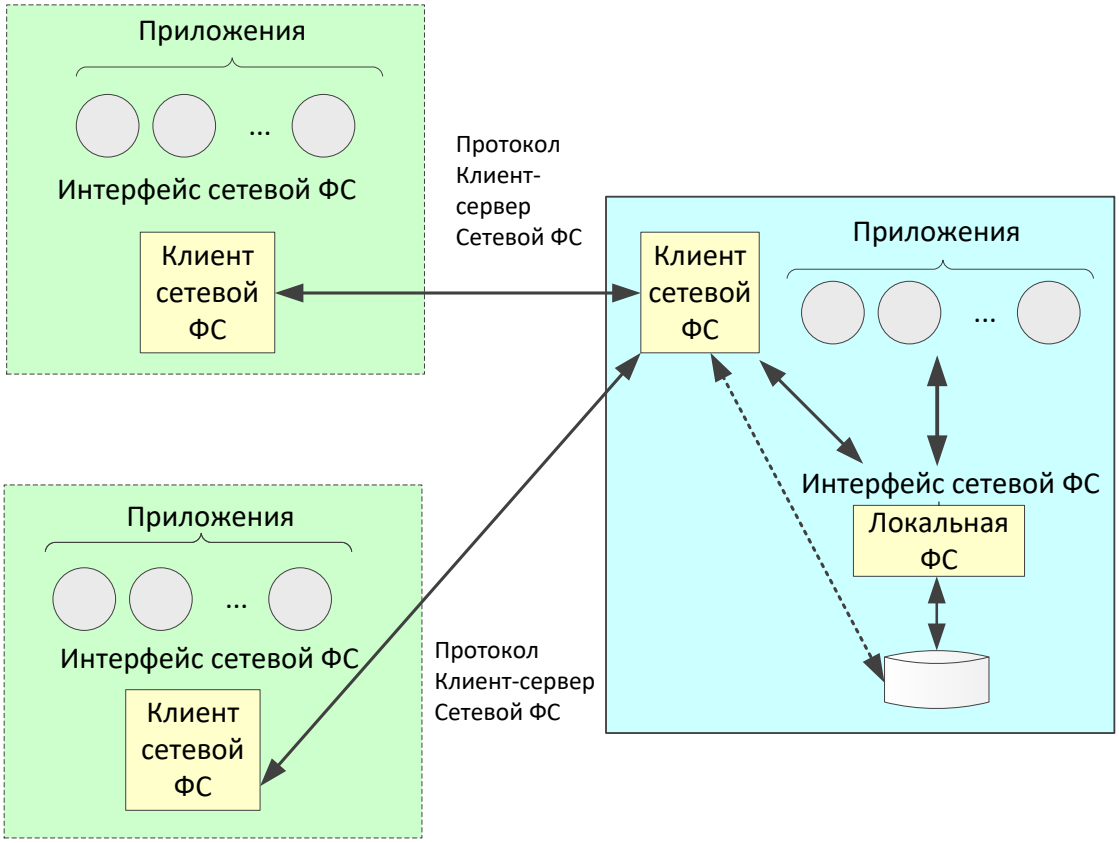


Рис. 9.1. Модель сетевой файловой системы

Приложения обращаются к клиенту сетевой ФС, используя определенный *программный интерфейс*, который в данном случае является интерфейсом сетевой файловой системы. Этот интерфейс стараются сделать как можно более похожим на интерфейс локальной файловой системы, чтобы соблюсти принцип прозрачности.

Клиент и сервер сетевой файловой системы взаимодействуют друг с другом по сети по определенному *протоколу*. В его функции будет входить ретрансляция серверу запросов, принятых клиентом от приложений, с которыми тот затем будет обращаться к локальной файловой системе. Одним из механизмов, используемых для этой ретрансляции, может быть механизм RPC.

Примером такого протокола является SMB (Server Message Block), разработанный компаниями Microsoft, Intel и IBM работает на прикладном уровне модели OSI. Для передачи по сети своих сообщений протокол SMB использует различные транспортные протоколы. Сообщения SMB могут передаваться и с помощью других протоколов, например, TCP/UDP.

Для одной и той же локальной файловой системы могут существовать различные протоколы сетевой файловой системы.

Так, к файловой системе NTFS сегодня можно получить доступ с помощью различных протоколов (рис. 9.2), в том числе таких распространенных, как SMB, NCP (NetWare Control Protocol – основной протокол доступа к файлам и принтерам сетевой ОС NetWare компании Novell) и NFS (Network File System).

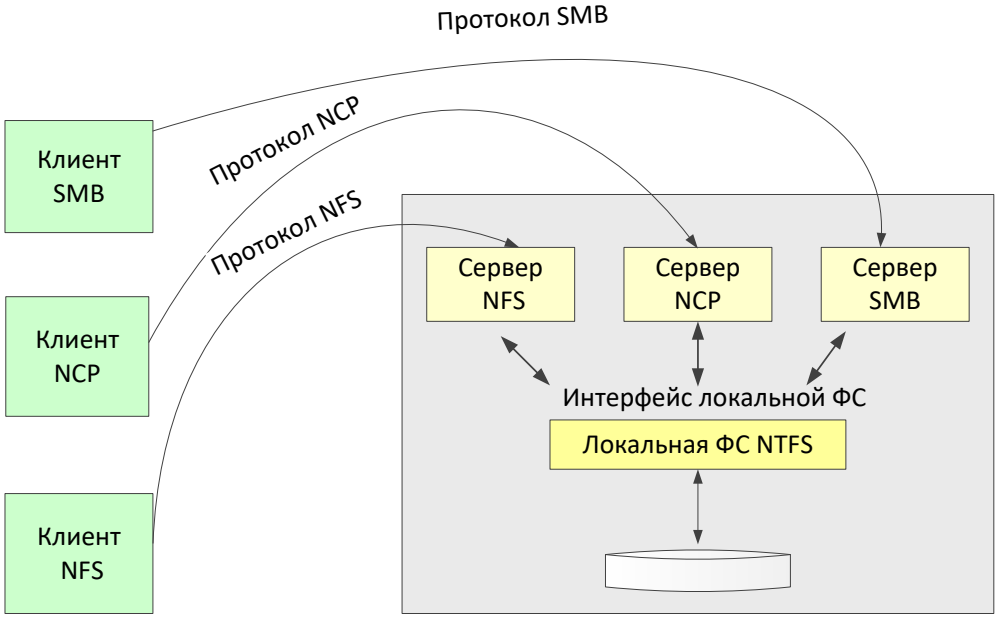


Рис. 9.2. Доступ к одной локальной файловой системе с помощью нескольких протоколов клиент-сервер

С другой стороны, с помощью одного и того же протокола может реализовываться удаленный доступ к локальным файловым системам разного типа.

Например, протокол SMB используется для доступа не только к файловой системе FAT, но и к файловым системам NTFS и HPFS (рис. 9.3). Эти файловые системы могут располагаться как на разных, так и на одном компьютере.

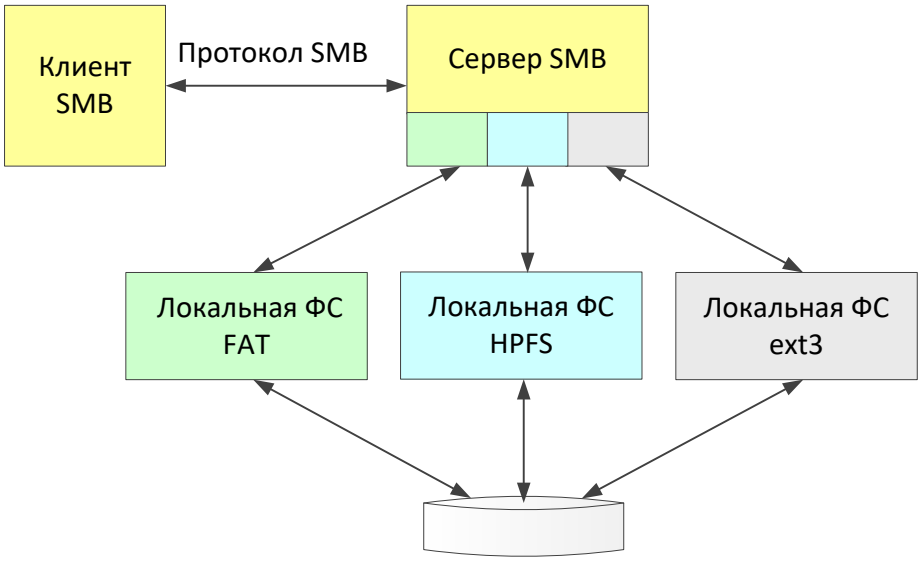


Рис. 9.3. Доступ к локальным файловым системам различного типа с помощью одного протокола клиент-сервер

За достаточно долгий срок развития сетей в них утвердилось несколько сетевых файловых систем. В среде операционной системы UNIX

наибольшее распространение получили две сетевые файловые системы – FTP (File Transfer Protocol) и NFS (Network File System). Они первоначально разрабатывались для доступа к локальной файловой системе ОС семейства UNIX. Со временем в крупных сетях стали одновременно применяться несколько сетевых файловых систем разных типов, например, NetWare и SMB или NetWare и NFS. Это часто происходило при объединении нескольких сетей в одну.

На рис. 9.4 показан вариант организации неоднородной сетевой файловой системы, в которой на компьютере с локальной файловой системой NTFS работает несколько файловых серверов, поддерживающих различные протоколы клиент-сервер.

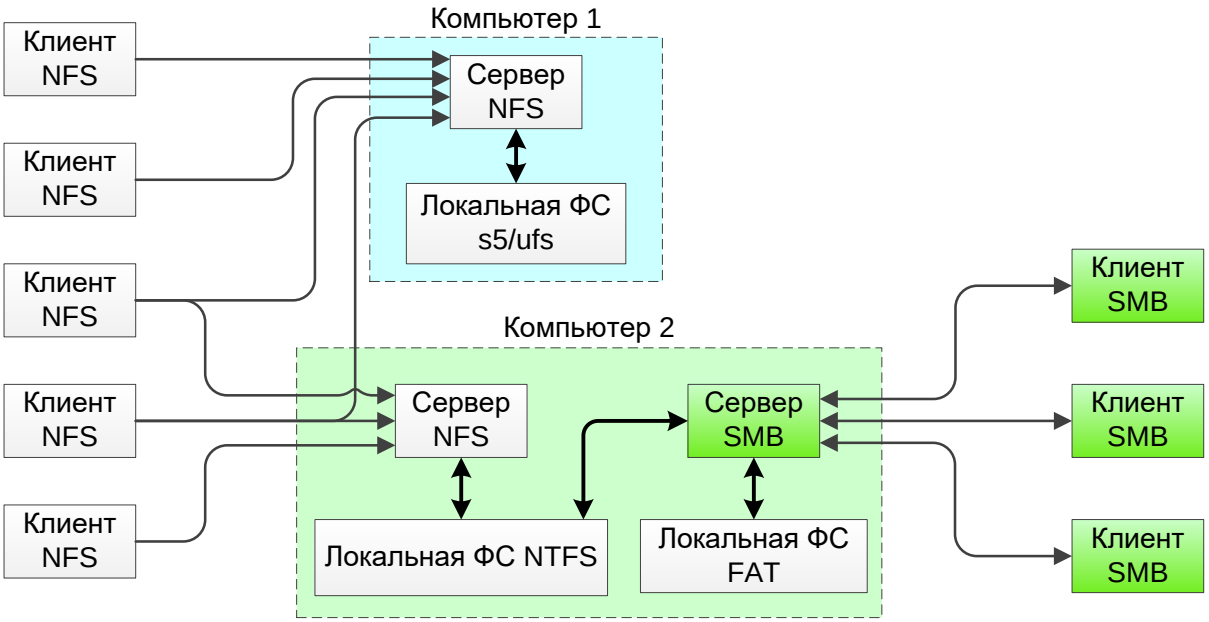


Рис. 9.4. Неоднородная сетевая файловая система

9.2. Интерфейс сетевой файловой службы

Во многих системах, таких как UNIX и Windows, файл – это не интерпретируемая последовательность байтов. Значение и структура информации в файле является заботой прикладных программ, а не ОС.

Модифицируемость файлов – возможность модификации файла после его создания. В большинстве сетевых файловых систем файлы могут модифицироваться, но в некоторых распределенных системах единственными операциями с файлами являются *create* (*создать*) и *read* (*прочитать*). Такие файлы называются неизменяемыми. Для неизменяемых файлов намного легче осуществить кэширование файла и его репликацию (тиражирование), так как исключаются все проблемы, связанные с обновлением всех копий файла при его изменении.

Семантика разделения файлов – порядок чтения и записи в случае, когда два или более пользователей разделяют один файл, чтобы избежать проблем с интерпретацией результирующих данных файла.

Различают следующие семантики разделения файлов.

- *Семантика UNIX*. В операционных системах UNIX, обычно определяется, что когда операция чтения следует за операцией записи, то читается только что обновленный файл. Аналогично, когда операция чтения следует за двумя операциями записи, то читается файл, измененный последней операцией записи. Тем самым система придерживается абсолютного временного упорядочивания всех операций и всегда возвращает самое последнее значение данных.
- *Сеансовая семантика*. В соответствии с этой моделью изменения в открытом файле сначала видны только процессу, который модифицирует файл, и только после закрытия файла эти изменения могут видеть другие процессы. При использовании сеансовой семантики возникает проблема одновременного использования одного и того же файла двумя или более клиентами.
- *Семантика неизменяемых файлов*. Заключается в том, чтобы сделать все файлы неизменяемыми. Тогда файл нельзя открыть для записи, а можно выполнять только операции *create* (*создать*) и *read* (*читать*).
- *Транзакционная семантика*, т. е. использование механизма неделимых транзакций.

Для контроля доступа с каждым разделяемым файлом обычно связан список управления доступом (ACL – Access Control List), обеспечивающий защиту данных от несанкционированного доступа. В том случае, когда локальная файловая система поддерживает механизм ACL для файлов и каталогов при локальном доступе, сетевая файловая система использует этот механизм и при доступе по сети. Если же механизм защиты в локальной файловой системе отсутствует, то сетевой файловой системе приходится поддерживать его самостоятельно.

Файловый интерфейс может быть отнесен к одному из двух типов в зависимости от того, поддерживает ли он модель загрузки-выгрузки или модель удаленного доступа.

В модели загрузки-выгрузки пользователю предлагаются средства чтения или записи файла целиком. Эта модель предполагает следующую схему обработки файла:

- чтение файла с сервера на машину клиента;
- обработка файла на машине клиента;
- запись обновленного файла на сервер.

Модель удаленного доступа, которая предполагает поддержку большого количества операций над файлами:

- открытие и закрытие файлов;
- чтение и запись частей файла;
- позиционирование в файле;
- проверка и изменение атрибутов файла и т.д.

В то время как в модели загрузки-выгрузки файловый сервер обеспечивал только хранение и перемещение файлов, в случае модели удаленного доступа все файловые операции выполняются на серверах, а клиенты только генерируют запросы на их отработку.

9.3. Размещение клиентов и серверов по компьютерам и в операционной системе

При организации в сети взаимодействия между ОС по концепции клиент-сервер, имеются различные способы распределения серверной и клиентской частей между компьютерами.

1. В некоторых файловых системах (например, в NFS или в файловых системах Windows) на всех компьютерах сети работает одно и то же базовое программное обеспечение, включающее как клиентскую, так и серверную части, так что любой компьютер, который захочет предложить услуги файловой службы, может это сделать.
2. В некоторых случаях выпускается так называемая серверная версия ОС (например, Microsoft Windows Server), которая использует то же программное обеспечение файловой службы, но только позволяющее (за счет выделения файловому серверу большего количества ресурсов (в основном оперативной памяти) обслуживать одновременно большее число пользователей, чем версии файлового сервера для клиентских компьютеров.
3. В других системах файловый сервер – это специализированный компонент серверной ОС, отсутствующий в клиентских компьютерах.

Для повышения эффективности работы файловый сервер и клиент обычно являются модулями ядра ОС, работающими в привилегированном режиме. В современных ОС эти компоненты оформляются как высокоуровневые драйверы, работающие в составе подсистемы ввода-вывода.

9.4. Кэширование данных

Кэширование данных в оперативной памяти, может существенно повысить скорость доступа к файлам, хранящимся на дисках. Кэширование широко используется в сетевых файловых системах, где оно позволяет не только повысить скорость доступа к удаленным данным (это по-прежнему является основной целью кэширования), но и улучшить масштабируемость и надежность файловой системы.

Схемы кэширования, применяемые в сетевых файловых системах, отличаются решениями по трем ключевым аспектам:

- месту расположения кэша;
- способу распространения модификаций;
- проверке достоверности кэша.

Кроме того, на схему кэширования влияет выбранная в файловой системе модель переноса файлов между сервером и клиентами: модель загрузки-выгрузки, переносящая файл целиком, или модель удаленного доступа, позволяющая переносить файл по частям. Соответственно в первом случае файл кэшируется целиком, а во втором кэшируются только те части файла, к которым выполняется обращение.

Место расположения кэша. В системах, состоящих из клиентов и серверов, имеются три различных места для хранения кэшируемых файлов и их частей:

- память сервера;
- диск клиента (если имеется);
- память клиента.

Способы распространения модификаций. Существование в одно и то же время в сети нескольких копий одного и того же файла, хранящихся в кэшах клиентов, порождает проблему согласования копий.

Для решения этой проблемы необходимо, чтобы модификации данных, выполненные над одной из копий, были своевременно распространены на все остальные копии. Существует несколько вариантов распространения модификаций:

- использование *алгоритма сквозной записи*. Когда кэшируемый элемент (файл или блок) модифицируется, новое значение записывается в кэш и одновременно посылается на сервер для обновления главной копии файла. Этот вариант распространения модификаций обеспечивает семантику разделения файлов в стиле UNIX.
- принятие *сеансовой семантики*, в соответствии с которой запись файла на сервер производится только после закрытия файла. Этот алгоритм называется «запись по закрытию» и приводит к тому, что если две копии одного файла кэшируются на разных машинах и последовательно записываются на сервер, то второй записывается поверх первого.

Проверка достоверности кэша. Распространение модификаций решает только проблему согласования главной копии файла, хранящейся на сервере, с клиентскими копиями. В то же время этот прием не дает никакой информации о том, когда должны обновляться данные, находящиеся в кэшах клиентов. Очевидно, что данные в кэше одного клиента становятся недостоверными, когда данные, модифицированные другим клиентом, переносятся в главную копию файла. Следовательно, необходимо проверять, являются ли данные в кэше клиента достоверными. В противном случае данные кэша должны быть повторно считаны с сервера. Существуют два подхода к решению этой проблемы.

1. *Инициирование проверки клиентом,* в этом случае клиент связывается с сервером и проверяет, соответствуют ли данные в его кэше данным главной копии файла на сервере.

Клиент может выполнять такую проверку одним из трех способов:

- *Перед каждым доступом к файлу*. Этот способ дискредитирует саму идею кэширования, так как каждое обращение к файлу вызывает обмен по сети с сервером. Но зато это обеспечивает семантику разделения файлов UNIX.
- *Периодические проверки*. Улучшают производительность, но делают семантику разделения неясной, зависящей от временных соотношений.
- *Проверка при открытии файла*. Этот способ подходит для сеансовой семантики. Необходимо отметить, что сеансовая семантика требует одновременного применения модели доступа загрузки-выгрузки, метода распространения модификаций «запись по закрытию» и проверки достоверности кэша при открытии файла.

2. *Инициирование проверки сервером.* Когда файл открывается, то клиент, выполняющий это действие, посылает соответствующее сообщение файловому серверу, в котором указывает режим открытия файла – чтение или запись. Файловый сервер сохраняет данную информацию и использует ее при доступе к файлу других клиентов.

   Если файл открыт для чтения, то нет никаких препятствий для разрешения другим процессам открыть его для чтения, но открытие его для записи должно быть запрещено. Аналогично, если некоторый процесс открыл файл для записи, то все другие виды доступа должны быть запрещены. При закрытии файла также необходимо оповестить файловый сервер для того, чтобы он обновил свои таблицы, содержащие данные об открытых файлах. Модифицированный файл также может быть выгружен на сервер в такой момент.

9.5. Репликация файлов

Сетевая файловая система может поддерживать репликацию (тиражирование) файлов в качестве одной из услуг, предоставляемых клиентам. Иногда репликацией занимается отдельная служба ОС.

Репликация (replication) подразумевает существование нескольких копий одного и того же файла, каждая из которых хранится на отдельном файловом сервере, при этом обеспечивается автоматическое согласование данных в копиях файла.

Имеется две главные причины для применения репликации.

1. *Увеличение надежности* за счет наличия независимых копий каждого файла, хранящихся на разных файловых серверах. При отказе одного из них файл остается доступным.

2. *Распределение нагрузки между несколькими серверами.* Клиенты могут обращаться к данным реплицированного файла на ближайший файловый сервер, хранящий копию этого файла. Ближайшим может считаться сервер, находящийся в той же подсети, что и клиент, или же первый откликнувшийся, или же выбранный некоторым сетевым устройством, балансирующим нагрузки серверов.

Репликация похожа на кэширование файлов на стороне клиентов тем, что в системе создается несколько копий одного файла. Однако существуют и принципиальные отличия – если кэширование предназначено для обеспечения локального доступа к файлу одному клиенту и повышения за счет этого скорости работы этого клиента, то репликация нужна для повышения надежности хранения файлов и снижения нагрузки на файловые серверы.

9.6. Примеры сетевых файловых служб: FTP и NFS

9.6.1. Протокол передачи файлов FTP

Сетевая файловая служба на основе протокола FTP (File Transfer Protocol) представляет собой одну из наиболее ранних служб, используемых для доступа к удаленным файлам. До появления службы WWW это была самая популярная служба доступа к удаленным данным в Интернете и корпоративных IP-сетях. Первые спецификации FTP относятся к 1971 г. Серверы и клиенты FTP имеются практически в каждой ОС семейства UNIX, а также во многих других сетевых ОС. Клиенты FTP встроены сегодня в программы просмотра (браузеры) Интернета, так как архивы файлов на основе протокола FTP по-прежнему популярны и для доступа к таким архивам браузером используется протокол FTP.

Протокол FTP (File Transfer Protocol) позволяет целиком переместить файл с удаленного компьютера на локальный и наоборот, то есть работает по схеме загрузки-выгрузки. Кроме того, он поддерживает несколько команд просмотра удаленного каталога и перемещения по каталогам удаленной файловой системы. Поэтому FTP особенно удобно использовать для доступа к тем файлам, данные которых нет смысла просматривать удаленно, а гораздо эффективней целиком переместить на клиентский компьютер (например, файлы исполняемых модулей приложений).

В протокол FTP встроены примитивные средства аутентификации удаленных пользователей на основе передачи по сети пароля в открытом виде. Кроме того, поддерживается анонимный доступ, не требующий указания имени пользователя и пароля, который является более безопасным, так как не подвергает пароли пользователей угрозе перехвата.

Протокол FTP выполнен по схеме клиент-сервер.

Клиент FTP состоит из нескольких функциональных модулей:

- *User Interface* – пользовательский интерфейс, принимающий от пользователя символьные команды и отображающий состояние сеанса FTP на символьном экране;
- *User-Pi* – интерпретатор команд пользователя. Этот модуль взаимодействует с соответствующим модулем сервера FTP;
- *User-DTP* – модуль, осуществляющий передачу данных файла по командам, получаемым от модуля *User-Pi* по протоколу клиент-сервер. Этот модуль взаимодействует с локальной файловой системой клиента.

FTP-сервер включает следующие модули:

- *Server-Pi* – модуль, который принимает и интерпретирует команды, передаваемые по сети модулем *User-PL*;
- *Server-DTP* – модуль, управляющий передачей данных файла по командам от модуля *Server-PL* взаимодействует с локальной файловой системой сервера.

Клиент и сервер FTP поддерживают параллельно два сеанса – управляющий сеанс и сеанс передачи данных. Управляющий сеанс открывается при установлении первоначального FTP-соединения клиента с сервером, причем в течение одного управляющего сеанса может последовательно выполняться несколько сеансов передачи данных, в рамках которых передаются или принимаются несколько файлов.

Общая схема взаимодействия клиента и сервера по этапам приведена ниже.

1. Сервер FTP всегда открывает управляющий порт TCP 21 для прослушивания, ожидая приход запроса на установление управляющего сеанса FTP от удаленного клиента.

2. После установления управляющего соединения клиент отправляет на сервер команды, которые уточняют параметры соединения:

- имя и пароль клиента;
- роль участников соединения (активная или пассивная);
- порт передачи данных;
- тип передачи;
- тип передаваемых данных (двоичные данные или ASCII-код);
- директивы на выполнение действий (читать файл, писать файл, удалить файл и т.п.).

3. После согласования параметров пассивный участник соединения переходит в режим ожидания открытия соединения на порт передачи данных. Активный участник инициирует это соединение и начинает передачу данных.

4. После окончания передачи данных соединение по портам данных закрывается, а управляющее соединение остается открытым. Пользователь может по управляющему соединению активизировать новый сеанс передачи данных.

Порты передачи данных выбирает клиент FTP (по умолчанию клиент может использовать для передачи данных порт управляющего сеанса), а сервер должен использовать порт, на единицу меньший порта клиента.

9.6.2. Файловая система NFS

Файловая система NFS (Network File System) создана компанией Sun Microsystems. В настоящее время это стандартная сетевая файловая система для ОС семейства UNIX, кроме того, клиенты и серверы NFS реализованы для многих других ОС. Принципы ее организации на сегодня стандартизованы сообществом Интернета, версия NFS v. 4 описывается спецификацией, выпущенной в 2000 г.

Файловая система NFS (Network File System) представляет собой систему, поддерживающую схему удаленного доступа к файлам. Работа пользователя с удаленными файлами производится после выполнения операции монтирования становится полностью прозрачной – поддерево файловой системы сервера NFS становится поддеревом локальной файловой системы.

Одной из целей разработчиков NFS была поддержка неоднородных систем с клиентами и серверами, работающими под управлением различных ОС на различной аппаратной платформе.

Для обеспечения устойчивости клиентов к отказам серверов в NFS принят подход *stateless*, то есть серверы при работе с файлами не хранят данных об открытых клиентами файлах.

Основная идея NFS – позволить произвольной группе пользователей разделять общую файловую систему.

В основе файловой системы NFS лежит представление о том, что пользоваться общей файловой системой может произвольный набор клиентов и серверов. Каждый сервер NFS экспортирует один или несколько своих каталогов, предоставляя доступ к ним удаленным клиентам. Как правило, доступ к каталогу предоставляется вместе со всеми его подкаталогами, поэтому все дерево каталогов экспортируется как единое целое. Список экспортируемых сервером каталогов хранится в файле, чтобы эти каталоги экспортировались автоматически при загрузке сервера. Клиенты получают доступ к экспортируемым каталогам, монтируя эти каталоги. Если клиент монтирует удаленный каталог, то этот каталог становится частью иерархии каталогов клиента. Выполнение программ почти не зависит от того, где расположен файл: локально или на удаленном диске. Если два или более клиента одновременно смонтировали один и тот же каталог, то они могут связываться путем разделения файла.

Архитектура NFS имеет три уровня:

- UNIX-интерфейс файловой системы (основан на вызовах open, read, write и close calls и на дескрипторах файлов);
- уровень Virtual File System (VFS) – различает локальные и удаленные файлы, и в дальнейшем локальные файлы обрабатываются

в соответствии с типами их ФС. VFS активизирует операции, специфичные для конкретной файловой системы, для обработки локальных запросов. Уровень VFS вызывает процедуры NFS – протокола для удаленных запросов;

- сервисный уровень – реализует NFS-протокол.

В своей работе файловая система NFS использует два протокола.

1. NFS-протокол управляет монтированием. Клиент посылает серверу полное имя каталога и запрашивает разрешение на монтирование этого каталога в какую-либо точку собственного дерева каталогов. Получив имя, сервер проверяет законность этого запроса и возвращает клиенту дескриптор файла, являющегося удаленной точкой монтирования.

2. NFS-протокол используется для доступа к удаленным файлам и каталогам. Клиенты могут послать запрос серверу для выполнения какого-либо действия над каталогом или операции чтения или записи файла. Кроме того, они могут запросить атрибуты файла, такие как тип, размер, время создания и модификации.

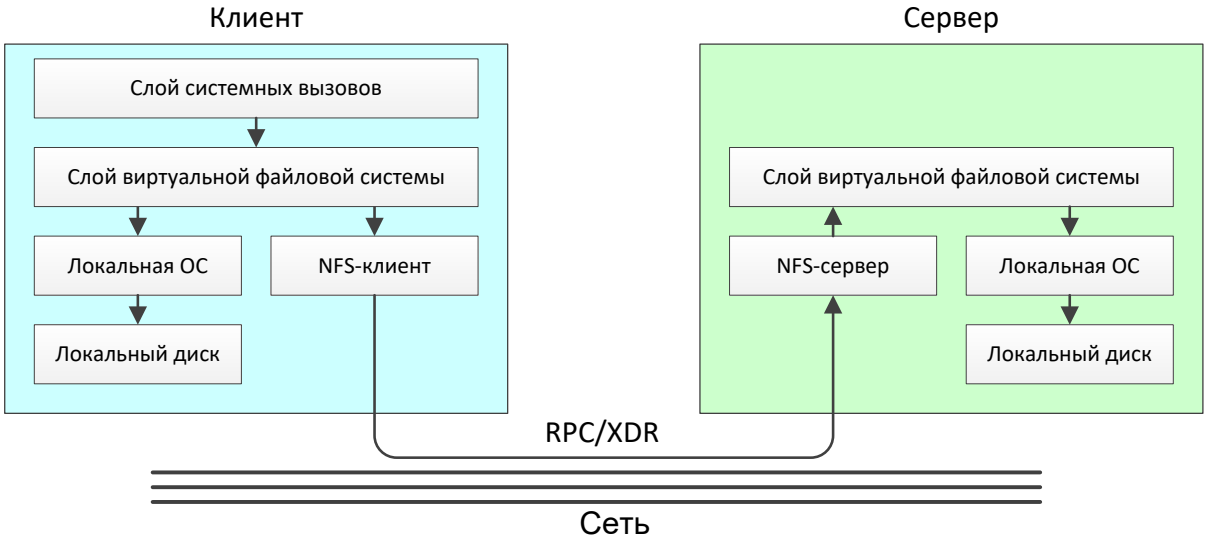


Рис. 9.5. Реализация NFS

В NFS используется кэширование на стороне клиента, данные в кэш переносятся поблочно и применяется упреждающее чтение, при котором чтение блока в кэш по требованию приложения всегда сопровождается чтением следующего блока по инициативе системы. Клиент при очередном открытии файла, имеющегося в его кэше, проверяет у сервера, когда файл был в последний раз модифицирован. Если это произошло после того, как файл был помещен в кэш, файл удаляется из кэша и от сервера получается новая копия файла. Клиенты распространяют модификации, сделанные в кэше, с периодом в 30 с, так что сервер может получить обновления с большой задержкой. В результате работы механизмов удаления данных из кэша и распространения модификаций данные, получаемые каким-либо клиентом, не всегда, являются самыми свежими. Репликация в NFS не поддерживается.

Файловая система NFS использует стандартный механизм защиты UNIX с битами rwx для владельца, группы и прочих пользователей. Изначально каждое сообщение с запросом просто содержало идентификаторы пользователя и группы вызывающей стороны, которые сервер NFS использовал для проверки прав доступа. Сервер NFS предполагал, что клиенты не будут его обманывать, однако это было ошибочно. Сейчас для аутентификации клиента и сервера при каждом запросе и ответе можно использовать шифрование с открытым ключом. При этом злоумышленник не сможет выдать себя за другого клиента, так как ему неизвестен секретный ключ этого клиента.

9.7. Служба каталогов

Подобно большой организации, большая компьютерная сеть нуждается в централизованном хранении как можно более полной справочной информации о самой себе. Решение многих задач в сети опирается на информацию о пользователях сети – их именах, используемых для логического входа в систему, паролях, правах доступа к ресурсам сети, а также о ресурсах и компонентах сети: серверах, клиентских компьютерах, маршрутизаторах, шлюзах, томах файловых систем, принтерах и т. п.

Примеры наиболее важных задач централизованной базы справочной информации сети приведены ниже.

- *Аутентификация пользователей*, на основе которой затем выполняется авторизация доступа. В сети должны каким-то образом централизованно храниться учетные записи пользователей, содержащие имена и пароли.
- *Поддержка прозрачности доступа к сетевым ресурсам.* В такой базе должны храниться имена этих ресурсов и отображения имен на числовые идентификаторы (например, IP-адреса), позволяющие найти этот ресурс в сети. Прозрачность может обеспечиваться при доступе к серверам, томам файловой системы, интерфейсам процедур RPC, программным объектам распределенных приложений и многим другим сетевым ресурсам.
- *Электронная почта* с единой для сети справочной службой, хранящей данные о почтовых именах пользователей.
- *Средства управления качеством обслуживания трафика* (QoS – Quality of Service), которые также требуют наличия сведений обо всех пользователях и приложениях системы, их требованиях к параметрам качества обслуживания трафика, а также обо всех сетевых устройствах, с помощью которых можно управлять трафиком (маршрутизаторах, коммутаторах, шлюзах и т. п.).
- *Организация распределенных приложений* может существенно упроститься, если в сети имеется база, хранящая информацию об имеющихся программных модулях-объектах и их расположении на серверах сети. Приложение, которому необходимо выполнить

некоторое стандартное действие, обращается с запросом к такой базе и получает адрес программного объекта, имеющего возможность выполнить требуемое действие.

- *Система управления сетью* должна располагать базой для хранения информации о топологии сети и характеристиках всех сетевых элементов, таких как маршрутизаторы, коммутаторы, серверы и клиентские компьютеры. Наличие полной информации о составе сети и ее связях позволяет системе автоматизированного управления сетью правильно идентифицировать сообщения об аварийных событиях и находить их первопричину.

Результатом развития систем хранения справочной информации стало появление в сетевых операционных системах специальной службы – так называемой *службы каталогов* (*Directory Services*), называемой также справочной *службой* (*directory* – справочник, каталог).

Служба каталогов (DS – Directory Services) хранит информацию обо всех пользователях и ресурсах сети в виде унифицированных объектов с определенными атрибутами, а также позволяет отражать взаимосвязи между хранимыми объектами, такие как принадлежность пользователей к определенной группе, права доступа пользователей к компьютерам, вхождение нескольких узлов в одну подсеть, коммуникационные связи между подсетями, производственную принадлежность серверов и т. д.

Служба каталогов позволяет выполнять над хранимыми объектами набор некоторых базовых операций, таких как добавление и удаление объекта, включение объекта в другой объект, изменение значений атрибута объекта, чтение атрибутов и некоторые другие. Обычно над службой каталогов строятся различные специфические сетевые приложения, которые используют информацию службы для решения конкретных задач: управления сетью, аутентификации пользователей, обеспечения прозрачности служб и других, перечисленных выше.

Служба каталогов обычно строится на основе *модели клиент-сервер*: серверы хранят базу справочной информации, которой пользуются клиенты, передавая серверам по сети соответствующие запросы. Для клиента службы каталогов она представляется единой централизованной системой, хотя большинство хороших служб каталогов имеют распределенную структуру, включающую большое количество серверов, но эта структура для клиентов прозрачна.

Проблемы сохранения производительности и надежности при увеличении масштаба сети обычно решаются за счет распределенных баз данных справочной информации. Разделение данных между несколькими серверами снижает нагрузку на каждый сервер, а надежность при этом достигается за счет наличия нескольких реплик каждой части базы данных.

10. Современные концепции и технологии проектирования операционных систем

10.1. Требования, предъявляемые к современной операционной системе

Операционная система является сердцевиной сетевого программного обеспечения, она создает среду для выполнения приложений и во многом определяет, какими полезными для пользователя свойствами эти приложения будут обладать. В связи с этим рассмотрим требования, которым должна удовлетворять современная ОС.

Очевидно, что главным требованием, предъявляемым к операционной системе, является способность выполнения основных функций: эффективного управления ресурсами и обеспечения удобного интерфейса для пользователя и прикладных программ.

Современная ОС, как правило, должна реализовывать:

- мультипрограммную обработку;
- виртуальную память, свопинг;
- поддерживать многооконный интерфейс;
- выполнять другие необходимые функции.

Кроме этих функциональных требований к операционным системам предъявляются не менее важные требованиям.

- Расширяемость. Код должен быть написан таким образом, чтобы можно было легко внести дополнения и изменения, если это потребуется, и не нарушить целостность системы.
- Переносимость. Код должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которая включает наряду с типом процессора и способ организации всей аппаратуры компьютера) одного типа на аппаратную платформу другого типа.
- Надежность и отказоустойчивость. Система должна быть защищена как от внутренних, так и от внешних ошибок, сбоев и отказов. Ее действия должны быть всегда предсказуемыми, а приложения не должны быть в состоянии наносить вред ОС.
- Совместимость. ОС должна иметь средства для выполнения прикладных программ, написанных для других операционных систем. Кроме того, пользовательский интерфейс должен быть совместим с существующими системами и стандартами.
- Безопасность. ОС должна обладать средствами защиты ресурсов одних пользователей от других.
- Производительность. Система должна обладать настолько хорошим быстродействием и временем реакции, насколько это позволяет аппаратная платформа.

Рассмотрим более подробно некоторые из этих требований.

10.1.1. Требования по расширяемости

Операционные системы всегда эволюционно изменяются со временем, и эти изменения более значимы, чем изменения аппаратных средств. Изменения ОС обычно представляют собой приобретение ею новых свойств. Например, поддержка новых устройств, таких как CD-ROM, возможность связи с сетями нового типа, поддержка многообещающих технологий, таких как графический интерфейс пользователя или объектно-ориентированное программное окружение, использование более чем одного процессора. Сохранение целостности кода, какие бы изменения не вносились в операционную систему, является главной целью разработки.

Расширяемость достигаться путем использования следующих подходов:

1. За счет модульной структуры ОС, при которой программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс. Новые компоненты могут быть добавлены в операционную систему модульным путем, они выполняют свою работу, используя интерфейсы, поддерживаемые существующими компонентами.
2. Использование объектов для представления системных ресурсов также улучшает расширяемость системы.

   Объекты – это абстрактные типы данных, над которыми можно производить только те действия, которые предусмотрены специальным набором объектных функций. Объекты позволяют единообразно управлять системными ресурсами. Добавление новых объектов не разрушает существующие объекты и не требует изменений существующего кода.
3. Прекрасные возможности для расширения предоставляет подход к структурированию ОС по типу клиент-сервер с использованием микроядерной технологии. В соответствии с этим подходом ОС строится как совокупность привилегированной управляющей программы и набора непривилегированных услуг-серверов. Основная часть ОС может оставаться неизменной в то время, как могут быть добавлены новые серверы или улучшены старые.
4. Средства вызова удаленных процедур (RPC – Remote Procedure Call) позволяют добавить новые программные процедуры в любую машину сети и немедленно поступить в распоряжение прикладных программ на других машинах сети.
5. Загружаемые драйверы, которые могут быть добавлены в систему во время ее работы. Новые файловые системы, устройства и сети могут поддерживаться путем написания драйвера устройства, драйвера файловой системы или транспортного драйвера и загрузки его в систему.

10.1.2. Требования по переносимости

Требование *переносимости кода* тесно связано с расширяемостью. *Расширяемость* позволяет улучшать операционную систему, в то время как переносимость дает возможность перемещать всю систему на машину, базирующуюся на другом процессоре или аппаратной платформе, делая при этом по возможности небольшие изменения в коде.

Написание переносимой ОС аналогично написанию любого переносимого кода – нужно следовать некоторым правилам.

- Во-первых, большая часть кода должна быть написана на языке, который имеется на всех машинах, куда вы хотите переносить систему. Обычно это означает, что код должен быть написан на языке высокого уровня, предпочтительно стандартизованном, например, на языке С. Программа, написанная на ассемблере, не является переносимой, если только вы не собираетесь переносить ее на машину, обладающую командной совместимостью с вашей.
- Во-вторых, следует учесть, в какое физическое окружение программа должна быть перенесена. Различная аппаратура требует различных решений при создании ОС. Например, ОС, построенная на 32-битовых адресах, не может быть перенесена на машину с 16-битовыми адресами (разве что с огромными трудностями).
- В-третьих, важно минимизировать или, если возможно, исключить те части кода, которые непосредственно взаимодействуют с аппаратными средствами. Некоторые очевидные формы зависимости включают прямое манипулирование регистрами и другими аппаратными средствами.
- В-четвертых, если аппаратно зависимый код не может быть полностью исключен, то он должен быть изолирован в нескольких хорошо локализуемых модулях. Аппаратно-зависимый код не должен быть распределен по всей системе. Например, можно спрятать аппаратно-зависимую структуру в программно-задаваемые данные абстрактного типа. Другие модули системы будут работать с этими данными, а не с аппаратурой, используя набор некоторых функций. Когда ОС переносится, то изменяются только эти данные и функции, которые ими манипулируют.

Для легкого переноса ОС при ее разработке должны быть соблюдены нижеуказанные требования.

- *Переносимый язык высокого уровня.* Большинство переносимых ОС написано на языке С (стандарт ANSI X3.159-1989). Ассемблер используется только для тех частей системы, которые должны непосредственно взаимодействовать с аппаратурой (например, обработчик прерываний). Однако непереносимый код должен быть тщательно изолирован внутри тех компонентов, где он используется.

- *Изоляция процессора*. Некоторые низкоуровневые части ОС должны иметь доступ к процессорно-зависимым структурам данных и регистрам. Однако код, который делает это, должен содержаться в небольших модулях, которые могут быть заменены аналогичными модулями для других процессоров.
- *Изоляция платформы*. Зависимость от платформы заключается в различиях между рабочими станциями разных производителей, построенными на одном и том же процессоре. Должен быть введен программный уровень, абстрагирующий аппаратуру (кэши, контроллеры прерываний ввода-вывода и т.п.) вместе со слоем низкоуровневых программ таким образом, чтобы высокоуровневый код не нуждался в изменении при переносе с одной платформы на другую.

10.1.3. Требования по совместимости

Совместимость – способность ОС выполнять программы, написанные для других ОС или для более ранних версий этой же операционной системы, а также для другой аппаратной платформы.

Необходимо разделять вопросы двоичной совместимости и совместимости на уровне исходных текстов приложений.

Двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее на выполнение на другой ОС. Для этого необходимы: совместимость на уровне команд процессора, совместимость на уровне системных вызовов и даже на уровне библиотечных вызовов, если они являются динамически связываемыми.

Совместимость на уровне исходных текстов требует наличия соответствующего компилятора в составе программного обеспечения, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция имеющихся исходных текстов в новый выполняемый модуль.

Обладает ли новая ОС двоичной совместимостью или совместимостью исходных текстов с существующими системами, зависит от многих факторов. Самый главный из них – архитектура процессора, на котором работает новая ОС. Если процессор, на который переносится ОС, использует тот же набор команд (возможно с некоторыми добавлениями) и тот же диапазон адресов, тогда двоичная совместимость может быть достигнута достаточно просто.

Гораздо сложнее достичь двоичной совместимости между процессорами, основанными на разных архитектурах. Для того, чтобы один компьютер выполнял программы другого (например, DOS-программу на Mac), этот компьютер должен работать с машинными командами, которые ему изначально непонятны. Выходом в таких случаях является использование так называемых прикладных сред. Учитывая, что основную часть программы, как правило, составляют вызовы библиотечных функций, при-

кладная среда имитирует библиотечные функции целиком, используя заранее написанную библиотеку функций аналогичного назначения, а остальные команды эмулирует каждую по отдельности.

Соответствие стандартам POSIX также является средством обеспечения совместимости программных и пользовательских интерфейсов. Использование стандарта POSIX (IEEE стандарт 1003.1 – 1988) позволяет создавать программы стиле UNIX, которые могут легко переноситься из одной системы в другую.

10.1.4. Требования по безопасности

Правила безопасности определяют такие свойства, как *защита ресурсов* одного пользователя от других и установление *квот по ресурсам* для предотвращения захвата одним пользователем всех системных ресурсов (таких как память).

Обеспечение защиты информации от несанкционированного доступа является обязательной функцией сетевых операционных систем.

Основы стандартов в области безопасности были заложены «Критериями оценки надежных компьютерных систем». Этот документ, изданный в США в 1983 г. национальным центром компьютерной безопасности (NCSC – National Computer Security Center), часто называют Оранжевой Книгой.

В соответствии с требованиями безопасности безопасной считается такая система, которая «посредством специальных механизмов защиты контролирует доступ к информации таким образом, что только имеющие соответствующие полномочия лица или процессы, выполняющиеся от их имени, могут получить доступ на чтение, запись, создание или удаление информации».

Иерархия уровней безопасности, приведенная в Оранжевой Книге, помечает низший уровень безопасности как *D*, а высший – как *A*.

1. В класс *D* попадают системы, оценка которых выявила их несоответствие требованиям всех других классов.
2. Основными свойствами, характерными для *C*-систем, являются: наличие подсистемы учета событий, связанных с безопасностью, и избирательный контроль доступа.

Уровень *C* делится на 2 подуровня:

- уровень *C*1, обеспечивающий защиту данных от ошибок пользователей, но не от действий злоумышленников.
- уровень *C*2. На уровне *C2* должны присутствовать:
    - *средства аутентификации входа,* обеспечивающие идентификацию пользователей путем ввода уникального имени и пароля перед тем, как им будет разрешен доступ к системе;
    - *избирательный контроль доступа*, требуемый на этом уровне позволяет владельцу ресурса определить, кто имеет

доступ к ресурсу и что он может с ним делать. Владелец делает это путем предоставляемых прав доступа пользователю или группе пользователей;
    - *средства учета и наблюдения* (*auditing*) – обеспечивают возможность обнаружить и зафиксировать важные события, связанные с безопасностью, или любые попытки создать, получить доступ или удалить системные ресурсы;
    - *защита памяти* – заключается в том, что память инициализируется перед тем, как повторно используется. На этом уровне система не защищена от ошибок пользователя, но поведение его может быть проконтролировано по записям в журнале, оставленным средствами наблюдения и учета.
3. Системы уровня *B* основаны на помеченных данных и распределении пользователей по категориям, то есть реализуют *мандатный контроль доступа*. Каждому пользователю присваивается рейтинг защиты, и он может получать доступ к данным только в соответствии с этим рейтингом. Этот уровень в отличие от уровня *C* защищает систему от ошибочного поведения пользователя.
4. Уровень *A* является самым высоким уровнем безопасности, он требует в дополнение ко всем требованиям уровня *B* выполнения формального, математически обоснованного доказательства соответствия системы требованиям безопасности.

Различные коммерческие структуры (например, банки) особо выделяют необходимость учетной службы, аналогичной той, что предлагают государственные рекомендации *C*2. Любая деятельность, связанная с безопасностью, может быть отслежена и тем самым учтена. Это как раз то, что требует *C*2 и то, что обычно нужно банкам.

10.2. Тенденции в структурном построении операционных систем

Как уже отмечалось выше, для удовлетворения требований, предъявляемых к современной ОС, большое значение имеет ее структурное построение. Операционные системы прошли длительный путь развития от монолитных систем к хорошо структурированным модульным системам, способным к развитию, расширению и легкому переносу на новые платформы.

10.2.1. Монолитные системы

В общем случае «структура» *монолитной системы* представляет собой отсутствие структуры (рис. 10.1). ОС написана как набор процедур, каждая из которых может вызывать другие, когда ей это нужно. При использовании этой техники каждая процедура системы имеет хорошо определенный интерфейс в терминах параметров и результатов, и каждая воль-

на вызвать любую другую для выполнения некоторой нужной для нее полезной работы.

Для построения монолитной системы необходимо скомпилировать все отдельные процедуры, а затем связать их вместе в единый объектный файл с помощью *компоновщика*. Каждая процедура видит любую другую процедуру (в отличие от структуры, содержащей модули, в которой большая часть информации является локальной для модуля, и процедуры модуля можно вызвать только через специально определенные точки входа).

Однако даже такие монолитные системы могут быть немного структурированными. Такая организация ОС предполагает следующую структуру:

- главная программа, которая вызывает требуемые сервисные процедуры;
- набор сервисных процедур, реализующих системные вызовы;
- набор утилит, обслуживающих сервисные процедуры.

В этой модели для каждого системного вызова имеется одна сервисная процедура. Утилиты выполняют функции, которые нужны нескольким сервисным процедурам. Это деление процедур на три слоя показано на рис. 10.2.

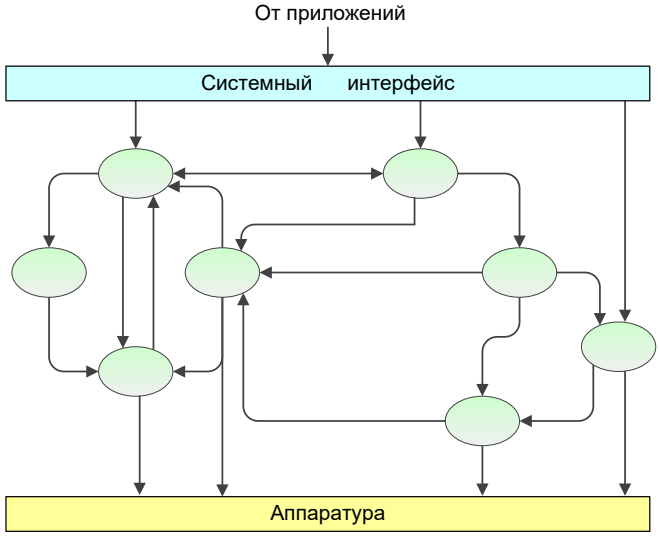


Рис. 10.1. Монолитная структура ОС

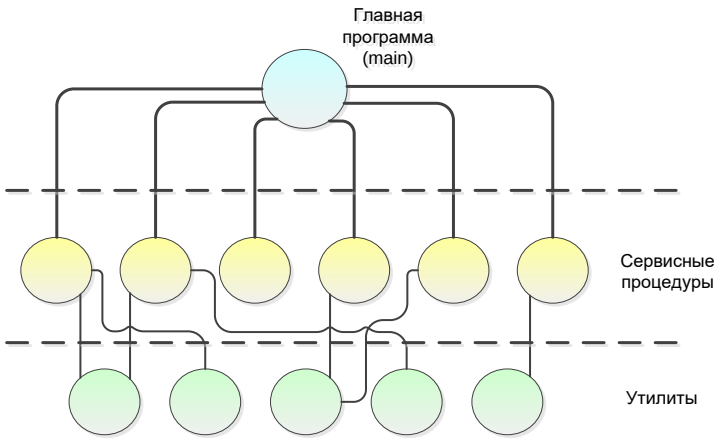


Рис. 10.2. Простая структуризация монолитной ОС

10.2.2. Многоуровневые системы

Обобщением предыдущего подхода является организация ОС как *иерархии уровней.* Уровни образуются группами функций операционной системы – файловая система, управление процессами и устройствами и т.п. Каждый уровень может взаимодействовать только со своим непосредственным соседом – выше- или нижележащим уровнем. Прикладные программы или модули самой операционной системы передают запросы вверх и вниз по этим уровням.

Первой системой, построенной таким образом была простая пакетная система THE, которую построил Дейкстра и его студенты в 1968 г. Система имела 6 уровней.

- Уровень 0 занимался распределением времени процессора, переключая процессы по прерыванию или по истечении времени.
- Уровень 1 управлял памятью – распределял оперативную память и пространство на магнитном барабане для тех частей процессов (страниц), для которых не было места в ОП, то есть слой 1 выполнял функции виртуальной памяти.
- Слой 2 управлял связью между консолью оператора и процессами. С помощью этого уровня каждый процесс имел свою собственную консоль оператора.
- Уровень 3 управлял устройствами ввода-вывода и буферизовал потоки информации к ним и от них. С помощью уровня 3 каждый процесс вместо того, чтобы работать с конкретными устройствами, с их разнообразными особенностями, обращался к абстрактным устройствам ввода-вывода, обладающим удобными для пользователя характеристиками.
- На уровне 4 работали пользовательские программы, которым не надо было заботиться ни о процессах, ни о памяти, ни о консоли, ни об управлении устройствами ввода-вывода.
- Процесс системного оператора размещался на уровне 5.

Многоуровневый подход был также использован при реализации различных вариантов ОС UNIX.

Хотя такой структурный подход на практике обычно работал неплохо, сегодня он все больше воспринимается монолитным. В системах, имеющих многоуровневую структуру было нелегко удалить один слой и заменить его другим в силу множественности и размытости интерфейсов между слоями. Добавление новых функций и изменение существующих требовало хорошего знания операционной системы и массы времени.

Когда стало ясно, что операционные системы должны иметь возможности развития и расширения, монолитный подход стал давать трещину, и на смену ему пришла *модель клиент-сервер* и тесно связанная с ней *концепция микроядра*.

10.2.3. Модель клиент-сервер и микроядра

Модель клиент-сервер – это еще один подход к структурированию ОС, которая предполагает наличие программного компонента-потребителя какого-либо сервиса- клиента, и программного компонента-поставщика этого сервиса-сервера. Взаимодействие между клиентом и сервером стандартизуется, так что сервер может обслуживать клиентов, реализованных различными способами и, может быть, разными производителями.

Инициатором обмена обычно является клиент, который посылает запрос на обслуживание серверу, находящемуся в состоянии ожидания запроса. Один и тот же программный компонент может быть клиентом по отношению к одному виду услуг, и сервером для другого вида услуг.

Модель клиент-сервер успешно применяется не только при построении ОС, но и на всех уровнях программного обеспечения, и имеет в некоторых случаях более узкий, специфический смысл, сохраняя, естественно, при этом все свои общие черты.

Применительно к структурированию ОС идея «клиент-сервер» состоит в разбиении ее на несколько процессов – серверов, каждый из которых выполняет отдельный *набор сервисных функций* – например, управление памятью, создание или планирование процессов.

Каждый сервер выполняется в пользовательском режиме. Клиент, которым может быть либо другой компонент ОС, либо прикладная программа, запрашивает сервис, посылая сообщение на сервер. Ядро ОС (называемое здесь микроядром), работая в привилегированном режиме, доставляет сообщение нужному серверу, сервер выполняет операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения (рис. 10.3).

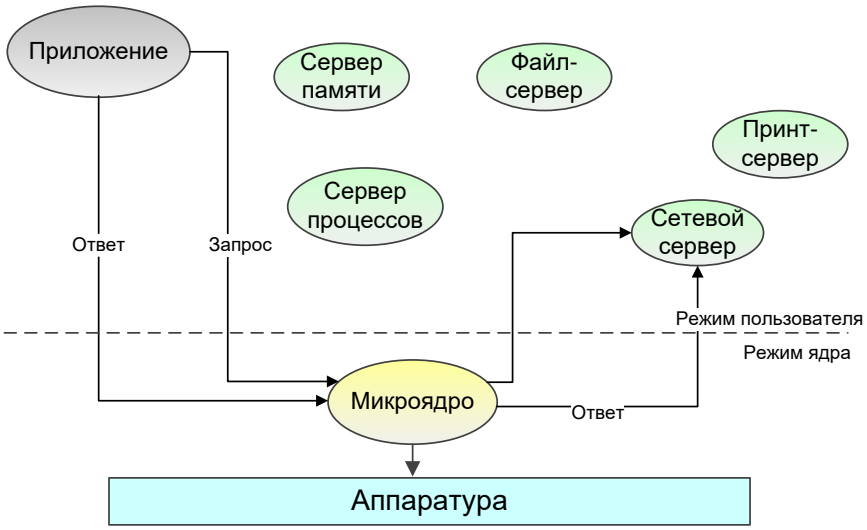


Рис. 10.3. Структура ОС клиент-сервер

Подход с использованием микроядра заменил вертикальное распределение функций операционной системы на горизонтальное. Компоненты, лежащие выше микроядра, хотя и используют сообщения, пересылаемые

через микроядро, взаимодействуют друг с другом непосредственно. Микроядро играет роль *регулировщика.* Оно проверяет сообщения, пересылает их между серверами и клиентами, и предоставляет доступ к аппаратуре.

Данная теоретическая модель является идеализированным описанием системы клиент-сервер, в которой ядро состоит только из средств передачи сообщений. В действительности различные варианты реализации модели клиент-сервер в структуре ОС могут существенно различаться по объему работ, выполняемых в режиме ядра.

Микроядро реализует жизненно важные функции, лежащие в основе операционной системы. Это базис для менее существенных системных служб и приложений. В общем случае, подсистемы, бывшие традиционно неотъемлемыми частями операционной системы – файловые системы, управление окнами и обеспечение безопасности – становятся периферийными модулями, взаимодействующими с ядром и друг с другом.

Главный принцип разделения работы между микроядром и окружающими его модулями – включать в микроядро только те функции, которым абсолютно необходимо исполняться в режиме супервизора и в привилегированном пространстве. Под этим обычно подразумеваются:

- машиннозависимые программы (включая поддержку нескольких процессоров);
- некоторые функции управления процессами;
- обработка прерываний;
- поддержка пересылки сообщений;
- некоторые функции управления устройствами ввода-вывода, связанные с загрузкой команд в регистры устройств.

Эти функции операционной системы трудно, если не невозможно, выполнить программам, работающим в пространстве пользователя.

В настоящее время именно операционные системы, построенные с использованием модели клиент-сервер и концепции микроядра, в наибольшей степени удовлетворяют требованиям, предъявляемым к современным ОС нижеуказанные свойства.

- *Высокая степень переносимости* обусловлена тем, что весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.
- *Технология микроядер* является основой построения множественных прикладных сред, которые обеспечивают совместимость программ, написанных для разных ОС. Абстрагируя интерфейсы прикладных программ от расположенных ниже операционных систем, микроядра позволяют гарантировать, что вложения в прикладные программы не пропадут в течение нескольких лет, даже если будут сменяться операционные системы и процессоры.
- *Расширяемость* также является одним из важных требований к современным операционным системам. Увеличивающаяся сложность монолитных операционных систем делала трудным, если

вообще возможным, внесение изменений в ОС с гарантией надежности ее последующей работы. Ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС.

- *Использование модели клиент-сервер* повышает надежность. Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти, и таким образом защищен от других процессов. Более того, поскольку серверы выполняются в пространстве пользователя, они не имеют непосредственного доступа к аппаратуре и не могут модифицировать память, в которой хранится управляющая программа. И если отдельный сервер может потерпеть крах, то он может быть перезапущен без останова или повреждения остальной части ОС. Эта модель хорошо подходит для распределенных вычислений, так как отдельные серверы могут работать на разных процессорах мультипроцессорного компьютера или даже на разных компьютерах.

10.2.4. Объектно-ориентированный подход

В настоящее время для цели обеспечения расширяемости в наибольшей степени соответствует объектно-ориентированный подход, при котором каждый программный компонент является функционально изолированным от других.

Основным понятием этого подхода является «объект».

Объект – это единица программ и данных, взаимодействующая с другими объектам посредством приема и передачи сообщений. Объект может быть представлением как некоторых конкретных вещей – прикладной программы или документа, так и некоторых абстракций – процесса, события.

Программы (функции) объекта определяют перечень действий, которые могут быть выполнены над данными этого объекта. Объект-клиент может обратиться к другому объекту, послав сообщение с запросом на выполнение какой-либо функции объекта-сервера.

Для обеспечения преемственности при переходе к более детальному описанию разработчикам предлагается *механизм наследования* свойств уже существующих объектов, то есть механизм, позволяющий порождать более конкретные объекты из более общих.

Например, при наличии объекта «текстовый документ» разработчик может легко создать объект «текстовый документ в формате Word», добавив соответствующее свойство к базовому объекту. Механизм наследования позволяет создать иерархию объектов, в которой каждый объект более низкого уровня приобретает все свойства своего предка.

Внутренняя структура данных объекта скрыта от наблюдения. Нельзя произвольно изменять данные объекта. Для того, чтобы получить данные из объекта или поместить данные в объект, необходимо вызывать со-

ответствующие объектные функции. Это изолирует объект от того кода, который использует его. Разработчик может обращаться к функциям других объектов, или строить новые объекты путем наследования свойств других объектов, ничего не зная о том, как они сконструированы. Это свойство называется инкапсуляцией.

Таким образом, объект предстает для внешнего мира в виде «черного ящика» с хорошо определенным интерфейсом. С точки зрения разработчика, использующего объект, пока внешняя реакция объекта остается без изменений, не имеют значения никакие изменения во внутренней реализации. Это дает возможность легко заменять одну реализацию объекта другой, например, в случае смены аппаратных средств; при этом сложное программное окружение, в котором находятся заменяемые объекты, не потребует никаких изменений.

Использование объектно-ориентированного подхода особенно эффективно при создании активно развивающегося программного обеспечения, например, при разработке приложений, предназначенных для выполнения на разных аппаратных платформах.

Полностью объектно-ориентированные операционные системы очень привлекательны так как, используя объекты системного уровня, программисты смогут залезать вглубь операционных систем для приспособления их к своим нуждам, не нарушая целостность системы.

Но особенно большие перспективы имеет этот подход в реализации распределенных вычислительных сред. В то время, как сейчас разные пакеты, работающие в этот момент в сети, представляют собой статически связанные наборы программ, в будущем, с использованием объектно-ориентированного подхода, они могут превратиться в единую совокупность динамически связываемых объектов, где каждый объект оперативно устанавливает и разрывает связи с другими объектами для выполнения актуальных в этот момент задач. Приложения, созданные для такой сетевой среды, основанной на объектах, могут выполняться, динамически обращаясь к множеству объектов, независимо от их местонахождения в сети и независимо от их операционной среды.

10.2.5. Множественные прикладные среды

В то время как некоторые идеи (например, объектно-ориентированный подход) непосредственно касаются только разработчиков и лишь косвенно влияют на конечного пользователя, *концепция множественных прикладных сред* приносит пользователю долгожданную возможность выполнять на своей ОС программы, написанные для других операционных систем и других процессоров.

Множественные прикладные среды обеспечивают совместимость конкретной ОС с приложениями, написанными для других ОС и процессоров, на двоичном уровне, а не на уровне исходных текстов.

При реализации множественных прикладных сред разработчики сталкиваются с противоречивыми требованиями.

1. С одной стороны, задачей каждой прикладной среды является выполнение программы по возможности так, как если бы она выполнялась на «родной» ОС.
2. С другой стороны потребности этих программ могут входить в конфликт с конструкцией современной операционной системы:
   - специализированные драйверы устройств могут противоречить требованиям безопасности;
   - могут конфликтовать схемы управления памятью и оконные системы;
   - чисто экономические вопросы (например, стоимость лицензирования программ и угроза судебного преследования) также могут повлиять на дизайн чужих прикладных сред;
   - большой потенциальной проблемой является производительность – прикладная среда должна выполнять программы с приемлемой скоростью.

Для сокращения времени на выполнение чужих программ прикладные среды используют имитацию программ на уровне библиотек. Эффективность этого подхода связана с тем, что большинство сегодняшних программ работают под управлением GUI (Graphical User Interface – графических интерфейсов пользователя). Они непрерывно выполняют вызовы библиотек GUI для манипулирования окнами и для других связанных с GUI действий. И это то, что позволяет прикладным средам возместить время, потраченное на эмулирование команды за командой. Тщательно сделанная прикладная среда имеет в своем составе библиотеки, имитирующие внутренние библиотеки GUI, но написанные на родном коде, то есть она совместима с программным интерфейсом другой ОС. Иногда такой подход называют трансляцией для того, чтобы отличать его от более медленного процесса эмулирования кода по одной команде за раз.

С позиции использования прикладных сред более предпочтительным является способ написания программ, при котором программист для выполнения некоторой функции обращается с вызовом к операционной системе, а не пытается более эффективно реализовать эквивалентную функцию самостоятельно, работая напрямую с аппаратурой.

Модульность операционных систем нового поколения позволяет намного легче реализовать поддержку множественных прикладных сред. В отличие от старых операционных систем, состоящих из одного большого блока для всех практических применений, разбитого произвольным образом на части, новые системы являются модульными, с четко определенными интерфейсами между составляющими. Это делает создание дополнительных модулей, объединяющих эмуляцию процессора и трансляцию библиотек, значительно более простым.

Существует много разных стратегий по воплощению идеи множественных прикладных сред, и некоторые из этих стратегий диаметрально противоположны.

- В случае UNIX, транслятор прикладных сред обычно делается, как и другие прикладные программы, плавающим на поверхности операционной системы.
- В более современных ОС типа Windows модули прикладной среды выполняются более тесно связанными с операционной системой, хотя и обладают по-прежнему высокой независимостью.
- В OS/2 с ее более простой, слабо структурированной архитектурой средства организации прикладных сред встроены глубоко в операционную систему.

Использование множественных прикладных сред обеспечит пользователям большую свободу выбора операционных систем и более легкий доступ к более качественному программному обеспечению.

11. Сетевая операционная система Windows Server

11.1. Операционная система Windows: особенности, архитектура, функциональность

Операционная система Windows в настоящее время является самой популярной из операционных систем. Общая доля Windows на рынке десктопных ОС (на ноябрь 2017 г.) составляет 82,74%. Для описания архитектуры и функциональности этой ОС в данном пособии использованы материалы из работы [7].

Отметим, что серверная ОС Windows в части внутренней архитектуры мало чем отличается от соответствующей ей по выпуску десктопной ОС. Основное различие заключается в том, что версии Windows Server конфигурируется по функционалу с помощь установки так называемых ролей и компонентов. После их установки в соответствии с инфраструктурой предприятия ОС Windows Server становится контроллером домена, либо файловым сервером, либо коммуникационным сервером и т.д.

11.1.1. Архитектура Windows

Симметричная многопроцессорная обработка данных (SMP) поддерживается во всех версиях Windows, но для управления высокопроизводительными настольными системами была специально разработана 64-разрядная версия Windows, В этой версии обеспечивается увеличение объема поддерживаемой памяти, необходимое для крупных SMP систем; 64-разрядная адресация теоретически позволяет работать с 264 байт (или 16 квинтиллионами байт) виртуального адресного пространства, что значительно отличается от допустимых 232 байт (или 4 Гбайт) в 32-разрядных версиях Windows. ОС Windows 64 Edition может работать с 7152 ГБ виртуальной памяти на системах IA-64 и 8192 Гбайт на x64. В настоящее время Windows 64-bit Edition поддерживает 1 терабайт кэш-памяти, 128 ГБ системной памяти со страничной организацией и 128 ГБ резидентного пула. Существующие 64-разрядные версии Windows 7/8/10 могут работать на процессорах с технологией EM64T, а также на 64-разрядных процессорах AMD. Эти версии Windows обеспечивают более высокое быстродействие и точность вычислений при работе с числами с плавающей запятой, поскольку для их хранения используются 64-битовые значения.

Архитектура. Windows (в этом разделе архитектурные особенности рассматриваются на примере Windows 7 если явно не указывается иное) построена на модифицированной архитектуре микроядра. Windows обладает модульной структурой и компактным ядром, которое предоставляет базовые сервисы для других компонентов ОС. Однако в отличие от ОС, построенных исключительно на архитектуре микроядра, ряд компонентов

ОС (например, диспетчер памяти или файловая система) выполняются не в пользовательском режиме, а в режиме ядра. Windows представляет собой многоуровневую ОС (рис. 11.1), в которой отдельные уровни подчиняются общей иерархии, но, в отличие от строго иерархических систем, возможны ситуации, когда между собой могут общаться и несмежные уровни.

Рис. 11.1. Архитектура Windows

Уровень абстракций аппаратуры (Hardware Abstraction Layer – HAL) взаимодействует непосредственно с оборудованием, скрывая детали взаимодействия ОС с конкретной аппаратной платформой для остальной части системы. HAL позволяет абстрагироваться от конкретного оборудования, которое в разных системах с одной и той же архитектурой может быть различным. Кроме того, уровень абстракций аппаратуры взаимодействует с микроядром. Микроядро обеспечивает основные механизмы функционировали системы, такие как планирование выполнения потоков для поддержи других компонентов режима ядра.

Микроядро управляет синхронизацией потоков, обслуживанием прерываний и обработкой исключений. Кроме того, микроядро позволяет не учитывать архитектурно-зависимые функциональные возможности, например, количество прерываний в различных архитектурах. Таким образом, микроядро и HAL обеспечивают переносимость операционной системы Windows, позволяя ей работать на самом разнообразном оборудовании.

Выше уровня ядра расположена исполняющая система, включающая компоненты режима ядра, отвечающие за администрирование подсистем ОС (например, диспетчер ввода-вывода, диспетчер виртуальной памяти, монитор безопасности, диспетчер устройств plug-and-play и др.). Собира-

тельное название этих компонентов исполняющие или управляющие программы (executive). Исполняющие программы предоставляют услуги пользовательским процессам через интерфейс прикладного программирования (API).

Тем не менее, большинство пользовательских процессов обращаются не к функциям собственного API, а вызывают функции API, предоставляемые системными компонентами пользовательского режима, которые называют подсистемами среды окружения (environment subsystems). Пользовательские приложения не вызывают напрямую системные службы Windows, вместо этого ими используется одна или несколько DLL-библиотек подсистемы. Хотя Windows и позволяет процессам, выполняемым в пользовательском режиме, вызывать функции собственного API, Microsoft рекомендует разработчикам использовать интерфейс подсистемы операционной среды.

По умолчанию, 32-разрядная Windows включает в себя только подсистему Win32. Для запуска 16-разрядных приложений DOS, Windows использует виртуальную машину DOS (Virtual DOS Machine – VDM), представляющую собой процесс Win32, который создает среду DOS для выполнения DOS-приложений.

В 64-разрядной версии Windows используемая по умолчанию подсистема среды поддерживает 64-разрядные приложения, но здесь предусмотрена подсистема под названием «Windows on Windows 64» (WOW64), позволяющая выполнять 32-разрядные приложения. Однако из 64-разрядной версии Windows убрана поддержка 16-разрядных приложений.

На верхнем уровне архитектуры Windows расположены процессы, выполняемые в пользовательском режиме. К ним относятся широко распространенные приложения (например, текстовые процессоры, компьютерные игры и веб-браузеры), а также динамически подключаемые библиотеки (DLL – dynamic-linked library). Библиотеки DLL это модули, предоставляющие процессам функции и данные. Поскольку библиотеки DLL подключаются динамически, это позволяет повысить модульность приложений. В случае внесения изменений в реализацию функций в DLL, достаточно будет перекомпилировать только подключаемую библиотеку, а не все использующее ее приложение.

В Windows также существуют специальные процессы пользовательского режима, называемые *системными службами* (system service) или *службами* Win32, которые напоминают демонов Linux. Эти процессы обычно выполняются в фоновом режиме, независимо от того, произошел ли вход пользователя в систему. В качестве примера системных служб можно привести:

- планировщик (с помощью которого пользователи могут задавать выполнение задач в определенные моменты времени);
- IPSec (который контролирует безопасность Интернета во время операций по передаче данных);

- службу «обозреватель компьютеров» (которая обеспечивает работу со списком компьютеров, подключенных к локальной сети).

11.1.2. Организация прерываний

В Windows используется концепция уровней прерываний (interrupt request level, IRQL). Уровни прерываний представляют собой способ описания приоритета прерываний. Одиночный процессор всегда работает с одним определенным уровнем прерываний, а в многопроцессорной системе разные процессоры могут работать на различных уровнях прерываний. Прерывание, выполняемое на более высоком уровне, может прервать обработку текущего прерывания и получить контроль над процессором. Система маскирует (то есть откладывает обработку) прерываний, которые выполняются на том же или более низком уровне, что и текущее прерывание. В Windows предусмотрены несколько уровней прерываний (рис. 11.2).

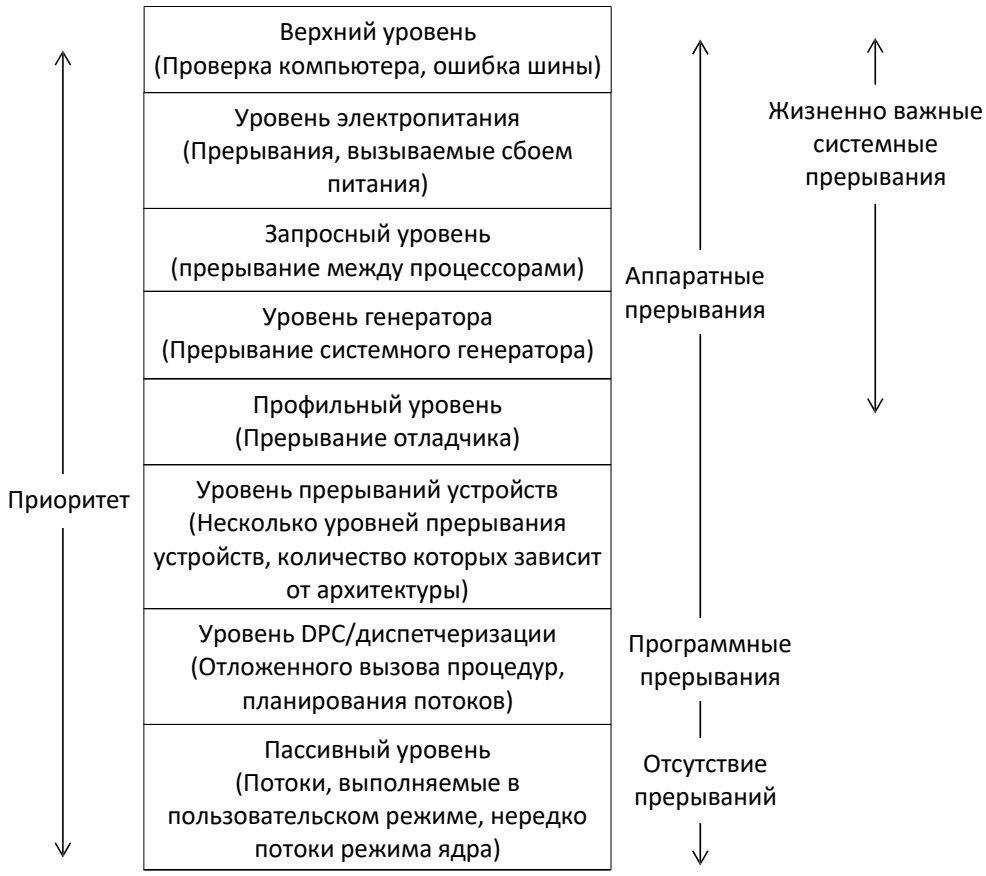


Рис. 11.2. Уровни прерываний в Windows

Потоки, исполняемые в режиме ядра и пользовательском режиме, обычно обрабатываются на пассивном уровне прерываний (passive IRQL) этот уровень прерываний имеет самый низкий приоритет.

Выше расположен уровень отложенного вызова процедур и диспетчеризации (DPC/Dispatch IRQL), на котором обрабатываются, программные прерывания, которые могут выполняться в контексте любого потока, а

также другие важные функции ядра, включая планирование потоков. Все аппаратные прерывания обрабатываются на более высоких уровнях прерываний. В Windows для этой цели предусмотрено несколько уровней прерываний устройств (device IRQL). Количество этих уровней зависит от числа прерываний, поддерживаемых архитектурой. Привязка аппаратных прерываний к уровням прерываний устройств происходит в Windows без учета приоритета. Пять верхних уровней прерываний зарезервированы системой для жизненно важных системных прерываний.

Первый уровень, размещающийся выше уровня прерываний устройств, носит название профильного уровня прерываний (profile IRQL), и используется при включенном режиме профилирования ядра. Системный генератор периодически формирует прерывания на уровне прерываний генератора (clock IRQL). Запросы от одного процессора к другому выполняются на запросном уровне прерываний (request IRQL). Уровень прерываний электропитания (power IRQL) зарезервирован для уведомления о сбоях по питанию. К прерываниям верхнего уровня (high IRQL) относят прерывания, связанные со сбоями оборудования и ошибками шин.

На каждом процессоре уровень аппаратных абстракций (HAL) маскирует все прерывания, уровень которых равен или меньше уровня прерываний, на котором работает этот процессор в конкретный момент времени. Таким образом, система распределяет прерывания по приоритетам, что позволяет ей обрабатывать важные прерывания настолько быстро, насколько это возможно.

11.1.3. Управление процессами и потоками

В Windows процесс состоит из программного кода, контекста выполнения, ресурсов (например, дескрипторов объектов) и одного или нескольких связанных потоков. Контекст выполнения включает виртуальное адресное пространство процесса и различные атрибуты (например, атрибуты безопасности и ограничения на размер рабочего набора). Потоки представляют собой элементарную единицу выполнения, которая выполняет фрагменты кода процесса в контексте процесса, используя его же ресурсы. Поток обладает собственным контекстом выполнения, включающим стек времени выполнения, состояние машинных регистров и ряд атрибутов (например, связанных с приоритетом планирования). Потоки и процессы представляют собой объекты, дескрипторы которых могут передаваться другим процессам. Кроме того, объекты процессов и потоков могут ожидать от других потоков или процессов наступления определенных событий так, как это имеет место для объектов других типов.

Windows хранит информацию о контексте потоков и процессов в определенных структурах данных. Исполняющий блок процесса (executive process block, EPROCESS) представляет собой главную структуру данных, описывающую процесс. Он содержит информацию, которая используется исполняющими компонентами при работе с объектами процессов: сведе-

ния об ID процесса, указатель на таблицу дескрипторов процесса и указатель на маркер доступа процесса. Блоки процессов EPROCESS хранятся системой в виде связного списка. Внутри каждого блока EPROCESS находится ядро блока процесса (kernel process block, KPROCESS). В блоке KPROCESS содержится информация, используемая микроядром, которое управляет планированием потоков и их синхронизацией: приоритет процесса, выделяемый по умолчанию квант времени для каждого потока и его блокировки. Блоки KPROCESS и EPROCESS размещены в пространстве ядра и содержат сведения о компонентах режима ядра для доступа к ним во время работы процесса.

Хранение данных для потоков организовано в Windows аналогичным образом. Исполняющий блок потока (executive thread block, ETHREAD) представляет собой основную структуру данных, описывающую поток. Он хранит информацию, используемую исполняющими компонентами во время работы с объектами потока. Эти сведения, включают ID процесса, к которому принадлежит данный поток, начальный адрес, маркер доступа и список отложенных запросов ввода-вывода. Внутри каждого блока ETHREAD размещено ядро блока потока (kernel iread block, KTHREAD). Микроядро использует информацию, хранимую блоке KTHREAD, для планирования работы потоков и их синхронизации.

В ряде случаев процессы объединяются в группы, называемые заданиями. Задания (job) позволяют определять правила и накладывать ограничения на количество процессов. Процесс может входить в состав нескольких разных заданий.

Потоки могут создавать нити (fiber), которые напоминают потоки, но планирование выполнения нитей осуществляется создавшим их потоком, а не микроядром. Операционная система Windows использует нити для экспорта фрагментов кода, написанного для других операционных систем, в Windows. Нити представляют собой единицы выполнения внутри единого потока: каждая нить должна содержать информацию о состоянии, например, следующую выполняемую инструкцию либо значения регистров процессора. Поток хранит информацию о состоянии для каждой нити.

Windows планирует выполнение потоков, а не процессов и поддерживает вытесняющее планирование между несколькими потоками. Диспетчер задает расписание выполнения каждого потока независимо от принадлежности потока к тому или иному процессу. Алгоритм планирования основывается на приоритете потоков.

Каждый поток выполняется в течение хотя бы одного кванта времени. В случае приоритетного вытеснения, потоки реального времени восстанавливают исходное значение выделенного им кванта времени, тогда как все остальные потоки при повторном получении контроля над процессором выполняются только в течение оставшейся части выделенного им кванта. Все уровни приоритетов (0-31) в Windows поделены на две категории: потоки реального времени (то есть такие потоки, которые должны обеспечивать быструю реакцию на запросы пользователей) занимают

верхние 16 уровней (16-31) и являются статическими, а динамические потоки (ОС может динамически изменять их приоритеты) занимают нижние 16 уровней (0-15).

11.1.4. Управление памятью

Диспетчер виртуальной памяти (Virtual Memory Manager – VMM) Windows 7 создает для каждого процесса иллюзию обладания непрерывным пространством памяти объемом в 4 ГБ. Поскольку система выделяет больше виртуальной памяти для процессов, которые помещаются в основную память, VMM выгружает часть данных на диск в файлы, называемые файлами подкачки (pagefile). В Windows вся память поделена на страницы фиксированного размера, которые физически хранятся в основной памяти либо в файлах на диске. Диспетчер виртуальной памяти использует двухуровневую иерархическую систему адресации.

В Windows применяется две стратегии: оптимизация использования основной памяти и уменьшение количества дисковых операций ввода-вывода. VMM использует страницы, копируемые при записи и принцип отложенного (позднего) выделения (lazy allocation) то есть политику, при которой выделение страниц памяти и записей таблиц страниц откладывается до тех пор, пока в них не возникнет крайняя необходимость. Однако во время выполнения диспетчером виртуальной памяти операций ввода-вывода, он осуществляет упреждающее чтение страниц с диска с помещением их в основную память перед использованием. Когда основная память переполняется, Windows начинает использовать алгоритм замены дольше всего не использовавшихся страниц (LRU).

Система Windows обеспечивает работу в 32- или 64-разрядном адресном пространстве, в зависимости от используемого процессора и версии Windows (ограничимся обсуждением операций с памятью для 32-разрядной архитектуры). ОС выделяет каждому процессу виртуальное адресное пространство размером в 4 ГБ. По умолчанию, процесс может работать только с первыми двумя гигабайтами своего виртуального адресного пространства. Оставшиеся 2 ГБ система резервирует для использования компонентами режима ядра эта часть адресного пространства носит название системной. Физическая память делится на страничные блоки фиксированного размера, который равен 4 КБ в 32-разрядной системе. VMM использует виртуальные адреса вместе с картами памяти для трансляции виртуальных адресов в физические. Виртуальный адрес состоит из трех частей: смещения в таблице каталогов страниц, смещения в таблице страницы и смещения внутри страницы физической памяти. Windows осуществляет трансляцию виртуальных адресов в три этапа (рис. 11.3).

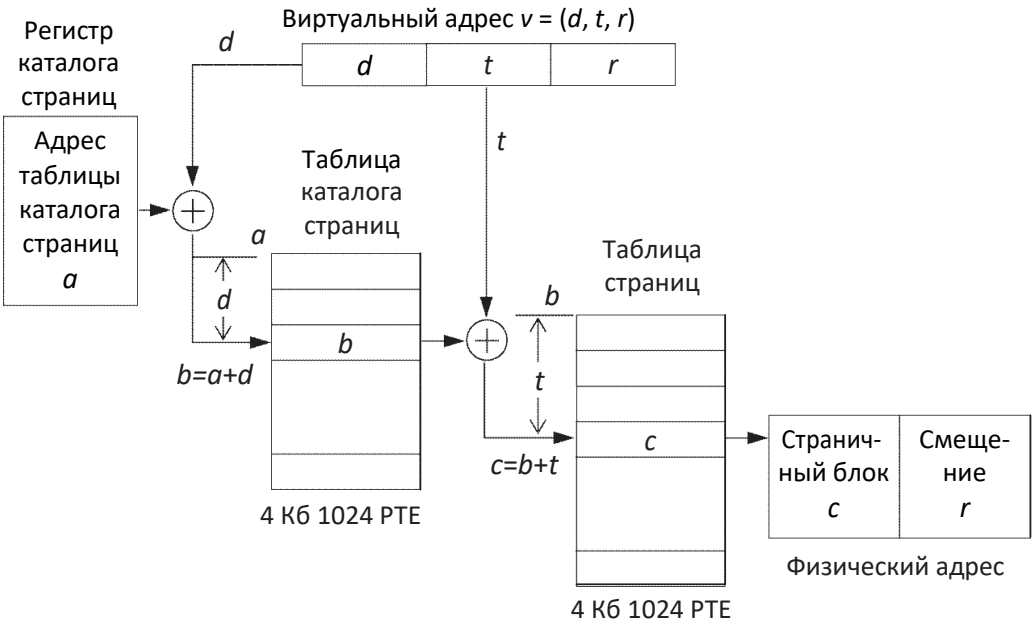


Рис. 11.3. Преобразование виртуальных адресов

На первом этапе преобразования адреса система вычисляет сумму (*a*+*d*), то есть складывает значение регистра каталога страниц с первой частью виртуального адреса, чтобы определить местоположение записи каталога страниц (PDE) в таблице каталога страниц. На втором этапе система вычисляет сумму (*b*+*t*), то есть складывает значение записи каталога страниц со второй частью виртуального адреса для определения записи таблицы страниц (PTE) в таблице страниц. В этом элементе хранится значение номера страничного блока (*c*), соответствующего размещению виртуальной страницы в основной памяти. И, наконец, на последнем этапе, система складывает с и смещение внутри страницы, чтобы получить физический адрес. Выполнение этих операций осуществляется достаточно быстро, задержки возникают лишь тогда, когда система вынуждена считывать PTE и PDE из основной памяти.

Пять из 32 разрядов записи таблицы страниц (PTE) предназначены для защиты. Они показывают, какие операции разрешено выполнять процессу над этой страницей памяти: чтение, запись или выполнение программного кода. 20-разрядный индекс указывает на страничный блок в памяти либо смещение в файле подкачки на диске этого достаточно, чтобы адресовать 1048576 виртуальных страниц. Если страница находится на диске, специальные четыре бита определяют, в каком из 16 возможных страничных файлов находится данная страница. Если страница расположена в оперативной памяти, три других бита определяют состояние страницы.

Выделение памяти в Windows осуществляется в три этапа. Сначала процесс должен зарезервировать (reserve memory) место в своем виртуальном адресном пространстве. Процессы могут выбирать диапазон резервируемых виртуальных адресов самостоятельно, либо предоставить право

выбора диспетчеру виртуальной памяти. Процесс не может обращаться к зарезервированному адресному пространству, пока оно не будет зафиксировано (commit memory). Когда процесс выполняет фиксацию страницы, VMM создает PTE, гарантируя, что для хранения страницы хватит места в основной памяти или страничном файле. Наконец, когда процесс готов к использованию зафиксированной памяти, он обращается к ней через виртуальную память. На этом этапе VMM записывает данные в очищенные страницы основной памяти. Такой трехэтапный процесс гарантирует, что процессы будут использовать ровно столько памяти, сколько им необходимо, а не всю зарезервированную память.

В Windows предусмотрен специальный механизм для борьбы с ситуациями, когда количество свободной памяти резко падает. Обнаружив, что в ее распоряжении осталось очень мало свободных страничных блоков, система переходит на постраничную обработку памяти. Регулирование ввода-вывода замедляет работу системы, поскольку VMM прекращает параллельную обработку запросов, извлекая с диска по одной странице одновременно, но в то же время это помогает сделать систему более устойчивой и предотвратить сбои.

Некоторые данные выгружать из памяти нельзя. Например, программный код, отвечающий за обработку прерываний, должен оставаться в основной памяти, поскольку обработка ситуации страничного промаха во время возникновения прерывания может вызвать неприемлемую задержку и даже вызвать сбой системы. Запрещенным с точки зрения рисков безопасности является замена страниц, содержащих пароли, поскольку в случае сбоя системы незашифрованная копия конфиденциальной информации окажется сохраненной на диске. Подобная информация хранится в специально выделенной области системной памяти, называемой невыгружаемым (резидентным) пулом (nonpaged pool). Данные, которые разрешено выгружать на диск, помещаются в нерезидентный пул (paged pool). Часть резидентного пула занимают драйверы устройств и программный код VMM.

В Windows уменьшено время, затрачиваемое на загрузку (запуск) приложений, включая саму ОС, благодаря предварительной (упреждающей) выборке файлов. Система записывает сведения о том, доступ к каким страницам и в какой последовательности осуществлялся в течение последних восьми запусков приложения. Перед загрузкой приложения система запрашивает все нужные страницы в рамках одного асинхронного запроса, уменьшая, таким образом, время поиска информации на диске.

Загрузка системы ускорена благодаря одновременному выполнению упреждающей выборки страниц и инициализации устройств. Во время запуска Windows должна инициализировать различные устройства ввода-вывода, поскольку без этого загрузка системы не может быть продолжена. Во время инициализации можно выполнить предварительную выборку страниц памяти, поскольку со всех других точек зрения система простаивает.

11.1.5. Управление файловой системой

Для работы с файловыми системами в Windows используется три уровня драйверов. На самом нижнем уровне находятся драйверы томов (volume driver), которые отвечают за управление конкретным оборудованием, например, жестким диском. Драйверы файловой системы (file system driver), формирующие следующий уровень, отвечают за реализацию системного формата конкретной файловой системы, например, NTFS или FAT. С помощью этих драйверов реализуется иерархическая организация хранения файлов и функции, предназначенные для управления этими файлами. Драйверы файловой системы обеспечивают взаимосвязь между логическим представлением файловой системы, видимым для приложений и пользователей и соответствующим ей физическим представлением на носителе информации. И, наконец, драйверы фильтров файловой системы (file system filter driver) выполняют такие задачи высокого уровня, как антивирусная защита, сжатие и шифрование. Windows поддерживается файловая система CDFS (compact disk file system – файловая система компакт-дисков) для работы с компакт-дисками, а также UDF (Universal Disk Format – универсальный дисковый формат) для CD-R(W) и DVD.

Файловой системой по умолчанию для Windows является NTFS. Также поддерживаются файловые системы FAT (FAT16, FAT32, exFAT). Подробное описание этих файловых систем представлено в подразделе 7.

11.1.6. Управление операциями ввода-вывода

Процессы пользовательского режима взаимодействуют с подсистемой среды (например, подсистемой Win32), а не непосредственно с компонентами режима ядра. Подсистема среды передает запросы ввода-вывода диспетчеру ввода-вывода (I/O manager), который взаимодействует с драйверами устройств для обработки подобных запросов. Нередко, несколько драйверов устройств, организованных в виде стека драйверов, занимаются совместным обслуживанием запросов ввода-вывода. Диспетчер Plug and Play (PnP manager) динамически распознает подключение к системе новых устройств (если эти устройства поддерживают технологию plug and play) и выполняет динамическое выделение или освобождение ресурсов, например, портов ввода-вывода либо каналов DMA, Большинство выпущенного в последнее время оборудования поддерживает технологию PnP. Диспетчер электропитания (power manager) регулирует политику энергопотребления в операционной системе. Рис. 11.4 иллюстрирует процесс ввода-вывода.

Windows хранит сведения о каждом устройстве в одном или нескольких объектах устройств (device objects). Объект устройства обычно содержит аппаратно-зависимую информацию (например, тип устройства и обрабатываемый в этот момент времени запрос ввода-вывода) и используется для обработки запросов ввода-вывода устройства. С определенным устройством могут связываться несколько объектов устройств, поскольку

обработкой запросов ввода-вывода для этих устройств занимается сразу несколько драйверов, организованных в стек драйверов (driver stack). В этом случае каждый драйвер создает свой объект устройства для этого оборудования. Стек драйверов состоит из нескольких драйверов, выполняющих разнообразные функции управления вводом/выводом.

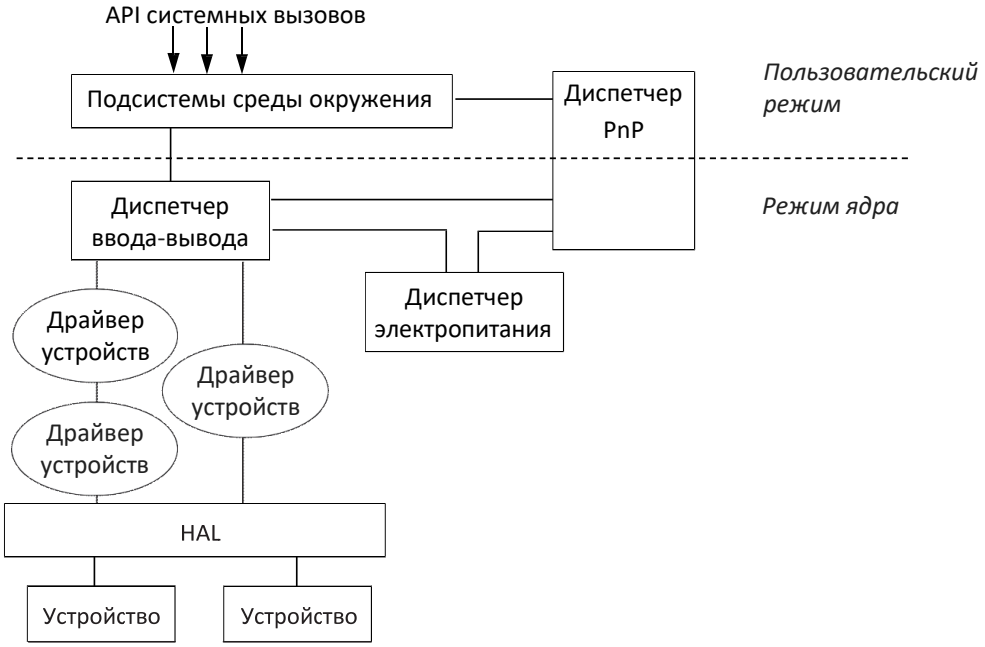


Рис. 11.4. Компоненты поддержки ввода-вывода в ОС Windows

Низкоуровневый драйвер (low-level driver) более тесно взаимодействует с HAL (уровнем абстракций аппаратуры). Такой драйвер обычно контролирует периферийное оборудование и не зависит от других низкоуровневых драйверов. В качестве примера низкоуровневого драйвера можно привести драйвер шины. Высокоуровневый драйвер (high-level driver) позволяет отвлечься от особенностей аппаратного обеспечения, передавая запросы ввода-вывода низкоуровневым драйверам. Драйвер файловой системы NTFS представляет собой высокоуровневый драйвер, для которого не имеет значения физический способ хранения данных на диске. Промежуточный драйвер (intermediate driver) может размещаться между высокоуровневыми и низкоуровневыми драйверами, выполняя функции фильтрации либо обработки запросов ввода-вывода, а также экспорта интерфейсов для определенных устройств. В качестве примера промежуточного драйвера можно привести драйвер класса (то есть драйвер, реализующий функции, общие для всего класса устройств).

Описанные типы драйверов называют драйверами режима ядра (kernel-mode drivers). Остальные драйверы, например, драйверы некоторых принтеров, относятся к драйверам пользовательского режима (user-mode drivers), которые выполняются в пространстве пользователя и зависят от подсистемы среды.

Все драйверы стека устройства должны использовать одинаковый метод ввода-вывода, иначе передача данных может быть прервана из-за выполнения драйверами конфликтующих операций.

11.1.7. Взаимодействие процессов

В Windows реализовано большинство традиционных механизмов: каналы, очереди сообщений и разделяемая память. Кроме того, в Windows взаимодействие процессов может осуществляться при помощи процедурно-ориентированных и объектно-ориентированных технологий (удаленный вызов процедур RPC, объектная модель программных компонентов от Microsoft Component Object Model – COM), а также взаимодействие через буфер обмена, технологию перетаскивания объектов (drag-and-drop), технологию связывания и внедрения объектов (OLE). В любом механизме взаимодействия процессов Windows, серверный процесс предоставляет доступ к некоторым коммуникационным объектам. Чтобы разместить запрос, клиентский процесс обращается к серверному процессу через этот коммуникационный объект. Запрос передается серверу для выполнения заданной функции либо возвращения данных или объектов.

11.1.8. Важнейшие компоненты операционной системы

11.1.8.1. Реестр

Реестр (Registry) представляет собой централизованную базу данных, которая содержит всю конфигурационную информацию аппаратной части, программного обеспечения, а также сведения о настройках пользователей.

Реестр содержит информацию об:

- оборудовании, установленном на компьютере, включая центральный процессор, тип шины, указательное устройство/ мышь и клавиатуру;
- установленных драйверах устройств;
- установленных приложениях;
- установленных сетевых протоколах;
- настройках платы сетевого адаптера (номер прерывания, базовый адрес памяти, базовый адрес портов ввода\вывода, тип трансивера);
- учетных записях пользователей (например, о принадлежности пользователей группам, их правах доступа и привилегиях).

Например, информация о пользователе может включать настройки рабочего стола и принтера. Настройки приложения могут включать недавно использовавшиеся меню, предпочтения пользователя и конфигурацию по умолчанию. Сведения об установленном оборудовании включают информацию об устройствах, подключенных к компьютеру в текущий момент времени, о том, какие драйверы обслуживают каждое устройство и какие ресурсы (например, аппаратные порты) связаны с тем или иным

устройством. Кроме того, в реестре хранится системная информация, например, ассоциации файлов (т.е. связь зарегистрированных типов файлов с определенными приложениями для их обработки, например, .docx с Microsoft Word). Приложения, драйверы устройств и сама система Windows используют реестр для централизованного хранения и чтения данных.

Логически данные реестра делятся на две основные группы:

- конфигурационные данные аппаратных средств и программного обеспечения локального компьютера, сохраняемые в разделе (ветви) HKEY_LOCAL_MACHINE (сохраняется на диске: Системный каталог\System32\Config).
- конфигурационные данные, специфичные для конкретного пользователя. Эти настройки сохраняются в разделе (ветви) HKEY_USERS (сохраняется на диске: Системный каталог\System32\Config). Пользователь может менять их в процессе сеанса, создавая для себя удобную рабочую среду.

Основные ветви (разделы) реестра и их содержание указаны далее.

HKEY LOCAL MACHINE (HLM) – содержит все данные о конфигурации локального компьютера. Их используют приложения, драйверы устройств и Windows. Часть данных используется при запуске Windows. Данные в этой ветви определяют, какие драйверы устройств и службы должны быть загружены при запуске. Информация в этой ветви практически не меняется и не зависит от пользователя.

HKEY USERS (HU) – в этом разделе 2 и более подразделов:

DEFAULT – принимаемые по умолчанию параметры профиля пользователя. Используются как часть скрытой папки системный_диск:\Documents and Settings\Default User (файл NTUSER.DAT) при начальной загрузке общего профиля и при первом входе пользователя на компьютер для создания нового локального профиля;

SID – имя подраздела пользователя, совпадающее с его идентификатором защиты. Здесь содержится специфическая информация пользователя, зарегистрировавшегося на компьютере (реестровая часть профиля пользователя).

HKEY CURRENT USER (HCU) – этот подраздел указывает на те же самые данные профиля пользователя, зарегистрировавшегося на компьютере, доступ к которым может быть осуществлен через HKEY_USERS\SID_зарегистрировавшегося_пользователя. Копия части реестра для пользователя, когда-либо работавшего на этом компьютере, хранится в папке системный_диск:\Documents and Settings\имя_пользователя в файле NTUSER.DAT

HKEY CLASSES ROOT (HCR) – содержит информацию о приложениях, обрабатывающих файлы с определенными расширениями, и данные, ассоциированные с объектами COM. Эта ветвь указывает на подраздел CLASSES в разделе HLM\SOFTWARE.

HKEY CURRENT CONFIG (HCC) – содержит сведения о текущей конфигурации программ и устройств компьютера, т.е. о текущем (активном) профиле оборудования.

Иерархическая структура реестра организована подобно иерархической структуре папок и файлов на диске. Для просмотра и редактирования реестра Windows служит Редактор реестра (Registry Editor), regedit.exe.

Панль управления (Control Panel), оснастки управления администратора и Редактор системных политик (System Policy Editor) тоже меняют содержимое реестра. Удобный интерфейс редактора помогает корректно настраивать систему.

11.1.8.2. Консоли управления

Консоль MMC (Microsoft Management Console) предлагает стандартизованный графически интерфейс пользователя для управления системными службами. Сама консоль MMC не выполняет каких-либо функций управления, но она служит общей средой для подключаемых модулей (Snap-In или «оснастка» в русском переводе), которые создаются корпорацией Microsoft и независимыми производителями программного обеспечения для реализации функций управления. С помощью консоли MMC можно управлять многими важными службами, включая:

- Internet Information Server;
- Transaction Server;
- SQL Server и пр.

Реализация консоли в Windows содержит различные оснастки, предназначенные для выполнения типичных задач управления.

Computer Management (управление компьютером) инструментальное средство администратора для настройки конфигурации компьютера. Оно предназначено для работы с одним компьютером, но все его функции можно инициировать с удаленного компьютера.

Disk Management (диспетчер дисков) графическое средство управления дисками, заменяющее программу Disk Administrator (администратор дисков) из предыдущих версий Windows NT. Поддерживаются разделы, логические диски и новые динамические тома.

System Service Management (управление системными службами) позволяет администраторам останавливать, запускать, приостанавливать и возобновлять службы на локальном и удаленном компьютерах. В модуле поддерживается мониторинг служб, позволяющий Windows автоматически перезапускать службы, запускать сценарии или exe-файлы, а также перезагружать сервер в случае сбоя важной службы.

Device Manager (диспетчер устройств) и Hardware Wizard (мастер установки оборудования) образуют оснастку, позволяющую администраторам настраивать конфигурацию устройств и ресурсов в системе.

11.2.8.3. Диспетчер объектов

В Windows физические (например, периферийные устройства) и логические (например, процессы и потоки) ресурсы представляются в виде объектов. Для представления объектов в Windows используются структуры, данных в памяти, в которых хранятся атрибуты и процедуры объекта. Каждый объект в Windows принадлежит к определенному типу, причем тип объекта также хранится в виде структуры данных. Типы объектов создаются с помощью исполняющих программ. Например, диспетчер ввода-вывода определяет тип файлового объекта. К типам объекта в Windows относятся файлы, устройства, процессы, потоки, каналы, семафоры и т.д. Процессы, исполняемые в пользовательском режиме, и компоненты ядра взаимодействуют с объектами при помощи дескрипторов (object handles). Дескриптор объекта представляет собой структуру данных, которая дает возможность процессам манипулировать объектами. Чтобы создать объект, сначала процесс передает запрос диспетчеру объектов, который инициализирует объект и возвращает дескриптор. Для каждого объекта диспетчер объектов ведет учет количества дескрипторов, которые ссылаются на этот объект и количества существующих ссылок на объект. Когда количество дескрипторов становится равным нулю, диспетчер объектов удаляет объект из пространства имен диспетчера объектов. Когда количество ссылок на объект снижается нуля, диспетчер объектов удаляет сам объект.

11.2.8.4. Интерфейс

Windows Aero (бэкроним англ. authentic, energetic, reflective, open – аутентичный, энергичный, отражающий, открытый) – комплекс технических решений графического пользовательского интерфейса. Некоторые возможности Windows Aero.

Панель задач («супербар»). Можно закреплять любые ярлыки непосредственно на панели задач, а чтобы воспользоваться панелью управления Windows Media Player необходимо просто открыть панель миниатюр (Aero Peek). Для этого надо навести на иконку плеера расположенную в новой панели задач. Всплывающие списки (Jump List) предоставляют доступ к недавно использованным файлам и действиям, связанным с конкретными приложениями. Еще одна дополнительная возможность: при копировании, перемещении или загрузке, прогресс выполнения отображается непосредственно на панели задач.

AeroSnap (оснастка аэро). Позволяет менять размер окна перетаскиванием его к левой или правой, а также к верхней границе экрана. При перетаскивании в левую или правую сторону окно приложения разворачивается по всей длине и при этом занимает половину ширины экрана.

Shake (встряхивание). Данная функция, заимствованная у Apple, позволяет свернуть все неактивные окна простым движением мыши. Достаточно просто захватить заголовок окна и немного потрясти.

Desktop SlideShow. Позволяет менять обои рабочего стола через стандартные промежутки времени от 10 с до 1 дня.

Библиотеки – это виртуальные папки, объединяющие несколько физических папок схожей тематики. Если объединить разрозненные папки одной тематики, то при открытии библиотеки, пропадает необходимость поиска необходимого файла, так как все документы будут отображены в одной библиотеке.

11.2.8.5. Стандартные приложения и компоненты

HomeGroup – сеть без централизованной системы управления с единой системой аутентификации. Она позволяет предоставлять доступ к определенным папкам, которые задаются самим пользователем. При этом отпадает необходимость обслуживать ресурсы, достаточно просто сгенерировать общий пароль и любой пользователь, сможет прозрачно получать доступ к необходимым папкам.

Federated Search («федеративный поиск»). Позволяет производить поиск не только на локальной машине, но также и по локальной сети, серверам SharePoint, а также по сети Интернет.

Device Stage – это домашняя страница для оборудования. При подключении устройства к компьютеру появится меню часто выполняемых задач для этого типа устройства. Например, при подключении многофункционального принтера будут отображены параметры печати и сканирования.

Action Center (центр поддержки) является центральным местом для различного рода системных сообщений и отправной точкой для проведения диагностики и устранения проблем. С помощью центра открываются такие настройки как (в скобках названия русской версии): Security Center (Безопасность), Problem, Reports and Solutions (Поиск решений для незарегистрированных проблем), Windows Defender (Защитник Windows), Windows Update (Центр обновления Windows), Network Access Protection (Центр управления сетями и общим доступом), Backup and Restore (Настройка архивации).

Credential Manager (диспетчер учетных данных) – единая система хранения паролей и сертификатов для доступа как к локальным, так и глобальным сетевым ресурсам.

Поддержка VDI и VHD-образов. В Windows добавлена встроенная поддержка образов виртуальных машин формата VHD: оффлайн обслуживание VHD-файлов, установка/удаление обновлений, языковых пакетов и иных компонентов ОС, возможность загрузки с VHD-файлов. Теперь администраторы могут использовать один и тот же мастер-образ для удаленных клиентов, использующих инфраструктуру VDI (Virtual Desktop Infrastructure), и традиционных настольных компьютеров. Это позволит избавиться от необходимости управления многочисленными образами системы и упростит переход к виртуализированному окружению.

Windows PowerShell 2.0. Обновился язык сценариев, построенный на базе технологии .Net. Теперь PowerShell являет необходимым компонентом системы. Среди новых возможностей можно отметить:

- удаленное управление (позволяет выполнять команды параллельно на нескольких компьютерах или последовательно на определенном компьютере);
- ограниченная оболочка (позволяет настроить ограничения на любые наборы команд, их выполнение будет возможно только при наличии необходимых прав);
- автоматизация групповых политик (можно использовать сценарии для управления объектами групповых политик и создавать новые групповые политики);
- расширенные сценарии авторизации (позволяет создавать простые или сложный сценарии при входе\выходе из системы и при выключении компьютера);

BitLocker To Go (система шифрования логических и системных дисков, а также съемных USB-накопителей). Является логическим продолжением возможности шифрования системных и логических дисков. После шифрования всей информации, расположенной на съемном устройстве, для того, чтобы получить к ней доступ необходимо будет ввести пароль или воспользоваться смарт-картой. Все необходимые настройки редактируются с помощью групповых политик.

AppLocker («блокировщик приложений»). Является расширением Software Restriction Policies (SRP) позволяет запрещать или разрешать запуск определенных приложений. В AppLocker появилась возможность блокировки исполняемых файлов, основываясь на их цифровой подписи. В результате можно блокировать программы на основе имени программы, версии и вендоре, пользователе, группе пользователей, пути к файлу, его размере, и т.д. Например, администратору разрешить запускать все приложения, а обычным пользователям только исполняемые файлы, находящиеся в папке Program Files. Позволяет повысить безопасность без использования сторонних приложений.

Branch Cache («кэширование филиалов»). Данная технология работает в связке десктопной версии Windows и сетевой Windows Server. Она позволяет кэшировать информацию, скачиваемую из головного офиса в филиал. Существует два режима работы:

- Hosted Cache – кэшируются все файлы, находящиеся на сервере в филиале.
- Distributed Cache Mode – наличие сервера не требуется, а информация хранится на клиентских компьютерах и при необходимости может быть перенаправлена на другие компьютеры.

Direct Access. Данная технология так же работает в связке десктопной версии Windows и сетевой Windows Server. Суть этой технологии сводится к упрощению подключения удаленных пользователей к корпоративной сети организации при наличии сети Интернет и без использования

VPN. При активации этой возможности системные администраторы могут обслуживать удаленный компьютер все время, пока он находится в сети Интернет.

DNSSEC. Поддерживает набор расширений, предназначенных для обеспечения безопасности протокола DNS. DNSSEC может проверять авторские полномочия, целостность данных и подлинность отрицания существования. Благодаря DNSSEC обеспечить безопасность DNS-серверов стало проще, так как DNS-ответы защищены цифровыми подписями.

Динамическое распределение драйверов. Позволяет централизованно хранить драйверы на сервере установки Windows (Windows Deployment Service). Также появилась возможность динамической установки только необходимых драйверов.

Multicast Multiple Stream Transfer («многоадресная передача нескольких потоков»). Multicast позволяет развертывать трансляцию образов с сервера на несколько клиентов одновременно. Данная технология позволяет ограничить полосу пропускания, тем самым снижая нагрузку на сеть.

Problem Steps Recorder (по-русски: «средство записи действий по воспроизведению неполадок»). Эта утилита является инструментом для осуществления захвата скриншотов (снимков экрана) с последующим автоматизированным составлением отчета. Приложение документирует все щелчки мыши, нажатия клавиш клавиатуры и собирает кое-какой технический материал, а затем сводит все это в отчет в формате MHTML. Сохраненный отчет можно отправить в службу поддержки.

Windows Recovery Environment (параметры восстановления системы). Если не удается запустить ОС, можно попробовать восстановить систему, используя различные возможности Windows Recovery Environment. Некоторые проблемы можно решить автоматически с помощью мастера восстановления.

Большое количество сделанных нововведений в новых версиях Windows направлены на облегчение работы с компьютером и его администрирование, позволяющие такому же количеству администраторов обслуживать большее количество компьютеров, увеличилась стабильность работы компьютеров и их надежность.

11.2. Основные особенности Windows 10

ОС Windows 10, выпущенная компанией Microsoft в 2015 г., должна стать последней «классической» ОС, устанавливаемой непосредственно на устройство. То есть, предполагается, что номера 11 у Windows уже не будет. Более поздние сборки имеют название версий и не оформляются пока в service pack, как раньше. Серверная ОС, соответствующая Windows 10 – Windows Server 2016. Следующая ОС, по заявлению Microsoft, полностью превратится из программы в облачный сервис.

Основная идея появления Windows 10 – это создание единой платформы для всех поддерживающих ее устройств: персональных компьюте-

ров, ноутбуков, нетбуков, смартфонов и т.д. Кроме того, новая система должна будет управлять элементами Интернета вещей. Система предлагается в нескольких редакциях, которые можно рассматривать как базовые (основные) и производные от базовых. Базовые редакции для ПК, ноутбуков, рабочих станций:

- Windows 10 «Домашняя» (англ. Home) – базовая версия для пользователей ПК, ноутбуков и планшетных компьютеров. Поставляется с ноутбуками и нетбуками;
- Windows 10 Pro – версия для ПК, ноутбуков и планшетов с функциями для малого бизнеса типа CYOD (Choose Your Own Device – дословный перевод: «выбери свое устройство»). В соответствии с этой концепцией предприятие предоставляет своим сотрудникам те устройства, которые оно само приобрело, с уже оформленным договором на оказание услуг связи. Сотрудник при этом может выбрать из предложенного ассортимента мобильных телефонов, смартфонов или планшетных ПК тот аппарат, который лучше всего соответствует его рабочим задачам и личным предпочтениям;
- Windows 10 «Корпоративная» (англ. Enterprise) – версия для более крупного бизнеса с расширенными функциями управления корпоративными ресурсами, безопасности и т. д.

Интерфейс Windows 10 заимствован у ставшей классической Windows 7, хотя и сохранил незначительную часть плиток от Windows 8. Обновлены меню Пуск, Панель задач, экран приветствия, часы, календарь, анимация окон, окно командной строки, некоторые значки. Внесены некоторые изменения в ряд приложения, которые не носят принципиального характера. В архитектурном плане Windows 10 в общем соответствует Windows 7.

11.3. Сетевая операционная система Windows Server 2008R2

Версия Windows Server 2008 R2, соответствующая десктопной ОС Windows 7, стала доступной на рынке летом 2009 г. и представляет собой седьмое поколение операционной системы Windows Server. Поставляется только в 64-разрядной версии (x64 или IA64). Оборудование с 32-разрядной архитектурой и установка в 32-разрядном не поддерживается. Последней версией, которая поддерживала 32-разрядную установку, является Windows Server 2008.

11.3.1. Редакции операционной системы

К числу главных редакций Windows Server 2008 R2 относятся: Windows Server 2008 R2 Standard Edition, Windows Server 2008 R2 Enterprise Edition, Windows Server 2008 R2 Datacenter Edition; Windows Web Server 2008 R2 и Windows Server 2008 R2 Server Core.

Редакция Windows Server 2008 R2 Standard Edition является самой распространенной и считается стандартной для развертывания в организациях. Поддерживает до четырех 64-разрядных процессорных сокетов и 32 Гбайт памяти, а также все доступные в Windows Server 2008 R2 роли сервера, кроме роли кластеризации, межфайловой репликации (обеспечиваемой технологией DFS-R) и федеративных служб Active Directory (Active Directory Federation Services).

Редакция Windows Server 2008 R2 Enterprise Edition больше ориентирована на серверные системы, требующие масштабных возможностей по обработке и хранению данных, а также кластеризации или федеративных служб Active Directory (Active Directory Federation Services).

Редакция Windows Server 2008 R2 Datacenter Edition представляет собой мощную версию операционной системы класса центра обработки данных (ЦОД), которая поддерживает крупномасштабные серверные операции для создания и обслуживания крупного централизованного хранилища данных в одном или нескольких серверных кластерах.

Редакция Windows Web Server 2008 R2 Edition представляет собой версию операционной системы класса интерфейсного веб-сервера и ориентирована на удовлетворение тех потребностей сервера приложений, которые касаются предоставления веб-служб.

Редакция Windows Server 2008 R2 Server Core представляет собой версию ОС Windows Server 2008 R2, не имеющую графического пользовательского интерфейса. Вместо него используется режим командной строки DOS. Server Core прекрасно подходит для служебных серверов, таких как контроллеры доменов, серверы DHCP, серверы DNS, благодаря тому, что из-за меньшего количества накладных расходов функционирующие на сервере приложения получают больше ресурсов, а из-за отсутствия графического пользовательского интерфейса и ассоциируемых с ним приложений значительно снижается вероятность нарушения безопасности.

11.3.2. Роли операционной системы

С помощью соответствующего Мастера можно установить на Windows Server 2008 R2 семнадцать ролей. Отметим важнейшие из них:

1. DHCP-сервер – позволяет автоматически назначать IP-адреса и другие параметры стека протоколов TCP/IP компьютерам и другим устройствам, которые могут функционировать как DHCP клиенты;
2. DNS-сервер – функционирует в пространстве системы доменных имен (Domain Name System – DNS) и предоставляет стандартный метод разрешения символьных имен устройств в IP-адреса;
3. Hyper-V – предоставляет службы для создания виртуальных машин (ВМ) и управления ими и их ресурсами. Каждая ВМ представляет собой виртуализованную компьютерную систему, кото-

рая функционирует в изолированной среде. Это позволяет одновременно работать с несколькими ОС;

4. Веб-сервер (IIS) – предоставляет надежную, управляемую и масштабируемую инфраструктуру веб-приложения;
5. Доменные службы Active Directory – с помощью контроллеров домена службы AD DS предоставляют сетевым пользователям доступ к разрешенным ресурсам в любом месте сети посредством единственного входа в систему;
6. Сервер приложений: предоставляет комплексное решение для хостинга и управления высокопроизводительными распределенными бизнес приложениями, например – .NET Framework, поддержка Web-сервера, поддержка отказоустойчивых кластеров и др.;
7. Службы удаленных рабочих столов – позволяют пользователям работать с приложениями Windows, установленными на сервере узла сеансов удаленных рабочих столов, а также обеспечивают доступ к полнофункциональному рабочему столу Windows;
8. Службы управления правами Active Directory – технология защиты информации, которая работает с приложениями, поддерживающими эту функцию, и помогает защитить информацию от несанкционированного использования.

11.3.3. Серверные функции операционной системы

Система Windows Server 2008 R2 соответствует десктопной ОС Windows 7. Поэтому, описывая далее возможности Windows Server 2008 R2, создаваемые, в том числе, путем установки соответствующих служб и приложений, будем останавливаться только на серверных функциях этой системы.

Active Directory. В Windows Server 2008 компонент Active Directory был переименован в Active Directory Domain Services (Доменные службы Active Directory), сокращенно AD DS, и в Windows Server 2008 R2 продолжает использоваться именно это название.

Корзина Active Directory (Active Directory Recycle Bin). Корзина AD предоставляет администраторам возможность отменять удаление объектов в Active Directory. Корзина AD позволяет администратору запустить утилиту восстановления и отменить случайное удаление объектов.

Управляемые учетные записи служб (Managed Service Accounts). При работе в режиме Active Directory R2 учетные записи служб можно идентифицировать и затем управлять ими так, чтобы изменение пароля для любой из них автоматически приводило к внесению соответствующих изменений на всех остальных серверах приложений в организации.

Технология виртуализации Hyper-V была встроена в ядро в Windows Server 2008 и расширена в Windows Server 2008 R2 и позволяет значительно улучшать производительность и возможности виртуализации сервера в среде Windows. Технология Hyper-V создает между абстрактным уровнем

оборудования системы и уровнем ОС еще один очень тонкий уровень, который позволяет гостевым сеансам в виртуальной среде взаимодействовать с уровнем оборудования системы напрямую гораздо быстрее, чем раньше, и гораздо стабильнее. Технология виртуализации поддерживает 32- и 64-разрядные гостевые сеансы, функцию «живой миграции» (Live Migration), которая позволяет быстрее обрабатывать сбои в виртуальном гостевом сеансе и восстанавливать его среди хост-серверов.

Технология сжатия файловых запросов Server Message Block 2.0 (SMB2) сжимает файловые запросы и посредством имеющего больший размер буфера таких запросов сокращает количество циклических проходов, необходимых при передаче данных между системами. Информация копируется в 10-30 раз быстрее.

Парковка ядер. В Windows Server 2008 R2 улучшена технология управления энергопотреблением под названием «парковка ядер» (core parking). В случае применения технологии парковки ядер серверы с новейшими процессорами, распознающими протоколы парковки ядер, будут отключать ядра в системе при отсутствии необходимости в их использовании.

Использование технологии BitLocker для защиты сервера. Технология BitLocker позволяет шифровать весь том сервера Windows Server 2008 R2 и является необходимым компонентом для организаций, беспокоящихся о возможной физической краже данных с сервера.

Windows SharePoint Services (WSS) – приложение для управления хранилищем документов, которое позволяет организациям более легко управлять, организовывать и обмениваться документами, а командам пользователям совместно работать над информацией.

Windows Rights Management Services (RMS) создает инфраструктуру для безопасного обмена информацией за счет шифрования содержимого и установки для него политики, защищающей файл и хранящуюся в нем информацию. Шифрование содержимого самого файла с секретной информацией, даже если он сохраняется в неправильном месте, делает невозможным его открытие и получение доступа к находящейся в нем информации без наличия соответствующих прав.

11.4. Основные особенности сетевой операционной системы Windows Server 2012 R2

Операционная система Windows Server 2012 R2, появившаяся на рынке серверных систем и быстро сменившая ОС Windows Server 2012, сумела закрепиться дольше и успешнее, чем соответствующая ей десктопная система Windows 8.1. По крайней мере, до появления ОС Windows Server 2016, которая рассматривается далее.

Windows Server 2012 R2 предлагается в четырех редакциях: Standard, Datacenter, Essentials и Foundation. Первые две отличаются только правами на средства виртуализации, а редакции Essentials и Foundation – это реше-

ние для небольших корпоративных систем, где число пользователей не превышает 25, а число устройств не превышает 50.

Windows Server 2012 R2 ориентирована на ЦОД и обеспечивает преимущества в четырех ключевых областях.

1. Обеспечивает позволяет построить динамичную многопользовательскую инфраструктуру, которая позволяет создать полноценную платформу для построения частного облака.
2. Обеспечивает производительность множества серверов и позволяет построить удобную в управлении многосерверную платформу, не вкладывая при этом значительных средств.
3. Многофункциональная, масштабируемая веб-платформа и платформа приложений, которая позволяет гибко строить и развертывать приложения в локальном режиме, в режиме облака и в гибридном режиме, используя последовательный набор инструментов и архитектур.
4. Предоставляет пользователям гибкий доступ к данным и приложениям из любого расположения и на любом устройстве, упрощая при этом управление и обеспечивая безопасность.

ОС позволяет клиентам перевести работу ЦОД на качественно новый уровень. При сохранении в целом функционала ОС Windows Server 2008R2, в Server 2012 и в Server 2012 R2 были внесены значительные и изменения. Рассмотрим важнейшие из них.

В систему добавлены три новые роли:

1. Режим Windows Server Essentials – настраивает компьютерную инфраструктуру и предоставляет удобные и эффективные функции: резервное копирование, удаленный веб-доступ.
2. Службы активации корпоративных лицензий – автоматизация и упрощение управления ключами узлов службы управления ключами (KMS) инфраструктурой активации ключей для сети. С помощью этих служб можно установить узел KMS и управлять им либо настроить активацию корпоративных лицензий для присоединенных к домену систем на основе Active Directory.
3. Удаленный доступ – обеспечивает легкое подключение через Direct Access, VPN и прокси веб-приложения. RAS предоставляет традиционные службы VPN, включая подключения типа «сеть-сеть» (для филиала или облака). Служба маршрутизации обеспечивает традиционные возможности, включая преобразование сетевых адресов (NAT).

Были сделаны улучшения в имевшихся ролях и компонентах:

- в службу Active Directory внедрен механизм утверждений. В предыдущих ОС решение об авторизации в основном принимались для каждого пользователя или на основе членства в группе в AD. Использование «утверждений» в AD в Windows Server 2012 позволило расширить возможности управления. На основе атрибутов пользователей и устройств в рамках каталога создаются

утверждения, которые становятся частью маркера, передаваемого в источники авторизации. Для службы динамического управления доступом – это будет файловый сервер. Решения в области доступа и авторизации можно принимать с учетом значений свойств в службе Active Directory;

- усовершенствован гипервизор Hyper-V 3.0 с расширенными возможностями для интеграции с облачными сервисами. В него добавлена новая функция: Replica, с помощью которой администраторы и IT-работники компаний могут поддерживать самые новые копии виртуальных машин на других вычислительных площадках;
- обновлена компонента Windows PowerShell версии 4, включающая в себя функцию Desired State Configuration («настройка требуемого состояния»). PowerShell Desired State Configuration – это очень мощный инструмент, который может значительно облегчить администратору процесс настройки системы;
- предлагаются два поколения виртуальных машин на выбор при создании новой виртуальной машины:
    - поколение 1. Предоставляет виртуальной машине такое же виртуальное оборудование, как в предыдущих версиях Hyper-V;
    - поколение 2. Предоставляет виртуальной машине следующие новые функциональные возможности:
        - PXE-загрузка с помощью стандартного сетевого адаптера;
        - загрузка с виртуального жесткого диска SCSI;
        - загрузка с виртуального DVD-диска SCSI;
        - безопасная загрузка (включена по умолчанию);
        - поддержка встроенного ПО UEFI.
- функция автоматического шифрования конфиденциальных файлов Microsoft Office на основе их классификации. Это делается с помощью задач управления файлами, которые вызывают защиту AD RMS (Rights Management Services, RMS, служба управления правами) для конфиденциальных файлов Office, спустя несколько секунд после того, как файл идентифицируется как конфиденциальный на файловом сервере. Шифрование RMS обеспечивает еще один уровень защиты для файлов. Пользователь, который хочет получить доступ к файлу, должен сначала пройти аутентификацию на сервере RMS, чтобы получить ключ дешифрования. Этот сценарий требует, чтобы перед этим была развернута реализация защиты RMS;
- встроена поддержка Microsoft Office 365 (в редакции Essentials);
- изменен интерфейс аналогично Windows 8.1.

11.5. Основные особенности сетевой операционной системы Windows Server 2016

Windows Server 2016 – ОС, готовая к использованию в облачной среде, обеспечивающая качественно новый уровень защищенности и новые возможности для повышения эффективности работы приложений и инфраструктуры.

Все новые и усовершенствованные функции направлены на то, чтобы ОС Windows Server 2016 стала ведущей платформой по следующим параметрам:

- как платформа приложений, способная не только поддерживать работу традиционных приложений, но и предоставлять необходимую структуру для подготовки приложений к переносу в облако;
- по возможностям программно-определяемых ЦОД. Эти возможности появились в Microsoft Azure, а теперь доступны и в локальной среде;
- по требованиям безопасности.

Рассмотрим важнейшие из этих возможностей.

Платформа приложений. В Windows Server 2016 поддерживаются новые способы развертывания и запуска приложений как в локальной среде, так и в Azure. Среди новых возможностей – контейнеры Windows и вариант сверхкомпактного развертывания Nano Server.

Контейнеры в Windows Server 2016 – обладают высокой гибкостью и плотностью на уровне, который требуется современным облачным приложениям. Поддержка контейнеров в Windows Server позволяет полноценно использовать их в экосистеме Windows и в некоторых случаях задействовать для работы важных приложений контейнеры Hyper-V с дополнительным уровнем изоляции, причем без написания дополнительного программного кода.

Вариант развертывания Nano Server обеспечивает высокую гибкость, которая требуется разработчикам для запуска приложений в контейнерах или с помощью микрослужб. Снижение объема используемых ресурсов ЦОД и обеспечение высокой доступности за счет установки минимального набора необходимых компонентов ОС. Вариант развертывания, который называется Nano Server, занимает в 25 раз меньше дискового пространства, чем полноценный Windows Server с графической оболочкой.

Переход к сервисной модели. Используя Nano Server, можно чаще обновлять ОС, получая больше новых возможностей в течение ее жизненного цикла.

Программно-определяемые вычислительные решения. Минимизация областей атак, повышение доступности и снижение использования ресурсов достигается за счет установки только необходимых компонентов при развертывании Nano Server. Такой вариант развертывания занимает в 25 раз меньше дискового пространства, чем полная установка Windows Server.

Упрощенное перемещение в облако из Microsoft Hyper-V. Развертывание приложений на множестве ОС, включая поддержку операционной системы Linux на Hyper-V.

Обновление кластеров инфраструктуры до Windows Server 2016 возможно без простоев в работе приложений и рабочих нагрузок и без необходимости приобретать новое оборудование. Эти возможности поддерживаются благодаря поддержке работы кластеров в смешанном режиме (режим, при котором кластерные узлы могут выполняться как на Windows Server 2012 R2, так и на Windows Server 2016).

Повышение доступности приложений достигается за счет улучшенной устойчивости кластера к временным сбоям сетей и систем хранения данных.

Автоматизация управления серверами реализована с помощью встроенных средств, таких как Desired State Configuration и Windows PowerShell 5.0.

Управление серверами с Windows может выполняться из любого места с помощью средства управления серверами в виде службы Azure с веб-интерфейсом. Это особенно полезно для серверов Nano Server или Server Core.

Программно-определяемые системы хранения данных. Создание масштабируемых программно-определяемых СХД реализуется при существенно меньших затратах, чем при использовании классических сетей хранения данных (SAN) или сетевых хранилищ данных (NAS). С помощью Storage Spaces Direct можно создавать гиперконвергентные (интегрированные в одно целое два и более разнородные компоненты) архитектуры хранения данных, используя локальные хранилища обычных серверов.

Синхронная и асинхронная репликация хранилища (Storage Replica) позволяет создавать доступные решения для обеспечения непрерывности бизнес-процессов и аварийного восстановления в центрах данных.

Предоставление важнейшим бизнес-приложениям приоритетного доступа к ресурсам хранения данных выполняется путем использования возможностей по обеспечению требуемого уровня качества обслуживания систем хранения (QoS).

Программно-определяемые сети. Развертывание сложных рабочих нагрузок с сотнями сетевых политик (изоляция, качество обслуживания, безопасность, балансировка нагрузки, коммутация, маршрутизация, шлюзы, DNS и т. д.) за несколько секунд с помощью масштабируемого сетевого контроллера, аналогично тому, как это делается в Azure.

Динамическая сегментация сети на основе потребностей рабочих нагрузок с помощью распределенного брандмауэра и групп сетевой безопасности с возможностью применения комплексных политик как внутри отдельных сегментов, так и между сегментами.

Более высокая доступность служб за счет устойчивого горизонтального и вертикального масштабирования программными средствами как

инфраструктуры (узел, программная система балансировки нагрузки, шлюз, сетевой контроллер), так и рабочих нагрузок.

Полный контроль над гибридными рабочими нагрузками, включая запуск их в контейнерах, перемещение между серверами, серверными стойками и облаками с использованием виртуальных сетей на основе VXLAN (Virtual Extensible LAN – технологии сетевой виртуализации) и NVGRE (Network Virtualization using Generic Routing Encapsulation – технология виртуализация сети с помощью универсальной инкапсуляции маршрутизации).

Оптимизация затрат и производительности при объединении удаленного прямого доступа к памяти (RDMA) и клиентского трафика на одних и тех же объединенных сетевых адаптерах. Такой подход позволяет снизить расходы и при этом добиться требуемой гарантированной пропускной способности сети 40 Гбит/с и более.

Безопасность. Windows Server 2016 позволяет предотвращать атаки и выявлять подозрительные действия с помощью новых возможностей для управления привилегированным доступом, защиты виртуальных машин и повышения устойчивости платформы к новым угрозам. В Windows Server 2016 доступны следующие возможности:

Предотвращение риска, связанного с утечкой учетных данных системных администраторов. Используя новые средства управления привилегированными удостоверениями, можно ограничить область доступа и время доступа. Кроме того, с помощью Credential Guard можно предотвратить кражу учетных данных администратора путем проведения атаки, основанной на получении хэша (Pass-the-Hash).

Защита виртуальных машин от несанкционированных действий администраторов серверной структуры (за счет использования экранированных виртуальных машин). Экранированная виртуальная машина – это виртуальная машина второго поколения, имеющая модуль Trusted Platform Module (TPM) и зашифрованная с помощью BitLocker. Такая виртуальная машина может быть запущена только на одобренных узлах серверной структуры.

Дополнительная защита всех развернутых экземпляров Windows Server 2016. При работе в облаке и локальной среде есть возможность использовать дополнительные средства безопасности, такие как проверка целостности кода (Core Integrity) и защита потока управления (Control Flow Guard). При этом возможен запуск только разрешенных двоичных файлов и обеспечивается защита от неизвестных уязвимостей.

Обнаружение вредоносного поведения с помощью расширенного аудита безопасности. Новые категории аудита касаются членства в группах и PnP. Они дают возможность выявлять события и получать о них более подробную информацию, благодаря чему администраторы могут полнее анализировать работу систем с целью обнаружения новых угроз.

Защита от вредоносных программ с помощью встроенного антивирусного решения. В состав Windows Server 2016 входит Windows Defender,

который оптимизирован для поддержки различных серверных ролей и интегрирован с Windows PowerShell с целью проверки наличия вредоносного ПО.

Ограничение затрагиваемой области в случае вторжения. В случае нарушения системы безопасности система Windows Server 2016 может ограничить затрагиваемую область путем сегментации сети на основе рабочих нагрузок или потребностей бизнеса. При этом используются распределенный брандмауэр и группы сетевой безопасности. Можно применять сложные политики как внутри каждого сегмента, так и между сегментами.

Использование контейнеров Hyper-V для получения уникального дополнительного уровня изоляции приложений в контейнерах без изменения содержимого самого контейнера. Контейнеры Hyper-V обеспечивают изоляцию на аппаратном уровне, при этом задействуются такие же методы изоляции, как при защите виртуализации на базе оборудования.

Поддержка платформы Linux. Microsoft поддерживает службы интеграции с Linux (Linux Integration Services, LIS), чтобы обеспечить максимум возможностей при использовании Linux в Hyper-V. В настоящее время в Hyper-V обеспечивается полная поддержка следующих дистрибутивов: Red Hat Linux, SUSE openSUSE, CentOS, Ubuntu, Debian, Oracle Linux.

12. Сетевая операционная система UNIX

12.1 История создания операционных систем семейства UNIX

Первая система UNIX была разработана в конце 1960-х – начале 1970-х годов в подразделении Bell Labs компании AT&T. С тех пор было создано большое количество различных UNIX-систем. Юридически лишь некоторые из них имеют полное право называться «UNIX»; остальные же, хотя и используют сходные концепции и технологии, объединяются термином «UNIX-подобные» (англ. Unix-like). Для краткости под UNIX-системами будем подразумевать как истинные UNIX, так и UNIX-подобные ОС. Некоторые отличительные признаки UNIX-систем включают в себя:

- использование простых текстовых файлов для настройки и управления системой;
- широкое применение командной строки;
- представление устройств и некоторых средств межпроцессного взаимодействия как файлов;
- использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу.

В настоящее время UNIX используются в основном на серверах, а также как встроенные ОС для различного оборудования. На рынке ОС для рабочих станций и домашнего применения UNIX уступили другим операционным системам, в первую очередь Microsoft Windows. UNIX-системы имеют большую историческую важность, поскольку благодаря им распространились некоторые популярные сегодня концепции и подходы в области ОС и программного обеспечения.

12.1.1. Первые UNIX

В 1957 г. в Bell Labs была начата работа по созданию операционной системы для собственных нужд. Под руководством В. Высотского (русского по происхождению) была создана система BESYS.

В 1964 г. появились компьютеры третьего поколения, для которых возможности BESYS уже не подходили. В. Высотский и его коллеги приняли решение не разрабатывать новую собственную операционную систему, а подключиться к совместному проекту General Electric и Массачусетского технологического института Multics. Телекоммуникационная компания AT&T, в состав которой входил Bell Labs, оказал проекту существенную поддержку, но в 1969 г. вышел из проекта, поскольку он не приносил никаких финансовых выгод.

Первоначально UNIX была разработана в конце 1960-х годов сотрудниками Bell Labs, в 1969 г. К. Томпсон, стремясь реализовать идеи,

что были положены в основу ОС MULTICS, но на более скромном аппаратном обеспечении (DEC PDP-7), написал первую версию новой операционной системы, а Брайан Керниган придумал для неё название – UNICS (UNIplexed Information and Computing System). Позже это название сократилось до UNIX. В 1971 г. вышла версия для PDP-11, наиболее успешного семейства миникомпьютеров 1970-х (в СССР оно известно, как СМ ЭВМ). Эта версия получила название «первая редакция» (Edition 1) и была первой официальной версией. Системное время все реализации UNIX отсчитывают с 1 января 1970 г.

Первые версии UNIX были написаны на ассемблере и не имели встроенного компилятора с языка высокого уровня. Примерно в 1969 г. Кен Томпсон при содействии Дениса Ритчи разработал и реализовал язык Би (B). В 1969-1973 гг. на основе Би был разработан компилируемый язык, получивший название Си (C).

В 1973 г. вышла третья редакция UNIX, со встроенным компилятором с Си. 15 октября того же года появилась четвёртая редакция, с переписанным на Си системным ядром, а в 1975 г. – пятая редакция, полностью переписанным на Си. С 1974 года UNIX стал бесплатно распространяться среди университетов и академических учреждений. С 1975 г. началось появление новых версий, разработанных за пределами Bell Labs, и рост популярности системы.

К 1978 г. система была установлена более чем на 600 машинах, прежде всего, в университетах. Седьмая редакция была последней единой версией UNIX. Именно в ней появился близкий к современному интерпретатор командной строки Bourne shell.

12.1.2. Создание BSD UNIX

С 1978 г. начинает свою историю BSD UNIX, созданный в университете Беркли. Его 1-я версия, была основана на шестой редакции. В 1979 г. выпущена новая версия, названная 3BSD, основанная на 7-ой редакции. BSD поддерживал такие полезные свойства, как виртуальную память и замещение страниц по требованию. Автором BSD был Б. Джой.

В начале 1980-х компания AT&T, которой принадлежали Bell Labs, осознала ценность UNIX и начала создание коммерческой версии UNIX. Эта версия, поступившая в продажу в 1982 г., носила название UNIX System III и была основана на седьмой версии системы.

Важной причиной раскола UNIX стала реализация в 1980 г. стека протоколов TCP/IP. До этого межмашинное взаимодействие в UNIX пребывало в зачаточном состоянии – наиболее существенным способом связи был UUCP (средство копирования файлов из одной UNIX-системы в другую, изначально работавшее по телефонным сетям с помощью модемов).

Было предложено два интерфейса программирования сетевых приложений:

1. Berkley sockets;
2. интерфейс транспортного уровня TLI (англ. Transport Layer Interface).

Интерфейс Berkley sockets был разработан в университете Беркли и использовал стек протоколов TCP/IP, разработанный там же. TLI был создан AT&T в соответствии с определением транспортного уровня модели OSI и впервые появился в системе System V версии 3. Хотя эта версия содержала TLI и потоки, первоначально в ней не было реализации TCP/IP или других сетевых протоколов, но подобные реализации предоставлялись сторонними фирмами. Реализация TCP/IP официально и окончательно была включена в базовую поставку System V версии 4. Это, как и другие соображения (по большей части, рыночные), вызвало окончательное размежевание между двумя ветвями UNIX – BSD (университета Беркли) и System V (коммерческая версия от AT&T). Впоследствии, многие компании, лицензировав System V у AT&T, разработали собственные коммерческие разновидности UNIX, такие, как AIX, HP-UX, IRIX, Solaris.

В середине 1983 г. была выпущена BSD версии 4.2, поддерживающая работу в сетях Ethernet и Arpanet. Система стала весьма популярной. Между 1983 и 1990 годами в BSD были добавлено много новых возможностей, таких как отладчик ядра, сетевая файловая система NFS, виртуальная файловая система VFS, и существенно улучшены возможности работы с файловыми сетями.

Тем временем AT&T выпускала новые версии своей системы, названной System V. В 1983 г. была выпущена версия 1 (SVR1 – System V Release 1), включавшая полноэкранный текстовый редактор vi, библиотеку curses, буферизацию ввода-вывода, кеширование inode. Версия 2 (SVR2), выпущенная в 1984 г., реализовывала монопольный доступ к файлам (file locking), доступ к страницам по требованию (demand paging), копирование при записи (copy-on-write). Версия 3 вышла в 1987 г. и включала, среди прочего, TLI, а также систему поддержки удалённых файловых систем RFS. Версия 4 (SVR 4), разработанная в сотрудничестве с фирмой Sun и вышедшая в 1988 г., поддерживала многие возможности BSD, в частности TCP/IP, сокеты, новый командный интерпретатор csh. Кроме того, там было много других добавлений, таких как символические ссылки, командный интерпретатор ksh, сетевая файловая система NFS (заимствованная у SunOS) и т. д.

Современные реализации UNIX, как правило, не являются системами V или BSD в чистом виде. Они реализуют возможности как System V, так и BSD.

12.1.3. Свободные UNIX-подобные операционные системы

В 1983 г. Р. Столлмэн объявил о создании проекта GNU – попытки создания свободной UNIX-подобной операционной системы с нуля, без использования проприетарного исходного кода. Большая часть программного обеспечения, разработанного в рамках этого проекта, – такого, как GNU toolchain, Glibc (стандартная библиотека языка Си) и Coreutils – играет ключевую роль в других свободных операционных системах. Однако работы по созданию замены для ядра UNIX, необходимые для полного выполнения задач GNU, продвигались крайне медленно.

В 1991 г., когда Линус Торвальдс опубликовал ядро Linux и привлёк помощников, использование инструментов, разработанных в рамках проекта GNU, было очевидным выбором. Операционная система GNU и ядро Linux вместе составляют ОС, известную, как GNU/Linux. Дистрибутивы этой системы (такие как Red Hat и Debian), включающие ядро, утилиты GNU и дополнительное программное обеспечение стали популярными как среди любителей, так и среди представителей бизнеса.

В результате урегулирования юридического дела, возбуждённого UNIX Systems Laboratories против университета Беркли и Berkeley Software Design Inc., было установлено, что университет может распространять BSD UNIX, в том числе и бесплатно. После этого были возобновлены эксперименты, связанные с BSD-версией UNIX. Вскоре разработка дистрибутива BSD была продолжена в нескольких направлениях одновременно, что привело к появлению проектов, известных как FreeBSD, NetBSD, OpenBSD, TrustedBSD и DragonFlyBSD.

В настоящий момент GNU/Linux и представители семейства BSD быстро отвоёвывают рынок у коммерческих UNIX-систем и одновременно проникают как на настольные компьютеры конечных пользователей, так и на мобильные и встраиваемые системы. Одним из свидетельств этого успеха служит тот факт, что, когда фирма Apple искала основу для своей новой ОС, она выбрала NEXTSTEP – операционную систему со свободно распространяемым ядром, разработанную фирмой NeXT и переименованную в Darwin после приобретения фирмой Apple. Данная система относится к семейству BSD и основана на ядре Mach. Применение Darwin BSD UNIX в Mac OS X делает его одной из наиболее широко используемых версий UNIX.

12.1.4. Современное состояние операционных систем семейства UNIX

После разделения компании AT&T, товарный знак UNIX и права на оригинальный исходный код неоднократно меняли владельцев, в частности, длительное время принадлежали компании Novell.

В 1993 г. Novell передала права на товарный знак и на сертификацию программного обеспечения на соответствие этому знаку консорциуму

X/Open, который затем объединился с Open Software Foundation, образовав консорциум The Open Group. Он объединяет ведущие компьютерные корпорации и государственные организации, в том числе IBM, Hewlett-Packard, Sun, NASA и многие другие. Консорциум занимается разработкой открытых стандартов в области операционных систем, самым важным из которых является Single UNIX Specification, ранее известный как POSIX. С точки зрения The Open Group, название UNIX могут носить только системы, прошедшие сертификацию на соответствие Single UNIX Specification.

Эволюция UNIX систем приведена на рис. 12.1.

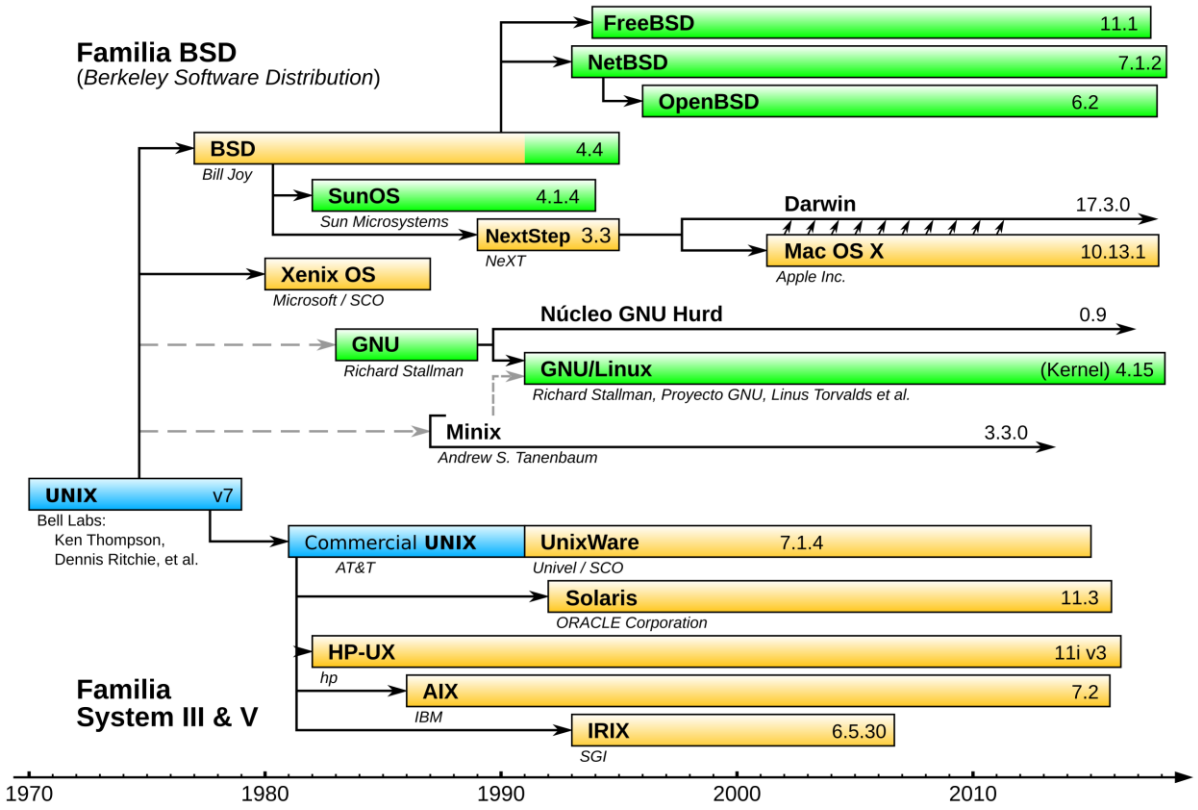


Рис. 12.1. Эволюция операционных систем семейства UNIX

Идеи, заложенные в основу UNIX, оказали огромное влияние на развитие компьютерных операционных систем. В настоящее время UNIX-системы признаны одними из самых исторически важных ОС.

UNIX была написана на языке высокого уровня, а не на ассемблере (доминировавшем в то время). Она содержала значительно упрощённую, по сравнению с современными ей операционными системами, файловую модель. Файловая система включала как службы, так и устройства (такие как принтеры, терминалы и жёсткие диски) и предоставляла внешне единообразный интерфейс к ним, но дополнительные механизмы работы с устройствами (такие как IOCTL и биты доступа) не вписывались в простую модель «поток байтов».

UNIX популяризовала идею иерархической файловой системы с произвольной глубиной вложенности. Другие операционные системы того времени позволяли разбивать дисковое пространство на каталоги или раз-

делы, но число уровней вложенности было фиксировано и, зачастую, уровень вложенности был только один. Позднее все основные фирменные операционные системы обрели возможность создания рекурсивных подкаталогов.

То, что интерпретатор команд стал просто одной из пользовательских программ, а в качестве дополнительных команд выступают отдельные программы, является ещё одной инновацией, популяризированной UNIX. Язык командной оболочки UNIX используется пользователем как для интерактивной работы, так и для написания скриптов, то есть не существует отдельного языка описания заданий. Так как оболочка и команды операционной системы являются обычными программами, пользователь может выбирать их в соответствии со своими предпочтениями, или даже написать собственную оболочку. Наконец, новые команды можно добавлять к системе без перекомпиляции ядра. Новый, предложенный в командной строке UNIX, способ создания цепочек программ, последовательно обрабатывающих данные, способствовал использованию параллельной обработки данных.

Существенными особенностями UNIX были полная ориентация на текстовый ввод-вывод и предположение, что размер машинного слова кратен восьми битам. Ориентация на текстовый 8-битный байт сделала UNIX более масштабируемой и переносимой, чем другие операционные системы. Со временем текстовые приложения одержали победу и в других областях, например, на уровне сетевых протоколов, таких как Telnet, FTP, SMTP, HTTP и других.

UNIX способствовала широкому распространению регулярных выражений, которые были впервые реализованы в текстовом редакторе ed для UNIX. Возможности, предоставляемые UNIX-программам, стали основой стандартных интерфейсов операционных систем POSIX (англ. Portable Operating System Interface – переносимый интерфейс операционных систем).

Широко используемый в системном программировании язык Си, созданный изначально для разработки UNIX, превзошёл UNIX по популярности. Си был первым высокоуровневым языком, предоставляющим доступ ко всем возможностям процессора, таким как ссылки, таблицы, битовые сдвиги, приращения и т. п. Первые разработчики UNIX способствовали внедрению принципов модульного программирования и повторного использования в инженерную практику.

UNIX предоставлял возможность использования протоколов TCP/IP на сравнительно недорогих компьютерах, что привело к быстрому росту Интернета. Это, в свою очередь, способствовало быстрому обнаружению нескольких крупных уязвимостей в системе безопасности, архитектуре и системных утилитах UNIX.

Со временем ведущие разработчики UNIX разработали культурные нормы разработки программного обеспечения, которые стали столь же важны, как и сам UNIX.

12.2. Цели и возможности операционных систем семейства UNIX

Операционная система UNIX проектировалась как инструментальная система для разработки программного обеспечения. Своей уникальностью система обязана во многом тому обстоятельству, что она была, по сути, создана всего двумя разработчиками, причем создававшие ее люди делали систему для себя, и первое время ее использовали на мини-ЭВМ с очень скромными вычислительными ресурсами. По этой причине UNIX, прежде всего, обладает простым, но очень мощным командным языком и независимой от устройств файловой системой. Поскольку при создании этой ОС использовался язык высокого уровня, на котором пишутся не только системные, но и прикладные программы (речь идет о языке С), то система и приложения, выполняющиеся в ней, получились легко переносимыми.

Первой целью при разработке этой системы было стремление сохранить простоту и обойтись минимальным количеством функций. Все реальные сложности оставлялись пользовательским программам.

Второй целью была общность. Одни и те же методы и механизмы должны были использоваться во многих случаях. Поэтому общность в UNIX-системах проявляется во многих аспектах, и в частности:

- обращения к файлам, устройствам ввода/вывода и буферам межпроцессных сообщений выполняются с помощью одних и тех же примитивов;
- одни и те же механизмы именования, присвоения альтернативных имен и защиты от несанкционированного доступа применяются к файлам с данными и директориями и устройствам;
- одни и те же механизмы работают в отношении программно и аппаратно инициируемых прерываний.

Наконец, третья цель заключалась в создании операционной среды, в которой большие задачи можно было бы решать, комбинируя существующие небольшие программы, а не разрабатывая программы заново.

К основным функциям операционной системы UNIX можно отнести следующее.

- Обработка прерываний.
- Создание и уничтожение процессов.
- Переключение процессов из одного состояния в другое.
- Диспетчеризация.
- Приостановка и активизация процессов.
- Синхронизация процессов.
- Организация взаимодействия между процессами.
- Манипулирование блоками управления процессами.
- Поддержка операции ввода-вывода.
- Поддержка операции распределения и перераспределения памяти.
- Поддержка работы файловых систем.

- Поддержка механизма вызова-возврата по обращению к процедурам.

12.3. Типовая архитектура операционной системы семейства UNIX

В архитектуре ОС UNIX можно выделить три основные части (рис. 12.2):

1. *Ядро* – самая низкоуровневая часть ОС. Оно непосредственно взаимодействует с аппаратными средствами и обеспечивает переносимость всего остального ПО на компьютеры с разным аппаратным обеспечением. Ядро предоставляет программам определенный набор системных API, с помощью которых производятся создание процессов, управление ими, их взаимодействие и синхронизация, а также файловый ввод/вывод.
2. *Уровень системных сервисов* (демонов), *конкретных служебных программ и языков программирования*. На этом уровне система получает ресурсы через обращение к ядру ОС (т.е. по прерываниям).
3. *Уровень пользовательских программ, утилит, вспомогательных процедур, интерпретаторов, компиляторов*. На этом уровне функционируют пользовательские приложения (текстовые редакторы, графические интерфейсы и собственно приложения).

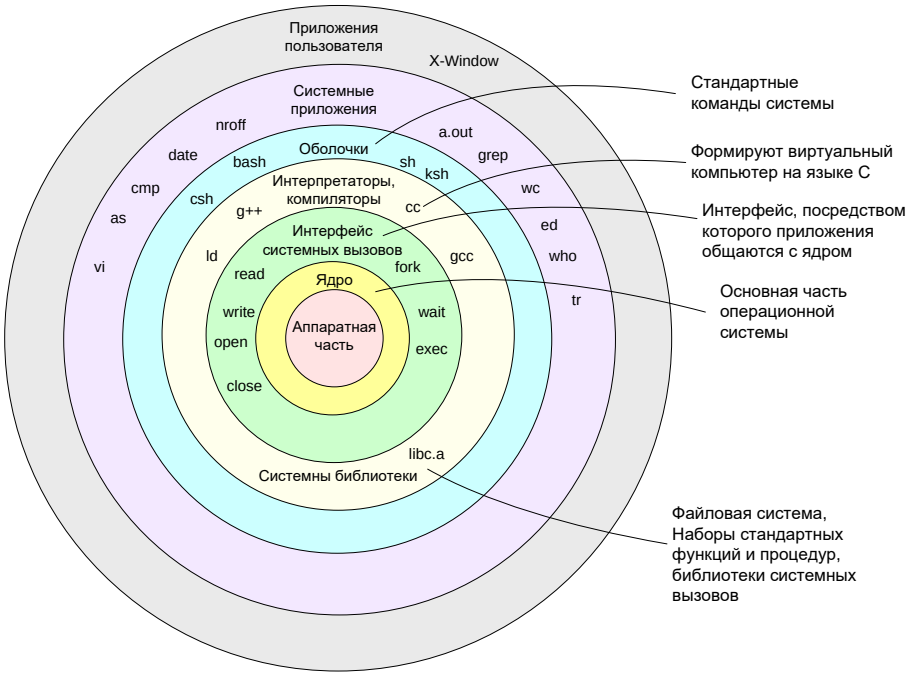


Рис. 12.2. Архитектура ОС UNIX

В системе принят ряд соглашений, которые соблюдаются всеми разработчиками программ под UNIX, в частности, поддержка POSIX.

Ядро – это набор системных таблиц и подпрограмм работы с ними. В ядро также входят драйверы устройств. Ядро состоит из статической части, которая загружается при старте системы, и модулей. Модули могут динамически загружаться при старте системы или во время работы, при необходимости поддержки той или иной функции. В частности, подсистема поддержки NFS и драйверы внешних устройств оформлены в виде модулей.

Демоны – это серверные резидентные приложения, отвечающие за обработку запросов от других (клиентских) программ.

Утилиты – это программы, которые нужны для выполнения разных базовых работ в системе: копирования файлов, управления процессами, восстановления файловой системы и т.п.

Рассмотрим ядро системы (рис. 12.3). Оно позволяет всем остальным программам общаться с аппаратной частью и с периферийными устройствами, регулирует доступ к файлам, управляет память и процессами. Основным достоинством ядра является строгая стандартизация системных API. За счет этого во многом достигается переносимость кода между разными версиями UNIX и абсолютно различным аппаратным обеспечением.

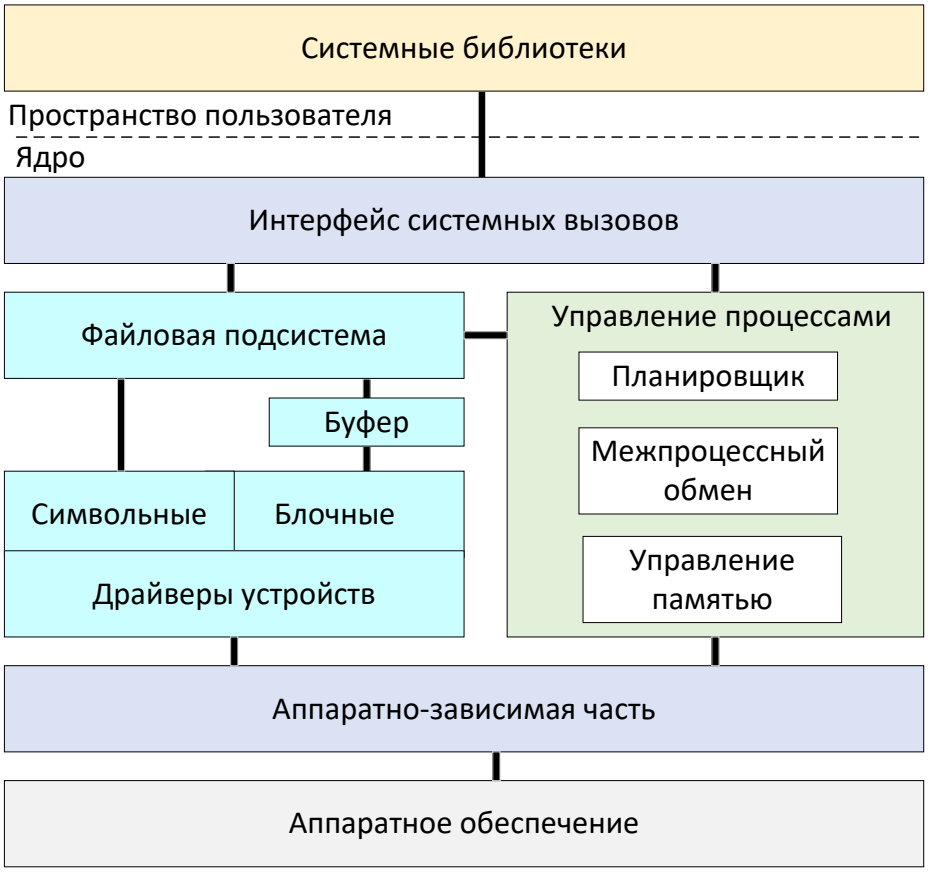


Рис. 12.3. Структура ядра ОС UNIX

Файловая подсистема – почти единственная из всех работает с драйверами, которые являются модулями ядра. «Почти», потому что существует еще и сетевая подсистема, которая работает, например, с драйвером сетевой карты и с драйверами различных современных сетевых устройств.

Обмен данными с драйверами может проходить двумя способами:

- с помощью буфера;
- с помощью потока.

Суть первого способа заключается в том, что для информации выделяется кэш (или сверхоперативная память, как его называли раньше), в который заносится необходимый блок данных. Далее информация из кэша передается к драйверу. Драйвер – единственный элемент ядра, способный управлять периферийными устройствами. Но подсистема управления файлами может взаимодействовать с драйвером и через поток. Поток представляет собой посимвольную передачу данных драйверу. Следует отметить, что способ взаимодействия с драйвером определяется не пользователем и не приложением. Он является характеристикой того устройства, которым управляет драйвер. Очевидно, что потоковое общение позволяет взаимодействовать более оперативно, чем общение через буфер. Ведь на заполнение буфера тратится время и, следовательно, возрастает время отклика.

Подсистема управления процессами – отвечает за синхронизацию и взаимодействие процессов, распределение памяти и планирование выполнения процессов. Для всех этих целей в подсистему управления процессами включены три модуля, которые наглядно продемонстрированы на схеме. Основные вызовы, служащие для работы с процессами (рис. 12.2):

- fork (создает новый процесса),
- exec (выполняет процесс),
- exit (завершает исполнение процесса),
- wait (один из способов синхронизации),
- brk (управляет памятью, выделенной процессу),
- signal (обработчики исключений) и др.

Модуль управления памятью – позволяет избежать нехватки оперативной памяти. Используя механизмы свопинга и виртуальной памяти модуль выполняет очень важную функцию - он определяет какому процессу сколько выделить памяти

Планировщик процессов (SWOPPER) – организует исполнение нескольких процессов и переключение между ними: планирование исполнения, переключение контекста, поддержку механизма приоритетов, обмен информацией между процессами и синхронизацию.

Модуль межпроцессного обмена – позволяет различным процессам обмениваться между собой информацией.

На уровне аппаратного управления происходит обработка прерываний и связь ядра с железом.

12.4. Обработка процессов

UNIX – это многопользовательская многопроцессная система, т.е. в ней может одновременно быть запущено несколько процессов от имени разных пользователей. Число процессов, которые можно одновременно за-

пустить, ограничивается размерами таблицы процессов в ядре и другими настройками ядра.

Схемы взаимодействия между процессами соответствуют механизму сопрограмм. Использование сопрограмм упрощает логику ядра системы и требует одного выделенного процесса, который создается нестандартным образом. С него начинается работа системы после запуска. В ОС UNIX этот процесс называется «диспетчерским» («swapper»), он не имеет пользовательской фазы.

Все процессы в ОС UNIX, кроме диспетчерского, создаются операцией «Порождение». В этой операции участвуют два процесса: порождающий и порожденный. Порождающий выполняет системный вызов (fork), в результате появляется порожденный процесс. При запуске процесс получает уникальный идентификатор процесса (Process IDentifier, PID), по которому он становится доступен другим процессам и планировщику.

На рис. 12.4 представлена блок-схема жизненного цикла процесса в ОС UNIX.

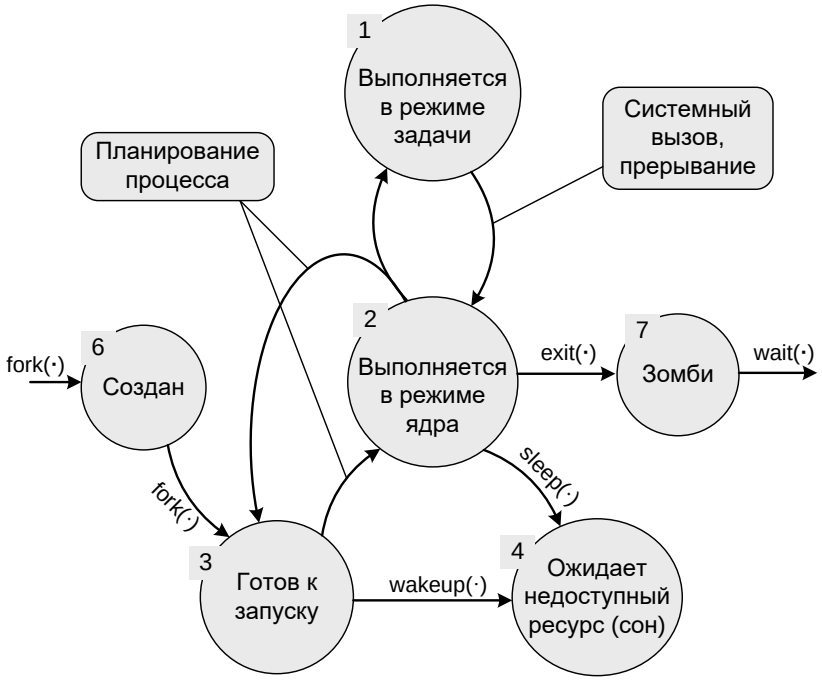


Рис.12.4. Блок схема жизненного цикла процесса в ОС UNIX

Для создания системного процесса используется системный вызов fork (разветвление), в результате которого получаются два идентичных процесса, называемые родительский процесс и порожденный (дочерний) процесс. Они не имеют общей первичной памяти, но совместно используют все открытые файлы. Для уничтожения процесса имеется вызов exit, который завершает работу этого процесса и передает код возврата (завершения) родительскому процессу, при этом сегменты процесса уничтожаются. Остается структура пользования для родительского процесса. Когда родительский процесс подучил информацию об уничтожении порожденно-

го им процесса, тогда уничтожается структура пользования и освобождается место в таблице процессов.

В большинстве систем UNIX реализована возможность выполнять несколько параллельных подпроцессов внутри процесса. Эти подпроцессы называют потоками (threads).

Каждый процесс в UNIX работает в своем собственном адресном пространстве, поэтому сбой в работе одного процесса никак не влияет на работу других. Подсистема виртуальной памяти, являющаяся частью ядра, запрещает процессам обращаться к чужим адресным пространствам.

Адресное пространство процесса состоит из 3 сегментов:

- текстового сегмента (инструкции);
- сегмент данных;
- сегмент стека.

Одной из функций ядра является планирование процессов, т.е. передача управления от одного процесса к другому. Для этого в ядре есть отдельная подпрограмма, называемая планировщиком задач. Процессы получают управление от планировщика задач в соответствии со своим приоритетом. Планировщик задач через определенное количество микросекунд решает, следует ли передать управление следующему в очереди процессу. Распределением ресурсов между процессами занимается так же ядро ОС.

Процесс может находится: в режиме пользователя или в режиме системы. Ядро представляет собой отдельный процесс, выполняющийся с наивысшим приоритетом.

Управление процессами осуществляется в ОС UNIX с помощью двух структур.

1. PROC-STRUCTRE (блок управления процессом). Составляющие этого блока:

- состояние процесса;
- размер и адрес процесса;
- кому принадлежит процесс;
- идентификация процесса;
- канал ожидания;
- поле сигналов;
- таймер и счетчик используемого времени.

2. USER-STRUCTRE (структура использования) – содержит информацию о процессе, которая должна быть доступна только на уровне исполнения. Содержание этой структуры:

- параметры ввода-вывода (I/O), т.е. адреса буферов и т.д.;
- окружение в файловой системе (текущий каталог, коренной каталог);
- таблица открытых файлов;
- код возврата, номера ошибок;
- поле сигналов (информация, как надо реагировать на сигнал).

В ОС UNIX нет принципиальной разницы между задачами и заданиями. Работы в системе представлены множеством конкурирующих процес-

сов. Процесс строго последователен, нельзя выполнять асинхронные действия внутри процесса. Даже операция I/O не может быть выполнена асинхронно.

Надо отметить, что в процесс диспетчеризации процессов ядро UNIX/Linux обрабатывает два вида исключений, которые обычно называют «oops» и «panic». Почти в каждой операционной системе «panic» происходит в тех случаях, когда ядро обнаруживает серьезную неисправность. Если система каким-либо образом повредила сама себя, ей требуется остановиться немедленно, пока она не произведет необратимых критических изменений (типа уничтожения файловой системы). Везде, где только возможно, UNIX/Linux пытается детектировать проблему и справиться с ней без остановки всей системы. Например, многие ситуации типа «oops» приводят к завершению процесса, который нормально запустился, но потом зациклил систему. Бывают, однако, ситуации, когда все настолько плохо, что полная «panic» является наилучшим выходом. Считается, что пользователи стабильных версий ядра не должны встречать ни «паник», ни «oops». Но в реальном мире они иногда происходят.

12.5. Организация пользователей

Каждый пользователь в UNIX имеет свою собственную *учетную запись пользователя* (account), которая содержит имя пользователя, пароль, идентификатор пользователя (UID), идентификатор главной группы пользователя (GID), описание пользователя, его домашний каталог и путь к командному процессору, который следует запустить при интерактивном входе пользователя в систему.

Пользователь, работающий в UNIX, имеет уникальное имя пользователя и уникальный идентификатор. Идентификатор пользователя (User ID, UID) - это целое число от 0 до 2147483647. Обычные пользователи в Solaris имеют идентификатор в диапазоне от 100 до 60000.

Пользователю не надо знать свой идентификатор, потому что он используется только системой, а для входа в систему пользователь указывает свое имя (username). Пользователи объединены в группы. Каждая группа имеет свое имя и уникальный идентификатор (Group ID, GID). В группе может быть сколько угодно пользователей, и каждый из них может быть участником любого количества групп. Однако у каждого пользователя есть главная группа - она указывается в свойствах любого файла, который создает пользователь. Идентификатор группы имеет значение от 100 до 60000, если только это не специальная группа. Для специальных (предопределенных) групп зарезервирован диапазон от 0 до 99.

Пользователей объединяют в группы для того, чтобы было удобнее администрировать систему.

Концепция прав доступа в UNIX требует объединять пользователей в группы всегда, когда нужно предоставить одинаковые права доступа к файлам или каталогам группе людей.

Домашний каталог и путь к командному процессору играют роль только при интерактивном входе пользователя в систему. В UNIX каждый пользователь может работать с системой как непосредственно (набирая команды ОС на клавиатуре), так и обращаясь через сеть к тем или иным службам, запущенным на компьютере под UNIX.

При установке системы автоматически создаются предопределенные пользователи и группы. Предопределенные группы и пользователи требуются для того, чтобы от их имени работали системные службы, и в то же время доступ к файлам этих служб был ограничен для всех остальных. Кроме того, это делается для того, чтобы было проще управлять правами доступа к системным файлам.

Один из предопределенных пользователей – это пользователь root с UID, равным нулю. Пользователь с таким UID называется суперпользователем (superuser) или привилегированным пользователем и всегда имеет имя root. Он имеет неограниченные права на доступ к любому объекту в системе. Суперпользователь, является как правило системным администратором системы и отвечает за безопасность ОС, ее стабильную работу, добавление и удаление пользователей, регулярное резервное копирование и т.д.

12.6. Работа с файловыми системами

В разных вариантах UNIX используются разные файловые системы, причем часто поддерживается несколько разных файловых систем. Например, Linux умеет работать с ext2, ext3 (это ее родные файловые системы), UFS, HPFS, NTFS, FAT и другими. ОС Solaris, и некоторые другие ОС UNIX, по умолчанию использует файловую систему типа UFS.

Файловая система UNIX характеризуется:

- иерархической структурой;
- согласованной обработкой массивов данных;
- возможностью создания и удаления файлов;
- динамическим расширением файлов;
- защитой информации в файлах;
- трактовкой периферийных устройств (например, таких как терминалы и принтеры) как файлов.

12.6.1. Организация разделов

Одной из целей разделения на разделы является повышение сохранности данных на случай непредвиденных происшествий. Путем разделения жесткого диска на разделы, данные могут быть сгруппированы и разобщены. Когда происходит авария, повреждаются данные только одного раздела, а данные других разделов скорее всего уцелеют. Даже журналируемая файловая система обеспечивает только защиту данных в случае сбоя питания и неожиданного отключения устройств хранения. Она не защищает

ваши данные от испорченных блоков и логических ошибок в файловой системе.

Есть два вида основных разделов в системе UNIX:

1. *раздел с данными*: обычные данные системы UNIX, включая *корневой раздел*, содержащий все данные для старта и запуска системы;
2. *раздел подкачки*: расширение физической памяти компьютера, представляет собой дополнительную память на жестком диске.

Пространство для подкачки (обозначается как swap) доступно только для самой системы, и скрыто при обычной работе.

Наряду с указанными двумя, UNIX поддерживает множество других типов файловых систем, такие как Reiser, JFS, NFS, FATxx и многие другие файловые системы, изначально доступные на других (проприетарных) операционных системах.

Большинство систем UNIX содержат корневой раздел, один или несколько разделов с данными, и один или несколько разделов подкачки. Системы в смешанных средах, могут содержать разделы данных других систем, такие как разделы файловых системам FAT или VFAT с данными ОС Windows.

Корневой раздел обозначается одиночной косой чертой, «/» и содержит системные конфигурационные файлы, большинство основных команд и серверные программы, системные библиотеки, некоторое временное пространство и домашний каталог пользователя с правами администратора.

Во многих дистрибутивах UNIX ядро находится на отдельном разделе, поскольку это самый важный файл вашей системы. В этом случае такой раздел содержащий ваше ядро (ядра) ОС и сопутствующие файлы данных монтируется в точку /boot.

Остаток жесткого диска(ов) обычно делится на разделы данных обычно по следующему принципу:

- раздел для пользовательских программ (/usr);
- раздел, содержащий данные пользователей (/home);
- раздел для хранения временных данных (/var);
- раздел для дополнительного программного обеспечения (/opt).

Все разделы подключаются к системе через точки монтирования. Точка монтирования определяет место расположения конкретных данных в файловой системе. Во время запуска системы, автоматически монтируются все разделы, которые описаны в файле /etc/fstab.

На сервере системные данные стремятся отделить от пользовательских данных. Программы различных служб хранятся отдельно от данных, которые они обрабатывают. На таких системах создаются различные разделы:

- раздел со всеми данными, необходимыми для загрузки машины;
- раздел с конфигурационными данными и серверными программами;

- один или несколько разделов, содержащих серверные данные, такие как таблицы базы данных, почта пользователей, FTP-архив и т.д.;
- раздел с пользовательскими программами и приложениями;
- один или несколько разделов для конкретных пользовательских файлов (домашние каталоги);
- один или несколько разделов подкачки (виртуальная память).

12.6.2. Организация древа файловой системы

Файловая система организована в виде древовидной структуры с одной исходной вершиной, которая называется *корневой директорией* или «корнем» файловой системы (записывается: «/»). Этот каталог, содержит все основные каталоги и файлы.

Имя пути поиска состоит из компонент, разделенных между собой наклонной чертой (/) – каждая компонента представляет собой набор символов, составляющих имя вершины (файла), которое является уникальным для каталога (предыдущей компоненты), в котором оно содержится. Полное имя пути поиска начинается с указания наклонной черты и идентифицирует файл (вершину), поиск которого ведется от корневой вершины дерева файловой системы с обходом тех ветвей дерева файлов, которые соответствуют именам отдельных компонент.

Большинство каталогов в файловой системе UNIX хранит специфические данные или содержит отдельные компоненты системы. Назначение основных системных каталогов в файловой системе UNIX приведено в таблице 12.1.

Таблица 12.1 – Назначение основных системных каталогов в файловой системе UNIX

Директория	Содержимое
/bin	Общие программы для совместного использования системой, системным администратором и пользователями
/boot	Загрузочные файлы и ядро, vmlinuz. В некоторых последних дистрибутивах также данные grub. Grub – это большой единый загрузчик, который представляет собой попытку избавиться от многих различных загрузчиков известных нам на сегодняшний день
/dev	Содержит ссылки на все периферийные устройства, которые представлены файлами с особыми свойствами
/etc	Большинство важных системных файлов конфигурации находятся в /etc, этот каталог содержит данные, аналогичные панели управления Windows
/home	Домашние каталоги обычных пользователей
/initrd	Информация для загрузки (в некоторых дистрибутивах)

/lib	Файлы библиотек, включает файлы для всех разновидностей программ, необходимых системе и пользователям
/lost+found	Каждый раздел имеет lost+found в его верхней директории. Здесь находятся файлы, которые были спасены во время сбоев.
/misc	Для разных целей
/mnt	Стандартные точки монтирования для внешних файловых систем, например, CD-ROM или цифровой камеры
/net	Стандартные точки монтирования для удаленных файловых систем
/opt	Как правило, содержит дополнительное ПО и ПО третьих сторон
/proc	Виртуальная файловая система, содержащая информацию о системных ресурсах. Более подробная информация о назначении файлов в proc можно получить, введя команду man proc в окне терминала. Файл proc.txt рассматривает виртуальную файловую систему в деталях
/root	Домашняя директория суперпользователя root
/sbin	Программы для использования системой и системным администратором
/tmp	Временное место для использования системой, которое очищается после перезагрузки
/usr	Программы, библиотеки, документация и т.д. для всех пользовательских программ
/var	Место хранения всех изменяемых и временных файлов, созданных пользователями, такие как log-файлы, почтовые очереди, the print spooler area, место для временного хранения файлов, загружаемых из Интернета, или сохранения образа CD перед записью

Вариант организации файловой системы UNIX (на примере ОС Linux Red Hat) приведен на рис. 12.5. В зависимости от системного администратора, операционной системы и назначения UNIX-машины, структура файловой системы может меняться, и каталоги по желанию могут быть опущены или добавлены.

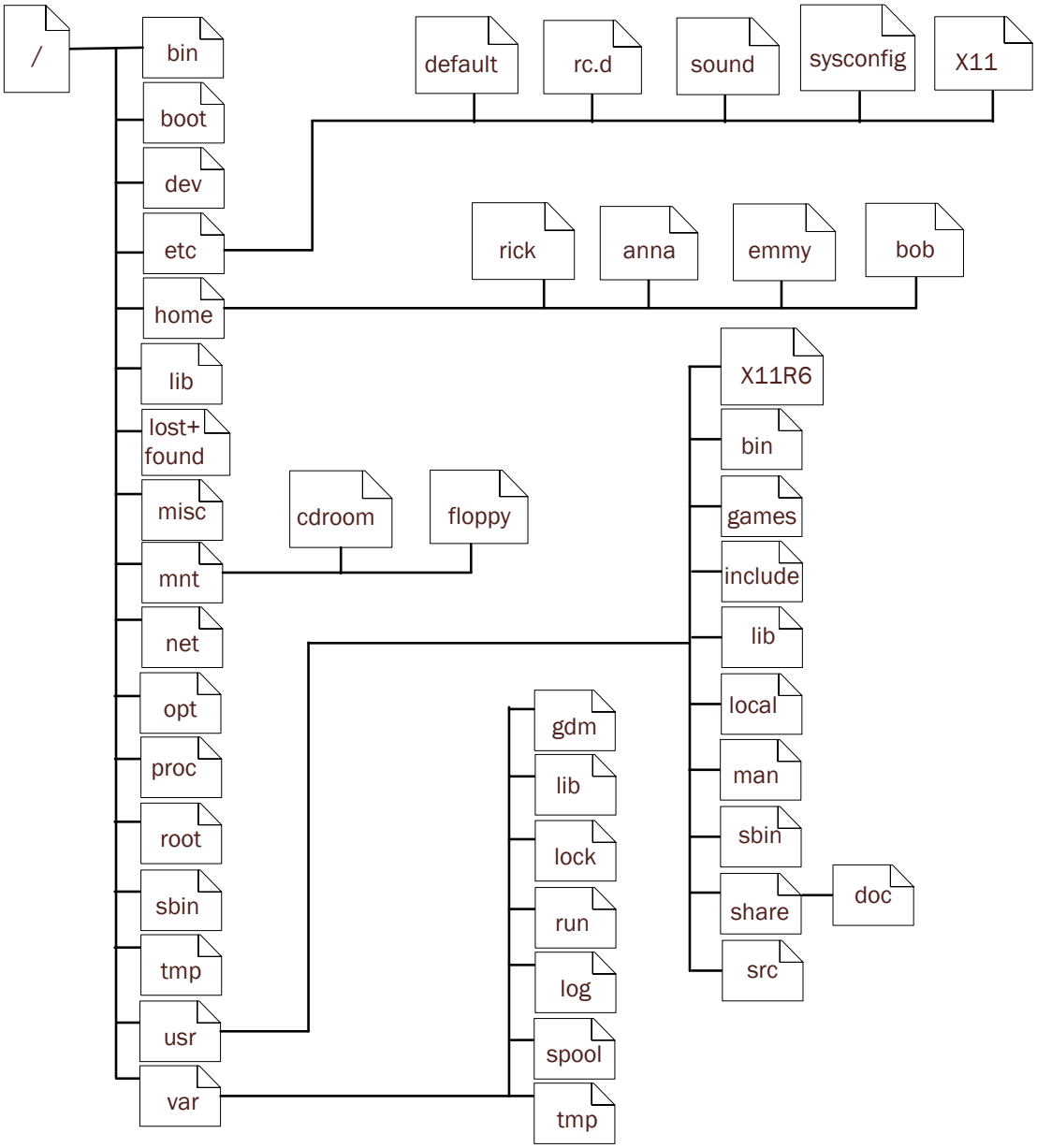


Рис. 12.5. Пример древовидной структуры файловой системы UNIX

12.6.3. Каталоги и файлы

Каталоги похожи на обычные файлы в следующем отношении – система представляет информацию в каталоге набором байтов, но эта информация включает в себя имена файлов в каталоге в объявленном формате для того, чтобы операционная система и программы, такие как ls (выводит список имен и атрибутов файлов), могли их обнаружить.

Для пользователя система UNIX трактует устройства так, как если бы они были файлами. Устройства, для которых назначены специальные файлы устройств, становятся вершинами в структуре файловой системы. Обращение программ к устройствам имеет тот же самый синтаксис, что и обращение к обычным файлам – семантика операций чтения и записи по отношению к устройствам в большой степени совпадает с семантикой операций чтения и записи обычных файлов. Способы защиты устройств сов-

падает со способом защиты обычных файлов – путем соответствующей установки битов разрешения доступа к ним (файлам).

Поскольку имена устройств выглядят так же, как и имена обычных файлов, и поскольку над устройствами и над обычными файлами выполняются одни и те же операции, большинству программ нет необходимости различать типы обрабатываемых файлов.

Таким образом, в общем случае, в файловой системе UNIX могут быть следующие объекты:

- *каталоги* – файлы, которые представляют собой списки других файлов;
- *специальные файлы*: механизм использования ввода-вывода. Большинство специальных файлов находятся в /dev;
- *ссылки*: механизм обеспечения видимости файла или каталога во множестве частей файлового дерева системы. Мы в деталях поговорим о ссылках;
- *домены* (*сокеты*) – особый тип файла, подобный сокетам TCP/IP, обеспечивающий взаимодействие в сети процессов, защищенных контролем файловой системы на доступ;
- *именованные каналы* – действуют более или менее похоже на сокеты и обеспечивают способ коммуникации между процессами без использования правил поведения сетевых сокетов.

В файловой системе, файл представлен с помощью *inode* (*индексного дескриптора*), своего рода серийного номера, содержащего информацию о данных этого файла: кому принадлежит этот файл, и где он находится на жестком диске.

Каждый раздел имеет свой собственный набор индексных дескрипторов; на всей системе с несколькими разделами могут существовать файлы с одним и тем же номером индексного дескриптора.

Каждый inode описывает структуру данных на жестком диске, хранит информацию о свойствах файла, в том числе физическое местоположение его данных. Когда жесткий диск назначается для хранения данных (обычно во время начала процесса установки системы или при добавлении дополнительных дисков к существующей) в разделе создается определенное количество индексных дескрипторов. Этот число будет максимальным количеством файлов всех типов (в том числе каталогов, специальных файлов, ссылок и т.д.), которые могут существовать в одно и то же время на этом разделе. Как правило, на 1 inode приходится от 2 до 8 килобайт памяти.

Во время создания нового файла, он получает свободный inode. В этом индексном дескрипторе содержится следующая информация:

- владелец и группа-владельца файла;
- тип файла (обычный, каталог, ...);
- разрешения на файл;
- дата и время создания, последнего открытия и изменения;

- дата и время, когда эта информация была изменена в индексном дескрипторе;
- количество ссылок на этот файл;
- размер файла;
- адрес, определяющий фактическое расположение данных файла.

12.6.3. Права доступа

Права доступа к файлу регулируются установкой специальных битов разрешения доступа, связанных с файлом. Устанавливая биты разрешения доступа, можно независимо управлять выдачей разрешений на чтение, запись и выполнение для трех категорий пользователей:

- владельца файла;
- группы пользователя;
- прочих пользователей.

Пользователи могут создавать файлы, если разрешен доступ к каталогу. Вновь созданные файлы становятся листьями в древовидной структуре файловой системы.

Программы, выполняемые под управлением системы UNIX, не содержат никакой информации относительно внутреннего формата, в котором ядро хранит файлы данных, так как данные в программах представляются как бесформатный поток байтов.

Программы могут интерпретировать поток байтов по своему желанию, при этом любая интерпретация никак не будет связана с фактическим способом хранения данных в операционной системе. Так, синтаксические правила, определяющие задание метода доступа к данным в файле, устанавливаются системой и являются едиными для всех программ, однако семантика данных определяется конкретной программой.

12.7. Примеры некоторых широко распространенных операционных систем семейства UNIX

12.7.1. Сетевая операционная система Sun Solaris

Sun Solaris – UNIX-подобная операционная система, разработанная компанией Sun Microsystems, входит в число самых известных коммерческих версий UNIX. Эта ОС обладает развитыми средствами поддержки сетевого взаимодействия и представляет собой одну из самых популярных платформ для разработки корпоративных решений. Для нее существует около 12 тыс. различных приложений, в том числе серверов приложений и СУБД почти от всех ведущих производителей. Solaris соответствует многим промышленным стандартам и характеризуется высокой масштабируемостью. Для подавляющего большинства приложений эта ОС обеспечива-

ет практически линейный рост производительности при увеличении числа процессоров за счет симметричных многопроцессорных вычислений.

Solaris работает на платформах SPARC, x86 и x64 – с процессорами от Sun, AMD и Intel, на компьютерах любого масштаба – лэптопах, десктопах, рабочих станциях, серверах и в кластерах. Solaris в силу архитектурных особенностей потенциально менее подвержена вирусам, чем широко распространенные системы других архитектур. На практике подтвержденных случаев заражения, например, Solaris 10 вирусами не зарегистрировано вообще. Четкое разделение прав между обычными и привилегированными пользователями гарантирует, что ошибочные действия пользователей не нанесут вреда системе в целом.

Solaris имеет ряд преимуществ перед другими ОС с открытым кодом.

Уникальный механизм виртуализации. В Solaris реализована уникальная схема виртуализации, которая пока нигде больше не применяется: внутри одного экземпляра ОС можно запустить множество неглобальных зон (т.е. «виртуальных» экземпляров системы). Первичный экземпляр Solaris, стартующий при включении компьютера, называется глобальной зоной, внутри нее располагаются неглобальные зоны. Таким образом, в пределах одного физического компьютера можно запустить до 8192 независимых друг от друга экземпляров Solaris. Важно, что все зоны разделяют одно общее ядро, а сами друг друга «не видят». Обработка системных вызовов приложений из всех зон одним ядром значительно снижает накладные расходы на переключение между разными виртуальными системами, по сравнению с VMware.

Зонная (контейнерная) схема виртуализации позволяет совершенно изолировать приложения друг от друга, запуская их в разных зонах. Это дает возможность:

1. создать идеальную лабораторную среду, например, для каждого выполняющего лабораторную работу выделить его собственную виртуальную систему и дать ему все права на управление этой системой;
2. разделить между собой приложения разных пользователей, например, хостинг-провайдер может каждому клиенту дать полный доступ к его собственной клиентской виртуальной системе, включая доступ по ssh;
3. ограничить нагрузку на сервер, которую создает приложение или группа приложений, например, поместить СУБД в одну зону, а веб- сервер в другую и сбалансировать нагрузку на память и процессор для зон, указав предельно допустимые значения для каждой из них.

Надежная транзакционная файловая система (ZFS). Файловая система ZFS, которая поддерживается в Solaris 10 (начиная с update 2) и современных выпусках Solaris. Кроме этих систем, поддержка ZFS портирована во FreeBSD (с некоторыми ограничениями).

Традиционная файловые системы, которые используют в UNIX (например, UFS или ext3), имеют ряд недостатков, которые стали особенно заметны с ростом количества и размера дисковых накопителей. К ним относятся:

- трудность администрирования массивов дисков из нескольких десятков, сотен или тем более тысяч штук (необходимость разбивать диск на разделы, настраивать монтирование этих разделов, управлять доступом и квотами каждого из них, расширять файловую систему и т.п.);
- ограничение по максимальному размеру файлов и разделов, которое становится препятствием с ростом объема накопленных данных;
- сложность и недостаточная надежность резервирования, сложность резервного копирования;
- значительные затраты времени на проверку и восстановление данных (например, при аварийной перезагрузке файлового сервера в любой системе UNIX без ZFS).

Поэтому ZFS спроектирована, чтобы повысить надежность хранения данных и облегчить управление дисковой подсистемой.

В основу ZFS положены три основных принципа:

1. объединение всего доступного дискового пространства в пул;
2. сквозной контроль целостности данных;
3. транзакционность.

Все диски, доступные системе, объединяются в единый пул, и уже из него по запросу выделяется пространство для отдельных файловых систем. Такой же принцип положен в основу виртуальной памяти – когда приложение запрашивает у ОС один мегабайт памяти для размещения данных, ему безразлично, в какой области памяти для него будет выделен этот мегабайт. В ZFS при необходимости расширить пул в него просто добавляется новый диск – одной командой. Необходимости объединять разделы дисков в тома больше нет.

Для каждого блока данных и метаданных выполняется контрольное суммирование, причем контрольная сумма блока хранится в родительском блоке метаданных – чтобы физически разделить места хранения информации и контрольной суммы для этой информации. Используемые в ZFS алгоритмы на современном процессоре класса Opteron позволяют вычислять контрольные суммы со скоростью 2-8 Гбайт/с.

В файловой системе ZFS данные всегда хранятся в целостном и непротиворечивом состоянии, так как запись на диск выполняется группами транзакций. Уже записанные на диск данные и метаданные при их изменении не перезаписываются, а пишутся в свободное пространство диска (такой метод записи обозначают термином «копирование при записи», copy-on-write). По окончании группы транзакций происходит атомарная операция – запись нового uber-блока (главного блока файловой системы). Это подтверждает завершение группы транзакций. С этого момента блоки, ко-

торые до начала группы транзакций были заняты данными, изменявшимися в ходе транзакции, считаются освобожденными. Если в ходе транзакции произошел сбой (например, отключилось питание), диск после восстановления питания оказывается в том же состоянии, в котором был до начала транзакции.

Такая организация файловой системы не только повышает надежность, но и увеличивает ее производительность, так как низкоуровневый драйвер файловой системы теперь сам заботится о порядке записи данных на диск в ходе транзакции и способен его оптимизировать.

Средство обеспечения надежности ZFS: 256-битные контрольные суммы блоков сохраняются не вместе с блоками, а в блоке адресации верхнего уровня, т.е. вместе с адресом блока хранится контрольная сумма адресуемого блока данных. Контрольные суммы считаются по любому объекту, не только по блоку данных, но и по блоку адресации, и если в блоке адресации из-за сбоя при хранении оказался неверный адрес, то система сама определит, что адрес неверный и ошибочные данные просто не будут выданы по запросу.

Интегрированная среда разработки приложений Sun Studio для ОС Solaris обладает хорошей функциональностью, компиляторами с нескольких языков. Пакет Sun Studio бесплатен и доступен для Solaris и Linux.

12.7.2. Сетевая операционная система FreeBSD

FreeBSD – сетевая ОС с современными техническими характеристиками, что обусловливает не только широкое ее распространение, но и делает ее простым и практичным инструментом для предоставления различных услуг.

FreeBSD основана на 4.4 BSD-Lite от Computer Systems Research Group (CSRG) Калифорнийского Университета в Беркли. Однако FreeBSD – это не продукт какой-то одной компании, это результат деятельности крупного свободного сообщества: в проект FreeBSD входят разработчики, пользователи и бесчисленное множество сторонников. Это ПО, распространяемое с открытыми исходными текстами, доступное для бесплатного использования и распространения. FreeBSD создается из кода, предоставленного сотнями людей со всего мира, сотрудничающих с помощью Интернета.

Инфраструктура разработки FreeBSD включает в себя следующие организационно-технические элекменты.

- *CVS-репозиторий*. Главное дерево исходных текстов FreeBSD поддерживается с помощью CVS (Concurrent Versions System), свободно доступной системой контроля исходных текстов, которая поставляется вместе с FreeBSD. Основной CVS репозиторий располагается на компьютере, находящемся в городе Санта Клара, Калифорния (США), откуда и распространяется на множество зеркал по всему миру. Дерево CVS, содержащее ветви -CURRENT

и -STABLE, может быть легко скопировано на любой локальный компьютер.

- *Список коммиттеров*. Коммиттеры (committers) – это люди, которые имеют доступ на запись к главному дереву CVS, и имеют право вносить изменения в главное дерево исходных текстов FreeBSD (термин «коммиттер» появился от названия команды cvs(1) commit, которая используется для внесения изменений в CVS-репозиторий). С коммиттерами сотрудничают помощники (contributors) и пользователи (users).
- *Core-группа FreeBSD*. Выборный общественный орган (выборы проходят каждые 2 года). Главная задача Core-группы – гарантировать, что проект в целом в хорошем состоянии и движется в правильном направлении. Приглашение постоянных и ответственных разработчиков присоединиться к группе коммиттеров – одна из функций Core-группы.

Существует два класса процессоров, на которых может работать FreeBSD/amd64. К первому принадлежат процессоры AMD64. Ко второму классу принадлежат процессоры архитектуры Intel EM64T.

Список поддерживаемого оборудования поставляется с каждым релизом в FreeBSD в информации о релизе, публикуемом на сайте FreeBSD.

Основные возможности и преимущества FreeBSD:

- FreeBSD имеет 32-разрядную и 64-разрядную версии;
- вытесняющая многозадачность с динамическим регулированием приоритетов, позволяющая плавно и справедливо распределить ресурсы компьютера между приложениями и пользователями, даже при тяжелейших нагрузках;
- виртуальная память с поддержкой сброса неиспользуемых страниц по требованию и объединение виртуальной памяти и буферного кэша спроектированы так, чтобы максимально эффективно удовлетворить приложения с большими запросами к памяти и, в то же время, сохранить интерактивность для остальных пользователей;
- многопользовательская поддержка, которая позволяет множеству людей использовать FreeBSD совместно для различных задач. Это значит, например, что системная периферия, такая как принтеры и ленточные устройства, правильно разделяется всеми пользователями в системе или сети, и что пользователям или группам пользователей могут быть установлены лимиты каждого ресурса, защищая критические системные ресурсы от перегрузок;
- мощный TCP/IP-стек с поддержкой промышленных стандартов, таких как SLIP, PPP, NFS, DHCP и NIS. Это означает, что FreeBSD может легко взаимодействовать с другими системами, а также работать сервером масштаба предприятия, предоставляя жизненно важные функции, такие как NFS (удаленный доступ к файлам) и услуги электронной почты, или представить вашу ор-

ганизацию в Интернете, обеспечивая работу служб WWW, FTP, маршрутизацию и функции межсетевого экрана (брандмауэра).

- защита памяти гарантирует, что приложения (или пользователи) не смогут чинить препятствия друг другу. Фатальная ошибка в выполнении одного приложения не скажется на работоспособности всей системы;
- поддержка симметричной многопроцессорности (SMP) для машин с несколькими процессорами;
- промышленный стандарт X Window System (X11R6) предоставляет графический интерфейс пользователя (GUI) для большинства VGA карт и мониторов, и поставляется с полными исходными текстами;
- двоичная совместимость с большинством программ, созданных для Linux, SCO, SVR4, BSDI и NetBSD;
- совместимость по исходным текстам: система совместима с большинством популярных коммерческих UNIX-систем и, таким образом, большинство приложений требуют лишь небольших изменений для сборки (или не требуют вообще);
- тысячи готовых к использованию приложений доступны из коллекций портов и пакетов FreeBSD;
- полный комплект инструментов для разработчика: C, C++ и Fortran. Множество дополнительных языков программирования для исследований и разработки также доступны из коллекций портов и пакетов;
- доступность исходных текстов всей системы означает, что пользователь системы имеет максимальный контроль над операционной средой.

Поскольку исходные тексты FreeBSD общедоступны, система может быть оптимизирована для специальных приложений или проектов, а это, обычно, невозможно при использовании ОС от большинства коммерческих производителей. Например, для Интернет-служб: мощнейший TCP/IP стек делает FreeBSD идеальной платформой для большинства Интернет-приложений, таких как:

- FTP-серверы;
- WWW-серверы, как стандартные, так и защищенные (SSL);
- межсетевые экраны (firewalls) и шлюзы NAT;
- серверы электронной почты;
- серверы новостей или дискуссионных групп USENET и т.д.

FreeBSD используется в качестве платформы на некоторых крупнейших сайтах в Интернет, включая: Yahoo!, Apache, Blue Mountain Arts, Pair Networks, Sony Japan, Netcraft, Weathernews TELEHOUSE America, и др.

13. Сетевая операционная система Linux

13.1. История создания Linux

Linux – UNIX-подобная сетевая многопользовательская, многозадачная операционная система с открытым кодом, была разработана в первой версии Л. Торвальдсом в сентябре 1991 г. в рамках концепции Open Source («свободного программного обеспечения»). Концепция Open Source предложена Ричардом Столлманом в 1983 г. Позднее он создал ФСПО – Фонд свободного программного обеспечения (FSF – Free Software Foundation), из которого финансируются многие проекты для ОС Linux.

К концу 1990-х Linux превратилась в зрелую ОС, которую, тем не менее, пользователи настольных систем в большинстве своем игнорировали. На рынке персональных компьютеров, где властвовали компании Microsoft и Apple, ОС Linux считалась чересчур сложной в использовании. Клиенты, желавшие установить себе Linux, вынуждены были загружать исходные коды, вручную модифицировать файлы конфигурации и самостоятельно компилировать ядро. Кроме того, пользователям требовалось еще загрузить и установить приложения и пользовательский интерфейс для выполнения рабочих задач. По мере эволюции ОС Linux разработчики пришли к выводу о необходимости реализации дружественного процесса инсталляции системы. Это привело к появлению дистрибутивов, которые включали, помимо ядра, приложения, пользовательские интерфейсы и прочие утилиты и аксессуары.

В настоящее время существует более 300 разновидностей дистрибутивов, каждый из которых обладает своим набором функций. Популярностью пользуются дистрибутивы с дружественным пользователю интерфейсом, включающие большое количество приложений. В качестве примера можно назвать: Debian, Mandrake, Red Hat, Slackware и SuSE. Причем Mandrake, Red Hat и SuSE – это коммерческие компании, выпускающие дистрибутивы Linux для рынка высокопроизводительных серверов и настольных систем. Debian и Slackware представляют собой бесприбыльные организации, штат которых составляют разработчики- добровольцы, занимающиеся обновлением и поддержкой дистрибутивов Linux. В последнее время получает все большую популярность ОС Ubuntu (рус. Убунту; в переводе с зулу: ubuntu – человечность), операционная система, использующая ядро Linux и основанная на Debian. Основным разработчиком и спонсором является компания Canonical. В настоящее время проект активно развивается и поддерживается свободным сообществом. Другие дистрибутивы приспособлены для использования в специальных средах, например, для портативных устройств (таких, как Open Zaurus) и во встроенных системах (таких, как uClinux).

13.2. Архитектура Linux

Хотя ядро Linux является монолитным по своей природе, последние усовершенствования, внесенные для обеспечения масштабируемости ядра, включают в себя возможности модульности, наподобие тех, которые поддерживаются операционными системами с микроядром.

Ядро Linux состоит из шести основных подсистем, контролирующих доступ к системным ресурсам (рис. 13.1):

- управления процессами;
- взаимодействия между процессами;
- управления памятью;
- управления файловой системой;
- управления операциями ввода-вывода;
- сетевая подсистема.

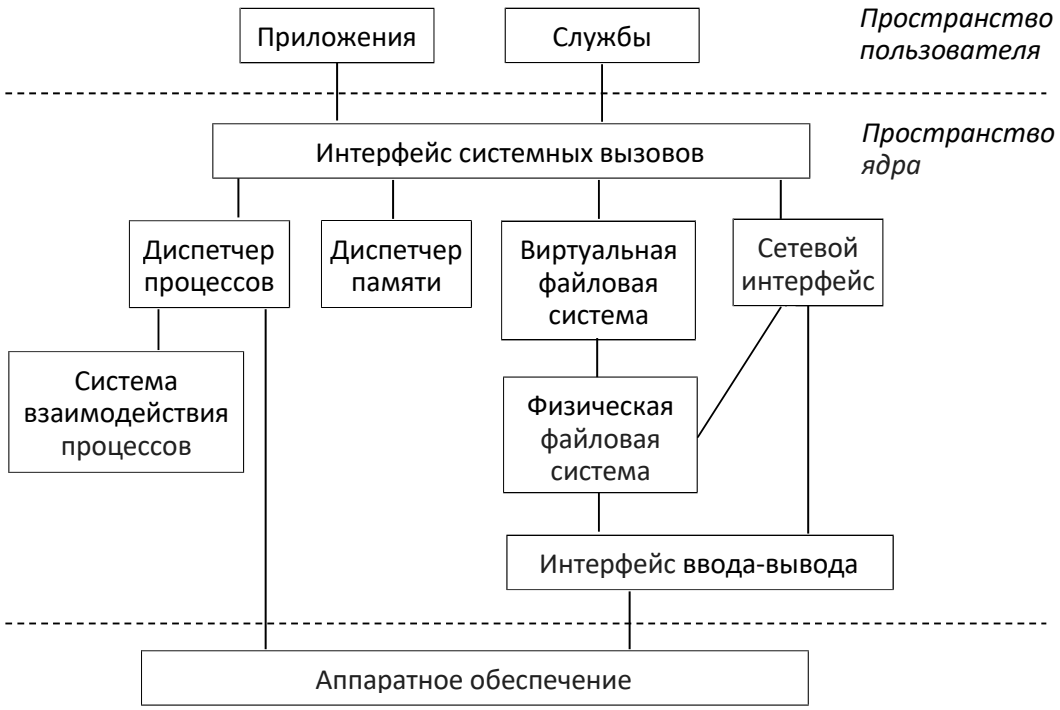


Рис. 13.1. Архитектура Linux

Процессы в Linux могут выполняться либо в режиме ядра, либо пользовательском режиме. Пользовательские процессы получают доступ к службам ядра через интерфейс системных вызовов. Когда от пользовательского процесса приходит разрешенный системный вызов (в пользовательском режиме), ядро обрабатывает системный вызов в режиме ядра от имени процесса. Если запрос некорректен (например, процесс пытается осуществить запись в файл, который не был открыт), ядро возвратит ошибку.

Диспетчер процессов (process manager) является главной подсистемой, отвечающей за создание процессов, обеспечение доступа к процессо-

ру (процессорам) системы и удалению процессов из системы по завершению их работы.

Система взаимодействия процессов (Interprocess communication – IPC) ядра позволяет процессам взаимодействовать друг с другом. Эта подсистема работает совместно с диспетчером процессов, обеспечивая совместный доступ к информации и передачу сообщений с помощью самых разнообразных методов.

Диспетчер памяти обеспечивает процессам доступ к памяти. Linux выделяет каждому процессу виртуальное адресное пространство, которое делится на пользовательское адресное пространство и пространство ядра. Включение пространства ядра в контекст каждого процесса уменьшает затраты на переключение между режимами ядра и пользователя, поскольку ядро может получить доступ к своим данным из виртуального адресного пространства любого процесса.

Пользователи получают доступ к файлам и папкам, перемещаясь по дереву каталогов. Корень дерева каталогов называется корневым (root) каталогом. Из корневого каталога пользователи могут перейти к любой доступной файловой системе. Пользовательские процессы запрашивают данные файловой системы через интерфейс системных вызовов. Когда системе нужен доступ к файлу или папке дерева каталогов, взаимодействие осуществляется через интерфейс виртуальной файловой системы VFS (virtual file system), обеспечивающий единый способ доступа ко всем файлам и каталогам, размещенным в неоднородных файловых системах (например, ext3 и NFS). Виртуальная файловая система передает запросы конкретной файловой системе, которая отвечает за схему размещении и место хранения данных.

Основываясь на модели UNIX, *интерфейс ввода-вывода* ОС Linux взаимодействует с устройствами, как с файлами, то есть использует те же механизмы доступа, что и при работе с файлами. Когда пользовательские процессы обмениваются данными с устройством, ядро передает запросы интерфейсу виртуальной файловой системы, которая перенаправляет их интерфейсу ввода-вывода. Интерфейс ввода-вывода передает запросы далее драйверам устройств, выполняющим операции ввода-вывода для системного оборудования.

В Linux существует *сетевая подсистема*, позволяющая процессам обмениваться данными с компьютерами по сети. Для отправки и приема пакетов сетевая подсистема использует сетевое оборудование системы, получая доступ к нему через интерфейс ввода-вывода. Эта подсистема позволяет приложениям и ядру инспектировать и модифицировать пакеты, проходящие по сетевым уровням системы с помощью интерфейса фильтрации пакетов. Такой интерфейс позволяет реализовывать брандмауэры, маршрутизаторы и другие сетевые средства.

Пользователь и программы взаимодействует с ядром при помощи интерфейса системных вызовов, частью которого является специальная программная оболочка (shell), в которых предусмотрены средства переда-

чи команд от пользователя в ядро. По существу, shell является одной из прикладных программ, предназначенной для интерпретации команд пользователя, которые передаются с клавиатуры (или мыши) в ядро Linux. Команды преобразуются в коды, которые понимает ядро ОС. Оболочки shell бывают различные: командные или графические, например, KDE или GNOME. К известным оболочкам командного типа можно отнести: ash, bash, sh, ksh и другие, которые для управления системой предоставляют пользователю командную строку и развитую систему сообщений. Все они имеют встроенный интерпретатор команд пользователя и работают примерно одинаково.

13.3. Организация процессов и потоков

Организация процессов и потоков (*задач* – в терминологии Linux). Диспетчер процессов хранит список всех задач в виде двух структур данных.

1. Первая структура представляет собой *кольцевой список*, каждая запись которого содержит указатели на предыдущую и последующую задачу. Обращение к этой структуре происходит в том случае, когда ядру необходимо проанализировать все задачи, которые должны быть выполнены в системе.
2. Второй структурой является *хэш-таблица*. При создании задачи ей присваивается уникальный идентификатор процесса (PID – process identifier). Идентификаторы процессов передаются хэш-функции для определения местоположения процесса в таблице процессов. Хэш-метод обеспечивает быстрый доступ к специфическим структурам данных задачи, если ядру известен ее PID.

Каждая задача таблицы процессов представляется в виде структуры task_struct, служащей в роли дескриптора процесса (т.е. блока управления процессором – PCB). В структуре task_struct хранятся переменные и вложенные структуры, описывающие процесс. Задача переходит в состояние running (выполнения) после ее передачи в процессор (см. рис. 13.2).

При блокировке задача переходит в состояние sleeping (спячки), а при приостановке работы – в состояние останова (stopped). Состояние zombie (зомби) показывает, что выполнение задачи прекратилось, однако она еще не была удалена из системы. Например, если процесс состоит из нескольких потоков, он будет пребывать в состоянии зомби, пока все потоки не получат уведомление о завершении работы основного процесса. Задача в состоянии dead (смерти) может быть спокойно удалена из системы. Состояния active (активный) и expired (неактивный) используются при планировании выполнения процесса, и поэтому они не сохраняются в переменной state.

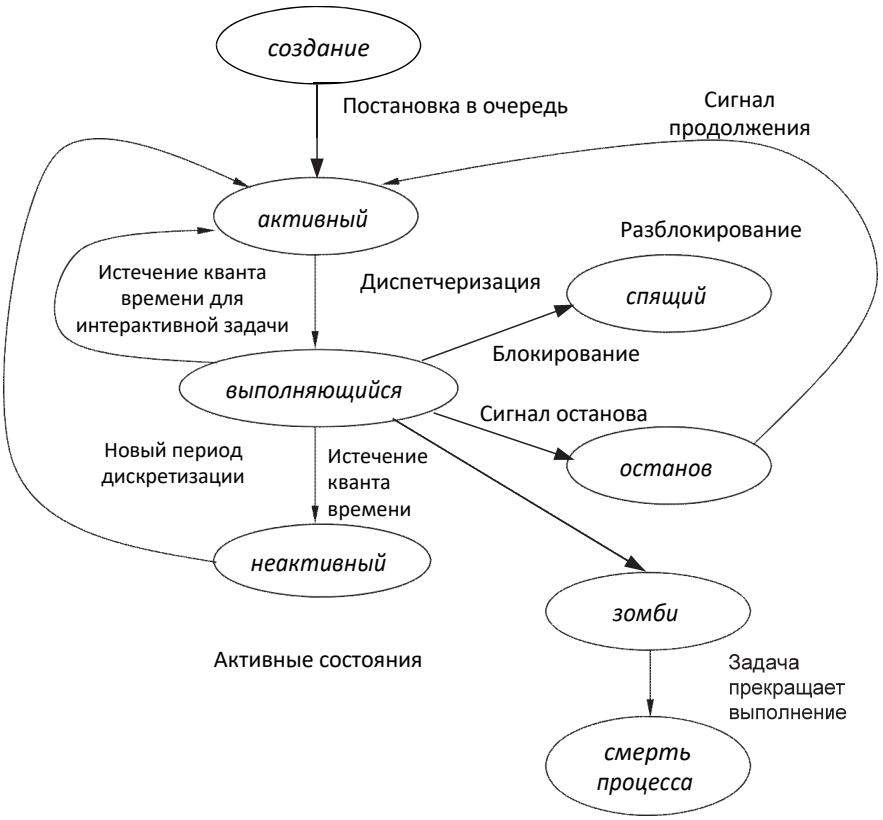


Рис. 13.2. Граф переходов состояний задач

При загрузке ядра обычно запускается процесс init, который использует ядро для создания всех остальных задач. Задачи создаются путем вызова системной функции clone.

Поддержка потоков в Linux также организована при помощи системного вызова процедуры clone, позволяющей вызывающему процессу задавать общий доступ к виртуальной памяти для потока, информацию о файловых системах, файловые дескрипторы и/или обработчики сигналов. При вызове процедуры clone из процесса выполняющего программный код ядра, создается поток ядра (kernel thread), отличающийся от остальных потоков тем, что он может обращаться непосредственно к адресному пространству ядра. В ядре в виде потоков реализованы несколько демонов. Демонами называются службы, пребывающие в спящем режиме до тех пор, пока ядро не разбудит их для выполнения таких задач, как сохранение страниц на вторичный носитель информации, планирование программных прерываний. Эти задачи обычно связаны с обслуживанием и потому выполняются регулярно.

После создания задачи с помощью clone, она помещается в очередь выполнения процессора, содержащую ссылки на все задачи, состязающиеся за процессорное время. Массив приоритетов содержит указатели на отдельные уровни очереди выполнения.

Когда задача переходит в состояние блокировки либо спячки, или же ее выполнение прекращается по какой-либо иной причине, задача удаляется из очереди выполнения.

Планировщик решает задачу бесконечного откладывания низкоприоритетных задач путем задания временных интервалов, называемых периодами дискретизации (epoch), в течение которых каждая задача из очереди выполнения должна быть запущена хотя бы раз. Планировщик поддерживает жесткое планирование задач реального времени, которые всегда выполняются с более высоким приоритетом, чем обычные задачи, поэтому обычная задача никогда не сможет их вытеснить.

13.4. Управление памятью

В Linux выделение памяти осуществляется, как правило, с использованием одного и того же размера страницы (чаще 4 КБайт или 8 КБайт). В ряде архитектур, где поддерживаются страницы большего размера (например, 4 МБ), код ядра может размещаться в страницах большего размера.

13.4.1. Организация виртуальной памяти

В 32-битовых системах Linux каждый процесс позволяет адресовать 232 байтов, то есть виртуальное адресное пространство процесса составляет 4 ГБайт. В 64-битовых системах ядро поддерживает виртуальную адресацию до 2 петабайт (т.е. 2 миллионов гигабайт). Записи, описывающие связь физических адресов с виртуальными, хранятся в таблицах страниц каждого процесса. Система виртуальной памяти поддерживает до трех уровней таблиц страниц для определения связей между виртуальными страницами и страничными блоками (см. рис. 13.3).

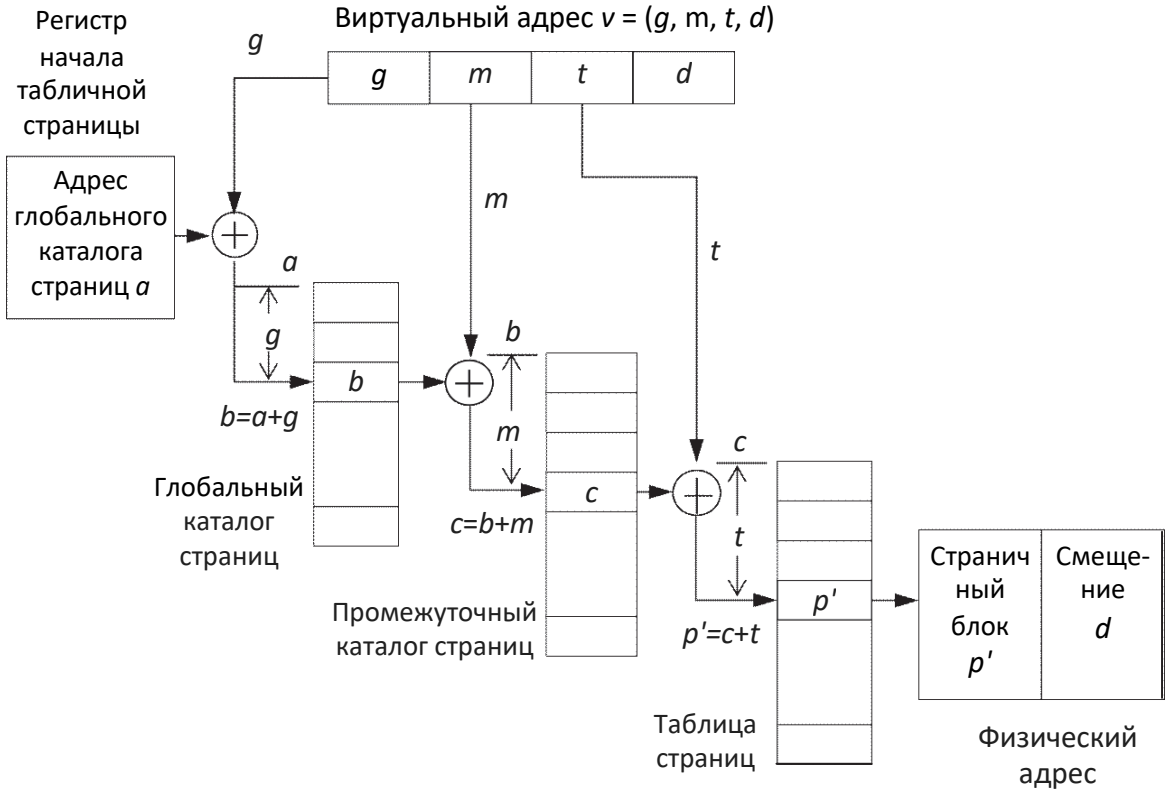


Рис. 13.3. Организация таблиц страниц

Первый уровень иерархии таблицы страниц (*глобальный каталог страниц*) хранит адреса таблиц второго уровня. Таблицы второго уровня, называемые *промежуточными каталогами страниц* (page middle directory) содержат ссылки на таблицы третьего уровня. Третий уровень, на котором размещаются сами *таблицы страниц*, непосредственно задает отображение виртуальных страниц в страничные блоки (кадры страниц).

13.4.2. Организация физической памяти

Система управления памятью делит физическое адресное пространство системы на три зоны (memory zone) (см. рис. 13.4).

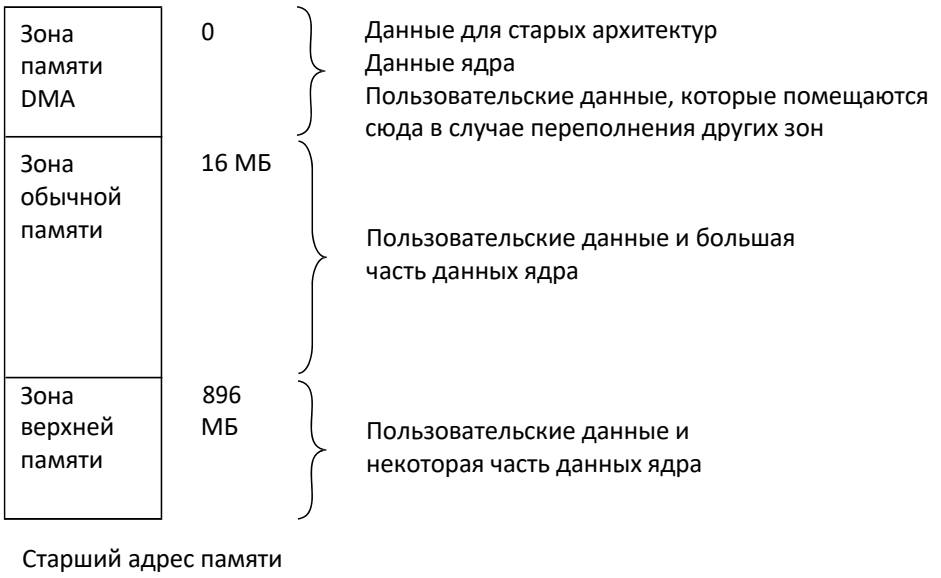


Рис. 13.4. Организация зон физической памяти (архитектура IA-32)

Размер каждой зоны зависит от конкретной архитектуры системы. Первая зона, называемая *зоной прямого доступа к памяти* (DMA memory) включает в себя основную память с адресами от 0 до 16 МБайт. Зона памяти DMA содержит данные и инструкции ядра (например, код программ начальной загрузки). Именно из этой зоны должна выделяться память для пользовательских процессов, когда свободная память в других зонах заканчивается.

Вторая зона, *зона обычной памяти* (normal memory), включает физическую память с адресами от 16 до 896 МБайт. Зона обычной памяти может использоваться для хранения пользовательских страниц и страниц ядра, а также данных устройств, способных работать в режиме DMA с памятью свыше 16 МБайт.

Третья зона, называемая *верхней памятью* (high memory), включит физическую память свыше 896 МБайт и вплоть до 64 ГБайт на процессорах x86. Верхняя память выделяется для пользовательских процессов, лю-

бых устройств, которые могут обращаться к этим областям памяти, и для временных структур данных ядра.

13.4.3. Замена страниц

Диспетчер памяти Linux самостоятельно определяет, какие страницы следует хранить в ОЗУ, а какие могут быть выгружены на диск (этот процесс носит название подкачки или свопинга) в случае нехватки свободной памяти. Свопинг возможен только для страниц из виртуального адресного пространства пользователя. Поэтому страницы, содержащие код и данные ядра, не могут участвовать в свопинге.

Для осуществления свопинга страниц диспетчер памяти использует два связанных списка для каждой зоны. В активном списке хранятся активные страницы, в неактивном списке – неактивные. Списки построены таким образом, что страницы, использовавшиеся в последнюю очередь, находятся в начале активного списка, а страницы, обращение к которым производилось довольно давно – в конце списка неактивных страниц.

13.4.4. Подкачка

Когда в системе возникает дефицит доступных страничных блоков, ядро должно принять решение о том, какие страницы памяти следует выгрузить, чтобы иметь свободные страницы памяти для обслуживания последующих запросов. За выполнение этой задачи отвечает демон подкачки kswapd, который высвобождает страницы, сохраняя неактуальные страницы на вторичном устройстве хранения.

13.5. Управление операциями ввода-вывода

13.5.1. Драйверы устройств

Драйвер устройства представляет собой программный интерфейс между системными вызовами и оборудованием. Авторами большинства драйверов для Linux-систем являются независимые разработчики. Из-за этого количество устройств, совместимых с операционной системой Linux, ограничено.

Как правило, драйверы устройств реализуются в виде загружаемых модулей ядра. Драйверы, выпущенные в виде модулей, легко можно загружать и выгружать по требованию, благодаря чему отпадает необходимость их постоянного включения в ядро.

Драйвер обрабатывает пакеты в соответствии с дисциплиной очереди, определяющей порядок обработки устройствами. По умолчанию используется политика FIFO (первый при шел – первый на выходе), но возможно и использование более сложных политик, например, на основе приоритетов.

13.5.2. Специальные файлы устройств

Большинство устройств в системе Linux представлено в виде специальных файлов устройств (device special file).

Специальные файлы устройств – это элементы каталога /dev, предоставляющие доступ к определенному устройству. Каждый файл в каталоге /dev соответствует блочному, либо символьному устройству. Список загруженных в текущий момент времени драйверов блочных и символьных устройств можно увидеть в файле /proc/devices.

Доступ к специальным файлам устройств организуется при помощи виртуальной файловой системы VFS. Системные вызовы передаются виртуальной файловой системе, которая, в свою очередь, вызывает драйверы устройств.

Большинство устройств, поддерживаемых Linux, относятся к одной из трех основных категорий:

1. символьные устройства;
2. блочные устройства;
3. сетевые устройства.

13.5.3. Символьные устройства

Символьные устройства (character device) передают данные в виде потока байтов. В данную категорию попадают принтеры, консоли, клавиатура, манипуляторы мышь и модемы. Поскольку передача данных осуществляется в виде потока байтов, большинство символьных устройств поддерживают только последовательный доступ к данным. В драйверах символьных устройств реализованы базовые операции, например, открытие, закрытие, чтение и запись в символьное устройство.

13.5.4. Блочные устройства

В отличие от символьных устройств, *блочные устройства* позволяют обращаться к данным, хранимым в блоках фиксированного размера, в любой момент времени, независимо от того, где именно на устройстве хранятся эти данные. Для облегчения произвольного доступа к большим массивам информации (например, файлам на жестком диске) ядро должно реализовать систему управления блочными устройствами ввода-вывода, которая по сравнению со схемой управления символьными устройствами ввода-вывода является более сложной. С этой целью в ядро включены алгоритмы, которые пытаются оптимизировать работу накопителей с подвижными головками. Подобно символьным устройствам, блочные устройства идентифицируются с помощью основного и дополнительного номеров. Подсистема блочного ввода-вывода ядра состоит из нескольких уровней, что позволяет придать модульность операциям ввода-вывода, благодаря помещению общего блока кода на каждом уровне.

На рис. 13.5 изображены уровни, через которые проходят блочные запросы ввода-вывода. Чтобы уменьшить количество времени, затрачиваемого на доступ к блочным устройствам, ядро использует две основных стратегии: кэширования данных и кластеризации операций ввода-вывода.

Рис. 13.5. Уровни блочной подсистемы ввода-вывода

13.5.5. Сетевые устройства

Принципиальное отличие между сетевым оборудованием и блочными или символьными устройствами заключается в том, что ядро не запрашивает данные с сетевых устройств. Здесь ситуация обратная – сетевые устройства используют прерывания, чтобы уведомить ядро о поступлении пакета. Как только ядро подготовит пакеты для передачи на другой узел сети, выполняется передача этих пакетов драйверу устройства соответствующей сетевой интерфейсной платы NIC (network interface card). Для определения платы-отправителя пакета ядро анализирует внутреннюю таблицу маршрутизации, в которой перечислены адреса получателей, доступные для каждой интерфейсной платы.

13.6. Взаимодействие процессов

Большинство механизмов взаимодействия процессов в Linux основаны на традиционных механизмах взаимодействия процессов UNIX, главной задачей которых является обмен информацией.

Первыми доступными механизмами взаимодействия процессов в UNIX-системах стали сигналы. Ядро использует сигналы, чтобы известить процессы о наступлении определенных событий. Сигналы не позволяют процессам обмениваться между собой объемами информации, превышающими по размеру слово данных, т.к. они предназначены для предупреждения процессов о случившемся событии. Сигналы, генерируемые ядром в ответ на прерывания и исключении, посылаются процессу либо потоку в результате выполнения инструкции, от другого процесса (например, когда один процесс прекращает выполнение другого, либо от асинхронного события (например, сигнал завершения операции ввода-вывода).

Ядро отправляет сигнал процессу, приостанавливая его выполнение и вызывая соответствующий обработчик сигналов. По завершении обработки сигнала выполнение процесса возобновляется.

Каналы (pipe) позволяют двум процессам взаимодействовать между собой с помощью модели источник/получатель. Процесс-источник осуществляет запись данных в канал, после чего процесс-получатель считывает данные из канала по принципу FIFO (первый пришел – первый ушел). При создании канала для него выделяется свой индексный узел (объект inode). Индексные узлы канала не ссылаются на дисковые блоки. Они указывают на определенную страницу данных, называемую буфером канала (pipe buffer), которая используется ядром в качестве циклического буфера. Каждый канал имеет свой уникальный буфер канала, содержащий информацию, передаваемую между двумя процессами. Каналы представляются в виде файлов, доступ к которым осуществляется с помощью виртуальной файловой системы. Чтобы начать обмен данными по каналу, процесс должен создать канал и породить дочерний процесс, с которым он сможет общаться с помощью этого канала. Однако в отличие от файлов с данными, именованные каналы ссылаются на буфер, размещенный в оперативной памяти, а не на диске.

Механизм взаимодействия процессов в Linux с помощью *сокетов* (socket) позволяет двум процессам обмениваться данными путем установки прямых двунаправленных каналов связи (в отличие от простых каналов, рассмотренных выше). Каждый процесс может использовать сокет для передачи данных другому процессу. Из-за своей гибкости сокеты имеют более низкую производительность. Если приложению требуется установить однонаправленную связь между двумя процессами, вместо сокета лучше воспользоваться каналом.

Существует два основных типа сокетов, которые могут использоваться процессами:

1. *потоковые сокеты* (stream socket) передают информацию в виде потоков байтов;
2. *дейтаграммные сокеты* (datagram socket) передают информацию в виде независимых фрагментов информации, называемых дейтаграммами.

Процессы, взаимодействующие друг с другом при помощи *потоковых сокетов*, оперируют в рамках модели клиент/сервер. Серверный процесс создает потоковый сокет и переходит в режим ожидания запросов связи. Клиентский процесс подключается к серверному процессу и начинает обмен информацией. Полезным свойством потоковых сокетов, в отличие от дейтаграммных, является использования TCP в качестве коммуникационного протокола, что гарантирует факт доставки всех данных.

Более быструю, хотя и менее надежную связь можно обеспечить с помощью *дейтаграммных сокетов*. Например, в некоторых распределенных системах сервер, имеющий множество клиентов, должен проводить

широковещательную передачу информации всем своим клиентам. В этом случае дейтаграммные сокеты предпочтительнее.

Список текущих реализаций Linux можно найти на Web-узле по адресу: www.linux.org.

13.7. Примеры некоторых отечественных операционных систем Linux, для решения государственных и специальных задач

В соответствии с Доктриной информационной безопасности Российской Федерации, к основным национальным интересам в информационной сфере относятся, в частности, обеспечение устойчивого и бесперебойного функционирования информационной инфраструктуры, в первую очередь критической информационной инфраструктуры Российской Федерации, а также развитие в Российской Федерации отрасли информационных технологий и электронной промышленности. Это предполагает внедрение отечественных разработок в ИТ сфере, в том числе, создание отечественного системного программного обеспечения. Новые российские ОС разрабатываются с учетом требований импортозамещения и возможности построения на их основе автоматизированных систем в защищенном исполнении. К таким продуктам относятся ОС линейки Rosa и Astra Linux.

13.7.1. Операционная система ROSA Linux

РОСА (ROSA) Linux – семейство операционных систем, разрабатываемых российской компанией «НТЦ ИТ РОСА» – «Российские операционные системы». В рамках этого семейства есть дистрибутивы различного назначения: корпоративные (ROSA Enterprise), для работы с государственной тайной (РОСА «ХРОМ»), для защиты персональных данных или конфиденциальной информации (РОСА «Кобальт»), для работы в среде виртуализации (ROSA Virtualization) и, наконец, «для домашнего пользования» (ROSA Fresh).

В корпоративном сегменте представлены две ОС:

- ROSA Enterprise Linux Server (RELS) серверная ОС (основана на ОС Red Hat Linux). Работает в редакциях для процессоров Intel 32-бит и 64-бит. Версия доступна для загрузки публично, доступ к репозиторию закрыт ключом;
- ROSA Enterprise Desktop (RED) ОС для рабочих станций (основана на Open Mandriva Linux). Доступна по запросу в редакциях для процессоров Intel 32-бит и 64-бит.

Указанные ОС не снабжены встроенными средствами защиты информации. Следовательно, обновляются более оперативно, чем сертифицированные. При организации доменов на базе ROSA Enterprise Linux Server возможно подключение клиентов на Windows 7 и выше.

Для обеспечения защищенности ОС можно настроить следующие функции:

- двухфакторная аутентификация с использованием одноразовых TOTP паролей, которые актуальны всего 30 с;
- локальная аутентификация с помощью токенов или смарт-карт Rutoken;
- установка СКЗИ Крипто-ПРО 4.0 и поддержка контейнеров ГОСТ на Rutoken.

Семейство ОС РОСА «КОБАЛЬТ» сертифицировано ФСТЭК России в настольном и серверном вариантах. Рекомендуется для использования коммерческими структурами, промышленными предприятиями и органами государственной власти, работающими с конфиденциальной информацией, включая персональные данные.

Семейство ОС РОСА «ХРОМ» сертифицировано ФСТЭК России в настольном и серверном вариантах. Рекомендуется для обработки сведений, составляющих государственную тайну с грифом не выше «секретно». Все ОС семейства имеют развитый графический интерфейс с несколькими рабочими столами (здесь они называются комнатами), файловый менеджер Dolfin, необходимый набор программ: Libre Office, торрент-клиент, два браузера, аудиоплеер, видеоплеер, редактирование видео и аудио, графический редактор, запись дисков, почтовый клиент. При необходимости нужное ПО можно установить из центра установки программ.

13.7.2. Операционная система Astra Linux

Astra Linux – дистрибутив, основанный на Debian Linux, разработанный российской компанией «РусБИТех» («Русские базовые информационные технологии») главным образом для нужд российских силовых ведомств и спецслужб. Производителем разработана базовая версии Astra Linux – Common Edition (общего назначения) и ее модификация Special Edition (специального назначения), разработанная совместно со специалистами Академии ФСБ.

Версия «общего назначения» – «Орел» (Common Edition) предназначена для «решения задач среднего и малого бизнеса». Распространяется свободно.

Версия «специального назначения» – «Смоленск» (Special Edition) предназначена для создания на ее основе автоматизированных систем в защищенном исполнении, обрабатывающих информацию со степенью секретности до «совершенно секретно» включительно. Новые версии выходят с периодичностью 1 год. Сертифицирована в системах сертификации средств защиты информации Минобороны, ФСБ и ФСТЭК России по последним нормативным актам. Включена в Единый реестр российских программ для электронно-вычислительных машин и баз данных Минкомсвязи России.

В состав дистрибутива Astra Linux SE входят пакеты с открытым исходным кодом: офисный пакет Libre Office, браузер Firefox, почтовый клиент Thunderbird, редактор растровой графики GIMP, проигрыватель муль-

тимедиа VLC и другие. В последние версии включен отечественный пакет офисных приложений «Мой Офис». Графический интерфейс больше сходен с ОС Windows, чем с Debian 7. В отличие от большинства ОС общего назначения, использующих дискреционную модель доступа к данным, Astra Linux реализует также мандатную модель доступа.

На основе ОС Astra Linux SE развернуты и функционируют десятки информационных систем, как в государственных, так и в коммерческих структурах. Система включена в стандарты «Росатома», «Ростеха», принята на снабжение Вооруженных Сил Российской Федерации, активно внедряется другими органами государственного управления, ведомствами и учреждениями. Среди них, например, такие крупные, как защищенная платформа для государственной автоматизированной системы гособоронзаказа.

В целях создания доверенных средств вычислительной техники на основе отечественной электронной компонентной базы и отечественной ОС, компании «НПО РусБИТех» и АО «МЦСТ» (Московский центр спарктехнологий) проводят совместные работы по обеспечению функционирования операционной системы «Astra Linux Special Edition» на вычислительных комплексах «Эльбрус», использующих микропроцессоры с одноименной архитектурой. В ходе работ достигнуты все поставленные цели: успешно осуществляется процесс сборки ОС с использованием отечественного компилятора для платформы Эльбрус, проведена серия успешных запусков ОС «Astra Linux Special Edition», в т.ч. графического пользовательского интерфейса. Для ускорения процесса разработки были использованы наработки и компетенции АО «МЦСТ», полученные при создании собственной ОС «Эльбрус». Для идентификации этого варианта ОС Astra Linux Special Edition в честь 75-летия прорыва советскими войсками кольца блокады Ленинграда принято решение о присвоении ему имени города-героя – «Ленинград».

Таким образом, разработана полностью отечественная программная экосистема на базе ядер Linux, включая собственный BIOS и средства разработки.

К недостаткам отечественных ОС специального назначения, как Astra Linux, так и рассмотренной ранее ОС Роса-Linux, можно отнести то, что обновления ОС включаются только в сертифицированные версии, а это происходит редко, гораздо реже, чем появляются новые угрозы.

14. Сетевая операционная система реального времени QNX для управления технологическими процессами

14.1. Общие понятия об операционных системах реального времени

Операционные системы реального времени (ОСРВ) – на англ. Real-Time OS (RTOS) применяются в системах промышленной автоматики и автоматизированных система управлении технологическими процессами (АСУ ТП), бортовых системах управления, радио- и робототехнике и потребительской электронике. ОСРВ управляет системами в автоматическом режиме, без контроля со стороны человека, и используется для разработки встраиваемого ПО при создании устройств на основе микроконтроллеров. Последнее время наиболее активно развивающимся приложением систем реального времени является Интернет вещей (системы «умного» дома, «умного города», промышленных «умных» устройств).

Среди ОС реального времени наиболее известны коммерческие QNX, Micrium (uC/OS), WindRiver (VxWorks) и свободно распространяемая FreeRTOS. Все перечисленные ОСРВ поддерживают многозадачность и являются POSIX-совместимыми, что обеспечивает переносимость прикладных программ на уровне исходного кода. Интерес представляет также Windows-совместимая RTOS-32.

QNX (QNX Neutrino RTOS) – проприетарная ОС «жесткого» реального времени канадской фирмы QNX Software Systems, предназначенная для использования в автомобильных, медицинских, транспортных, военных и промышленных встроенных системах. QNX основана на концепции микроядра с вытесняющей приоритетной многозадачностью и модульной архитектурой. Благодаря микроядерной архитектуре, каждый драйвер, стек протоколов, файловая система и приложение работают в своей защищенной области памяти, поэтому в случае сбоя практически любой компонент может быть перезапущен автоматически, не затрагивая другие части системы. В качестве основных достоинств QNX, ее разработчики отмечают высокую производительность и предсказуемость, поддержку широкого спектра аппаратных платформ, в том числе многоядерных, надежность и возможности самовосстановления микроядра. Система поддерживает такие механизмы безопасности, как управление уровнями системных привилегий, контроль доступа, использование песочницы, контроль переполнения и защита памяти, шифрование на уровне файловой системы, обнаружение аномалий и контроль целостности. Кроме того, QNX реализует стек протоколов TCP/IP и встраиваемый браузер, поддерживает большое число различных файловых систем и предоставляет оконный графический интерфейс с поддержкой 3D-графики и мультимедиа. Более подробно QNX будет рассмотрена далее.

uC/OS (μC/OS) компании Micrium – полнофункциональная ОСРВ с поддержкой графического интерфейса пользователя, стека протоколов TCP/IP, протоколов связи для промышленных устройств, USB, файловой системы FAT12/16/32 с дополнительными модулями журналирования и оптимизации работы с SSD-накопителями. Она включает ядра μC/OS-II и μC/OS- III объемом от 6 до 24 Кбайт кода, требующие для работы чуть больше 1 Кбайт памяти. Дополнительно могут быть поставлены модули, реализующие протокол SSL, инфраструктуру для разработки Java-приложений, реляционную базу данных и собственную отказоустойчивую файловую систему. μC/OS доступна для большого числа платформ и допускает настройку объема требуемой системной памяти за счет выбора необходимых функций. Лицензия Micrium предусматривает бесплатное распространение несколько урезанной μC/OS для личного использования и обучения.

RTOS-32 – операционная система «жесткого» реального времени с открытым исходным кодом компании On Time Informatik, предназначенная для устройств на базе архитектуры x86. Ядро RTOS-32 реализует подмножество Win32 API с расширениями реального времени, что позволяет переносить в нее прикладной код, разработанный в стандартной среде Microsoft Visual C/C++, Borland Delphi или Borland C/C++ в ОС Windows. Имеет встроенную поддержку многопроцессорности и многоядерных процессоров, сетей TCP/IP, файловую систему exFAT и USB 3.0.

Одной из наиболее популярных ОСРВ является свободно распространяемая FreeRTOS с открытым исходным кодом. FreeRTOS разрабатывалась с 2003 года в партнерстве с ведущими мировыми компаниями производителями микросхем и является де-факто стандартным решением для микроконтроллеров и небольших микропроцессоров. В 2017 г. управление проектом FreeRTOS передано компании Amazon Web Services (AWS), которая занимается развитием приложений FreeRTOS для Интернета вещей с использованием собственных облачных сервисов. Основными достоинствами FreeRTOS является ее легкость и минимальные требования к аппаратным ресурсам при достаточно широких функциональных возможностях. Исходный код ядра FreeRTOS описывается всего тремя файлами на языке С, а в двоичном представлении занимает от 6 до 12 Кб в зависимости от типа платформы и настроек ядра. FreeRTOS поддерживает более 35 архитектур и может быть портирована на новые платформы. Она хорошо документирована, и, что немаловажно для производителей, может свободно использоваться в коммерческих продуктах без требования раскрывать собственный исходный код. FreeRTOS поддерживает три типа многозадачности: вытесняющую, кооперативную и гибридную и имеет развитые средства синхронизации. Для FreeRTOS доступно множество библиотек, включая поддержку графики, сетевых протоколов, функций безопасности и др.

Работа по созданию ОСРВ ведется и в нашей стране. Так, в 2012 г. компанией Эремекс выпущена ОСРВ для встраиваемых систем с ограниченными ресурсами FX-RTOS.

В 2017 г. компания АстроСофт представила продукт ОСРВ МАКС (операционная система реального времени для мультиагентных когерентных систем), базирующийся на микроядерной архитектуре и предназначенный в текущей реализации для работы на недорогих микроконтроллерах с ограниченными ресурсами.

Главной особенностью ОСРВ МАКС является поддержка на уровне ядра возможностей по организации взаимодействия множества устройств на основе концепции распределенной общей памяти. Используя общий контекст, набор параметров, несколько независимых устройств могут обмениваться данными и синхронизировать их так, как будто все они имеют физический доступ к общей памяти. ОСРВ МАКС поддерживает популярные иностранные аппаратные платформы, а также всю линейку 32-разрядных микроконтроллеров и процессоров АО «ПКК Миландр», одного из ведущих российских разработчиков интегральных микросхем различного функционального назначения. ОСРВ МАКС является оригинальной отечественной разработкой и включена в Единый реестр российского ПО.

Еще один пример – новая бортовая ОСРВ JetOS, предназначенная для использования на пассажирских самолетах МС-21 и воздушных судах «Суперджет» и призванная заменить используемые в настоящее время зарубежные системы. Основными особенностями ОСРВ JetOS является поддержка актуальных версий авиационных стандартов, поддержка работы на многоядерных процессорах и возможность портирования на различные аппаратные платформы.

Далее более подробно будут рассмотрены особенности ОСРВ QNX, во-первых, как типичного представителя ОСРВ, во-вторых, как ОСРВ получивший широкое применение в промышленных, военных и робототехнических системах.

14.2. Назначение и архитектура QNX

Основным назначением операционной системы QNX является реализация программного интерфейса POSIX в масштабируемой, отказоустойчивой форме, подходящей для широкого круга открытых систем, начиная от небольших встроенных систем с ограниченными ресурсами и заканчивая крупными распределенными вычислительными средами. Данная ОС поддерживает несколько семейств процессоров, в том числе x86, ARM, XScale, PowerPC, MIPS и SH-4.

Операционная система QNX идеально подходит для приложений реального времени. Она может быть масштабирована до компактных конфигураций и способна работать в многозадачном режиме, управлять потоками, осуществлять планирование процессов по приоритетам и выполнять быстрое переключение контекстов. Более того, операционная система предоставляет все эти возможности посредством программного интерфейса, основанного на стандартах POSIX.

Рис. 14.1. Основные сферы применения ОС QNX

На базе ОС QNX возможно:

- создавать системы, работающие в режиме реального времени и способные к самовосстановлению – в QNX любой компонент в случае отказа может быть перезапущен динамически, не нарушая работу микроядра и других компонентов.
- использовать одну и ту же ОС во всей своей линейке продуктов – благодаря исключительной модульности QNX, любые уже испытанные и проверенные компоненты – драйверы, приложения, дополнительные сервисы ОС, возможно использовать повторно в других продуктах;
- производить программное и аппаратное обновление систем «на лету» - поскольку любой компонент в QNX может быть добавлен или удален динамически, система может продолжать работать даже в процессе замены или добавления в нее новых приложений, драйверов или стеков протоколов.

14.2. Архитектура QNX

Высокая производительность, отказоустойчивость и универсальность достигаются в ОС QNX благодаря следующим фундаментальным принципам, заложенным в основу ее архитектуры:

1. микроядерная архитектура;
2. взаимодействие отдельных процессов на основе «прозрачного» межмашинного и межсетевого обмена сообщениями;
3. использование администратора высокой готовности для быстрого восстановления отказавших компонентов ОС;
4. гибкая система управления задачами, их приоритетностью, а также реализация режимов «жесткого» реального времени;
5. изначальная поддержка многопроцессорных систем и распределенных вычислений.

14.2.1. Микроядро

Микроядерная операционная система построена на основе миниатюрного ядра, обеспечивающего минимальные службы для произвольной группы взаимодействующих процессов, которые, в свою очередь, обеспечивают функциональность более высокого уровня. ОС QNX строится на основе компактного микроядра, способного управлять группой взаимодействующих процессов (рис. 14.2).

Рис. 14.2. Обмен сообщениями
между структурными элементами ОС

В QNX ядро отвечает только за базовые примитивы ОС (сигналы, таймеры, планирование, и т.п.). Все остальные компоненты системы: драйверы, файловые системы, стеки протоколов, пользовательские приложения – выполняются вне пределов ядра как отдельные пользовательские процессы, каждый в своем защищенном адресном пространстве. Такая схема обладает исключительной «встроенной» отказоустойчивостью.

В QNX любой компонент ОС в случае отказа может быть перезапущен динамически, не нарушая работу микроядра и других компонентов. Например, если драйвер попытается обратиться к памяти за пределами своего адресного пространства (что для большинства ОС является фатальной ошибкой), ядро корректно завершит этот драйвер и освободит все занятые им ресурсы. В дальнейшем можно автоматически перезапустить этот драйвер, используя администратор систем высокой готовности QNX.

Благодаря исключительной модульности QNX, любые уже ранее разработанные компоненты: драйверы, приложения, дополнительные сервисы ОС – можно использовать повторно в других системах. Фактически, можно создать универсальный набор модулей, а затем применять его либо в однопроцессорном устройстве, либо в SMP-системе, либо в вычислительном кластере. Кроме того, высокая модульность, позволяет произво-

дить апгрейды «на лету» – поскольку любой компонент в QNX может быть добавлен или удален динамически, система может продолжать работать даже в процессе замены или добавления в нее новых приложений, драйверов или стеков протоколов.

14.2.2. Межзадачное взаимодействие

Для того чтобы осуществить выполнение нескольких потоков одновременно в многозадачной операционной системе реального времени, эта ОС должна иметь механизмы обеспечения взаимодействия потоков между собой. Все компоненты QNX используют для общения друг с другом единый, четко детерминированный механизм – обмен сообщениями. Он образует между компонентами системы виртуальную «программную шину», позволяющую подключать к ней или, наоборот, отключать любой компонент «на лету». Сообщения могут свободно передаваться между узлами вычислительной сети, предоставляя прозрачный доступ к любому ресурсу, где бы он ни находился (рис. 14.3).

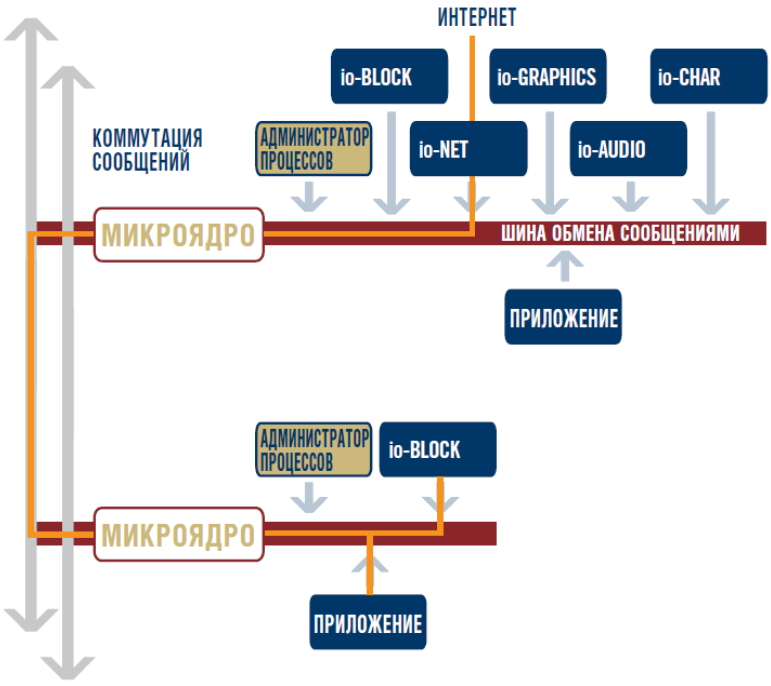


Рис. 14.3 Обмен сообщениями между отдельными узлами вычислительной сети

QNX изначально разрабатывалась как сетевая ОС. В некотором смысле, компьютерная сеть, построенная на основе QNX, больше напоминает единую кластерную ЭВМ, чем набор индивидуальных микрокомпьютеров, В распоряжении пользователей имеется огромный набор ресурсов, которые могут быть применены в любом приложении. Однако, в отличие от универсальной ЭВМ, ОС QNX является гибкой средой, так как на любом ее узле может быть предоставлен необходимый объем вычислительных мощностей в соответствии с потребностями каждого пользователя.

В отличие от монолитных систем, QNX построена на основе принципа эффективного взаимодействия, который является ключом к эффективному функционированию. Таким образом, механизм обмена сообщениями – это краеугольный камень архитектуры микроядра. Он увеличивает эффективность всех транзакций, происходящих между всеми процессами во всей системе независимо от используемой среды передачи, будь это простое прямое соединение между компьютерами или сеть Интернет.

Обмен сообщениями в QNX эффективен, поскольку в QNX обмен сообщениями – это больше, чем просто форма межзадачного взаимодействия. Это один из фундаментальных механизмов, делающих системы на основе QNX столь гибкими в разработке. Этот механизм, в частности:

- автоматически синхронизирует выполнение взаимодействующих компонентов ОС;
- избавляет приложения от необходимости следить за очередностью доставки данных;
- позволяет программистам разбить сложное приложение на четко разграниченные функциональные блоки, которые можно разрабатывать и тестировать по отдельности;
- действует по всей сети, предоставляя приложениям пользователя прозрачный доступ к сервисам и ресурсам удаленных узлов.

Обмен сообщениями в QNX происходит непосредственно между отправителем и получателем. Соответственно, не происходит никакого промежуточного копирования или кэширования данных, и не требуется дополнительных действий по синхронизации. Кроме того, нет никакой необходимости реализовывать дополнительные уровни для обработки сложных сообщений. Чтобы обмениваться сообщениями с системными сервисами, приложения могут использовать стандартные вызовы POSIX.

В QNX каждая программа, предоставляющая некий сервис (например, драйвер), может зарегистрировать в пространстве имен путей «файл» или «каталог». Потом любое приложение может соединиться с этим драйвером, применив к такому файлу или каталогу стандартную операцию open(). Результатом будет обычный файловый дескриптор, через который приложение сможет обращаться к сервисам драйвера при помощи вызовов POSIX, предназначенных для работы с файловыми дескрипторами – read(), write(), lseek() и т.п. Библиотека языка Си автоматически преобразует эти вызовы в соответствующие сообщения и передает их драйверу. Например, когда приложение вызывает функцию read(), чтобы считать готовые данные, библиотека преобразует этот вызов в сообщение «запрос на чтение». Фактически, в QNX приложения используют обмен сообщениями каждый раз, когда работают с файловыми дескрипторами или указателями на файлы. Такой подход позволяет:

- упростить обслуживание систем – поскольку пространство путей четко отделяет сервисы от клиентских приложений, обновление систем становится элементарной задачей. Любой сервис можно

заменить на его новую версию, в том числе в процессе эксплуатации – и пользователи сами автоматически найдут его.

- расширять ОС для нестандартных задач – QNX предоставляет разработчикам библиотеку администратора ресурсов, которая позволяет системным программам регистрировать свои имена в пространстве путей и обрабатывать запросы от клиентских приложений.

В дополнение ко всему, в QNX любые системные сервисы, включая драйверы, являются программами пользовательского уровня, а значит, разрабатываются точно также, как и любые другие приложения. В результате разработчик получает возможность легко расширить ОС совершенно новыми, специфичными возможностями.

14.2.3. Администратор систем высокой готовности

Микроядерная архитектура и исполнение всех остальных компонентов ОС на пользовательском уровне позволяет формировать системы, устойчивое к отказам отдельных компонент. При этом, принятый в QNX подход к обеспечению высокой готовности очень прост – время, расходуемое на перезапуск одного компонента, по определению гораздо меньше времени перезагрузки всей системы. Например, драйвер или стек протоколов, в котором возникла проблема, может быть немедленно выгружен и перезапущен, зачастую за единицы миллисекунд. А на перезагрузку всей системы потребовались бы единицы секунд. Именно такой подход к изоляции сбоев позволяет QNX обеспечивать гораздо меньшее время восстановления, чем у других ОС.

Для обеспечения высокой отказоустойчивости в QNX применяется *администратор систем высокой готовности*, позволяющий системе в случае сбоя самовосстанавливаться автоматически.

Администратор систем высокой готовности обеспечивает:

- уведомления об отказах – в администраторе систем высокой готовности реализован механизм квитанций работоспособности, следящий за состоянием каждого компонента ОС и позволяющий обнаруживать отказы на самой ранней стадии. Если администратор обнаруживает определенное стечение обстоятельств или отказ, он автоматически немедленно оповещает об этом другие компоненты;
- настраиваемые сценарии восстановлений – используя библиотеку администратора систем высокой готовности, приложение может явно указать администратору, какие действия по его восстановлению и в каком порядке нужно предпринять в случае сбоя.
- автоматическое восстановление соединений – администратор систем высокой готовности предоставляет клиентскую библиотеку, которая позволяет системе в случав отказа восстановить прерван-

ные соединения. Эта библиотека содержит замену для стандартных функций ввода/вывода из библиотеки языка С.

- «посмертный анализ» – если процесс завершается некорректно, администратор систем высокой готовности может сохранить его образ для последующей обработки. Анализируя этот образ, возможно определить, какая команда вызвала сбой, а также узнать содержимое переменных, чтобы точно определить, что именно произошло.

Администратор систем высокой готовности обладает способностью к самовосстановлению и поэтому устойчив к внутренним сбоям. Если он по какой-либо причине завершается некорректно, он полностью восстанавливает свое предыдущее состояние.

14.2.4. Управление процессами и реализация прерываний

QNX обеспечивает высокую производительность в реальном масштабе времени, поскольку в ней реализованы:

- сверхмалые задержки обработки прерывания и переключения контекста – с временем переключения контекста в 600 нс, QNX позволяет обеспечить максимум производительности вычислительной аппаратуры;
- распределенный механизм наследования приоритетов – в QNX драйверы, файловые системы и прочие сервисы могут выполняться с приоритетом процесса, запросившего обслуживание, даже если он расположен на другом узле сети. Такое наследование приоритетов при обмене сообщениями дает гарантию, что задача, выполняемая по заказу низкоприоритетного процесса, всегда будет вытеснена задачей от высокоприоритетного процесса. Инверсия приоритетов исключается.
- свобода выбора дисциплины планирования потоков – QNX не просто предоставляет несколько дисциплин планирования, она позволяет назначать каждому процессу свою дисциплину;
- гарантированная доступность процессора для процессов с жестким графиком выполнения (процессов реального времени) – возможно назначать лимит времени выполнения для процессов в пределах определенного интервала. В результате, процессы будут готовы обработать нерегулярно (асинхронно) возникающие события, не рискуя нарушить график выполнения других процессов и потоков. Эта дисциплина особенно полезна при реализации в системе, обрабатывающей одновременно периодические и апериодические события;
- автоматическая синхронизация системных компонентов – синхронизация, предоставляемая механизмом обмена сообщениями в QNX, значительно упрощает реализацию поведения системы в ре-

альном времени. Во многих других ОС такое поведение приходится реализовывать при помощи двухуровневого планирования и с большими накладными расходами.

- вложенные прерывания – QNX предоставляет поддержку вложенных прерываний для процессов, в сочетании с фиксированной верхней границей времени реакции ОС.

14.2.5. Поддержка симметричных многопроцессорных систем

QNX в полной мере поддерживает симметричные мультипроцессорные системы (SMP) – любой поток любого процесса в этой ОС можно запланировать на выполнение на любом из доступных процессоров.

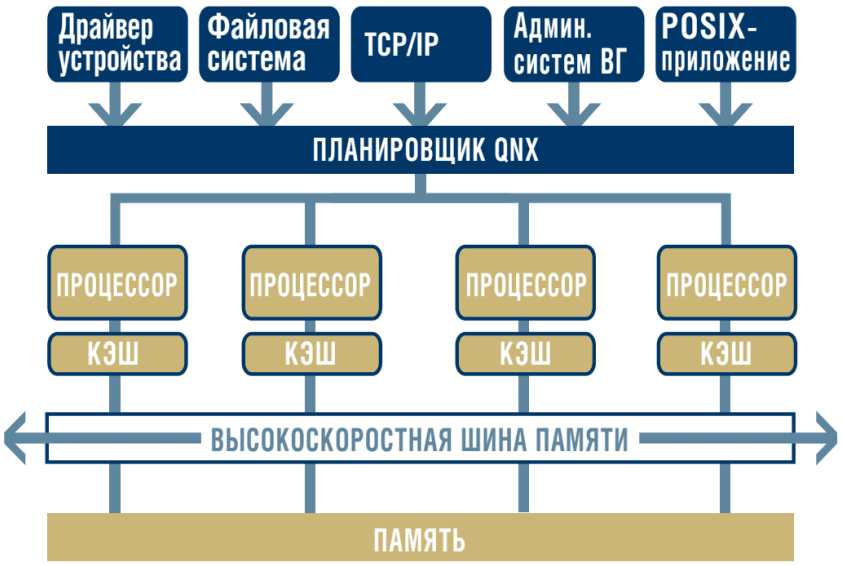


Рис. 14.4. Реализация поддержки симметричных многопроцессорных систем

Встроенная «прозрачная» поддержка SMP позволяет:

- настраивать производительность, используя родственность процессоров – для оптимизации использования процессорного кэша. QNX пытается запланировать поток на процессоре, где он выполнялся в последний раз, если это допустимо. Чтобы иметь возможность дополнительно оптимизировать работу кэша, микроядро QNX предоставляет «маску родственности» процессоров, позволяя закрепить поток за одним или несколькими выбранными процессорами;
- использовать максимум возможностей каждого процессора – поскольку QNX может запланировать любой поток на любом процессоре, все процессоры можно использовать по максимуму, получая наибольший выигрыш в производительности. При этом, в отличие от традиционных ядер ОС, чтобы поддерживать SMP, микроядру QNX не требуется множества модификаций кода,

снижающих производительность т.к. микроядро с поддержкой SMP всего на несколько килобайт больше стандартного;

- строить отказоустойчивые кластеры, обладающие высокой вычислительной мощностью – комбинируя SMP-возможности QNX с ее встроенным механизмом распределенных вычислений, можно легко конструировать массивные отказоустойчивые кластеры, включающие в себя сотни одно- и многопроцессорных систем.

14.2.6. Поддержка распределенных вычислений

Поддержка SMP, а также механизм «прозрачного» межсетевого обмена сообщениями позволяет реализовать гибкую поддержку распределенных вычислений. Чтобы обращаться к сервисам на удаленных узлах, приложению не нужно выполнять каких-либо специальных действий. Для этого можно посылать такие же сообщения, которые использовались бы для доступа к локальному сервису, и эти сообщения будут автоматически перенаправлены по сети на нужный узел. Таким образом, любые удаленные ресурсы – диски, сетевые адаптеры, стеки протоколов, и т.п., становятся доступны так, как будто они находятся на локальной машине.

Используя распределенные возможности QNX, возможно:

- уменьшить затраты на оборудование – при использовании распределенных вычислений узлы сети могут совместно использовать ресурсы вместо их дублирования. Например, если на одном узле расположена большая файловая система в ППЗУ другим узлам иметь такую же не обязательно – они смогут использовать файловую систему того узла. на котором она уже есть. Аналогично, если на одном узле уже запущен стек TCP/IP, все остальные узлы смогут использовать этот узел как TCP/IP-шлюз. исключая необходимость в настройке нескольких IP-адресов.
- добавлять вычислительную мощность без дополнительных разработок – чтобы добавить в распоряжение приложения больше вычислительных мощностей или больше физических интерфейсов, достаточно вставить дополнительную процессорную карту или добавить в сеть еще один компьютер. Приложения на уже имеющихся узлах смогут пользоваться ресурсами нового узла без внесения в них каких-либо изменений.
- упростить проектирование отказоустойчивых кластеров – поскольку обмен сообщениями в QNX предоставляет прозрачный доступ к сервисам вне зависимости от их местоположения, приложения могут полностью абстрагироваться от принятия решений о том, как будут обрабатываться запросы от клиента, где этот сервис расположен, и есть ли другие сервисы, способные обработать этот запрос (например, в случае дублирования сервиса на нескольких узлах для обеспечения отказоустойчивости или балансировки нагрузки).

- увеличить пропускную способность сети резервированными соединениями – в QNX сообщения могут передаваться по нескольким соединениям одновременно, увеличивая пропускную способность и повышая надежность связи. Например, при отказе одного из соединений QNX может перенаправить поток данных по одному или нескольким альтернативным маршрутам. Возможно также настроить QNX на балансировку сетевого трафика между всеми доступными соединениями, повысив тем самым суммарную пропускную способность.

14.3. Файловые системы

В традиционных ОС файловые системы встроены в ядро. В QNX файловые системы расположены вне пределов ядра и выполняются в отдельных защищенных областях памяти как пользовательские процессы. В результате, вы можете запустить, остановить или обновить поддержку той или иной файловой системы «на лету», без необходимости в перезагрузке.

В дополнение, несколько файловых систем: дисковая, встраиваемая в ППЗУ, CD-ROM, CIFS и т.д. – могут выполняться одновременно на одной и той же целевой системе. Они даже могут работать совместно, расширяя возможности друг друга. Например, файловая система со сжатием может работать совместно со встраиваемой файловой системой, существенно снижая потребности устройства в объеме ППЗУ.

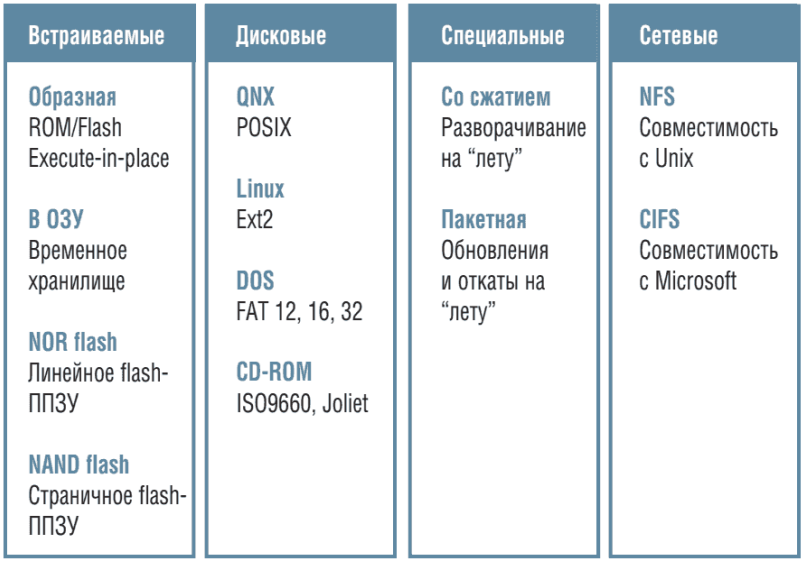


Рис. 14.5. Реализация поддержки файловых систем

14.4. Поддержка сетевых протоколов

В QNX все сетевые сервисы выполняются вне пределов ядра как отдельные процессы с защищенными адресными пространствами. Поэтому можно запускать, останавливать или обновлять любой драйвер или протокол «на лету». Кроме того, приложение может пользоваться любым сете-

вым сервисом через один и тот же стандартный POSIX API, применяемый для работы со всеми остальными сервисами. Поддержка сетевых протоколов позволяет:

- сочетать любое количество сетевых протоколов, включая TCP/IP и распределенную сеть QNX;
- создавать многочисленные виртуальные сети, запуская несколько копий стека TCP/IP на одном и том же физическом интерфейсе;
- использовать богатую базу сетевого кода «третьих» производителей, основанного на POSIX и BSD API.

14.5. Драйвера устройств

QNX содержит множество готовых драйверов для различных плат и периферийных устройств. Однако, если необходимо написать драйвер для нестандартного устройства, это реализуется за счет использования библиотеки администратора ресурсов.

Рис. 14.6. Использование пакетов драйверов (DDK)

Библиотека администратора ресурсов предоставляет простой интерфейс для регистрации драйвера в пространстве имен путей и обработки запросов от клиентов. Однако, применение этой библиотеки не ограничивается только драйверами – упрощая процессы установления и разрыва соединений она помогает ускорить разработку любого сервиса с которым работают множество клиентских приложений.

Такой подход к «конструированию» системы драйверов, позволяет:

- использовать пакеты разработки драйверов (DDK) для ускорений разработки – для упрощения разработки драйверов в состав комплекта разработчика QNX Momentics входят пакеты разработки драйверов (DDK) для устройств различного типа, включая аудио-, графические и сетевые адаптеры, терминальные устройства, устройства ввода, принтеры и USB-устройства. В состав пакетов

входит детальная документация, исходные тексты, а также готовый программный каркас, в котором весь высокоуровневый аппаратно-независимый код уже реализован;
- отлаживать драйверы на уровне исходного текста, используя обычный инструментарий – поскольку драйверы и прочие администраторы ресурсов выполняются как обычные пользовательские процессы, отлаживать их можно при помощи тех же интегрированных инструментов разработки приложений, которые входят в состав QNX Momentics;
- тестировать новые драйверы без перезагрузки – можно тестировать изменения в коде драйверов без перезагрузки системы, и даже не начиная новую отладочную сессию, просто перекомпилируя и перезапуская драйвер. Мало того, поскольку все драйверы представляют собой обычные процессы, возможно тестировать и отлаживать их прямо на инструментальном компьютере, еще до того, как будет готова целевая аппаратура;
- программировать многопогочные сервисы без написания больших объемов кода – чтобы сделать многопоточные драйверы проще и меньше по объему, библиотека администратора ресурсов включает в себя функции управления пулами потоков, которые автоматически регулируют число потоков в системе в зависимости от загрузки. Драйвер, использующий пулы потоков, изначально поддерживает SMP и может легко масштабироваться в многопроцессорных системах.

14.6. Интегрированный комплекс разработчика

Комплекс разработчика QNX Momentics поддерживает различные языки программирования, различные инструментальные ОС и различные целевые процессоры. Полнофункциональный и в то же время простой в использовании, он позволяет ускорить разработку нового проекта, вне зависимости от его сложности и масштаба.

В состав комплекта входят:
- тесно интегрированный инструментарий для анализа памяти, профилирования кода, анализа событий трассировки, управления версиями. удаленной диагностики, генерации графических интерфейсов. и т.д.;
- мастера, позволяющие экономить время при создании новых проектов, оптимизации целевых образов и организации сессий удаленной отладки;
- богатый выбор пакетов поддержки процессорных плат, библиотек и исходных текстов драйверов различного типа.

Используя QNX Momentics, можно писать на C, C++, встраиваемом C++ или Java, работать в среде Windows, Solaris или QNX, и компилировать код для процессоров ARM, MIPS. PowerPC. SH-4, Strong ARM.

XScalE или x86 – и все это из одной и той же IDE. Существует даже возможность работать с различными языками и процессорами одновременно.

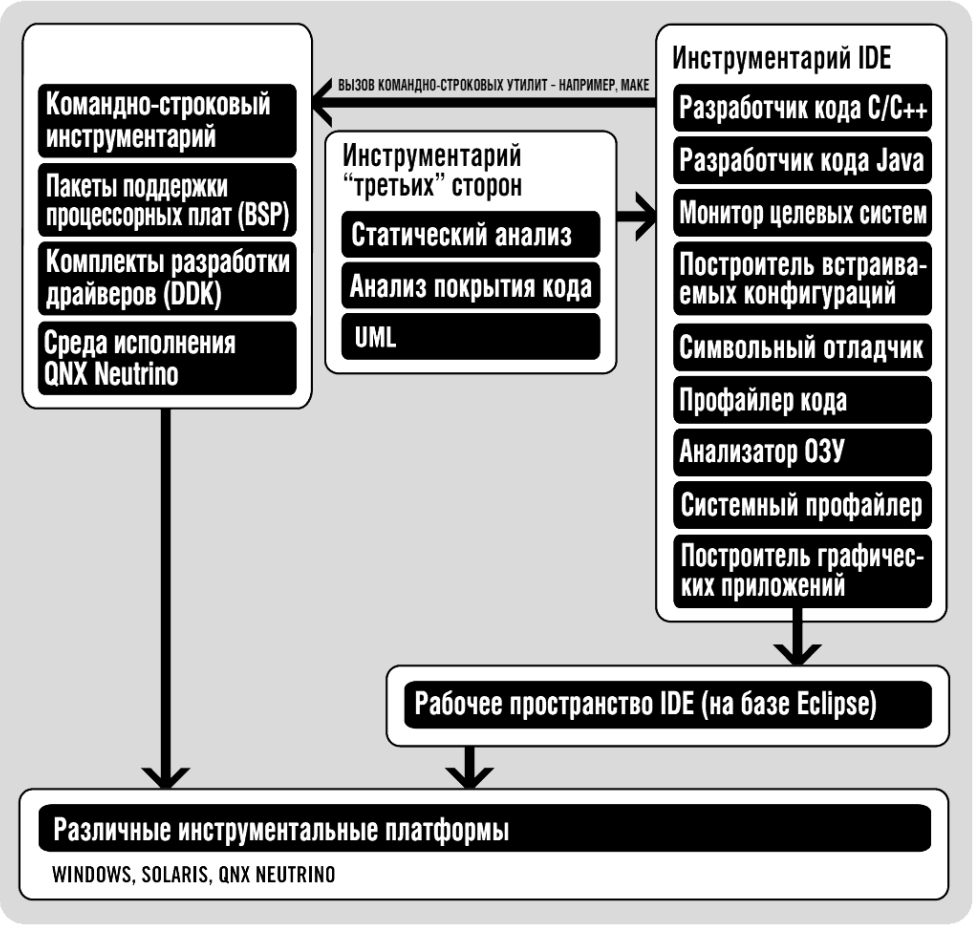


Рис. 14.7. Структура комплекса разработчика QNX Momentics

14.7. Графический интерфейс пользователя Photon

В состав QNX включен Photon micro GUI – модульная графическая оболочка, способная обеспечить даже самому маленькому встраиваемому устройству профессиональный, современный графический интерфейс. По аналогии с самой QNX, Photon основан на компактном микроядре и предоставляет большинство своих сервисов посредством опциональных процессов, работающих в защищенных областях памяти.

В отличие от функционально ограниченных графических библиотек, традиционных для других ОС, Photon представляет собой оконную систему, имеющую следующие возможности:

- строить сложные многослойные дисплеи – Photon предоставляет высокотехнологичные функции типа внеэкранной растеризации, перекрытии изображений, канала прозрачности, подстановки текстуры и прямого режима. В результате, можно создавать работающие многослойные дисплеи, способные отображать комбинацию из графики и живого видео – идеальное решение для систем

динамической навигации, телеприставок с функцией «картинка в картинке», и т.п.;

- создавать уникальный облик – используя механизм, называемый стилями виджетов (widgetstyles), можно настраивать внешний вид кнопок, меню, окон и других элементов интерфейса;
- отображать и вводить текст на нескольких языках одновременно – Photon поддерживает стандарт Unicode;
- отображать высококачественные шрифты на дисплеях любого размера – администратор шрифтов Photon поддерживает множество шрифтовых форматов, включая растровые и TrueType;
- подключать различные мультимедийные форматы - расширяемая архитектура медиапроигрывателя Photon предоставляет готовую поддержку множества форматов данных, включая CD-аудио, MP3, потоки MPEG-1, WAV; AIFF, IFF; AU и прочие;
- обновлять графический интерфейс «на лету» – в Photon большинство графических сервисов (видеодрайверы, менеджеры окон, драйверы устройств ввода и т.п.) реализованы в виде плагинов. Таким образом, можно динамически изменять практически любой компонент интерфейса без перезагрузки;
- создавать полноценные интерфейсы, не написав ни единой строчки кода – для упрощения разработки графических интерфейсов, Photon поддерживает визуальное средство разработки.

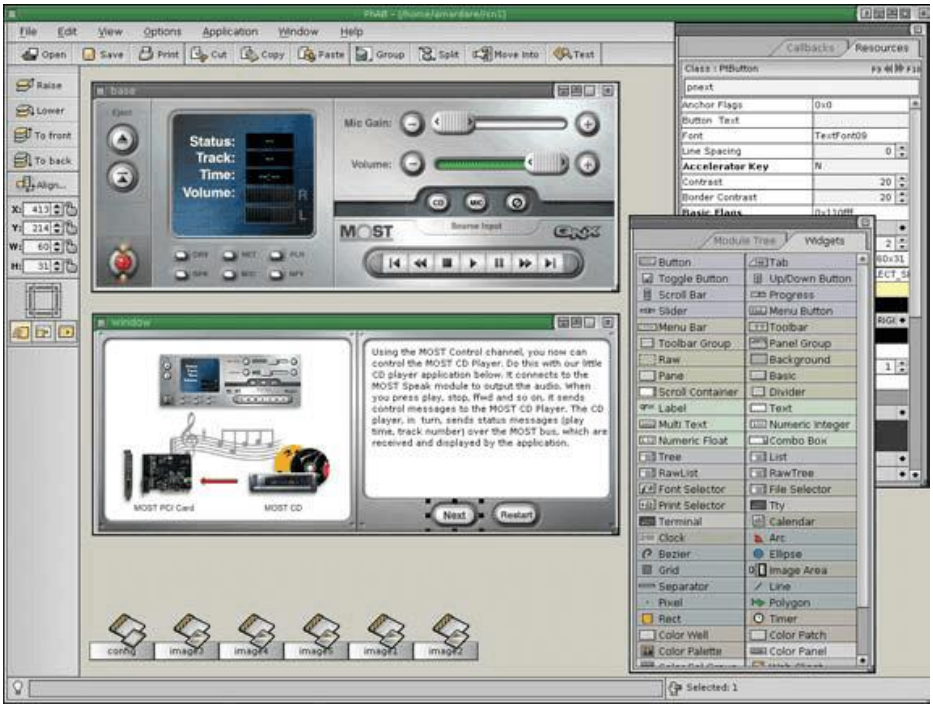


Рис. 14.8. Графический интерфейс пользователя Photon

РАЗДЕЛ 2. ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ В ТЕЛЕКОММУНИКАЦИОННЫХ СЕТЯХ

15. Типовые информационно-технические воздействия на телекоммуникационные сети

Телекоммуникационные сети (ТКС) постоянно подвергаются информационно-техническим воздействия (ИТВ) со стороны профессиональных и непрофессиональных злоумышленников.

Атакующие ИТВ ориентированы на непосредственное воздействие на информацию, системы ее сбора, передачи, хранения, обработки и представления, а также на используемые в этих системах информационные технологии, как правило, с целью снижения уровня ИБ или эффективности функционирования. Применение атакующих ИТВ против ТКС направлено на срыв выполнения системой своих целевых задач – передачи информации с заданным качеством.

Далее обзорно представлена классификация и основные типы атакующих ИТВ. Более подробная информация об атакующих ИТВ представлена в работе [7].

15.1. Классификация атакующих информационно-технических воздействий со стороны злоумышленников

Классификация средств и способов атакующих ИТВ представлена на рис. 15.1.

Атакующие ИТВ, в зависимости от их ориентированности на нарушение конкретного свойства ИБ, можно классифицировать на четыре основных типа:

- ориентированные на нарушение конфиденциальности информации;
- ориентированные на нарушение целостности информации;
- ориентированные на нарушение доступности информации;
- ориентированные на информационно-психологическое воздействие (компьютерное психотронное воздействие) на пользователей информационной системы.

По способу реализации ИТВ классифицируются на:

- алгоритмические:
    - эксплойты, ориентированные на управляющую программу информационной системы (ядро или модули операционной системы, драйвера, BIOS);

- эксплойты, ориентированные на прикладные программы информационной системы (пользовательские приложения, серверные приложения, сетевые приложения, браузеры);
- эксплойты, ориентированные на сетевые протоколы информационной системы;
- эксплойты, ориентированные на перевод информационной системы или управляемой ею технологической системы в нештатные или технологически опасные режимы функционирования;

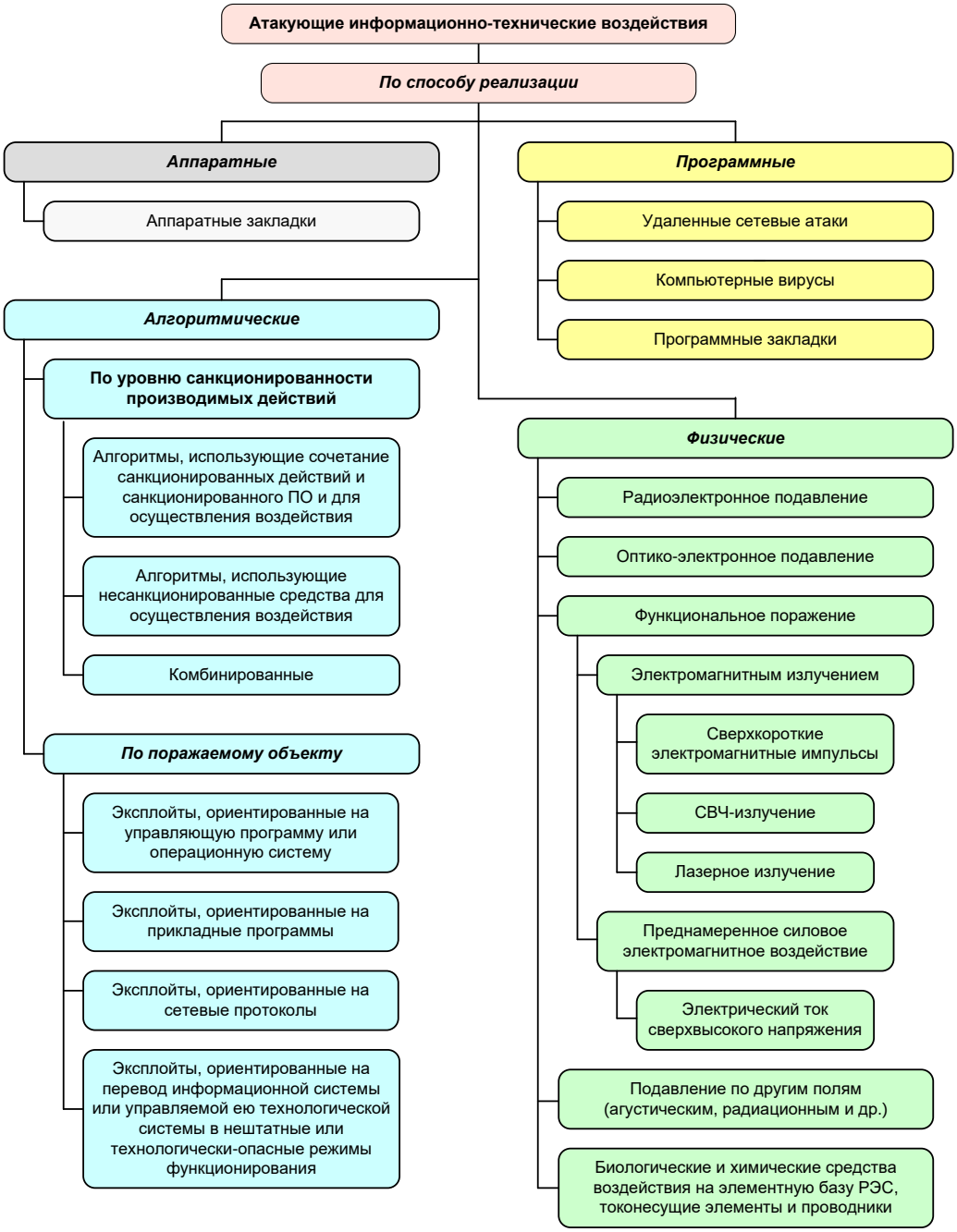


Рис. 15.1 Классификация средств и способов атакующих ИТВ

- программные:
  - компьютерные вирусы;
  - программные закладки;
  - нейтрализаторы тестовых программ и программ анализа кода;
- аппаратные:
  - аппаратные закладки;
- физические:
  - электромагнитные:
    - радиоэлектронное подавление;
    - оптико-электронное подавление;
    - функциональное поражение электромагнитным излучением (электромагнитные импульсы, СВЧ-излучение, лазерное излучение);
    - функциональное поражение преднамеренными силовыми электромагнитными воздействиями (электрический ток сверхвысокого напряжения);
  - по другим полям (акустическим, радиационным и др.);
  - биологические и химические средства воздействия на элементную базу РЭС, токонесущие элементы и проводники.

Рассмотрим более подробно основные широко распространенные атакующие ИТВ, которые могут быть использованы против объектов ТКС:

- удаленные сетевые атаки;
- компьютерные вирусы;
- программные закладки;
- аппаратные закладки.

15.2. Удаленные сетевые атаки

15.2.1. Определение и классификация удаленных сетевых атак

С учетом определения и классификации удаленных воздействий на распределенные вычислительные системы, представленных в работе [54] можно дать следующее определение.

Удаленная сетевая атака – это разрушающее или дестабилизирующее информационно-техническое воздействие, осуществляемое по каналам связи, удаленным относительно атакуемой системы субъектом и характерное для структурно- и пространственно-распределенных информационных систем.

Удаленные сетевые атаки становятся возможными благодаря уязвимостям в существующих протоколах обмена данными и в подсистемах защиты распределенных информационных систем. При этом к основным уязвимостям информационных систем, которые позволяют проводить против них успешные удаленные сетевые атаки, относятся [7, 54]:

- открытость информационной системы, свободный доступ к информации по организации сетевого взаимодействия и способам защиты, применяемым в системе;
- наличие ошибок в операционных системах, в прикладном ПО и в протоколах сетевого обмена;
- разнородность используемых версий ПО и операционных систем;
- сложность организации защиты межсетевого взаимодействия;
- ошибки конфигурирования систем и средств защиты;
- неправильное или ошибочное администрирование систем;
- несвоевременное отслеживание и выполнение рекомендаций специалистов по защите и анализу случаев вторжения для ликвидации эксплойтов и ошибок в ПО;
- «экономия» на средствах и системах обеспечения безопасности или игнорирование их.

Удаленные сетевые атаки можно классифицировать в соответствии с различными основаниями. Общая схема классификации удаленных сетевых атак представлена на рис. 15.2.

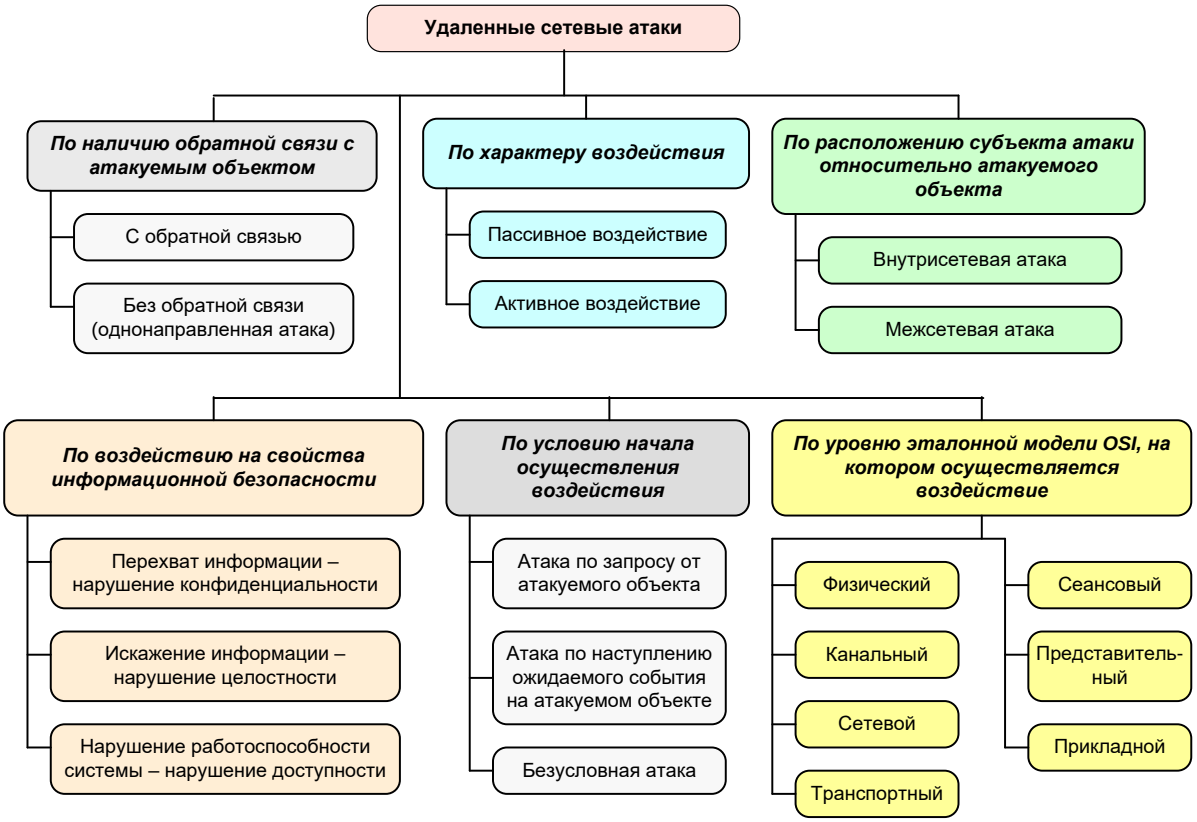


Рис. 15.2. Классификация удаленных сетевых атак

1. По характеру воздействия [7, 54]:
- пассивное воздействие;
- активное воздействие.

Пассивное воздействие не оказывает непосредственного влияния на работу информационной системы, но может нарушать ее политику безопасности. Отсутствие непосредственного влияния на функционирование

атакуемой системы приводит к тому, что пассивную сетевую атаку практически невозможно обнаружить. Примером типовой пассивной удаленной сетевой атаки является прослушивание канала связи.

Активное воздействие оказывает непосредственное влияние на функционирование информационной системы (изменение конфигурации системы, нарушение работоспособности и т. д.) и нарушает принятую в ней политику безопасности. Практически все типы удаленных сетевых атак относятся к активным воздействиям. Очевидной особенностью активного воздействия, по сравнению с пассивным, является принципиальная возможность его обнаружения, так как в информационной системе происходят определенные деструктивные изменения.

2. По воздействию на свойства информационной безопасности ресурсов системы [7, 54]:

- перехват информации – нарушение конфиденциальности;
- искажение информации – нарушение целостности;
- нарушение работоспособности системы – нарушение доступности.

Перехват информации означает получение к ней доступа, но невозможность ее модификации. Следовательно, перехват информации ведет к нарушению ее конфиденциальности. В этом случае осуществляется несанкционированный доступ к информации без возможности ее искажения. Также очевидно, что нарушение конфиденциальности информации является пассивной сетевой атакой. Примером такой атаки, связанной с перехватом информации, может служить прослушивание канала связи в сети.

Искажение информации означает либо полный контроль над информационным потоком между объектами информационной системы, либо возможность передачи сообщений от имени другого объекта. Таким образом, искажение информации ведет к нарушению целостности информационных ресурсов системы. Примером удаленной сетевой атаки, целью которой является нарушение целостности информационных ресурсов, может служить атака, связанная с внедрением ложного сетевого объекта в систему, например, внедрения ложного DNS-сервера.

При нарушении работоспособности системы атакующей стороной не предполагается получение несанкционированного доступа к информации. Ее основная цель – добиться, чтобы элементы распределенной информационной системы на атакуемом объекте вышли из строя, а для всех остальных объектов системы доступ к информационным ресурсам атакованного объекта был бы невозможен. Примером удаленной атаки, целью которой является нарушение работоспособности системы, может служить DoS-атака.

3. По условию начала осуществления воздействия [7, 54]:

- атака по запросу от атакуемого объекта;
- атака по наступлению ожидаемого события на атакуемом объекте;
- безусловная атака.

При атаке по запросу от атакуемого объекта атакующий ожидает передачи от потенциальной цели атаки запроса определенного типа, который и будет условием начала осуществления воздействия. Примером подобных запросов могут служить DNS- и ARP-запросы. Важно отметить, что этот тип удаленных атак наиболее характерен для распределенных сетевых информационных систем.

При атаке по условию наступления ожидаемого события атакующий осуществляет наблюдение за состоянием информационной системы, которая является целью атаки. При возникновении определенного события в этой системе атакующий начинает воздействие на нее. Как и в предыдущем случае, инициатором осуществления начала атаки выступает сама атакуемая система. Такие сетевые атаки довольно распространены. Примером такой атаки может быть атака, связанная с несанкционированным доступом к информационным ресурсам компьютера по сети после факта его успешного заражения backdoor-вирусом, который создает дополнительные уязвимости в подсистеме защиты компьютера.

При безусловной атаке она осуществляется немедленно и безотносительно к состоянию информационной системы и атакуемого объекта. Следовательно, в этом случае атакующий является инициатором начала осуществления атаки.

4. По наличию обратной связи с атакуемым объектом [7, 54]:

- с обратной связью;
- без обратной связи (однонаправленная атака).

Удаленная сетевая атака, осуществляемая при наличии обратной связи с атакуемым объектом, характеризуется тем, что на некоторые запросы, переданные на атакуемый объект, атакующему требуется получить ответ. Следовательно, между атакующим и целью атаки существует обратная связь, которая позволяет атакующему адаптивно реагировать на все изменения, происходящие на атакуемом объекте. Подобные удаленные атаки наиболее характерны для распределенных сетевых информационных систем.

В отличие от атак с обратной связью, удаленным сетевым атакам без обратной связи не требуется реагировать на какие-либо изменения, происходящие на атакуемом объекте. Атаки такого вида обычно осуществляются передачей на атакуемый объект одиночных запросов, ответы на которые атакующему не нужны. Подобную сетевую атаку можно называть однонаправленной удаленной атакой. Примером такой однонаправленной атаки может служить DoS-атака.

5. По расположению субъекта атаки относительно атакуемого объекта [7, 54]:

- внутрисетевая атака;
- межсетевая атака.

В случае внутрисетевой атаки субъект и объект атаки находятся в одной сети. При межсетевой атаке субъект и объект атаки находятся в разных сетях. Важно отметить, что межсетевая удаленная атака представляет

гораздо бóльшую опасность, чем внутрисетевая. Это связано с тем, что в случае межсетевой атаки ее объект и непосредственно атакующий могут находиться на значительном расстоянии друг от друга, что может существенно воспрепятствовать мерам по отражению атаки.

6. По уровню эталонной модели OSI, на котором осуществляется воздействие [7, 54]:
   - физический;
   - канальный;
   - сетевой;
   - транспортный;
   - сеансовый;
   - представительный;
   - прикладной.

Удаленные атаки могут быть ориентированы на сетевые протоколы, функционирующие на различных уровнях модели OSI. При этом надо отметить, что атаки, ориентированные на физический, канальный, сетевой и транспортный уровни, как правило, направлены против сетевой инфраструктуры – оборудования узлов сети и каналы связи. Атаки, ориентированные на сеансовый, представительный и прикладной уровни, как правило, направлены против оконечных терминалов сети. В связи с этим, в зависимости от уровня OSI, на который ориентирована атака, конкретный вид используемого воздействия может значительно меняться. Это может быть воздействие средств РЭП или ЭМИ при атаке, ориентированной на физический уровень, при этом эффекты от такого воздействия отображаются на более верхних уровнях модели OSI. Либо DoS-атака на узловое оборудование сети, либо вирус, поражающий операционную систему конечного терминального оборудования.

15.2.2. Примеры способов информационно-технических воздействий на основе удаленных сетевых атак

В связи с тем, что удаленные сетевые атаки совместно с воздействием вирусных средств составляют подавляющее большинство всех информационно-технических воздействий, рассмотрим их более подробно.

Общая классификация способов информационно-технического воздействия на основе удаленных сетевых атак представлена на рис. 15.3.

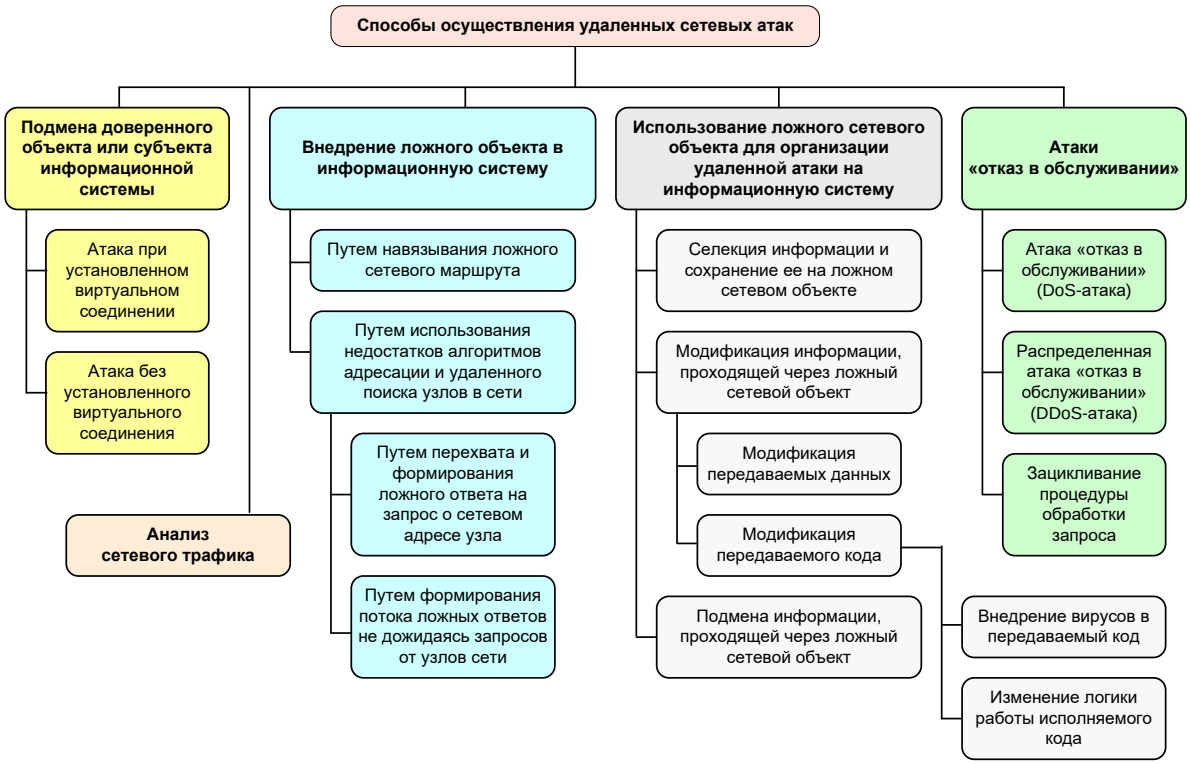


Рис. 15.3. Классификация способов осуществления удаленных сетевых атак

К основным способам информационно-технического воздействия, которые можно отнести к удаленным сетевым атакам, относятся [7, 54]:

- анализ сетевого трафика;
- подмена доверенного объекта или субъекта информационной системы;
- внедрение ложного объекта в информационную систему:
    - внедрение ложного объекта путем навязывания ложного сетевого маршрута;
    - внедрение ложного объекта путем использования недостатков алгоритмов адресации и удаленного поиска узлов в сети;
    - путем перехвата и формирования ложного ответа на запрос о сетевом адресе узла;
    - путем формирования потока ложных ответов, не дожидаясь запросов от узлов сети;
- использование ложного сетевого объекта для организации удаленной атаки на информационную систему:
    - селекция информации и сохранение ее на ложном сетевом объекте;
    - модификация информации, проходящей через ложный сетевой объект;
    - подмена информации, проходящей через ложный сетевой объект;
- атаки типа «отказ в обслуживании»:

- отказ в обслуживании (DoS-атака);
- распределенная атака «отказ в обслуживании» (DDoS-атака);
- зацикливание процедуры обработки запроса.

15.2.2.1. Анализ сетевого трафика

Основной особенностью сетевой информационной системы является то, что ее объекты распределены в пространстве и связь между ними осуществляется по сетевым соединениям. Таким образом, сообщения и данные, пересылаемые между объектами информационной системы, передаются по каналам связи в виде пакетов. Эта особенность привела к появлению специфичного для сетевой информационной системы типового удаленного воздействия, заключающегося в прослушивании канала связи. Такое воздействие называется *анализом сетевого трафика*.

Анализ сетевого трафика позволяет [7, 54]:

- изучить логику работы сетевой информационной системы, то есть получить взаимно однозначное соответствие событий, происходящих в системе, команд и данных, пересылаемых друг другу ее объектами, в момент появления этих событий. Это достигается путем перехвата и анализа пакетов сетевого трафика. Знание логики работы информационной системы позволяет смоделировать и осуществлять другие удаленные сетевые атаки;
- перехватить поток данных, которыми обмениваются объекты сетевой информационной системы. Таким образом, эта атака заключается в получении на удаленном объекте несанкционированного доступа к информации, которой обмениваются два сетевых абонента. Отметим, что при этом отсутствует возможность модификации трафика и сам анализ возможен только внутри одного сегмента сети. Примером информации, перехваченной при помощи такой типовой удаленной атаки, могут служить имя и пароль пользователя, пересылаемые в незашифрованном виде по сети.

В соответствии с выше представленной классификацией анализ сетевого трафика является пассивным воздействием. Осуществление такой атаки без обратной связи ведет к нарушению конфиденциальности информации внутри одного сегмента сети на канальном или сетевом уровне OSI. При этом начало атаки является безусловной по отношению к цели атаки [7, 54].

15.2.2.2. Подмена доверенного объекта или субъекта информационной системы

Одной из проблем безопасности сетевой информационной системы является недостаточная идентификация и аутентификация ее объектов, удаленных друг от друга. Основная трудность заключается в осуществлении однозначной идентификации сообщений, передаваемых между субъ-

ектами и объектами сетевого взаимодействия. Обычно в сетевых информационных системах эта проблема решается следующим образом: в процессе создания виртуального канала объекты системы обмениваются определенной информацией, уникально идентифицирующей этот канал. Однако такой обмен производится не всегда. Зачастую, особенно при передаче служебной и адресной информации, в сети используются одиночные сообщения, не требующие подтверждения. Так как сетевой адрес достаточно просто подделывается, его можно использовать для виртуальной подмены доверенного объекта или субъекта информационной системы. Таким образом, в том случае, когда в сети используются нестойкие алгоритмы идентификации удаленных объектов, оказывается возможной удаленная атака *подмена доверенного объекта или субъекта информационной системы*, заключающаяся в передаче по сети сообщений от имени произвольного объекта или субъекта информационной системы [7, 54].

Существуют две разновидности этой сетевой атаки, в зависимости от принятой в системе политики информационной безопасности и подхода к защите сетевых соединений [7, 54]:

- атака при установленном виртуальном соединении;
- атака без установленного виртуального соединения.

В случае если в сети для сеанса обмена данными устанавливаются виртуальные соединения, атака будет заключаться в присвоении прав доверенного субъекта сетевого взаимодействия, легально подключившегося к объекту системы. Это позволит атакующему вести сеанс работы с объектом информационной системы от имени доверенного субъекта. Реализация таких удаленных атак обычно состоит в передаче пакетов обмена с атакующего объекта на цель атаки от имени доверенного субъекта взаимодействия (при этом переданные сообщения будут восприняты системой как корректные). Однако для осуществления атаки этого типа необходимо преодолеть систему идентификации и аутентификации сетевых сообщений [7, 54].

Атаки на информационную систему, в которой не используются виртуальные соединения, заключаются в передаче служебных сообщений от имени сетевых управляющих устройств, например, от имени маршрутизаторов. В этом случае возможна подделка сетевого адреса отправителя. Например, так реализуется удаленная атака, использующая навязывание ложного маршрута путем посылки ложных адресных сообщений [7, 54].

Подмена доверенного объекта или субъекта информационной системы может быть классифицирована как активное воздействие, совершаемое с целью нарушения конфиденциальности и целостности информации по наступлению на атакуемом объекте определенного события. Такая удаленная атака может являться как внутрисетевой, так и межсетевой, как с обратной связью, так и без обратной связи с атакуемым объектом и осуществляться на сетевом или транспортном уровнях модели OSI [7, 54].

15.2.2.3. Внедрение ложного объекта в информационную систему

Зачастую в распределенной информационной системе бывают недостаточно надежно решены проблемы идентификации сетевых управляющих устройств (например, маршрутизаторов) при их взаимодействии с объектами системы. В этом случае такая распределенная система может подвергнуться сетевой атаке, связанной с изменением параметров маршрутизации и внедрением в сеть ложного объекта. В том случае, если настройки сети таковы, что для взаимодействия объектов необходимо использовать алгоритмы удаленного поиска узлов, то это также может быть использовано для внедрения в систему ложного объекта. Таким образом, существуют два принципиально разных способа проведения атаки «внедрение ложного объекта в информационную систему» [7, 54]:

- внедрение ложного объекта путем навязывания ложного сетевого маршрута;
- внедрение ложного объекта путем использования недостатков алгоритмов адресации и удаленного поиска узлов в сети:
    - путем перехвата и формирования ложного ответа на запрос о сетевом адресе узла;
    - путем формирования потока ложных ответов, не дожидаясь запросов от узлов сети.

Современные глобальные сети представляют собой совокупность сетевых сегментов, связанных между собой через узлы-маршрутизаторы. Каждый маршрутизатор имеет специальную таблицу, называемую таблицей маршрутизации, в которой для каждой пары адресатов сети указывается оптимальный маршрут. Основная цель атаки, связанной с внедрением ложного объекта путем навязывания ложного маршрута, состоит в том, чтобы изменить исходную маршрутизацию на объекте сетевой информационной системы так, чтобы новый маршрут проходил через ложный объект сети – узел атакующего. Реализация этой атаки состоит в несанкционированном использовании протоколов управления сетью для изменения исходных таблиц маршрутизации. Данная атака проходит в две стадии [7, 54].

1. Атакующему необходимо от имени сетевых управляющих устройств (например, маршрутизаторов) произвести рассылку по сети специальных служебных сообщений, что приведет к изменению маршрутизации в сети. В результате успешного изменения маршрута атакующий получит полный контроль над потоком информации, который будет проходить через его узел.
2. Атакующий наращивает количество трафика, перенаправленного через свой узел, и получает возможность вести прием, анализ и передачу сообщений, передаваемых по сети.

Внедрение ложного объекта путем навязывания ложного сетевого маршрута – активное воздействие безусловное по отношению к цели атаки. Данная удаленная атака может осуществляться как внутри одного сегмента

сети, так и межсетевым образом, как с обратной связью, так и без обратной связи с атакуемым объектом на сетевом, транспортном и прикладном уровне модели OSI [7, 54].

В распределенной информационной системе часто оказывается, что ее удаленные объекты изначально не имеют достаточно информации, необходимой для адресации передаваемых сообщений. Обычно такой информацией являются аппаратные и логические адреса объектов системы. Для получения подобной информации в распределенных системах используются различные алгоритмы удаленного поиска, заключающиеся в передаче по сети специальных поисковых запросов. После получения ответа на запрос запросивший субъект системы обладает всеми необходимыми данными для адресации. Руководствуясь полученными из ответа сведениями об искомом объекте, запросивший субъект системы начинает передачу информации. Примером подобных запросов, на которых базируются алгоритмы удаленного поиска, могут служить ARP- и DNS-запросы в сети Интернет [7, 54].

В случае использования в распределенной информационной системе механизмов удаленного поиска существует возможность на атакующем объекте перехватить посланный запрос и послать на него ложный ответ, где указать данные, использование которых приведет к адресации на атакующий ложный узел. В дальнейшем весь поток информации между субъектом и объектом взаимодействия будет проходить через этот ложный объект информационной системы [7, 54].

Другой вариант внедрения в распределенную информационную систему ложного объекта использует недостатки алгоритма удаленного сетевого поиска и состоит в периодической передаче на атакуемый объект заранее подготовленного ложного ответа без приема поискового запроса. При этом атакующий может спровоцировать атакуемый объект на передачу поискового запроса – и тогда его ложный ответ будет немедленно принят и обработан. Такая удаленная атака чрезвычайно распространена в глобальных сетях, когда у атакующего из-за нахождения его в другом сетевом сегменте относительно цели атаки просто нет возможности перехватить поисковый запрос [7, 54].

Внедрение ложного объекта путем использования недостатков алгоритмов адресации и удаленного поиска узлов в сети – активное воздействие, совершаемое с целью нарушения конфиденциальности и целостности информации, которое может являться атакой по запросу от атакуемого объекта, а также безусловной атакой. Данная удаленная атака может быть как внутрисетевой, так и межсетевой, имеет обратную связь с атакуемым объектом и осуществляется на канальном, сетевом и прикладном уровнях модели OSI [7, 54].

15.2.2.4. Использование ложного объекта для организации удаленной атаки на систему

После внедрения ложного объекта в сеть и получения контроля над проходящим потоком информации в сети ложный объект может применяться для различных способов воздействия на перехваченную информацию. Выделяют следующие основные воздействия на информацию, перехваченную ложным объектом [7, 54]:

- селекция информации и сохранение ее на ложном сетевом объекте;
- модификация информации, проходящей через ложный сетевой объект;
- подмена информации, проходящей через ложный сетевой объект.

Селекция информации и сохранение ее на ложном сетевом объекте являются пассивной сетевой атакой, сходной с атакой «анализ сетевого трафика», которая дополнена динамическим семантическим анализом, производимым на ложном объекте. Вместе с тем наибольший интерес представляет возможность использования ложного объекта для модификации или подмены информации.

Рассматривают два основных вида модификации информации [7, 54]:

- модификация передаваемых данных;
- модификация передаваемого кода:
    - внедрение вирусов в передаваемый код;
    - изменение логики работы исполняемого кода.

Для модификации передаваемых данных на внедренном объекте производятся селекция потока перехваченной информации и его анализ. При этом может быть распознан тип передаваемых файлов (исполняемый или файл, содержащий данные). При обнаружении файла данных появляется возможность модифицировать эти данные, проходящие через ложный объект. При этом, если модификация данных является достаточно стандартным воздействием, то на модификации передаваемого кода стоит остановиться отдельно.

Ложный объект, проводя семантический анализ информации, проходящей через него, может выделять среди потока информации файлы, содержащие исполняемый код. Для того чтобы определить, что передается по сети – код или данные, – необходимо распознавать определенные особенности, свойственные конкретным типам исполняемых файлов. При этом можно выделить два различных по цели вида модификации кода [7, 54]:

- внедрение вирусов в передаваемый код;
- изменение логики работы исполняемого кода.

При внедрении вирусов в передаваемый код к исполняемому файлу дописывается тело вируса, а также изменяется точка начала исполнения кода так, чтобы она указывала на начало кода внедренного вируса. Описанный способ, в принципе, ничем не отличается от стандартного заражения исполняемого файла вирусом, за исключением того, что файл оказыва-

ется заражен вирусом в момент передачи его по сети. Такое возможно лишь при использовании воздействия «внедрение ложного объекта» [7, 54].

При изменении логики работы исполняемого файла при передаче его по сети происходит похожая модификация исполняемого кода. Однако ее цель – алгоритмическое воздействие, ориентированное на внедрение программных закладок, внесение в исполняемый файл дополнительных уязвимостей или эксплойтов. Сложностью такого воздействия является то, что для него, как правило, требуется предварительное исследование логики функционирования исполняемого файла [7, 54].

Внедрение ложного объекта позволяет не только модифицировать, но и подменять перехваченную им информацию. При возникновении в сети определенного контролируемого ложным объектом события одному из участников обмена посылается заранее подготовленная дезинформация. При этом такая дезинформация, в зависимости от контролируемого события, может быть, как исполняемым кодом, так и данными.

15.2.2.5. Атаки типа «Отказ в обслуживании»

В общем случае в сетевой информационной системе каждый ее субъект должен иметь возможность подключиться к любому объекту системы и получить в соответствии со своими правами удаленный доступ к его информационным ресурсам. Обычно в сетевых информационных системах возможность предоставления удаленного доступа реализуется следующим образом: на объекте системы запускаются на выполнение ряд программ-серверов (например, *FTP*-сервер, *WWW*-сервер и т. п.), предоставляющих удаленный доступ к ресурсам этого объекта. В случае получения запроса на соединение сервер должен по возможности передать на запросивший объект ответ, в котором либо разрешить подключение, либо нет. Очевидно, что сервер способен отвечать лишь на ограниченное число запросов. Эти ограничения зависят от параметров информационной системы, пропускной способности ее сети и быстродействия ЭВМ, на которых он функционирует. Атака «отказ в обслуживании» направлена на блокировку доступа к объекту путем исчерпания его ресурсов за счет отправки большого числа запросов к нему.

Различают три типа этих удаленных атак.

1. *Отказ в обслуживании* (DoS-атака) – передача с одного адреса такого количества запросов на атакуемый объект, которое позволяет передать пропускная способность канала связи. В данном случае, если в системе не предусмотрены правила, ограничивающие число принимаемых запросов с одного объекта (адреса) системы, то результатом этой атаки может являться как переполнение очереди запросов и отказа одной из телекоммуникационных служб, так и полная блокировка объекта из-за невозмож-

ности системы заниматься ничем другим, кроме обработки запросов.

2. *Распределенная атака «отказ в обслуживании»* (DDoS-атака) – передача с нескольких объектов системы на другой атакуемый объект бесконечного числа запросов на подключение от имени этих или других объектов. Результатом применения этой удаленной атаки является нарушение на атакованном объекте работоспособности соответствующей службы предоставления удаленного доступа, то есть невозможность получения удаленного доступа с других объектов сетевой информационной системы.
3. *Зацикливание процедуры обработки запроса* – передача на атакуемый объект некорректного, специально подобранного запроса. В этом случае при наличии ошибок в удаленной системе возможно переполнение буфера с последующим зависанием системы.

Удаленная сетевая атака «отказ в обслуживании» классифицируется как активное воздействие, осуществляемое с целью нарушения работоспособности системы, безусловное относительно цели атаки. Данная атака является однонаправленным воздействием, осуществляемым как межсетевым, так и внутрисетевым образом, осуществляемым на сетевом, транспортном и прикладном уровнях модели OSI [7, 54].

Схема классификации основных способов осуществления атаки «отказ в обслуживании» представлена на рис. 15.4.

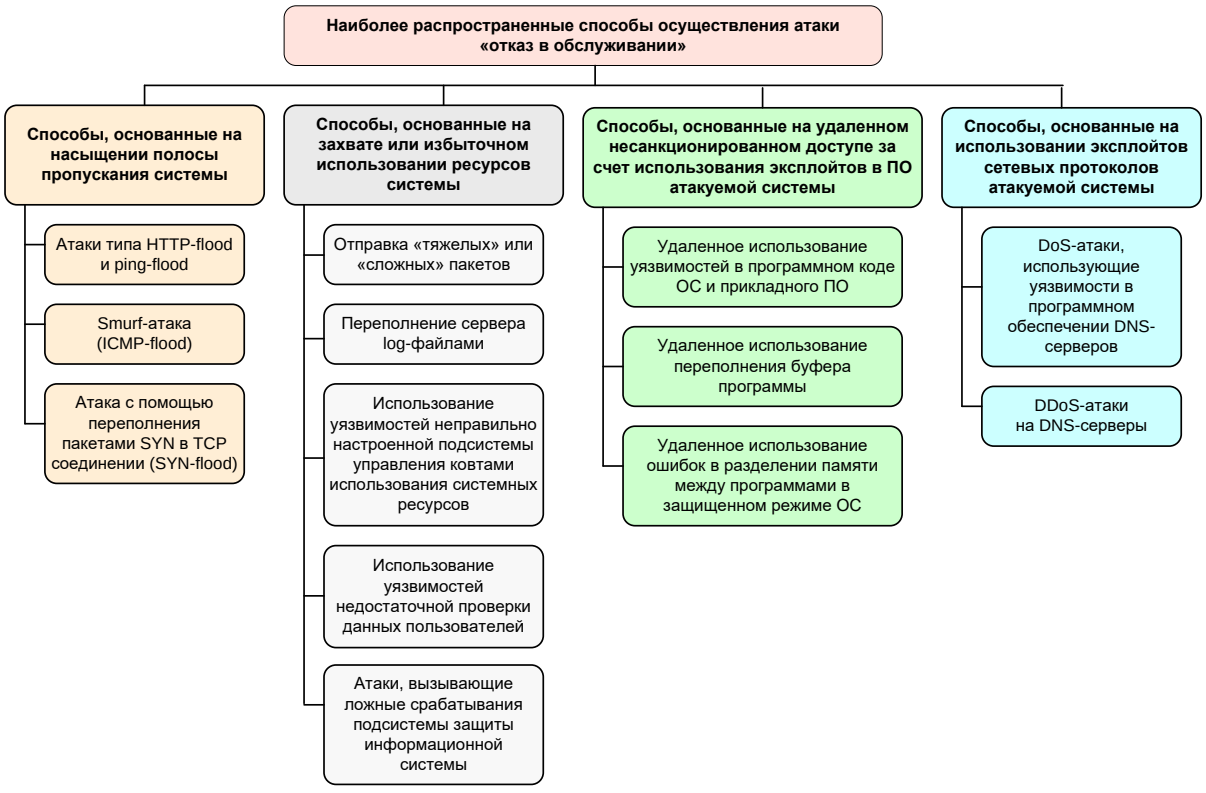


Рис. 15.4. Наиболее распространенные способы осуществления атаки «отказ в обслуживании»

Наиболее распространенными способами осуществления атаки «отказ в обслуживании» являются следующие.

1. Способы, основанные на насыщении полосы пропускания системы, – атаки, связанные с большим количеством, как правило, бессмысленных или сформированных в неправильном формате запросов к информационной системе или ее сетевому оборудованию, с целью обеспечить отказ в работе системы из-за исчерпания ее системных ресурсов (процессорного времени, памяти или пропускной способности каналов связи). К наиболее распространенным таким способам относятся [56]:

- атаки типа HTTP-flood и ping-flood;
- Smurf-атака (ICMP-flood);
- атака с помощью переполнения пакетами SYN в TCP соединении (SYN-flood).

2. Способы, основанные на недостатке ресурсов системы, – атаки, связанные с захватом или избыточным использованием ресурсов информационной системы. К наиболее распространенным таким способам относятся [56]:

- отправка «тяжелых» или «сложных» пакетов;
- переполнение сервера log-файлами;
- использование уязвимостей неправильно настроенной подсистемы управления квотами использования системных ресурсов;
- использование уязвимостей недостаточной проверки данных пользователей;
- атаки, вызывающие ложные срабатывания подсистемы защиты информационной системы.

3. Способы, основанные на удаленном несанкционированном доступе за счет использования эксплойтов в ПО атакуемой системы. К наиболее распространенным таким способам относятся [56]:

- удаленное использование уязвимостей в программном коде операционной системы и прикладного ПО информационной системы;
- удаленное использование переполнения буфера программы;
- удаленное использование ошибок в разделении памяти между программами в защищенном режиме операционной системы.

4. Способы, основанные на использовании эксплойтов сетевых протоколов атакуемой системы. К наиболее распространенным таким способам относятся [56]:

- DoS-атаки, использующие уязвимости в ПО DNS-серверов;
- DDoS-атаки на DNS-серверы.

В настоящее время атаки типа «отказ в обслуживании» являются не только наиболее распространенными, но и наиболее опасными воздействиями. Так в ноябре 2002 г. была проведена глобальная DDoS-атака на корневые DNS-серверы с целью полного блокирования общедоступного сегмента сети Интернет. В результате этой атаки злоумышленники смогли вывести из строя 7 из 13 корневых DNS-серверов [56].

15.3. Компьютерные вирусы и другие вредоносные программы

В настоящее время вирусы являются наиболее известным и широко распространённым средством ИТВ воздействия на удаленные системы, в том числе, на ТКС, а также на их сетевые ОС и ПО.

Несмотря на долгую историю компьютерной вирусологии, использование вирусов в качестве профессиональных средств информационно-технического воздействия начато сравнительно недавно. К первому случаю такого использования относится использование в 2010 г. вируса Stuxnet с целью срыва ядерной программы Ирана за счет инфицирования АСУ технологическим процессом обогащения урана [47]. Особенностью современных профессиональных вирусов является то, что они, как правило, являются комплексными продуктами и состоят из различных модулей, которые относятся к различным типам и ориентированы на решение конкретной задачи (модули типа «классический вирус» – для саморазмножения в информационной системе, модули типа «червь» – для распространения по сети, модуль типа «троян» – для организации дестабилизирующего воздействия).

В отличие от своих «непрофессиональных собратьев», средства информационно-технического воздействия на основе вирусов обладают следующими особенностями функционирования [47]:

- избирательность цели и действий;
- использование уязвимостей, в том числе уязвимостей 0-дня, закладок и скрытых каналов;
- маскировка, скрытность, криптозащита, самоликвидация;
- широкая функциональность в плане решения целевых задач;
- гибкая система саморазмножения;
- инфраструктурная поддержка, обновление и управление;
- масштабируемость, наличие СУБД-атак;
- высокое качество кода и возможности обработки некорректных ситуаций.

Классификация компьютерных вирусов, до сих пор обладает некоторой неоднозначностью, т.к. многие вирусы сложно отнести к какому-либо одному типу. Вместе с тем по базовой функциональности и по способами распространения вирусы можно классифицировать по четырем основным типам (рис. 11.5) [3]:

1. классические вирусы – размножаются путем самокопирования, как правило, поражая отдельные вычислительные системы;
2. программы типа «червь» – предусматривает функционал самостоятельного перемещается по сети и проникает в ее отдельные узлы;
3. программы типа «троян» – имитирует полезную для пользователя функциональность, провоцируя его на активные действия по своему запуску или активации;
4. другие вредоносные программы.

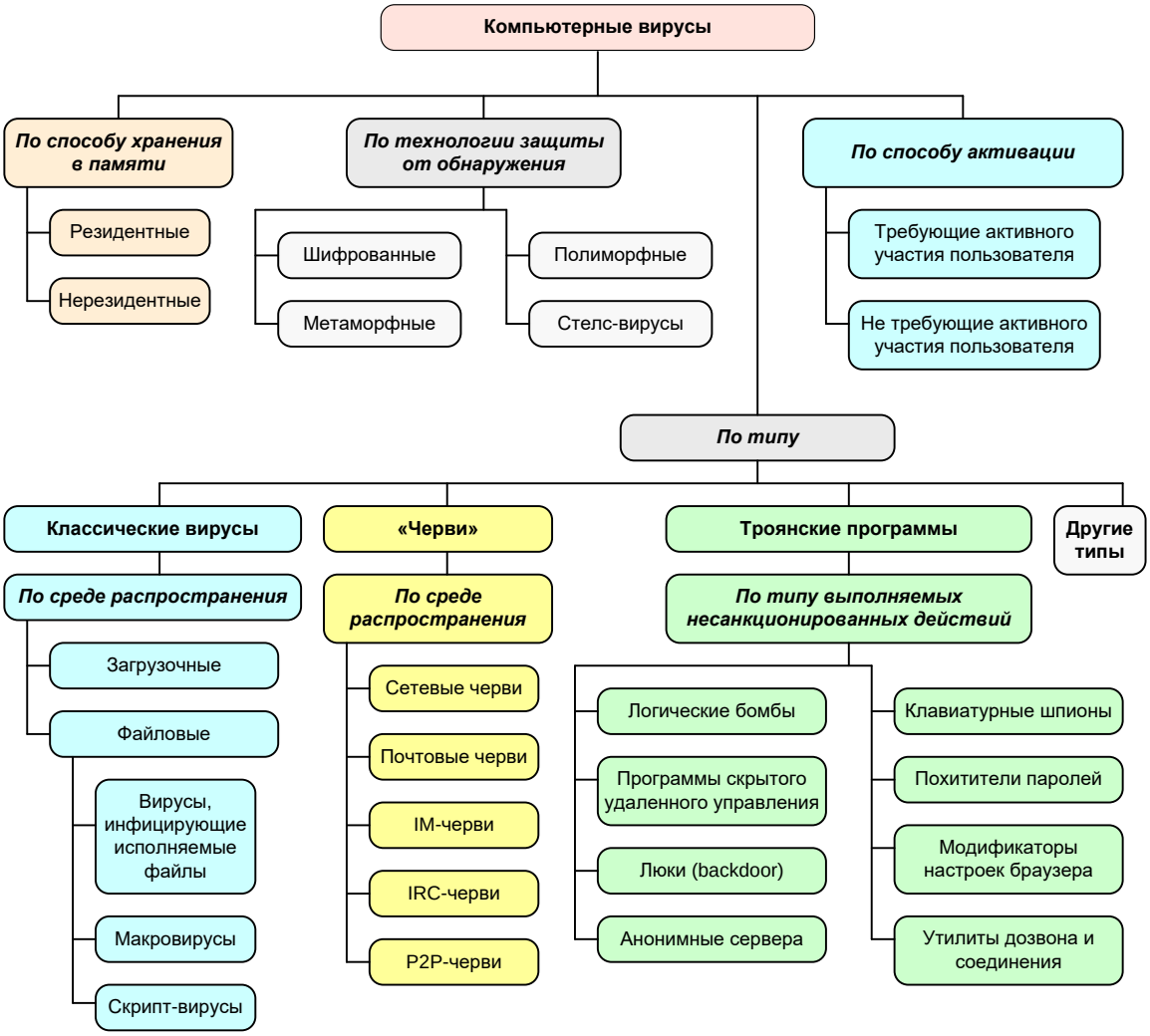


Рис. 11.5. Классификация компьютерных вирусов

По способу хранения в памяти информационной системы компьютерные вирусы можно классифицировать на [46]:

- резидентные;
- нерезидентные.

Резидентные вирусы после активации хранят свои копии в оперативной памяти системы, способны перехватывать события операционной системы и программ (например, обращения к файлам или дискам) и инициировать при этом процедуры заражения обнаруженных объектов. Поэтому резидентные вирусы опасны не только во время работы инфицированной программы, но и после ее окончания. Резидентные копии таких вирусов остаются жизнеспособными вплоть до выключения или перезагрузки информационной системы [46].

Нерезидентные вирусы, напротив, активны на довольно непродолжительных интервалах времени – пока функционирует инфицированная вирусом программа [46].

Для защиты от обнаружения со стороны антивирусов и средств защиты информационной системы в вирусах могут применяться следующие технологии [3]:

- *шифрование* – вирус состоит из двух функциональных элементов: собственно вирус и шифратор. При этом каждая конкретная копия вируса состоит из шифратора, собственного случайного ключа и собственно вируса, зашифрованного этим ключом;
- *метаморфизм* – создание различных копий вируса путем замены блоков команд на эквивалентные, путем перестановки местами участков кода, вставки между значащими участками кода незначащих команд и др.;
- *перехват управления* при обращении операционной системы или системы защиты к инфицированным элементам информационной системы.

Использование этих технологий маскировки вирусов привело к появлению следующих типов вирусов, которые классифицируются по технологии защиты от обнаружения [3]:

- *шифрованный вирус* – вирус, использующий простое шифрование своего тела со случайным ключом и неизменный шифратор. Такие вирусы могут быть обнаружены по сигнатуре шифратора;
- *метаморфный вирус* – вирус, применяющий метаморфизм ко всему своему телу для создания новых копий;
- *полиморфный вирус* – вирус, использующий метаморфный шифратор для шифрования тела вируса со случайным ключом. При этом часть информации, используемой для получения новых копий шифратора, также может быть зашифрована. Например, вирус может реализовывать несколько алгоритмов шифрования и при создании новой копии менять не только команды шифратора, но и сам алгоритм шифрования;
- *стелс-вирус* (*stealth virus* – вирус-невидимка) – вирус, полностью или частично скрывающий свое присутствие в системе путем перехвата обращений к операционной системе на осуществление чтения или записи в инфицированных объектах (загрузочных секторах, элементах файловой системы, памяти и т. д.) и подмены их содержимого с целью демонстрации подсистеме защиты оригинального содержимого объекта до его заражения.

Кроме того, компьютерные вирусы можно классифицировать по способу активации [3].

- *Требующие активного участия пользователя*. Отличительной особенностью таких компьютерных вирусов является использование обманных методов. Это проявляется, например, когда получатель инфицированного файла вводится в заблуждение текстом письма и добровольно открывает вложение с «почтовым червем», тем самым активируя его.
- *Не требующие активного участия пользователя*. Активация компьютерных вирусов без участия пользователя возможна за счет того, что вирус самостоятельно находит и использует уязвимости в безопасности информационной системы.

Далее представлены особенности основных типов вирусов более подробно.

15.3.1. Классические компьютерные вирусы

Основное свойство классического компьютерного вируса – это способность к саморазмножению [3].

Вирус – это программа, способная создавать свои дубликаты (не обязательно совпадающие с оригиналом) и внедрять их в вычислительные системы и/или файлы, системные области операционных систем и прочие информационные ресурсы. При этом дубликаты сохраняют способность к дальнейшему распространению [3].

Условно жизненный цикл вируса можно разделить на пять стадий [3]:

1. проникновение на чужой компьютер;
2. активация;
3. поиск объектов для заражения;
4. подготовка копий;
5. внедрение копий;

Путями проникновения вируса могут служить мобильные носители, сетевые соединения, а также любые другие каналы, по которым можно скопировать файл. Однако, в отличие от «червей», вирусы не используют сетевые ресурсы – заражение вирусом возможно, только если пользователь сам каким-либо образом его активировал [3].

После проникновения следует активация вируса. В соответствии с выбранным методом активации вирусы делятся на следующие виды [3]:

- *загрузочные вирусы* – заражают загрузочные сектора жестких дисков и мобильных носителей;
- *файловые вирусы* – заражают файлы. Отдельно по типу среды обитания в этой группе также выделяют следующие типы вирусов:
    - *вирусы, инфицирующие исполняемые файлы* – различными способами внедряются в исполняемые файлы программ (внедряют свой вредоносный код или полностью их перезаписывают), создают файлы-двойники, свои копии в различных каталогах жесткого диска или используют особенности организации файловой системы;
    - *макровирусы* – инструкции, написанные на внутреннем языке команд заражаемого приложения (на так называемых, макросах);
    - *скрипт-вирусы* – инструкции, написанные на внутреннем языке для определенной командной оболочки (скриптов).

Дополнительным отличием вирусов от других вредоносных программ служит их жесткая привязанность к операционной системе или про-

граммной оболочке, для которой каждый конкретный вирус был написан [3].

Основная цель вируса – распространение на другие ресурсы информационной системы и выполнение деструктивных действий при определенных событиях или действиях пользователя [3].

15.3.2. Черви

В отличие от классических вирусов, программы типа «червь» – это вполне самостоятельные программы, которые также способны к саморазмножению, однако при этом они способны и к самостоятельному распространению с использованием сетевых каналов. Для подчеркивания этого свойства иногда используют термин «*сетевой червь*» [3].

Программа типа «червь» – это программа, распространяющаяся по сетевым каналам и способная к самостоятельному преодолению подсистем защиты информационных систем, а также к созданию и дальнейшему распространению своих копий, не обязательно совпадающих с оригиналом [3].

Жизненный цикл «червей» состоит из следующих стадий [3]:

- проникновение в информационную систему;
- активация;
- поиск объектов для заражения;
- подготовка копий;
- распространение копий.

В зависимости от способа проникновения в систему «черви» классифицируются на следующие типы [3]:

- «сетевые черви» – используют для распространения локальные сети, каналы в информационно-вычислительных сетях, глобальные сети, в том числе и Интернет;
- «почтовые черви» – распространяются с помощью почтовых программ;
- «IM-черви» – используют для распространения системы мгновенного обмена сообщениями типа Internet Massager;
- «IRC-черви» – распространяются по каналам IRC (Internet Relay Chat) для обмена информацией в чатах и форумах;
- «P2P-черви» – распространяются при помощи пиринговых файлообменных P2P-сетей.

Сетевые черви могут кооперироваться с вирусами – такая пара способна самостоятельно распространяться по сети (благодаря червю) и в то же время заражать ресурсы компьютера (функции вируса) [3].

15.3.3. Троянские программы

В отличие от вирусов и червей, в программах типа «троянский конь» не всегда предусмотрен функционал саморазмножения. Довольно большая часть таких программ функцией саморазмножения вообще не обладает [3].

Программа типа «троян» (*«троянский конь»*) – программа, осуществляющая вредоносное воздействие, при этом отличительной чертой программы этого типа является то, что она имитирует полезную для пользователя функциональность, провоцируя его на активные действия по своему запуску или активации [3].

Некоторые «трояны» способны к самостоятельному преодолению подсистемы защиты информационной системы с целью проникновения в нее. Однако в большинстве случаев они проникают в систему вместе с вирусом либо с червем. В этом случае вирус или червь следует рассматривать как средство доставки, а «троян» – как средство информационного поражении [3].

Жизненный цикл «троянов» состоит всего из трех стадий [3]:

1. проникновение в систему;
2. активация;
3. выполнение вредоносных действий.

Как уже говорилось выше, проникать в информационную систему «трояны» могут двумя путями – самостоятельно и за счет кооперации с вирусом или «сетевым червем». В первом случае может быть использована маскировка, когда «троян» выдает себя за полезное приложение, которое пользователь самостоятельно копирует и запускает. При этом программа-носитель действительно может быть полезна, однако наряду с основными функциями она может выполнять действия, свойственные «трояну» [3].

Для проникновения в информационную систему «трояну» необходима активация, и здесь он похож на «червя»: либо требует активных действий от пользователя, либо самостоятельно заражает его, используя уязвимости в защите информационной системы [3].

Программы типа «троян», как правило, классифицируются по типу выполняемых несанкционированных действий [3].

- *Программы скрытого удаленного управления* – это «трояны», которые обеспечивают несанкционированный удаленный контроль над инфицированной информационной системой. Такие программы предоставляют удаленному пользователю возможность скрытого исполнения программ в информационной системе, поиска, модификации и удаления информации, возможности скрыто загружать или отсылать информацию.
- *Анонимные сервера* – разновидность «троянов», которые используют ресурсы зараженной информационной системы в своих целях, связанных с несанкционированной сетевой активностью: создание bot-сетей и управление ими, выполнение несанкционированных распределенных вычислений, организация и координация DDOS-атак, массовая отправка электронной почты и другие подобные действия.
- *Клавиатурные шпионы* – находясь в оперативной памяти, записывают все данные, набираемые на клавиатуре, с целью последующей их передачи.

- *Похитители паролей* – предназначены для кражи паролей и другой конфиденциальной информации путем поиска на зараженной системе специальных файлов, которые ее содержат.
- *Модификаторы настроек браузера* (или других программ просмотра информации в сети) – изменяют настройки браузера таким образом, чтобы стали возможными удаленное исполнение кода в браузере, доступ к хранящейся в нём конфиденциальной информации, подмена сертификатов безопасности, перенаправление на ложные страницы и т. п.
- *Утилиты дозвона и соединения* – в скрытом от пользователя режиме инициируют несанкционированное подключение к удаленным сервисам.

Отдельно отметим, что существуют программы из класса «троянов», которые наносят вред другим удаленным информационным системам и сетям, при этом не нарушая работоспособности инфицированной системы. Примером таких программ могут служить анонимные сервера – организаторы DDoS-атак [3].

15.3.4. Примеры средств информационно-технических воздействий на основе компьютерных вирусов

Рассмотрим наиболее известные к настоящему времени средства информационно-технических воздействий на основе вирусов, а также особенности их применения в информационных операциях современности.

Stuxnet. Вирус Stuxnet – компьютерный червь, поражающий компьютеры под управлением операционной системы Microsoft Windows и промышленные системы, управляющие технологическими процессами. Это первое широко известное вирусное программное средство, имеющее точечную целевую функцию инфицирования конкретной АСУ с целью функционального поражения управляемого ею технологического процесса [47].

Вирус Stuxnet, внедренный в АСУ технологическим процессом обогащения урана, за счет перехвата и модификации команд от промышленного контролера марки Simatic S7 и рабочими станциями SCADA-системы Simatic WinCC фирмы Siemens, в течение длительного времени задавал для центрифуг нештатный режим работы, что привело к отказу более 1000 центрифуг на иранском заводе по обогащению урана. Уникальность программы заключалась в том, что впервые в истории кибератак вирус физически разрушал инфраструктуру [47].

По заявлению бывших сотрудников NSA и JCS DoD, этот вирус был разработан в рамках американо-израильской операции противодействия ядерным планам Ирана. Stuxnet включает десяток выполняемых компонент общим объемом в 1,2 Мбайта, написанных в основном на языках *C* и *C++*. Базовой функциональной средой является *Win*32 [47].

Stuxnet поддерживает как минимум 8 способов саморазмножения, эксплуатацию более десяти уязвимостей, в том числе программной закладки (мастер-пароля), осуществляет контроль и управление через удаленные компьютеры (при обнаружении выхода в Интернет), включает Windows- и PLC-руткиты и другие способы маскировки и скрытия, а также выполняет обработку ошибочных ситуаций и др. Кроме того, Stuxnet способен внедряться в ряд системных «доверенных» процессов, в том числе инициируемых антивирусными продуктами, такими как: Avira, BitDefender, Computer Associates, Eset, F-Secure, Kaspersky Lab, McAfee, Symantec и Trend Micro [47].

Отличительными особенностями Stuxnet являются [47]:

- использование уязвимостей 0-дня;
- возможность распространения в изолированной среде (без выхода в Интернет), посредством flash-накопителей или собственной локальной p2p-сети;
- наличие компонент, подписанных 2 похищенными электронными цифровыми подписями;
- заражение системы управления технологическими процессами Siemens Simatic Step7;
- выполнение модификации PLC-кода на контроллерах Siemens с целью деструктивного воздействия на физическое оборудование (центрифуг обогащения урана) и дезинформацию операторов завода.

Функциональные клоны Stuxnet, адаптированные под новые цели, могут успешно использоваться для диверсий в АСУ промышленных предприятий, электростанций, управления движением транспорта и т. п.

Flame. Вирус-червь Flame, известный также как Skywiper или Flamer, является примером средства, ориентированного на решение конкретных разведывательных задач на Ближнем Востоке, в первую очередь в Иране. Отмечено около 700 заражений этим вирусом. Активный период действия вируса составлял порядка 6 лет до момента его обнаружения в 2012 г. [47].

Вирус Flame представляет собой комплекс программ объемом около 20 Мбайт (устанавливается поэтапно) и значительно превосходит в этом отношении вирус Stuxnet. В состав Flame входят: криптобиблиотеки, библиотеки архивирования zlib, libbz2, ppmd, СУБД sqlite3, веб-сервер, виртуальная машина Lua. Базовой функциональной средой является Win32 [47].

Ключевыми особенностями Flame являются [47]:

- использование уязвимостей, в том числе 0-дня;
- компрометация ЭЦП в ОС Windows (путем атаки на MD5);
- поиск офисных документов, проектной документации и чертежей (например, pdf и drw файлов), контактной информации, в том числе из соцсетей;
- возможность перехвата аудио и экранной информации;
- поиск и подключение к BIuetooth-устройствам;
- закрытая передача информации на удаленный компьютер;

- наличие инструментария для взлома механизмов защиты.

Что касается последнего, то в составе Flame имеются средства инвентаризации, мониторинга трафика, включая подсистему сбора парольной информации, поиска остаточной информации, анализа файловой системы, архивов и множества типов файлов, кейлогер и т. п. [47].

Flame может распространяться через USB-носители и сеть, также имеет оригинальную возможность обновления путем компрометации обновлений ОС Windows. Работа программы обеспечивается сложной динамичной инфраструктурой – например, известно около сотни доменов, задействованных для передачи данных на командные серверы Flame, попеременно располагавшиеся в различных странах мира. Несмотря на явно разведывательные цели, Flame содержит модуль удаления файлов [47].

Предположительно вирусы Flame и Stuxnet были разработаны одной командой в рамках американо-израильского сотрудничества и использовались совместно. При этом Flame имел разведывательные цели, а Stuxnet – диверсионные.

Duqu – вирус-червь, обнаруженный в 2011 г. Некоторые исследователи полагают, что он связан с червем Stuxnet. Распространение этого вируса происходило через электронную почту. Заражение системы происходит посредством использования уязвимости в ядре Windows, допускающей выполнение вредоносного кода. После заражения системы и установления связи с сервером происходит загрузка и установка дополнительного модуля, предназначенного для сбора сведений о системе, поиска файлов, снятия скриншотов, перехвата паролей и ряда других функций. Особенностью вируса Duqu является то, что он использует объектно-ориентированный фреймворк, написан на чистом C и скомпилирован Microsoft Visual C++ с необычными настройками оптимизации [48].

Специалисты компании Symantec считают, что создатели Duqu имели доступ к исходному тексту Stuxnet и целью Duqu был сбор информации для следующей версии Stuxnet.

Regin – вирус-червь, инфицирующий компьютеры под управлением ОС Windows. Этот вирус обнаружен Лабораторией Касперского и Symantec в 2014 г. Первые сообщения об этом вирусе появились весной 2012 г., а самые ранние выявленные экземпляры датируются 2003 г. Статистика заражений вирусом Regin по странам: 28% – в России; 24% – в Саудовской Аравии; по 9% – в Мексике и Ирландии; и по 5% – в Индии, Афганистане, Иране, Бельгии, Австрии и Пакистане. Статистика по объектам заражений: 28% – телекоммуникационные компании; 48% – частные лица и малый бизнес; остальные 24% – компьютеры государственных, энергетических, финансовых и исследовательских компаний. Regin представляет собой вирус-троянец, использующий модульный подход, который позволяет ему загрузить функции, необходимые для учета индивидуальных особенностей заражаемого компьютера или сети. Структура вируса рассчитана на постоянное, долговременное целевое наблюдение за многочисленными объектами. Regin не хранит данные в файловой системе зараженного

компьютера, вместо этого он имеет свою собственную зашифрованную виртуальную файловую систему (EVFS), которая выглядит как единый файл. В качестве метода шифрования EVFS использует вариант блочного шифра RC5. Regin осуществляет коммуникации через Интернет с использованием ICMP/Ping, команд, встраиваемых в HTTP cookie и протоколов TCP и UDP, превращая заражаемую сеть в ботнет [49].

Эксперты по уровню сложности и ресурсоемкости разработки сравнивают Regin с вирусом Stuxnet, в связи с чем высказываются мнения, что вирус мог быть создан на государственном уровне в качестве многоцелевого инструмента сбора данных.

Следует отметить, что для вышеперечисленных вирусных программ: Stuxnet, Flame, Duqu, а также других боевых вирусов (Gauss, MiniFlame, MiniDuqu и др.) эксперты отмечают общие высокотехнологические черты, такие как: библиотеки (в том числе open source), среды, используемые уязвимости, приемы противодействия средствам защиты, а также качество кода. Однако среди вирусных средств встречаются и программы другой архитектуры, уровня технологичности и качества реализации. Наглядным примером может быть троянская программа Sputnik, используемая в рамках разведывательной операции Red October.

Sputnik – троянская программа, предназначения для шпионажа. Целевой функцией программы Sputnik является сбор информации, касающейся деятельности дипломатических, правительственных, научных организаций. Максимальное число заражений пришлось на Россию. Особенностями указанной вредоносной программы являются [47]:

- использование известных уязвимостей Windows-приложений (Word, Excel, Outlook) и механизма социальных атак;
- сбор нескольких десятков типов офисных, графических, почтовых, адресных файлов, в том числе удаленных;
- сбор данных со сменных носителей, дистанционных почтовых серверов и мобильных устройств (iPhone, Nokia, Windows Mobile);
- сбор параметров сетевых устройств;
- сложная распределенная система управления (около 60 доменов).

По отношению к Sputnik отмечают следующие его особенности, такие как поддержка кириллицы и сбор файлов, закрытых с помощью Cryptofiler и PGP. Sputnic использует итерационные атаки с участием людей, когда в очередных атаках используются данные, полученные ранее [47].

Несмотря на то, что Sputnik не так технологически изящен, как вирусы класса Stuxnet, указанный вирус встречается с 2007 г. по сей день [47].

Wanna Crypt – сетевой червь, специализирующийся на шифровании пользовательских данных и требовании денег за расшифровку. Этот вирус не использовался как боевое средство, а использовался злоумышленниками. Однако в основу вируса Wanna Cry был положен код вирусного средства, разработанного АНБ США и украденного у него в результате хакер-

ской атаки. Украденный код использовал эксплойт Eternal Blue в Microsoft Windows [57].

Использование в вирусе Wanna Crypt боевого ядра оказалось весьма впечатляющим. Вирус имел просто сверхвысокую скорость распространения. Во время атаки в мае 2017 г. этот вирус заразил более 75 000 компьютеров в 99 странах, заблокировав работу многих организаций. Среди них – организации национальной службы здравоохранения Великобритании, основной железнодорожный оператор Германии компания Deutsche Bahn, телефонные компании Испании и Португалии, а также различные компании из США, Китая, Италии, Вьетнама, Тайваня. В России в результате атаки вируса Wanna Crypt были дестабилизирована работа телефонных операторов «Йота», «Мегафон», «Билайн», компаний «Сбербанк», «Связной», «РЖД», нарушена работа информационно-управляющих систем МВД, МЧС и Следственного комитета. Большое количество атак этого вируса именно на государственные учреждения России, в том числе на информационные системы ее силовых ведомств позволило некоторым специалистам сделать вывод, что атака Wanna Crypt является проверкой эффективности воздействия вирусных средств против критической информационной инфраструктуры России, которая проводится спецслужбами США под прикрытием некой «хакерской группировки» [57].

Факты применения представленных выше вирусных средств показывают, что противодействие таким программам не связано исключительно с использованием антивирусных средств. Так, в современных вирусных средствах используются технологии, которые позволяет им успешно преодолевать средства антивирусной защиты. К таким технологиям относятся [47]:

- наличие программных закладок, главным образом мастер-паролей;
- наличие уязвимостей 0-дня;
- отсутствие своевременного закрытия известных уязвимостей;
- нарушения политики безопасности;
- другие факторы, связанные с недостаточностью традиционных мер защиты.

15.3.5. Проблемные вопросы использования средств информационно-технических воздействий на основе компьютерных вирусов

Необходимо отметить, что использование вирусных средств не только позволяет решать целевые задачи ВС и органам безопасности, но и требует корректности в их конфигурировании и использовании.

Как показано в работе [50], использование вирусного ПО для целей обеспечения государственной безопасности, которое не обладает широким спектром защиты собственных каналов управления, может привести к утрате контроля над этими программами и бот-сетями на их основе. Кроме

того, при использовании таких средств необходимо тщательно продумывать тактику действий вируса и допустимый уровень вреда, который он может причинять инфицируемой системе.

Так в Германии для проведения оперативно-розыскных мероприятий правоохранительными органами был использован вирус-троян типа backdoor под названием R2D2. Этот вирус был предназначен для прослушивания телефонных разговоров через Skype и перехвата шифрованных SSL-соединений. При этом троян был способен запускать произвольный код на инфицированном компьютере, к тому же в него были встроены средства для наращивания функциональности путем дополнительной установки компонентов через сеть. Например, можно было добавлять компоненты для дистанционного включения через Интернет микрофона и видеокамеры, встроенных в компьютер, и использовать их для непосредственной слежки. Однако, внедряя широкую функциональность в области оперативно-розыскных мероприятий, разработчики не снабдили это вирусное средство элементарными функциями собственной информационной безопасности. Так, шифрование передаваемых данных об итогах работы R2D2 происходит лишь в одну сторону – в зашифрованном виде отсылаются лишь снимки экрана и аудиофайлы перехвата; при этом во всех версиях трояна применяется один и тот же ключ, который жестко встроен в код. Команды управления поступают к трояну в отрытом виде. При этом ни управляющие команды для троянца, ни его ответные сигналы совершенно никак не аутентифицированы и не содержат никакой защиты для обеспечения целостности соединения [50].

Эти факты позволили экспертному сообществу продемонстрировать перехват управления сетью вирусов-троянов R2D2. В результате успешного перехвата управления можно либо использовать получаемые от вирусов-троянов данные по своему усмотрению, либо фальсифицировать их с последующей передачей правоохранительным органам. Кроме этого, возможен сценарий, при котором серверные информационные системы правоохранительных органов могут быть атакованы через слабо защищенный служебный канал управления троянами [50].

Кроме вышесказанных собственных уязвимостей, троян R2D2 отличался тем, что существенно снижал уровень информационной безопасности инфицируемой системы, тем самым делая возможным несанкционированный доступ к ее информационным ресурсам со стороны других нарушителей [50].

Еще одним проблемным вопросом при использовании вирусных средств является контроль над профессиональными способами их создания, применения и распространения со стороны государственных служб.

В мае 2017 г. произошла одна из крупнейших мировых атак вирусом Wanna Crypt. Данный вирус продемонстрировал сверхбыструю скорость распространения и высокую результативность поражения пользовательских данных. За 2 недели своего функционирования Wanna Crypt заразил более 75 000 компьютеров в 99 странах, заблокировав работу многих орга-

низаций. Отличительной особенностью вируса Wanna Cry является то, что в его основу был положен код вирусного средства, разработанного АНБ США и украденного у него в результате хакерской атаки [57].

Низкая стоимость разработки и наличие большого количества документации по принципам функционирования критической информационной инфраструктуры делают весьма вероятными разработку и использование вирусных средств со стороны иррегулярных воинских формирований и террористических групп и организованной преступности для проведения акций информационной войны.

Как отмечается в работе [52], в настоящее время уже имеются профессиональные платформы для создания вирусных средств, разработка которых финансируется международной организованной преступностью.

Одним из таких вирусных средств является троян CosmicDuke, собранный на платформе BotGenStudio, которая позволяет индивидуально сконфигурировать трояна-шпиона с учетом особенностей цели, против которой ведется шпионаж, а также выпускать индивидуальное обновление для этого вируса. В зависимости от настроек троян CosmicDuke может использовать различные способы для маскировки в информационной системе, собирать различные наборы данных и отправлять их несколькими способами. Троян CosmicDuk маскируется под легитимные приложения, которые санкционированно обращаются к Интернету: агенты обновления Java, Acrobat, Chrome и др. Троян CosmicDuke способен копировать документы разных типов, следить за клавиатурой, делать снимки экрана, получать доступ к адресным книгам из почтовых приложений и к паролям, сохраненным в системе и популярных мессенджерах, а также к файлам сертификатов безопасности. Собранная информация передается на управляющие серверы несколькими способами: по FTP и тремя вариантами HTTP-взаимодействия. При этом CosmicDuke использует все возможности для продолжения непрерывного функционирования – к примеру, он даже умеет запускаться через планировщик задач операционной системы [52].

Аналитики Лаборатории Касперского полагают, что платформа BotGenStudio может быть создана не только для нужд разработавшей ее группы, но и для продажи узкому кругу заказчиков [52].

Таким образом, уже сейчас наблюдается процесс, в результате которого иррегулярные воинские формирования, террористические группы и организованная преступность могут перейти от использования вирусов в качестве средств шпионажа и хищения финансовой информации к созданию этих вирусных средств на основе профессиональных технологий, а в дальнейшем – их применения для организации высокоэффективных терактов, ориентированных на информационные системы государственного и военного управления, энергосети, транспортную инфраструктуру и особо опасные промышленные производства.

15.4. Программные закладки

Программная закладка – скрытно внедренная в защищенную информационную систему программа, либо намеренно измененный фрагмент программы, которые позволяют осуществить несанкционированный доступ к ресурсам системы на основе изменения свойств подсистемы защиты. При этом в большинстве случаев закладка внедряется самим разработчиком ПО для реализации в информационной системе некоторых сервисных или недекларируемых функций.

Программные закладки, получая несанкционированный доступ к данным в памяти информационной системы, перехватывают их. После перехвата эти данные копируются и сохраняются в специально созданных разделах памяти или передаются по сети. Программные закладки, подобно вирусам, могут искажать или уничтожать данные, но, в отличие от вирусов, деструктивное действие таких программ, как правило, более выборочно и направлено на конкретные данные. Довольно часто программные закладки играют роль перехватчиков паролей, сетевого трафика, а также служат в качестве скрытых интерфейсов для входа в систему. Однако, в отличие от вирусов, программные закладки не обладают способностью к саморазмножению, они встраиваются в ассоциированное с ними программное обеспечение и латентно функционируют вместе с ним. При этом особенностью закладок, внедренных на стадии разработки ПО, является то, что они становятся фактически неотделимы от прикладных или системных программ информационной системы.

Классификация программных закладок представлена на рис. 15.6.

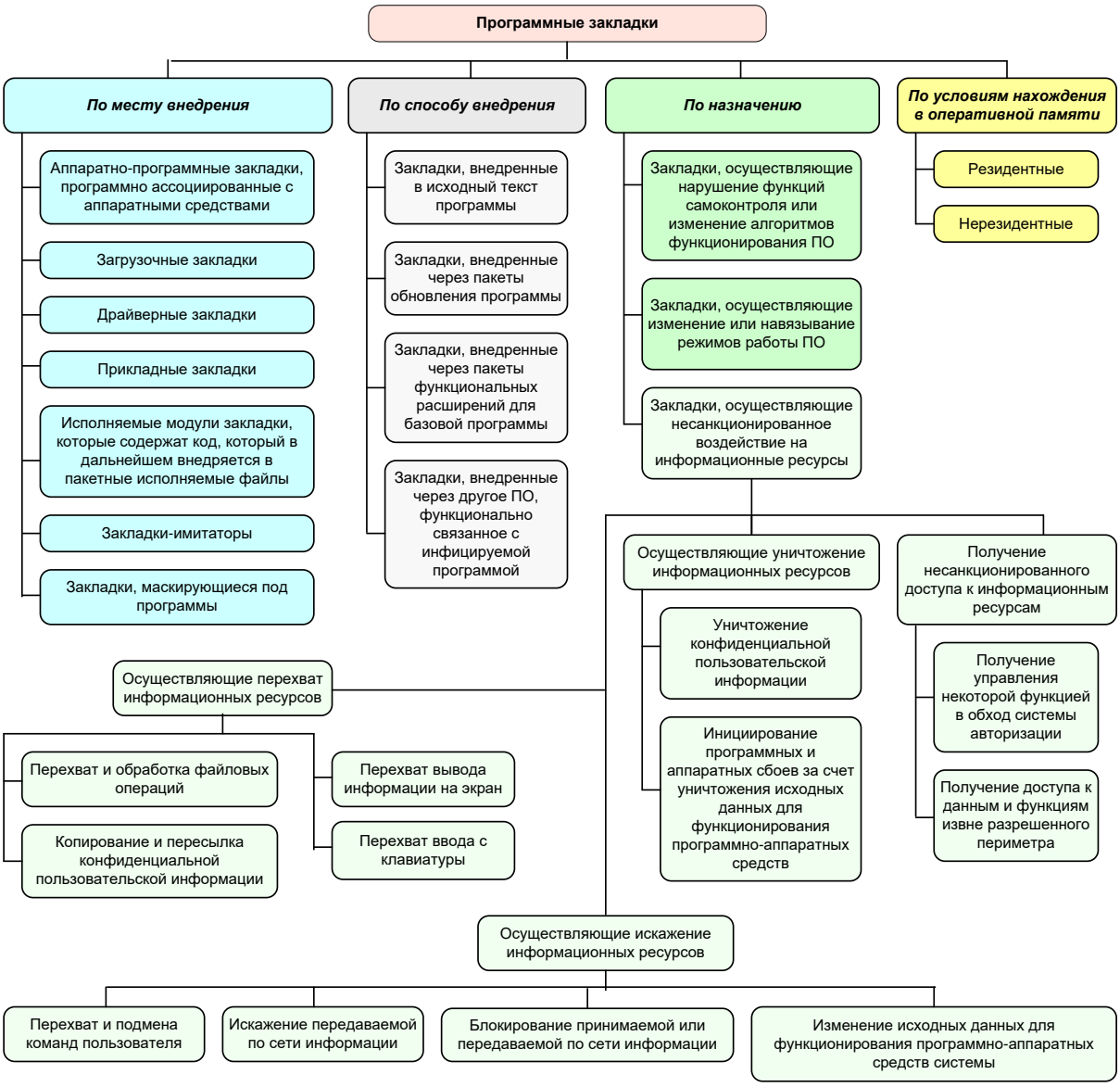


Рис. 15.6. Классификация программных закладок

Как и вирус, программная закладка должна скрывать свое присутствие в программной среде информационной системы. Однако программные закладки невозможно обнаружить при помощи стандартных антивирусных средств, их выявление возможно только специальными тестовыми программами, выявляющими аномальное поведение и недекларируемые возможности ПО. В связи с этим средства маскировки программных закладок преимущественно ориентированы на противодействие отладчикам программ, анализаторам кода и дисассемблерам. В качестве одного из широко применяемых способов маскировки является обфускация (запутывание) программ, в которые внедрена закладка.

К отдельным типам программных закладок можно отнести:

- *логические бомбы* – характеризуются способностью при выполнении заранее заложенных в них условий (конкретный день, время суток, определенное действие пользователя или команда извне) выполнять несанкционированные действия по уничтожению или искажению информации, воспрещения доступа к тем или иным

важным фрагментам информационного ресурса, либо дезорганизации работы технических средств;

- *люки* (*backdoor*) – программы, находящие или создающие уязвимости в защите информационной системы с целью дальнейшего предоставления удаленного несанкционированного доступа к ней.

15.5. Аппаратные закладки

Аппаратная закладка – электронное устройство, скрытно внедряемое к остальным элементам, которое способно вмешаться в работу аппаратных или технических средств информационной системы.

Результатом работы аппаратной закладки может быть, как полное выведение системы из строя, так и нарушение ее нормального функционирования, например, несанкционированный доступ к информации, ее изменение или блокирование.

Классификация аппаратных закладок приведена на рис. 15.7.

Схематическая сложность современного микроэлектронного оборудования, тенденции к миниатюризации его элементов ведут к тому, что производители такого оборудования могут бескомпроматно и практически неограниченно наращивать функциональные возможности аппаратных закладок, функционирующих в интересах тестирования такого оборудования, а при подключении устройств к глобальной сети – осуществлять обновление алгоритма их функционирования, а также условий срабатывания. Краткая характеристика технологий современных аппаратных закладок представлена в таблице 15.1.

Таблица 15.1 – Технологии современных аппаратных закладок

<table>
<tr><th>Методы внедрения</th><th>Методы обнаружения</th><th>Методы маскировки</th></tr>
<tr><td rowspan="2">Встраивание закладок в технологию микроядра управления в современных СБИС, построенного на уникальном списке команд (управление основной работой и блокировка и замена неисправных узлов для продления срока службы СБИС)</td><td>Технологии послойного сканирования кристаллов</td><td>Механизм технологической защиты топологии кристалла от послойного сканирования (впервые внедрен в i486)</td></tr>
<tr><td>Вычитывание и дизассемблирование аппаратно доступных микрокодов</td><td rowspan="2">Размещение микроядер с закладками и ресурсов памяти в области, недоступной пользователю. Шифрование (мутирование) участков кода, антитрассировка</td></tr>
<tr><td>Виртуализация вычислений</td><td>Анализ контента проходящих по сети данных</td></tr>
<tr><td>Встраивание целевых микроядер и узлов, реализующих стратегию влияния</td><td>Мониторинг аномальной активности платформы. Радио-мониторинг. Электромагнитный контроль.</td><td></td></tr>
</table>

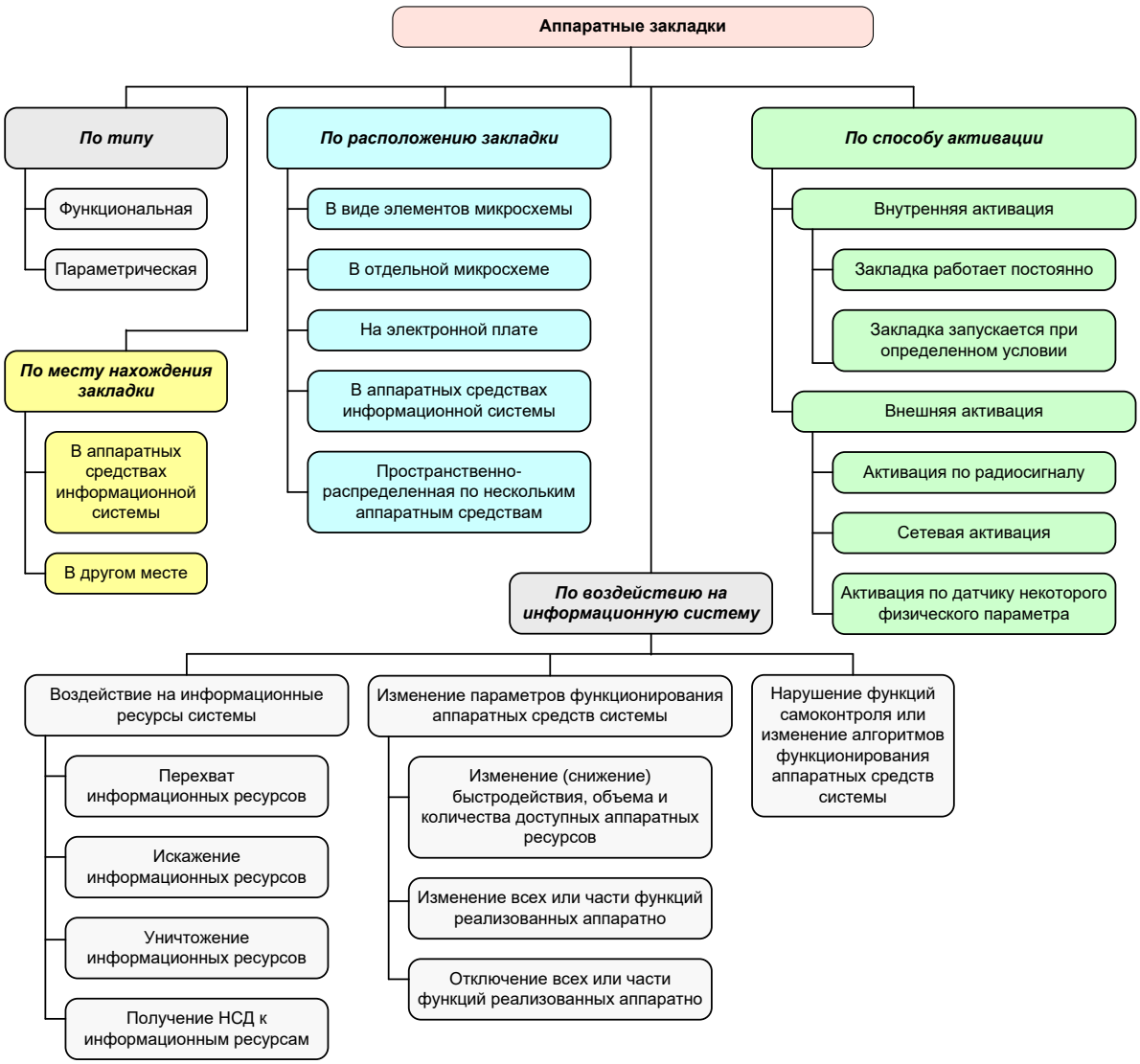


Рис. 15.7. Классификация аппаратных закладок

16. Механизмы реализации удаленных атак в глобальной телекоммуникационной сети Internet

В настоящее время отдельный ТКС зачастую интегрируются через сеть Internet. При этом, такое подключение к сети Internet создают угрозы для реализации ИТВ типа «уделенные сетевые атаки» на ТКС. Возможности по реализации удаленных сетевых атак в сети Internet настолько многообразны, что рассмотреть их все не представляется возможным. Цель данного раздела, не производя исчерпывающего анализа механизмов отдельных удаленных сетевых атак, на отдельных простых примерах продемонстрировать реализации типовых уязвимостей рабочих станций в сети Internet.

В основу рассмотренных примеров положены типовые удаленные сетевые атаки, представленные в Базовой модели угроз безопасности персональных данных при их обработке в информационных системах [100], утвержденной ФСТЭК России.

16.1. Анализ сетевого трафика

В сети Internet основными базовыми протоколами удаленного доступа являются TELNET и FTP (File Transfer Protocol). TELNET – это протокол виртуального терминала (ВТ), позволяющий с удаленных хостов подключаться к серверам Internet в режиме ВТ. FTP-протокол, предназначенный для передачи файлов между удаленными хостами. Для получения доступа к серверу по этим протоколам пользователю необходимо пройти на нем процедуру идентификации и аутентификации. В качестве информации, идентифицирующей пользователя, выступает его идентификатор (имя), а для аутентификации используется пароль. Особенностью протоколов FTP и TELNET является то, что пароли и идентификаторы пользователей передаются по сети в открытом, незашифрованном виде. Таким образом, необходимым и достаточным условием для получения удаленного доступа к хостам по протоколам FTP и TELNET являются имя и пароль пользователя.

Одним из способов получения паролей и идентификаторов пользователей в сети Internet является анализ сетевого трафика. Сетевой анализ осуществляется с помощью специальной программы – анализатора пакетов, перехватывающей все пакеты, передаваемые по сегменту сети, и выделяющей среди них те, в которых передаются идентификатор пользователя и его пароль (рис. 16.1). Сетевой анализ протоколов FTP и TELNET показывает, что TELNET разбивает пароль на символы и пересылает их по одному, помещая каждый символ из пароля в соответствующий пакет, а FTP, напротив, пересылает пароль целиком в одном пакете.

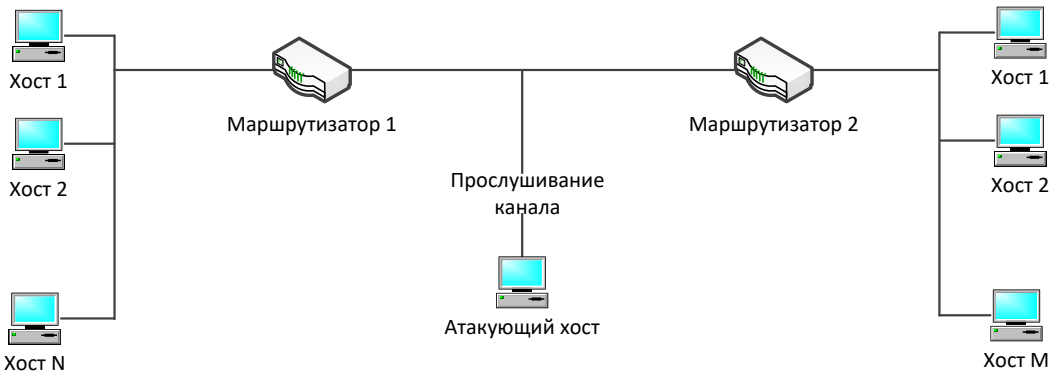


Рис. 16.1. Анализ сетевого трафика

16.2. Ложный ARP-сервер

В вычислительных сетях связь между двумя удаленными хостами осуществляется путем передачи по сети сообщений, которые заключены в пакеты обмена. В общем случае передаваемый по сети пакет независимо от используемого протокола и типа сети (Token Ring, Ethernet, X.25 и др.) состоит из заголовка пакета и поля данных. В заголовок пакета обычно заносится служебная информация, определяемая используемым протоколом обмена и необходимая для адресации пакета, его идентификации, преобразования и т. д. В поле данных помещаются либо непосредственно данные, либо другой пакет более высокого уровня OSI.

Так, например, пакет транспортного уровня может быть вложен в пакет сетевого уровня, который, в свою очередь, вложен в пакет канального уровня. Таким образом, пакет TCP (транспортный уровень) вложен в пакет IP (сетевой уровень), который, в свою очередь, вложен в пакет Ethernet (канальный уровень). Схема на рис. 16.2 наглядно иллюстрирует как выглядит, например, TCP-пакет в сети Internet.

Ethernet-заголовок
IP-заголовок
TCP-заголовок
Данные

Рис. 16.2. Структура TCP-пакета

Базовым сетевым протоколом обмена в сети Internet является протокол IP (Internet Protocol). Протокол IP – это межсетевой протокол, позволяющий передавать IP-пакеты в любую точку глобальной сети. Для адресации на сетевом уровне (IP-уровне) в сети Internet каждый хост имеет уникальный 32-разрядный IP-адрес. Для передачи IP-пакета на хост необходимо указать в IP-заголовке пакета в поле Destination Address IP-адрес этого хоста. Однако, как видно из рис. 16.2, IP-пакет находится внутри

Ethernet-пакета, поэтому каждый пакет в конечном счете адресуется на аппаратный адрес сетевого адаптера, непосредственно осуществляющего прием и передачу пакетов в сеть (при рассмотрении Ethernet-сети).

То есть, для адресации IP-пакетов в сети Internet кроме IP-адреса хоста необходим еще либо Ethernet-адрес его сетевого адаптера (в случае адресации внутри одной подсети), либо Ethernet-адрес маршрутизатора (в случае межсетевой адресации). Первоначально хост может не иметь информации о Ethernet-адресах других хостов, находящихся с ним в одном сегменте, в том числе и о Ethernet-адресе маршрутизатора. Следовательно, перед хостом встает стандартная проблема, решаемая с помощью алгоритма удаленного поиска.

В сети Internet для решения проблемы удаленного поиска Ethernet-адресов используется протокол ARP (Address Resolution Protocol). Протокол ARP позволяет получить взаимно однозначное соответствие IP- и Ethernet-адресов для хостов, находящихся внутри одного сегмента.

Это достигается следующим образом.

- При первом обращении к сетевым ресурсам хост отправляет широковещательный ARP-запрос на Ethernet-адрес FFFFFFFFFFFFh, в котором указывает IP-адрес маршрутизатора и просит сообщить его Ethernet-адрес. Этот широковещательный запрос получат все станции в сегменте сети, в том числе и маршрутизатор.
- Получив такой запрос, маршрутизатор внесет запись о запросившем хосте в свою ARP-таблицу, а затем отправит на запросивший хост ARP-ответ, в котором сообщит свой Ethernet-адрес.
- Полученный в ARP-ответе Ethernet-адрес будет занесен в ARP-таблицу, находящуюся в памяти операционной системы на запросившем хосте и содержащую записи соответствия IP- и Ethernet-адресов для хостов внутри одного сегмента.

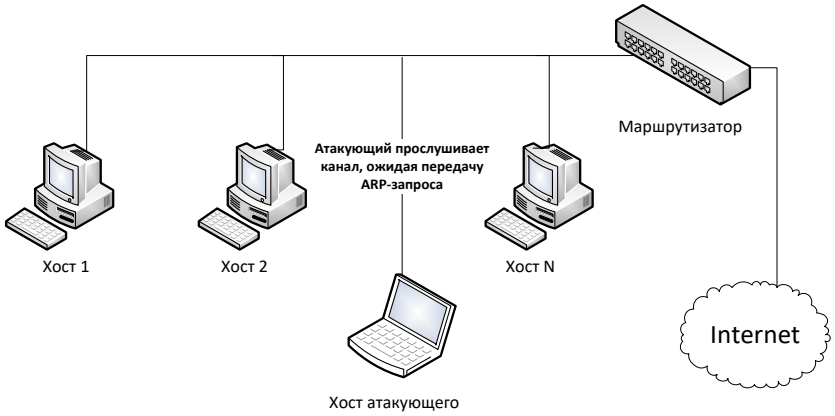
В случае использования в сети алгоритмов удаленного поиска существует возможность осуществления в такой сети типовой удаленной атаки «Ложный объект сети». Из анализа безопасности протокола ARP становится ясно, что, перехватив на атакующем хосте внутри конкретного сегмента сети широковещательный ARP-запрос, можно послать ложный ARP-ответ, в котором объявить себя искомым хостом (например, маршрутизатором), и в дальнейшем активно контролировать и воздействовать на сетевой трафик «обманутого» хоста по схеме «Ложный объект сети».

Рассмотрим обобщенную функциональную схему ложного ARP-сервера (рис. 16.3):

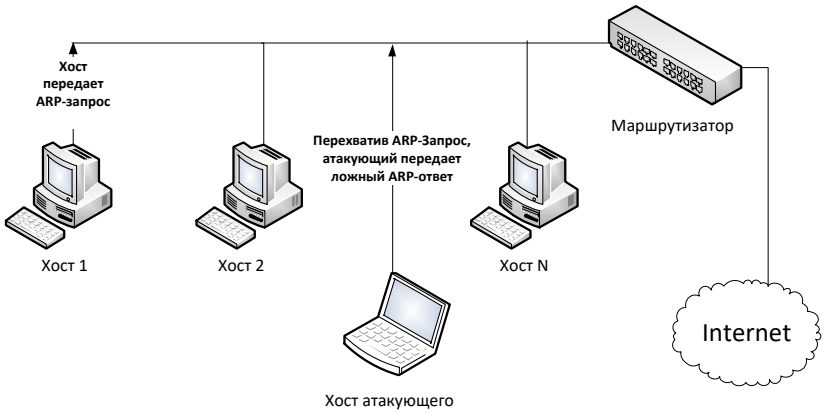
- ожидание ARP-запроса;
- при получении ARP-запроса передача по сети на запросивший хост ложного ARP-ответа, в котором указывается адрес сетевого адаптера атакующей станции (ложного ARP-сервера) или тот Ethernet-адрес, на котором будет принимать пакеты ложный ARP-сервер (совершенно необязательно указывать в ложном ARP-ответе свой настоящий Ethernet-адрес, так как при работе

непосредственно с сетевым адаптером его можно запрограммировать на прием пакетов на любой Ethernet-адрес);

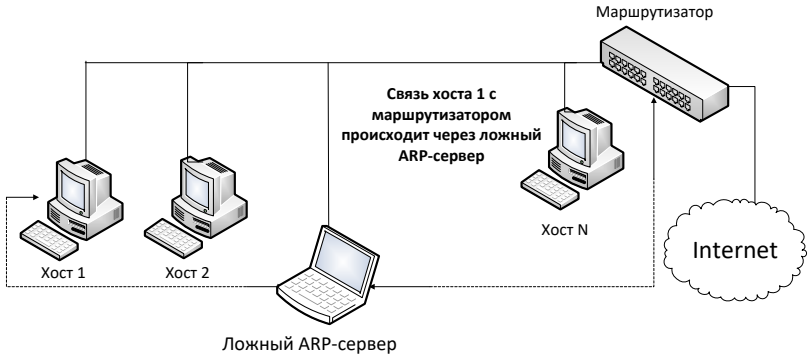
- прием, анализ, воздействие и передача пакетов обмена между взаимодействующими хостами, а также по возможности воздействие на перехваченную информацию.



а) Фаза ожидания ARP-запроса



б) Фаза атаки



в) Фаза приема, анализа, воздействия и передачи перехваченной информации на ложном ARP-сервере

Рис. 16.3. Ложный ARP-сервер

В заключение необходимо отметить, что, во-первых, причина успеха такой удаленной атаки кроется, не столько в Internet, сколько в широковещательной среде Ethernet и, во-вторых, очевидно, что эта удаленная атака

является внутрисегментной и поэтому представляет угрозу только в случае нахождения атакующего внутри вашего сегмента сети.

16.3. Ложный DNS-сервер

Как известно, для обращения к хостам в сети Internet используются 32-разрядные IP-адреса, уникально идентифицирующие каждый сетевой компьютер в этой глобальной сети. Однако, для пользователей применение IP-адресов при обращении к хостам является не слишком удобным и далеко не самым наглядным. Использование в Internet породило проблему преобразования имен в IP-адреса. Такое преобразование необходимо, так как на сетевом уровне адресация пакетов идет не по именам, а по IP-адресам, следовательно, для непосредственной адресации сообщений в Internet имена не годятся. Для решения задачи преобразования мнемонически понятных для пользователей имен в IP-адреса была создана система преобразования имен, позволяющая хосту в случае отсутствия у него информации о соответствии имен и IP-адресов получить необходимые сведения от ближайшего информационно-поискового сервера – DNS (Domain Name System)-сервера.

Основной задачей, решаемой службой DNS является поиск по имени удаленного хоста его IP-адреса, который и необходим для непосредственной адресации.

Рассмотрим DNS-алгоритм удаленного поиска IP-адреса по имени в сети Internet.

- Хост посылает на IP-адрес ближайшего DNS-сервера (он устанавливается при настройке сетевой ОС) DNS-запрос, в котором указывает имя сервера, IP-адрес которого необходимо найти.
- DNS-сервер, получив запрос, просматривает свою базу имен на наличие в ней указанного в запросе имени. В случае, если имя найдено, а, следовательно, найден и соответствующий ему IP-адрес, то на запросивший хост DNS-сервер отправляет DNS-ответ, в котором указывает искомый IP-адрес.
- В случае, если указанное в запросе имя DNS-сервер не обнаружил в своей базе имен, то DNS-запрос отсылается DNS-сервером на один из корневых DNS-серверов и описанная в этом пункте процедура повторяется, пока имя не будет найдено (или не найдено).

Анализируя с точки зрения безопасности уязвимость этой схемы удаленного поиска с помощью протокола DNS, можно сделать вывод о возможности осуществления в сети, использующей протокол DNS, типовой удаленной атаки «Ложный объект сети». Практические изыскания и критический анализ безопасности службы DNS позволяют предложить три возможных варианта удаленной атаки на эту службу.

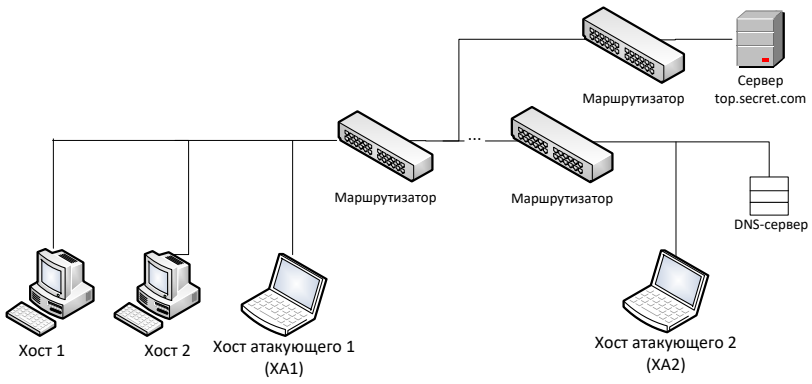
16.3.1. Внедрение в сеть ложного DNS-сервера путем перехвата DNS-запроса

Для реализации атаки путем перехвата DNS-запроса атакующему необходимо перехватить DNS-запрос, извлечь из него номер UDP-порта отправителя запроса, двухбайтовое значение ID идентификатора DNS-запроса и искомое имя и затем послать ложный DNS-ответ на извлеченный из DNS-запроса UDP-порт, в котором указать в качестве искомого IP-адреса настоящий IP-адрес ложного DNS-сервера. Это позволит в дальнейшем полностью перехватить трафик между атакуемым хостом и сервером и активно воздействовать на него по схеме «Ложный объект сети».

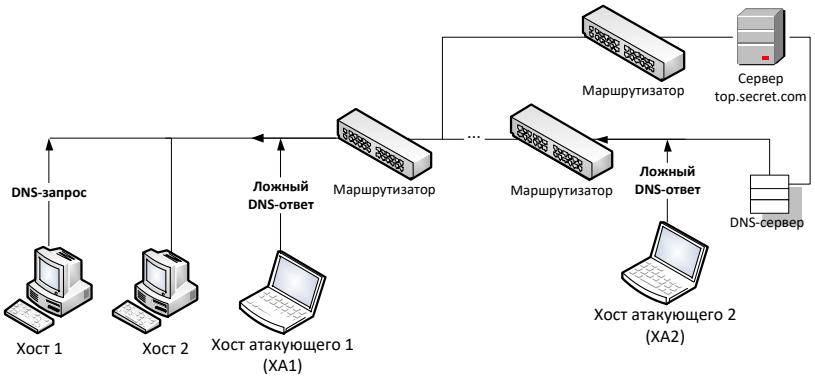
Рассмотрим обобщенную схему работы ложного DNS-сервера (рис. 16.4):

- ожидание DNS-запроса;
- извлечение из полученного запроса необходимых сведений и передача по сети на запросивший хост ложного DNS-ответа, от имени (с IP-адреса) настоящего DNS-сервера, в котором указывается IP-адрес ложного DNS-сервера;
- в случае получения пакета от хоста, изменение в IP-заголовке пакета его IP-адреса на IP-адрес ложного DNS-сервера и передача пакета на сервер (то есть ложный DNS-сервер ведет работу с сервером от своего имени);
- в случае получения пакета от сервера, изменение в IP-заголовке пакета его IP-адреса на IP-адрес ложного DNS-сервера и передача пакета на хост (для хоста ложный DNS-сервер и есть настоящий сервер).

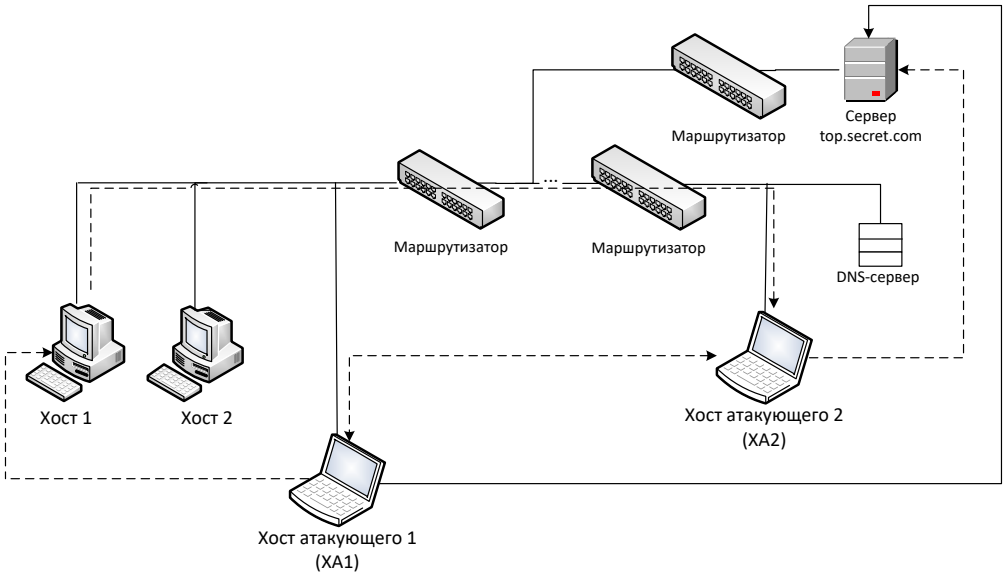
Необходимым условием осуществления такого варианта атаки является перехват DNS-запроса. Это возможно только в том случае, если атакующий находится либо на пути основного трафика, либо в сегменте настоящего DNS-сервера. Выполнение одного из этих условий местонахождения атакующего в сети делает подобную удаленную атаку трудно осуществимой на практике. Однако в случае выполнения этих условий возможно осуществить межсегментную удаленную атаку на сеть Internet.



а) Фаза ожидания атакующим DNS-запроса
(он находится на ХА1, либо на ХА2)



б) Фаза передачи атакующим ложного DNS-ответа



в) Фаза приема, анализа, воздействия и передачи перехваченной информации на ложном сервере

Рис. 16.4. Функциональная схема ложного DNS-сервера

16.3.2. Внедрение в сеть ложного сервера путем создания направленного «шторма» ложных DNS-ответов на атакуемый хост

Другой вариант осуществления удаленной атаки, направленной на службу DNS, основан на второй разновидности типовой УА «Ложный объект сети» (при использовании недостатков алгоритмов удаленного поиска). В этом случае атакующий осуществляет постоянную передачу на атакуемый хост заранее подготовленного ложного DNS-ответа от имени настоящего DNS-сервера без приема DNS-запроса! Другими словами, атакующий создает в сети Internet направленный «шторм» ложных DNS-ответов.

Это возможно, так как обычно для передачи DNS-запроса используется протокол UDP, в котором отсутствуют средства идентификации пакетов.

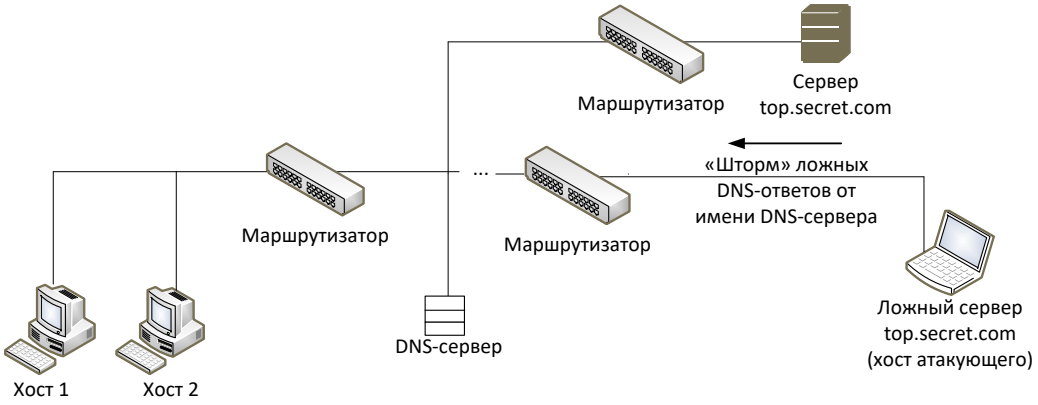
Критериями, предъявляемыми сетевой ОС хоста к полученному от DNS-сервера ответу, является:

- совпадение IP-адреса отправителя ответа с IP-адресом DNS-сервера;
- DNS-ответ должен содержать то же имя, что и в DNS-запросе;
- DNS-ответ должен быть направлен на тот же UDP-порт, с которого был послан DNS-запрос (в данном случае это первая проблема для атакующего);
- в DNS-ответе поле идентификатора запроса в заголовке DNS (ID) должно содержать то же значение, что и в переданном DNS-запросе (а это вторая проблема).

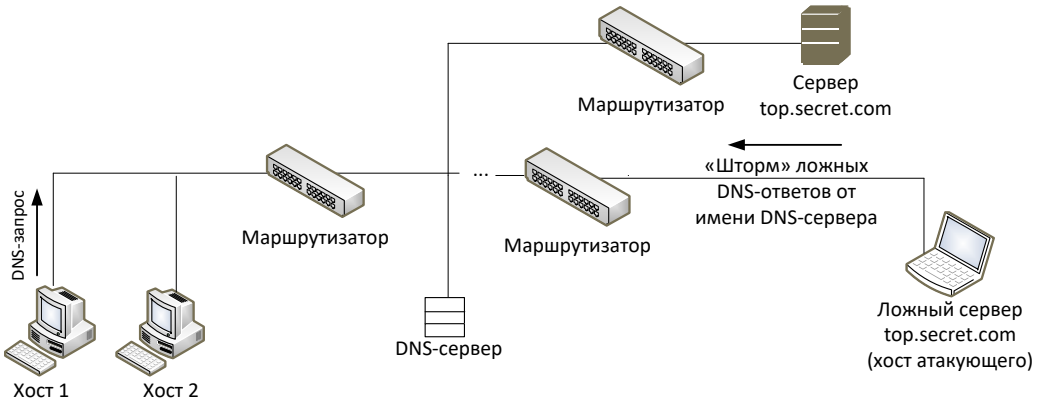
В данном случае, так как атакующий не имеет возможности перехватить DNS-запрос, то основную проблему для него представляет номер UDP-порта, с которого был послан запрос. Однако, как было отмечено ранее, номер порта отправителя принимает ограниченный набор значений (≥ 1023), поэтому атакующему достаточно действовать простым перебором, направляя ложные ответы на соответствующий перечень портов. На первый взгляд, второй проблемой может быть двухбайтовый идентификатор DNS-запроса, но, в связи с особенностями функционирования протокола DNS он либо равен единице, либо в случае DNS-запроса от Netscape Navigator (например) имеет значение близкое к нулю (один запрос – ID увеличивается на 1).

Поэтому для осуществления такой удаленной атаки атакующему необходимо выбрать интересующий его хост (например, top.secret.com), маршрут к которому требуется изменить так, чтобы он проходил через ложный сервер – хост атакующего. Это достигается постоянной передачей (направленным «штормом») атакующим ложных DNS-ответов на атакуемый хост от имени настоящего DNS-сервера на соответствующие UDP-порты. В этих ложных DNS-ответах указывается в качестве IP-адреса хоста top.secret.com IP-адрес атакующего.

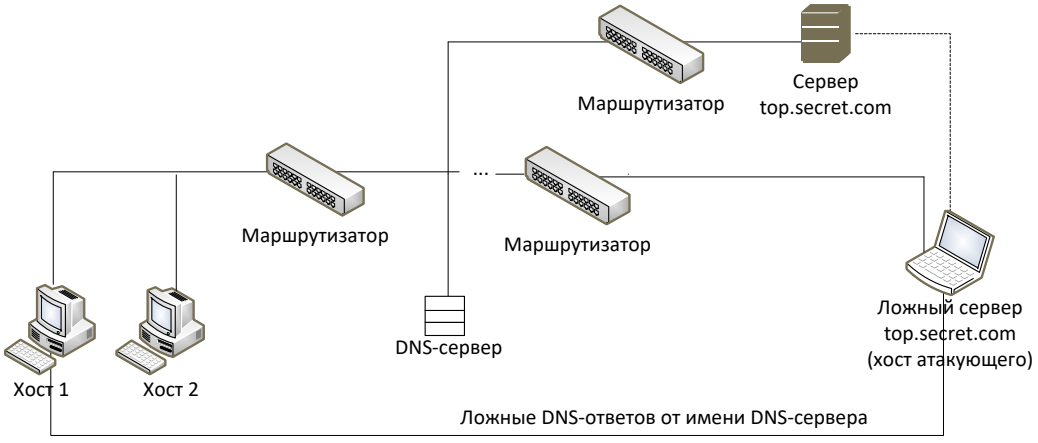
Далее атака развивается по следующей схеме. Как только цель атаки (атакуемый хост) обратится по имени к хосту top.secret.com, то от этого хоста в сеть будет передан DNS-запрос, который атакующий никогда не получит, но этого ему и не требуется, так как на хост сразу же поступит постоянно передаваемый ложный DNS-ответ, что и будет воспринят ОС атакуемого хоста как настоящий ответ от DNS-сервера. Все! Атака состоялась, и теперь атакуемый хост будет передавать все пакеты, предназначенные для top.secret.com, на IP-адрес хоста атакующего, который, в свою очередь, будет переправлять их на top.secret.com, воздействуя на перехваченную информацию по схеме «Ложный объект сети».



а) Атакующий создает направленный «шторм» ложных DNS-ответов на Хост 1



б) Хост 1 посылает DNS-запрос и немедленно получает ложный DNS-ответ



в) Фаза приема, анализа, воздействия и передачи перехваченной информации на ложном сервере

Рис. 16.5. Внедрение в Internet ложного сервера путем создания направленного «шторма» ложных DNS-ответов на атакуемый хост

Рассмотрим функциональную схему предложенной удаленной атаки на службу DNS:

- постоянная передача атакующим ложных DNS-ответов на атакуемый хост на различные UDP-порты и, возможно, с различными ID, от имени (с IP-адреса) настоящего DNS-сервера с указанием имени интересующего хоста и его ложного IP-адреса, которым будет являться IP-адрес ложного сервера – хоста атакующего;
- в случае получения пакета от хоста, изменение в IP-заголовке пакета его IP-адреса на IP-адрес атакующего и передача пакета на сервер (то есть ложный сервер ведет работу с сервером от своего имени – со своего IP-адреса);
- в случае получения пакета от сервера, изменение в IP-заголовке пакета его IP-адреса на IP-адрес ложного сервера и передача пакета на хост (для хоста ложный сервер и есть настоящий сервер).

Таким образом, реализация такой удаленной атаки, использующей пробелы в безопасности службы DNS, позволяет из любой точки сети Internet нарушить маршрутизацию между двумя заданными объектами (хостами)! Данная удаленная атака осуществляется межсегментно по отношению к цели атаки и угрожает безопасности любого хоста Internet, использующего обычную службу DNS.

16.3.3. Внедрение в сеть ложного сервера путем перехвата DNS-запроса или создания направленного «шторма» ложных DNS-ответов на атакуемый DNS-сервер

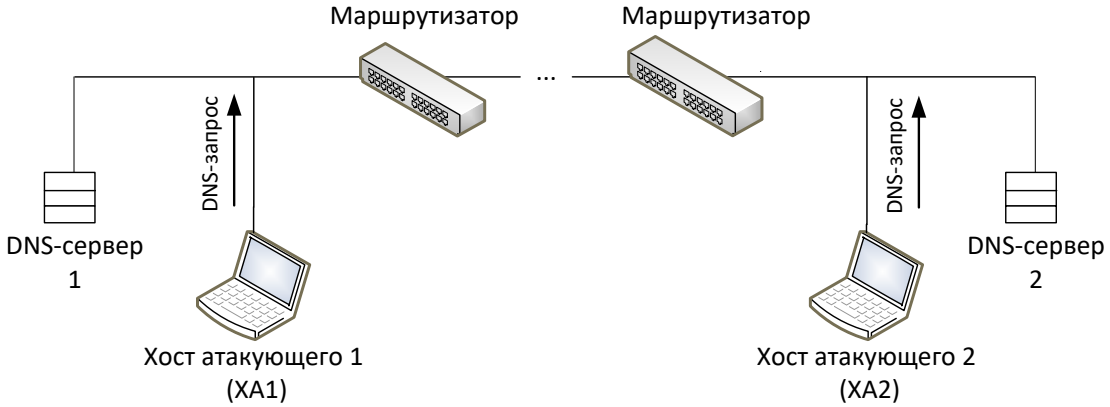
Из рассмотренной схемы удаленного DNS-поиска следует, что в том случае, если указанное в запросе имя DNS-сервер не обнаружил в своей базе имен, то запрос отсылается сервером на один из корневых DNS-серверов, адреса которых содержатся в файле настроек сервера root.cache.

Итак, в случае если DNS-сервер не имеет сведений о запрашиваемом хосте, то он сам, пересылая запрос далее, является инициатором удаленного DNS-поиска. Поэтому ничто не мешает атакующему, действуя описанными в предыдущих пунктах методами, перенести свою атаку непосредственно на DNS-сервер. В качестве цели атаки теперь будет выступать не хост, а DNS-сервер и ложные DNS-ответы будут направляться атакующим от имени корневого DNS-сервера на атакуемый DNS-сервер.

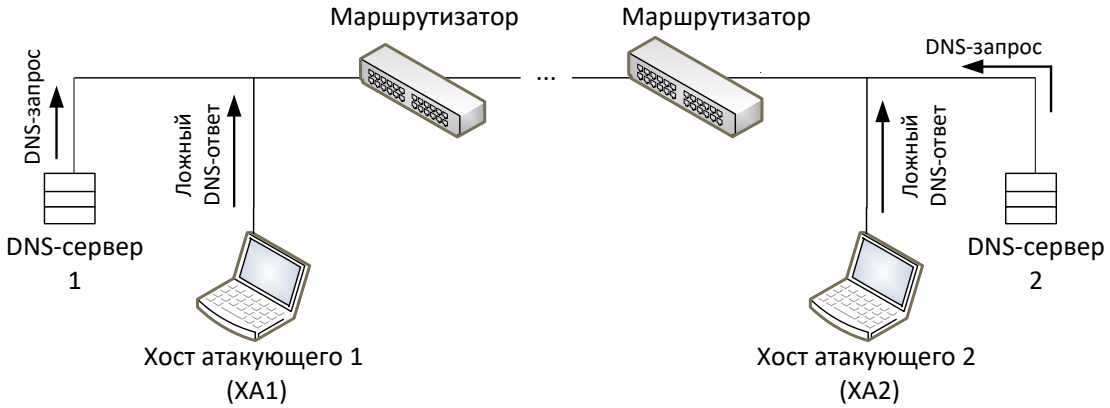
При этом важно учитывать следующую особенность работы DNS-сервера. Для ускорения работы каждый DNS-сервер кэширует в области памяти свою таблицу соответствия имен и IP-адресов хостов. В том числе в кэш заносится динамически изменяемая информация об именах и IP-адресах хостов, найденных в процессе функционирования DNS-сервера, а именно, если DNS-сервер, получив запрос, не находит у себя в кэш-таблице соответствующей записи, он пересылает ответ на следующий сервер и, получив ответ, заносит найденные сведения в кэш-таблицу в память.

Таким образом, при получении следующего запроса DNS-серверу уже не требуется вести удаленный поиск, так как необходимые сведения уже находятся у него в кэш-таблице.

Из анализа только что описанной схемы удаленного DNS-поиска становится очевидно, что в том случае, если в ответ на запрос от DNS-сервера атакующий направит ложный DNS-ответ (или в случае «шторма» ложных ответов будет вести их постоянную передачу), то в кэш-таблице сервера появится соответствующая запись с ложными сведениями и в дальнейшем все хосты, обратившиеся к этому DNS-серверу, будут дезинформированы, и при обращении к хосту, маршрут к которому атакующий решил изменить, связь с ним будет осуществляться через хост атакующего по схеме «ложный объект сети». И, что хуже всего, с течением времени эта ложная информация, попавшая в кэш DNS-сервера, будет распространяться на соседние DNS-серверы высших уровней, а, следовательно, все больше хостов в Internet будут дезинформированы и атакованы!



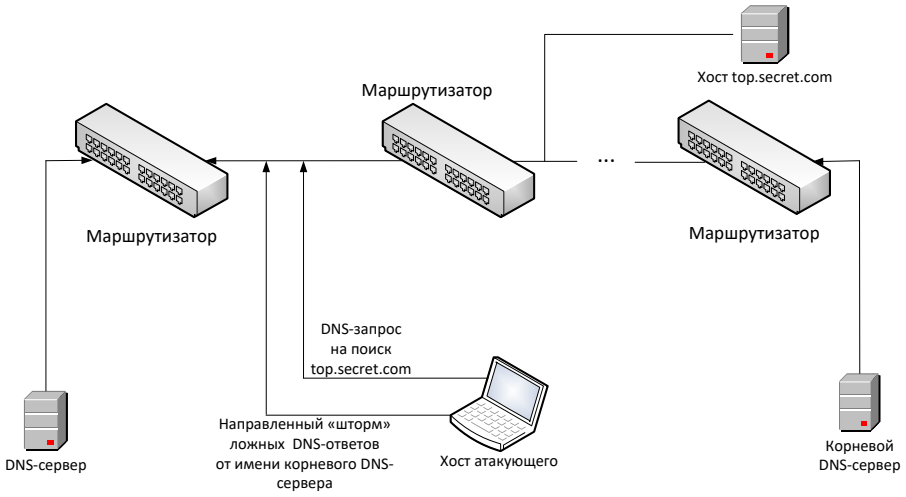
а) Фаза ожидания атакующим DNS-запроса от DNS-сервера (для ускорения атакующий генерирует необходимый DNS-запрос)



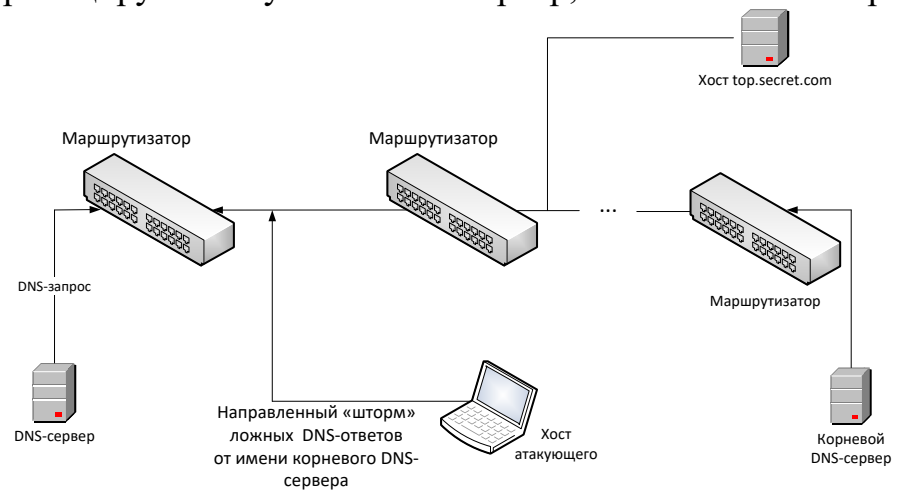
б) Фаза передачи атакующим ложного DNS-ответа на DNS-сервер 1

Рис. 16.6. Внедрение в Internet ложного сервера путем перехвата DNS-запроса от DNS-сервера

В том случае, если атакующий не может перехватить DNS-запрос от DNS-сервера, то для реализации атаки ему необходим «шторм» ложных DNS-ответов, направленный на DNS-сервер. При этом возникает следующая проблема, отличная от проблемы подбора портов в случае атаки, направленной на хост. Как уже отмечалось ранее, DNS-сервер, посылая запрос на другой DNS-сервер, идентифицирует этот запрос двухбайтовым значением (ID). Это значение увеличивается на единицу с каждым передаваемым запросом. Узнать атакующему это текущее значение идентификатора DNS-запроса не представляется возможным. Поэтому предложить что-либо, кроме перебора 2^{16} возможных значений ID, достаточно сложно. Зато исчезает проблема перебора портов, так как все DNS-запросы передаются DNS-сервером на 53 порт.



а) Атакующий создает направленный «шторм» ложных DNS-ответов от имени одного из корневых DNS-серверов и при этом провоцирует атакуемый DNS-сервер, посылая DNS-запрос



б) DNS-сервер передает DNS-запрос на корневой DNS-сервер и немедленно получает ложный DNS-ответ от атакующего

Рис. 16.7. Внедрение в Internet ложного сервера путем создания направленного «шторма» ложных DNS-ответов на атакуемый DNS-сервер

Следующая проблема, являющаяся условием осуществления этой удаленной атаки на DNS-сервер при направленном «шторме» ложных DNS-ответов, состоит в том, что атака будет иметь успех только в случае, если DNS-сервер пошлет запрос на поиск имени, которое содержится в ложном DNS-ответе. DNS-сервер посылает этот столь необходимый и желанный для атакующего запрос в том и только том случае, когда на него приходит DNS-запрос от какого-либо хоста на поиск именно этого имени и такого имени не оказывается в кэш-таблице DNS-сервера. В принципе, этот запрос может возникнуть, когда угодно, и атакующему придется ждать результатов атаки неопределенное время. Однако, ничто не мешает атакующему, не дожидаясь никого, самому послать на атакуемый DNS-сервер подобный DNS-запрос и спровоцировать DNS-сервер на поиск указанного в запросе имени. Тогда эта атака с большой вероятностью будет иметь успех практически сразу же после начала ее осуществления.

Для примера приведем скандал (28 октября 1996 г.) с одним из московских провайдеров Internet – компанией РОСНЕТ, когда пользователи этого провайдера при обращении к обычному информационному WWW-серверу попадали, как было сказано в репортаже СМИ, на WWW-сервер «сомнительного» содержания. В связи с абсолютным непониманием случившегося как журналистами (их можно понять – они не специалисты в этом вопросе), так и теми, кто проводил пресс-конференцию (специалистов к общению с прессой, наверное, просто не допустили) информационные сообщения об этом событии были настолько убоги, что понять, что случилось, было толком невозможно. Тем не менее, этот инцидент вполне укладывается в только что описанную схему удаленной атаки на DNS-сервер. С одним исключением: вместо адреса хоста, атакующего в кэш-таблицу DNS-сервера был занесен IP-адрес хоста www.playboy.com.

Использование в сети Internet службы удаленного поиска DNS позволяет атакующему организовать в Internet удаленную атаку на любой хост, пользующийся услугами этой службы, и может пробить серьезную брешь в безопасности этой и так отнюдь не безопасной глобальной сети.

16.4. Навязывание хосту ложного маршрута с использованием протокола ICMP с целью создания в сети ложного маршрутизатора

В сети Internet используется управляющий протокол ICMP, одной из функций которого является удаленное управление маршрутизацией на хостах внутри сегмента сети. Удаленное управление маршрутизацией необходимо для предотвращения возможной передачи сообщений по неоптимальному маршруту. В сети Internet удаленное управление маршрутизацией реализовано в виде передачи с маршрутизатора на хост управляющего ICMP-сообщения: Redirect Message. Исследование протокола ICMP показало, что сообщение Redirect бывает двух типов:

- первый тип сообщения носит название Redirect Net и уведомляет хост о необходимости смены адреса маршрутизатора, то есть default-маршрута;
- второй тип – Redirect Host – информирует хост о необходимости создания нового маршрута к указанной в сообщении системе и внесения ее в таблицу маршрутизации. Для этого в сообщении указывается IP-адрес хоста, для которого необходима смена маршрута (адрес будет занесен в поле Destination), и новый IP-адрес маршрутизатора, на который необходимо направлять пакеты, адресованные этому хосту (этот адрес заносится в поле Gateway).

Необходимо обратить внимание на важное ограничение, накладываемое на IP-адрес нового маршрутизатора: он должен быть в пределах адресов этой же подсети!

Что касается управляющего сообщения ICMP Redirect Host, то единственным идентифицирующим его параметром является IP-адрес отправителя, который должен совпадать с IP-адресом маршрутизатора, так как это сообщение может передаваться только маршрутизатором. Особенность протокола ICMP состоит в том, что он не предусматривает никакой дополнительной аутентификации источников сообщений. Таким образом, ICMP-сообщения передаются на хост маршрутизатором однонаправлено, без создания виртуального соединения.

Следовательно, ничто не мешает атакующему послать ложное ICMP-сообщение о смене маршрута от имени маршрутизатора. Приведенные выше факты позволяют осуществить типовую удаленную атаку «Внедрение в сеть ложного объекта путем навязывания ложного маршрута».

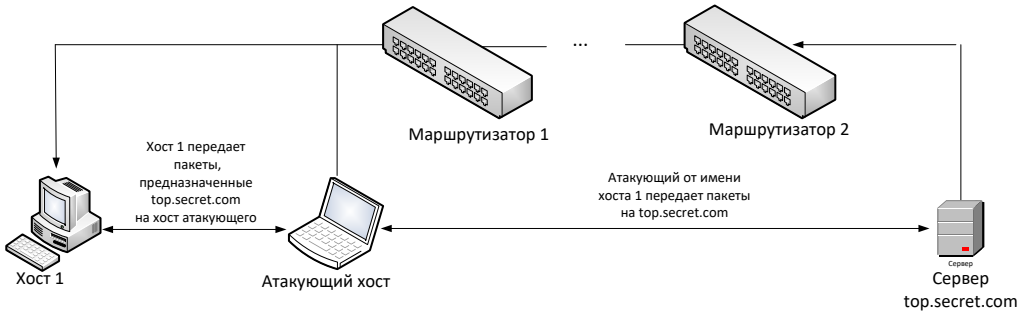
Для осуществления этой удаленной атаки необходимо подготовить ложное ICMP Redirect Host сообщение, в котором указать конечный IP-адрес маршрута (адрес хоста, маршрут к которому будет изменен) и IP-адрес ложного маршрутизатора. Далее это сообщение передается на атакуемый хост от имени маршрутизатора. Для этого в IP-заголовке в поле адреса отправителя указывается IP-адрес маршрутизатора. В принципе, можно предложить два варианта этой удаленной атаки.

В первом случае атакующий находится в том же сегменте сети, что и цель атаки. Тогда, послав ложное ICMP-сообщение, он в качестве IP-адреса нового маршрутизатора может указать либо свой IP-адрес, либо любой из адресов этой же подсети. Это даст атакующему возможность изменить маршрут передачи сообщений, направляемых атакованным хостом на определенный IP-адрес, и получить контроль над трафиком между атакуемым хостом и интересующим атакующего сервером. После этого атака перейдет во вторую стадию, связанную с приемом, анализом и передачей пакетов, получаемых от «обманутого» хоста.

Рассмотрим функциональную схему осуществления этой удаленной атаки (рис 16.8):

- передача на атакуемый хост ложного ICMP Redirect Host сообщения;
- отправление ARP-ответа в случае, если пришел ARP-запрос от атакуемого хоста;
- перенаправление пакетов от атакуемого хоста на настоящий маршрутизатор;
- перенаправление пакетов от маршрутизатора на атакуемый хост;
- при приеме пакета возможно воздействие на информацию по схеме «Ложный объект сети».

а) Фаза передачи ложного ICMP Redirect сообщения от имени маршрутизатора

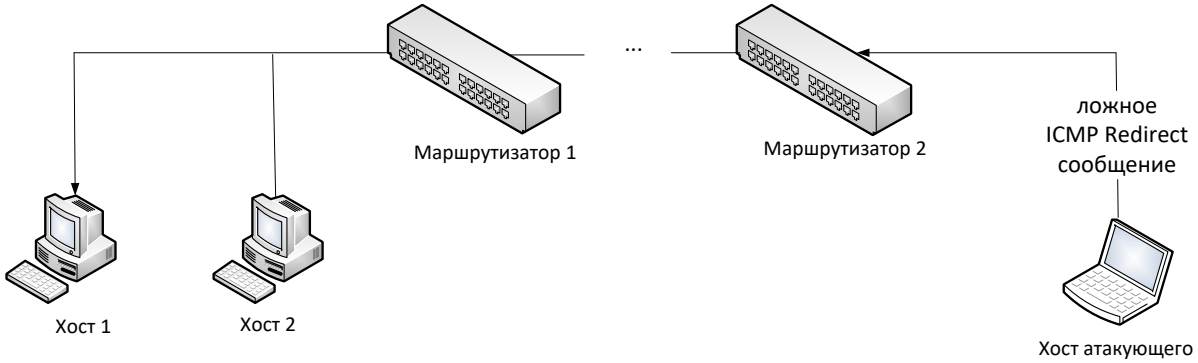


б) Фаза приема, анализа, воздействия и передачи перехваченной информации на ложном сервере

Рис. 16.8. Внутрисегментное навязывание хосту ложного маршрута при использовании протокола ICMP

В случае осуществления второго варианта удаленной атаки атакующий находится в другом сегменте относительно цели атаки. Тогда, в случае передачи на атакуемый хост ложного ICMP Redirect сообщения, сам атакующий уже не сможет получить контроль над трафиком, так как адрес нового маршрутизатора должен находиться в пределах подсети атакуемого хоста, поэтому использование такого варианта этой удаленной атаки не позволит атакующему получить доступ к передаваемой по каналу связи информации. Однако, в этом случае атака достигает другой цели: нарушается работоспособность хоста.

Атакующий с любого хоста в Internet может послать подобное сообщение на атакуемый хост. В случае, если сетевая ОС на этом хосте не проигнорирует это сообщение, то связь между таким хостом и сервером, указанным в ложном ICMP-сообщении, будет нарушена. Это произойдет из-за того, что все пакеты, направляемые хостом на этот сервер, будут отправлены на IP-адрес несуществующего маршрутизатора. Схема этой атаки приведена на рис. 16.9.



а) Передача атакующим на хост 1 ложного ICMP Redirect сообщения от имени маршрутизатора 1

б) Дезинформация хоста 1. Его таблица маршрутизации содержит информацию о ложном маршруте к хосту top.secret.com

Рис. 16.9. Межсегментное навязывание хосту ложного маршрута при использовании протокола ICMP, приводящее к отказу в обслуживании

Оба варианта рассмотренной удаленной атаки удается осуществить (как межсегментно, так и внутрисегментно) на ОС Linux, Windows. Остальные сетевые ОС (Linux 2.0 и защищенный по классу B1 UNIX), игнорировали такое ICMP Redirect сообщение (что, не правда ли, кажется вполне логичным с точки зрения обеспечения безопасности).

16.5. Подмена одного из субъектов TCP-соединения в сети

Протокол TCP (Transmission Control Protocol) является одним из базовых протоколов транспортного уровня сети Internet. Этот протокол поз-

воляет исправлять ошибки, которые могут возникнуть в процессе передачи пакетов, и является протоколом с установлением логического соединения – виртуального канала. По этому каналу передаются и принимаются пакеты с регистрацией их последовательности, осуществляется управление потоком пакетов, организовывается повторная передача искаженных пакетов, а в конце сеанса канал разрывается. При этом протокол TCP является единственным базовым протоколом из семейства TCP/IP, имеющим дополнительную систему идентификации сообщений и соединения. Именно поэтому протоколы прикладного уровня FTP и TELNET, предоставляющие пользователям удаленный доступ на хосты Internet, реализованы на базе протокола TCP.

Для идентификации TCP-пакета в TCP-заголовке существуют два 32-разрядных идентификатора, которые также играют роль счетчика пакетов. Их названия – *Sequence Number* и *Acknowledgment Number*. Также нас будет интересовать поле, называемое *Control Bits*.

Это поле размером 6 бит может содержать следующие командные биты (слева направо):

- *URG*: Urgent Pointer field significant;
- *ACK*: Acknowledgment field significant;
- *PSH*: Push Function;
- *RST*: Reset the connection;
- *SYN*: Synchronize sequence numbers;
- *FIN*: No more data from sender.

Далее рассмотрим схему создания TCP-соединения (рис 16.10).

Предположим, что хосту *A* необходимо создать TCP-соединение с хостом *B*. Тогда *A* посылает на *B* следующее сообщение:

1. $A \rightarrow B$: *SYN, ISSa*

Это означает, что в передаваемом *A* сообщении установлен бит *SYN* (synchronize sequence number), а в поле *Sequence Number* установлено начальное 32-битное значение *ISSa* (Initial Sequence Number).

2. *B* отвечает:

$B \rightarrow A$: *SYN, ACK, ISSb, ACK*(*ISSa*+1)

В ответ на полученный от *A* запрос *B* отвечает сообщением, в котором установлен бит *SYN* и установлен бит *ACK*; в поле Sequence Number хостом *B* устанавливается свое начальное значение счетчика – *ISSb*; поле Acknowledgment Number содержит значение *ISSa*, полученное в первом пакете от хоста *A* и увеличенное на единицу.

3. *A*, завершая рукопожатие (handshake), посылает:

$A \rightarrow B$: *ACK, ISSa*+1, *ACK*(*ISSb*+1)

В этом пакете установлен бит *ACK*; поле *Sequence Number* содержит *ISSa* + 1; поле *Acknowledgment Number* содержит значение *ISSb* + 1. Посылкой этого пакета на хост *B* заканчивается трехступенчатый *handshake*, и TCP-соединение между хостами *A* и *B* считается установленным.

4. Теперь хост *A* может посылать пакеты с данными на хост *B* по только что созданному виртуальному TCP-каналу:

$$A \rightarrow B: ACK, ISSa+1, ACK(ISSb+1); DATA$$

Из рассмотренной выше схемы создания TCP-соединения видно, что единственными идентификаторами TCP-абонентов и TCP-соединения являются два 32-битных параметра Sequence Number и Acknowledgment Number. Следовательно, для формирования ложного TCP-пакета атакующему необходимо знать текущие идентификаторы соединения – *ISSa* и *ISSb*.

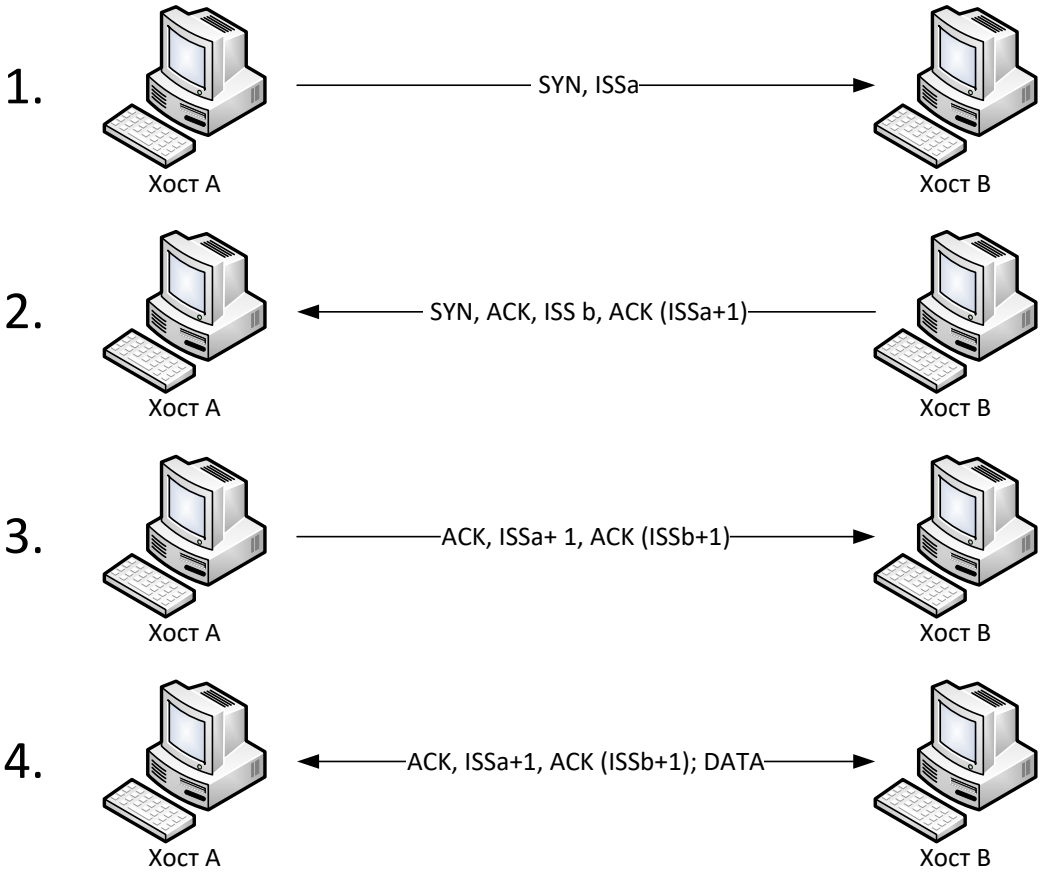


Рис. 16.10. Схема создания TCP-соединения

Проблема возможной подмены TCP-сообщения становится еще более важной, так как анализ протоколов FTP и TELNET, реализованных на базе протокола TCP, показал, что проблема идентификации FTP- и TELNET-пакетов целиком возлагается этими протоколами на транспортный уровень, то есть на TCP. Это означает, что атакующему достаточно, подобрав соответствующие текущие значения идентификаторов TCP-пакета для конкретного TCP-соединения (например, такое соединение может представлять собой FTP- или TELNET-подключение), послать пакет с любого хоста в сети Internet от имени одного из участников соединения (например, от имени клиента), и такой пакет будет воспринят как верный! К тому же, так как FTP и TELNET не проверяют IP-адреса отправителей, от которых им приходят сообщения, то в ответ на полученный ложный пакет, FTP- или TELNET-сервер отправит ответ на указанный в ложном пакете настоящий IP-адрес атакующего, то есть атакующий начнет работу с

FTP- или TELNET-сервером со своего IP-адреса, но с правами легально подключившегося пользователя, который, в свою очередь, потеряет связь с сервером из-за рассогласования счетчиков.

16.6. Нарушение работоспособности хоста в сети путем использовании направленного «шторма» ложных TCP-запросов на создание соединения, либо при переполнении очереди запросов

Из рассмотренной в предыдущем пункте схемы создания TCP-соединения следует, что на каждый полученный TCP-запрос на создание соединения операционная система должна сгенерировать начальное значение идентификатора ISN и отослать его в ответ на запросивший хост. При этом, так как в сети Internet (стандарта IPv4) не предусмотрен контроль за IP-адресом отправителя сообщения, то невозможно отследить истинный маршрут, пройденный IP-пакетом, и, следовательно, у конечных абонентов сети нет возможности ограничить число возможных запросов, принимаемых в единицу времени от одного хоста. Поэтому возможно осуществление типовой удаленной атаки «Отказ в обслуживании», которая будет заключаться в передаче на атакуемый хост как можно большего числа ложных TCP-запросов на создание соединения от имени любого хоста в сети (рис. 16.11).

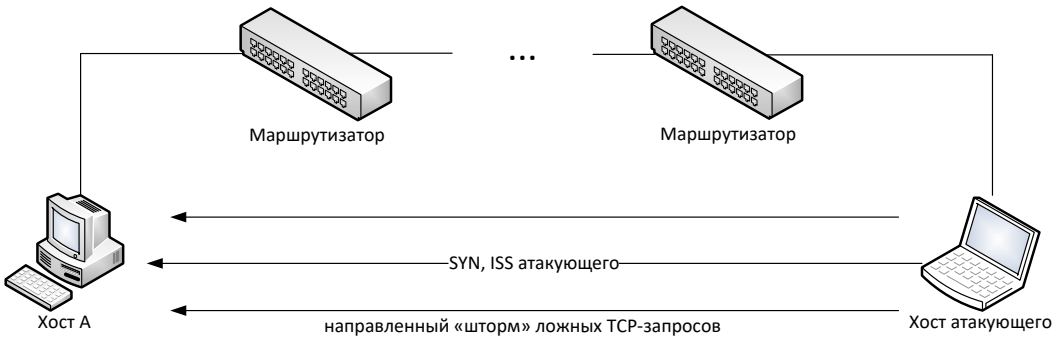


Рис. 16.11. Нарушение работоспособности хоста в Internet, использующее направленный шторм ложных TCP-запросов на создание соединения

При этом атакуемая сетевая ОС в зависимости от вычислительной мощности компьютера либо – в худшем случае – практически зависает, либо – в лучшем случае – перестает реагировать на легальные запросы на подключение (отказ в обслуживании). Это происходит из-за того, что для всей массы полученных ложных запросов система должна, во-первых, сохранить в памяти полученную в каждом запросе информацию и, во-вторых, выработать и отослать ответ на каждый запрос. Таким образом, все ресурсы системы «съедаются» ложными запросами: переполняется очередь запросов и система занимается только их обработкой. Эффективность такой удаленной атаки тем выше, чем больше пропускная способ-

ность канала между атакующим и целью атаки, и тем меньше, чем больше вычислительная мощь атакуемого компьютера (число и быстродействие процессоров, объем ОЗУ и т. д.).

Другая разновидность атаки «Отказ в обслуживании» состоит в передаче на атакуемый хост нескольких десятков (сотен) запросов на подключение к серверу, что может привести к временному (до 10 минут) переполнению очереди запросов на сервере. Это происходит из-за того, что некоторые сетевые ОС устроены так, чтобы обрабатывать только первые несколько запросов на подключение, а остальные – игнорировать. То есть при получении N запросов на подключение, ОС сервера ставит их в очередь и генерирует соответственно N ответов. Далее, в течение определенного промежутка времени, сервер будет дожидаться от предполагаемого клиента сообщения, завершающего handshake и подтверждающего создание виртуального канала с сервером. Если атакующий пришлет на сервер количество запросов на подключение, равное максимальному числу одновременно обрабатываемых запросов на сервере, то в течение тайм-аута остальные запросы на подключение будут игнорироваться и к серверу будет невозможно подключиться.

Необходимо отметить, что в существующем стандарте сети Internet IPv4 нет приемлемых способов надежно обезопасить свои системы от этой удаленной атаки. К счастью, атакующий в результате осуществления описанной атаки не сможет получить несанкционированный доступ к вашей информации. Он сможет лишь «съесть» вычислительные ресурсы вашей системы и нарушить ее связь с внешним миром. Остается надеяться, что нарушение работоспособности вашего хоста просто никому не нужно.

17. Обеспечение безопасности систем, входящих в состав телекоммуникационных сетей

Рассмотренные ранее угрозы нарушения ИБ в ТКС, способствуют активному развитию средств защиты, ориентированных на анализ трафика и сетевой активности пользователей. К основным таким средствам относятся:

- межсетевые экраны;
- виртуальные частные сети;
- системы предотвращения вторжений.

В данном подразделе, вышеуказанные средства защиты ТКС будут рассмотрены более подробно.

17.1. Межсетевые экраны (Firewall)

Межсетевой экран (*firewall*) – это устройство контроля доступа в сеть, предназначенное для блокировки всего трафика, за исключением разрешенных данных. Прохождение трафика на межсетевом экране можно настраивать по службам, IP-адресам отправителя и получателя, по идентификаторам пользователей, запрашивающих службу. Межсетевые экраны позволяют осуществлять централизованное управление безопасностью. В одной конфигурации администратор может настроить разрешенный входящий трафик для всех внутренних систем организации. Этим оно отличается от маршрутизатора, функцией которого является доставка трафика в пункт назначения в максимально короткие сроки.

Межсетевые экраны, представляют собой программные пакеты, базирующиеся на операционных системах общего назначения (таких как Windows и Unix) или на специализированной аппаратно-программной платформе. Набор правил политики определяет, каким образом трафик передается из одной сети в другую. Если в правиле отсутствует явное разрешение на пропуск трафика, межсетевой экран отклоняет или аннулирует пакеты. Правила политики безопасности усиливаются посредством использования модулей доступа.

Типы межсетевых экранов:

- межсетевые экраны прикладного уровня;
- межсетевые экраны с пакетной фильтрацией;
- гибридные межсетевые экраны.

17.1.1. Межсетевые экраны прикладного уровня

В межсетевом экране прикладного уровня каждому разрешаемому протоколу должен соответствовать свой собственный модуль доступа. Лучшими модулями доступа считаются те, которые построены специально для разрешаемого протокола.

Например, модуль доступа FTP предназначен для протокола FTP и может определять, соответствует ли проходящий трафик этому протоколу и разрешен ли этот трафик правилами политики безопасности. При использовании межсетевого экрана прикладного уровня все соединения проходят через него (см. рис. 17.1).

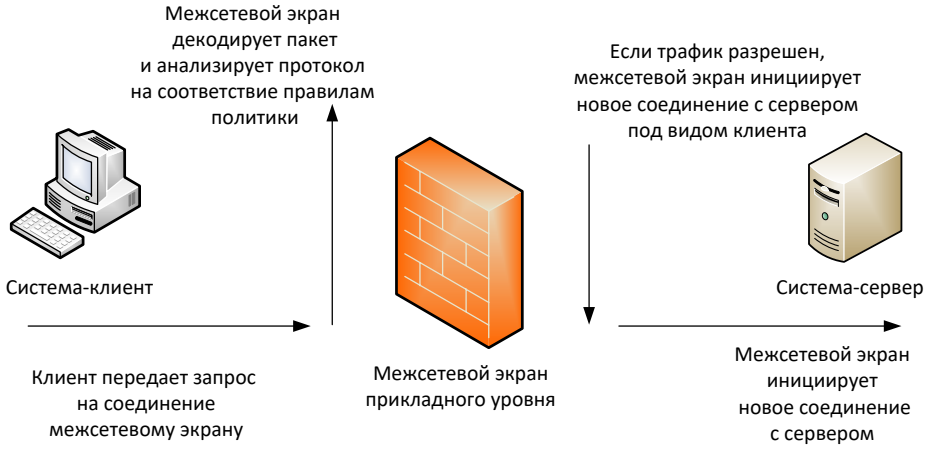


Рис. 17.1. Соединения модуля доступа межсетевого экрана прикладного уровня

Как показано на рисунке, соединение начинается на системе-клиенте и поступает на внутренний интерфейс межсетевого экрана. Межсетевой экран принимает соединение, анализирует содержимое пакета и используемый протокол и определяет, соответствует ли этот трафик правилам политики безопасности. Если это так, то межсетевой экран инициирует новое соединение между своим внешним интерфейсом и системой-сервером.

Межсетевые экраны прикладного уровня используют модули доступа для входящих подключений. Модуль доступа в межсетевом экране принимает входящее подключение и обрабатывает команды перед отправкой трафика получателю. Таким образом, межсетевой экран защищает системы от атак, выполняемых посредством приложений.

17.1.2. Межсетевые экраны с пакетной фильтрацией

Правила политики в межсетевых экранах с пакетной фильтрацией устанавливаются посредством использования *фильтров пакетов*. Фильтры изучают пакеты и определяют, является ли трафик разрешенным, согласно правилам политики и состоянию протокола (проверка с учетом состояния). Если протокол приложения функционирует через TCP, определить состояние относительно просто, так как TCP сам по себе поддерживает состояния. Это означает, что когда протокол находится в определенном состоянии, разрешена передача только определенных пакетов.

При использовании межсетевого экрана с пакетной фильтрацией соединения не прерываются на межсетевом экране (см. рис. 17.2), а направляются непосредственно к конечной системе. При поступлении пакетов

межсетевой экран выясняет, разрешена ли передача этого пакета и соединение в соответствии с правилами политики безопасности. Если это так, то пакет передается по своему маршруту. В противном случае пакет отклоняется или аннулируется.

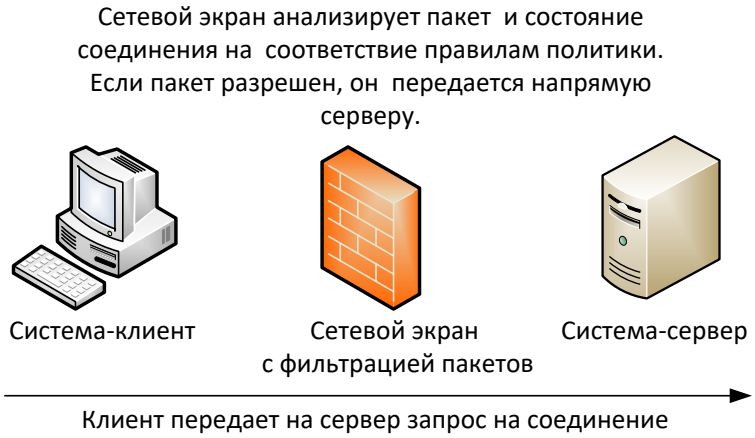


Рис. 17.2. Передача трафика через межсетевой экран с фильтрацией пакетов

Межсетевые экраны с фильтрацией пакетов не используют модули доступа для каждого протокола и поэтому могут использоваться с любым протоколом, работающим через IP. Некоторые протоколы требуют распознавания межсетевым экраном выполняемых ими действий. Например, FTP будет использовать одно соединение для начального входа и команд, а другое – для передачи файлов. Соединения, используемые для передачи файлов, устанавливаются как часть соединения FTP, и поэтому межсетевой экран должен уметь считывать трафик и определять порты, которые будут использоваться новым соединением. Если межсетевой экран не поддерживает эту функцию, передача файлов невозможна.

17.1.3. Гибридные межсетевые экраны

Как и многие другие устройства, межсетевые экраны изменяются и совершенствуются с течением времени, т. е. эволюционируют. Так технология модуля доступа Generic Services Proxy (GSP) разработана для поддержки модулями доступа прикладного уровня других протоколов, необходимых системе безопасности и при работе сетевых администраторов. В действительности GSP обеспечивает работу межсетевых экранов прикладного уровня в качестве экранов с пакетной фильтрацией.

Производители межсетевых экранов с пакетной фильтрацией также добавили некоторые модули доступа в свои продукты для обеспечения более высокого уровня безопасности некоторых широко распространенных протоколов. В то время как базовая функциональность межсетевых экранов обоих типов осталась прежней, (что является причиной большинства «слабых мест» этих устройств), сегодня на рынке присутствуют гибридные межсетевые экраны. Практически невозможно найти межсетевой экран,

функционирование которого построено исключительно на прикладном уровне или фильтрации пакетов. Это позволяет администраторам, отвечающим за безопасность, настраивать устройство для работы в конкретных условиях.

17.1.4. Пример конфигурирования межсетевого экрана

Стандартная архитектура использования межсетевого экрана показана на рис. 17.3. В представленной архитектуре используется только один межсетевой экран для защиты как внутренней сети, так и любых других систем, доступных из интернета. Эти системы располагаются в отдельной сети.

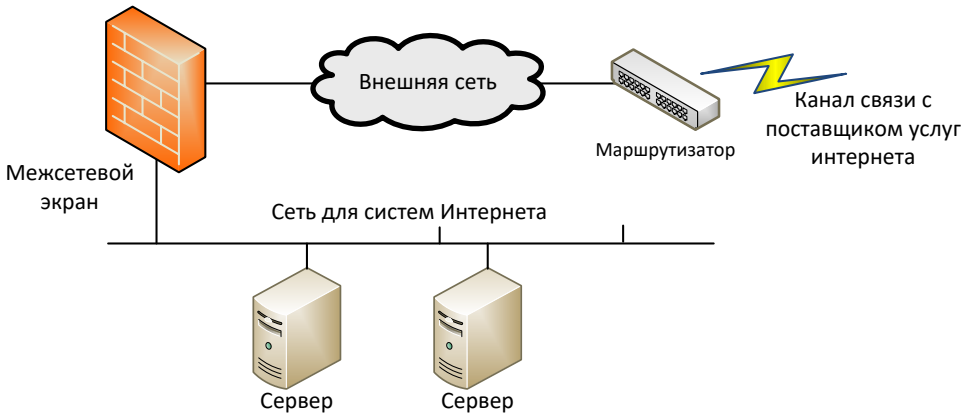


Рис. 17.3. Использование межсетевого экрана

Таблица 17.1 – Правила межсетевого экрана для сети на рис. 17.3

Номер	Исходный IP	Конечный IP	Служба	Действие
1	Любой	Веб-сервер	HTTP	Принятие
2	Любой	Почтовый сервер	SMTP	Принятие
3	Почтовый сервер	Любой	SMTP	Принятие
4	Внутренняя сеть	Любой	HTTP, HTTPS, FTP, Telnet	Принятие
5	Внутренняя DNS	Любой	DNS	Принятие
6	Любой	Любой	Любая	Сброс

Таблица 17.2 – Краткий список номеров портов (для протокола TCP)

Протокол : номер порта	Протокол : номер порта
HTTP: 80, 8080	FTP: 21 для команд, 20 для данных
SSH: 22	TFTP: 69/UDP
POP3: 110	ICQ: 5190
SMTP: 25	telnet: 23
IMAP: 143	DNS: 53 (обычно UDP)

17.2. Организация и эксплуатация виртуальных частных сетей (VPN)

Частные сети состоят из каналов связи, арендуемых у различных телефонных компаний и поставщиков услуг интернета. Эти каналы связи характеризуются тем, что они соединяют только два объекта, будучи отделенными от другого трафика, так как арендуемые каналы обеспечивают двустороннюю связь между двумя сайтами.

Частные сети обладают множеством преимуществ:

- информация сохраняется в секрете;
- удаленные сайты могут осуществлять обмен информацией незамедлительно;
- удаленные пользователи не ощущают себя изолированными от системы, к которой они осуществляют доступ.

К сожалению, этот тип сетей обладает одним большим недостатком – высокой стоимостью. С увеличением числа пользователей интернета многие организации перешли на использование виртуальных частных сетей (Virtual Private Network – VPN). Виртуальные частные сети обеспечивают многие преимущества частных сетей за меньшую цену.

17.2.1. Определение виртуальных частных сетей

Значительная часть Internet трафика передается в открытом виде, и любой пользователь, наблюдающий за этим трафиком, сможет его распознать. Это относится к большей части почтового и веб-трафика, а также сеансам связи через протоколы telnet и FTP. Трафик Secure Shell (SSH) и Hyper text Transfer Protocol Secure (HTTPS) является шифруемым трафиком, и его не сможет просмотреть пользователь, отслеживающий пакеты. Тем не менее, трафик типа SSH и HTTPS не образует виртуальную частную сеть VPN.

Виртуальные частные сети обладают следующими характеристиками:

- трафик шифруется для обеспечения защиты от прослушивания;
- осуществляется аутентификация удаленного сайта;
- виртуальные частные сети обеспечивают поддержку множества протоколов;
- соединение обеспечивает связь только между двумя конкретными абонентами.

Так как SSH и HTTPS не способны поддерживать несколько протоколов, то же самое относится и к реальным виртуальным частным сетям. VPN-пакеты смешиваются с потоком обычного трафика в интернете и существуют отдельно по той причине, что такой трафик может считываться только конечными точками соединения.

VPN соединяет два конкретных объекта, образуя таким образом уникальный канал связи между двумя абонентами. Каждая из конечных точек VPN может единовременно поддерживать несколько соединений VPN с

другими конечными точками, однако каждая из точек является отдельной от других, и трафик разделяется посредством шифрования.

Виртуальные частные сети, по методу использования, подразделяются на два типа:

- пользовательские VPN;
- узловые VPN.

17.2.2. Пользовательские VPN

Пользовательские VPN представляют собой виртуальные частные сети, построенные между отдельной пользовательской системой и узлом или сетью организации (часто пользовательские *VPN* используются сотрудниками, находящимися в командировке или работающими из дома).

Сервер *VPN* может являться межсетевым экраном организации либо быть отдельным VPN-сервером. Пользователь подключается к интернету через телефонное подключение к локальному поставщику услуг, через канал DSL или кабельный модем и инициирует VPN-соединение с узлом организации через интернет. Узел организации запрашивает у пользователя аутентификационные данные и, в случае успешной аутентификации, позволяет пользователю осуществить доступ ко внутренней сети организации, как если бы пользователь находился внутри узла и физически располагался внутри сети.

Пользовательские VPN позволяют организациям ограничивать доступ удаленных пользователей к системам или файлам. Это ограничение должно базироваться на политике организации и зависит от возможностей продукта VPN.

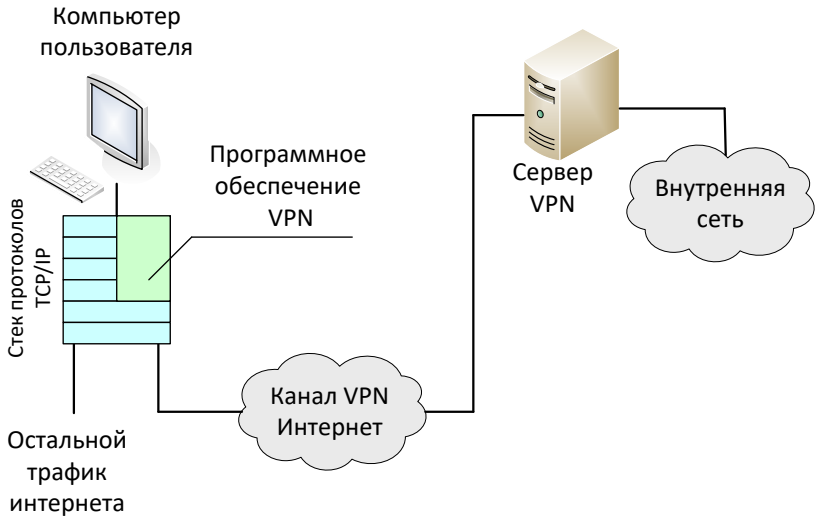


Рис. 17.4. Конфигурация пользовательской VPN

Пользовательские VPN обладают следующими двумя основными преимуществами.

1. Сотрудники, находящиеся в командировке, могут осуществлять доступ к электронной почте, файлам и внутренним системам в

любое время без необходимости в осуществлении дорогостоящих междугородних и международных телефонных вызовов для соединения с серверами.

2. Сотрудники, работающие из дома, могут осуществлять доступ к службам сети, как и сотрудники, работающие в организации, без аренды дорогостоящих выделенных каналов.

Оба эти преимущества можно приписать к экономии денежных средств. Экономия может заключаться в отказе от использования дорогостоящих междугородних и международных соединений, арендуемых каналов связи и т.д.

Самой большой проблемой безопасности при использовании VPN сотрудником является одновременное соединение с другими сайтами интернета. Как правило, программное обеспечение VPN на компьютере пользователя определяет, должен ли трафик передаваться через VPN, либо его необходимо отправить на какой-либо другой сайт в открытом виде. Если на компьютер пользователя была произведена атака с использованием «троянского коня», возможно, что некий внешний нелегальный пользователь использует компьютер сотрудника для подключения к внутренней сети организации. Атаки такого типа осуществляются довольно сложно, но они совершенно реальны.

13.2.3. Узловые VPN

Узловые виртуальные частные сети используются организациями для подключения к удаленным узлам без применения дорогостоящих выделенных каналов или для соединения двух различных организаций, между которыми необходима связь для осуществления информационного обмена, связанного с деятельностью этих организаций. Как правило, VPN соединяет один межсетевой экран или пограничный маршрутизатор с другим аналогичным устройством (см. рис. 17.5).

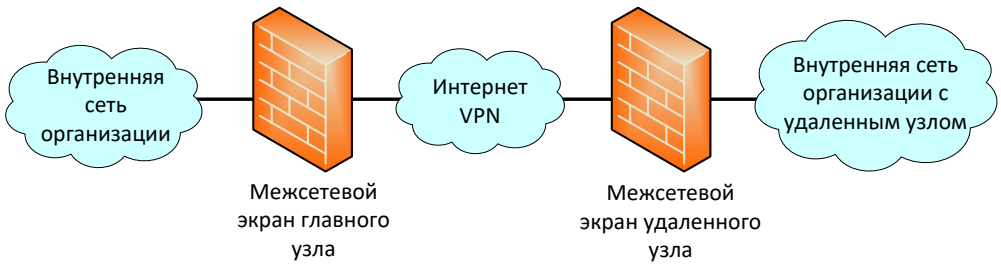


Рис. 17.5. Межузловое соединение VPN, проходящее через Интернет

Чтобы инициировать соединение, один из узлов осуществляет попытку передать трафик другому узлу. Вследствие этого на обоих противоположных узлах соединения VPN инициируется VPN. Оба конечных узла определяют параметры соединения в зависимости от политик, имеющихся на узлах. Оба сайта будут аутентифицировать друг друга посредством не-

которого общего предопределенного секрета либо с помощью сертификата с открытым ключом. Некоторые организации используют узловые VPN в качестве резервных каналов связи для арендуемых каналов.

Основным преимуществом узловой VPN является экономичность. Организация с небольшими, удаленными друг от друга офисами может создать виртуальную частную сеть, соединяющую все удаленные офисы с центральным узлом (или даже друг с другом) со значительно меньшими затратами. Сетевая инфраструктура также может быть применена значительно быстрее, так как в удаленных офисах могут использоваться локальные ISP для каналов ISDN или DSL.

Проблемы, связанные с узловыми VPN. Узловые VPN расширяют периметр безопасности организации, добавляя новые удаленные узлы или даже удаленные организации. Если уровень безопасности удаленного узла невелик, VPN может позволить злоумышленнику получить доступ к центральному узлу и другим частям внутренней сети организации. Следовательно, необходимо применять строгие политики и реализовывать функции аудита для обеспечения безопасности организации в целом. В случаях, когда две организации используют узловую VPN для соединения своих сетей, очень важную роль играют политики безопасности, установленные по обе стороны соединения. В данной ситуации обе организации должны определить, какие данные могут передаваться через VPN, а какие – нет, и соответствующим образом настроить политики на своих межсетевых экранах.

13.2.4 Понятие стандартных технологий функционирования VPN

Сеть VPN состоит из четырех ключевых компонентов.

1. *Сервер VPN.* Сервер VPN представляет собой компьютер, выступающий в роли конечного узла соединения VPN. Сервер должен обладать характеристиками, достаточными для поддержки ожидаемой нагрузки. Большая часть производителей программного обеспечения VPN должна предоставлять рекомендации по поводу производительности процессора и конфигурации памяти, в зависимости от числа единовременных VPN-соединений.

2. *Алгоритмы шифрования.* Выбор алгоритма не имеет принципиального значения, если он будет стандартным и в достаточной степени мощным. Гораздо больше влияет на общий уровень безопасности реализация системы. Неправильно реализованная система может сделать бесполезным самый мощный алгоритм шифрования. Приняв во внимание, сказанное выше, давайте изучим риски, связанные с использованием VPN.

3. *Система аутентификации.* Система аутентификации VPN должна быть двухфакторной. Пользователи могут проходить аутентификацию с использованием того, что они знают, того, что у них есть или с помощью данных о том, кем они являются. Хорошей комбинацией средств аутенти-

фикации являются смарт-карты в паре с персональным идентификационным номером или паролем.

4. *Протокол VPN.* Протокол VPN определяет, каким образом система VPN взаимодействует с другими системами в интернете, а также уровень защищенности трафика. Протокол VPN оказывает влияние на общий уровень безопасности системы. Причиной этому является тот факт, что протокол VPN используется для обмена ключами шифрования между двумя конечными узлами. Если этот обмен не защищен, злоумышленник может перехватить ключи и затем расшифровать трафик, сведя на нет все преимущества VPN. В настоящее время стандартным протоколом для VPN является IPSec. Этот протокол представляет собой дополнение к IP, осуществляющее инкапсуляцию и шифрование заголовка TCP и полезной информации, содержащейся в пакете. IPSec также поддерживает обмен ключами, удаленную аутентификацию сайтов и согласование алгоритмов (как алгоритма шифрования, так и хэш-функции). IPSec использует UDP-порт: 500 для начального согласования, после чего используется IP-протокол 50 для всего трафика. Для правильного функционирования VPN эти протоколы должны быть разрешены.

Эти компоненты реализуют соответствие требованиям по безопасности, производительности и способности к взаимодействию. То, насколько правильно реализована архитектура VPN, зависит от правильности определения требований.

Определение требований по безопасности в VPN должно включать в себя следующие аспекты:

- количество времени, в течение которого необходимо обеспечивать защиту информации;
- число одновременных соединений пользователей;
- ожидаемые типы соединений пользователей (сотрудники, работающие из дома или находящиеся в поездке);
- число соединений с удаленным сервером;
- типы сетей VPN, которым понадобится соединение;
- ожидаемый объем входящего и исходящего трафика на удаленных узлах;
- политика безопасности, определяющая настройки безопасности.

При разработке системы также может оказаться полезным указать дополнительные требования, связанные с местоположением сотрудников, находящихся в поездке (имеются в виду узлы в других организациях или в номерах отелей), а также типы служб, которые будут работать через VPN.

17.2.5. Типы систем VPN

На настоящее время можно выделить три типа систем, на основе которых организуются VPN:

- аппаратные системы;
- программные системы;

- веб-системы.

Аппаратные системы VPN, как правило, базируются на аппаратной платформе, используемой в качестве VPN-сервера. На этой платформе выполняется программное обеспечение производителя, а также, возможно, некоторое специальное программное обеспечение, предназначенное для улучшения возможностей шифрования. В большинстве случаев для построения VPN на системе удаленного пользователя необходимо наличие соответствующего программного обеспечения.

Аппаратная система VPN имеет два преимущества.

- Скорость. Оборудование, как правило, оптимизировано для поддержки VPN, посредством чего обеспечивается преимущество в скорости по сравнению с компьютерными системами общего назначения. За счет этого достигается возможность поддержки большего числа одновременных VPN-соединений.
- Безопасность. Если аппаратная платформа специально разработана для приложения VPN, из ее системы удалены все лишние программы и процессы. За счет этого снижается степень подверженности атакам по сравнению с компьютерной системой общего назначения, в которой работают другие процессы. Это не значит, что компьютер общего назначения не может быть должным образом защищен. Как правило, использование компьютера общего назначения требует дополнительных усилий по настройке безопасности.

Программные VPN работают на компьютерных системах общего назначения. Они могут быть установлены на выделенной для VPN системе либо совместно с другим программным обеспечением, таким как межсетевой экран. При загрузке программного обеспечения необходимо обеспечить достаточную мощность аппаратной платформы для поддержки VPN. Так как VPN-продукт устанавливается на компьютеры, имеющиеся в организации, руководство организации должно позаботиться о соответствии компьютеров предъявляемым требованиям.

Веб-системы. Главным недостатком большинства пользовательских систем VPN является потребность в установке программного обеспечения на систему-клиент. Указанные проблемы привели к тому, что некоторые производители VPN стали рассматривать веб-браузеры в качестве VPN-клиентов и реализовывать этот подход на практике. Он заключается в том, что пользователь с помощью браузера подключается к VPN через SSL. SSL обеспечивает шифрование трафика, а подтверждение подлинности пользователя выполняется с помощью средств аутентификации, встроенных в систему. Для предоставления пользователю необходимых услуг используется несколько различных механизмов. Среди них можно выделить надстройки браузера и виртуальные машины Java.

В то время как стоимость поддержки и обслуживания несомненно ниже, на момент написания этой книги ни одна из бесклиентных систем VPN не обеспечивает полную функциональность. Этим сетям VPN прису-

щи ограничения, заключающиеся в наборе используемых приложений и методе подключения пользователей к внутренним системам.

17.3. Системы предотвращения вторжений (IDS)

17.3.1. Общие понятия о функционировании IDS

Обнаружение вторжений – задача, выполняемая сотрудниками, ответственными за безопасность информации в организации, при обеспечении защиты от атак, это активный процесс, при котором происходит обнаружение хакера при его попытках проникнуть в систему.

Система обнаружения вторжений (Intrusion detection system – IDS) обнаруживает несанкционированные попытки проникновения в зашищаемый периметр. Обнаружение вторжений помогает при превентивной идентификации активных угроз посредством оповещений и предупреждений о том, что злоумышленник осуществляет сбор информации, необходимой для проведения атаки.

Базовая концепция системы обнаружения вторжений заключается в необходимости определения периметра защиты компьютерной системы или сети.

Периметр защиты сети представляет собой виртуальный периметр, внутри которого находятся компьютерные системы. Этот периметр может определяться межсетевыми экранами, точками разделения соединений или настольными компьютерами с модемами. Периметр может быть расширен для содержания домашних компьютеров сотрудников, которым разрешено соединяться друг с другом, или партнеров по бизнесу, которым разрешено подключаться к сети. С появлением в деловом взаимодействии беспроводных сетей периметр защиты организации расширяется до размера беспроводной сети. В случае если компания имеет части информационных ресурсов доступных напрямую из глобальной сети периметр защиты дополняется демилитаризированной зоной (DMZ – Demilitarized Zone). Суть DMZ заключается в том, что она не входит непосредственно ни во внутреннюю, ни во внешнюю сеть, и доступ к ней может осуществляться только по заранее заданным правилам межсетевого экрана. В DMZ нет пользователей – там располагаются только серверы.

Демилитаризованная зона (Demilitarized Zone – DMZ) служит для предотвращения доступа из внешней сети к ресурсам и компьютерам внутренней сети за счет выноса из локальной сети в особую зону всех сервисов, требующих доступа извне.

Сигнализация, оповещающая о проникновении злоумышленника, предназначена для обнаружения любых попыток входа в защищаемую область т. е. система обнаружения вторжений IDS предназначена для разграничения авторизованного входа и несанкционированного проникновения.

Цели использования IDS определяют требования для политики IDS. Потенциально целями применения IDS являются следующие.

- *Обнаружение атак.* Распознавание атак является одной из главных целей использования IDS. Система IDS запрограммирована на поиск определенных типов событий, которые служат признаками атак. В качестве простого примера приведем соединение через TCP-порт: 80 (HTTP), за которым следует URL, содержащий расширение .bat. Это может быть признаком того, что злоумышленник пытается использовать уязвимость на веб-сервере IIS.
- *Предотвращение атак.* При обнаружении атаки IDS должна выполнить действия по нейтрализации угрозы.
- *Обнаружение нарушений политики.* Целью системы IDS, настроенной на отслеживание политики, является отслеживание выполнения или невыполнения политики организации. В самом простом случае NIDS можно настроить на отслеживание всего вебтрафика вне сети. Такая конфигурация позволяет отслеживать любое несоответствие политикам использования Интернета.
- *Принуждение к использованию политик безопасности.* Применение системы IDS в качестве средства принудительного использования политики выводит конфигурацию мониторинга политики на более высокий уровень. При отслеживании политики IDS настраивается на выполнение действий при нарушении политики.
- *Принуждение к следованию политикам соединений.* Использования принудительного блокирования не запрошенных или запрещенных соединений.
- *Сбор доказательств.* Система IDS может оказаться полезной после обнаружения инцидента. В этом случае с помощью IDS можно собрать доказательства. Сетевую IDS можно настроить на отслеживание определенных соединений и ведение полноценного журнала по учету трафика.

Существуют два основных типа IDS:

1) *узловые* (Host IDS HIDS) – располагается на отдельном узле и отслеживает признаки атак на этот узел;
2) *сетевые* (Network IDS – NIDS) – находится на отдельной системе, отслеживающей сетевой трафик на наличие признаков атак, проводимых в подконтрольном сегменте сети.

На рис. 17.6 показаны два типа IDS, которые могут присутствовать в сетевой среде.

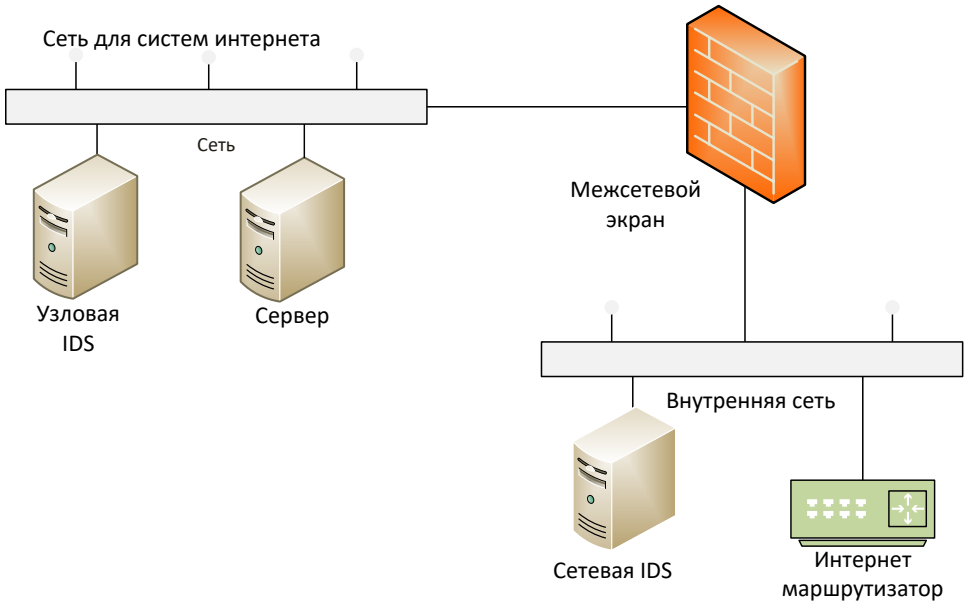


Рис. 17.6. Примеры размещения IDS в сетевой среде

17.3.2. Узловые IDS

Узловые IDS (Host Intrusion Detection System – HIDS) представляют собой систему датчиков, загружаемых на различные сервера организации и управляемых центральным диспетчером. Датчики отслеживают различные типы событий (более детальное рассмотрение этих событий приводится в следующем разделе) и предпринимают определенные действия на сервере либо передают уведомления. Датчики HIDS отслеживают события, связанные с сервером, на котором они загружены. Сенсор HIDS позволяет определить, была ли атака успешной, если атака имела место на той же платформе, на которой установлен датчик.

Существует пять основных типов датчиков HIDS.

1. *Анализаторы журналов.* Большая часть анализаторов журналов настроена на отслеживание записей журналов, которые могут означать событие, связанное с безопасностью системы. Анализаторы журналов по своей природе являются реактивными системами. Иными словами, они реагируют на событие уже после того, как оно произошло. Таким образом, журнал будет содержать сведения о том, что проникновение в систему выполнено. В большинстве случаев анализаторы журналов не способны предотвратить осуществляемую атаку на систему.

2. *Датчики признаков.* Системы, основанные на сопоставлении признаков, обеспечивают возможность отслеживания атак во время их выполнения в системе, поэтому они могут выдавать дополнительные уведомления о проведении злоумышленных действий. Тем не менее, атака будет успешно или безуспешно завершена перед вступлением в действие датчика HIDS, поэтому датчики этого типа считаются реактивными. Датчик признаков HIDS является полезным при отслеживании авторизованных пользователей внутри информационных систем.

3. *Анализаторы системных вызовов.* Анализаторы системных вызовов осуществляют анализ вызовов между приложениями и операционной системой для идентификации событий, связанных с безопасностью. Датчики HIDS этого типа размещают в программном слое между операционной системой и приложениями. Когда приложению требуется выполнить действие, его вызов к операционной системе анализируется и сопоставляются с базой данных признаков. Эти признаки являются примерами различных типов поведения, которые являют собой атакующие действия, или объектом интереса для администратора IDS. Анализаторы системных вызовов отличаются от анализаторов журналов и датчиков признаков HIDS тем, что они могут предотвращать действия. Если приложение генерирует вызов, соответствующий, например, признаку атаки на переполнение буфера, датчик позволяет предотвратить этот вызов и сохранить систему в безопасности.

4. *Анализаторы поведения приложений.* Анализаторы поведения приложений аналогичны анализаторам системных вызовов в том, что они применяются в виде программной спайки между приложениями и операционной системой. В анализаторах поведения датчик проверяет вызов на предмет того, разрешено ли приложению выполнять такое действие, вместо определения соответствие вызова признакам атак. Например, веб-серверу обычно разрешается принимать сетевые соединения через порт 80, считывать файлы в веб-каталоге и передавать эти файлы по соединениям через порт 80. Если веб-сервер попытается записать или считать файлы из другого места или открыть новые сетевые соединения, датчик обнаружит несоответствующее норме поведение сервера и заблокирует действие.

5. *Контролеры целостности файлов.* Контролеры целостности файлов отслеживают изменения в файлах. Это осуществляется посредством использования криптографической контрольной суммы или цифровой подписи файла. Конечная цифровая подпись файла будет изменена, если произойдет изменение хотя бы малой части исходного файла (это могут быть атрибуты файла, такие как время и дата создания). Алгоритмы, используемые для выполнения этого процесса, разрабатывались с целью максимального снижения возможности для внесения изменений в файл с сохранением прежней подписи.

17.3.3. Сетевые IDS

Сетевые IDS (Network Intrusion Detection System – NIDS) представляет собой программный процесс, работающий на специально выделенной системе. NIDS переключает сетевую карту в режим работы, при котором сетевой адаптер пропускает весь сетевой трафик (а не только трафик, направленный на данную систему) в программное обеспечение NIDS. После этого происходит анализ трафика с использованием набора правил и признаков атак для определения того, представляет ли этот трафик какой-либо интерес. Если это так, то генерируется соответствующее событие.

На данный момент большинство систем NIDS базируется на признаках атак. Это означает, что в системы встроен набор признаков атак, с которыми сопоставляется трафик в канале связи. Если происходит атака, признак которой отсутствует в системе обнаружения вторжений, система NIDS не реагирует на такую атаку.

NIDS-системы позволяют указывать интересуемый трафик по адресу источника, конечному адресу, порту источника или конечному порту. Это дает возможность отслеживания трафика, не соответствующего признакам атак.

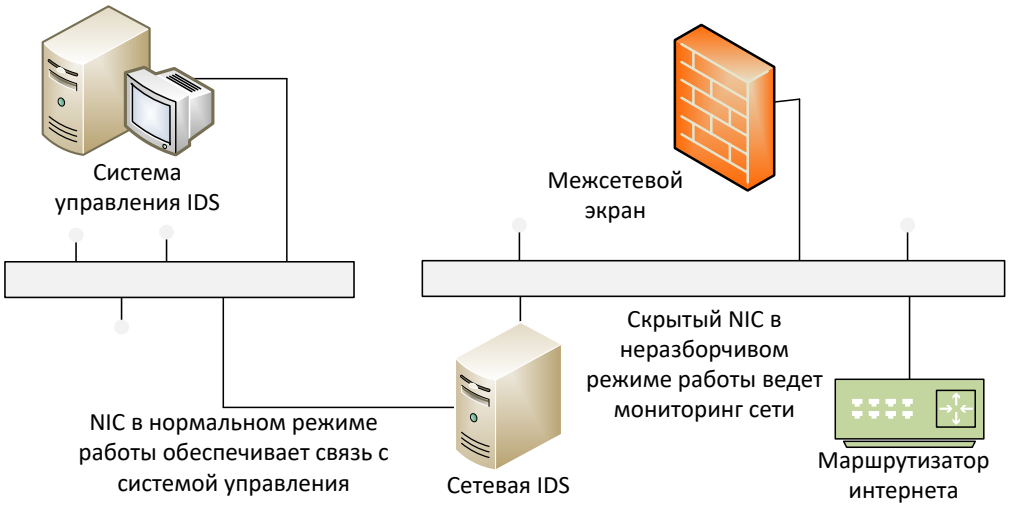


Рис. 17.7. Конфигурация NIDS с двумя сетевыми картами

Среди преимуществ использования NIDS можно выделить следующие аспекты.

- NIDS можно полностью скрыть в сети таким образом, что злоумышленник не будет знать о том, что за ним ведется наблюдение.
- Одна система NIDS может использоваться для мониторинга трафика с большим числом потенциальных систем-целей.
- NIDS может осуществлять перехват содержимого всех пакетов, направляющихся на систему-цель.

Среди недостатков NIDS необходимо отметить следующие аспекты.

- Система NIDS может только выдавать сигнал тревоги, если трафик соответствует предустановленным правилам или признакам.
- NIDS может упустить нужный интересуемый трафик из-за использования широкой полосы пропускания или альтернативных маршрутов.
- Система NIDS не может определить, была ли атака успешной.
- Система NIDS не может просматривать зашифрованный трафик.
- В коммутируемых сетях (в отличие от сетей с общими носителями) требуются специальные конфигурации, без которых NIDS будет проверять не весь трафик.

17.3.4. Использование IDS

Пример использования IDS приведен на рис. 17.8.

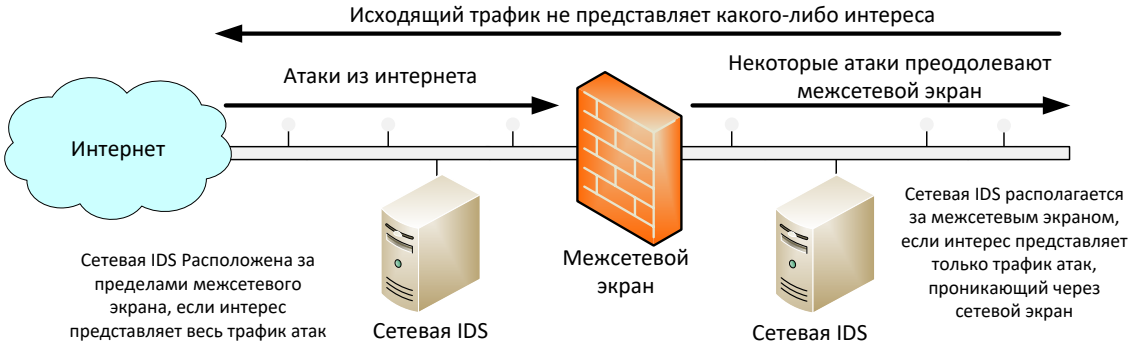


Рис. 17.8. Пример выбора объекта мониторинга

Таблица 17.3 – События, отслеживаемые при наличии политики IDS

Политика	NIDS	HIDS
Обнаружение атак	Весь трафик, поступающий на потенциально атакуемые системы (сетевые экраны, веб-серверы, серверы приложений и т.д.)	Неудачные попытки входа. Попытки соединения. Удачный вход с удаленных систем.
Предотвращение атак	То же, что и для обнаружения атак	То же, что и для обнаружения атак.
Обнаружение нарушений политики	Весь трафик HTTP, формируемый на системах клиентах. Весь трафик FTP, формируемый на системах клиентах	Успешные HTTP-соединения. Успешные FTP соединения. Загружаемые файлы.
Принуждение к использованию политик	То же, что и для обнаружения нарушений политики	То же, что и для обнаружения нарушения политики.
Принуждение к соответствию политикам соединений	Весь трафик, нарушающий принудительно используемую политику соединения	Успешные соединения с запрещенных адресов или по запрещенным портам.
Сбор доказательств	Содержимое всего трафика, формируемого на системе-цели или атакующей системе	Все успешные подключения, исходящие с атакующей системы. Все неудачные соединения с атакующих систем. Все нажатия клавиш из интерактивных сеансов на атакующих системах.

При обнаружении вторжения IDS должна выработать методы противодействия вторжению.

Рассматривают следующие виды действий при обнаружении вторжений.

- *Пассивная обработка* – это наиболее распространенный тип действий, предпринимаемых при обнаружении вторжения. Причина

этому проста – пассивные ответные действия обеспечивают меньшую вероятность повреждения легитимного трафика, являясь, в то же время, наиболее простыми для автоматического применения. Как правило, пассивные ответные действия осуществляют сбор большего числа информации или передают уведомления лицам, имеющим право на принятие более жестких мер.

- *Активная обработка события* – позволяет наиболее быстро предпринять возможные меры для снижения уровня вредоносного действия события. Однако если недостаточно серьезно отнестись к логическому программированию действий в различных ситуациях и не провести должного тестирования набора правил, активная обработка событий может вызвать повреждение системы или полный отказ в обслуживании легитимных пользователей. Среди активной обработки событий различают следующие.
- *Прерывание соединений, сеансов или процессов.* Вероятно, самым простым действием для понимания является прерывание события. Оно может осуществляться посредством прерывания соединения, используемого атакующим злоумышленником (это возможно только в том случае, если событие использует TCP-соединение), с закрытием сеанса пользователя или завершением процесса, вызвавшего неполадку.
- *Определение того, какой объект подлежит уничтожению*, выполняется посредством изучения события. Если процесс использует слишком много системных ресурсов, лучше всего завершить его. Если пользователь пытается использовать конкретную уязвимость или осуществить нелегальный доступ к файлам, то рекомендуется закрыть сеанс этого пользователя. Если злоумышленник использует сетевое соединение в попытках изучения уязвимостей системы, то следует закрыть соединение.

Таблица 17.4 – Примеры ответных действий, определяемые политикой IDS

Политика	Пассивные ответные действия	Активные ответные действия
Обнаружение атак	Ведение журналов. Ведение дополнительных журналов. Уведомление	Нет ответного активного действия.
Предотвращение атак	Ведение журналов. Уведомление.	Закрытие соединения. Завершение процесса. Возможна перенастройка маршрутизатора или межсетевого экрана.
Обнаружение нарушений политики	Ведение журналов. Уведомление	Нет ответного активного действия
Принудительное использование политик	Ведение журналов. Уведомление	Закрытие соединения. Возможно перенастройка прокси

Политика	Пассивные ответные действия	Активные ответные действия
Принудительное использование политик соединения	Ведение журналов. Уведомление	Закрытие соединения. Возможно перенастройка маршрутизатора или межсетевого экрана
Сбор доказательств	Ведение журналов. Ведение дополнительных журналов. Уведомление	Обманные действия. Возможно закрытие соединения

18. Обеспечение безопасного взаимодействия в телекоммуникационных сетях

Рассмотренные в гл. 15-16 угрозы нарушения ИБ в ТКС, способствуют активному развитию средств защиты. Однако средства защиты не ограничиваются только такими средствами анализа трафика и сетевой активности пользователей, как межсетевые экраны, виртуальные частные сети, системы предотвращения вторжений. Существенной частью способов обеспечения безопасности является аутентификация пользователей, использование цифровых подписей и шифрования. Кроме того, используемые способы обеспечения безопасности не должны использоваться бессистемно – отдельные способы и средства должны применяться совместно по единому замыслу, определяемому политикой информационной безопасности. Данная глава посвящена рассмотрению именно этих аспектов ИБ.

18.1. Аутентификация и управление сертификатами

18.1.1. Цифровые подписи

Цифровые подписи – это форма шифрования, обеспечивающая аутентификацию (подтверждение подлинности) цифровой информации.

С помощью цифровой подписи можно повысить уровень этой защиты и обезопасить информацию от изменения после получения и дешифрования.

До настоящего времени наиболее часто для построения схемы цифровой подписи использовался алгоритм RSA. В основе этого алгоритма лежит концепция Диффи-Хеллмана. Она заключается в том, что каждый пользователь сети имеет свой закрытый ключ, необходимый для формирования подписи; соответствующий этому секретному ключу открытый ключ, предназначенный для проверки подписи, известен всем другим пользователям сети.

На рис. 18.1 показана схема формирования цифровой подписи по алгоритму RSA. Подписанное сообщение состоит из двух частей: незашифрованной части, в которой содержится исходный текст T, и зашифрованной части, представляющей собой цифровую подпись.

Если результат расшифровки цифровой подписи совпадает с открытой частью сообщения, то считается, что документ подлинный, не претерпел никаких изменений в процессе передачи, а автором его является именно тот человек, который передал свой открытый ключ получателю. Если сообщение снабжено цифровой подписью, то получатель может быть уверен, что оно не было изменено или подделано по пути.

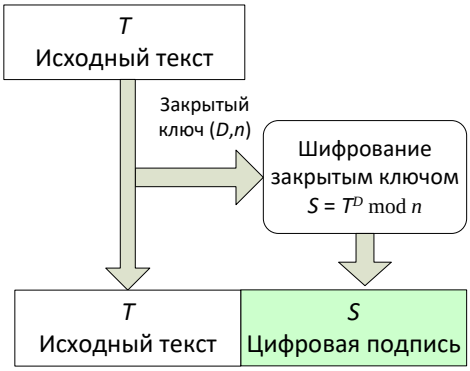


Рис. 18.1. Схема формирования цифровой подписи по алгоритму RSA

Цифровые подписи применяются к тексту до того, как он шифруется. Если помимо снабжения текста электронного документа цифровой подписью надо обеспечить его конфиденциальность, то вначале к тексту применяют цифровую подпись, а затем шифруют все вместе: и текст, и цифровую подпись (рис. 18.2).

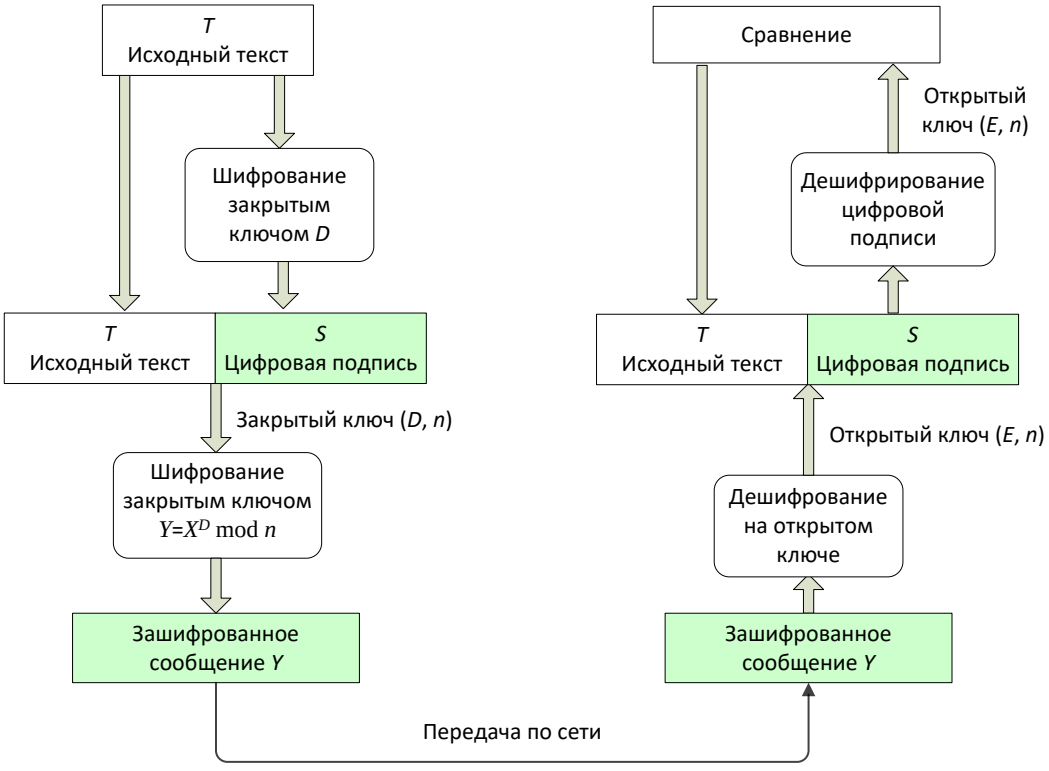


Рис. 18.2. Обеспечение конфиденциальности документа с цифровой подписью

18.1.2 Управление ключами и сертификация ключей

Управление ключами является самой неприятной задачей при использовании любой системы шифрования. Ключи представляют собой самые важные объекты во всей системе, так как если злоумышленник полу-

чает ключ, у него появляется возможность расшифровывать все данные, зашифрованные с помощью этого ключа. Управление ключами заключается не только в защите их при использовании. Данная задача предусматривает создание надежных ключей, безопасное распространение ключей среди удаленных пользователей, обеспечение корректности ключей, отмену в случае их раскрытия или истечения срока действия.

Если ключи некоторым образом передаются в удаленное место расположения, они должны проверяться при получении на предмет того, не подверглись ли они вмешательству в процессе передачи. Это можно делать вручную либо использовать некоторую форму цифровой подписи.

Открытые ключи предназначены для публикации или передачи другим пользователям и должны *сертифицироваться* как принадлежащие владельцу ключевой пары. Сертификация осуществляется с помощью *центрального бюро* сертификатов (Certificate Authority – CA). В данном случае CA предоставляет цифровую подпись на открытом ключе, и благодаря этому CA с доверием воспринимает тот факт, что открытый ключ принадлежит владельцу ключевой пары (см. рис. 18.3).

Цифровой сертификат представляет собой цифровой документ (небольшой файл), заверяющий подлинность и статус владельца для пользователя или компьютерной системы.

Бюро сертификатов (Certificate Authority – CA) – объединение, включающее сервер сертификатов и создающее сертификаты.

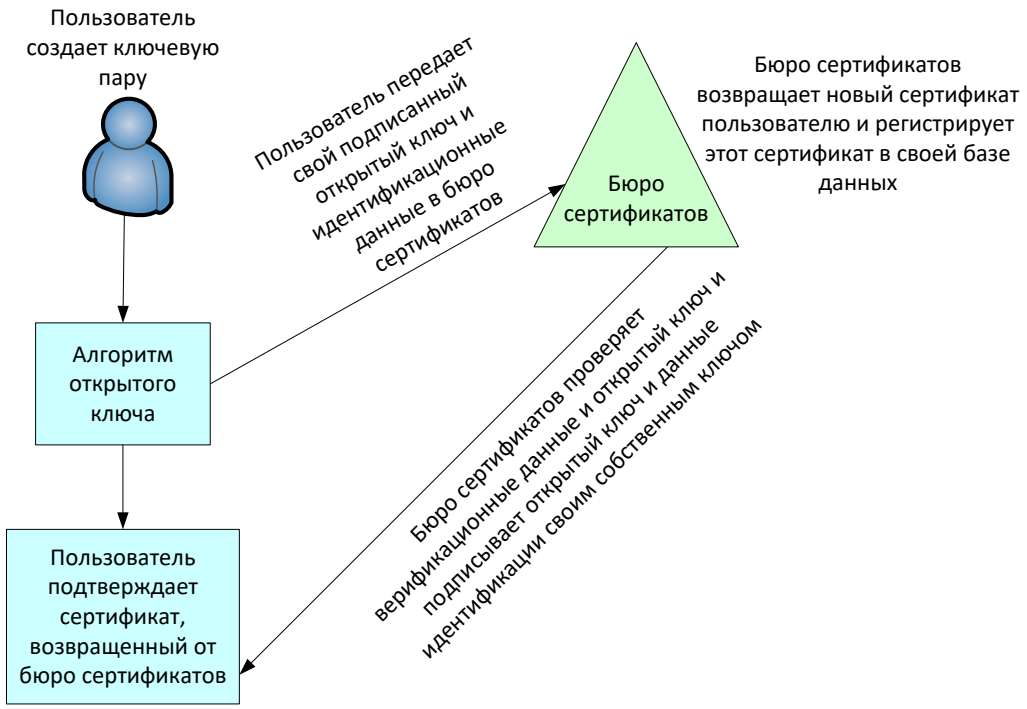


Рис. 18.3. Сертификация открытого ключа в бюро сертификатов

18.1.3. Концепция доверия в информационной системе

Концепция доверия является основополагающим принципом информационной безопасности и шифрования, в частности. Для работы шифрования необходима уверенность в том, что ключ шифра не будет раскрыт, и что используемый алгоритм шифрования является достаточно мощным. В случае с аутентификацией и цифровыми подписями необходима также уверенность в том, что открытый ключ на самом деле принадлежит тому, кто его использует.

18.1.3.1. Иерархическая модель доверия

Иерархическая модель доверия наиболее проста для восприятия. Говоря простым языком, в данном случае вы доверяете человеку, который находится выше в иерархической цепи, так как от него было получено соответствующее указание о необходимости доверия. На рис. 18.4 изображена схема этой модели.

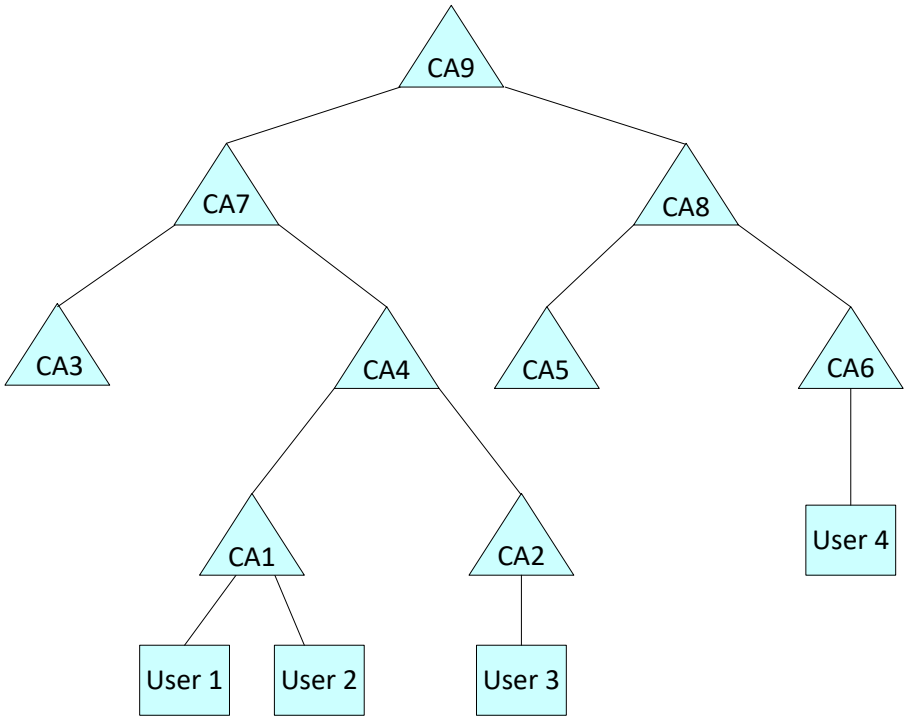


Рис. 18.4. Иерархическая модель доверия

Как видно из рисунка, пользователи $User_1$ и $User_2$ располагаются под CA_1. Следовательно, если CA_1 говорит, что сертификат открытого ключа принадлежит пользователю $User_1$, пользователь $User_2$ будет верить этому. На практике $User_2$ передает пользователю $User_1$ свой сертификат открытого ключа, подписанный CA_1. Пользователь $User_1$ проверяет подпись CA_1 с использованием открытого ключа CA_1. Так как CA_1 находится в иерархии выше, чем $User_1$, то $User_1$ доверяет CA_1 и, следовательно, доверяет сертификату пользователя $User_2$.

Если пользователю $User_1$ нужно проверить информацию от пользователя $User_3$, все несколько усложняется. CA_1 не знает о пользователе $User_3$, в отличие от CA_2. Тем не менее, пользователь $User_1$ не доверяет CA_2, так как это бюро сертификатов напрямую не принадлежит цепочке пользователя $User_1$. Следующий уровень вверх по цепочке – CA_4. Пользователь *User1* может верифицировать информацию от пользователя $User_3$ посредством проверки с помощью CA_4 следующим образом.

- Пользователь $User_1$ смотрит на сертификат пользователя $User_3$. Он подписан в CA_2.
- Пользователь $User_1$ получает сертификат пользователя CA_2. Он подписан в CA_4.
- Так как пользователь $User_1$ доверяет CA_4, открытый ключ CA_4 может использоваться для верификации сертификата *CA2*.
- Как только сертификат CA_2 верифицирован, пользователь $User_1$ может верифицировать сертификат пользователя $User_3$.
- Как только будет верифицирован сертификат пользователя $User_3$, пользователь $User_1$ может использовать открытый ключ пользователя $User_3$ для верификации данных.

18.1.3.2. Сетевая модель доверия

Сеть с доверием представляет собой альтернативную модель доверия. Эта концепция была впервые использована в технологии Pretty Good Privacy (PGP).

Сетевая модель доверия заключается в том, что каждый пользователь сертифицирует свой сертификат и передает его известным ассоциированным объектам. Эти объекты могут подписать сертификат другого пользователя, так как он известен (см. рис. 18.5).

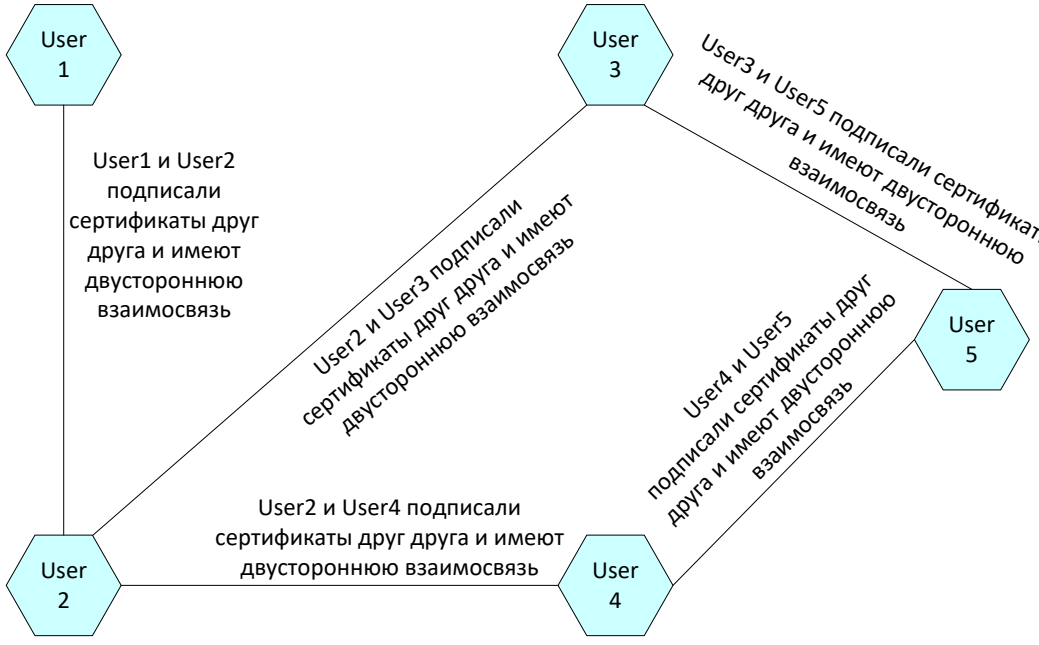


Рис. 18.5. Сетевая модель доверия

В такой модели не существует центрального бюро сертификатов. Если пользователю $User_1$ требуется верифицировать информацию, поступающую от пользователя $User_2$, он запрашивает сертификат пользователя $User_2$. Так как пользователь $User_1$ знает пользователя *User2*, то доверяет сертификату и даже может его подписать.

Теперь рассмотрим ситуацию, в которой $User_1$ получает информацию от $User_3$. Пользователь $User_3$ не известен пользователю $User_1$, но у пользователя $User_3$ есть сертификат, подписанный пользователем $User_2$. Таким образом, рассматриваемая модель распространяется на всю компьютерную сеть. Единственным решением, которое должно приниматься в процессе работы, является число переходов, которому доверяет пользователь. Как правило, это число равно 3 или 4. Кроме того, может возникнуть ситуация, в которой для установления доверия другому пользователю есть два пути. Например, $User_2$ может использовать два пути установления доверия с пользователем $User_5$: один через пользователя $User_3$ и другой через пользователя $User_4$. Так как оба пользователя $User_3$ и $User_4$ сертифицируют пользователя $User_5$, пользователь $User_2$ может быть уверен в сертификате пользователя $User_5$.

Главной проблемой, связанной с такой моделью доверия, является *недостаток масштабируемости.* Так как модель сети состоит из двусторонних взаимоотношений, каждый пользователь должен иметь некоторое число таких взаимосвязей, чтобы пользоваться в сети каким-либо доверием. На практике такие взаимосвязи могут отсутствовать, так как большинство пользователей работают с небольшим числом связей и редко выходят на уровень трех или четырех переходов.

18.1.4. Аутентификация с использованием протоколов открытого ключа

Протоколы открытых ключей позволяют устанавливать авторизованные шифруемые связи между узлами внутренних сетей и в интернете.

Существуют три модели аутентификации, проводимой в этих протоколах; они используются как по отдельности, так и в комбинации.

- *Аутентификация клиента.* Позволяет серверу Windows 2012 VPN или веб-серверу IIS идентифицировать пользователя с использованием стандартных методов шифрования на открытом ключе. Осуществляет проверку подлинности сертификата клиента и общего ID, а также проверку того, что эти данные сгенерированы бюро сертификатов, корневой сертификат которого установлен в перечне доверенных CA. Эта проверка очень важна, если сервером является банк, который передает конфиденциальную финансовую информацию клиенту и должен подтвердить личность получателя. На рис. 18.6 отображен процесс аутентификации.

- *Аутентификация сервера.* Позволяет клиенту VPN или браузеру клиента SSL/TLS подтверждать идентичность сервера, проверяя правильность сертификата сервера и идентификатора ID, а также то, что сертификаты выпущены бюро сертификатов (CA), корневой сертификат которого присутствует в перечне доверенных CA клиента. Это подтверждение имеет важное значение для пользователя веб-сайта, который отправляет номер кредитной карты через сеть и хочет удостовериться в том, что это именно тот сервер, который ему нужен.
- *Взаимная аутентификация.* Позволяет клиенту и серверу авторизовать друг друга единовременно. Взаимная аутентификация требует, чтобы клиент и сервер имели цифровые сертификаты и соответствующие корневые сертификаты CA в перечнях доверенных CA.

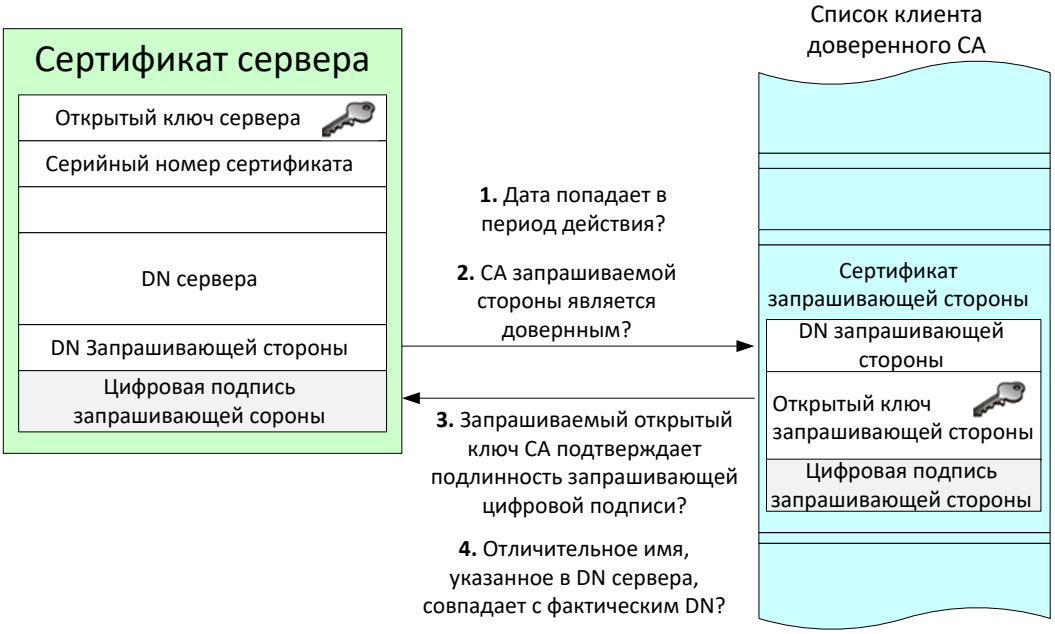


Рис. 18.6. Аутентификация сервером сертификата клиента

18.2. Протокол конфиденциального обмена данными SSL

Протокол SSL спроектирован для обеспечения конфиденциальности обмена между двумя прикладными процессами клиента и сервера. Он предоставляет возможность аутентификации сервера и, опционально, клиента. SSL требует применения надежного транспортного протокола (например, TCP).

Преимуществом SSL является то, что он независим от прикладного протокола. Протоколы приложения, такие как HTTP, FTP, TELNET и т.д. могут работать поверх протокола SSL совершенно прозрачно. Протокол SSL может согласовывать алгоритм шифрования и ключ сессии, а также

аутентифицировать сервер до того, как приложение примет или передаст первый байт данных.

Все протокольные прикладные данные в SSL передаются зашифрованными с гарантией конфиденциальности.

Протокол SSL предоставляет «безопасный канал», который имеет три основные свойства.

- *Канал является частным.* Шифрование используется для всех сообщений после простого диалога, который служит для определения секретного ключа.
- *Канал аутентифицирован.* Серверная сторона диалога всегда аутентифицируется, в то время как клиентская – аутентифицируется опционально.
- *Канал надежен.* Транспортировка сообщений включает в себя проверку целостности (с привлечением MAC – Message Authentication Code).

Сеанс SSL между клиентом и сервером устанавливается следующим образом.

1. Клиент открывает сокет и запрашивает подключение к серверу.
2. Сервер аутентифицирует клиента (либо по паролю, либо посредством сертификата, отправляемого клиентом).
3. После установки соединения сервер передает браузеру свой открытый ключ посредством отправки сертификата сервера, выпущенного доверенным бюро сертификатов.
4. Клиент аутентифицирует сертификат.
5. Клиент и сервер осуществляют обмен настроечной информацией для определения типа и силы шифрования, используемых в сеансе соединения.
6. Клиент создает сеансовый ключ, используемый для шифрования данных.
7. Клиент шифрует сеансовый ключ с помощью открытого ключа сервера (полученного из сертификата сервера) и отправляет его серверу. Секретный ключ, с помощью которого можно расшифровать сеансовый ключ, находится только на сервере.
8. Сервер расшифровывает сеансовый ключ и использует его для создания безопасной сессии, через которую будет осуществляться обмен данными с клиентом.

В несколько упрощенном варианте диалог SSL представлен на рис. 18.7.

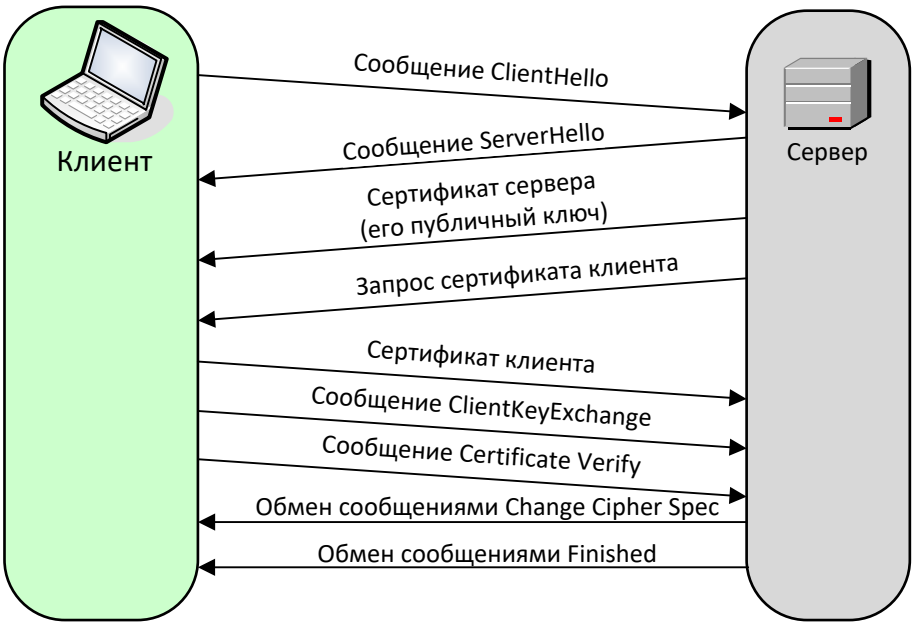


Рис. 18.7. Алгоритм работы SSL

Необходимым условием успешной реализации этих шагов является заранее установленный на клиенте корневой сертификат, полученный от доверенного бюро сертификатов. При использовании сертификата, полученного от коммерческого CA, корневой сертификат которого уже имеется в браузере (например, Verisign), не нужно беспокоиться об этом. При использовании сертификатов клиентов серверу необходимо установить клиентский корневой сертификат, выпущенный клиентским бюро сертификатов.

Протокол диалога SSL имеет две основные фазы. Первая фаза используется для установления конфиденциального канала коммуникаций. Вторая служит для аутентификации клиента.

Фаза 1. Первая фаза является фазой инициализации соединения, когда оба партнера посылают сообщения HELLO. Клиент инициирует диалог посылкой сообщения CLIENT-HELLO. Сервер, получив это сообщение, обрабатывает его и откликается сообщением SERVER-HELLO.

К этому моменту, как клиент, так и сервер имеют достаточно информации, чтобы знать, нужен ли новый мастерный ключ. Когда новый мастерный ключ не нужен, клиент и сервер немедленно переходят в фазу 2.

Когда нужен новый мастерный ключ, сообщение SERVER-HELLO будет содержать достаточно данных, чтобы клиент мог сформировать такой ключ. Сюда входит:

- подписанный сертификат сервера;
- список базовых шифров (см. ниже);
- идентификатор соединения (последний представляет собой случайное число, сформированное сервером и используемое на протяжении сессии).

Клиент генерирует мастерный ключ и посылает сообщение CLIENT-MASTER-KEY (или сообщение ERROR, если информация сервера указывает, что клиент и сервер не могут согласовать базовый шифр).

Здесь следует заметить, что каждая оконечная точка SSL использует пару шифров для каждого соединения (т.е. всего 4 шифра). На каждой конечной точке, один шифр используется для исходящих коммуникаций и один – для входящих. Когда клиент или сервер генерирует ключ сессии, они в действительности формируют два ключа, SERVER-READ-KEY (известный также как CLIENT-WRITE-KEY) и SERVER-WRITE-KEY (известный также как CLIENT-READ-KEY). Мастерный ключ используется клиентом и сервером для генерации различных ключей сессий.

Наконец, после того как мастерный ключ определен, сервер посылает клиенту сообщение SERVER-VERIFY. Этот заключительный шаг аутентифицирует сервер, так как только сервер, который имеет соответствующий общедоступный ключ, может знать мастерный ключ.

Фаза 2. Вторая фаза является фазой аутентификации. Сервер уже аутентифицирован клиентом на первой фазе, по этой причине здесь осуществляется аутентификация клиента. При типичном сценарии серверу необходимо получить что-то от клиента, и он посылает запрос. Клиент пришлет позитивный отклик, если располагает необходимой информацией, или пришлет сообщение об ошибке, если нет. Эта спецификация протокола не определяет семантику сообщения ERROR, посылаемого в ответ на запрос сервера (например, конкретная реализация может игнорировать ошибку, закрыть соединение, и т.д. и, тем не менее, соответствовать этой спецификации). Когда один партнер выполнил аутентификацию другого партнера, он посылает сообщение FINISHED. В случае клиента сообщение CLIENT-FINISHED содержит зашифрованную форму идентификатора CONNECTION-ID, которую должен верифицировать сервер. Если верификация терпит неудачу, сервер посылает сообщение ERROR.

Раз партнер послал сообщение FINISHED он должен продолжить воспринимать сообщения до тех пор, пока не получит сообщение FINISHED от партнера. Как только оба партнера послали и получили сообщения FINISHED, протокол диалога SSL закончил свою работу. С этого момента начинает работать прикладной протокол.

18.3. Обеспечение безопасности беспроводных сетей

В беспроводных локальных сетях главным образом используется группа стандартов 802.11x (a, b, g и т. д.) технологии Wi-Fi. Эти стандарты позволяют соединять рабочие станции каналами с пропускной способностью до 54 Мбит/с с использованием беспроводной точки доступа, которая подключается к кабельной сети или напрямую к другой рабочей станции (см. рис. 18.8).

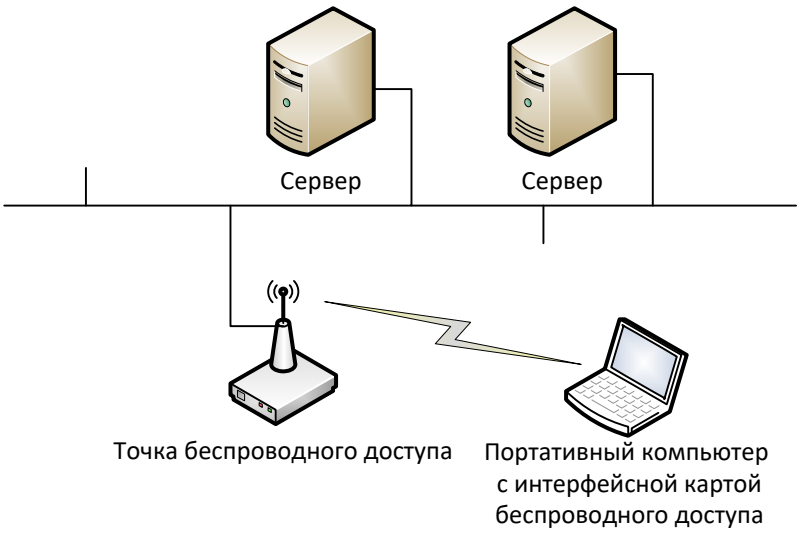


Рис. 18.8. Типичная архитектура беспроводной сети

Так как беспроводные сети используют воздух и пространство для передачи и приема информации (сигналы являются открытыми для любого лица, находящегося в зоне действия), безопасность передачи данных является очень важным аспектом безопасности всей системы в целом. Без обеспечения должной защиты конфиденциальности и целостности информации при ее передаче между рабочими станциями и точками доступа нельзя быть уверенным в том, что информация не будет перехвачена злоумышленником, и что рабочие станции и точки доступа не будут подменены посторонним лицом.

18.3.1. Угрозы безопасности беспроводных соединений

18.3.1.1. Обнаружение беспроводных сетей

Обнаружить WLAN очень легко. Действительно, именно для этой цели был разработан ряд средств. Одной из таких утилит является NetStumber (http://www.netstumber.com/); она работает в операционных системах семейства Windows и может использоваться совместно со спутниковым навигатором (ресивером глобальной системы позиционирования, GPS) для обнаружения беспроводных сетей WLAN. Данная утилита идентифицирует SSID сети WLAN, а также определяет, используется ли в ней WEP. Существуют и другие средства, идентифицирующие рабочие станции, подключенные к точке доступа, а также их MAC-адреса например, Kismet (http://www.kismetwireless.net/).

18.3.1.2. Прослушивание

Беспроводные сети по своей природе позволяют соединять с физической сетью компьютеры, находящиеся на некотором расстоянии от нее, как если бы эти компьютеры находились непосредственно в сети. Такой подход позволит подключиться к беспроводной сети организации, располага-

ющейся в здании, человеку, сидящему в машине на стоянке рядом с ним (см. рис. 18.9).

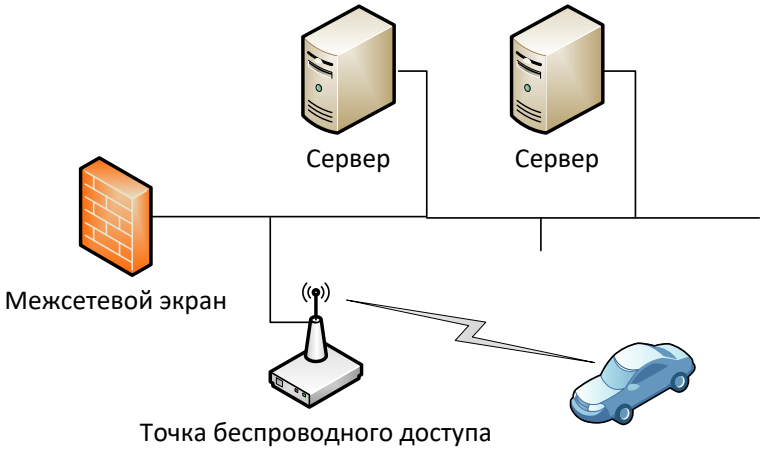


Рис. 18.9. Прослушивание сети WLAN

18.3.1.3. Активные атаки

Несмотря на то, что прослушивание сети представляет серьезную опасность, активные атаки могут быть еще более опасными. Основной риск, связанный с беспроводными сетями, состоит с том, что злоумышленник может успешно преодолеть периметр сетевой защиты организации. Не следует полагать, что атаки с использованием уязвимостей – это единственный способ злонамеренного воздействия злоумышленников. Если хакер прослушивает сеть, он может также перехватить пароли и пользовательские идентификаторы. Основные атаки, проводимые на WLAN, связанны с перехватом информации передаваемой по сети за счет низкой криптостойкости алгоритмов шифрования WEP.

18.3.2. Протокол WEP

Стандарт 802.11x определяет протокол Wired Equivalent Privacy (WEP) для защиты информации при ее передаче через WLAN.

WEP предусматривает обеспечение трех основных аспектов, обеспечивающих безопасность.

- *Аутентификация.* Служба аутентификации WEP используется для аутентификации рабочих станций на точках доступа. В аутентификации открытых систем рабочая станция рассматривается как аутентифицированная, если она отправляет ответный пакет с MAC-адресом в процессе начального обмена данными с точкой доступа. В реальных условиях данная форма аутентификации не обеспечивает доказательства того, что к точке доступа подключается именно конкретная рабочая станция, а не какой-либо другой компьютер.

- *Конфиденциальность.* Механизм обеспечения конфиденциальности базируется на RC4. RC4 – это стандартный мощный алгоритм шифрования, поэтому атаковать его достаточно сложно. WEP определяет систему на базе RC4, обеспечивающую управление ключами, и другие дополнительные службы, необходимые для функционирования алгоритма. WEP поддерживает ключи длиной 40 бит и 128 бит (непосредственный ключ комбинируется с вектором инициализации алгоритма). К сожалению, WEP не определяет механизм управления ключами. Это означает, что многие инсталляции WEP базируются на использовании статических ключей. Действительно, часто на всех рабочих станциях сети используются одни и те же ключи.
- *Целостность.* Спецификация протокола WEP включает контроль целостности для каждого пакета. Используемая проверка целостности представляет собой циклическую 32-битную проверку избыточности (CRC). CRC вычисляется для каждого пакета перед его шифрованием, после чего данные в комбинации с CRC шифруются и отправляются в пункт назначения. Несмотря на то что CRC с криптографической точки зрения небезопасна, она защищается шифрованием. Используемая здесь система шифрования может быть достаточно надежной, если алгоритм шифрования обладает достаточной мощностью. Однако недостатки WEP представляют угрозу и для целостности пакетов.

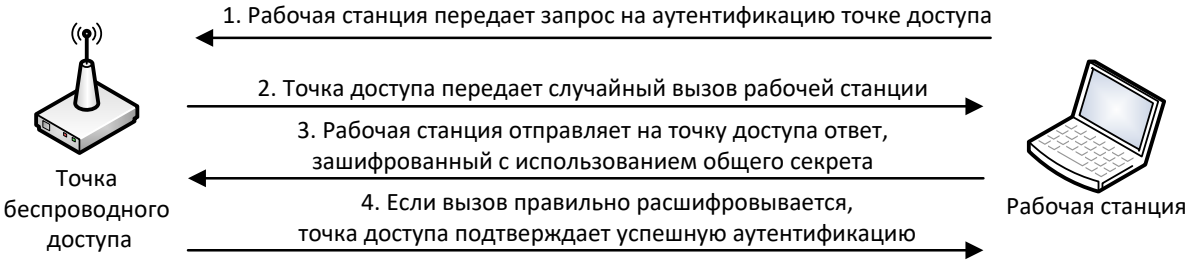


Рис. 18.10. Аутентификационный обмен WEP

Протокол WEP предусматривает использование службы аутентификации. К сожалению, эта служба осуществляет только аутентификацию рабочей станции относительно точки доступа. Она не обеспечивает взаимную аутентификацию, поэтому рабочая станция не получает доказательства того, что точка доступа действительно является авторизованной точкой доступа к сети. Таким образом, использование WEP не предотвращает перехват данных или атаки через посредника (см. рис. 18.11).

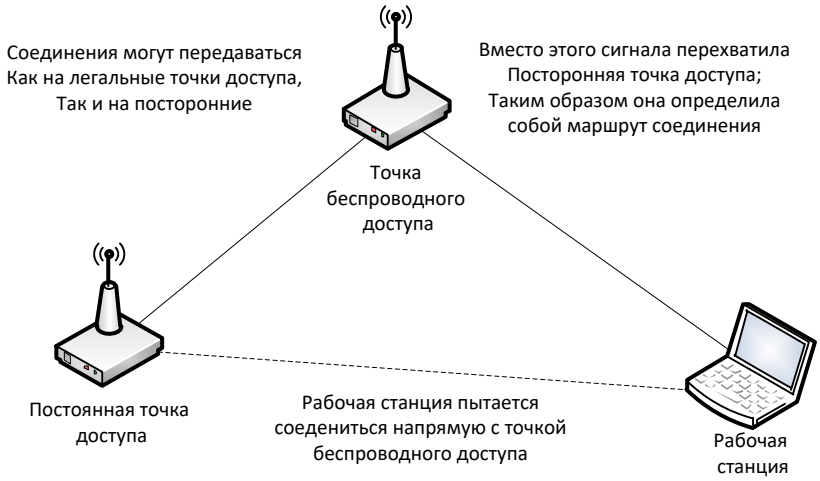


Рис. 18.11. Атака на WEP через посредника

14.3.3. Протокол 802.1X – контроль доступа в сеть по портам

Протокол 802.1X разработан в качестве надстройки для всех протоколов контроля доступа 2 уровня, включая Ethernet и WLAN. Так как этот протокол был разработан в то время, когда создатели WLAN искали решения проблем, связанных с WEP, он пришелся как нельзя кстати.

Протокол предназначен для обеспечения обобщенного механизма аутентификации при доступе в сеть и предусматривает следующий набор элементов.

1. Аутентификатор. Сетевое устройство, осуществляющее поиск других объектов для аутентификации; для WLAN это может быть точка доступа.
2. Соискатель. Объект, которому требуется доступ. В случае с WLAN это может быть рабочая станция.
3. Сервер аутентификации. Источник служб аутентификации. 802.1X разрешает централизацию этой функции, поэтому данный сервер является, например, сервером RADIUS.
4. Сетевая точка доступа. Точка присоединения рабочей станции к сети. По сути, это порт на коммутаторе или концентраторе. В беспроводной технологии является связью между рабочей станции и точкой доступа.
5. Процесс доступа через порт (PAE). PAE – это процесс, выполняющий протоколы аутентификации. PAE есть как у аутентификатора, так и у соискателя.
6. Расширяемый протокол аутентификации (EAP). Протокол EAP (определен в стандарте RFC 2284) представляет собой протокол, используемый при обмене аутентификационными данными. Поверх EAP могут работать и другие протоколы аутентификации более высокого уровня.

Использование протокола 802.1x позволяет применить более надежный механизм аутентификации, нежели возможности, доступные в 802.11x. При использовании совместно с сервером RADIUS становится возможным централизованное управление пользователями.

18.4. Обеспечение безопасности электронной почты

18.4.1. Риски, связанные с использованием электронной почты

Электронная почта – один из наиболее широко используемых видов сервиса, как в корпоративных сетях, так и в Интернет. Она является не просто способом доставки сообщений, а важнейшим средством коммуникации, распределения информации и управления различными процессами в бизнесе. Роль электронной почты становится очевидной, если рассмотреть функции, которые выполняет почта:

- обеспечивает внутренний и внешний информационный обмен;
- является компонентом системы документооборота;
- формирует транспортный протокол корпоративных приложений;
- является средством образования инфраструктуры электронной коммерции.

Благодаря выполнению этих функций электронная почта решает одну из важнейших на настоящий момент задач – формирует единое информационное пространство. В первую очередь это касается создания общей коммуникационной инфраструктуры, которая упрощает обмен информацией между отдельными людьми, подразделениями одной компании и различными организациями.

Электронная почта обладает рядом преимуществ по сравнению с обычными способами передачи сообщений (традиционная почта или факсимильная связь). К ним относятся следующие.

1. Оперативность и легкость использования.
2. Доступность практически в любом месте.
3. Универсальность форматов писем и вложений.
4. Дешевизна сервиса.
5. Надежность и скорость инфраструктуры доставки.
6. Использование для обработки электронной почты прикладного специального программного обеспечения.

Электронная почта обладает многочисленными достоинствами, но именно из-за этих достоинств возникают основные риски, связанные с ее использованием. К примеру, доступность электронной почты превращается в недостаток, когда пользователи начинают применять почту для рассылки спама, легкость в использовании и бесконтрольность приводит к утечкам информации, возможность пересылки разных форматов документов – к распространению вирусов и т.д.

В конечном итоге любой из этих рисков может привести к серьезным последствиям для компании. Это и потеря эффективности работы, и снижение качества услуг информационных систем, и разглашение конфиденциальной информации. Недостаточное внимание к этой проблеме грозит значительными потерями в бизнесе, а в некоторых случаях, даже привлечением к юридической ответственности в связи с нарушением законодательства.

Компания подвергается этим рискам в силу ряда свойств электронной почты. Например, благодаря применению MIME-стандарта электронная почта может переносить большие объемы информации различных форматов данных в виде прикрепленных к сообщениям файлов. Такой возможностью сразу воспользовались злоумышленники.

Достоинство электронной почты превратилось в угрозу, поскольку электронная почта стала представлять собой практически идеальную среду для переноса различного рода «опасных» вложений, а именно компьютерных вирусов, вредоносных программ, «троянских» программ и т.п.

Если надлежащий контроль за использованием электронной почты не обеспечен, это может привести к чрезвычайно серьезным последствиям и даже нанести непоправимый ущерб. Избавиться от этого риска можно лишь путем блокировки писем с «опасными» вложениями, а также антивирусной проверки прикрепленных файлов. На практике же оптимальным средством может оказаться блокировка определенных типов файлов. Это, как правило, исполняемые файлы (exe, com, bat) и файлы, содержащие макросы и OLE-объекты (файлы, созданные в приложениях MS Office).

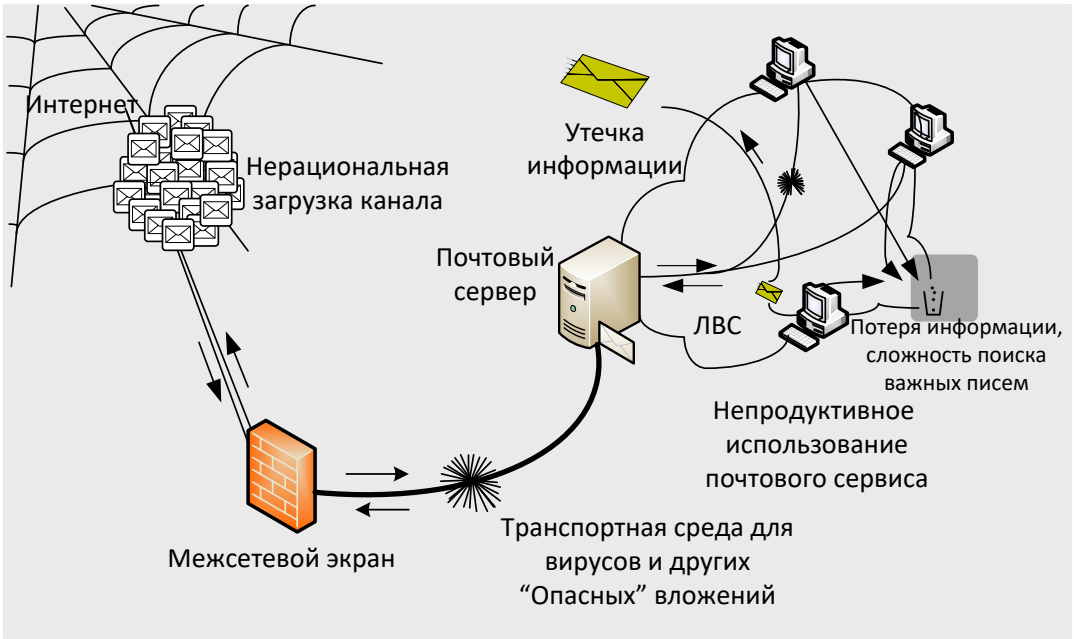


Рис. 18.12. Негативное воздействие различных факторов на незащищенную почтовую систему

Серьезную опасность для корпоративной сети представляют различного рода атаки с целью «засорения» почтовой системы. Это, в первую

очередь, пересылка в качестве вложений в сообщениях электронной почты файлов больших объемов или многократно заархивированных файлов. «Открытие» таких файлов или попытка «развернуть» архив может привести к «зависанию» системы. При этом одинаково опасны как умышленные атаки этого типа, например, «отказ в обслуживании» (Denial of Service) и «почтовые бомбы» (mail-bombs), так и «неумышленные», когда пользователи отправляют электронные письма с вложениями большого объема, просто не подумав о том, к каким последствиям может привести открытие подобного файла на компьютере адресата.

Действенный способ избавиться от «засорения» почтовой системы и ее перегрузки – фильтрация по объему передаваемых данных, по количеству вложений в сообщения электронной почты и глубине вложенности архивированных файлов.

Другой особенностью электронной почты является ее доступность и простота в использовании. Во многом результатом этого стало широкое и повсеместное применение этого вида сервиса Интернет. Стихийность развития и отсутствие единых правил функционирования почтового сервиса привели к неконтролируемому использованию электронной почты, а в связи с этим, и к возникновению целого ряда рисков, связанных с неуправляемой циркуляцией электронной почты в сети.

Отсутствие контроля над почтовым потоком, как правило, становится причиной того, что сотрудники компании используют электронную почту в целях, не связанных с деятельностью компании (например, для обмена видео-файлами и графикой, частной переписки, ведения собственного бизнеса с использованием почтовых ресурсов компании, рассылки резюме в различные организации и т.п.).

Кроме того, к такому же результату может привести непродуктивное использование почтовых ресурсов в трудовой деятельности сотрудников (например, чрезмерное увлечение почтовой перепиской в случаях, когда необходимости в такой переписке нет, использование электронной почты не по назначению и т.п.). Причиной этого, как правило, является отсутствие в компании правил, регламентирующих использование системы электронной почты. Последствиями непродуктивного использования почтового сервиса являются снижение производительности труда в компании, а также излишние финансовые затраты. Сэкономить средства в значительной степени поможет проведение анализа эффективности использования системы электронной почты, который основывается на базе статистических данных о функционировании системы. Подобную статистику возможно получить лишь в случае ведения архива электронной почты. Обработка информации, содержащейся в архиве, позволяет получать отчеты о различных параметрах электронной почты, ее объемах и структуре, представить наглядную картину использования почтового трафика сотрудниками компании, а это, в свою очередь, поможет предотвратить использование электронной почты, несвязанное с деятельностью компании, и повысить эффективность работы корпоративной почтовой системы.

Передача в электронных письмах графических, видео и звуковых файлов, которые, как правило, имеют большой объем даже если такая передача предусмотрена условиями ведения бизнеса, приводит к значительной перегрузке сети, соответственно к дополнительным финансовым затратам на ее обслуживание. Избежать этого, а значит и добиться значительной экономии средств компании, поможет, так называемая, отложенная доставка писем, которая позволяет доставлять сообщения больших объемов в то время, когда загрузка сети не имеет критического значения (например, в ночное время, в выходные дни и т.п.).

К «засорению» трафика также ведет рассылка спама. Как правило, это письма, содержащие навязчивые предложения самых разнообразных услуг, товаров и т.п. Такого рода почта является «группой риска» с точки зрения переноса вирусов. Большое количество ненужной почты загружает каналы, «замусоривает» почтовые ящики, отнимает время на удаление ненужных писем и повышает вероятность случайного удаления нужных. Конечно рассылка, например, сообщений рекламного характера, напрямую не преследует цели «засорить» почтовую систему организации, однако косвенно приводит к негативным последствиям. Использование списков рассылки, в которую могут входить все пользователи одной корпоративной сети, и получение одновременно всеми этими пользователями сообщений рекламного характера грозит компании снижением производительности ее сетевых ресурсов. Блокировка спама, в первую очередь, связана с контекстным анализом сообщений, то есть проверкой электронной почты на наличие ключевых слов и выражений, которые обычно употребляются в сообщениях рекламного характера.

Переписка с внешними корреспондентами представляет наибольшую угрозу из-за особенностей электронной почты (невозможность контролировать маршрут передачи писем, а также их копирование и перенаправление, осуществлять аутентификацию отправителя/получателя, возвращать письма после их отправления). Кроме того, невозможен либо затруднен контроль количества отправляемых копий письма. Содержимое сообщения может быть прочитано в процессе передачи его по Интернету, поскольку заголовки и содержимое электронных писем часто передаются в открытом виде.

Другой проблемой, связанной с особенностями электронной почты, является то, что электронная почта позволяет неконтролируемое накопление информации в архивах и практически неуничтожима. В противоположность бытующему мнению, удалить электронную почту непросто. Резервные копии сообщений могут оставаться на персональных компьютерах отправителя и получателя или в сети компаний, где они работают. Если электронное почтовое сообщение отправлено через коммерческую службу или через Интернет, то оно будет передаваться через несколько различных серверов. Каждый сервер в цепочке между отправителем и получателем может сохранить копию сообщения в своих архивах. Даже методичное выяснение местонахождения каждой копии электронного письма с последу-

ющим его удалением не дает никакой гарантии того, что сообщение не осталось на жестком диске компьютера или сервера.

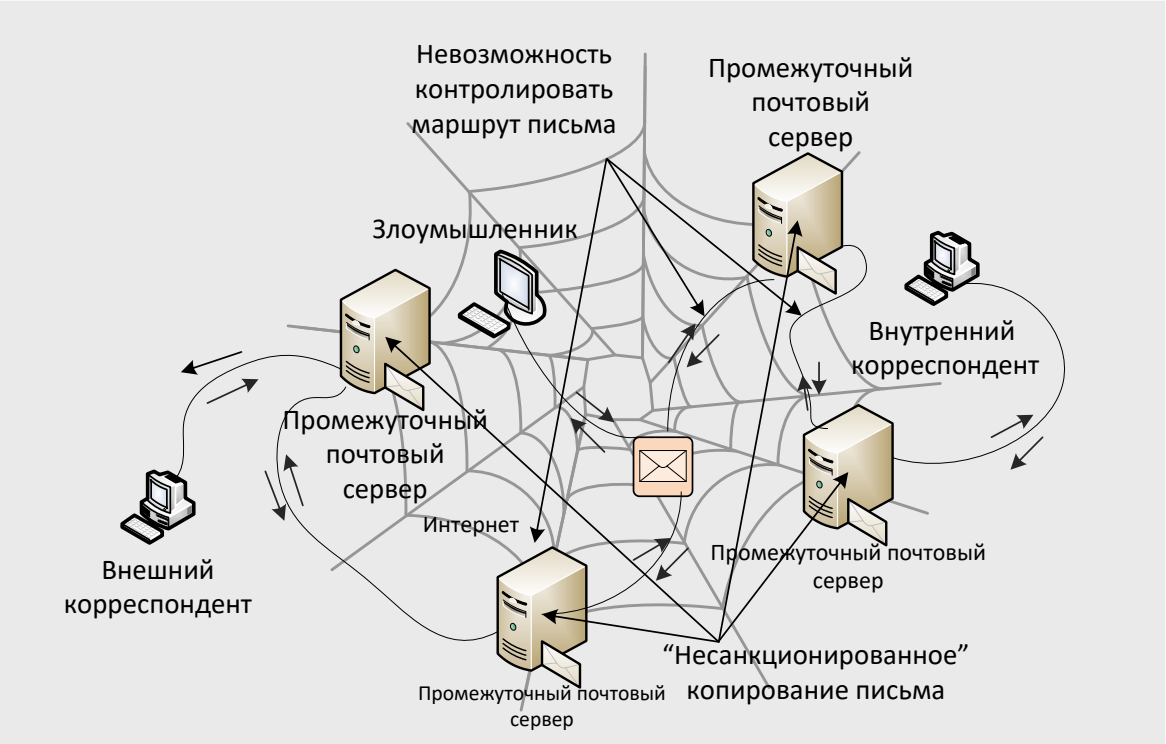


Рис. 18.13. Проблемы, возникающие при пересылке электронной почты через Интернет

Все эти особенности, а также простота копирования электронного сообщения и невозможность проконтролировать данную операцию приводят к тому, что сотрудник может передать корпоративную информацию любому количеству людей как внутри, так и за пределами компании анонимно и без соответствующего разрешения, сразу или по истечении какого-либо времени. При этом, такая информация может представлять собой служебную информацию компании это грозит серьезным нарушением конфиденциальности и может привести к неприятным для компании последствиям.

Чтобы обеспечить защиту от утечки конфиденциальной информации из сети, необходимо осуществлять контроль адресатов, фильтрацию передаваемых данных на наличие в текстах сообщений или в прикрепленных к электронному письму файлах слов и выражений, имеющих отношение к «закрытой» тематике, осуществлять разграничение доступа различных категорий пользователей к архивам электронной почты и т.п.

Одно из основных отличий электронной почты состоит в формальном к ней отношении (по сравнению с другими видами коммерческих коммуникаций):

– во-первых, большинство пользователей относятся к электронной почте как к чему-то временному, то есть поступают с ней по

принципу «прочитал и выкинул». При таком отношении существует риск случайного удаления значимой информации. Кроме того, существует опасность потери переписки с важным клиентом. Все эти проблемы решаются путем создания в организации архива электронной почты;

– во-вторых, такое отношение к электронной почте приводит к тому, что из-за кажущейся недолговечности электронных сообщений люди часто используют их для того, чтобы выразить чувства и мнения в выражениях, которые они никогда не позволили бы себе употребить в традиционных письмах. Публикация таких писем в сети может нанести серьезный ущерб репутации компании или явиться причиной юридических исков к ней.

Еще одна область связана с возможностью привлечения к юридической ответственности компании и ее сотрудников – за нарушение авторского права. Защищенные этим правом материалы могут содержаться или в сообщении электронной почты, или в присоединенных файлах. К подобным материалам относятся графическая, аудио, видео и различная текстовая информация, т. е. любая информация, которая может быть представлена в электронном виде и передана по компьютерным сетям. Копирование или распространение этих материалов без предварительного согласия автора или владельца авторских прав является нарушением закона. Если компания допускает, чтобы материалы почты, защищенные авторским правом, использовались сотрудниками, не имеющими на это полномочий, то она может быть привлечена к ответственности за прямое или косвенное пособничество нарушению авторского права.

18.4.2. Средства обеспечения безопасности электронной почты

Учитывая описанные выше риски, связанные с использованием электронной почты, организациям необходимо принять соответствующие меры для защиты от них.

Подход к защите должен быть всесторонним и комплексным – необходимо сочетать организационные меры с использованием соответствующих технических средств.

К организационным мерам относятся разработка и внедрение в компании политики использования электронной почты. Технические средства должны обеспечить выполнение этой политики как за счет мониторинга почтового трафика, так и за счет адекватного реагирования на нарушения.

Очень важно отметить, что политика использования электронной почты первична по отношению к средствам ее реализации, поскольку составляет основу для формирования комплекса мер по защите информационной системы от вышеперечисленных рисков. Сначала необходимо сформулировать политику, составить правила использования электронной почты, определить, как созданная система должна реагировать на опреде-

ленные нарушения политики и только затем переводить правила на компьютерный язык того средства, которое используется для контроля выполнения положений политики использования электронной почты.

К таким техническим средствам относится специальное программное обеспечение, называемое система контроля содержимого электронной почты (content security software). В функции таких систем входит контроль почтового трафика и ведение архива переписки по электронной почте. К таким системам предъявляются следующие требования:

- проведение текстового анализа;
- фильтрация передаваемых данных:
  - по размеру и объему данных;
  - по количеству вложений в сообщения электронной почты;
  - по типу файлов (вложенных в электронную почту);
  - по адресу электронной почты;
- контроль использования почтовых ресурсов и разграничение доступа к ним различных категорий пользователей;
- отложенную доставку сообщений электронной почты по расписанию;
- ведение полнофункционального архива электронной почты.

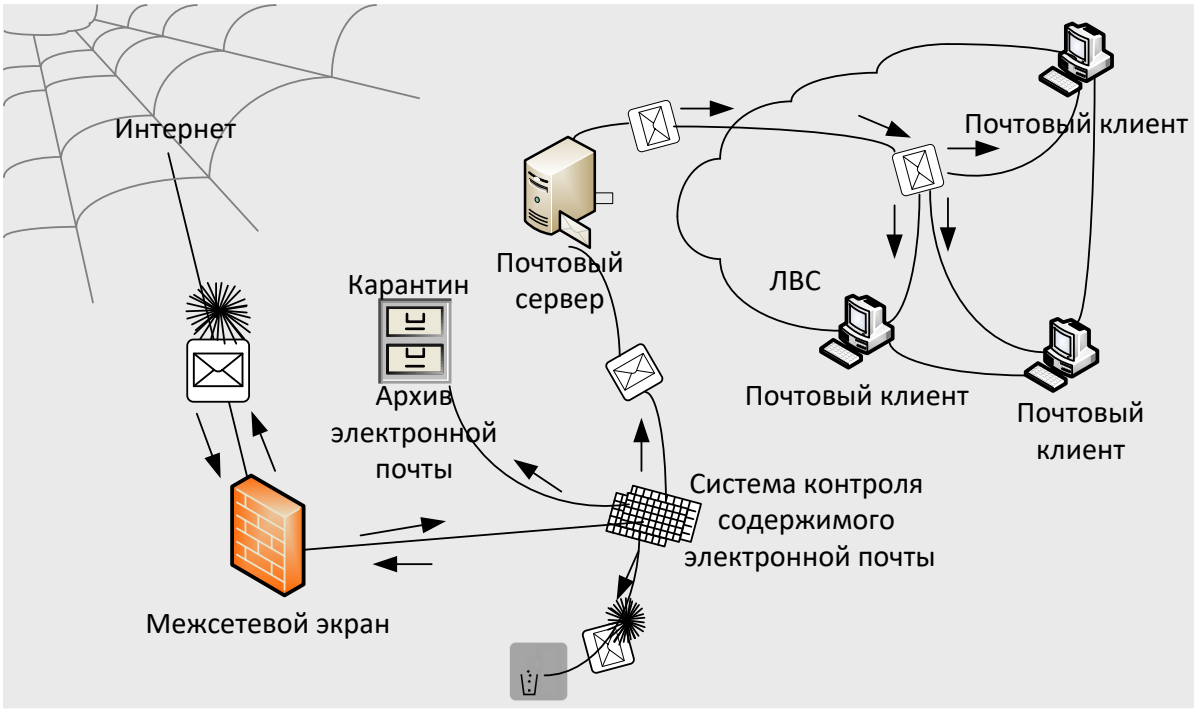


Рис. 18.14. Решение проблем защиты почтовой системы

Выполнение этих требований обеспечивается применением в средствах защиты определенных механизмов. К таким механизмам могут относиться:

1. рекурсивная декомпозиция (специальный алгоритм, применяемый для разбора сообщений электронной почты на составляющие компоненты с последующим анализом их содержимого);
2. эвристическое определение кодировок текстов;
3. определение типа файлов по сигнатуре;
4. полнотекстовый поиск по архиву электронной почты и т.п.

18.4.3. Политика использования электронной почты

Средство защиты – система контроля содержимого электронной почты, само по себе никаких задач по обеспечению безопасности не решает. Это всего лишь «машина», которая помогает человеку в решении этой проблемы. Поэтому задачу по обеспечению безопасности необходимо такой «машине» поставить. Это означает, что должен быть выработан специальный набор правил, который в дальнейшем будет переведен на язык машины. Этот набор правил называется «*политикой использования электронной почты*».

Во многих организациях такие правила существуют уже длительное время. Как и всякая ограничительная мера, они создают определенные неудобства пользователям системы, а если пользователю что-то неудобно, он либо перестает этим пользоваться, либо старается обойти препятствия. Поэтому такого рода политики, не подкрепленные техническими средствами контроля за их выполнением, постепенно теряют силу. Программные системы, ориентированные на фильтрацию почты, следует позиционировать именно как инструмент для внедрения и контроля исполнения этих правил.

Таким образом*, политика использования электронной почты* – это закрепленные в письменном виде и доведенные до сотрудников инструкции и другие документы, которые регламентируют их деятельность и процессы, связанные с использованием системы электронной почты. Эти документы и инструкции обладают юридическим статусом и, как правило, предоставляются для ознакомления сотрудникам организации.

Политика использования электронной почты является важнейшим элементом общекорпоративной политики информационной безопасности и неотделима от нее.

Политика должна соответствовать следующим критериям:

- быть лаконично изложенной и понятной всем сотрудникам компании, простота написания не должна привести к потере юридического статуса документа;
- исходить из необходимости защиты информации в процессе экономической деятельности компании;

- быть согласованной с другими организационными политиками компании (регламентирующими финансовую, экономическую, юридическую и другие сферы деятельности компании);
- иметь законную силу, т.е. политика, как документ, должна быть одобрена и подписана всеми должностными лицами руководящего звена компании, а ее выполнение должно быть детально продумано;
- не противоречить федеральным и местным законам;
- определять меры воздействия на сотрудников, нарушивших положения этой политики;
- соблюдать баланс между степенью защищенности информации и продуктивностью деятельности компании;
- детально определять мероприятия по обеспечению политики использования электронной почты в компании.

Политика использования электронной почты, как правило, рассматривается с двух сторон – как официально оформленный юридический документ и как материал, который описывает технику реализации политики.

Как документ политика должна включать:

- положение, что электронная почта является собственностью компании и может быть использована только в рабочих целях;
- указание на то, что применение корпоративной системы электронной почты не должно противоречить законодательству РФ и положениям политики безопасности;
- инструкции и рекомендации по использованию и хранению электронной почты;
- предупреждение о потенциальной ответственности сотрудников компании за злоупотребления при использовании электронной почты в личных целях и возможном использовании электронной почты в судебных и служебных разбирательствах;
- письменное подтверждение того, что сотрудники компании ознакомлены с политикой использования электронной почты и согласны с ее положениями.

С технической точки зрения политика устанавливает правила использования электронной почты, то есть определяет следующее.

Что контролируется	Прохождение каких сообщений входящей, исходящей или внутренней электронной почты должно быть разрешено или запрещено.
На кого распространяется	Категории лиц, которым разрешено или запрещено отправлять исходящие или получать входящие сообщения электронной почты.
Как реагирует система	Что необходимо делать с теми или иными сообщениями электронной почты, которые удовлетворяют или не удовлетворяют критериям, определенным правилами использования электронной почты.

18.4.4. Системы контроля содержимого электронной почты

Внедрение политики использования электронной почты требует от руководства компании понимания, что наличие только документально оформленной политики не гарантирует ее выполнения. Необходимо создание в компании соответствующих условий реализации этой политики. При этом важнейшим условием является наличие в корпоративной сети программно-технических средств контроля выполнения положений и требований политики. К таким средствам относятся системы контроля содержимого электронной почты.

Системы контроля содержимого электронной почты – это программное обеспечение, способное анализировать содержание письма по различным компонентам и структуре в целях реализации политики использования электронной почты.

К особенностям такого программного обеспечения относится:

1. применение при анализе содержания специально разработанной политики использования электронных писем;
2. способность осуществлять «рекурсивную декомпозицию» электронных писем;
3. возможность распознавания реальных форматов файлов вне зависимости от различных способов их маскировки (искажение расширения файлов, архивирование файлов и т.п.);
4. анализ множества параметров сообщения электронной почты;
5. ведение архива электронной почты;
6. анализ содержимого сообщения электронной почты и прикрепленных файлов на наличие запрещенных к использованию слов и выражений.

Системы контроля электронной почты помимо основной своей задачи мониторинга почтового трафика способны выполнять и другие функции. Практика показала, что в настоящее время такие системы используются в качестве:

1. средств управления почтовым потоком;
2. средств управления доступом;
3. средств администрирования и хранения электронной почты;
4. средств аудита контента (важнейшую функцию которого осуществляет архив электронной почты);
5. основы системы документооборота.

18.4.5. Требования к системам контроля содержимого электронной почты

Спектр возможностей всех категорий систем контроля содержимого электронной почты достаточно широк и существенно меняется в зависимости от производителя. Однако ко всем системам предъявляются наибо-

лее общие требования, которые позволяют решать задачи, связанные с контролем почтового трафика.

Основные требования, предъявляемые к таким системам – полнота и адекватность.

1. *Полнота* – это способность систем контроля обеспечить наиболее глубокую проверку сообщений электронной почты. Это предполагает, что фильтрация должна производиться по всем компонентам письма. При этом ни один из объектов, входящих в структуру электронного сообщения, не должен быть «оставлен без внимания». Условия проверки писем должны учитывать все проблемы, риски и угрозы, которые могут существовать в организации, использующей систему электронной почты.

2. *Адекватность* – это способность систем контроля содержимого как можно более полно воплощать словесно сформулированную политику использования электронной почты, иметь все необходимые средства реализации написанных людьми правил в понятные системе условия фильтрации.

К другим наиболее общим требованиям относятся следующее.

3. *Текстовый анализ электронной почты* – анализ ключевых слов и выражений с помощью встроенных словарей. Данная возможность позволяет обнаружить и своевременно предотвратить утечку конфиденциальной информации, установить наличие запрещенного содержания, остановить рассылку спама, а также передачу других материалов, запрещенных политикой безопасности. При этом качественный анализ текста должен предполагать морфологический анализ слов, то есть система должна иметь возможность генерировать и определять всевозможные грамматические конструкции слова. Эта функция приобретает большое значение в связи с особенностями русского языка, в котором слова имеют сложные грамматические конструкции.

4. *Контроль отправителей и получателей сообщений электронной почты.* Данная возможность позволяет фильтровать почтовый трафик, тем самым реализуя некоторые функции межсетевого экрана в почтовой системе.

5. *Разбор электронных писем на составляющие их компоненты* (MIME-заголовки, тело письма, прикрепленные файлы и т.п.), устранение «опасных» вложений и последующий сбор компонентов письма воедино, причем с возможностью добавлять к сообщению электронной почты необходимые для администраторов безопасности элементы (например, предупреждения о наличии вирусов или «запрещенного» текста в содержании письма).

6. *Блокировка или задержка сообщений большого размера* до того момента, когда канал связи будет менее всего загружен (например, в нерабочее время). Циркуляция в почтовой сети компании таких сообщений может привести к перегрузке сети, а блокировка или отложенная доставка позволит этого избежать.

7. *Распознавание графических, видео и звуковых файлов.* Как правило, такие файлы имеют большой размер, и их циркуляция может привести к потере производительности сетевых ресурсов. Поэтому способность распознавать и задерживать эти типы файлов позволяет предотвратить снижение эффективности работы компании.

8. *Обработка сжатых/архивных файлов.* Это дает возможность проверять сжатые файлы на содержание в них запрещенных материалов.

9. *Распознавание исполняемых файлов.* Как правило, такие файлы имеют большой размер и редко имеют отношение к коммерческой деятельности компании. Кроме того, исполняемые файлы являются основным источником заражения вирусами, передаваемыми с электронной почтой. Поэтому способность распознавать и задерживать эти типы файлов позволяет предотвратить снижение эффективности работы компании и избежать заражения системы.

10. *Контроль и блокирование спама.* Циркуляция спама приводит к перегрузке сети и потере рабочего времени сотрудников. Функция контроля и блокирования спама позволяет сберечь сетевые ресурсы и предотвратить снижение эффективности работы компании. Основными способами защиты от спама являются: проверка имен доменов и IP-адресов источников рассылки спама по спискам, запрос на указанный адрес отправителя (блокировка в случае отсутствия ответа), текстовый анализ спам-сообщения на наличие характерных слов и выражений в заголовках электронной почты (from/subject), проверка заголовков на соответствие спецификации RFC-822 и т.п.

11. *Способность определять число вложений в сообщениях электронной почты.* Пересылка электронного письма с большим количеством вложений может привести к перегрузке сети, поэтому контроль за соблюдением определенных политикой информационной безопасности ограничений на количество вложений обеспечивает сохранение ресурсов корпоративной сети.

12. *Контроль и блокирование программ-закладок* (cookies), вредоносного мобильного кода (Java, ActiveX, JavaScript, VBScript и т.д.), а также файлов, осуществляющих автоматическую рассылку (так называемые «Automatic Mail-to»). Эти виды вложений являются крайне опасными и приводят к утечке информации из корпоративной сети.

13. *Категоризация ресурсов почтовой системы* («административный», «отдел кадров», «финансы» и т.д.) и разграничение доступа сотрудников компании к различным категориям ресурсов сети (в т.ч. и в зависимости от времени суток).

14. *Реализация различных вариантов реагирования*, в том числе: удаление или временная блокировка сообщения; задержка сообщения и помещение его в карантин для последующего анализа; «лечение» зараженного вирусом файла; уведомление администратора безопасности или любого другого адресата о нарушении политики безопасности и т.п.

15. *Возможность модификации данных*, которая предусматривает, например, удаление неприемлемых вложений и замену их на тексты заданного содержания. Такая возможность позволит администратору удалять из писем прикрепленные файлы, тип которых запрещен политикой безопасности компании. К таким типам могут относиться исполняемые, видео и звуковые файлы, не имеющие отношения к деятельности компании. А это, в конечном итоге, позволит избежать заражения сети вирусами и добиться от сотрудников продуктивного использования почтового сервиса.

16. *Ведение полнофункционального архива электронной почты*, способного обеспечить хранение в режиме on-line большого количества электронной почты с высоким уровнем доступности данных. На основании хранящейся в архиве информации, возможно проводить дальнейший анализ почтового потока компании, корректировать работу системы, осуществлять анализ инцидентов, связанный с злоупотреблением сотрудниками компании почтовым сервисом и т.п.

На рис. 18.15 представлена последовательность работы типичной системы контроля содержимого электронной почты. Схема обработки сообщения, как правило, включает в себя следующие этапы: рекурсивная декомпозиция электронного письма; анализ содержимого электронного письма; «категоризация» электронного письма (отнесение к определенной категории); действие над письмом по результатам присвоения категории.

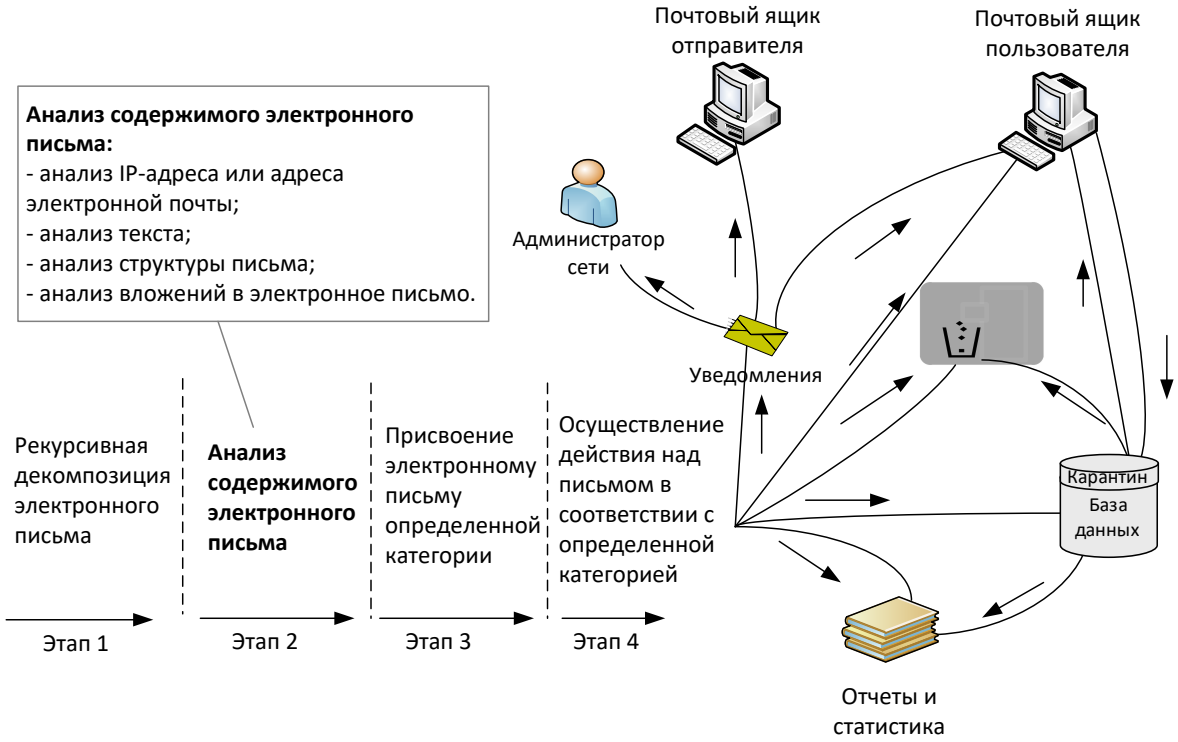


Рис. 18.15. Схема обработки сообщения системой контроля содержимого электронной почты

18.4.6. Принципы функционирования систем контроля содержимого электронной почты

Каждое попадающее в систему электронное письмо должно проверяться на соответствие заданным условиям. При этом, по меньшей мере, должны выполняться следующие условия отбора писем:

- условия на почтовые заголовки;
- условия на структуру письма (наличие, количество и структура вложений);
- условия на типы вложений (MS Office, исполняемые, архивы и т.п.);
- условия на содержимое (текст) писем и вложений;
- условия на результат обработки письма.

Кроме того, система должна позволять анализировать почтовые сообщения по всем их составляющим: атрибутам конверта, заголовкам сообщения, MIME-заголовкам, телу сообщения, присоединенным файлам.

18.4.6.1. Категоризация писем и фильтрация спама

Рассмотрим вопрос категоризации писем. Важно отметить, что гибкость при фильтрации почтовых сообщений особенно необходима, когда это касается такой проблемы, как спам. Одним из главных критериев выбора системы контроля содержимого электронной почты в настоящее время является как раз ее способность как можно более качественно справляться с этой проблемой.

Существует четыре основные методики определения, какое письмо относится к спаму, а какое нет.

1) Выявление спама по наличию в письме определенных признаков, таких как наличие ключевых слов или словосочетаний, характерное написание темы письма (например, все заглавные буквы и большое количество восклицательных знаков), а также специфическая адресная информация.

2) Определение адреса отправителя и его принадлежности к, так называемым, «черным спискам» почтовых серверов Open Relay Black List (ORBL). В эти списки заносятся те серверы, которые замечены в массовых рассылках спама, и идея состоит в том, чтобы вообще не принимать и не транслировать почту, исходящую с этих серверов.

3) Совместное использование методик по фильтрации по определенным признакам и проверкой «черного списка». По продуктивности мало чем отличается от двух первых. Результаты тестирования хорошо настроенного фильтра с применением обеих методик показывают, что из 100% спам-сообщений обнаруживается только 79,7%. При этом был выявлен значительный процент ложных срабатываний, а это значит, что к спаму были отнесены обычные письма (1,2% от задержанных писем), а это грозит для компании потерей важной информации. Некачественное разделение спама и обычных писем обусловлено, в том числе и некоторой «однобокостью» стандартных фильтров. При отбраковке писем учитываются «пло-

хие» признаки и не учитываются «хорошие», характерные для полезной переписки.

4) автоматическая настройка фильтров согласно особенностям индивидуальной переписки, а при обработке учет признаков как «плохих», так и «хороших» фильтров. Эта четвертая методика, предложенная американским программистом и предпринимателем Полом Грэмом. Методика основывается на теории вероятностей и использует для фильтрации спама статистический алгоритм Байеса. По имеющимся оценкам, этот метод борьбы со спамом является весьма эффективным. Так, в процессе испытания через фильтр были пропущены 8000 писем, половина из которых являлась спамом. В результате система не смогла распознать лишь 0,5% спам-сообщений, а количество ошибочных срабатываний фильтра оказалось нулевым.

Требование полного разбора письма при решении задачи категоризации следует дополнить требованием устойчивости.

- Во-первых, система должна быть устойчивой по отношению к обработке писем с некорректной структурой. Структура письма подчиняется определенным правилам. Разбор письма на составляющие основан на применении этих правил к конкретному письму. Возможны случаи, когда почтовая программа автора письма формирует письмо с нарушением этих правил. В этом случае письмо не может быть корректно разобрано.
- Во-вторых, система должна надежно определять типы файлов-вложений. Под «надежностью» имеется в виду определение, не основанное на имени файла, а также на информации, вписываемой в письмо почтовым клиентом при прикреплении файла (mime-type). Такая информация может быть недостоверна либо в результате сознательных попыток обмануть систему контроля, либо в результате неправильных настроек почтовой программы отправителя. Бессмысленно запрещать пересылку файлов типа JPEG, если файл picture.jpg после переименования в page.txt пройдет незамеченным.
- В-третьих, система должна обеспечивать полноту проводимых проверок, то есть высокое количество и разнообразие критериев анализа электронной почты. При этом система должна осуществлять фильтрацию по любым атрибутам сообщений, по объему сообщений и вложенных файлов, по количеству и типу вложений, по глубине вложенности, а также уметь анализировать содержимое прикрепленных файлов вне зависимости от того, являются ли эти файлы сжатыми или архивными. Существенным преимуществом многих продуктов является возможность создания собственного сценария обработки сообщений электронной почты.

При анализе текста нужно иметь возможность работать с нормализованными словоформами и т.д.

18.4.6.2. Реализация политики использования

Рассмотрим не отдельные правила, а все множество правил, составляющих политику. Любая реалистичная политика состоит из целого множества правил, которые, естественно, объединяются в группы. Очевидно, что правила для исходящей почты отличаются от правил для входящей, правила для руководства компании – от правил для рядовых сотрудников и т.д. Более того, поскольку правила применяются к письму в определенной последовательности, хотелось бы, чтобы эта последовательность была логичной и могла зависеть от результатов анализа письма. Все это вместе приводит к требованию «прозрачности»: правила, заданные в системе, должны «читаться» как правила, написанные на естественном языке, понятном человеку.

Все сказанное выше относилось к анализу письма. Однако сам по себе анализ ничего не дает. По его результатам письмо должно быть отнесено к какой-нибудь категории (безопасное, важное, неразрешенное и т.п.). Если такая категоризация проведена, то можно говорить о каких-либо действиях по отношению к проанализированному письму, например, доставить его адресату, заблокировать, и т.д. Другими словами, необходима возможность задавать системе правила, по которым она обрабатывает письма.

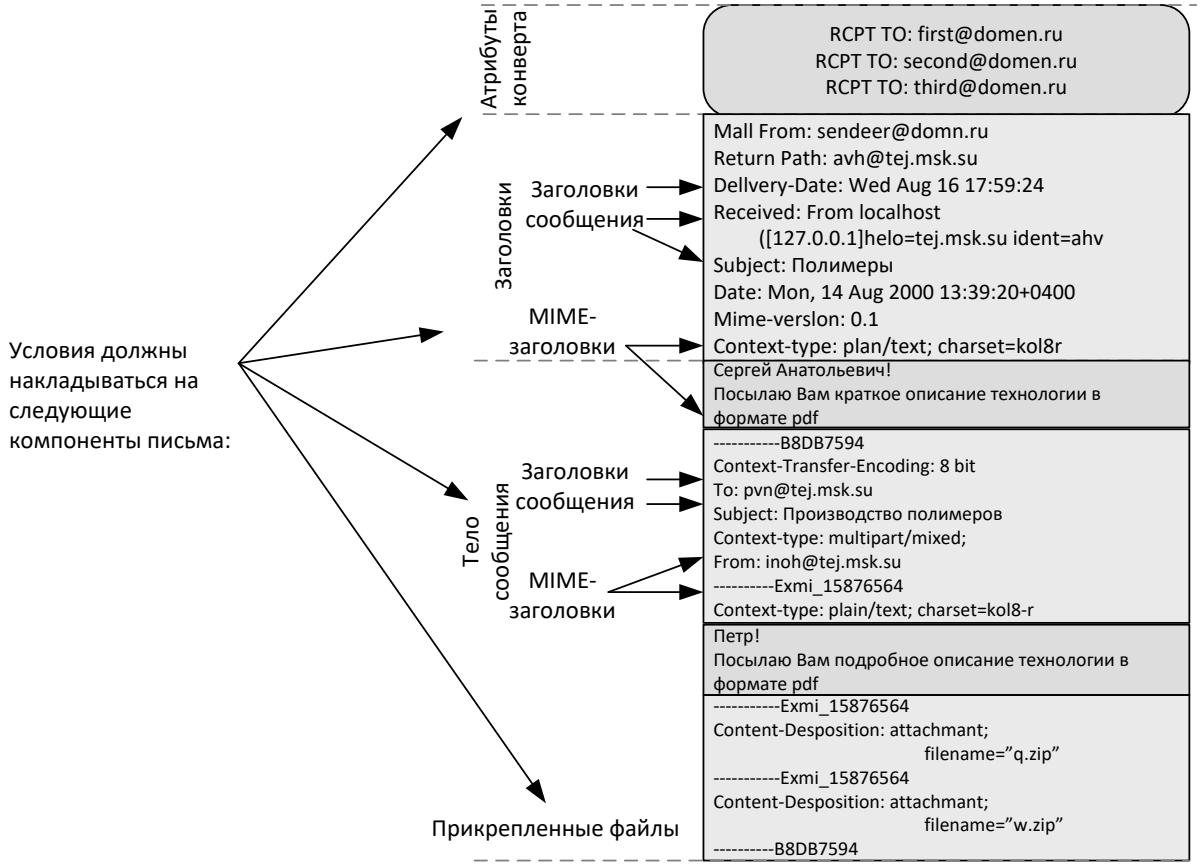


Рис. 18.16. Фильтрация по всем компонентам письма

Любое правило можно представить себе, как связку «условие + действие». Какие же действия нужны для того, чтобы обеспечить реализацию разумной политики?

Само по себе отнесение письма к определенной категории уже может рассматриваться как неявное действие. На этом действии следует остановиться подробнее. Дело в том, что жесткая категоризация как основа для принятия решений по электронному письму оказывается весьма непрактичной. Действительно, пусть мы выделили категорию писем «письмо, отправленное на запрещенный адрес» для того, чтобы блокировать доставку. С другой стороны, у нас может быть категория «письма руководства компании», которые надо отправлять безусловно. Что делать с письмом президента, отправленным по «запрещенному» адресу? Здравый смысл подсказывает, что приоритет должен быть отдан категории «письма руководства компании», что, безусловно, и будет сделано в системе с жесткой категоризацией. Однако будет потеряна существенная информация о письме. Разумный выход из таких ситуаций заключается в возможности относить письма сразу к нескольким категориям. Такая «свободная категоризация» позволит системе гибко реагировать на самые различные комбинации данных, содержащихся в письмах.

На рис. 18.17 показана схема действий, поддерживаемых правилами фильтрации почтовых сообщений типичной системы контроля содержимого электронной почты.

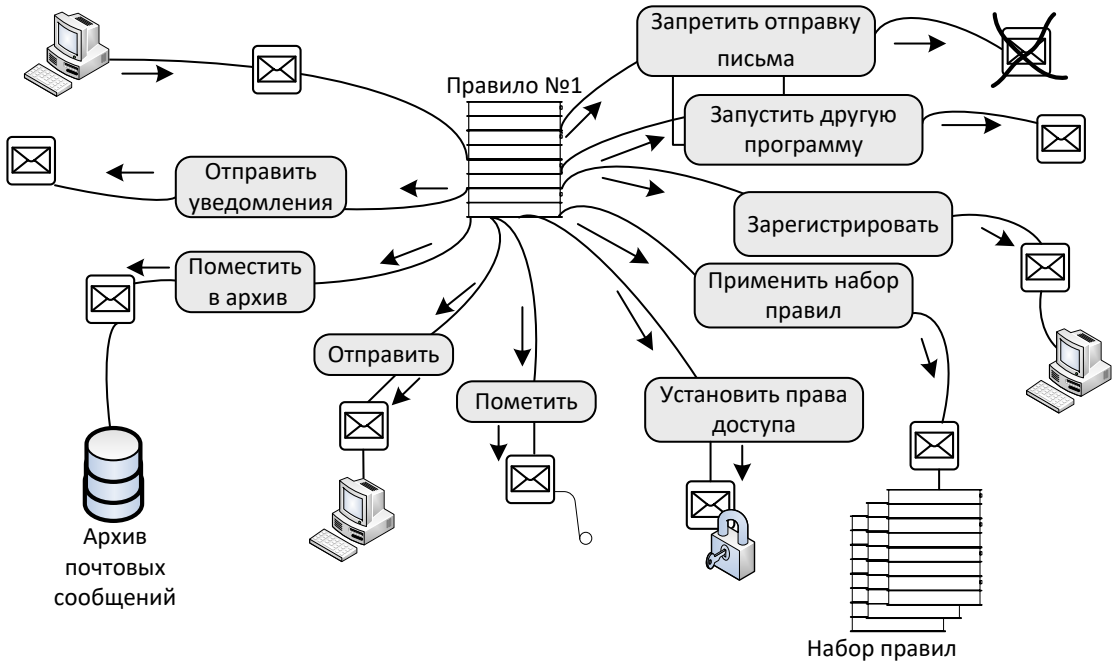


Рис. 18.17. Схема реагирования типичной системы контроля содержимого электронной почты

18.4.6.3. Долговременное хранение и архивирование

В последнее время большое значение для обеспечения безопасности информационных систем приобрело наличие в компании архива почтовых сообщений. Некоторые разработчики систем контекстного анализа предусматривают прикрепление к своим продуктам специальных модулей архивирования. Именно наличие архива электронной почты и определяет в настоящее время полнофункциональность продуктов этой категории. При этом ведение архива – это не просто автоматическая архивация почтовых сообщений в файл, а способность регистрации сообщений и учета необходимой информации на протяжении всего жизненного цикла сообщения, возможность получения любых выборок и статистики из архива по запросам, созданным с использованием любых критериев.

Кроме того, долговременный архив предоставляет возможность ретроспективного анализа почтовых потоков и не только позволяет найти виновных в нарушении принятых в компании правил по прошествии определенного времени, но и дает материал для построения объективной и обоснованной политики использования электронной почты.

18.4.6.4. Контекстный контроль содержимого

Отличительным признаком средств контекстного анализа является способность накопления статистики и генерации отчетов. Многие продукты имеют в своем арсенале только встроенные формы отчетов, другие способны осуществлять только просмотр статистики работы конкретного пользователя системы электронной почты.

Одним из основных критериев оценки систем контекстного анализа для российского рынка является поддержка продуктом различных кодировок кириллицы (CP1251, CP866, ISO8859-5, KOI8-R, MAC), что дает возможность анализа русскоязычных текстов. Кроме того, «проклятие» множественных кодировок тяготеет над российскими информационными системами. Все осложняется тем, что разные части письма, включая почтовые заголовки, могут быть написаны в разных кодировках. Вдобавок эти кодировки не всегда указаны или не всегда указаны верно.

Рассмотрим вопрос, касающийся архитектуры систем контроля содержимого электронной почты. В подобных продуктах уникальной особенностью является открытая архитектура, которая позволяет разработчикам расширять функциональные возможности системы, интегрируя в нее дополнительные модули и не затрагивая ее ядра. Это дает возможность постоянно наращивать способности системы контроля содержимого по защите электронной почты и одновременно с этим экономить значительные средства, которые могут потребоваться на модернизацию всей системы.

Заключение

Проблема обеспечения информационной безопасности телекоммуникационных систем становится все более актуальной для российских компаний. Это связано и с обострением конкурентной борьбы на внутренних рынках, и с выходом компаний на международный уровень. Многие из них уже не могут обеспечить защиту коммерческой информации собственными силами и вынуждены пользоваться услугами профильных профессиональных IT-консультантов. Особое значение имеет обеспечение информационной безопасности для критической инфраструктуры – ТКС специального и военного назначения.

В течение многих лет компании отчаянно боролись с вирусными эпидемиями, обносили периметр межсетевыми экранами и системами предотвращения вторжений, внедряли мощные инструменты против неавторизованного доступа. Однако компании упустили из вида главную опасность. Отсутствие единой политики информационной безопасности, а также единой концепции построения профиля информационной защиты компании зачастую обесценивает многомиллионные затраты на программные и аппаратные комплексы ИБ. Еще пару лет назад IT-службы отвечали за защиту от внешних угроз, а с внутренними угрозами разбиралась служба безопасности. Сегодня она просто физически не может контролировать перемещение информации по электронным сетям и с помощью переносных носителей. Для этого нужны специально разработанные регламенты, ликбез сотрудников, специально подготовленные сотрудники безопасности и технические средства для выявления попыток несанкционированного доступа или перемещения информации. Все эти меры должны реализоваться специалистом ИБ в рамках единой концепции.

Бурное развитие ИТ технологий (ОС, ПО и ТКС), а также направления ИБ приводит к росту спроса на профессиональных специалистов в этой области. Это актуализирует получение образования в области ИБ и широкой востребованности полученных профильных знаний на рынке труда.

Хочется надеется, что представленное учебное пособие поможет будущим профессионалам в сфере ИТ получить тот общий набор знаний и умений в области ОС, ТКС и ИБ, чтобы оказаться востребованными и высокооплачиваемыми сотрудниками престижных компаний.

Список сокращений

ACE – Access Control Entry – система контроля доступа;
ACL – Access Control List – список управления доступом;
AD – Active Directory – служба каталогов, являющаяся масштабируемой структурой домена, управляемого ОС Windows;
APEG – Automatic Patch–based Exploit Generation – технология автоматической генерации кода атаки по имеющейся коду заплатки к ПО;
API – Application Programming Interface – интерфейс прикладного программирования
ARP – Address Resolution Protocol – протокол разрешения адресов;
ARPANET – Advanced Research Projects Agency Network – глобальная сеть, которая являлась праобразом сети Internet;
CA – Certificate Authority – центральное бюро сертификатов;
CRC – Cyclic Redundancy Code – алгоритм вычисления контрольной суммы, предназначенный для проверки целостности передаваемых данных;
DACL – Discretionary Access Control List – список разрешений при управлении доступом;
DDF – Driver Device Interface – интерфейс «драйвер-устройство
DES – Data Encryption Standard – симметричный алгоритм шифрования;
DKI – Driver Kernel Interface – интерфейс «драйвер-ядро»
DLL – Dynamical Link Library – динамически подключаемая библиотека;
DMZ – Demilitarized Zone – демилитаризованная зона;
DNS – Domain Name System – система доменных имён;
DOS – Disks Operation System – дисковая операционная система
DRAM – –Dynamic Random Access Memory – динамическая оперативная память с произвольным доступом;
DS – Directory Services – Служба каталогов;
DSL – Digital Subscriber Line – цифровая абонентская линия;
EAP – Extensible Authentication Protocol – расширяемый протокол аутентификации;
EFS – Encrypting File System – шифрующая файловая система;
Ext – Extended (File System) – расширенная файловая система в операционных системах Unix/Linux;
FAT – File Allocation Table – файловая система, хранящая информацию о расположении файлов в таблицах;
FTP – File Transfer Protocol – протокол передачи файлов;
GC – Global Catalog – глобальный каталог в службе каталогов ОС Windows;
GID – Group ID – идентификатор группы пользователя;
GP – Group Policies – групповые политики в ОС Windows;

GSP – Generic Services Proxy – технология модуля доступа прикладного уровня для поддержки внешних протоколов обеспечения безопасности;
GUI – Graphical User Interface – графический интерфейс пользователя;
HTTP – HyperText Transfer Protocol – протокол передачи гипертекстовых Internet страниц;
HTTPS – Hypertext Transfer Protocol Secure – расширение протокола HTTP, поддерживающее шифрование;
ID – IDentification – идентификатор;
IDS – Intrusion Detection System – системы обнаружения вторжений;
IIS – Internet Information Services – сервер службы Web определяющий тип запрашиваемого ресурса;
IP – Internet Protocol Address – уникальный сетевой адрес узла в компьютерной сети;
IPX – Internet Packet eXchange – межсетевой обмен пакетами;
IRQ – Interrupt ReQuest – запрос на прерывание;
IRQL – Interrupt Request Levels – уровень запросов прерываний;
ISDN – Integrated Services Digital Network – цифровая сеть интегрального обслуживания;
ISP – Internet Service Provider – поставщик интернет–услуги (провайдер);
IT – Information Technology – информационные технологии;
LCN – Logical Cluster Number – логический номер кластера
LSA – Local Security Authority – локальный администратор безопасности используемый в ОС Windows;
MAC – Media Access Control – адрес, по которому ведется доступ абонентов к общему каналу связи на канальном уровне OSI;
MAC – Message Authentication Code – идентификационный код сообщения;
MFT – Master File Table – главная таблица файлов в файловой системе NTFS;
NCP – NetWare Control Protocol – основной протокол доступа к файлам и принтерам сетевой ОС NetWare компании Novell;
NCSC – National Computer Security Center – национальным центром компьютерной безопасности США;
NFS – Network File System – протокол сетевой файловой системы;
NIDS – Network Intrusion Detection System – сетевая система обнаружения вторжений;
NTFS – New Technology File System – файловая система, используемая в Windows 10;
OSI – Open System Interconnections – модель взаимодействия открытых систем;

OSPF – Open Shortest Path First – протокол динамической маршрутизации, основанный на технологии отслеживания состояния канала;
OU – – Organizacion Unit – организационная единица, описывающая тип объектов службы каталога ОС Windows;
PAE – процесс доступа через порт;
PGP – Pretty Good Privacy – протокол с открытым ключом для шифрования сообщений электронной почты;
PID – Process IDentifier – идентификатор процесса;
QoS – Quality of Service – качество обслуживания;
RPC – Remote Procedure Call – вызов удаленных процедур;
RSA – криптографический алгоритм с открытым ключом;
SACL – System Access Control List – системный список управления доступом;
SAM – Security Account Manager – RPC–сервер ОС Windows, оперирующий базой данных учетных записей, который управляет безопасностью пользователей;
SD – Security Descriptor – дескриптор безопасности;
SI – Standard Information – стандартная информация;
SID – Security IDentifier – идентификатор безопасности, используемый в ОС Windows;
SMB – Server Message Block;
SMP – Symmetric Multiprocessing – симметричная многопроцессорная архитектура;
SRAM – Synchronous Dynamic Random Access Memory – синхронная память с произвольным доступом;
SRM – Security Reference Monitor – Диспетчер доступа ОС Windows;
SSH – Secure SHell – сетевой протокол прикладного уровня «безопасная оболочка», позволяющий туннелировать TCP–соединения;
SSL – Secure Socket Layer – протокол защищенной связи через Интернет по системе «клиент – сервер»;
TCP – Transmission Control Protocol – протокол управления передачей;
TLS – Transport Layer Security – протокол обеспечения безопасности транспортного уровня;
UDP – User Datagram Protocol – протокол передачи пользовательских данных;
UID – User ID – идентификатор пользователя;
URL – Uniform Resource Locator – формат символьного указателя ресурса в сети Internet;
VCN – Virtual Cluster Number – виртуальный номер кластера;
VPN – Virtual Private Network – виртуальная частная сеть;
VFS – Virtual File System – виртуальная файловая система;

WEP – –Wired Equivalent Privacy – алгоритм для обеспечения безопасности сетей Wi–Fi;
Wi-Fi – беспроводная сеть стандарта IEEE 802.11;
WLAN – Wireless Local Area Network – беспроводная локальная сеть;
АБС – автоматизированная банковская система;
АНБ – агентство национальной безопасности;
АС – автоматизированная система;
АСОИ – автоматизированная система обработки информации;
ВК – виртуальный канал;
ВС – вычислительная система;
ВТ – виртуальный терминал;
ГВС – глобальная вычислительная сеть;
ЗУ – запоминающее устройство
ИБ – информационная безопасность;
ИС – информационная система;
ИТ – информационные технологии;
КС – компьютерная система;
ЛВС – локальная вычислительная сеть;
МСЭ – межсетевой экран;
МЭ – межсетевой экран;
НСД – несанкционированный доступ;
ОЗУ – оперативное запоминающие устройство;
ОК – общие критерии;
ОС – операционная система;
ОС – операционная система;
ПЗ – профиль защиты;
ПЗУ – постоянное запоминающие устройство;
ПИБ – политика информационной безопасности;
ПО – программное обеспечение
ПО – программное обеспечение;
РВС – распределенная вычислительная система;
РД – руководящий документ;
РПС – разрушающее программное средство;
СБИ – система безопасности информации;
СИУБ – система управления информационной безопасности;
УА – удаленная атака;
ФБР – федеральное бюро расследований;
ФС – файловая система;
ЦРУ – центральное разведывательное управление;
ЭВМ – электронно-вычислительная машина;
ЭЦП – электронная цифровая подпись.

Литература

1. Макаренко С. И., Коровин В. М. Принципы построения и функционирования аппаратно-программных средств телекоммуникационных систем: учеб. пособие. Часть 1: Принципы функционирования аппаратных средств телекоммуникационных и вычислительных систем. – СПб.: ВКА имени А. Ф. Можайского, 2014. – 197 с.

2. Макаренко С. И. Операционные системы, среды и оболочки: учебное пособие. – Ставрополь: СФ МГГУ им. М.А. Шолохова, 2008. – 210 с.

3. Макаренко С. И. Информационная безопасность: учебное пособие. – Ставрополь: СФ МГГУ им. М.А. Шолохова, 2009. – 372 с.

4. Макаренко С. И., Федосеев В. Е. Системы многоканальной связи. Вторичные сети и сети абонентского доступа. Учебное пособие. – СПб.: ВКА имени А.Ф. Можайского, 2014. – 179 с.

5. Макаренко С. И. Вычислительные системы, сети и телекоммуникации: учебное пособие. – Ставрополь: СФ МГГУ им. М.А. Шолохова, 2008. – 352 с.

6. Макаренко С. И., Сапожников В. И., Захаренко Г. И., Федосеев В. Е. Системы связи: учебное пособие для студентов (курсантов) вузов / Под ред. С.И. Макаренко. – Воронеж: ВАИУ, 2011. – 285 с.

7. Макаренко С. И. Аудит безопасности критической инфраструктуры специальными информационными воздействиями. Монография. – СПб.: Наукоемкие технологии, 2018. – 122 с.

8. Хомоненко А. Д. *и др.* Системы искусственного интеллекта: Практикум. – СПб: ВКА им. А.Ф. Можайского, 2014. – 69 с.

9. Хомоненко А. Д., Басыров А. Г., Бубнов В. П., Забродин А. В., Краснов С. А. *и др.* Модели и методы исследования информационных систем. Монография / под общ. ред. А.Д. Хомоненко. – СПб: Лань, 2019. – 204 с.

10. Бубнов В. П. *и др.* Модели информационных систем: учебное пособие. – М.: ФГБОУ «Учебно-методический центр по образованию на железнодорожном транспорте», 2015. – 188 с.

11. Макаренко С. И. Аудит информационной безопасности: основные этапы, концептуальные основы, классификация мероприятий // Системы управления, связи и безопасности. 2018. № 1. С. 1-29.

12. Антонович П. И., Макаренко С. И., Михайлов Р. Л., Ушанев К. В. Перспективные способы деструктивного воздействия на системы военного управления в едином информационном пространстве // Вестник академии военных наук. 2014. № 3 (48). С. 93-101.

13. Макаренко С. И., Михайлов Р. Л. Информационные конфликты - анализ работ и методологии исследования // Системы управления, связи и безопасности. 2016. № 3. С. 95-178.

14. Макаренко С. И. Информационное оружие в технической сфере: терминология, классификация, примеры // Системы управления, связи и безопасности. 2016. № 3. С. 292-376.

15. Макаренко С. И. Описательная модель сети связи специального назначения // Системы управления, связи и безопасности. 2017. № 2. С. 113-164.

16. Макаренко С. И. Перспективы и проблемные вопросы развития сетей связи специального назначения // Системы управления, связи и безопасности. 2017. № 2. С. 18-68.

17. Макаренко С. И. Эталонная модель взаимодействия стеганографических систем и обоснование на ее основе новых направлений развития теории стеганографии // Вопросы кибербезопасности. 2014. № 2 (3). С. 24-32.

18. Макаренко С. И. Проблемы и перспективы применения кибернетического оружия в современной сетецентрической войне // Спецтехника и связь. 2011. № 3. С. 41-47.

19. Краснов С. А. О возможности смыслового анализа информации для выявления информационных интересов пользователей // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление. 2019. № 2. С. 157-163.

20. Краснов С. А., Борисов А. А., Нечай А. А. Технология блокчейн и проблемы её применения в различных информационных системах // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление. 2018. № 2. С. 63-67.

21. Краснов С. А., Илатовский А. С., Хомоненко А. Д., Арсеньев В. Н. Оценка семантической близости документов на основе латентно-семантического анализа с автоматическим выбором ранговых значений // Труды СПИИРАН. 2017. № 5 (54). С. 185-204.

22. Хомоненко А. Д., Логашев С. В., Краснов С. А. Автоматическая рубрикация документов с помощью латентно-семантического анализа и алгоритма нечеткого вывода Мамдани // Труды СПИИРАН. 2016. № 1 (44). С. 5-19.

23. Краснов С. А., Хомоненко А. Д., Дашонок В. Л. Выявление противоречий в семантически близкой информации на основе латентно-семантического анализа // Проблемы информационной безопасности. Компьютерные системы. 2014. № 2. С. 73-84.

24. Войцеховский С. В., Калиниченко С. В., Краснов С. А., Уланов А. В. Модель оценивания оперативности обработки устаревающей информации // Научное обозрение. 2014. № 3. С. 155-157.

25. Краснов С. А., Уланов А. В., Матвеев С. В. Анализ оперативности обработки информации с ограниченным временем актуальности // Бюллетень результатов научных исследований. 2013. № 3 (8). С. 39-47.

26. Хомоненко А. Д., Краснов С. А., Еремин А. С. Оценка оперативности автоматической рубрикации документов с помощью модели нестационарной системы обслуживания с Эрланговским распределением длительности интервалов между запросами // Проблемы информационной безопасности. Компьютерные системы. 2012. № 3. С. 14-21.

27. Хомоненко А. Д., Бубнов В. П., Краснов С. А., Еремин А. С. Модель функционирования системы автоматической рубрикации документов в нестационарном режиме // Проблемы информационной безопасности. Компьютерные системы. 2011. № 4. С. 16-23.

28. Ковальский А. А. Организация адаптивного мультиплексирования трафика мультисервисных сетей в каналообразующей аппаратуре земных станций спутниковой связи с учетом изменяющейся помеховой обстановки // Системы управления, связи и безопасности. 2017. № 1. С. 175-212.

29. Зиннуров С. Х., Ковальский А. А., Ушанев К. В. Имитационная модель и алгоритм адаптивного резервирования канального ресурса космического аппарата связи при обслуживании нестационарного трафика с учетом задержки в управлении // Труды Военно-космической академии имени А.Ф. Можайского. 2019. № 666. С. 58-67.

30. Ковальский А. А., Митряев Г. А., Новиков Е. А., Алгоритм и методика оперативного планирования и распределения радиоресурса системы спутниковой связи для организации устойчивого управления орбитальной группировкой космических аппаратов // Труды Военно-космической академии имени А.Ф. Можайского. 2019. № 666. С. 68-76.

31. Зиннуров С. Х., Ковальский А. А., Митряев Г. А. Решение задачи оптимального планирования радиоресурса спутниковой системы связи для сеансов управления орбитальной группировкой космических аппаратов // Труды учебных заведений связи. 2018. Т. 4. № 1. С. 67-74.

32. Ковальский А. А. Методика и алгоритм оперативного планирования сеансов спутниковой связи на основе технологии гибких стратегий управления // Труды учебных заведений связи. 2018. Т. 4. № 4. С. 36-46.

33 Олифер В. Г., Олифер Н. А. Сетевые операционные системы: учебник. 3-е издание. – СПб.: Питер, 2009. – 672 с.

34. Таненбаум Э. С. Современные операционные системы. 2-е изд. – СПб.: Питер, 2005. – 1038 с.

35. Карпов В. Е., Коньков К. А. Основы операционных систем. – М.: Интернет-университет информационных технологий, 2005. – URL: www.INTUIT.ru (дата доступа 01.09.2018).

36. Дубаков А. А. Операционные системы: учебное пособие. – Томск: ТПУ, 1999. – 141 с.

37. Коньков К. А. Устройство и функционирование ОС Windows. – М.: БИНОМ. Лаборатория знаний. Интернет-университет информационных технологий, 2008. – URL: www.INTUIT.ru (дата доступа 1.09.2018).

38. Курячий Г. В. Операционная система Unix. – М.: Интернет-университет информационных технологий, 2004. – URL: www.INTUIT.ru (дата доступа 1.09.2018).

39. Курячий Г. В., Маслинский К. А. Операционная система Linux. – М.: Интернет-университет информационных технологий, 2005. – URL: www.INTUIT.ru (дата доступа 1.09.2018).

40. Операционная система реального времени QNX Neutrino 6.3. Системная архитектура: пер. с англ. – СПб.: БХВ-Петербург, 2006. – 336 с.

41. Крюков В. А. Операционные системы распределенных вычислительных систем (распределенные ОС): курс лекций. – М.: факультет ВМиК МГУ, 2008. – URL: parallel.ru/parallel/russia/people/krukov.html (дата доступа 01.09.2018).

42. Чернокнижный Г. М., Васильева И. Н. Операционные системы: учебное пособие. – СПб.: СПбГЭУ, 2018. – 176 с.

43. Куприянов А. И., Сахаров А. В., Шевцов В. А. Основы защиты информации: учебное пособие. – М.: Академия, 2006. – 256 с.

44. Марков А. С., Фадин А. А. Организационно-технические проблемы защиты от целевых вредоносных программ // Вопросы кибербезопасности. 2013. № 1 (1). С. 28-36.

45. Duqu: A Stuxnet-like malware found in the wild, technical report. Laboratory of Cryptography of Systems Security (CrySyS). – Budapest: Budapest University of Technology and Economics Department of Telecommunications, 2011. – 60 p. – URL: http://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf (дата обращения: 20.08.2016).

46. Вирус Regin // Security Lab [Электронный ресурс]. 28.05.2015. – URL: http://www.securitylab.ru/analytics/473080.php (дата обращения: 14.08.2016).

47. Киви Б. Кивино гнездо: государственный троянец // Компьютера [Электронный ресурс]. 21.10.2011. – URL: http://old.computerra.ru/own/kiwi/641530/ (дата обращения: 10.09.2017).

48. ГОСТ Р 51275-2006. Защита информации. Объект информатизации. Факторы, воздействующие на информацию. Общие положения. – М.: Стандартинформ, 2007. – 11 с. – URL: http://docs.cntd.ru/document/gost-r-51275-2006 (дата обращения: 14.08.2016).

49. «Лаборатория Касперского» раскрыла новый виток кампании кибершпионажа // PC Week [Электронный ресурс]. 04.07.2014. – URL: http://www.pcweek.ru/security/news-company/detail_print.php?ID=164797&print=Y (дата обращения: 04.07.2014).

50. Шабанов А. Программные закладки в бизнес-приложениях // Anti-Malware [Электронный ресурс]. 13.01.2011. – URL: http://www.anti-malware.ru/software_backdoors# (дата обращения: 14.08.2016).

51. Медведовский И. Д., Семьянов П. В., Платонов В. В. Атака через Интернет / Под ред. П.Д. Зегжды. – СПб.: Изд. НПО «Мир и семья-95», 1997.

52. Паршин С. А., Горбачев Ю. Е., Кожанов Ю. А. Кибервойны – реальная угроза национальной безопасности. – М.: КРАСАНД, 2011. – 96 с.

53. DoS-атака // Wikipedia [Электронный ресурс]. 19.05.2016. – URL: https://ru.wikipedia.org/wiki/DoS-%D0%B0%D1%82%D0%B0%D0%BA%D0%B0 (дата обращения: 19.05.2016).

54. WannaCry и всё о крупнейшей всемирной вирусной кибератаке всего Интернета // Avast Forum [Электронный ресурс]. 13.04.2017. – URL: https://forum.avast.com/index.php?topic=202335.0 (дата обращения: 13.04.2017).

55. Галатенко В. А. Основы информационной безопасности**. –** М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2008. – 208 с.

56. Астахов А. Анализ защищенности корпоративных автоматизированных систем // Jet Info [Эл. ресурс] – URL: www.jetinfo.ru\2002\7\1\article1.7.2002.html (дата доступа 01.09.2018).

57. Доля А. Внутренние угрозы ИТ-безопасности. // Byte-Россия [Эл. ресурс]. 2004. № 12. – URL: www.bytemag.ru/?ID=603365 (дата доступа 01.09.2018).

58. Доля А. Внутренние ИТ-угрозы в России 2006 // КомпьютерПресс. 2007. № 5.

59. Грудзаев С. Полезные мелочи - Aladdin Security Solution // LAN [Эл. ресурс] – URL: http://www.osp.ru/lan/2008/05/5068377/ (дата доступа 01.09.2018).

60. Романец Ю. В., Тимофеев П. А., Шаньгин В. Ф. Защита информации в компьютерных системах и сетях / Под ред. В.Ф. Шаньгина. – М.: Радио и связь, 2001. – 376 с.

61. Галатенко В. А. Стандарты информационной безопасности. – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2005. – 264 с.

62. Lonely R. Алгоритм шифрования данных с открытым ключом RSA. [Эл. ресурс] – URL: www.rusdoc.ru/material/raznoe/rsa.shtml (дата доступа 01.09.2018).

63. Лапонина О. Р. Криптографические основы безопасности: курс лекций для Интернет-университета информационных технологий. – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2005. – URL: https://www.intuit.ru/studies/courses/28/28/info (дата доступа 01.09.2018).

64. Антивирусная защита компьютерных систем: курс лекций для Интернет-университета информационных технологий от лаборатории Касперского – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2007. – URL: www.intuit.ru/department/security/antiviruskasp/ (дата доступа 01.09.2018).

65. Вирусы и средства борьбы с ними: курс лекций для Интернет-университета информационных технологий от лаборатории Касперского – М.: Интернет-университет информационных технологий - www.INTUIT.ru, 2007. – URL: www.intuit.ru/department/security/viruskasper/ (дата доступа 01.09.2018).

66. Медведовский И. Д., Семьянов П. В., Платонов В. В. Атака через Интернет / под ред. П.Д. Зегжды. – СПб.: изд. НПО «Мир и семья-95», 1997.

67. Мэйволд Э. Безопасность сетей: курс лекций для Интернет-университета информационных технологий – М.: Интернет-университет

информационных технологий – www.INTUIT.ru, 2006. – URL: www.intuit.ru/department/security/netsec/ (дата доступа 01.09.2018).

68. Кобб М. Джост М. Безопасность IIS: курс лекций для Интернет-университета информационных технологий – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2006. – URL: www.intuit.ru/department/internet/iissecurity/ (дата доступа 01.09.2018).

69. Бейс Р. Введение в обнаружение атак и анализ защищенности // НИП «Информзащита» [Эл. ресурс]. – URL: http://bugtraq.ru/library/books/icsa/ (дата доступа 01.09.2018).

70. Семенов Ю. А. Процедуры, диагностики и безопасность в Интернет: курс лекций для Интернет-университета информационных технологий – М.: Интернет-университет информационных технологий - www.INTUIT.ru, 2007. – URL: www.intuit.ru/department/network/pdsi/ (дата доступа 01.09.2018).

71. Пировских А. Взлом WPA // TNG.ru [Эл. ресурс]. – URL: www.thg.ru/network/20050806/print.html (дата доступа 01.09.2018).

72. Таранов А., Слепов О. Безопасность систем электронной почты // Jet Info. 2003. № 6. – URL: www.citforum.ru/security/internet/email/article1.6.2003.html#AEN11 (дата доступа 01.09.2018).

73. Иржавский А. Безопасность электронной почты // CIO. 2003. № 8. – URL: offline.cio-world.ru/2003/18/29383/index.html (дата доступа 01.09.2018).

74. Карпов В. Е., Коньков К. А. Основы операционных систем. – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2005. – 536 с.

75. Коньков К. А. Устройство и функционирование ОС Windows. – М.: Интернет-университет информационных технологий – ИНТУИТ.ру, БИНОМ. Лаборатория знаний, 2008. – 208 с.

76. Коньков К. А. Основы организации операционных систем Microsoft Windows: курс лекций для Интернет-университета информационных технологий – М.: Интернет-университет информационных технологий – www.INTUIT.ru, 2007. – URL: www.intuit.ru/department/os/osmswin/ (дата доступа 01.09.2018).

77. Казарин О. В. Безопасность программного обеспечения компьютерных систем. – М.: МГУЛ, 2003. – 212 с. – URL: www.citforum.ru/security/articles/kazarin/#1-5 (дата доступа 01.09.2018).

78. Малюк А. А. Информационная безопасность: концептуальные и методологические основы защиты информации. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2004. – 280 с.

79. Белов Е. Б., Лось В. П., Мещериков Р. В., Шелупанов А. А. Основы информационной безопасности. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2006. – 544 с.

80. Грушо А. А. Тимонина Е. Е. Теоретические основы защиты информации. – М.: Изд. агентства «Яхтсмен», 1996. – 72 с.

81. Хорошко В. А., Чекатков А. А. Методы и средства защиты информации. – М.: Юниор, 2003. – 504 с.

82. Ярочкин В. И. Информационная безопасность. Учебное пособие для студентов непрофильных вузов. – М.: Междунар. отношения, 2000. – 400 с.

83. Корнюшин П. Н. Костерин С. С. Информационная безопасность. – Владивосток: ДВГУ, 2003. – 155 с.

84. Халяпин Д. Б. Защита информации. Вас подслушивают? Защищайтесь! – М.: НОУ ШО «Баярд», 2004. – 432 с.

85. Щеглов А. Ю. Защита компьютерной информации от несанкционированного доступа. – СПб.: Наука и Техника, 2004. – 384 с.

86. Шнайер Б. Секреты и ложь. Безопасность данных в цифровом мире. – СПб.: Питер, 2003. – 368 с.

87. Шнайер Б. Прикладная криптография: Протоколы, алгоритмы, исходные тексты на языке Си. 2-е издание. – М.: Триумф, 2002. – 595 с.

88. Ященко В. В. и др. Ведение в криптографию // CIT Forum [Эл. ресурс]. – URL: www.citforum.ru\security\cryptography\yaschenko\ (дата доступа 01.09.2018).

89. Будко В. Н. Информационная безопасность и защита информации. Конспект лекций. – Воронеж: ВГУ, 2003. – 86 с.

90. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии. – М.: МЦНМО, 2003. – 328 с.

91. Биккенин Р. Р. Стеганография – современный метод обеспечения безопасности информации // Информация и космос. 2006. № 2. С. 89-93.

92. Девянин П. Н. Модели безопасности компьютерных систем: учебное пособие для студентов высш. учеб. заведений. – М.: Академия, 2005. – 144 с.

93. Левин М. Библия хакера 2. Кн. 1. – М.: Майор, 2003. – 640 с.

94. Левин М. Библия хакера 2. Кн. 2. – М.: Майор, 2003. – 688 с.

95. Владимиров А. А., Гавриленко К. В., Михайловский А. А. Wi-фу: боевые приемы взлома и защиты беспроводных сетей. – М.: НТ Пресс, 2005. – 463 с.

96. Белоусов С. А., Гуц А. К., Планков М. С. Троянские кони. Принципы работы и методы защиты: учебное пособие. – Омск: изд. Наследие. Диалог-Сибирь, 2003. – 84 с.

97. Аналитический бюллетень Secure List – URL: http://www.securelist.com/ru/ (дата доступа 01.09.2018).

98. Электронный учебник по разработке информационной безопасности компьютеров // Help Antivirus – URL: http://help-antivirus.ru/developmentsafety/Menu.php (дата доступа 01.09.2018).

99. Хогланд Г., Мак-Гроу Г. Взлом программного обечерспечения: анализ и использование кода. – М.: Издательский дом «Вильямс», 2005. – 400 с.

100. Базовая модель угроз безопасности персональных данных при их обработке в информационных системах персональных данных. – М.: ФСТЭК России, 2008. – 69 с.

Макаренко Сергей Иванович
Ковальский Александр Александрович
Краснов Сергей Александрович

Принципы построения и функционирования аппаратно-программных средств телекоммуникационных систем

Часть 2
Сетевые операционные системы
и принципы обеспечения информационной безопасности в сетях

Учебное пособие

Издательство «Наукоемкие технологии»
ООО «Корпорация «Интел групп»
197372, Санкт-Петербург,
пр. Богатырский, дом 32, к. 1 лит. А, пом. 6Н.
http://publishing.intelgr.com
Тел.: +7 (812) 945-50-63
E-mail: publishing@intelgr.com

Отпечатано:
Типография «Скифия-Принт»
Санкт-Петербург, ул. Б. Пушкарская, д.10.
http://www.skifia-print.ru
Тел.: +7 (812) 644-41-63
E-mail: skifia-print@mail.ru

ISBN 978-5-6044429-8-2

Гарнитура «TimesNewRoman». 17,15 п.л.
Тираж 600 экз. Подписано в печать 20.06.2020.
Материалы изданы в авторской редакции