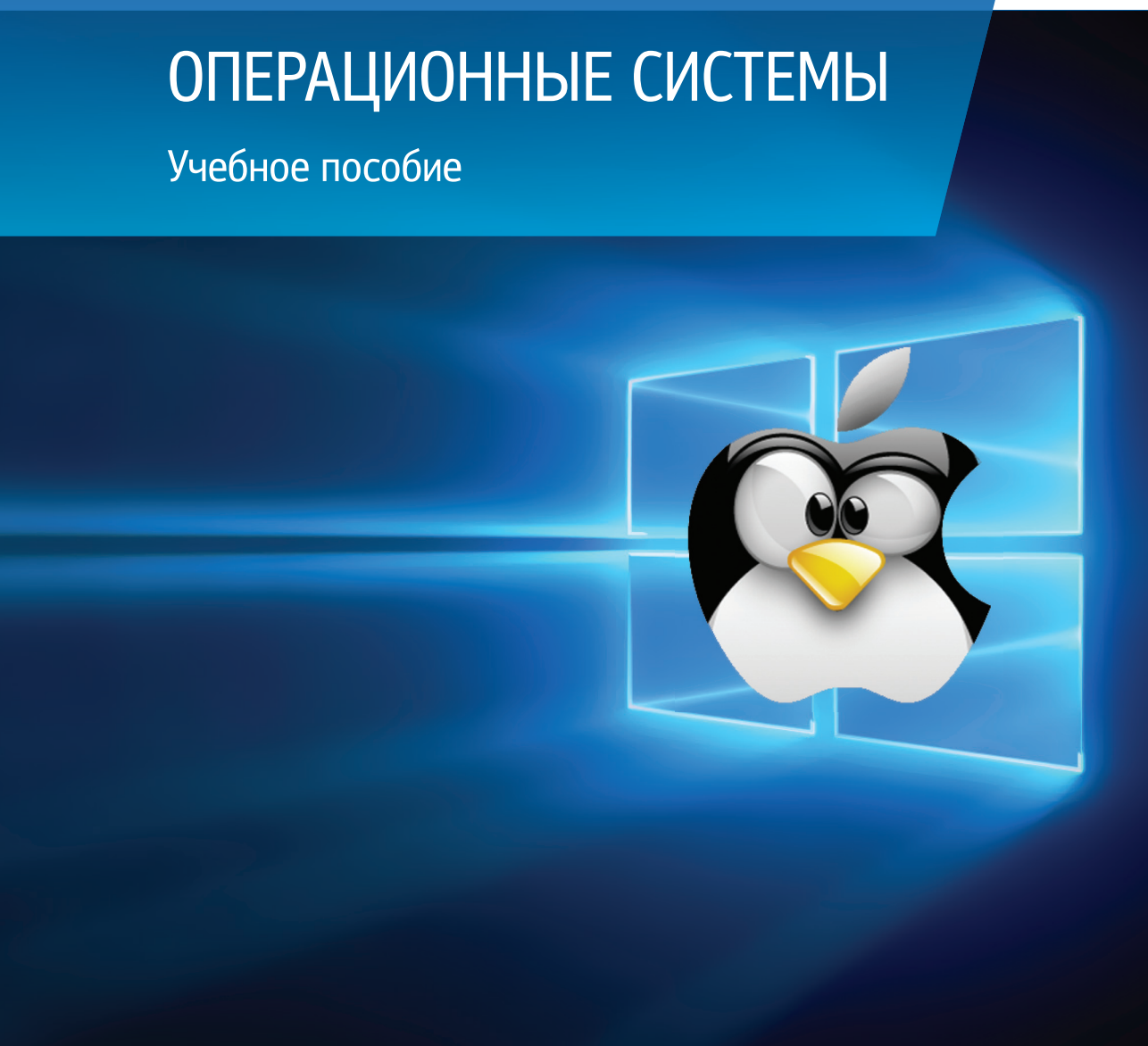
Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

О. М. ЗВЕРЕВА

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие



Министерство науки и высшего образования
Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

О. М. Зверева

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

Рекомендовано методическим советом
Уральского федерального университета для студентов вуза,
обучающихся по направлениям подготовки:
09.03.01 — Информатика и вычислительная техника;
09.03.04 — Программная инженерия

Екатеринбург
Издательство Уральского университета
2020

УДК 004.451(075.8)
ББК 32.973-018.2я73
З-43

Рецензенты:
ученый совет института инженерно-педагогического образования РГППУ (директор Е. В. Чубаркова);
д-р техн. наук, проф. В. Г. Лабунец (Урал. гос. лесотехн. ун-т)

Научный редактор — д-р техн. наук, проф. Л. Г. Доросинский

Зверева, О. М.
З-43 Операционные системы : учебное пособие / О. М. Зверева ; Мин-во науки и высш. образ. РФ. — Екатеринбург : Изд-во урал. ун-та, 2020. — 220 с.

ISBN 978-5-7996-3146-8

Учебное пособие предназначено для подготовки бакалавров. Основной целью пособия является развитие компетенций студентов в области общих принципов построения и функционирования операционных систем — того класса программного обеспечения, без которого компьютер неработоспособен. Материал снабжен примерами из современных версий систем, которые призваны подтвердить действенность теоретических положений. В конце каждой главы есть перечень контрольных вопросов для проверки степени усвоения прочитанного.

Библиогр.: 54 назв. Табл. 5. Рис. 70.

УДК 004.451(075.8)
ББК 32.973-018.2я73

ISBN 978-5-7996-3146-8

Оглавление

Предисловие

В данной книге представлен теоретический материал по основам построения и функционирования операционных систем — того класса программного обеспечения, благодаря которому то, что называют «железом», становится не просто набором железных и пластмассовых компонентов, а устройством, способным решать сложнейшие задачи. Основная цель издания не научить читателей устанавливать различные переключатели и кнопки, находя их в определенных окнах, а научиться понимать, как функционирует компьютер, почему и зачем нужно устанавливать те или иные параметры, чтобы компьютер работал и работал эффективно.

Автор проиллюстрировала теоретический материал примерами из современных версий операционных систем настольных компьютеров (Windows 10 и Ubuntu (Linux)) для подтверждения того, что большинство описанных концепций не просто теоретические построения, а реальные основы технологий и алгоритмов, реализованных в работающих системах.

Материал построен следующим образом: сначала вводная часть, описывающая историю развития операционных систем и описание состояния дел на рынке этого ПО, далее следуют главы, посвященные основным подсистемам, существующим в любой операционной системе. В конце каждой главы есть контрольные вопросы для тестирования степени понимая и усвоения прочитанного материала.

1. Понятие операционной системы. Появление, развитие и особенности современного состояния ОС

Определение понятия «операционная система»

Все многообразное ПО, установленное на персональном компьютере, можно отнести к одному из 3-х классов:

1) системное ПО — характеризуется тем, что без него ПК неработоспособен. Его в свою очередь можно условно разделить на 3 подкласса:
   - операционные системы;
   - отдельные системные программы, не входящие в состав ОС, — программы работы с диском, архиваторы, антивирусные программы и т. д.;
   - системное ПО для организации работы прикладного ПО, написанного на языке высокого уровня, — компиляторы, интерпретаторы, загрузчики, редакторы связей, системные библиотеки и т. д.;
2) прикладное ПО — характеризуется тем, что крайне разнообразно, сложно найти для него единую классификацию [1, 2];
3) инструментальное ПО — характеризуется тем, что предоставляет инструментарий для создания других двух классов.

Определив место ОС в составе всего ПО, дадим определение этого понятия, базируясь на понимании того, какие основные функции должна выполнять ОС. **ОС** является центральной составляющей системного ПО, основная задача которого — обеспечить работоспособность компьютера. Следует также учесть, что персональный компьютер подразумевает взаимодействие с пользователем, т. е. ОС должна выполнять интерфейсную функцию между аппаратурой компьютера, с одной стороны, и пользователем, т. е. его приложениями — с другой (рис. 1.1).

Рис. 1.1. Место операционной системы в компьютерной системе

Операционная система — это программно-аппаратный комплекс, который выполняет 2 основные функции:

1) эффективное управление ресурсами компьютера;
2) создание удобного интерфейса между пользователем и компьютером.

Следует отметить, что ОС не просто комплекс взаимосвязанных программ, а программно-аппаратный комплекс, потому что в выполнение части функций аппаратура вовлечена непосредственно, примером может служить система прерываний или аппаратная защита областей памяти.

Управление ресурсами состоит в решении следующих общих для всех типов ресурсов задач:

1) планировании ресурса — определение момента времени и объема выделяемого ресурса;
2) удовлетворении запросов на ресурсы — выделение ресурса в запланированном объеме и в запланированный момент времени;
3) отслеживании уровня и учете использования ресурса — поддержание информации о ресурсе в оперативном состоянии;
4) разрешении конфликтов — разрешение конфликтов при попытке одновременного доступа к ресурсу.

Создание удобного интерфейса — интерфейсная функция — предполагает, что вместо реальной аппаратуры компьютера, ОС представляет пользователю расширенную виртуальную машину, с которой удобнее работать и которой можно управлять определенным способом.

Операционная система образует ту программную среду, в которой выполняются прикладные программы пользователей. Такая среда называется операционной; это следует понимать в том плане, что при запуске программы она будет обращаться к операционной системе с со-

ответствующими запросами на выполнение определенных действий, или функций [3].

История появления операционных систем

История любой отрасли науки или техники позволяет не просто удовлетворить любопытство, но и дает основания для более глубокого понимания существующих тенденций, оценки перспективности тех или иных тенденций развития.

Исходя из того факта, что ОС — это та часть ПО, которая наиболее тесно связана с аппаратной частью, можно утверждать, что история ОС непосредственно связана с историей вычислительной техники вообще, и рассматривать историю ОС следует в контексте истории развития аппаратуры вычислительных систем.

В истории вычислительных машин (ВМ) обычно выделяют 4 (иногда 5) поколений вычислительных машин. Деление на поколения производят на основе определения элементной базы этих машин: лампы, полупроводниковые элементы, интегральные схемы (ИС), большие и сверхбольшие ИС. К пятому поколению иногда относят мобильные устройства (несколько отходя тем самым от принятого деления).

Первое поколение ЭВМ. Отсутствие операционных систем

Настоящее рождение ЭВМ произошло вскоре после окончания Второй мировой войны, в середине 1940-х гг. были созданы первые ламповые вычислительные устройства. Профессор Д. Атанасов и его аспирант Клиффорд создали в университете штата Айовы конструкцию, которая сейчас считается первым действующим цифровым компьютером. В ней использовалось 300 электронных ламп. Примерно в то же время Конрад Цузе в Берлине построил Z3 компьютер, основанный на использовании электромеханических реле.

Отличительные черты этого периода:

1) одна и та же группа людей участвовала в проектировании, эксплуатации и программировании вычислительной машины — не было разделения труда в этой области деятельности, была единая научно-исследовательская работа в области вычислительной техники;

2) ЭВМ не использовались в качестве инструмента решения каких-либо практических задач из других прикладных областей;
3) программирование осуществлялось исключительно на машинном языке;
4) операционные системы все еще не появились, все задачи организации вычислительного процесса решались вручную.

Второе поколение ЭВМ. Появление первых операционных систем

Шестидесятые годы XX в. ознаменовали новый этап в развитии вычислительных устройств, он был связан с появлением новой элементной базы — полупроводников. Появились устройства, названные мэйнфреймами, они занимали большие площади, требовали кондиционирования воздуха и наличия большого штата работников, в основном профессиональных операторов.

Отличительные черты данного периода:

1) ЭВМ стали более надежными, они могли непрерывно работать настолько долго, чтобы на них можно было возложить выполнение реально востребованных практических задач;
2) появились первые алгоритмические языки (языки высокого уровня), такие как АЛГОЛ, ФОРТРАН, КОБОЛ. Выполнение каждой программы стало состоять из нескольких этапов — поиска и запуска на выполнение нужного транслятора, подключения нужных библиотек, получения программы в машинных кодах, загрузки программы в основную память с нужных адресов, запуска на выполнение программы, вывода результатов на внешнее устройство. Был дан значительный толчок развитию системного программного обеспечения;
3) появилось разделение труда, потребовались профессионалы, обслуживающие оборудование, и те, кто профессионально выполнял работу по организации вычислительного процесса для всех пользователей вычислительного центра;
4) большую часть времени процессор простаивал в ожидании, пока оператор запустит очередную задачу — процессорное время использовалось неэффективно;
5) были разработаны первые ОС — программы, которые занимались организацией вычислительного процесса.

Существует несколько версий того, какую ОС считать первой. По одним источникам это ОС для ЭВМ IBM 701, созданная в компании General Motors Research Laboratories в начале 1950-х гг.; по другим — это ОС для ЭВМ MARK1, созданная в Университете Стэнфорда в то же время.

Первый тип реально существовавших в ЭВМ ОС — это системы пакетной обработки. Сущность таких систем заключалось в том, что несколько задач составлялись в пакет, что позволяло эффективнее использовать дорогое процессорное время. В ходе реализации систем пакетной обработки был разработан специальный язык управления заданиями, типовой набор директив которого включал признак начала или конца отдельной работы, вызов транслятора, вызов загрузчика, признаки начала или конца исходных данных.

Оператор составлял специальный пакет из имевшихся заданий, этот пакет запускался далее под управлением специальной системной программы — монитора. Монитор мог без участия оператора обрабатывать ошибочные ситуации, которые возникали при работе пользовательских программ, такие как отсутствие или несоответствие форматов входных данных, деление на ноль, обращение по запрещенным или несуществующим адресам в памяти и т. д.

В качестве наиболее известных программ мониторов можно назвать монитор IBSYS, разработанный в 1960 г. для IBM 7090, и монитор для ЭВМ БЭСМ-4, датированный 1962 г.

Временная диаграмма вычислительного процесса до появления систем пакетной обработки представлена на рис. 1.2, *а*, где показано подготовительное время задачи (загрузка задачи в основную память, настройка адресов и т. д.) и заключительное время (выгрузка содержимого основной памяти после счета), необходимые для выполнения очередной задачи. Временная диаграмма в системе с ОС пакетной обработки показана на рис. 1.2, *б*. Хотя здесь также существует подготовительное и заключительное время, но они для всего пакета задач, и хорошо видно, что выполняется условие

$$\sum_{i=1}^{N} П_i + \sum_{i=1}^{M} З_i > П_П + З_П. \qquad (1.1)$$

Из выражения (1.1) очевидно следует, что подготовительное и заключительное время (а значит, и общее время) выполнения пакета задач в системах с пакетной ОС значительно меньше, чем сумма подго-

товительных и заключительных времен выполнения задач в системе, где нет пакетов задач.

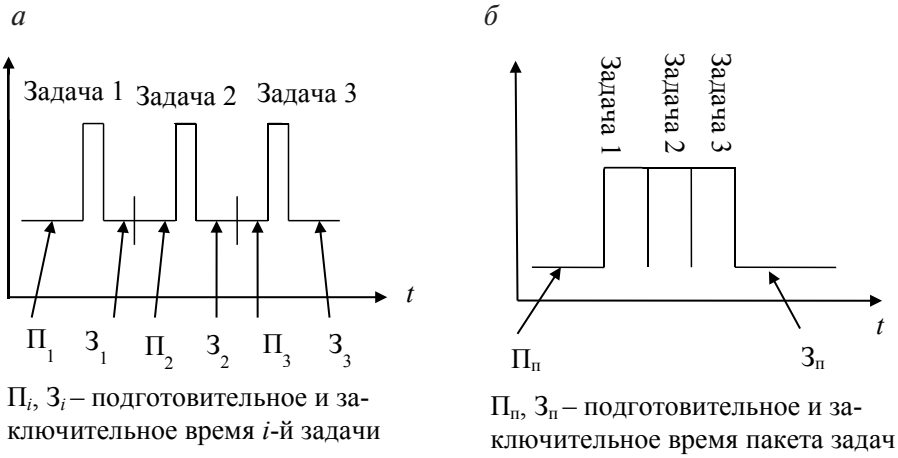


Рис. 1.2. Временная диаграмма вычислительного процесса:
а — до ОС пакетной обработки; *б* — с ОС пакетной обработки

К достоинствам систем пакетной обработки можно отнести то, что они:

1) значительно сократили затраты времени на вспомогательные действия по организации вычислительного процесса;
2) позволяли эффективно использовать машинное время.

Недостатком систем пакетной обработки можно считать то, что пользователи лишились непосредственного доступа к ЭВМ, что не могло не сказаться на эффективности работы — внесение любого исправления требовало значительно больше времени, чем при интерактивной работе за пультом машины.

Третье поколение ЭВМ: мультипрограммирование и другие передовые концепции

Следующий важный период развития ОС относится к 1965–1975 гг. В этот период произошла смена элементной базы с полупроводниковых элементов на интегральные микросхемы, а это в свою очередь привело к смене поколения вычислительных устройств. Новые функциональные возможности интегральных микросхем обеспечили возможность создания сложных машинных архитектур, таких как IBM/360.

В этот период были реализованы практически все основные механизмы, присущие современным ОС:

1) мультипрограммирование (многозадачность);
2) мультипроцессирование;
3) поддержка многотерминального многопользовательского режима;
4) виртуальная память;
5) файловые системы;
6) разграничение доступа и сетевая работа.

Реализацию мультипрограммирования можно считать одним из основных событий данного этапа. Вычислительные машины становятся достаточно мощными, могут решать практические задачи, поэтому неэффективно их использовать для обработки и хранения данных только одной программы в каждый момент времени. Решением этой задачи является мультипрограммирование. **Мультипрограммирование** — это способ организации вычислительного процесса, при котором в памяти компьютера находится одновременно несколько программ, которые могут попеременно выполняться на одном процессоре [4].

Мультипрограммирование было реализовано в 2-х версиях: в системах пакетной обработки и в системах разделения времени.

Основным критерием эффективности мультипрограммных систем пакетной обработки, как и для однопрограммных систем, являлась максимальная загрузка процессора, однако эта задача решалась более эффективно. В мультипрограммном режиме процессор не простаивал, пока одна из задач не общалась с ним, например, выполняя операцию ввода-вывода (как это было при последовательном выполнении задач в однопрограммных системах), а переключался на другую задачу, которая была готова к выполнению. В результате достигалась максимальная загрузка всех устройств и увеличивалось число задач, решаемых в единицу времени.

Однако, недостаток, присущий этим системам, — отсутствие возможности интерактивного взаимодействия с ЭВМ — преодолен не был. Этот недостаток был ликвидирован ОС другого типа — ОС разделения времени. Первоначально этот вариант был рассчитан на многотерминальные системы, когда каждый пользователь работает за своим терминалом. Мультипрограммирование в системах разделения времени создавало для каждого пользователя иллюзию единоличной ра-

боты с данным вычислительным устройством за счет периодического выделения каждой готовой к выполнению программе определенного кванта процессорного времени (рис. 1.3).

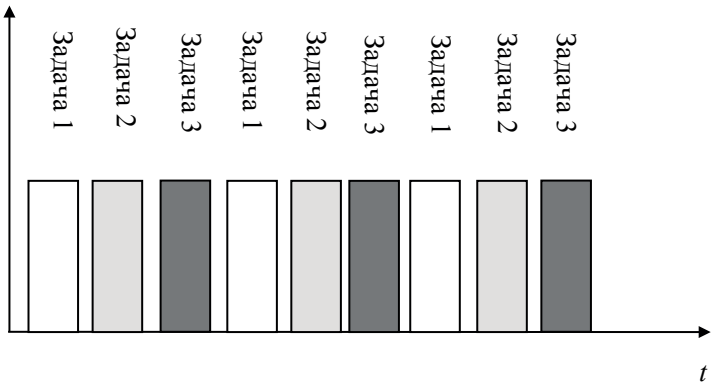


Рис. 1.3. Временная диаграмма вычислительного процесса с ОС разделения времени

Примерами ОС разделения времени являются: CTSS и MULTICS Массачусетского технологического института совместно с компаниями Bell Labs и General Electric и TSS/360 компании IBM. Несмотря на коммерческую неудачу, система MULTICS оказала существенное влияние на более поздние ОС (особенно на UNIX и ее производные, на FreeBSD, Linux, IOS и Android) [4]. В СССР первая развитая система разделения времени АИСТ-0 была создана в конце 1960-х гг. Эта ОС предназначалась для работы на многомашинном комплексе, организованном из отечественных ЭВМ М-220 и Минск-22. В ОС АИСТ-0 были реализованы ставшие классическими сейчас принципы построения ОС, в частности иерархическая архитектура на выделенном ядре, а также новаторские для того времени идеи многопроцессорной обработки и разделения времени. Руководителем проекта АИСТ был академик Андрей Петрович Ершов [5].

Как изменения в аппаратуре приводят к появлению новых технологий и алгоритмов работы ОС, так и развитие ОС предъявляет новые требования к аппаратной части вычислителя: реализация мультипрограммирования поставила новые задачи перед разработчиками аппаратуры, т. к. появились новые требования к организации всего вычислительного процесса, а именно — нужно было обеспечить наиболее быстрое переключение между задачами, а также защиту свя-

занных с ними областей ОП, где хранятся их коды и данные. Поэтому появились:

1) в процессорах — привилегированный и пользовательский режимы работы и специальные регистры для быстрого переключения с одной программы на другую, развитая система прерываний;
2) в основной памяти — средства защиты областей памяти.

Кроме разработки и реализации передовых технических концепций, этот период характеризуется рядом других прогрессивных тенденций:

1) появлением семейств программно совместимых машин и ОС для них;
2) разделением цен на аппаратуру и ПО;
3) появлением миникомпьютеров.

Что означает «семейство программно совместимых машин» и в чем состоят особенности ОС, созданных для них? До этого на каждой ВМ была установлена своя версия ОС, она не могла быть перенесена на машину другой серии того же производителя или на машину, произведенную другой компанией. Программы, написанные для некоторой ВМ, не работали на ВМ другой серии.

В семействе программно совместимых машин вычислители различались только ценой и производительностью (максимальным объемом памяти, быстродействием процессора, количеством возможных устройств ввода-вывода и т. д.), они имели одинаковые структуру и набор команд; программы, написанные для одной машины, могли работать на всех других, по крайней мере в теории.

Примерами семейств программно совместимых машин являются серии машин IBM/360 и IBM/370 (подобные серии машин производились и в СССР — это машины серии ЕС — ЕС-1020, ЕС-1030, ЕС-1040 и т. д.) и машины PDP-11 (в СССР подобные машины — СМ-3, СМ-4, СМ-1420).

Программная совместимость означала и совместимость ОС. Такая совместимость подразумевает возможность использования одной и той же ОС для работы на больших и малых машинах, обеспечения поддержки различного периферийного оборудования и решения задач различной сложности. Удовлетворить всем требованиям достаточно сложно, поэтому и сами ОС оказались очень сложными, потребовали огромных трудозатрат программистов, состояли из многих миллионов ассемблерных строк, содержали тысячи ошибок, вызывающих нескончаемый поток исправлений. Как результат, такие ОС были очень доро-

гими. Так, разработка 0S/360, объем кода которой был равен 8 Мбайт, стоил для компании IBM 80 млн долл. Один из разработчиков OS/360, Фредерик Ф. Брукс, по окончании работы написал об этом книгу [6], в которой изложил опыт своего участия в данном проекте.

Однако, несмотря на множество проблем, 0S/360 и другие ей подобные ОС реализовали на практике множество передовых идей, чем заложили прочный фундамент для создания современного системного ПО.

В конце 1960-х гг. произошло еще одно важное событие: разделение цен на аппаратуру и ПО. До определенного момента производители аппаратуры, поставляя ее вместе с установленным ПО, декларировали принцип, что деньги берутся только за аппаратуру. В связи с этим производитель не нес ответственности за ошибки в кодах поставляемого ПО. Такое положение дел не могло устраивать потребителя, кроме того, сами фирмы-производители аппаратуры готовы были передать заботу о создании программ сторонним фирмам. И такие фирмы стали появляться. Они создавали и продавали программы для машин, созданных разными производителями.

Следует упомянуть еще об одной разработке этого времени — миникомпьютере. Одной из наиболее удачных моделей такого миникомпьютера считается PDP-1 компании DEC. Такая модель обладала основной памятью в 4 Кбайта 18-битовых слов и стоила 120 тыс. долл.

Четвертое поколение ЭВМ. Сетевые операционные системы

В начале 1970-х гг. ЭВМ способны решать реальные задачи. ЭВМ есть практически в каждом филиале даже небольшой компании, и необходимо совместно использовать данные, хранящиеся на этих машинах. Возникает потребность в организации связи между ними, появляются сети машин. Сначала возникают глобальные сети, т. к. пока машины еще территориально удалены друг от друга.

В начале 1970-х гг. появляются первые сетевые ОС, которые позволяли не только осуществлять взаимодействие пользователей, но и организовать распределенное хранение и обработку массивов информации.

Любая сетевая ОС:

1) выполняет все функции локальной ОС — управление ресурсами и интерфейсную функцию;
2) обладает некоторыми дополнительными средствами, позволяющими ей взаимодействовать по сети с ОС других компьютеров.

В 1969 г. министерство обороны США начало вести работы по объединению вычислительных машин, установленных в оборонных и научно-исследовательских центрах, в единую сеть. Эта сеть получила название ARPANET и объединяла вычислительные машины, работающие под управлением разных ОС. Результатом выполнения данного проекта стала разработка стека протоколов TCP/IP и в не очень отдаленном будущем — появление сети Интернет.

Параллельно с этим разработки велись в несколько другом направлении. Появляется четвертое поколение ВМ, которые в качестве элементной базы используют большие интегральные микросхемы. Появляются микрокомпьютеры. С точки зрения архитектуры они были во многом похожи на миникомпьютеры (такие как PDP-11), но значительно дешевле.

В 1974 г. корпорация Intel выпустила новый процессор Intel 8080, который стал первым универсальным восьмиразрядным процессором, и для него потребовалась собственная ОС. Корпорация Intel привлекла к этим разработкам одного из своих сотрудников — Гэри Килдэлла. Он создал свою дисковую ОС, получившую название CP/M (*Control Program for Microcomputers* — управляющая программа для микрокомпьютеров).

В 1981 г. корпорация IBM разработала первый персональный компьютер (ПК), названный IBM PC (*Personal Computer* — персональный компьютер), и начала искать для него уникальное ПО. Сотрудники IBM обратились к Биллу Гейтсу для получения лицензии на право использования его интерпретатора языка Бейсик, его же попросили создать операционную систему. Б. Гейтс выяснил, что у местного изготовителя компьютеров, Seattle Computer Products, есть подходящая операционная система DOS (*Disk Operating System* — дисковая операционная система). Когда корпорация IBM захотела внести в ОС ряд усовершенствований, Б. Гейтс пригласил для этой работы Тима Патерсона — человека, написавшего DOS, он стал первым служащим компании Microsoft. Видоизмененная система была переименована в MS-DOS (дисковая ОС от Microsoft), она и заняла лидирующее положение на рынке.

Развитие операционных систем в 1980-е гг.

Наиболее популярной версией ОС первых ПК была MS-DOS компании Microsoft. Ее особенности:

1) однопрограммная;

2) однопользовательская;
3) с интерфейсом командной строки;
4) способная стартовать с дискеты (размер 5-дюймовой дискеты — 1,44 Мбайт; сравните с современными версиями Windows, требующими для нормального функционирования не менее 20 Гбайт дискового пространства);
5) имеющая UNIX-подобную иерархическую файловую систему.

У этой MS-DOS были следующие недостатки:

1) сама ОС не была защищена от программ пользователя, т. к. процессор Intel 8088 не поддерживал привилегированного режима;
2) не поддерживала мультипрограммирование; разработчики первых ПК считали, что при индивидуальном использовании компьютера и ограниченных возможностях аппаратуры, нет смысла в поддержке мультипрограммирования, поэтому в процессоре не был предусмотрен привилегированный режим и другие механизмы поддержки мультипрограммных систем;
3) недружественный интерфейс — интерфейс командной строки (это компенсировалось внешними программами, предоставлявшими пользователю более удобный графический интерфейс (например, Norton Commander) или средствами тонкого управления дисками (например, PC Tools);
4) несетевая ОС (сетевые функции также реализовывались в основном сетевыми оболочками, работавшими поверх ОС).

Еще одной ОС того времени была ОС Netware компании Novell. Эта компания сразу сделала ставку на разработку ОС со встроенными сетевыми функциями. Ее сетевые ОС на долгое время стали эталоном для локальных сетей, т. к. были надежно работающими и обеспечивали нужный функционал. Своеобразием семейства этих ОС было то, что они выпускались в двух версиях: серверной и рабочей станции — причем наборы поддерживаемых API-функций в этих версиях различались, результатом чего являлось то, что приложения, разработанные на компьютерах с серверной версией ОС, не работали на компьютерах с ОС версии рабочей станции, и наоборот. Данная концепция, как показало время, являлась не самой удачной, и эти ОС исчезли с рынка. В отличие от Novell, большинство других компаний создавали разные версии ОС с универсальным набором API.

В 1987 г. в результате сотрудничества двух корпораций, Microsoft и IBM, появилась OS/2 — первая многозадачная ОС для ПК с про-

цессором Intel 80286. Эта система была хорошо продумана. Она поддерживала:

1) вытесняющую многозадачность;
2) виртуальную память;
3) графический пользовательский интерфейс (не с первой версии);
4) виртуальную машину для выполнения DOS-приложений;
5) многопоточность;
6) новую файловую систему HPFS со встроенными средствами многопользовательской защиты.

Однако эта ОС имела не очень удачную рыночную судьбу, т. к. поддерживала очень ограниченный круг оборудования.

Для ПК применялись не только специально разработанные для них ОС, такие как MS-DOS, NetWare и OS/2, но и перерабатывались уже существующие и хорошо зарекомендовавшие себя версии ОС, такие как UNIX. Наиболее известной системой подобного типа была ОС UNIX, разработанная компанией Santa Cruz Operation (SCO UNIX).

Современный этап развития операционных систем

С начала 1990-х гг. выделяют 5-й этап в развитии ОС, связывая его с появлением мобильных устройств (планшетов и смартфонов) [4].

В нашей книге основное внимание будет уделяться ОС настольных компьютеров, т. к. это наиболее полные версии ОС. ОС мобильных устройств — это «усеченные», в связи с ненужностью определенных функций в мобильных устройствах, версии ОС.

К описываемому времени практически все ОС для настольных компьютеров, занимающие заметное место на рынке, стали:

1) многозадачными (принцип мультипрограммирования);
2) многопользовательскими;
3) имеющими графический интерфейс;
4) имеющими встроенную поддержку сети;
5) осуществляющими поддержку работы в Internet.

Интерфейс командной строки ориентирован на профессионалов и не дружественен рядовому пользователю, поэтому большим шагом вперед можно считать разработку графического интерфейса. Основные концепции графического интерфейса были разработаны в 1960-х гг.

Дагом Энгельбартом в научно-исследовательском институте Стэнфорда. Он создал то, что сегодня называют «графический интерфейс пользователя» (*GUI*, *Graphical User Interface*), включающий окна, иконки, меню, управление мышью. Эту идею воплотили в компании Xerox PARC, реализовав ее в создаваемых ими машинах. Стив Джобс, один из «отцов» компьютера Apple, посетил PARC, увидел GUI, оценил его потенциал, который недооценило руководство Xerox, и включил его в ОС компьютера Apple.

При рассмотрении настоящего положения дел на рынке ОС следует разделить этот рынок на 3 части в соответствии с типами устройств: ОС для настольных компьютеров, ОС для планшетных устройств и ОС для мобильных устройств. Если на рынке планшетов и мобильных устройств ситуация достаточно похожая, то рынок ОС для настольных компьютеров отличается значительно.

Безусловным лидером на рынке ОС настольных компьютеров является ОС семейства Windows (рис. 1.4), на рынке мобильных устройств ситуация сложнее (рис. 1.5): на нем конкурируют две ОС — Android разных версий и iOS (iPhone).

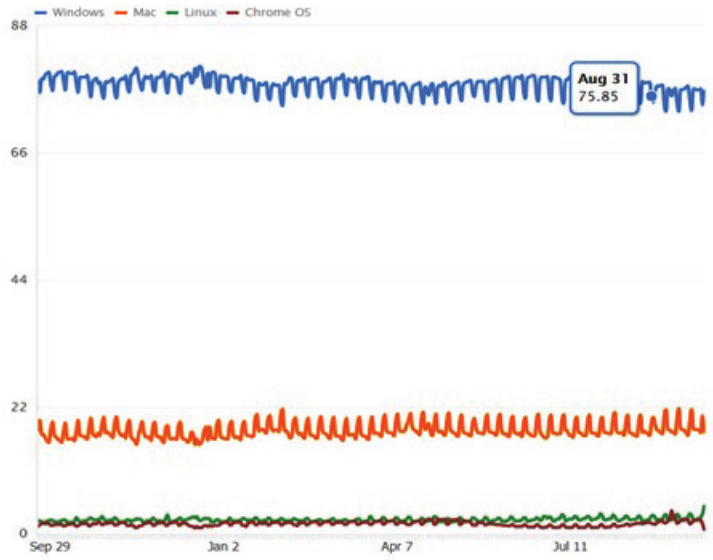


Рис. 1.4. Распространенность ОС для настольных компьютеров (2019) по версии https://clicky.com/marketshare/global/operating-systems/

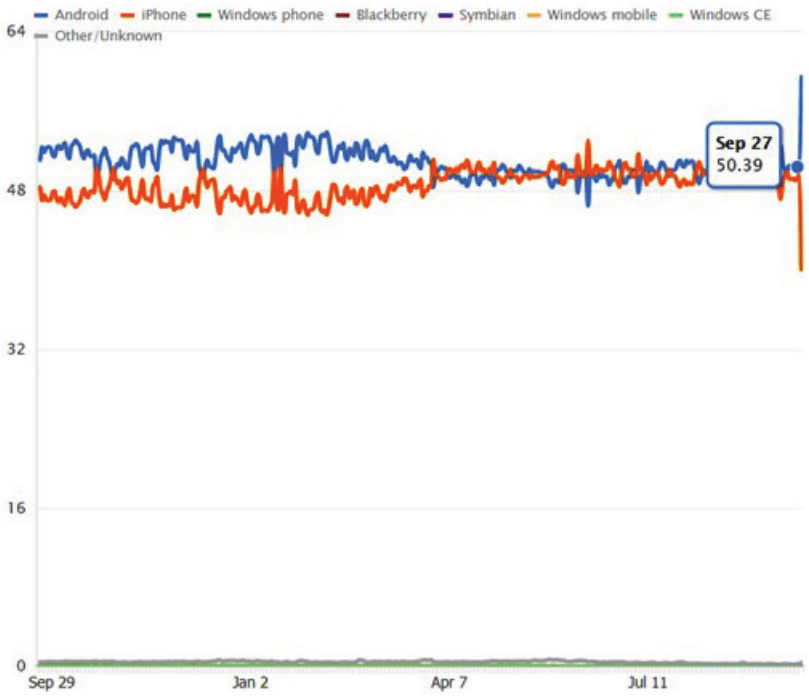


Рис. 1.5. Распространенность ОС для мобильных устройств (2019) по версии https://clicky.com/marketshare/global/operating-systems/

Семейство Windows

Поскольку версии Windows — самые распространенные версии для настольных компьютеров, рассмотрим историю этой ОС более подробно. Хронология появления версий Windows показана на рис. 1.6 [7].

Windows — коммерческие системы, код их закрыт. Днем рождения Windows считается 20 нояб. 1985 г. — именно тогда вышла первая версия ОС Microsoft Windows 1.0. Однако ни эта версия, ни версия 2.0 не имели коммерческого успеха.

Основные клиентские версии даного семейства в хронологическом порядке следующие [8]:

1) Windows 3.XX (версии 3.0; 3.1; 3.10; 3.11); интерфейс системы Windows 3.1 показан на рис. 1.7;
2) Windows 95, Windows 98, Windows 98 SE (Second Edition), Windows ME (Millenium);

3) Windows NT (New Technology — новая технология), версии 3.1; 3.5; 3.51; 4.0;
4) Windows 2000;
5) Windows XP (XP 64-bit Edition, XP Media Center Edition 2003; XP Media Center Edition 2005, XP Professional x64 Edition);
6) Windows Vista (вышла 30 нояб. 2006 г.);
7) Windows 7 (22 октяб. 2009 г.);
8) Windows 8 (октяб. 2012 г.);
9) Windows 10 (29 июля 2015 г.).

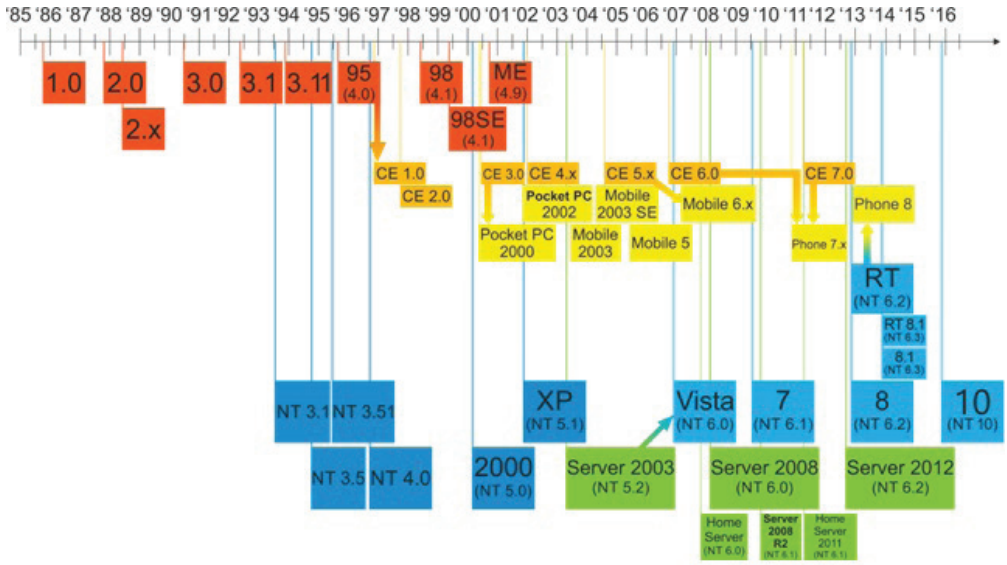


Рис. 1.6. Хронология появления версий Windows [7]

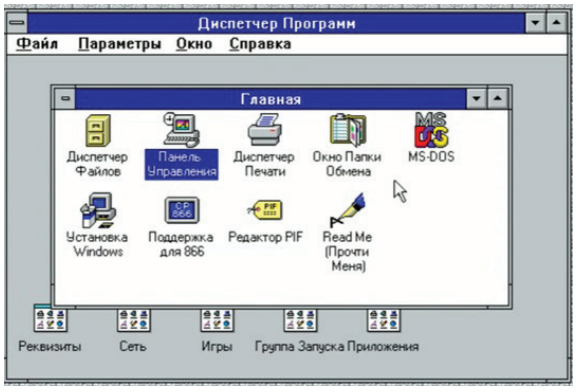


Рис. 1.7. Вид окна с ОС Windows 3.1

Начиная с Windows Vista версий в каждом новом семействе становится значительно больше. Рассмотрим версионность на примере Windows 10.

Для продажи в розницу поступают 3 версии Windows 10 для установки на новые компьютеры:

1) Windows 10 Домашняя (Windows 10 Home) — для использования дома; считается, что дома не нужны функции системы, применяемые на предприятиях, поэтому нет смысла переплачивать за лишнюю функциональность;
2) Windows 10 Профессиональная (Windows 10 Pro) — для предприятий малого бизнеса и домашних пользователей, которым необходимы расширенные возможности системы (доступен гипервизор (виртуальная машина) Hyper-V, BitLocker, и др.); существует вложенная версионность — Windows 10 Pro, Windows 10 Pro Education, Windows 10 Pro for Workstations
3) Windows 10 Корпоративная (Windows 10 Enterprise) — имеет все возможности профессиональной версии, а также дополнительные функции, которые актуальны для применения на предприятиях.

Windows 10 ориентирована на работу с устройствами различного типа:

1) настольными ПК;
2) ноутбуками;
3) планшетами;
4) смартфонами;
5) телевизорами.

В апреле 2018 г. агентство Bloomberg выпустило статью под названием «Microsoft официально перестала быть Windows-компанией». Журналисты обратили внимание на то, что ОС приносят американской компании далеко не самые большие доходы по сравнению с другими продуктами и сам программный гигант все сильнее сосредотачивается на облачных технологиях [8].

ОС, построенные на принципах UNIX

К ОС, построенным на принципах UNIX, относятся:

1) коммерческие версии (с закрытым кодом);
2) свободно распространяемые (с открытым кодом).

Отличительной чертой коммерческих ОС является то, что производители ОС являются и производителями аппаратных платформ. С одной стороны, у этих версий существуют определенные положительные особенности (предоставляются дополнительные возможности), с другой стороны, закрытые корпоративные решения отпугивают потребителей. Основные варианты коммерческих версий UNIX и их характеристики приведены далее в таблице.

Характристики коммерческих версий UNIX

ОС	Компания	Где используется	Особенности
AIX	IBM	Для своего спецоборудования. Ориентир. на рынок серверов	Имеет черты SYSTEM V, BSD
HP/UX	Hewlett Packard	Специализированное оборудование	Включена поддержка симметричных мультипроцессор. систем. ФС большого размера
IRIX	Silicon Graphics	Большинство графических рабочих станций этой фирмы работает под управлением этой ОС	Достоинство — оптимизация обработки высококачественной графики. Полностью 64-разрядная версия
Digital UNIX (OSF/1)	Digital Equipment Corporation (DEC)	Для процессоров Alpha	BSD-версия. Сейчас много черт SYSTEM V. 64-разрядная
SCO UNIX (OpenServer, UnixWare)	Santa Cruz Corp.	Компания SCO позиционирует OpenServer как систему для серверов младшего класса. UnixWare (исходные коды которой SCO приобрела у Novell) — это одна из самых мощных и перспективных UNIX на платформе Intel	Версия SYSTEM V (к SCO UNIX перешла лицензия на SYSTEM V комп. AT&T)
Solaris	Sun	Используется в серверных платформах (процессоры SPARC), недавно перенесена на платформу Intel	Имеет небольшой перечень драйверов

Некоммерческие версии (свободно распространяемое ПО с открытым кодом) включают в себя Linux, FreeBSD и др.

Операционная система Linux

Система Linux создавалась для ПК с элементной базой от Intel. В ее основе лежит проект, выполненный студентом факультета вычислительной техники Хельсинкского университета Линусом Торвальдсом. Эту ОС можно назвать наследницей UNIX, поскольку в нее введены возможности и особенности, присущие стандартным UNIX-системам, например:

1) перенесены практически все основные программы-менеджеры окон;
2) используются все утилиты Internet, включая ftp, telnet и slip.
3) имеется полный набор средств разработки на языке C++.

Несмотря на такие широкие возможности, ОС Linux является нетребовательной к аппаратным ресурсам, достаточно стабильной и быстродействующей. В минимальной конфигурации она может эффективно работать даже при наличии основной памяти объемом всего лишь 4 Мбайта.

С этой ОС связано понятие свободного ПО. Она находится в соответствии с лицензией GPL — General Public License (Универсальной общественной лицензии GNU) [9]. GNU в свою очередь расшифровывается как «GNU's not UNIX» — это рекурсивный акроним, придуманный Ричардом Столлманом, известным идеологом открытого и свободного программного обеспечения.

GPL допускает любое использование и распространение программы и любых ее модификаций, включая и коммерческое; однако она требует сохранять свободными все модификации программы, предоставляя их на условиях той же лицензии GPL и делая доступными исходные тексты.

В 1985 г. Ричард Столлман написал «Манифест GNU» (The GNU Manifesto), в котором описал свои цели так: «По моему мнению, золотое правило требует, если мне нравится программа, раздать ее другим людям, которым она тоже понравится. Торговцы ПО хотят разделить пользователей и властвовать над ними, заставив каждого из них не делиться с другими. Я отказываюсь от такого нарушения солидарности с другими пользователями. Я не могу, не погрешив против совести, подписать соглашение о нераспространении... Чтобы продолжать пользоваться компьютерами, сохранив честь, я решил собрать достаточную базу свободного программного обеспечения. Тогда я смо-

гу обойтись без каких-либо несвободных программ» (с сайта URL: http://www.gnu.org/gnu/manifesto.ru.html).

Существует 4 разновидности свободы пользователей программы:

1) свобода запускать программу в любых целях (свобода 0);
2) свобода изучения работы программы и адаптация ее к вашим нуждам (свобода 1), основанием для этого является доступ к исходным текстам;
3) свобода распространять копии (свобода 2);
4) свобода улучшать программу и публиковать ваши улучшения (свобода 3).

Программа считается свободной, если пользователи располагают всеми четырьмя свободами.

Следует отметить еще одно своеобразие ОС Linux — множество версий и множество компаний-дистрибьюторов. Каждый дистрибьютор видит свою версию по-своему, поэтому они могут значительно отличаться друг от друга. Наиболее известные версии: Red Hat, Ubuntu, ASP Linux, Debian.

Для лучшего понимания особенностей этой ОС приведем более подробное описание некоторых ее версий.

Версии Red Hat Enterprise Linux

Компания Red Hat [10] была основана в 1993 г. Бобом Янгом (Bob Young) и Марком Юингом (Marc Ewing) и обязана своим названием шляпе, которую Юинг носил во время учебы в Университете Карнеги-Меллона. Компания уникальна тем, что зарабатывает почти все свои деньги за счет разработки и продажи ПО с открытым кодом. Red Hat также является крупнейшей компанией-участником проекта разработки ядра Linux по размеру программного вклада.

Red Hat Enterprise Linux (RHEL) — это высокопроизводительные серверные и настольные системы для предприятий, пришедшие на смену свободной и бесплатной Red Hat Linux, последняя стабильная версия которой вышла в 2003 г.

В то время как RHEL приносит компании деньги, дух той первой Red Hat Linux живет в ОС Fedora, которая была впервые представлена 2003 г. и базировалась на коде Red Hat Linux, а теперь развивается в рамках спонсируемого Red Hat одноименного проекта с открытым

кодом. Сегодня Fedora служит чем-то вроде полигона для обкатки новых функций и технологий, которые затем предлагаются корпоративным заказчикам в составе коммерческого продукта RHEL [11].

Исходный код RHEL также доступен на условиях лицензирования GPL и может свободно компилироваться всеми желающим. Готовые продукты на основе этого кода Red Hat предоставляет только за деньги. Компания Red Hat предлагает семейство подписок на Red Hat Enterprise Linux, которые адресованы как для серверного, так и для клиентского окружения. Варианты Red Hat Enterprise Linux AS и Red Hat Enterprise Linux ES ориентированы на высокопроизводительные серверы и серверы начального уровня. Вариант Red Hat Enterprise Linux WS ориентирован на технические рабочие станции. Red Hat Desktop — это новейший представитель семейства Red Hat Enterprise Linux и спроектирован для использования на рабочих местах пользователей.

Версии Debian

Проект Debian — это ассоциация людей, общим делом которых является создание свободной ОС. Созданная ОС называется Debian. Большая часть основных инструментов, которые наполняют ОС, взята из Проекта GNU (GNU project). Эти инструменты также являются свободными.

Debian содержит более 59 000 пакетов (скомпилированного заранее программного обеспечения в удобном для установки на компьютер формате), менеджер пакетов (APT), а также другие утилиты, благодаря которым можно управлять тысячами пакетов на тысячах компьютеров так же просто, как установить одно-единственное приложение. И все это свободно.

Приведем цитату с сайта Debian, которая хорошо объясняет, что значит свободное ПО: «Вы можете спросить: если это программное обеспечение свободно, то почему я должен платить продавцу деньги за CD или платить провайдеру интернет за загрузку?

Когда вы покупаете диск, то вы платите за время и работу по созданию этого диска, а также за риск (на случай, если диски не будут проданы все). Другими словами, вы платите за физический носитель, используемый для распространения программного обеспечения, а не за само ПО.

Когда мы говорим «free» (переводится как «свободный», так и «бесплатный»), мы говорим о свободе ПО, а не о том, что оно ничего не стоит» [12].

Версии Ubuntu

Дистрибутив Ubuntu [12] основан на дистрибутиве Debian и поддерживается компанией Canonical Ltd. Новые версии Ubuntu появляются каждые полгода. Это довольно высокая скорость обновления — многие другие дистрибутивы Linux обновляются в среднем раз в год. Версия 4.10 — это первая версия Ubuntu, которая стала известна широким массам. Она вышла 20 октяб. 2004 г. По сведениям [13], «в мире Linux самым популярным дистрибутивом считается Ubuntu».

По одной из версий, «ubuntu» — это понятие, заимствованное у африканских племен, которое может быть определено как: «Humanity to others» (гуманность по отношению к остальным) или «I am what I am because of who we all are» (я такой, потому что я часть сообщества).

Цель Ubuntu — создать удобные для пользователя дистрибутивы Linux для тех, кто не знаком с Linux в целом. «Ubuntu уделяет большое внимание удобству использования и простоте установки» [14], тогда как «Debian известен относительно строгим соблюдением Unix и принципов свободного программного обеспечения» [13] и пожертвует удобством для пользователей в пользу этих идеалов.

Основные («canonical») варианты системы:

1) Ubuntu — версия ОС, в которой сочетаются простота и удобство интерфейса с функциональностью, основана на графической среде Unity собственной разработки и компонентах рабочей среды GNOME (ее экран показан на рис. 1.8);
2) Ubuntu Server — серверная версия ОС, включает эффективные средства развертывания облачной инфраструктуры;
3) Edubuntu — вариант Ubuntu, предназначенный для образовательных учреждений, содержит различные приложения для этой сферы деятельности;
4) Kubuntu — Ubuntu с графической средой KDE и приложениями, предназначенными для использования в этой среде;
5) Lubuntu — минималистичный вариант Ubuntu;
6) Ubuntu Studio — вариант Ubuntu, предназначенный для людей, активно занимающихся редактированием и созданием мультимедийного контента.

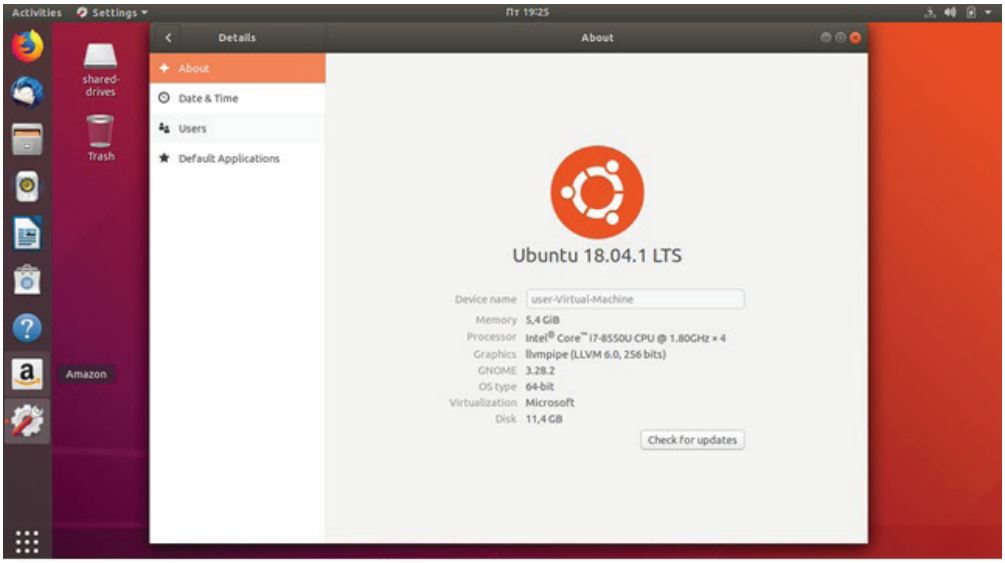


Рис. 1.8. Экран ОС Ubuntu

FreeBSD

Данная система занимает 2-е место в мире среди систем с открытым кодом. FreeBSD [15] — своего рода компромисс между энтузиазмом и корпоративной рутиной.

Эта ОС, как и Linux, обязана своим появлением ОС UNIX. Развитие UNIX пошло в определенный момент времени по двум направлениям, одно из них развивалось в Калифорнийском университете в Беркли, именно туда перешел Кен Томпсон, и результатом совместной его работы со студентами и преподавателями явилась эта ОС, названная Berkley Software Distribution — BSD. Данная версия была настолько удачной, что именно она легла в основу проекта ARPANET.

Каждый проект внутри FreeBSD поддерживает публично доступное дерево исходных текстов программ. При помощи CVS (*Concurrent Version System*) можно получить доступ к коду проекта, его документации и вспомогательным файлам. Это позволяет пользователю в любой момент получить копию дерева любого из проектов или системы в целом.

Всех, кто в той или иной мере участвует в разработке FreeBSD, можно разделить на 3 категории:

1) контрибьюторы (contributors) — те, кто пишет код или документацию, но не имеет права вносить изменения непосредственно

в код разработки; они только предоставляют изменения и дополнения к коду, а решения об их внесении принимают коммитеры;
2) коммитеры (commiters) — участники группы разработки, имеющие право записи в дерево CVS;
3) Core team — группа людей, управляющих деятельностью разработчиков FreeBSD, именно они определяют стратегию развития FreeBSD.

У каждой ОС FreeBSD есть свой номер версии, например, 10.4, 12.0. Кроме того, проект FreeBSD предоставляет пользователю 3 различных варианта системы:

1) Current — версия для разработчиков (например, 10.0. Current) — все новые разработки проходят тестирование именно на этой ветке;
2) Release — версия для конечных пользователей (как правило, появляется раз в 3–6 мес.);
3) Stable — версия FreeBSD, являющаяся результатом работы с версиями Release; по мере того как в Release обнаруживаются ошибки и в дерево CVS вносятся изменения, дистрибутив переходит на стадию Stable.

Лицензия, используемая FreeBSD (не GPL), не содержит так называемых передающихся свобод. Она не обязывает вас открывать исходный текст программ, если вы этого не хотите. В этом отношении BSD-лицензия проявляет больше либерализма.

На рис. 1.9 представлена страница проекта FreeBSD на русском языке.

Рис. 1.9. Страница проекта FreeBSD

Mac OS

Mac OS фирмы Apple [16] — это коммерческая версия ОС с закрытым кодом. Она является второй по распространенности для настольных компьютеров ОС в мире, уступает только Windows (но уступает значительно).

Одной из отличительных черт данной системы является стабильность ее работы и надежность. Это обусловлено тем, что ОС была создана специально для ПК компании Apple, вследствие чего Mac OS и оборудование полностью совместимы. Apple использует Mac OS с 1984 г., с момента появления первого ПК Apple Macintosh.

Mac OS была первой из современных ОС, которая применила графический интерфейс пользователя (это упоминалось при описании истории развития ОС). С 1984 по 2001 г. свет увидели версии System 1 — Mac OS 9, которые принято считать классическими. С 2000 г. ОС получила наименование «Mac OS X» (X — римская цифра десять), а с 2016 — «macOS» [17]. Последнюю на сегодняшний день версию macOS Mojave, названную так в честь калифорнийской достопримечательности, представили публике 4 июня 2018 г. на конференции WWDC-2018. На рис. 1.10 представлен экран с первой версией этого семейства систем, а на рис. 1.11 — с последней из существующих.

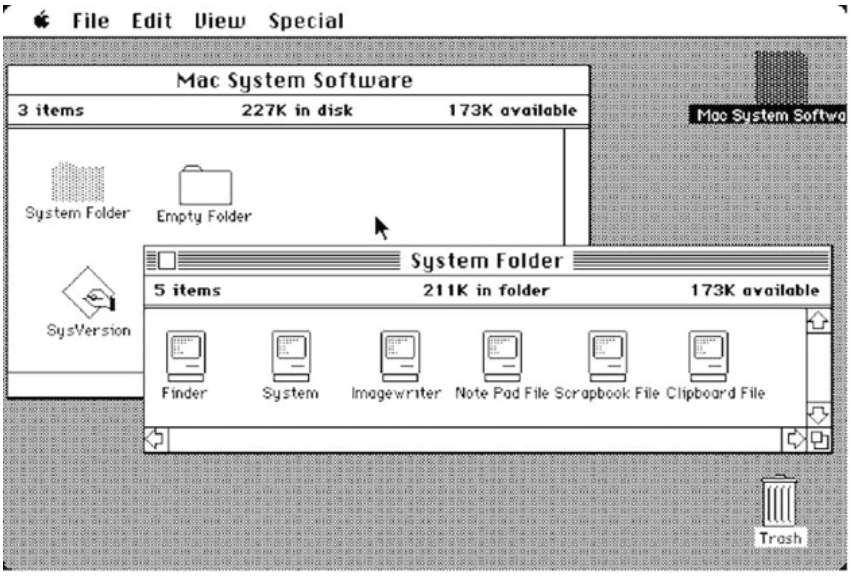


Рис. 1.10. Экран с версией ОС System 1

Рис. 1.11. Экран с версией macOS Mojave 10.14

Современные тенденции развития системы — это обеспечение ее безопасности и сохранение приватности данных пользователя.

Еще одна концепция развития была заявлена в рамках презентации новой версии системы. Представитель Apple ответил на главный вопрос, интересовавший публику, — станет ли Apple объединять мобильную и настольную ОС по примеру Microsoft Windows. Ответ был — нет, компания по-прежнему будет разрабатывать две отдельные системы в соответствии со спецификой работы iГаджетов и компьютеров Mac, но отныне разработчики смогут относительно легко и удобно портировать приложения с iOS на macOS и обратно.

Контрольные вопросы

1. На какие классы обычно разделяется ПО и каковы основные характеристики этих классов?
2. Что называется операционной системой?
3. Сколько поколений вычислительной техники обычно выделяют? Что лежит в основе этого разделения?

4. Какие обычно периоды выделяют в истории развития ОС и чем они характеризуются?
5. Что можно отнести к особенностям современных ОС? Какие ОС являются лидерами на рынке ПО этого типа?
6. Что такое «свободное ПО» и в чем состоят эти свободы?
7. Совпадают ли понятия «свободное ПО» и «бесплатное ПО»?

2. Требования к современным операционным системам. Функциональные компоненты операционной системы автономного компьютера

Требования к современным операционным системам

Главным требованием, предъявляемым к ОС, является выполнение ею основных функций: эффективного управления ресурсами и обеспечения удобного интерфейса для пользователя и прикладных программ. Кроме требований функциональной полноты, к современной ОС должны предъявляться требования, которые обычно относят к классу эксплуатационных. Они перечислены ниже.

Расширяемость. Аппаратная часть компьютера изменяется значительно быстрее, чем меняются ОС, жизненный цикл которых обычно длится несколько лет (если не учитывать версионность). Яркий пример такой ОС-долгожителя — это ОС семейства UNIX. ОС должна поддерживать появляющиеся технологии, новые типы устройств без полного переписывания ее кода. Если код ОС написан так, что все дополнения и модификации могут вноситься без его полного переписывания, то такую ОС называют расширяемой. Это может быть получено за счет модульной, объектно ориентированной структуры, с хорошо разработанными интерфейсами между модулями.

Переносимость. Код ОС должен легко переноситься с одной аппаратной платформы на другую. Под аппаратной платформой понимается не только тип процессора, но и способ организации всей аппаратуры. Иногда употребляют другой термин — многоплатформенность — это свойство ОС иметь несколько версий реализации для разных аппаратных платформ.

Совместимость. Существуют ОС, имеющие множество версий, часто для них употребляют термин «семейство». В течение жизненного

цикла этих семейств для них создается множество различных приложений. Пользователь привыкает работать с каким-либо приложением и ему хотелось бы продолжать с ним работать и в другой версии этой же ОС. ОС обладает свойством совместимости с другой ОС (или другой версией той же ОС), если в ее среде выполняются приложения, разработанные для этой другой ОС. При этом различают совместимость на уровне двоичных кодов (исполняемый код сразу запускается при переносе в среду новой ОС) и совместимость на уровне исходных текстов (когда требуется ряд дополнительных действий по трансляции и компоновке приложения).

Защищенность. Современная ОС должна защищать данные и другие ресурсы вычислительной системы от основных классов угроз. Защищенная ОС обязательно должна содержать средства разграничения доступа пользователей к своим ресурсам, а также средства проверки подлинности пользователя, начинающего работу с ней. Кроме того, защищенная ОС должна содержать средства противодействия случайному или преднамеренному выводу ее из строя.

Надежность и отказоустойчивость. Система должна быть защищена от ошибок, сбоев и отказов. Ее реакции на любые события должны быть предсказуемыми, а приложения не должны иметь возможности наносить вред. Файловые системы такой ОС должны быть восстанавливаемыми, она должна поддерживать аппаратные средства обеспечения отказоустойчивости, такие, например, как RAID-массивы и (или) источники бесперебойного питания.

Производительность. ОС должна обладать таким максимальным быстродействием и временем реакции, какое позволяет аппаратная платформа, на которой она установлена. На производительность ОС влияет множество факторов, наиболее значимые — это ее архитектура, алгоритмы выполнения ее функций, технологии и качество программирования, возможность мультипроцессирования и т. д.

Классификация операционных систем

Существует множество классификаций ОС, основанных на различных характеристиках. Приведем несколько примеров таких классификаций.

Поддержка многозадачности. По числу одновременно выполняемых задач ОС могут быть разделены на два класса:

1) однозадачные (например, MS-DOS);
2) многозадачные (все современные ОС для настольных компьютеров), которые можно разделить на 2 класса:
   - с вытесняющей многозадачностью (NetWare, Windows 3.x);
   - невытесняющей многозадачностью (современные версии Linux и Windows).

Основным ресурсом любой вычислительной системы является процессорное время. Способ разделения процессорного времени между несколькими одновременно существующими в системе процессами (или потоками, если таковые определены в системе) во многом определяет свойства ОС. Разницей между вытесняющей и невытесняющей многозадачностью является степень централизации планирования процессов. При невытесняющей многозадачности, планирование процессов целиком осуществляется ОС, а при вытесняющей — оно распределено между ОС и самими процессами. При невытесняющей многозадачности, активный процесс сам определяет моменты освобождения процессора и передает управление ОС, а та выбирает из очереди готовый к выполнению процесс. При вытесняющей многозадачности, все решения о моментах переключения и процессах, которые будут загружены на процессор, принимаются ОС, а не активными процессами.

Под поддержкой многопользовательского режима чаще всего понимают возможность работы под разными регистрационными именами с различными настройками. По числу одновременно работающих пользователей ОС делятся:

1) на однопользовательские (MS-DOS, ранние версии OS/2);
2) многопользовательские (современные версии Linux и Windows).

Главным отличием многопользовательских систем от однопользовательских является наличие разграничения доступа (защиты) к данным пользователей. Далее будут рассмотрены различные примеры таких систем.

Многопроцессорные ОС делятся на асимметричные и симметричные. Критерием деления является способ организации вычислительного процесса в системе с несколькими процессорами. Все системные процессы асимметричной ОС выполняются только на одном из процессоров системы, а процессы, запущенные пользователем, распределяются на остальные процессоры. В симметричной ОС любой процесс

может выполняться на одном из процессоров в соответствии с алгоритмом распределения, заложенным в ОС.

По типу аппаратуры различают ОС мобильных устройств, планшетов, настольных компьютеров, специализированного компьютерного оборудования (графические станции, мейнфреймы). Среди перечисленных типов устройств могут быть как однопроцессорные, так и многопроцессорные конфигурации. В любом из этих случаев особенности аппаратной платформы отражаются на специфике ОС. Очевидно, что ОС большой машины является более сложной и функциональной, чем ОС мобильного устройства.

По критерию «особенности областей использования» (критерий эффективности) многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

1) системы пакетной обработки (например, ОС ЕС);
2) системы разделения времени (Windows, Linux, MAC OS);
3) системы реального времени (QNX, RT/11).

О первых двух типах систем рассказывалось при описании истории развития ОС. Системы реального времени применяются в основном в технических системах для управления различными объектами или процессами. В таких системах важна такая характеристика, как предельно допустимое время — это максимальное время, за которое должна быть выполнена программа, осуществляющая управляющие воздействия на объект или процесс. Критерием эффективности для систем реального времени является их способность удерживать значение данной характеристики в определенных пределах. При этом следует отметить, что не всегда удается эффективно использовать ресурсы.

Функциональные компоненты операционной системы автономного компьютера

Основные задачи, решаемые ОС, следующие:

1) управление ресурсами;
2) интерфейс с пользователем.

Функции ОС автономного компьютера обычно группируются по двум основным признакам:

1) в соответствии с типами локальных ресурсов;
2) со специфическими задачами, решаемыми в отношении всех ресурсов.

Такие группы функций называют подсистемами, следовательно, можно выделить подсистемы двух типов: функциональные и общие.

Рассмотрим ОС автономного компьютера (не рассматриваем сетевые функции). Для понимания того, какими ресурсами должна управлять ОС, обратимся к классической схеме устройства, являющегося вычислительной машиной, — к машине фон Неймана (рис. 2.1).

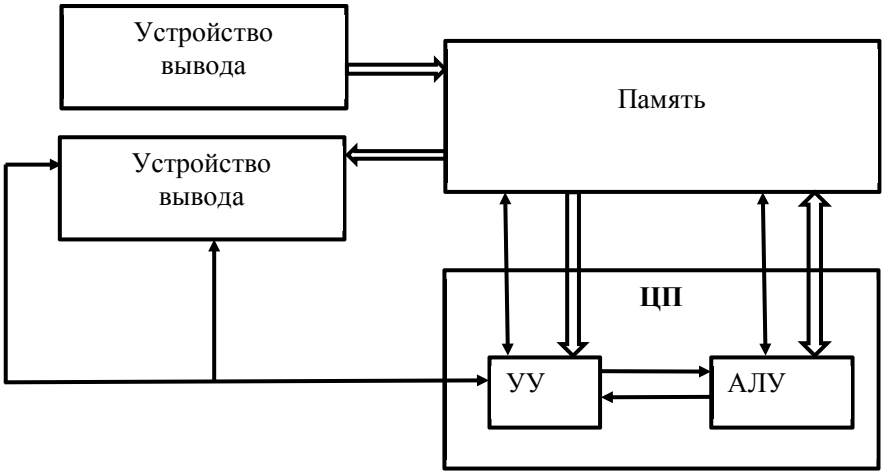


Рис. 2.1. Схема вычислительной машины фон Неймана

Машина фон Неймана образована запоминающим устройством (ЗУ), арифметико-логическим устройством (АЛУ), устройством управления (УУ) и устройствами ввода-вывода. В современных компьютерах функции АЛУ и УУ выполняет единое устройство — центральный процессор (ЦП). На рисунке толстыми (двойными) стрелками показаны потоки команд и данных, а обычными — передача между отдельными устройствами компьютера управляющих и информационных сигналов. Основными ресурсами являются: время центрального процессора, память (основная, или оперативная, память) и устройства ввода-вывода. Следовательно, основными функциональными подсистемами являются подсистемы управления временем ЦП (процессами), основной памятью и устройствами ввода-вывода.

К общим подсистемам обычно относят: подсистему интерфейса с пользователем, подсистему администрирования и подсистему безопасности. Структура ОС как системы показана на рис. 2.2.

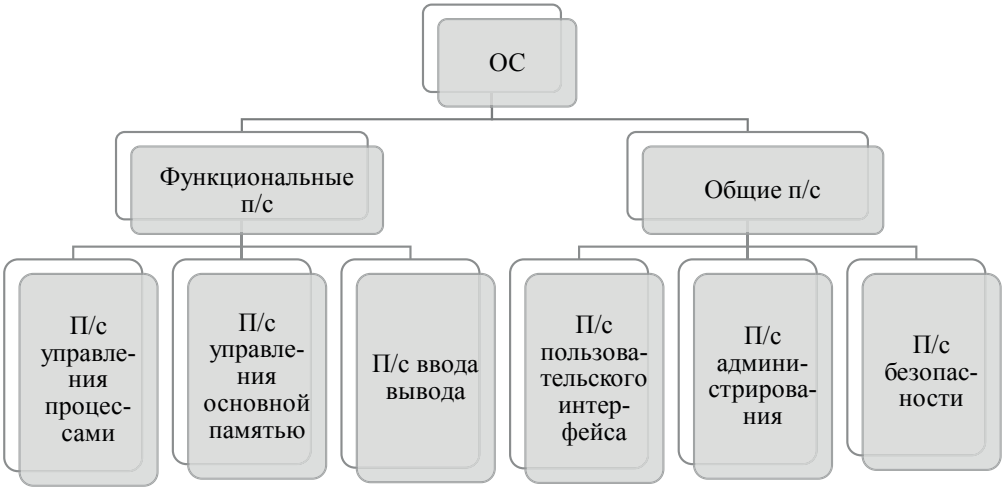


Рис. 2.2. Структура ОС

Контрольные вопросы

1. Какие основные компоненты должно содержать устройство, чтобы считаться вычислительной машиной?
2. Какие два типа подсистем составляют операционную систему?
3. Какими основными типами ресурсов управляет ОС и какие подсистемы в связи с этим можно выделить в ее структуре?
4. Какие основные требования предъявляются к современной ОС и как их можно ранжировать по значимости?
5. Какие структурные особенности обеспечивают расширяемость операционной системы?
6. К каким классам операционных систем можно отнести современные версии Windows?
7. Что можно отметить как отличительную особенность ОС реального времени?

3. Подсистема управления процессами

Подсистема управления процессами является одной из основных в структуре ОС.

Под **процессом** традиционно понимают программу в стадии выполнения. Для каждого вновь создаваемого процесса ОС создает системные информационные структуры, которые содержат как данные о потребностях процесса в ресурсах системы, так и актуальные данные о фактически выделенных ему ресурсах. Поэтому процесс часто определяют как некоторую заявку на системные ресурсы. К этим ресурсам относятся:

1) область основной памяти, в которой будут размещены коды и данные процесса (эта область называется адресным пространством процесса);
2) процессорное время;
3) файлы и устройства ввода-вывода.

Поскольку все современные ОС являются мультипрограммными, основными задачами, которые должна выполнять подсистема управления процессами, являются:

1) создание и уничтожение процессов (т. е. информационных структур, связанных с процессами);
2) выделение и учет ресурсов, выделенных каждому процессу;
3) защита ресурсов, выделенных одному процессу, от доступа к ним остальных процессов;
4) организация и обслуживание очередей заявок процессов на ресурсы (очереди к ЦП, к устройству печати и т. д.);
5) переключение с процесса на процесс;
6) организация взаимодействия процессов.

При выполнении этих функций, подсистема управления процессами обращается к другим функциональным подсистемам ОС, таким как подсистема управления основной памятью и подсистема ввода-вывода.

Примерами информационных структур, содержащих сведения о процессах, являются: блок управления задачей (TCB — Task Control

Block) в OS/360, управляющий блок процесса (PCB — Process Control B1ock) в OS/2, дескриптор процесса в UNIX (Linux), объект-процесс (process object) в Windows.

В информационные структуры процесса обычно входит следующее:

1) системный идентификатор процесса (в системах Linux — дополнительно идентификатор родительского процесса);
2) системный идентификатор пользователя, создавшего процесс;
3) данные о расположении в памяти кодов и данных процесса;
4) степень привилегированности процесса
5) и др.

Часто также включаются вспомогательные данные, характеризующие историю пребывания процесса в системе, такие как:

1) доля времени, потраченная процессом на операции ввода-вывода, и доля, потраченная на вычисления;
2) текущее состояние процесса.

Данные подобного типа учитываются ОС при принятии решения о моменте и объеме предоставления ресурсов процессу.

В мультипрограммной ОС одновременно может существовать несколько процессов. Некоторые из них порождаются по инициативе пользователей и их приложений, эти процессы называют **пользовательскими**. Другие процессы инициализируются ОС для выполнения своих функций, они называются **системными**.

Во время своего жизненного цикла процесс может быть многократно прерван и восстановлен на процессор. Для того чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды процесса идентифицируется:

1) по состоянию регистров;
2) состоянию программного счетчика;
3) режиму работы процессора;
4) указателям на открытые файлы;
5) информации о незавершенных операциях ввода-вывода;
6) кодам ошибок выполняемых данным процессом системных вызовов.

Эту информацию определяют как контекст процесса. В таком случае говорят, что при смене процесса происходит переключение контекстов.

ОС выполняет также функции по синхронизации процессов, которые обеспечивают приостановку процесса до наступления определен-

ного события в системе, например завершения операции ввода (или вывода), которую выполняет ОС по запросу этого процесса.

Когда процесс завершается, ОС удаляет всю информацию о его пребывании в системе: закрывает все файлы, которые читал и в которые записывал процесс, освобождает области основной памяти, отведенные для размещения его кодов и данных. Выполняется коррекция системных очередей и списков ресурсов, ссылающихся на завершенный процесс.

Понятие «процесс» и «поток»

В некоторых современных ОС кроме понятия «процесс» существует понятие «поток» (thread). **Поток** — это средство, с помощью которого можно распараллелить процесс, иными словами, поток — это некая сущность внутри процесса, которой выделяется процессорное время для выполнения.

Существует множество процессов, в которых определенные действия можно выполнять одновременно (например, в текстовом процессоре какой-то текст можно редактировать, а какой-то — выводить на печать), эти действия и оформляются в виде отдельных потоков команд. Можно было бы оформлять их как отдельный процесс, но это не всегда целесообразно в связи с накладными расходами, которые требует процесс (например, на создание достаточно сложных и больших по размеру занимаемой памяти информационных структур), также часто необходимо разделять ресурсы, которые при различных процессах будут изолированы друг от друга.

В каждом процессе существует хотя бы один поток. Первичный поток создается системой сразу при создании процесса. Далее этот поток может породить другие потоки, те в свою очередь новые и т. д. Таким образом, один процесс может содержать несколько потоков, а они исполняют код в адресном пространстве процесса.

В ОС, где одновременно существуют процессы и потоки, процесс — это заявка на потребление всех видов ресурсов, кроме единственного — времени процессора. Этот ресурс распределяется ОС между другими единицами работы — потоками.

Преимущества введения понятия потока:

1) создание потоков требует от ОС меньше накладных расходов, чем при создании процессов;
2) мультипрограммирование на уровне потоков повышает производительность системы;
3) использование потоков позволяет создавать более читабельные и логичные программы.

Создание процессов и потоков в ОС Windows

Создание процесса Windows осуществляется вызовом такой функции, как CreateProcess () (или ее аналогов), и проходит в несколько этапов:

1) открывается исполняемый файл, который будет выполняться в процессе;
2) создается объект Win32 «процесс»;
3) создается первичный поток;
4) подсистеме Win32 посылается сообщение о создании процесса и потока;
5) начинается выполнение созданного первичного потока;
6) в контексте процесса и первичного потока инициализируется адресное пространство (загружаются библиотеки) — программа начинает выполняться.

Пример создания процесса, в котором выполняется приложение «Калькулятор»:

```
#include <windows.h>
int main(int argc, char* argv[])
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    if(!CreateProcess(NULL, "c:/windows/calc.exe", NULL,
    NULL, FALSE,0, NULL, NULL, &si, &pi))
    return 0;
    //Закрыть структуры, определяющие процесс и поток
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}
```

Процесс завершается:

1) если входная функция первичного потока возвратила управление;
2) один из потоков процесса вызвал функцию ExitProcess();
3) поток другого процесса вызвал функцию TerminateProcess().

Если необходимо создание дополнительных потоков (кроме первичного), необходимо вызывать функции CreateThread() [18].

Планирование и диспетчеризация потоков

Как обсуждалось ранее в отношении процессов, потоки во время своего жизненного цикла могут быть многократно прерваны и возобновлены.

В системе, не поддерживающей понятие потока, все изложенное далее может быть отнесено к процессу.

Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации.

Активность по определению того, в какой момент прервать выполнение текущего активного потока и какой поток выбрать на выполнение, называется **планированием.** Планирование потоков осуществляется на основе данных, хранящихся в информационных структурах потоков. При планировании может приниматься во внимание:

1) приоритет потоков;
2) время их ожидания в очереди;
3) накопленное время выполнения;
4) интенсивность обращений к вводу-выводу
5) и другие факторы.

Существует множество различных алгоритмов планирования потоков, по-своему решающих каждую из приведенных выше задач. Алгоритмы планирования могут иметь разные цели. Например, в одном случае может быть выбран алгоритм планирования, который гарантирует, что ни один поток не будет занимать процессор дольше определенного значения, в другом случае целью может быть максимально быстрое выполнение небольших по времени задач, а в третьем случае процессорное время в первую очередь будут получать потоки интерактивных приложений. Именно алгоритмы планирования заложены в основу одной из наиболее распространенных классификаций ОС,

когда система относится либо к системам пакетной обработки, либо к системам разделения времени, либо к системам реального времени

У ОС есть планировщик статического типа, если он принимает решения заранее, а не во время работы системы. Результатом работы такого планировщика является расписание — таблица, в которой указывается, какому потоку, когда и в течение какого временного интервала должен быть предоставлен процессор. После того как расписание создано, оно применяется ОС для переключения потоков и процессов.

Динамический планировщик принимает решения не заранее, а во время работы компьютера, основываясь на сложившейся в нем ситуации. ОС, имеющая статического планировщика, на составление расписания затрачивает ресурсов меньше, чем при динамическом планировщике, дальнейшие действия такой ОС сводятся в основном к диспетчеризации потоков или процессов.

Диспетчеризация является реализацией найденного в результате планирования решения. Если планирование можно считать стратегией, то диспетчеризация — это тактика, реализованная через поведение системы.

Прежде чем прервать выполнение потока, ОС запоминает его контекст, т. е. ту информацию, которая будет необходима для возобновления его выполнения. Контекст отражает как состояние аппаратуры компьютера в момент прерывания, так и параметры операционной среды: коды ошибок системных вызовов, ссылки на открытые файлы, данные об операциях ввода-вывода и др.

Диспетчеризация сводится к следующему:

1) сохранение контекста текущего потока, который требуется вытеснить с процессора;
2) загрузка контекста нового потока, который был выбран планировщиком;
3) запуск выбранного потока на выполнение.

Поскольку операция переключения контекстов оказывает значительное влияние на производительность компьютера, диспетчеризация реализуется совместно с аппаратными средствами процессора (эти средства можно считать частью ОС, поэтому ОС определяют не просто как совокупность взаимосвязанных программных модулей, а как программно-аппаратный комплекс).

Состояния потока

ОС выполняет планирование потоков на основе анализа их состояний. В мультипрограммной системе поток может находиться в одном из трех основных состояний:

1) выполнения — активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
2) ожидания — 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) готовности — также пассивное состояние потока, но в этом случае поток заблокирован из-за внешнего по отношению к нему обстоятельства (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

Состояния выполнения и ожидания могут быть отнесены и к задачам, выполняющимся в однопрограммном режиме, а вот состояние готовности характерно только для режима мультипрограммирования.

Жизненный цикл любого потока есть смена его состояний в соответствии с алгоритмом планирования, принятым в данной ОС.

Рассмотрим граф, отражающий смену состояний потока (рис. 3.1). Каждый вновь созданный поток находится в состоянии готовности, он готов в любой момент начать выполняться, для чего стоит в очереди из таких же готовых к выполнению потоков к процессору. Когда в результате планирования подсистема управления потоками принимает решение о выполнении данного потока, тот переходит в состояние выполнения и пребывает в нем до тех пор, пока сам добровольно не освободит процессор или пока не будет вытеснен с процессора по окончании выделенного ему кванта времени. Поток освобождает процессор добровольно, если ожидает наступления какого-нибудь события, например окончания ввода-вывода на внешнее устройство, при этом он переходит в состояние ожидания. В случае принудительного вытеснения с процессора, поток все еще готов выполняться, поэтому переходит в состояние готовности. В состояние готовности по-

ток переходит также из состояния ожидания, когда это ожидаемое событие наступило.

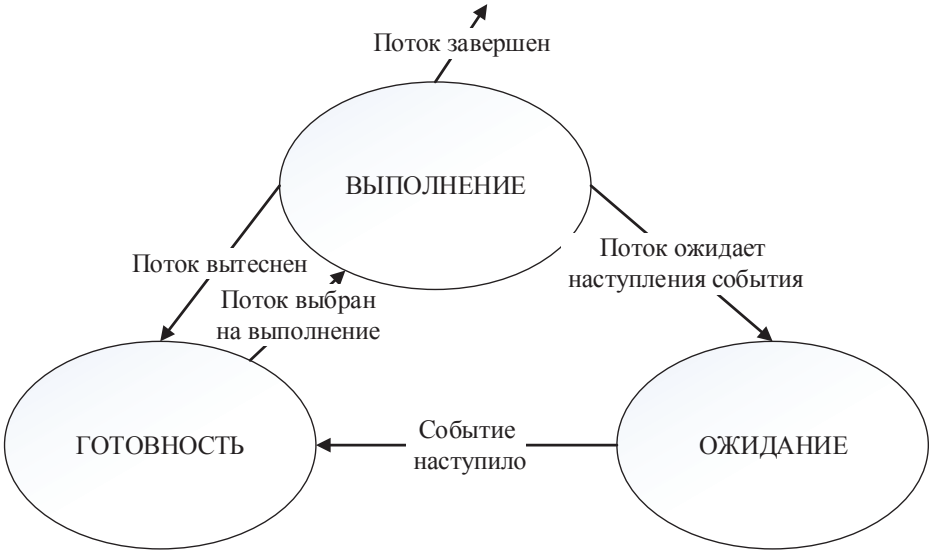


Рис. 3.1. Граф смены состояний потока

В состоянии выполнения в однопроцессорной системе может находиться не более одного потока, а в состоянии ожидания и готовности — несколько потоков. Эти потоки образуют очереди, программно это реализуется через организацию списковых структур из описателей таких потоков. Списковые структуры подразумевают, что каждый описатель потока содержит по крайней мере один указатель на другой описатель, соседствующий с ним в очереди, (односвязные списки) или два указателя — на предшествующий и последующий элемент (двусвязные списки). Такая организация очередей позволяет легко их переупорядочивать, включать и исключать из них потоки.

Алгоритмы планирования

С самых общих позиций все множество алгоритмов планирования можно разделить на два класса: вытесняющие и невытесняющие. Невытесняющие (*non-preemptive*) алгоритмы основаны на том, что ак-

тивному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление ОС для того, чтобы та выбрала из очереди другой готовый к выполнению поток. Вытесняющие (*preemptive*) алгоритмы — такие способы планирования потоков, в которых решение о переключении процессора с выполнения одного потока на выполнение другого потока принимается ОС, а не активной задачей.

Степень централизации механизма планирования потоков лежит в основе различий между вытесняющими и невытесняющими алгоритмами. Используя невытесняющие алгоритмы, можно создать достаточно эффективную систему, т. к. переключение с потока на поток возможно производить в те моменты времени, когда для этого требуется меньше системных ресурсов. Примером достаточно успешного использования невытесняющего планирования являлись версии ОС Netware, в которых высокая скорость выполнения файловых операций была достигнута именно благодаря такому планированию. Однако недостатком таких систем является то, что ошибка в кодировании некоторого потока может привести к тому, что он не будет добровольно уступать процессорный ресурс, а удалить его принудительно в такой системе практически невозможно. Это приводит к «зависанию» подобных систем. Поэтому почти во всех современных ОС реализованы вытесняющие алгоритмы планирования потоков (процессов).

В основе многих вытесняющих алгоритмов планирования лежит концепция квантования (рис. 3.2), которая предполагает, что каждому потоку, находящемуся в системе, поочередно для выполнения предоставляется ограниченный непрерывный период процессорного времени, называемый квантом. Смена активного потока происходит:

1) если поток завершился и покинул систему;
2) произошла ошибка;
3) поток перешел в состояние ожидания;
4) исчерпан квант процессорного времени, отведенный данному потоку.

Поток, исчерпавший свой квант, переходит в состояние готовности и ожидает в очереди готовых потоков предоставления ему нового кванта процессорного времени, а планировщик выбирает новый поток из очереди готовых к выполнению в соответствии с заложенным алгоритмом. Кванты времени, выделяемые потокам, могут быть равными для всех потоков или могут различаться. На рис. 3.2 показан ва-

риант планирования, при котором потокам предоставляются кванты одинаковой длины q. При этом можно приблизительно оценить время ожидания потока в очереди, оно равно $q(n-1)$.

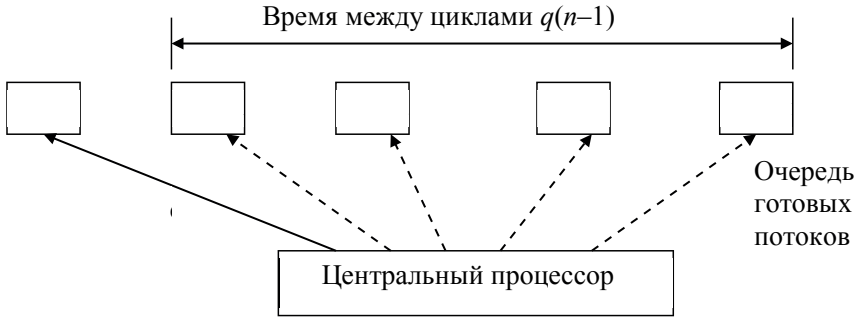


Рис. 3.2. Схема планирования, основанная на квантовании

Примечание. В алгоритмах, основанных на квантовании, какую бы цель они не преследовали (предпочтение коротких или длинных задач, компенсация недоиспользованного кванта или минимизация накладных расходов, связанных с переключениями), не используется никакой предварительной информации о задачах. При поступлении задачи на обработку, ОС не имеет предварительных сведений о длительности выполнения задачи, интенсивности ее запросов к внешним устройствам, требованиях к скорости ее выполнения и т. д. Правила обслуживания при квантовании основываются на предыстории нахождения потока в системе.

Алгоритмы планирования, основанные на приоритетах

Еще одним важным понятием, которое использует большинство вытесняющих алгоритмов планирования, является понятие приоритета.

Приоритет — это число, характеризующее степень привилегированности потока при использовании ресурсов компьютера. Это важно в отношении процессорного времени: чем выше приоритет, тем более этот поток значим для системы, получает привилегии по отношению к другим потокам и тем меньше он будет находиться в очередях, быстрее будет выполнен.

Приоритет обычно выражается целым отрицательным или положительным числом. В некоторых ОС существует правило, что приоритет потока тем выше, чем больше число, определяющее приоритет, в других системах — наоборот.

В большинстве ОС, в которых существуют потоки, приоритет потока зависит от приоритета процесса, в котором выполняется данный поток. Приоритет процесса назначает ОС при его создании. Значение приоритета содержится в описателе процесса и учитывается при установлении приоритетов потокам этого процесса.

Во многих ОС существует возможность изменения приоритетов во время жизни потока. Приоритет может изменить сам поток, для этого он обращается с соответствующим вызовом к ОС; или пользователь, для этого он выдает соответствующую команду. Также сама ОС может изменить приоритеты потоков в зависимости от того, какая ситуация складывается в системе. В этом случае приоритеты называются динамическими, в отличие от фиксированных (статических) приоритетов.

Система приоритетов в ОС Windows

В ОС Windows существует 32 уровня приоритетов и два класса потоков — это потоки реального времени и потоки с переменными приоритетами. Диапазон от 1 до 15 включительно отведен для потоков с переменными приоритетами, а от 16 до 31 — для более критичных ко времени потоков реального времени (приоритет 0 — системный уровень, зарезервированный для потока обнуления страниц (имеются в виду страницы основной памяти).

При создании процесса, ему назначается базовый приоритет — один из шести классов приоритетов:

1) Real time class (значение 24);
2) High class (значение 13);
3) Above normal class (значение 10);
4) Normal class (значение 8);
5) Below normal class (значение 6);
6) Idle class (значение 4).

В дальнейшем этот приоритет может быть изменен.

Приоритет каждого потока (базовый приоритет потока) складывается из приоритета его процесса и относительного приоритета самого потока. Есть семь относительных приоритетов потоков.

1) Normal: такой же, как и у процесса;
2) Above normal: +1 к приоритету процесса;
3) Below normal: –1;
4) Highest: +2;
5) Lowest: –2;
6) Time critical: устанавливает базовый приоритет потока для Real time класса в 31, для остальных классов в 15.
7) Idle: устанавливает базовый приоритет потока для Real time класса в 16, для остальных классов в 1.

Очевидно, что, изменяя базовый приоритет процесса, ОС может изменять и базовые приоритеты его потоков. ОС также может изменять и относительный приоритет потока в зависимости от истории его существования в системе. Схематично все это представлено на рис. 3.3.

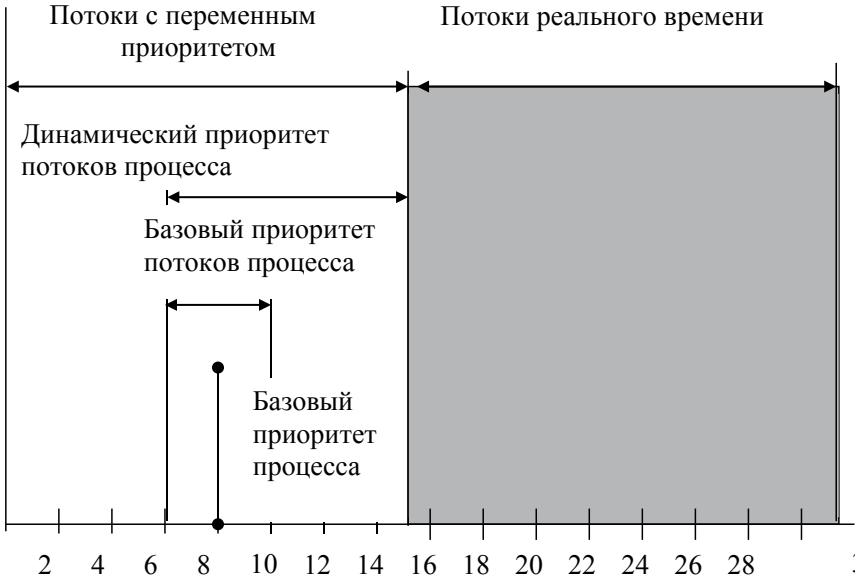


Рис. 3.3. Приоритеты процессов и потоков в ОС Windows

Основные сведения о выполняющихся процессах в Windows можно получить, используя программу **Диспетчер задач** (рис. 3.4). С помощью этой программы можно также изменять приоритеты процессов.

Также приоритет процесса можно изменить с помощью команды start из окна командного интерпретатора Windows (рис. 3.5).

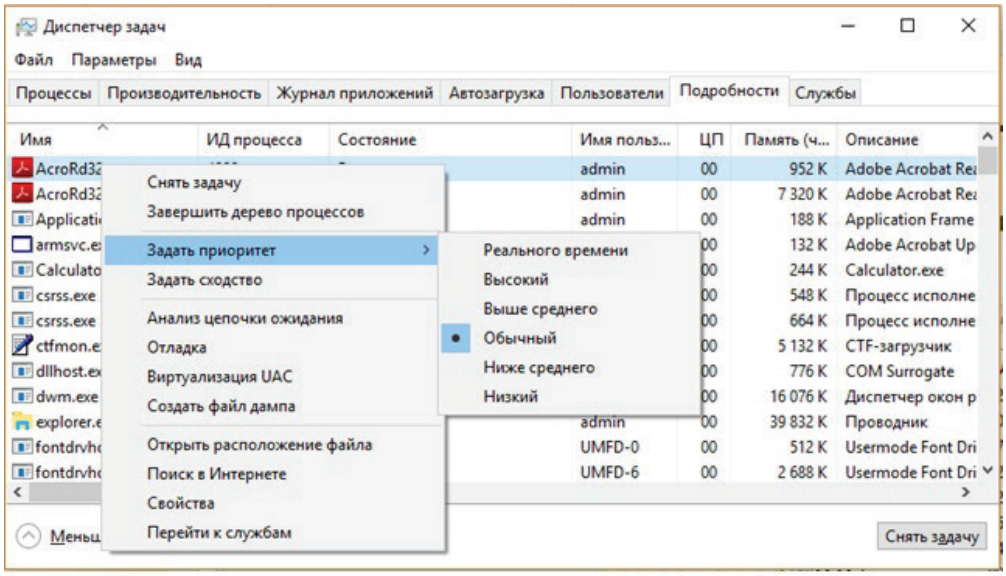


Рис. 3.4. Окно Диспетчер задач в Windows 10

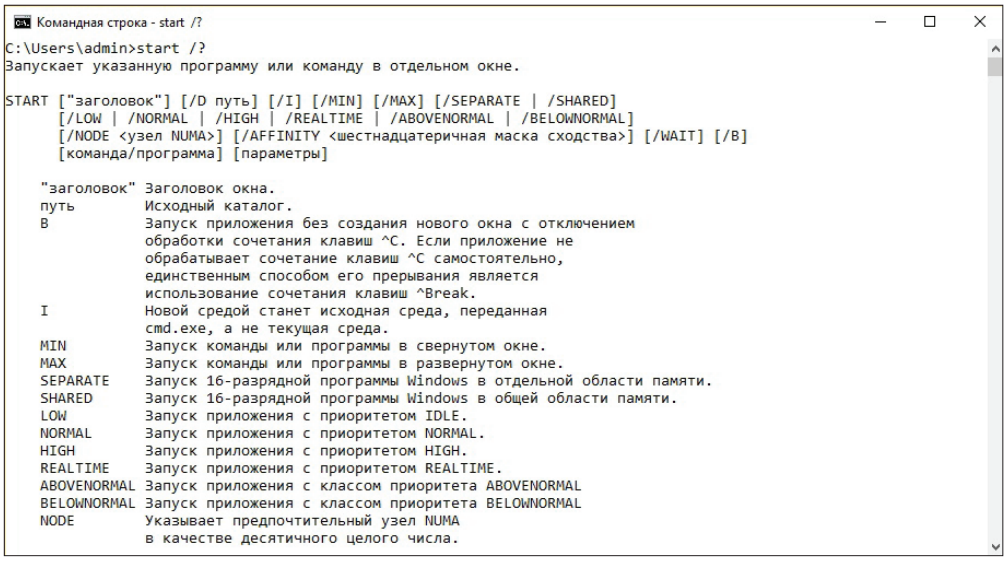


Рис. 3.5. Окно с описанием команды start

Контрольные вопросы

1. Что такое процесс с точки зрения операционной системы?
2. Приведите примеры ОС, в которых существуют одновременно понятия «процесс» и «поток».
3. В чем отличие планирования от диспетчеризации?
4. Какие существуют состояния процессов (в системах, в которых отсутствует понятие «поток»)? Перечислите все возможные смены состояний процессов.
5. Какие технологии обычно используют при планировании выполнения процессов в системе?
6. В чем особенности системы приоритетов, реализованной в Windows?

4. Управления процессами в операционной системе Linux

В системе Linux в любой момент времени существует множество процессов. Каждый запущенный процесс в ней может породить дополнительные процессы, при этом формируется дерево процессов и всегда прослеживается связь вида «дочерний — родительский» процесс.

Каждый процесс характеризуется набором атрибутов, который отличает данный процесс от всех остальных процессов. К таким атрибутам относятся:

1) идентификатор процесса (PID) — каждый процесс в системе имеет уникальный численный идентификатор;
2) идентификатор родительского процесса (PPID) — PID родительского процесса;
3) реальный и эффективный идентификаторы пользователя (UID, EUID) и группы (GID, EGID) — идентификаторы пользователя и группы, запустивших данный процесс. Эффективные идентификаторы определяют, от чьего имени был запущен процесс, т. к. в Linux существует возможность запуска исполняемых файлов от имени их владельца (или группы-владельца);
4) приоритет и nice-число — определяет степень привилегированности процесса.

Nice-число — это относительный приоритет процесса. Пользователь не может изменять абсолютный приоритет, а может изменять nice-число (при условии приобретения административных привилегий, как правило, за счет использования команд sudo или su). Nice-число задается в диапазоне от –20 до 19 (по умолчанию используется значение 0), причем значение «–20» соответствует наиболее высокому приоритету.

Установить или изменить приоритет можно используя одну из двух команд: *nice* или *renice*.

Команда *nice* позволяет задавать приоритет, с которым будет выполняться процесс после запуска. Если пользователь запустил процесс и понял, что он должен выполняться с другим приоритетом, то его можно изменить с помощью команды *renice*.

Типы процессов

Согласно классической теории ОС существует 2 вида процессов: системные и пользовательские (см. соответствующий раздел). В Linux, кроме этих двух типов, выделяют сервисные службы, или демоны (от англ. daemon). Рассмотрим эти типы процессов в ОС Linux.

Системные процессы:

1) не имеют собственных исполняемых файлов;
2) запускаются из ядра ОС;
3) стартуют при загрузке ОС и выполняются в течение всего времени ее работы;
4) находятся постоянно в ОП;
5) не интерактивны (не общаются с пользователем).

Сервисные службы (демоны):

1) выполняют внутренние функции, нужные системе (например, запуск заданий по расписанию, проверка целостности файловой системы);
2) выполняются в фоновом режиме;
3) не связаны с конкретными терминалами;
4) не интерактивны;
5) обычно стартуют в процессе загрузки системы после запуска системных процессов, запускаются пользователями с правами администратора, по запросу пользовательской программы, по сетевому запросу или по наступлению какого-либо системного события.

Пользовательские процессы:

1) запускаются из исполняемого файла пользователем;
2) могут выполняться в интерактивном или фоновом режиме;
3) срок их выполнения обычно ограничен продолжительностью сеанса работы пользователя;
4) наследуют идентификатор пользователя и, как правило, имеют соответствующие права на доступ к объектам;

5) самый важный пользовательский процесс — командный интерпретатор (оболочка или шелл), обеспечивающий диалоговый режим работы с пользователем.

Жизненный цикл процесса

Процессы в ОС Linux создаются:
1) при запуске системы;
2) действиями другого процесса;
3) действиями пользователя;
4) действиями команды batch/диспетчером задач.

Создание нового процесса начинается с системного вызова fork(), инициированного в родительском процессе. Вызов fork() создает копию исходного процесса, идентичную родителю, но имеющую следующие отличия:
1) у нового процесса свой PID;
2) PPID нового процесса равен PID родителя;
3) учетная информация нового процесса обнулена;
4) у нового процесса имеется свой собственный экземпляр дескрипторов файлов.

Процессы, выполняющие разные программы, образуются благодаря использованию системных функций семейства exec(). Эти функции отличаются форматом вызова, но в конечном итоге делают одну и ту же вещь: замещают внутри текущего процесса исполняемый код на код, содержащийся в указанном файле.

Когда процесс завершается, он вызывает системную функцию exit(), чтобы уведомить ядро о своей готовности «умереть». Параметром в функцию exit() передается код завершения — целое число, определяющее причину завершения процесса. Код завершения, равный нулю, означает, что процесс успешно выполнился.

Код завершения необходим родительскому процессу, поэтому ядро должно хранить его, пока родительский процесс не запросит его вызовом системной функции wait(). Если по каким-то причинам потомок завершил свою работу, а родительский процесс не получил сигнал, свидетельствующий об этом, то этот процесс-потомок становится зомби (переходит в состояние Zombie). Наличие таких зомби-процессов

опасно для системы, т. к. они не освобождают определенные структуры ядра, количество которых ограничено.

Пример кода запуска процесса в системе:

```
int main(int argc, char **argv)
    {
          char *name = argv[0];
          int child_pid = fork();
          if(child_pid == 0) {
          printf("Дочерний процесс");
          //exec(…);
          …
          //exit(…);
          }
    else {
          printf("Родительский процесс");
          wait(pid);
          return 0;
    }
```

Пример жизненного цикла типичного процесса в Linux представлен на рис. 4.1. Представлен процесс, который инициирован командой ls интерпретатора bash. Эта команда выдает содержание каталога, переданного ей параметром.

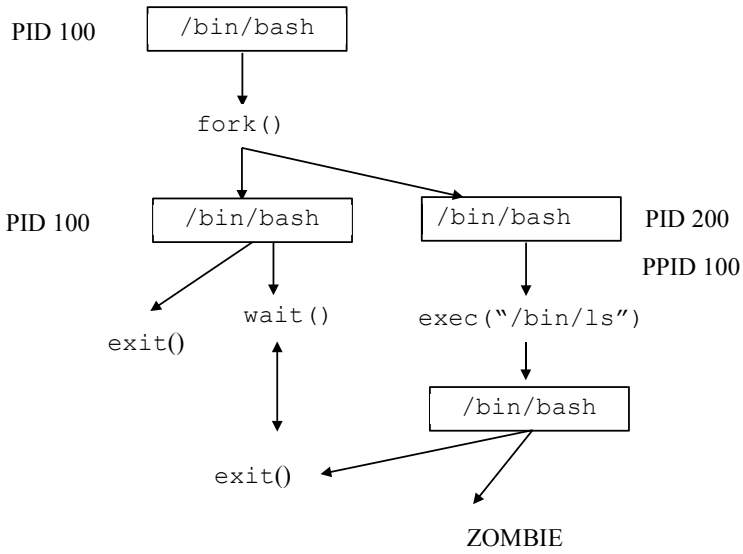


Рис. 4.1. Жизненный цикл процесса

Жизненный цикл процесса состоит из следующих этапов:

1) процесс /bin/bash (PID=100) вызывает fork();
2) создается клон /bin/bash, у которого PID=200, а PPID=100;
3) процесс с PID=200 вызывает exec(), передавая параметром полное имя файла, исполняющего команду ls;
4) процесс /bin/bash (PID=100) ждет завершения потомка — функция wait();
5) если /bin/bash (PID=100) не получает сигнала о завершении процесса (с PID=200), то этот процесс не завершается, а переходит в состояние Zombie.

Состояния процессов в системе

Возможное множество состояний процессов в системе Linux несколько отличается от теоретического (рис. 4.2).

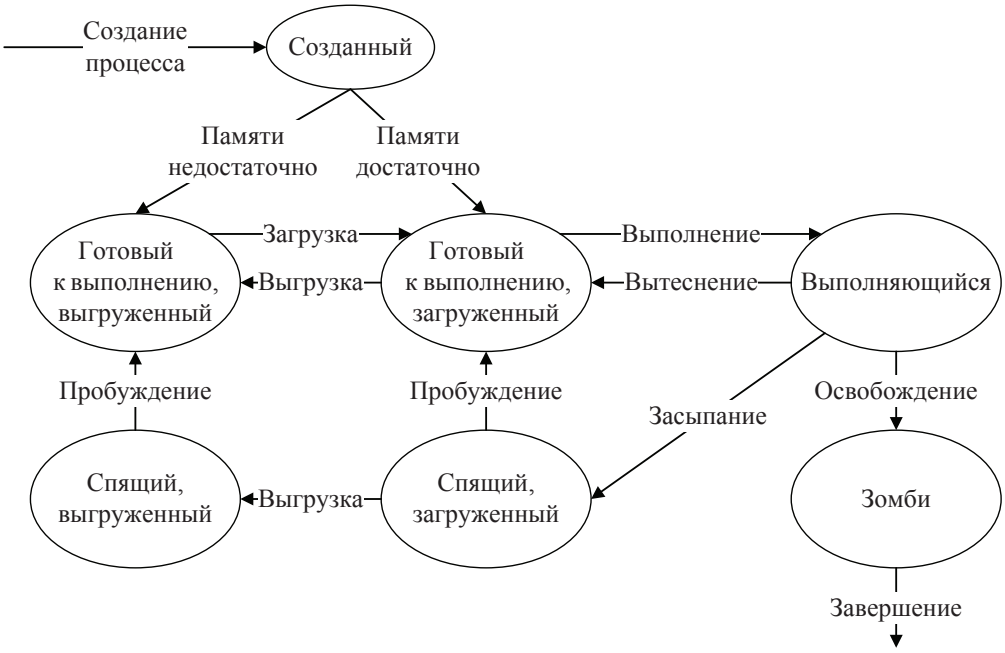


Рис. 4.2. Схема состояний процесса

Каждый запущенный процесс в любой момент времени находится в одном из следующих состояний (которое называют еще статусом процесса):

1) активен (R = Runnable) — процесс выполняется или готов к выполнению (находится состоянии готовности);
2) спит (S = Sleeping) — процесс находится в состоянии ожидания, т. е. ожидает какого-то события, прихода некоторого сигнала или освобождения некоторого ресурса;
3) задержка (D = Delay) — процесс ожидает определенного («прямого») сигнала от аппаратной части и не реагирует на другие сигналы;
4) приостановлен (T = Traced or Stopped) — процесс находится в режиме трассировки (обычно такое состояние возникает при отладке программ);
5) зомби (Z = Zombie) — это процесс, выполнение которого завершилось, но относящиеся к нему структуры ядра по каким-то причинам не освобождены.

В системе существует особый вид процессов — демоны. Данный вид процессов работает в фоновом режиме (подобно службам в Windows) без терминала и выполняет задачи для других процессов. Данный вид процессов является основным на серверных системах.

Управление процессами

Управление процессами осуществляется частью ядра, именуемой планировщиком задач, или планировщиком (scheduler). В ОС Linux реализована вытесняющая многозадачность, когда планировщик задач в замкнутом цикле передает управление следующему процессу, не запрашивая согласия на это у выполняющегося процесса.

Во время работы процесса, ядро контролирует его состояние и, в случае возникновения непредвиденной ситуации, управляет процессом с помощью посылки ему сигнала.

Сигнал — это средство общения с процессами, с помощью которого можно передать сообщения о событиях в системе. Сигнал — это способ организации взаимодействия между процессами.

Существует несколько типов сигналов. Каждый из них имеет действие, предусмотренное по умолчанию. Процесс может использовать это действие по умолчанию, задействовав обработчик сигнала, может перехватить и обработать (в некоторых случаях — игнорировать) сиг-

нал. Исключение — сигнал SIGKILL, его невозможно перехватить или игнорировать.

По умолчанию возможно несколько действий:

1) игнорировать — продолжать работу несмотря на то, что получен сигнал;
2) завершить — завершить работу процесса;
3) остановить — приостановить выполнение процесса, но не завершать его работу и не выгружать код из памяти.

В таблице представлен список некоторых сигналов, действующих в системах Linux.

Сигналы в ОС Linux

Название	Действие по умолчанию	Значение
SIGTERM	Завершить	Сигнал, посылающий предупреждение процессу, что он вскоре будет уничтожен. Этот сигнал позволяет процессу соответствующим образом подготовиться к завершению, удалив временные файлы, завершив определенные операции (транзакции) и т. д. По умолчанию команда kill отправляет этот сигнал
SIGALRM	Завершить	Сигнал, отправляемый при срабатывании установленного таймера
SIGCHLD	Игнорировать	Сигнал, посылаемый родительскому процессу по завершении его процесса-потомка
SIGFPE	Завершить	Сигнал, посылаемый в случае возникновения особых ситуаций (деление на ноль, переполнение при выполнении операций с плавающей точкой и т. д.)
SIGHUP	Завершить	Сигнал, посылаемый лидеру сеанса, связанному с управляющим терминалом, что терминал отсоединился (потеря линии). Также посылается всем процессам текущей группы при завершении выполнения лидера
SIGINT	Завершить	Сигнал, посылаемый ядром всем процессам при нажатии пользователем комбинации клавиш (<CTRL>+<C>)
SIGKILL	Завершить	Сигнал, прерывающий выполнение процесса. Его нельзя перехватить или игнорировать
SIGQUIT	Завершить	Сигнал, посылаемый всем процессам текущей группы при нажатии пользователем комбинации клавиш <CTRL>+<\>

Окончание таблицы

Название	Действие по умолчанию	Значение
SIGSTOP	Остановить	Сигнал, посылаемый всем процессам текущей группы при нажатии пользователем комбинации клавиш <CTRL>+<Z>. Получение сигнала вызывает останов выполнения процесса

Послать сигнал можно, выполнив команду kill. Команда имеет следующий синтаксис:

```
kill [-номер сигнала] PID
```

где PID — идентификатор процесса.

Инструменты работы с процессами

Для просмотра запущенных процессов в ОС Linux можно использовать несколько инструментов, среди них утилиты командной строки, такие как ps (рис. 4.3) и top (рис. 4.4), и графические инструменты (как «Системный монитор» из Ubuntu, окно которого показано на рис. 4.5).

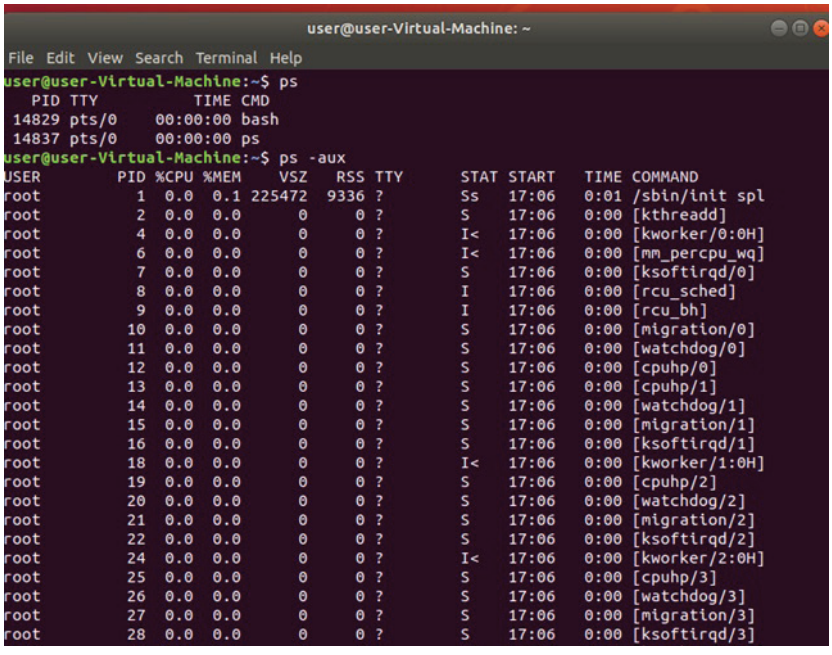
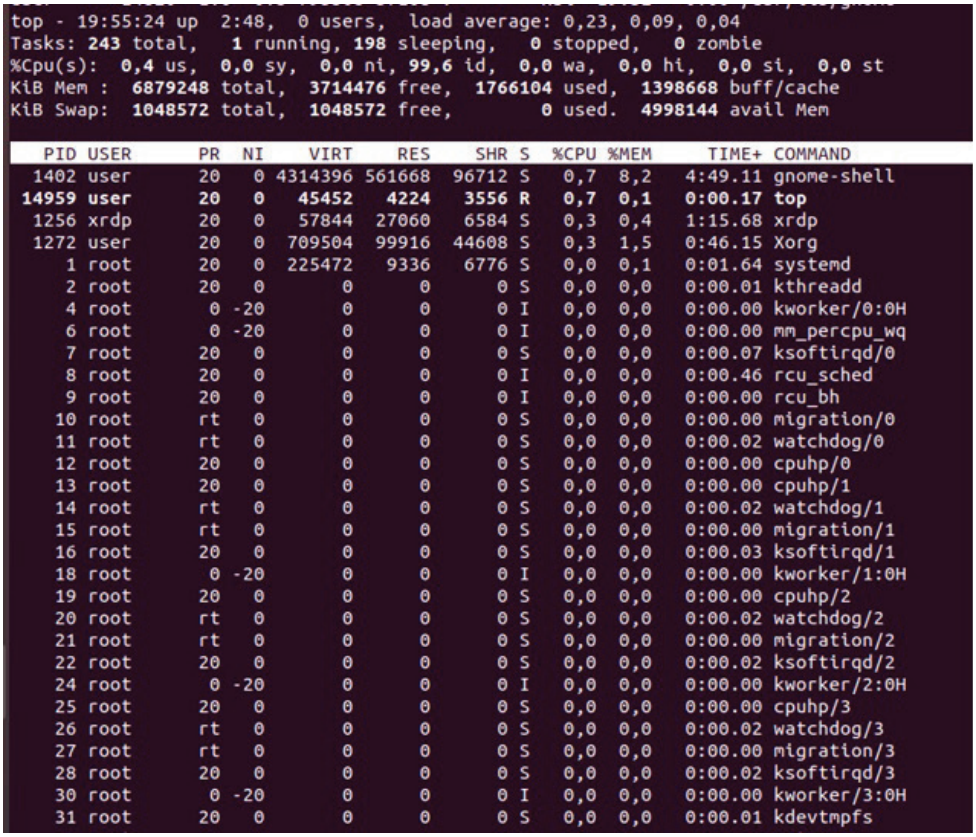


Рис. 4.3. Выполнение команды ps в окне Терминала



```
top - 19:55:24 up  2:48,  0 users,  load average: 0,23, 0,09, 0,04
Tasks: 243 total,   1 running, 198 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,4 us,  0,0 sy,  0,0 ni, 99,6 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  6879248 total,  3714476 free,  1766104 used,  1398668 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  4998144 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1402 user      20   0 4314396 561668  96712 S   0,7  8,2   4:49.11 gnome-shell
14959 user      20   0   45452   4224   3556 R   0,7  0,1   0:00.17 top
 1256 xrdp      20   0   57844  27060   6584 S   0,3  0,4   1:15.68 xrdp
 1272 user      20   0  709504  99916  44608 S   0,3  1,5   0:46.15 Xorg
    1 root      20   0  225472   9336   6776 S   0,0  0,1   0:01.64 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0,0  0,0   0:00.07 ksoftirqd/0
    8 root      20   0       0      0      0 I   0,0  0,0   0:00.46 rcu_sched
    9 root      20   0       0      0      0 I   0,0  0,0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S   0,0  0,0   0:00.02 watchdog/0
   12 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0,0  0,0   0:00.02 watchdog/1
   15 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/1
   16 root      20   0       0      0      0 S   0,0  0,0   0:00.03 ksoftirqd/1
   18 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/1:0H
   19 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/2
   20 root      rt   0       0      0      0 S   0,0  0,0   0:00.02 watchdog/2
   21 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/2
   22 root      20   0       0      0      0 S   0,0  0,0   0:00.02 ksoftirqd/2
   24 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/2:0H
   25 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/3
   26 root      rt   0       0      0      0 S   0,0  0,0   0:00.02 watchdog/3
   27 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/3
   28 root      20   0       0      0      0 S   0,0  0,0   0:00.02 ksoftirqd/3
   30 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/3:0H
   31 root      20   0       0      0      0 S   0,0  0,0   0:00.01 kdevtmpfs
```

Рис. 4.4. Пример работы утилиты top

Вывод утилиты ps зависит от переданных ей ключей, что очевидно из рис. 4.3. Она может вывести:

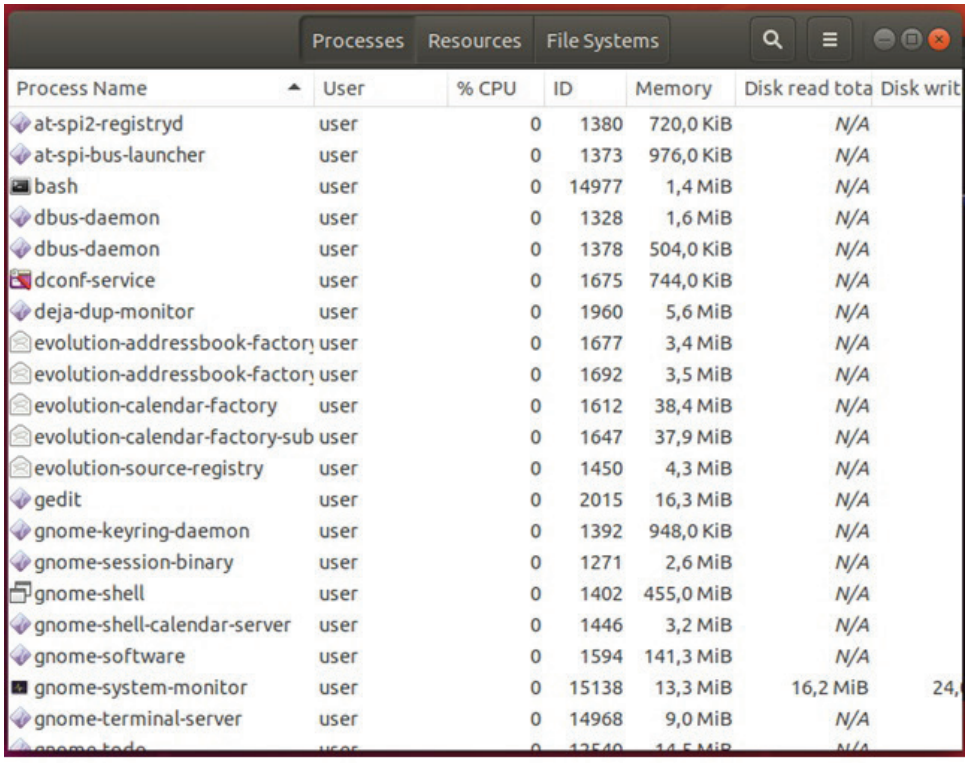
1) PID — ID процесса;
2) TTY — управляющий терминал процесса;
3) STAT — статус (состояние) процесса;
4) TIME — потребовавшееся на данный момент время расчета;
5) COMMAND — команду, запустившую процесс (длинные имена даются в сокращенном виде);
6) USER — имя пользователя, от имени которого выполняется процесс;
7) %CPU — долю имеющихся расчетных возможностей, использованную в последнее время для решения текущей задачи;
8) %MEM — долю задействованной на данный момент ОП;
9) SIZE — размер занятой памяти (в килобайтах);

10) RSSResidental Size — занятую ОП (на данный момент). Выгруженные страницы не учитываются;
11) START — точное время запуска процесса;
12) PPID — ID родительского процесса.

При выполнении top (рис. 4.4) выдаются сведения о процессах в реальном времени. Над данными о запущенных процессах отображаются:

1) астрономическое время;
2) время, прошедшее с момента запуска системы;
3) число пользователей в системе;
4) число запущенных процессов и число процессов, находящихся в разных состояниях;
5) данные об использовании ЦП, ОП.

Ниже на экране таблица, характеризующая отдельные процессы. Данные, выдаваемые этой утилитой, такие же, как и ps. Различие этих утилит в том, что ps дает моментальный снимок процессов, а top — утилита реального времени, содержимое ее окна обновляется каждые 5 с.



Process Name	User	% CPU	ID	Memory	Disk read tota	Disk writ
at-spi2-registryd	user	0	1380	720,0 KiB	N/A	
at-spi-bus-launcher	user	0	1373	976,0 KiB	N/A	
bash	user	0	14977	1,4 MiB	N/A	
dbus-daemon	user	0	1328	1,6 MiB	N/A	
dbus-daemon	user	0	1378	504,0 KiB	N/A	
dconf-service	user	0	1675	744,0 KiB	N/A	
deja-dup-monitor	user	0	1960	5,6 MiB	N/A	
evolution-addressbook-factory	user	0	1677	3,4 MiB	N/A	
evolution-addressbook-factory	user	0	1692	3,5 MiB	N/A	
evolution-calendar-factory	user	0	1612	38,4 MiB	N/A	
evolution-calendar-factory-sub	user	0	1647	37,9 MiB	N/A	
evolution-source-registry	user	0	1450	4,3 MiB	N/A	
gedit	user	0	2015	16,3 MiB	N/A	
gnome-keyring-daemon	user	0	1392	948,0 KiB	N/A	
gnome-session-binary	user	0	1271	2,6 MiB	N/A	
gnome-shell	user	0	1402	455,0 MiB	N/A	
gnome-shell-calendar-server	user	0	1446	3,2 MiB	N/A	
gnome-software	user	0	1594	141,3 MiB	N/A	
gnome-system-monitor	user	0	15138	13,3 MiB	16,2 MiB	24,
gnome-terminal-server	user	0	14968	9,0 MiB	N/A	

Рис. 4.5. Окно программы «Системный монитор»

Контрольные вопросы

1. Какими атрибутами определяется процесс в Linux?
2. Из каких основных стадий состоит жизненный цикл процесса в Linux?
3. Зачем нужны сигналы в системе?
4. Какие команды и программы в системах Linux выдают сведения о процессах?
5. В чем своеобразие системы при назначении состояний процессам? Сравните это с классической схемой.

5. Подсистема управления основной памятью

Организация и управление основной, или первичной, оперативной, памятью (ОП) вычислительной машины (англ. *memory*, точнее, *RAM — Random Access Memory*) — один из самых важных факторов, определяющих построение ОС. Для выполнения программ или обращения к данным необходимо, чтобы они предварительно были размещены в ОП. Вторичная, или внешняя, память (англ. *storage*) — это, как правило, накопители на магнитных дисках, накопители на CD- (DVD-) дисках, накопители на магнитных лентах и т. д. В отличие от внешней памяти, ОП для сохранения информации требуется электропитание.

Иерархия запоминающих устройств

Всю имеющуюся в компьютере память можно рассматривать как единую иерархическую систему запоминающих устройств (ЗУ), которые отличаются такими характеристиками, как размер, среднее время доступа к данным и стоимость единицы хранения.

Часто выделяют 4 основных (укрупненных) уровня иерархии [19]:

1) внутреннюю память процессора (регистры, организованные в регистровый файл, и кэш процессора);
2) ОП системы и вспомогательные карты памяти;
3) накопители с «горячим» доступом, или вторичную компьютерную память — это жесткие диски и твердотельные накопители, не требующие длительных подготовительных действий для получения данных;
4) накопители, требующие переключения носителей (Off-line bulk storage), или третичную, память. Сюда относятся магнитные ленты, ленточные и дисковые библиотеки, требующие длительной

перемотки либо механического (или ручного) переключения носителей информации.

В соответствии с рис. 5.1 можно сделать следующие выводы: чем больше объем устройства, тем менее быстродействующим оно является; стоимость же хранения данных в расчете на один бит увеличивается с ростом быстродействия устройств.

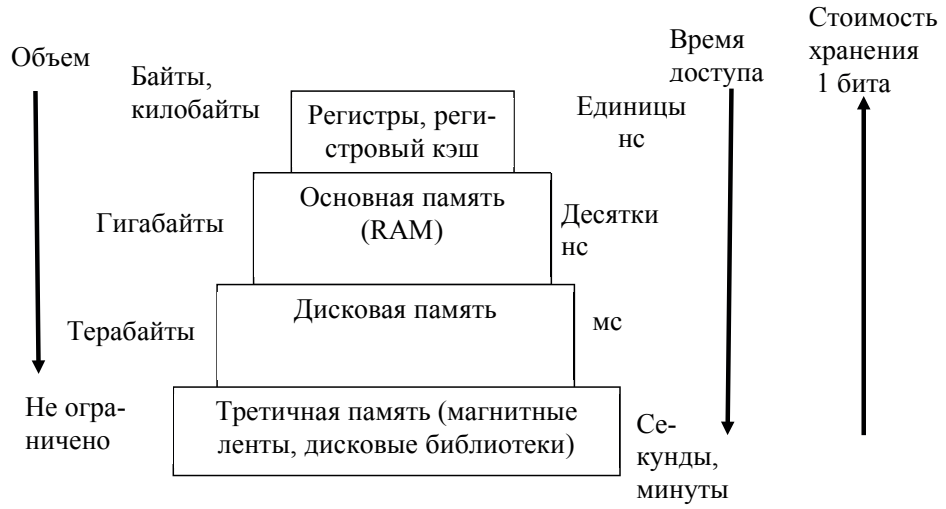


Рис. 5.1. Иерархия устройств памяти (обобщенная схема)

Функции ОС по управлению основной памятью

В прошлом ОП представляла собой самый дорогостоящий ресурс. В связи с этим она требовала особого внимания со стороны разработчиков систем — необходимо было обеспечить как можно более эффективное использование столь дорогостоящего ресурса. В первых ЭВМ главной задачей считалось оптимальное использование основной памяти благодаря рациональной организации и управлению ею.

Под **организацией памяти** обычно понимают то, каким образом представляется и используется ОП: сколько программ размещается в памяти и, если их несколько, способ их размещения — одинаковое количество ячеек выделять каждой программе или только столько, сколько запрашивает; как с течением времени перераспределять память и т. д.

Функциями ОС по управлению ОП в мультипрограммной системе являются:

1) отслеживание свободной и занятой памяти;
2) выделение памяти процессам и освобождение памяти по завершении процессов;
3) вытеснение кодов и данных из ОП на диск (полное или частичное), когда физически размеры основной памяти недостаточны для размещения в ней всех процессов, и возвращение их в ОП, когда в ней освобождается место;
4) настройка адресов процесса на конкретную область физической памяти;
5) защита областей памяти, выделенных одному процессу от доступа другого процесса (эта функция реализуется ОС в т. ч. аппаратными средствами).

Стратегии управления памятью

Стратегии управления памятью должны быть направлены на наиболее эффективное использование ее ресурсов. Они делятся на следующие категории:

1) стратегии выборки (загрузки);
   - стратегии выборки по запросу (по требованию);
   - стратегии упреждающей выборки;
2) стратегии размещения;
3) стратегии замещения.

Стратегии выборки должны определять, когда следует разместить очередной блок программы и (или) данных в ОП. Выборка по запросу предполагает, что очередной блок кода или данных загружается в ОП, когда его запрашивает выполняющийся процесс. Упреждающая выборка предполагает возможность прогнозирования, какие данные могут потребоваться в ближайшее время, и загрузку их заранее.

Стратегии размещения должны определить, куда (место ОП) следует помещать поступающий блок кода и (или) данных. Обычно это стратегии первого подходящего, наиболее подходящего или наименее подходящего по размеру свободного участка памяти.

Стратегии замещения ставят своей целью определить, что (какой блок кода и (или) данных) нужно изъять из ОП, чтобы освободить место для записи поступающих кодов и (или) данных.

Типы адресов

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются символьные имена, виртуальные адреса и физические адреса, как показано на рис. 5.3.

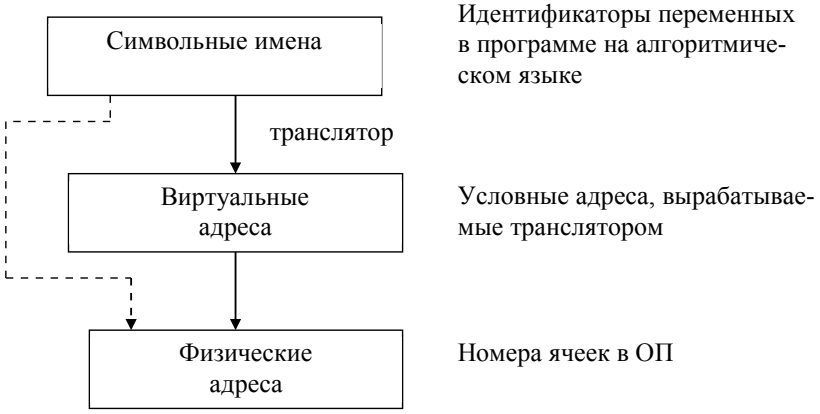


Рис. 5.2. Типы адресов

Символьные имена присваивает пользователь, создавая переменные, когда пишет программу на алгоритмическом языке. Виртуальные адреса вырабатывает транслятор, который переводит программу на машинный язык. Поскольку во время трансляции обычно не известно, в какое место ОП будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, отсчитывая их от нулевого адреса. Физические адреса соответствуют номерам ячеек ОП, в которых будут размещаться переменные и команды этой программы.

Виртуальным адресным пространством процесса называется совокупность всех виртуальных адресов этого процесса. Диапазон возможных адресов виртуального пространства у всех процессов, работающих с одной и той же ОС, одинаков. Например, в 32-разрядной ОС этот диапазон задается границами 00000000 и FFFFFFFF (в шестнадцатеричной системе счисления).

Алгоритмы распределения памяти

При распределении ОП между процессами возникает множество вопросов, например: следует ли назначать каждому процессу одну непрерывную область в ОП или можно выделять адреса некоторыми пулами; должны ли части программы, загруженные в память, находиться на одном месте в течение всего периода выполнения процесса или можно их время от времени перемещать; что делать, если программа не помещается в имеющуюся свободную память?

Разные ОС по-разному отвечают на эти и другие вопросы по управлению памятью. Далее будут рассмотрены наиболее общие подходы к распределению памяти, которые были характерны для разных этапов истории ОС. Некоторые из них сохранили актуальность и используются сегодня, другие представляют только познавательный интерес.

На рис. 5.4 все алгоритмы распределения памяти разделены на два основных класса: алгоритмы, в которых используется внешняя память для перемещения в нее сегментов данных и кода, и алгоритмы, в которых она не используется.

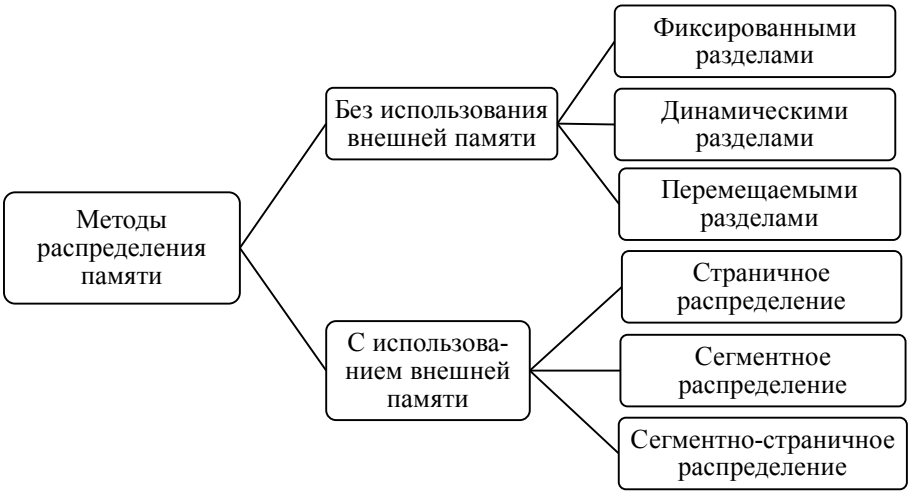


Рис. 5.3. Классификация методов распределения памяти

Наиболее простой способ управления ОП состоит в том, что память разбивается на несколько участков с фиксированным объемом, их называют разделами. Такое разбиение может быть выполнено вруч-

ную оператором (или самой системой) во время старта системы. После этого границы разделов остаются постоянными. Очередная задача, поступившая на выполнение, помещается либо в общую очередь готовых процессов, либо в отдельную очередь к разделу.

При простоте реализации, которая является очевидным достоинством системы, она имеет существенный недостаток — отсутствие гибкости. В каждом разделе может выполняться только один процесс, поэтому уровень мультипрограммирования ограничен количеством созданных разделов. Еще один недостаток заключается в том, что невозможно выполнить 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. Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени в основном благодаря небольшим затратам на реализацию. Детерминированность вычислительного процесса систем реального времени (заранее известен набор выполняемых задач, их требования к памяти, а иногда и моменты запуска) компенсирует недостаточную гибкость данного способа управления памятью.

В случае распределения памяти динамическими разделами, память машины не делится на разделы заранее. Каждой вновь поступившей на выполнение задаче на этапе создания процесса выделяется вся необходимая память (если такого объема памяти в данный момент нет, то задача не принимается на выполнение и процесс для нее не создается). По завершении процесса память освобождается, и на это место может быть загружен другой процесс. Таким образом, в момент времени *t* память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

По сравнению с методом распределения памяти фиксированными разделами, данный метод обладает большей гибкостью, но имеет серьезный недостаток — склонность к фрагментации памяти. Под **фрагментацией** понимается наличие значительного числа несмежных между собой участков свободной памяти, которые имеют небольшой размер (это и есть фрагменты). Размер каждого из фрагментов недостаточен для того, чтобы хотя бы одна из готовых к выполнению программ могла поместиться в нем, при этом суммарный объем фрагментов может значительно превышать потребности любой из этих программ.

Распределение памяти динамическими разделами лежит в основе подсистем управления памятью многих мультипрограммных ОС 1960–70-х гг., например такой, как OS/360.

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в одном направлении таким образом, чтобы вся свободная память объединилась в одну свободную область. В результате этого у ОС появляется еще одна функция по управлению ОП — перемещение содержимого разделов из одного места памяти в другое с одновременной корректировкой имеющейся в системе таблицы свободных и занятых областей памяти. Эту процедуру называют сжатием, хотя чаще употребляется другой термин — «сбор мусора». Хотя процедура сжатия и приводит к более эффективному использованию памяти, она значительно расходует ресурсы системы, что перевешивает ее достоинства.

Свопинг и виртуальная память

Программа должна быть загружена в ОП для того, чтобы она могла выполняться. Только тогда процессор сможет получать команды и выполнять определенные в них действия. Объем ОП компьютера сказывается на характере и скорости протекания вычислительного процесса. Он определяет число одновременно работающих процессов и размеры их адресных пространств.

Если необходимо выполнять одновременно большое количество задач, требуется иметь достаточно большой объем ОП. В условиях, когда для обеспечения приемлемого уровня мультипрограммирования имеющейся ОП недостаточно, было предложено так организовать вычислительный процесс, чтобы данные и коды некоторых процессов целиком или частично временно выгружались на диск.

В режиме мультипрограммирования, наряду с активным (находящимся в состоянии выполнения) процессом, имеются процессы, находящиеся в состояниях ожидании или готовности (т. е. приостановленные процессы). Коды и данные таких процессов могут быть временно вытеснены на диск. Несмотря на то что коды и данные процесса в какой-то момент времени отсутствуют в ОП, ОС знает о его существовании и учитывает это при планировании процессорного вре-

мени и других системных ресурсов. К тому моменту, когда процесс выбран для загрузки на процессор, его коды и данные возвращаются с диска в ОП. Если при этом оказывается, что свободного места в ОП не хватает, то на диск выгружаются данные и коды какого-то другого процесса. Такая подмена (виртуализация) ОП дисковой памятью позволяет повысить уровень мультипрограммирования, при этом объем ОП компьютера не столь строго ограничивает количество одновременно выполняемых процессов, поскольку суммарный объем памяти, занятый этими процессами, может существенно превосходить объем ОП, имеющейся на данном ПК.

«**Виртуальным** называется **ресурс**, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает» [4]. В данном случае в распоряжение программиста предоставляется виртуальная ОП, размер которой значительно больше объема ОП, реально установленной в ПК. Пользователь пишет программу, а транслятор, используя виртуальные адреса, переводит ее в машинные коды так, как будто в распоряжении программы имеется однородная ОП большого объема. В реальности коды и данные, используемые программой, хранятся во вторичной, дисковой, памяти и только при необходимости загружаются в физическую ОП. В этом случае становится очевидным, что работа такой ОП происходит значительно медленнее.

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

1) размещение данных в запоминающих устройствах разного типа, например, часть кодов программы в ОП, а часть — на диске;
2) выбор образов процессов или их частей для перемещения из ОП на диск и обратно;
3) перемещение по мере необходимости данных между ОП и диском;
4) преобразование виртуальных адресов в физические.

Очень важно то, что все действия по организации взаимодействия диска и ОП осуществляются ОС и аппаратурой процессора без участия пользователя и никак не отражаются на работе приложений.

Виртуализация памяти может быть осуществлена на основе двух различных подходов:

1) свопинга (*swapping*), или подкачки — образы процессов выгружаются на диск и возвращаются в ОП целиком;
2) виртуальной памяти (*virtual memory*) — между ОП и диском перемещаются части (сегменты, страницы) образов процессов.

Свопинг является частным случаем виртуальной памяти и представляет собой более простой способ совместного использования ОП и диска. Однако этой технологии присуща избыточность: для выполнения процесса не нужно загружать в ОП все его коды и данные, вполне достаточно загрузить небольшую часть кода, которая содержит нужную для выполнения инструкцию и те данные, которые использует эта инструкция, а также предоставить место для размещения сегмента стека. Подобным же образом не требуется выгружать другой процесс на диск целиком для освобождения памяти — часто достаточно вытеснить на диск только часть его кода и данных. Перемещение избыточных объемов информации замедляет работу всей системы и приводит к неэффективному использованию памяти. Более того, системы, использующие свопинг, имеют еще один недостаток: они не способны работать с процессами, виртуальные адресные пространства которых превышают имеющуюся в наличии свободную ОП.

Из-за указанных недостатков свопинг как основной механизм управления памятью не используется в современных версиях ОС. На смену ему пришел более эффективный механизм виртуальной памяти, основная идея которого состоит в том, что при нехватке места в физической ОП, данные и коды процесса выгружаются на диск частично.

Примечание. В настоящее время термин «свопинг» часто используется для определения процесса перемещения данных между ОП и диском, в т. ч. и в системах виртуальной памяти.

Основной проблемой систем виртуальной памяти является преобразование виртуальных адресов в физические, поскольку коды и данные процессов многократно меняют свое местоположение в ОП. Решение этой проблемы зависит от того, какой способ структуризации виртуального адресного пространства принят в данной системе управления памятью.

Реализацию систем виртуальной памяти можно разделить на три класса:

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 пространства 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 в этом классе является страница. Такой способ управления памятью объединяет в себе элементы двух первых классов.

Для хранения сегментов и страниц на диске отводится либо специальная область (дисковый раздел, часто называемый swap-разделом), либо специальный файл, который называют файлом подкачки. Другое популярное название этого файла — страничный файл (*page file*, или *paging file*), его текущий размер является важным системным параметром, который оказывает влияние на возможности ОС: чем он больше, тем больше процессов может одновременно выполняться в ОС (при фиксированном размере ОП). Однако необходимо понимать, что увеличение числа одновременно работающих процессов за счет увеличения размера страничного файла замедляет их работу, т. к. значительная часть времени при этом тратится на перекачку кодов и данных из ОП на диск и обратно.

Пример окна конфигурирования параметров виртуальной памяти (в т. ч. файла подкачки) в системе Windows 10 показан на рис. 5.4.

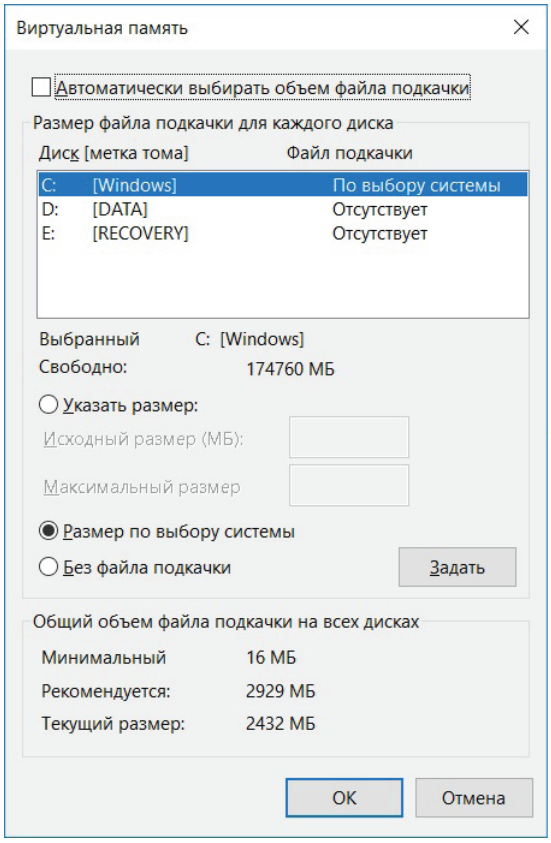


Рис. 5.4. Окно конфигурирования виртуальной памяти в Windows 10

Страничное распределение

Наиболее часто используемым механизмом распределения памяти в современных компьютерах является страничное распределение (рис. 5.5). Рассмотрим его подробнее.

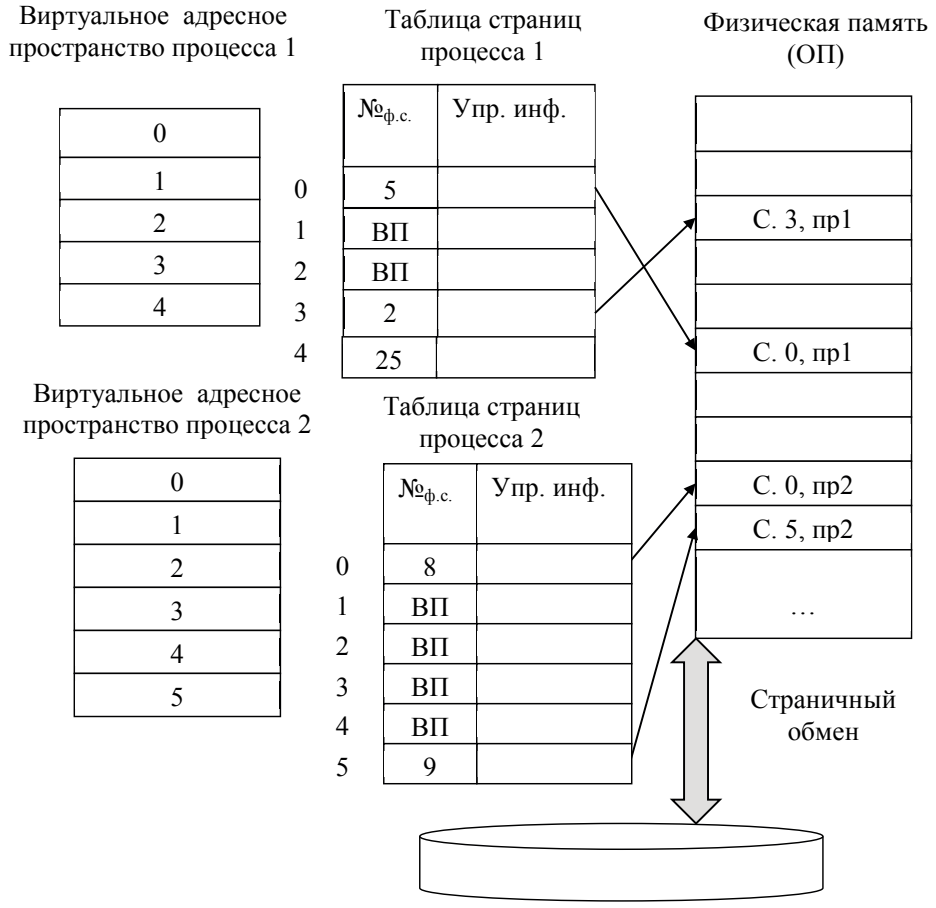


Рис. 5.5. Схема работы подсистемы страничной виртуальной памяти

Виртуальное адресное пространство каждого процесса делится на части одинакового фиксированного размера. Такая область называется виртуальной страницей (*virtual page*).

Вся физическая ОП машины делится на части такого же размера, они называются физическими страницами, или страничными фреймами. Страница поддерживается аппаратно — процессором, поэто-

му ее размер зависит от типа процессора. В процессорах Intel Pentium размер страницы равен 4 Кбайтам.

При создании процесса, в ОП загружаются несколько его виртуальных страниц (первые страницы кодового сегмента и сегмента данных). Смежные виртуальные страницы могут располагаться в несмежных физических страницах. Для каждого процесса подсистема виртуальной памяти создает таблицу страниц, которая представляет собой информационную структуру, в которой отдельное поле соответствует отдельной виртуальной странице.

Запись таблицы, называемая дескриптором страницы, включает следующую информацию:

1) номер физической страницы, в которую загружена данная виртуальная страница;
2) признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;
3) признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
4) признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Все описанные признаки в большинстве моделей современных процессоров устанавливаются аппаратно самим процессором. Информация из таблиц страниц используется при решении вопроса о необходимости перемещения страницы между памятью и диском, а также для преобразования виртуального адреса в физический. Таблицы страниц размещены в ОП. Адрес таблицы страниц включается в состав информации, определяющей соответствующий процесс. При запуске процесса, адрес его таблицы страниц загружается в специальный регистр процессора.

При обращении по некоторому адресу выполняется поиск номера виртуальной страницы, содержащей этот адрес; по данному номеру виртуальной страницы определяется нужный элемент таблицы страниц, из которого извлекается определяющая страницу информация. Затем анализируется признак присутствия: если данная виртуальная страница находится в ОП, то выполняется преобразование виртуального адреса в физический, т. е. виртуальный адрес замещается физическим адресом, взятым из записи таблицы.

Если в данный момент нет нужной виртуальной страницы в ОП — она выгружена на диск, то ОС инициирует страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, выбирается другой процесс из очереди готовых процессов. Одновременно с этим запускается программа обработки страничного прерывания, которая ищет на диске требуемую виртуальную страницу, а затем пытается загрузить ее в ОП. Если в физической ОП имеется свободная страница, то загрузка выполняется немедленно; если такой страницы нет, то решается вопрос о том, какую страницу следует выгрузить из ОП. Для этого используется принятая в данной системе стратегия замещения страниц.

После того как выбрана страница, которая должна быть освобождена, обнуляется ее бит присутствия и анализируется признак модификации. Если освобождаемая страница в интервал последнего своего пребывания в ОП была изменена, то ее новая версия будет переписана на диск. Если нет, то, поскольку на диске есть предыдущая копия этой виртуальной страницы, никакой записи на диск не происходит. Физическая страница объявляется свободной.

В некоторых системах из соображений безопасности физическая страница перезаписывается некоторым кодом для того, чтобы невозможно было прочитать ее содержимое.

При страничном распределении, виртуальный адрес представляется как пара значений (n_v, s_v), где n_v есть порядковый номер виртуальной страницы процесса (при нумерации страниц с 0), а s_v — смещение от начала виртуальной страницы. Физический адрес может быть представлен в подобном формате как (n_p, s_p), где n_p — номер физической страницы, а s_p — смещение от начала физической страницы. Необходимо отобразить (n_v, s_v) в (n_p, s_p).

Схема отображения виртуального адреса в физический основана на двух основных свойствах страничной памяти:

1) объем страницы выбирается равным степени двойки (2^m). Из этого следует, что смещение можно получить, выделив m младших разрядов в двоичной записи адреса. Оставшиеся старшие разряды адреса представляют собой двоичную запись номера страницы (это верно как для виртуальной, так и для физической страниц). Например, если размер страницы 1 Кбайт (2^{10}), имеется адрес $11\mathbf{1000111011}_2$, можно определить, что он принадлежит странице, номер которой в двоичном выражении равен 11_2 и смещен от ее начала на 1000111011_2 байт;

2) в пределах страницы непрерывная последовательность виртуальных адресов однозначно отображается в непрерывную последовательность физических адресов, следовательно, смещения в виртуальном и физическом адресах s_v и s_p должны быть равны между собой.

Из вышесказанного следует алгоритм преобразования виртуального адреса в физический, действие которого схематично представлено на рис. 5.6. В младшие разряды физического адреса, которые соответствуют смещению, записывают значения младших разрядов из виртуального адреса. Старшие разряды физического адреса, соответствующие номеру физической страницы, определяются по таблице страниц.

Рис. 5.6. Схема преобразования виртуального адреса в физический при страничной организации памяти

При обращении по некоторому виртуальному адресу, аппаратно процессором выполняются следующие действия:

1) из специального регистра процессора извлекается адрес A_T — таблицы страниц этого процесса; используя адрес таблицы страниц,

номер виртуальной страницы n_v, извлеченный из виртуального адреса, и длину записи в таблице страниц L, которая является системной константой, определяется адрес нужной записи в таблице страниц: $a = A_T + (n_v \times L)$;

2) из найденной записи извлекается номер соответствующей физической страницы — n_p;
3) к номеру физической страницы приписывается смещение n_v (помещается в младшие разряды физического адреса).

Каждый оператор в программе требует нескольких обращений к памяти, при этом происходит либо преобразование виртуального адреса в физический, либо обработка страничного прерывания. Время выполнения этих операций значительно влияет на общую производительность вычислительной системы, поэтому преобразования адресов осуществляется аппаратно — во всех процессорах предусмотрен механизм преобразования виртуального адреса в физический.

Частота страничных прерываний также значительно влияет на производительность системы, а она в свою очередь зависит от принятой в данной системе стратегии выбора страниц для откачки и подкачки в ОП. При неправильно выбранной стратегии могут возникать ситуации, при которых система тратит большую часть времени именно на эти операции.

Стратегии управления страничной виртуальной памятью

При выборе страницы на выгрузку могут быть использованы различные критерии, основная идея которых состоит в том, что на диск необходимо выгрузить страницу, к которой в будущем не будет обращений дольше всего. Поскольку предсказать ход вычислительного процесса достаточно сложно, невозможно точно определить номер искомой страницы. Поэтому используются различные эмпирические критерии, которые основаны на концепции инерционности вычислительного процесса, например из того, что страница не использовалась достаточно долго, делается вывод, что она не будет использоваться и в ближайший промежуток времени.

Рассмотрим наиболее часто используемые стратегии замещения. Одним из часто используемых критериев выбора страницы на выгрузку на диск является число обращений к ней за некоторый недавний интервал времени, для чего используется переменная — счетчик.

Когда не хватает памяти для размещения новых страниц, ОС находит страницу с наименьшим значением переменной — счетчика. Для того чтобы учитывать обращения только за последнее время, ОС с определенной периодичностью обнуляет значения счетчиков.

Рассмотрим стратегию выборки страниц. Интенсивность страничного обмена может можно снизить за счет применения упреждающей выборки, которая предполагает, что при возникновении страничного прерывания, в память загружается не только страница, содержащая адрес обращения, а несколько соседних страниц. Используется следующее эмпирическое правило: если обращение произошло по некоторому адресу, то велика вероятность того, что следующие обращения произойдут по соседним адресам (такое свойство определяют как пространственную локальность). Хотя некоторые современные ОС (например, версии UNIX) используют выборку по запросу.

Что касается стратегии размещения виртуальной страницы в памяти, то здесь, как правило, используется первая свободная физическая страница. Если незанятого места в ОП нет, то ОС выгружает некоторые страницы на диск, используя стратегию замещения (о чем говорилось ранее в этом разделе).

Определение размера страницы

Другим важным фактором повышения производительности системы является выбор оптимального размера страницы. Каким он должен быть? С одной стороны, для уменьшения частоты страничных прерываний, его следовало бы увеличить. С другой стороны, если размер страницы большой, то велик и объем данных, перемещаемых между ОП и диском. Кроме того, увеличивается и размер фиктивной области в последней виртуальной странице каждого процесса, что приводит к непроизводительному использованию памяти. Из этого следует, что выбор размера страницы является многокритериальной оптимизационной задачей.

На практике разработчики ОС и аппаратуры ограничиваются неким приближенным решением, которое может быть использовано для большого числа вычислительных систем. Обычный размер страницы составляет несколько килобайтов, наиболее распространенные процессоры компании Intel, и ОС, установленные на них, поддерживают страницы размером 4 096 байт (4 Кбайт).

Размер страницы также влияет на число записей в таблицах страниц. Чем меньше размер страницы, тем больше размер каждой из таблиц страниц процессов, тем больше памяти используется для их хранения. Учитывая, что в современных процессорах процессу выделяется, как правило, не меньше 4 Гбайт (2^{32}) адресного пространства, то при размере страницы 4 Кбайт и длине записи в таблице 4 байт, для хранения одной таблицы страниц может потребоваться 4 Мбайт памяти. Одновременно с этим выполняется множество процессов, тогда для определения размера памяти, нужной для хранения данных таблиц, 4 Мбайт нужно еще умножить на число процессов.

Выходом в такой ситуации является хранение в памяти только части таблицы страниц, а именно той, которая активно используется в данный период времени, остальное содержимое можно временно вытеснять из ОП.

Контрольные вопросы

1. Какие основные функции выполняет подсистема управления основной памятью?
2. Что понимается под организацией основной памяти и какие основные стратегии при этом используются?
3. Какие вы знаете основные типы организации основной памяти? В чем их достоинства и недостатки?
4. Как можно определить понятие виртуальной памяти?
5. На каких основных принципах построено преобразование виртуального адреса в физический?
6. Какая основная системная структура используется в подсистеме виртуальной памяти для каждого процесса?
7. Какова величина страницы, с которой оперирует страничная виртуальная память?
8. Оформите в виде блок-схемы алгоритм преобразования виртуального адреса в физический в подсистеме виртуальной памяти.

6. Подсистема управления внешними устройствами (подсистема ввода-вывода)

Дисковая подсистема ОС. Понятие «геометрии диска»

Существует множество различных типов дисков. Наиболее распространенными являются жесткие магнитные диски. Их характеризует сравнительно высокая скорость чтения-записи данных, что позволяет им играть роль вторичной памяти, например, для реализации виртуальной памяти. Массивы, составленные из таких дисков, иногда используются для организации высоконадежного хранилища данных. Однако для распространения программ, данных и фильмов чаще используются разнообразные оптические диски (DVD и Blu-Ray диски). И, наконец, растущей популярностью, благодаря высокой скорости работы и отсутствию механических частей, пользуются твердотельные диски. Далее более подробно будут рассмотрены жесткие магнитные диски.

Одной из основных характеристик дисковых систем является тип интерфейса. **Интерфейс** — это способ взаимодействия жесткого диска и материнской платы компьютера, представляющий собой набор специальных линий и специального протокола (набора правил передачи данных) [20]. Приведем примеры без детального описания нескольких современных интерфейсов для серийно выпускаемых внутренних жестких дисков.

IDE (*Integrated Drive Electronics*) — буквально означает «встроенный контроллер», один из самых ранних (1980-е гг.), также называемый ATA (*Advanced Technology Attachment*). ATA — это параллельный интерфейс передачи данных, за что вскоре он был переименован в PATA (*Parallel ATA*). IDE хоть и был очень медленный (пропускная способность канала передачи данных составляла от 100 до 133 Мбайт в секунду в разных версиях), однако позволял присоединять одновременно сразу два устройства к материнской плате, используя при этом один шлейф.

Следующим не менее популярным, чем IDE в свое время, интерфейсом является SATA (*Serial ATA*), характерной особенностью которого является последовательная передача данных. Стоит отметить, что он является самым массовым для применения в ПК.

SCSI (*Small Computer System Interface*) — параллельный интерфейс, используется для подключения различных внешних устройств (не только жестких дисков). Он постепенно вытесняется новым стандартом — SAS (*Serial Attached SCSI*), который устраняет ряд недостатков SCSI.

Жесткий диск (рис. 6.1) (иногда употребляют название «винчестер») был впервые реализован для ПК в 1983 г. и с тех пор стал основным устройством внешней памяти. Работа такого диска основана на 2 принципах:

1) магнитной технологии записи;
2) быстром вращении диска (5400 и 7200 об/мин, хотя имеются модели с 10 000 и 15 000 об/мин).

Рис. 6.1. Жесткий диск для ПК

Жесткие диски представляют собой несколько пластин с магнитным покрытием, которые расположены на одной оси (шпинделе) и вращаются с большой скоростью (рис. 6.2). Считывание-запись информации осуществляется с помощью специальных устройств — головок диска, которые располагаются одна под другой между пластинами и перемещаются от центра к краям пластин.

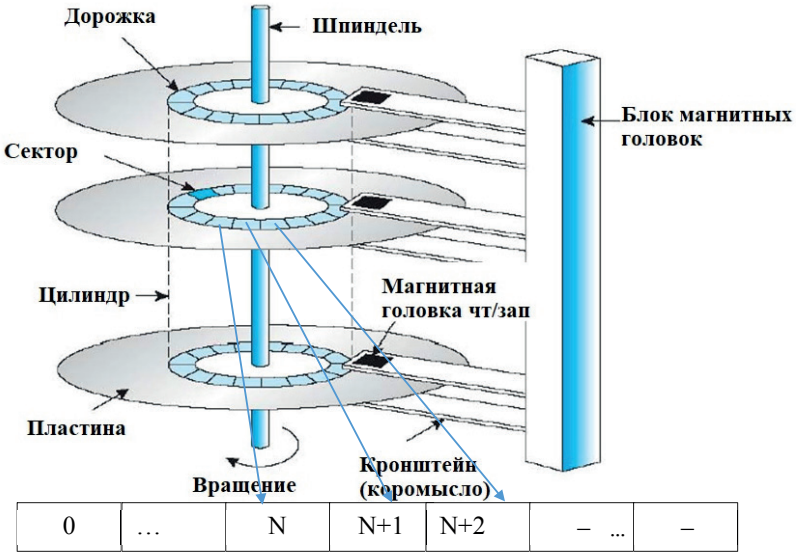


Рис. 6.2. Принципиальная схема работы жесткого диска

Окружность на магнитной пластине, которую описывает головка при вращении пластин, называется **дорожкой**, а совокупность таких дорожек, расположенных одна под другой на всех пластинах (определяемая каждым фиксированным положением головок), называется **цилиндром**. Каждая дорожка разбита на секторы, в секторе содержится 512 байт информации.

Существует понятие геометрии диска. Она определяется совокупностью трех цифр: числом цилиндров — числом дорожек в цилиндре — числом секторов на дорожке, или CHS (от первых букв соответствующих английских слов: *Cylinder–Head–Sector*, т. е. цилиндр–головка–сектор). «Сырая» емкость диска определяется как произведение: CHS·512 (байт).

Диски являются блочными устройствами, т. е. считывание и запись информации производится блоками, и минимальный размер блока равен одному сектору (512 байт). Только низкоуровневые драйверы работают с физическими адресами, образованными как: номер цилиндра, номер считывающей головки (или дорожки) и порядковый номер сектора на дорожке. В основном программы ОС используют другое представление диска — в виде линейной последовательности блоков (секторов), перенумерованных от нуля до некоторого максимального числа (см. рис. 6.2), начиная с нулевого сектора на внешней дорожке.

Понятие раздела. Схема разделов, основанная на MBR

Диски принято разбивать на разделы. Объясняется это тем, что первые версии MS-DOS не могли обеспечить доступ к большим дискам (объемы дисков росли быстрее, чем возможности DOS).

Раздел можно определить как последовательность смежных секторов, выделенных для размещения одной файловой системы. Каждый раздел может рассматриваться как отдельный физический диск.

Согласно классической схеме (BIOS-MBR), сведения о разделах содержатся в специальной таблице — таблице разделов (*Partition Table*), которая находится в Главной загрузочной записи (MBR — *Master Boot Record*) (рис. 6.3). MBR располагается в первом физическом секторе жесткого диска.

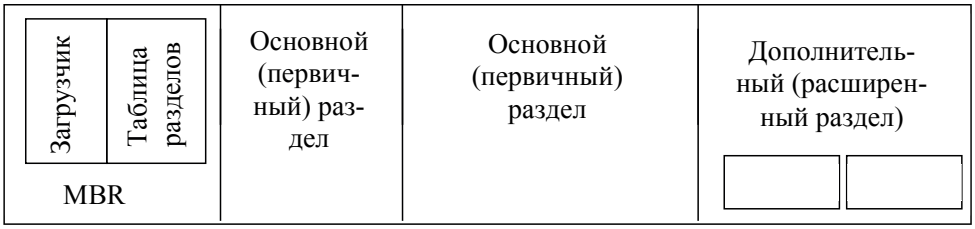


Рис. 6.3. Логическая структура диска

В MBR (рис. 6.4) под таблицу разделов выделено 64 байт. Каждая запись имеет длину в 16 байт. Таким образом, всего на жестком диске может быть создано не более 4 разделов. Когда разрабатывалась структура MBR, это считалось достаточным. Позднее данное ограничение было преодолено за счет того, что ввели новый тип раздела. Дисковый раздел может быть одного из двух типов: основного (первичного) и дополнительного (расширенного). Разница в том, что дополнительный раздел может быть разбит на части — подразделы, однако нужно помнить, что такой раздел может быть только один на диске.

Число логических подразделов в принципе не ограничено, потому что каждый логический раздел может содержать таблицу разделов и вложенные логические разделы. В реальности ограничения есть: ОС Linux работает максимально с 15 логическими разделами на SCSI-дисках и с 63 логическими разделами на IDE-дисках.

Address		Description		Size (bytes)
Hex	Dec			
000_{hex}	+0	Bootstrap code area		446
$1BE_{hex}$	+446	Partition entry №1	*Partition table* (for primary partitions)	16
$1CE_{hex}$	+462	Partition entry №2		16
$1DE_{hex}$	+478	Partition entry №3		16
$1EE_{hex}$	+494	Partition entry №4		16
$1FE_{hex}$	+510	55_{hex}	*Boot signature*[a]	2
$1FF_{hex}$	+511	AA_{hex}		
			Total size: 446 + 4×16 + 2	**512**

Рис. 6.4. Структура главной загрузочной записи (MBR)

Кроме понятия основного и дополнительного разделов, важным является понятие активного раздела. Активным назначается раздел, ОС которого должна руководить процессом загрузки. Эта ОС загружается первой, а дальше она может предложить загрузить другую ОС, если это необходимо. Только один раздел в таблице разделов может быть помечен как активный, и именно в нем должны содержаться файлы, необходимые для загрузки ОС. При такой классической схеме загрузку начинает BIOS, пример экрана конфигурирования которой показана на рис. 6.5.

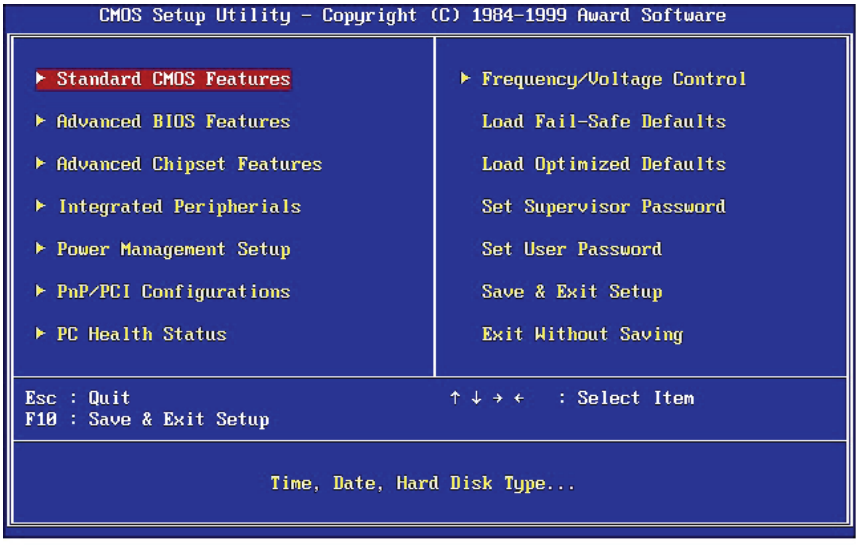


Рис. 6.5. Пример экрана программы конфигурирования BIOS

BIOS и UEFI

BIOS – *Basic Input-Output system* — базовая система ввода-вывода. Это программа низкого уровня (англ. *firmware*), хранящаяся на чипе материнской платы компьютера [21].

Существует программа конфигурирования BIOS (см. рис. 6.5), с помощью которой можно изменять множество параметров: аппаратную конфигурацию компьютера, системное время, последовательность поиска загрузочных устройств. Программу конфигурирования BIOS можно вызвать в начале загрузки компьютера по нажатию определенной клавиши — на разных компьютерах она разная, но чаще всего используются клавиши Esc, F2, F10, Delete. Сохраняя настройку, вы сохраняете ее в памяти материнской платы. При загрузке компьютера, BIOS настроит его так, как указано в сохраненных настройках.

Перед загрузкой ОС BIOS проходит через POST, или *Power-On Self Test* — самотестирование при включении. Проверяется корректность настройки минимума необходимого аппаратного обеспечения и его работоспособности. Если что-то не так, на экране появляется серия сообщений об ошибках или раздается предупредительный сигнал. Окончив POST, BIOS проверяет устройства, с которых может производиться загрузка (согласно хранящейся последовательности загрузки), и на найденном устройстве (если это жесткий диск) ищет MBR. Для этого BIOS считывает один сектор (512 байт), который находится по адресу: «цилиндр 0, головка 0, сектор 1», и помещает его в область памяти по физическому адресу 0x7C00. Далее BIOS проверяет, что этот сектор оканчивается сигнатурой 55 AAh (если это не так, то управление возвращается обратно в BIOS).

BIOS передает управление по физическому адресу 0x7C00 (т. е. сектору MBR). В MBR существует раздел, помеченный как активный, задача содержащегося в этой же записи загрузчика –передать управление на этот раздел. Код такого первичного загрузчика крайне мал (см. рис. 6.4), и никакой более серьезный функционал он выполнить не может. Все остальные действия по загрузке возлагаются на ОС, содержащуюся в активном разделе.

BIOS существует уже давно и эволюционировала мало. Даже у компьютеров с ОС MS-DOS, выпущенных в 1980-х гг., был BIOS. Конечно, со временем BIOS все-таки менялась и улучшалась. Разрабатывались ее расширения, в частности ACPI — *Advanced Configuration and*

Power Interface (усовершенствованный интерфейс управления конфигурацией и питанием). Это позволяло BIOS проще настраивать устройства и более эффективно управлять питанием, например уходить в спящий режим, но BIOS развилась не так активно, как другие компьютерные технологии со времен MS-DOS.

У традиционной BIOS есть серьезные ограничения. Загрузка возможна только с жестких дисков объемом не более 2,1 Тбайт. Сейчас уже повсеместно встречаются диски на 3 Тбайта, и с них компьютер с BIOS не загрузится. Это ограничение BIOS MBR. С BIOS возможна работа только в 16-битном режиме процессора, при этом доступен всего 1 Мбайт памяти. Существуют проблемы с одновременной инициализацией нескольких устройств, что ведет к замедлению процесса загрузки, во время которого инициализируются все аппаратные интерфейсы и устройства.

BIOS необходимо было заменить. Intel начала работу над *Extensible Firmware Interface* (EFI) еще в 1998 г. Apple выбрала EFI, перейдя на архитектуру Intel на своих компьютерах Mac в 2006 г., но другие производители не пошли за ней. В 2007 г. Intel, AMD, Microsoft и производители PC договорились о новой спецификации *Unified Extensible Firmware Interface* (UEFI) — унифицированного интерфейса расширяемой прошивки. Это индустриальный стандарт, обслуживаемый форумом UEFI, и он зависит не только от Intel. Поддержка UEFI в ОС Windows появилась с выходом Windows Vista Service Pack 1 и Windows 7. Большая часть компьютеров, которые сегодня продаются, используют UEFI вместо BIOS.

UEFI заменяет традиционную BIOS на ПК. На существующем компьютере никак нельзя поменять BIOS на UEFI. Нужно покупать аппаратное обеспечение, поддерживающее UEFI. Большинство версий UEFI поддерживают эмуляцию BIOS, так что обратная совместимость у них есть.

Новый стандарт обходит ограничения BIOS. Прошивка UEFI может осуществлять загрузку с дисков объемом более 2,2 Тбайта; теоретический предел для них составляет 9,4 Эбайт (это примерно в три раза больше всех данных, содержащихся в сегодняшнем интернете).

UEFI использует GPT (*GUID Partition Table* — графическая таблица разделов) и специально отформатированный раздел на диске вместо MBR. Раздел имеет файловую систему FAT32. Структура GPT показана на рис. 6.6.

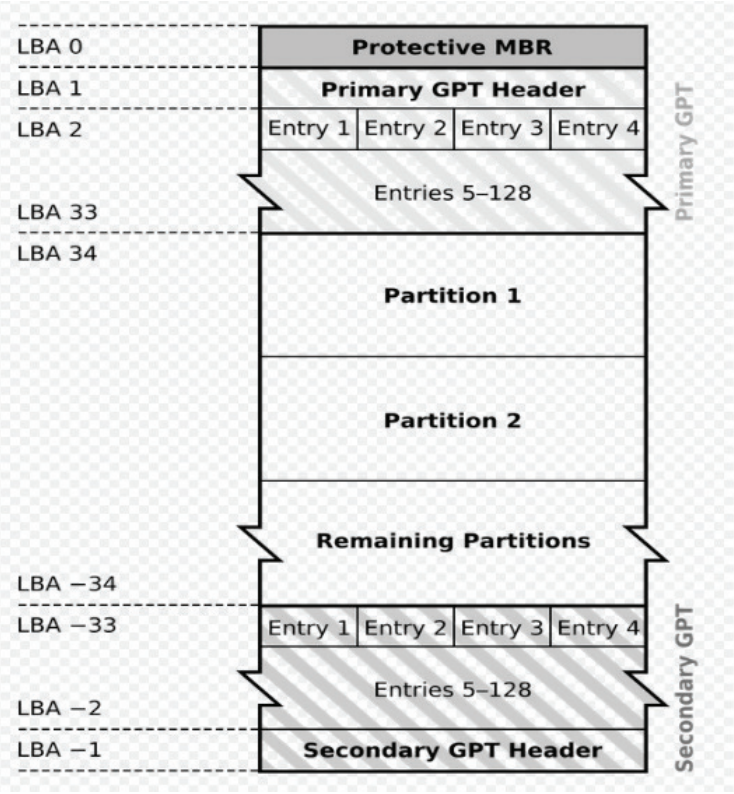


Рис. 6.6. Структура GPT

GPT начинается с оглавления таблицы разделов (англ. *Partition Table Header*). Она использует современную систему адресации логических блоков (LBA) вместо применявшейся в MBR адресации «цилиндр — головка — сектор» (CHS). MBR, доставшаяся по наследству со всей своей информацией, содержится в блоке LBA 0, оглавление GPT — в блоке LBA 1. В оглавлении содержится адрес блока, где начинается сама таблица разделов, обычно это следующий блок — LBA 2. Количество разделов не ограничено стандартом и зависит от ОС. Так, в Windows в таблице разделов резервируется место для 128 записей по 128 байт каждая (в Linux ядро поддерживает до 256 разделов).

У UEFI стандартизирован процесс загрузки, и она запускает исполняемые программы EFI вместо кода, расположенного в MBR. UEFI может работать в 32-битном или 64-битном режимах, и ее адресное пространство больше, чем у BIOS, а значит, быстрее загрузка. Также это означает, что экраны настройки UEFI можно

сделать красочнее и понятней, чем у BIOS, включить туда графику и поддержку мыши.

В UEFI встроено множество других функций. Она поддерживает безопасный запуск Secure Boot, в котором можно проверить, что загрузку ОС не изменила никакая вредоносная программа. Она может поддерживать работу по сети, что позволяет проводить удаленную настройку и отладку. Для настройки компьютера с традиционной BIOS необходимо было сидеть прямо перед ним. И это не просто замена BIOS. UEFI — это небольшая ОС, работающая над прошивкой PC, поэтому она способна на гораздо большее, чем BIOS. Ее можно хранить во флеш-памяти на материнской плате или загружать с жесткого диска или с сети.

У разных компьютеров бывает разный интерфейс и свойства UEFI. Все зависит от производителя компьютера, но основные возможности одинаковы у всех. В современных версиях ОС существуют программы — менеджеры UEFI (*UEFI boot manager*), т. е. конфигурацией загрузки можно управлять изнутри ОС (рис. 6.7).

Суммируя все вышесказанное, можно выделить следующие преимущества UEFI в сравнении с BIOS [22]:

1) простой и понятный графический интерфейс программы-менеджера на многих языках, включая русский с поддержкой управления мышью;
2) поддержка накопителей вместительнее 2,2 Тбайта с неограниченным количеством разделов;
3) практически неограниченное количество создаваемых на диске разделов;
4) намного более быстрая загрузка ОС. Так, Windows 10, установленная на SSD-диск, размеченный по новому стандарту GPT, грузится всего за 4–15 с;
5) собственный менеджер загрузки ОС. Позволяет компьютеру грузиться с носителей, которые не имеют своих загрузчиков, и изменять конфигурацию загрузки непосредственно из среды ОС;
6) поддержка установки приложений и драйверов сторонних производителей, которые расширяют функциональность UEFI;
7) защита от внедрения вредоносного кода в системные загрузчики и собственную среду (обеспечивает встроенный в интерфейс протокол Secure Boot);
8) настройка UEFI из среды ОС, например, из Windows 8 и 10, различных версий Linux.

Рис. 6.7. Пример экрана настройки UEFI

Для определения, с чем (BIOS или UEFI) работает компьютер с ОС Windows, можно запустить программу Diskpart. Ее выдача укажет на стандарт UEFI, если в соответствующем столбце «GPT» есть символ «*» (рис. 6.8, *а*), и на BIOS, если ее нет (рис. 6.8, *б*).

Особенности работы с дисками и разделами в разных операционных системах

В Linux диск в целом (т. е. физический диск) доступен через файл, зарегистрированный в каталоге /dev. Примеры имен файлов, ассоциированных с дисками: /dev/hda, /dev/hdb, /dev/sda. По имени такого файла можно многое сказать об этом диске.

1) IDE-диски — имя файла начинается с hd:
   - главный диск (master) первичного контроллера -> a;
   - подчиненный (slave) диск первичного контроллера -> b;
   - главный диск (master) вторичного контроллера -> c;
   - подчиненный (slave) диск первичного контроллера -> d;

2) SCSI- (SATA- и др.) диски — имя начинается с sd:
   - диски нумеруются последовательно (a, b, c, d, ...).

а

```
Выбрать Администратор: Командная строка - diskpart
Microsoft Windows [Version 10.0.17134.1069]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.

C:\WINDOWS\system32>diskpart

Microsoft DiskPart, версия 10.0.17134.1

(C) Корпорация Майкрософт (Microsoft Corporation).
На компьютере: DESKTOP-LJA0BRN

DISKPART> list disk

  Диск ###  Состояние      Размер   Свободно Дин  GPT
  --------  -------------  -------  -------  ---  ---
  Диск 0    В сети          223 Gбайт  1024 Кбайт

DISKPART>
```

б

```
Администратор: Командная строка - diskpart
Microsoft Windows [Version 10.0.18362.356]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\WINDOWS\system32>diskpart

Microsoft DiskPart, версия 10.0.18362.1

(C) Корпорация Майкрософт (Microsoft Corporation).
На компьютере: LAPTOP-SV416JA8

DISKPART> list disk

  Диск ###  Состояние      Размер   Свободно Дин  GPT
  --------  -------------  -------  -------  ---  ---
  Диск 0    В сети          931 Gбайт      0 байт        *
  Диск 1    В сети          238 Gбайт  6144 Кбайт        *
  Диск 2    В сети         7680 Мбайт  1024 Кбайт
```

Рис. 6.8. Использование утилиты diskpart для определения, с BIOS (*а*) или UEFI (*б*) работает компьютер

Разделы на дисковых устройствах также ассоциированы с отдельными файлами, и нумеруются цифрами, начиная с 1.

Для случая BIOS и MBR действуют следующие правила. Файлы, ассоциированные с первичными разделами, содержат имя файла диска и цифру от 1 до 4: /dev/hda1, /dev/hda2, /dev/hda3, /dev/hda4, — а логические разделы в Linux доступны по именам: /dev/hda5, /dev/hda6 ... (начиная с 5). Из сказанного выше ясно, что /dev/hda3 и /dev/hda4 могут быть пропущены, т. к. третий и четвертый первичные разделы не были созданы, а был создан расширенный раздел, в котором сформированы логические подразделы.

В Windows существует понятие «логический диск», такой диск формируется в каждом разделе. Он представляет собой отдельную файловую систему, начинающуюся с корневого каталога. В дополнительном разделе можно сформировать несколько логических дисков (в каждом подразделе существует отдельный логический диск).

В Windows логические диски имеют однобуквенные имена — буквы латинского алфавита. Например, имеется один жесткий диск с двумя основными разделами (в них сформируются диски C: и D:) и одним расширенным разделом, в котором созданы два логических диска; они получат имена E: и F:.

Контрольные вопросы

1. Какие типы дисковых устройств вам известны?
2. Что такое цилиндр и дорожка? Какой стандартный размер сектора?
3. В чем преимущества стандарта UEFI по сравнению с BIOS?

7. Файловые системы

Одна из основных функций ОС — это интерфейсная функция, которая предполагает и удобство работы с данными, хранящимися на внешних носителях. Для этого ОС подменяет физическую структуру хранящихся данных удобной для пользователя логической моделью. Логическая модель файловой системы материализуется в виде иерархии каталогов, которая может быть выведена на экран такими программами, как Проводник в Windows или Nautilus в Linux. Конечный пользователь имеет дело с файлами, которые упорядочены с помощью каталогов (в Windows используют понятие папки, хотя папка более широкое понятие, чем каталог).

Базовым элементом логической модели является файл. **Файл** — это именованная область данных на носителе информации [4, 5].

Основные цели использования файла:

1) 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;
2) совместное использование информации. Файлы обеспечивают естественный и легкий способ использования информации несколькими приложениями и пользователями за счет понятного символьного имени, долговременного хранения информации и известного расположения файла.

Файловая система (ФС) — это часть ОС, включающая:

1) множество файлов различного типа, хранимых на диске;
2) системные структуры, используемые для управления файлами, такие, например, как различные системные таблицы с информацией о файлах, индексные дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;

3) комплекс программ, реализующих различные операции с файлами, такие как создание, уничтожение, чтение, запись и т. д.

ФС предоставляет программам (и программистам) набор стандартных функций, при вызове которых выполняются действия над некоторым объектом, представляющим файл. При этом программистам не нужно оперировать с физическими адресами в формате CHS, буферизацией данных, решать другие низкоуровневые проблемами передачи данных. Все это делает ФС; именно она распределяет дисковую память, отображает имена файлов в соответствующие физические адреса, обеспечивает доступ к данным. ФС играет роль промежуточного слоя, интерфейса, который экранирует все сложности физической организации, предоставляя программам более простую логическую модель внешнего хранилища и набор удобных в использовании команд для манипулирования данными.

Способ организации вычислительного процесса определяет задачи, которые должна решать ФС. Самый простой тип ФС — это ФС в однопользовательских и однопрограммных ОС, такая как была разработана для MS-DOS. Основные функции в этой ФС призваны решать следующие задачи:

1) именования файлов;
2) предоставления программного интерфейса для приложений;
3) отображения логической модели ФС на физическую организацию хранилища данных;
4) устойчивости ФС к сбоям питания, ошибкам аппаратных и программных средств.

Задачи ФС усложняются, если они работают в многопользовательских и мультипрограммных ОС. К перечисленным выше задачам добавляются задачи обеспечения совместного доступа к файлу нескольких процессов и защиты файлов одного пользователя от несанкционированного доступа других пользователей.

В мультипрограммных системах файл является разделяемым ресурсом, и ФС должна решать весь комплекс проблем, связанных с такими ресурсами, например, в ФС должны быть предусмотрены средства блокировки файла и его частей в случае попыток одновременного доступа, предотвращения гонок, синхронизация копий и т. п.

Еще более усложняются функции ФС, разработанной для сетевой ОС.

Типы файлов

В ФС существуют файлы функционально различающихся типов, в число которых обычно входят: обычные файлы, файлы-каталоги, файлы устройств, символические ссылки, сокеты и др.

Обычные файлы содержат информацию, которая была введенна пользователем или которая является результатом работы системных утилит и пользовательских приложений. Большинство современных ОС не накладывает никаких ограничений на содержимое и структуру такого файла. Содержание и структура обычного файла определяется программой, которая с ним работает. Существует одно исключение — все ОС должны уметь распознавать собственные исполняемые файлы.

Каталоги — это особый тип файлов, они содержат системную справочную информацию о множестве файлов, сгруппированных пользователями по некоторому признаку. Во многих ОС в каталоге могут быть зарегистрированы файлы разных типов, в т. ч. другие каталоги, за счет чего образуется иерархическая структура, удобная для поиска. Каталоги поддерживают соответствие между именами файлов и их характеристиками, которые ФС использует для управления файлами. В число характеристик могут входить: информация о типе файла, его расположении на диске, правах доступа к файлу и датах его создания и модификации, причем эта информация может содержаться непосредственно в файле-каталоге или в нем может содержаться указатель на некую системную структуру, где информация находится.

Файлы устройств — это фиктивные файлы, которые ассоциированы с устройствами ввода-вывода, созданными для организации универсального механизма доступа к файлам и внешним устройствам. Файлы устройств позволяют организовать ввод-вывод на устройство в виде последовательности команд записи в файл или чтения из файла. Сначала команды ввода-вывода обрабатываются программами ФС, а затем преобразуются ОС в команды управления соответствующим устройством.

В Windows-системах наиболее часто используются такие файлы устройств, как PRN (файл, связанный с устройством печати) и NUL (фиктивное устройство, по сути «черная дыра», который в основном используется в командной строке для того, чтобы сообщение, выдаваемое командой, не выводилось на экран).

В Linux вся работа с устройствами организована через соответствующие файлы. О файлах, связанных с дисковыми устройствами, говорилось в п. «Особенности работы с дисками и разделами в разных ОС» (см. с. 92).

Современные ФС поддерживают и другие типы файлов, такие как символические ссылки (о которых будет рассказано подробнее дальше), именованные конвейеры и др.

Иерархическая структура файловой системы

Пользователи обращаются к файлам используя символьные имена. Однако человек способен запомнить ограниченное число имен объектов. Иерархическая организация пространства имен позволяет расширить эти границы, поэтому большинство ФС имеет иерархическую структуру, в которой уровни создаются за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня, как это показано на рис. 7.1.

Граф, описывающий иерархию каталогов, может иметь структуру дерева (рис. 7.1, *б*) или сеть (рис. 7.1, *в*). Каталоги образуют дерево, если файлу (каталогу) разрешено быть зарегистрированным только в одном каталоге (рис. 7.1, *б*), и сеть, если файл (каталог) может быть зарегистрирован в нескольких каталогах (рис. 7.1, *в*). Например, в файловой системе FAT каталоги образуют древовидную структуру, а в NTFS, ext2fs — сетевую. Каталог самого верхнего уровня называется корневым каталогом. Он обычно обозначается символом разделителя, принятым в ОС («\» — обратный слеш в ОС Windows, «/» — обычный слеш в ОС Linux).

Иерархическая структура удобна для многопользовательской работы: каждый пользователь регистрирует свои файлы в определенном каталоге (поддереве каталогов), и в то же время все файлы в системе логически связаны.

Частным случаем иерархической структуры является одноуровневая организация, когда все файлы входят в один каталог (рис. 7.1, *а*)

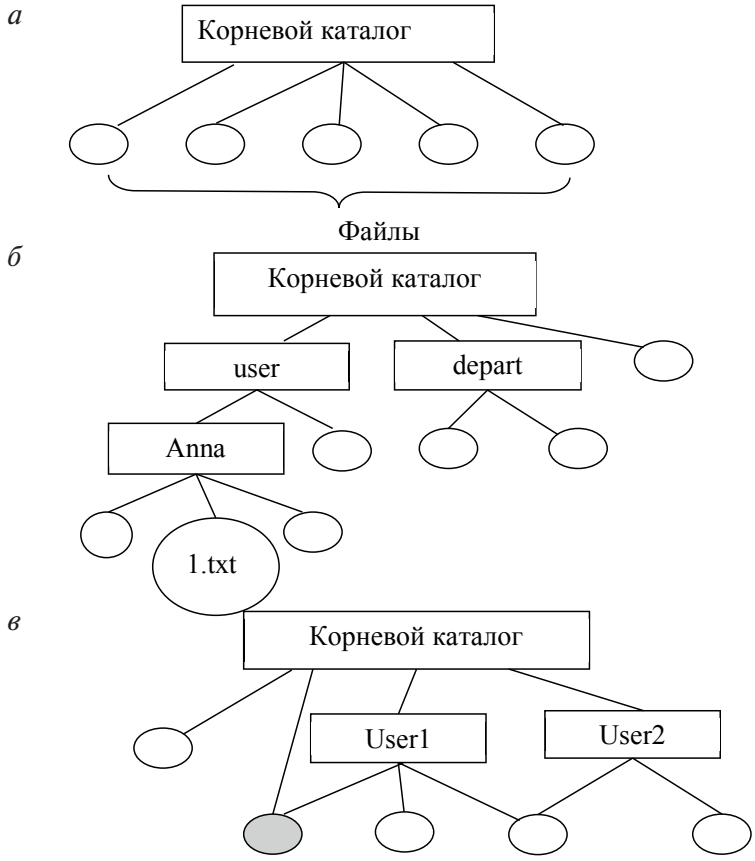


Рис. 7.1. Примеры иерархии файловых систем

Имена файлов

Неотъемлемой характеристикой файла является его символьное имя. В ФС обычно используются три типа имен файлов: простые, составные и относительные.

Простое (короткое) имя идентифицирует файл в пределах одного каталога. Простые имена присваивают файлам пользователи и программисты, при этом они должны учитывать ограничения ОС как на номенклатуру символов, так и на длину имени. До недавнего времени эти границы были весьма узкими. Например, в ФС FAT16 имя файла определялось схемой 8.3 (8 символов — собственно имя, 3 символа — расширение имени, причем расширение указывало на тип хранящейся в нем информации). Однако пользователю гораздо удобнее рабо-

тать с длинными именами, позволяющими дать любому файлу легко запоминающееся название, которое точно определяет, что за информация в нем содержится. Большинство современных ФС поддерживают длинные простые символьные имена файлов. Как правило, длинное имя файла может содержать до 255 символов и, при употреблении его в командной строке, необходимо заключать его в кавычки.

Примеры простых имен файлов и каталогов:

quest_u1.xlsx

“письмо на деревню дедушке.doc”

В иерархических ФС разным файлам разрешено иметь одинаковые простые символьные имена при условии, что они принадлежат разным каталогам, т. е. истинно высказывание «много файлов — одно простое имя». Для однозначной идентификации файла в таких системах используются имена другого типа.

Полное имя представляет собой упорядоченную последовательность простых символьных имен всех каталогов, содержащихся в пути от корня до данного файла и отделенных друг от друга разделителем, принятым в данной ОС. Таким образом, полное имя можно считать составным. На рис. 7.1, *б* показан файл, который имеет простое имя 1.txt, его составное имя/user/anna/1.txt — в ОС Linux (или\user\anna\1.txt — в Windows; в этой системе может быть также указан логический диск — C:\user\anna\1.txt).

При структуре ФС «дерево», истинным является высказывание «один файл — одно полное имя». В ФС, имеющих сетевую структуру, файл может входить в несколько каталогов, а значит, иметь несколько полных имен, здесь справедливо другое высказывание: «один файл — много полных имен». Эти равнозначные полные имена есть то, что называют «жесткими ссылками». Несколько жестких ссылок, например, имеет файл, выделенный цветом на рис. 7.1, *в*.

Файл может быть идентифицирован также относительным именем. Относительное имя файла основано на понятии «текущий каталог». Для каждого пользователя в определенный момент времени один из каталогов ФС назначен текущим, этот каталог выбирается непосредственно самим пользователем. ФС запоминает имя текущего каталога, а затем использует его как префикс к относительному имени, таким образом формируя полное имя файла. При использовании относительного имени, пользователь дополняет имя файла последовательностью имен каталогов, через которые необходимо пройти от те-

кущего каталога до данного файла. Например, если текущим каталогом является каталог /user, то относительное имя файла /user/anna/1.txt выглядит следующим образом: anna/1.txt.

Как уже упоминалось выше, в ФС с сетевой структурой между файлами и их полными именами может не существовать взаимно однозначного соответствия, это не всегда удобно с точки зрения ОС. С этой целью часто используют уникальное имя, для которого справедливо высказывание «один файл — одно уникальное имя». Уникальное имя обычно представлено числовым идентификатором и предназначено для использования самой ОС. Примером такого уникального имени файла является номер индексного дескриптора (*inode*) в ФС, работающих с Linux, или номер записи в главной файловой таблице в ФС NTFS из ОС Windows.

Жесткие и символические ссылки

Жесткая ссылка (англ. *hard link*) является еще одним именем для набора данных, размещенного во внешней памяти (того, что и было определено как файл). Такие ссылки появляются в ФС, имеющих сетевую структуру. Одни и те же данные можно зарегистрировать под разными или одинаковыми именами в разных каталогах. Жесткие ссылки равноправны: невозможно различить, где исходное имя файла, а где ссылка. Каждая такая ссылка связана с уникальным именем файла — идентификатором структуры, которая однозначно описывает данные и задает их адреса во внешней памяти. Как описано в предыдущем разделе, для ФС ОС Linux — это структура, называемая индексным дескриптором. Из понимания того, что такое жесткая ссылка, и вытекают все возможности ее использования:

1) при удалении жесткой ссылки, дынные будут удалены из внешней памяти в том случае, если эта ссылка последняя (поэтому, например, при распечатке содержимого каталогов в Linux всегда можно увидеть количество жестких ссылок на каждый из файлов, зарегистрированных в каталоге);
2) нельзя создать жесткую ссылку на файл из другой ФС.

Практически неразличимы (если не изучать временные отметки) исходное имя файла и позже созданные жесткие ссылки, поэтому жест-

кие ссылки применяются там, где отслеживать различия не требуется. Одна из основных причин использования жестких ссылок состоит в том, чтобы предотвратить возможность случайного удаления файла, т. к. данные не удаляются с внешнего носителя, пока на них есть хотя бы одна жесткая ссылка.

Имеется другой тип ссылок — так называемые символические ссылки (англ. *symbolic links*). Эти ссылки также можно рассматривать как дополнительные имена файлов, но в то же время в ФС создаются файлы специального формата, имеющего тип символической ссылки. В этих файлах содержится перенаправление в другое место, где есть жесткая ссылка на данные из внешней памяти. Особенности использования символических ссылок:

1) символическую ссылку можно создать на файл из другой ФС;
2) удаление символической ссылки никак не сказывается на данных во внешней памяти;
3) при удалении всех жестких ссылок на данные во внешней памяти, символическая ссылка становится недействительной.

В ОС Windows несколько изменена идеология понимания и работы с жесткими и символическими ссылками. В этой системе идеология ссылок поддерживается только в ФС NTFS (в FAT есть только символические ссылки на файлы). В ФС NTFS можно создать жесткую ссылку только на файл, а символическую ссылку — на файл и каталог. Символические ссылки на файлы известны под названием «ярлыки». Кроме того, можно создать точку соединения (Junction Point) для каталога. Она также является символической ссылкой на каталог, только созданной по-другому [23].

В NTFS ссылки создаются из командной строки командой mklink [24]. Следует также иметь в виду, что удаление символической ссылки на каталог приводит к очистке содержимого исходного каталога (то же действие с точкой соединения такого эффекта не дает).

На рис. 7.2 показан экран командного интерпретатора ОС Windows, с командами:

1) выдачи системной подсказки на команду MKLINK (MKLINK /?);
2) создания символической ссылки на каталог;
3) создания точки соединения для каталога.

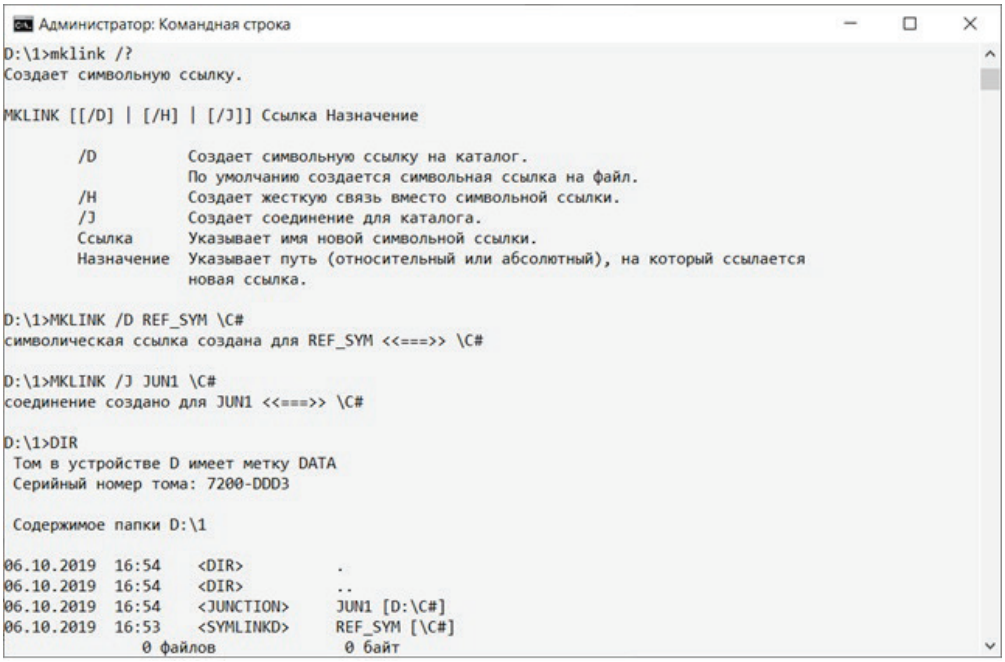


Рис. 7.2. Пример создания ссылок на каталоги с помощью команды MKLINK

Монтирование файловых систем

В компьютере может быть установлено несколько дисковых устройств. Каждое такое устройство может быть разбито на несколько разделов. Как работать с несколькими разделами одновременно?

Одно из решений этой задачи состоит в том, что в каждом разделе создается отдельная ФС, т. е. иерархическая структура, представляющая собой дерево каталогов, которое начинается с корневого каталога. Такая структура автономна — никак не связана с подобными структурами из других дисковых разделов или с других устройств. В этом случае, для однозначной идентификации файла, кроме пути по дереву каталогов необходимо указать устройство (или раздел), в котором находится данная ФС. Как правило, для этого используют понятие логического устройства, для которого задают имя. Примером такого решения является ОС Window, в которой вводится понятие логического диска, для которого задается имя в виде латинской бук-

вы со знаком двоеточия после нее (например, C:). В этом случае полное имя файла начинается с имени этого логического диска, например: C:\user\doc\letter1.docx.

Другое решение стоящей задачи состоит в том, что имеющиеся ФС объединяются в единую структуру, в которой одна иерархическая структура каталогов становится основанием, т. е. устанавливается начиная с корневого каталога, а другие становятся подкаталогами этой базовой структуры. Появляется единая иерархия каталогов. Такая операция называется монтированием. Это решение характерно для UNIX- и Linux-систем.

Для понимания того, что такое монтирование, рассмотрим вариант с двумя ФС (рис. 7.3), он может быть расширен на любое количество ФС.

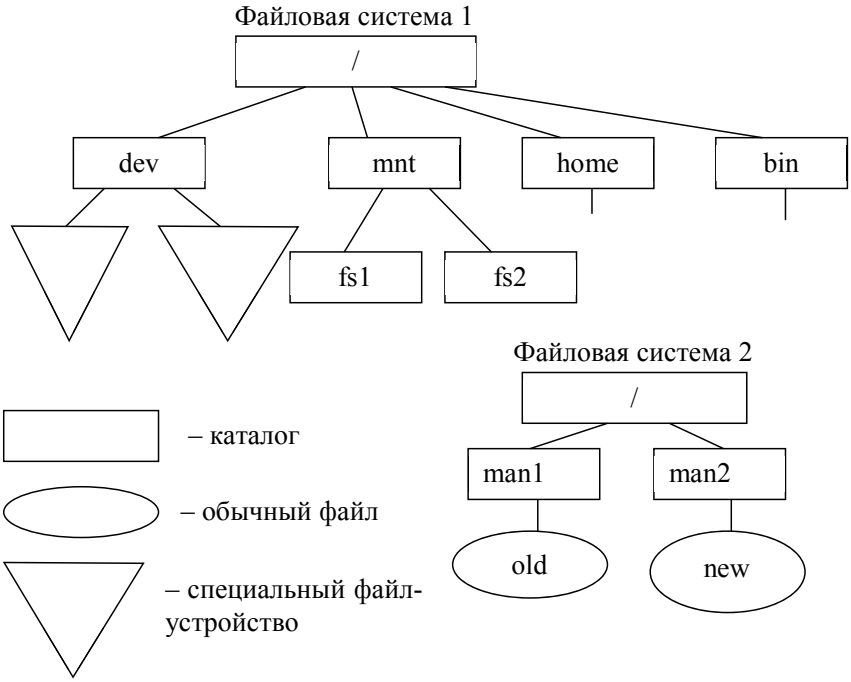


Рис. 7.3. Две файловые системы

Файловая система 1 назначается корневой. В составе этой ФС выбран пустой каталог, который становится точкой, на которую будет монтироваться вторая ФС. В данном примере выбран каталог/mnt/fs1. После выполнения монтирования, каталог/mnt/fs1 стал корневым ка-

талогом второй ФС. Через этот каталог монтируемая ФС подсоединилась как поддерево к общему дереву (рис. 7.4).

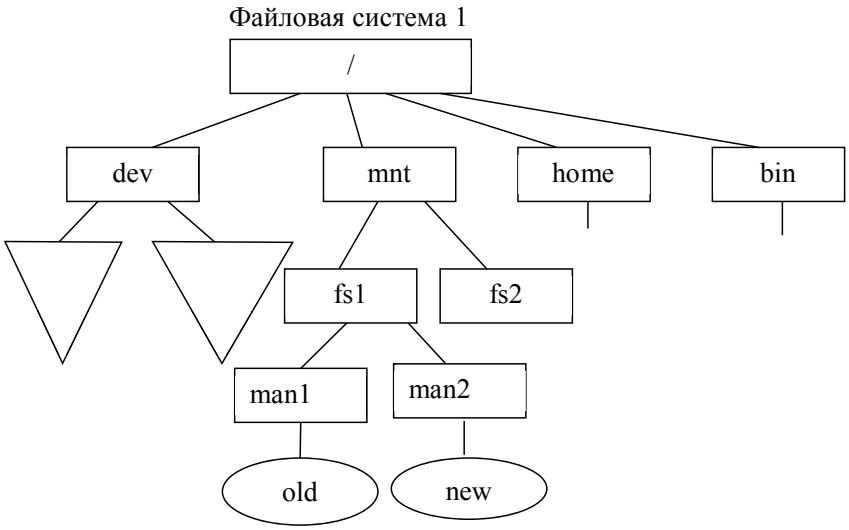


Рис. 7.4. Смонтированные файловые системы

После монтирования общей ФС, для пользователя не существует различия между корневой и смонтированной ФС, а именование файлов производится так же, как в ФС, которые сразу были единой структурой.

Атрибуты файлов

Понятие «файл» включает не только хранимые им данные и имя, но и атрибуты. **Атрибуты** — это информация, описывающая свойства файла. В различных ФС атрибуты могут быть различными, наиболее часто встречаются следующие: тип файла, владелец файла, права доступа к файлу, время создания, время последнего доступа, время последнего изменения, текущий размер файла, признак блокировки и др.

Набор атрибутов файла определяется спецификой ФС: в ФС разного типа могут использоваться разные наборы атрибутов. Например, в однопользовательской ОС в наборе атрибутов не будет таких атрибутов, как владелец файла, информация о правах доступа к файлу. Пользователь может получать доступ к атрибутам, используя средства, предоставленные для этих целей ФС. В большинстве случаев разрешено просматривать значения любых атрибутов, а изменять — только неко-

торых. Например, пользователь может изменить права доступа к файлу (при условии, что он обладает необходимыми для этого полномочиями), но изменить дату создания или размер файла ему не удастся.

В ФС, работающих в ОС Windows, существует 4 стандартных атрибута: только чтение, скрытый, системный и архивный. Файлу присваивается атрибут «только чтение», если его нельзя редактировать или удалять. Файл, имеющий атрибут «скрытый», не обрабатывается системными программами без специального указания (специального ключа в команде встроенного интерпретатора команд). Атрибут «системный» есть сумма атрибутов «скрытый» и «только для чтения». Атрибут «архивный» воспринимается системными программами архивирования. Такой атрибут получают все вновь созданные и измененные файлы: при запуске системная утилита архивирования включает файлы с таким атрибутом во вновь создаваемый архив.

Значения атрибутов файлов могут содержаться в каталогах, как это сделано в ФС FAT. На рис. 7.5, *а* показан упрощенный вариант записи в каталоге для файла в системе FAT. Здесь латинскими буквами обозначены стандартные атрибуты файла: R — только для чтения, A — архивный, H — скрытый, S — системный.

а

Имя файла		Расширение					Резервные
Резервные	Время	Дата	№ первого кластера				Размер

б

№ индексного дескриптора	Имя файла

Рис. 7.5. Структура записи в каталогах:
а — ФС FAT; *б* — ФС ext2fs

Другим вариантом расположения является размещение атрибутов в специальных структурах, когда в каталогах содержатся только ссылки на эти структуры. Такой подход реализован, например, во второй и третьей расширенных файловых система (ext2fs, ext3fs) в ОС Linux.

В этих ФС структура каталога достаточно простая. Запись о каждом файле содержит короткое символьное имя файла и указатель на структуру — индексный дескриптор файла, в которой сосредоточены значения всех атрибутов файла (рис. 7.5, *б*).

В обоих случаях каталоги обеспечивают связь между именами файлов и расположением данных на диске. При этом подход, когда имя файла отделено от его атрибутов, делает систему более гибкой (но менее быстрой). В этом случае файл может быть достаточно просто включен сразу в несколько каталогов.

Физическая организация и адресация файла

Важным аспектом ФС является **физическая организация файла** — способ размещения файла на диске. Основными критериями оценки эффективности организации ФС являются:

1) скорость доступа к данным;
2) объем системной информации, необходимой для поддержания функционирования ФС;
3) склонность к фрагментации дискового пространства;
4) максимально возможный размер файла.

Для дальнейшего обсуждения физической организации и адресации файла следует ввести понятие кластера. Ранее обсуждалось, что диски — это блочные устройства: обмен информацией между диском и ОП ведется блоками по 512 байт (размер сектора). Однако, если ФС будет следить за каждым сектором на диске, системные данные будут очень большого объема, т. е. пространство на диске будет расходоваться неэффективно, поэтому ФС работает не с сектором, а с кластером (такое понятие существует в большинстве современных ФС, иногда употребляют термин «блок»).

Кластер — это минимальная единица, с которой работает ФС, состоит из последовательно расположенных секторов, число которых равно степен 2. Таким образом, размер кластера может быть равен 1 сектору (512 байт), 2 секторам (1024 байт), 4 секторам (4096 байт) и т. д. Максимальный размер блока в ФС, работающих под Windows, равен 64 Кбайта.

Нужно учитывать, что любой файл не может занимать место на диске меньше 1 кластера, даже если в нем содержится 1 символ, т. е. чем

больше кластер, тем больше места на диске используется непроизводительно из-за пустых «хвостов» файлов (реальная длина файла не всегда кратна размеру кластера). Это так называемая внутренняя фрагментация. В некоторых современных ФС, например в Reiserfs, предложены решения по упаковке «хвостов» файлов, что приводит к 5 % экономии дискового пространства. Обычно ищется некоторый компромисс между увеличением системных (адресных) данных в ФС и явлением внутренней фрагментации.

Как правило, используются следующие варианты размещения файлов в ФС.

Непрерывное размещение — это наиболее простой вариант физической организации файла, при котором ему предоставляется последовательность секторов диска, составляющих непрерывный участок дисковой памяти. Основным достоинством этого метода является высокая скорость доступа, т. к. затраты на поиск и считывание файла минимальны, также минимален объем адресной информации — достаточно хранить только номер первого сектора (кластера) и объем файла. При данной физической организации файла его максимально возможный размер не ограничен.

Данный вариант имеет и существенные недостатки, которые делают его применимость на практике невозможной, несмотря на всю логическую простоту. При более пристальном рассмотрении оказывается, что реализовать эту схему не так уж просто. Возникает вопрос: какого размера должна быть непрерывная область, выделяемая файлу, если файл при редактировании каждый раз может увеличить свой размер? Возникает еще одна серьезная проблема — фрагментация. В процессе работы ФС, в результате выполнения операций создания и удаления файлов, пространство диска становится сильно фрагментированным, оно включает множество свободных областей небольшого размера. Как обычно бывает в случае с фрагментацией, суммарный объем свободной памяти может быть очень большим, а выбрать место для размещения нового файла невозможно. На практике в основном применяются методы, в которых под файл выделяется несколько, в общем случае несмежных, областей диска.

Еще один способ физической организации — использование связного списка отрезков дисковой памяти. Связный список — достаточно хорошо изученная структура данных. При этом задается упорядоченная последовательность отрезков, в начале каждого отрезка содержит-

ся указатель на следующий отрезок. При такой организации адресная информация минимальна: расположение файла может быть задано по номеру первого отрезка, остальные адреса определяются прямым проходом по списку. В сравнении с предыдущим способом, фрагментация значительно меньше. Файл может изменять свой размер во время своего существования, увеличивая число отрезков. Недостатком этой организации является сложность реализации доступа к произвольному участку файла (особенность списковой структуры): для того чтобы прочитать десятый по порядку отрезок файла, необходимо сначала прочитать девять первых отрезков. В некоторых ФС отрезок равен кластеру.

Разновидностью этого способа физической организации является ФС FAT, о которой более подробно будет рассказано далее. В ней данная схема несколько модифицирована: связные цепочки отрезков можно определить по системной структуре — таблице FAT (FAT — *File Allocation Table*). При такой физической организации сохраняются все достоинства организации: минимальность адресной информации, незначительная фрагментация, возможность простого изменения размера файла.

Еще один способ задания физического расположения файла заключается в явном перечислении адресов участков памяти, в которых размещен файл. Недостаток этого способа в том, что длина адреса зависит от размера файла, для большого файла она может составить значительную величину. Достоинством этого способа является высокая скорость доступа к произвольному месту файла, т. к. здесь применяется прямая адресация, она не требует просмотра цепочки предшествующих отрезков файла. Фрагментация в данном случае также незначительная. Разновидности этой физической организации можно обнаружить в ФС NTFS (ОС Windows) и в системе ext2fs (ОС Linux).

Современные файловые системы

В ОС Windows современных версий основной ФС является система NTFS. На флеш-носителях в основном используется ФС FAT — одна из самых давно используемых и устойчивых ФС (она также используется в системном разделе при работе с UEFI). Ее путь начался с ори-

гинальной 8-битной системы FAT в 1977 г., которая функционировала внутри автономного диска Microsoft Standalone Disk Basic-80. Он был запущен специально для Intel 8080 NCR 7200 в 1977–78 гг., работая терминалом ввода данных с 8-дюймовыми гибкими дисками. После обсуждений о введении системы с учредителем Microsoft Биллом Гейтсом, код был написан первым наемным сотрудником компании Марком Макдональдом [25].

На рис. 7.6 представлены экраны программ «Свойства диска» системы Windows 10 с указанием того, какие ФС сформированы в этих дисках.

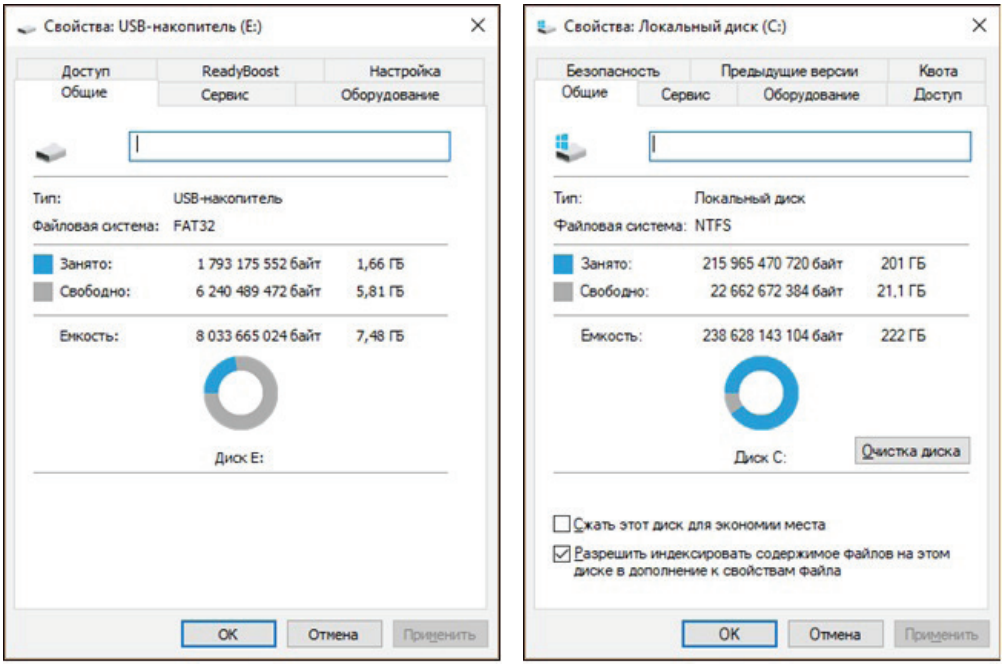


Рис. 7.6. Экраны программ «Свойства диска» с указанием типа файловой системы

Анонсирована также новая ФС — ReFS (*Resilient File System*). ReFS — это новая система, созданная на базе NTFS. Она была первоначально представлена в ОС Windows Server 2012. Эта ФС также включена в Windows 10, в составе инструмента «Дисковое пространство», который позволяет создавать программный RAID из нескольких физических дисков. С выходом Windows Server 2016 ФС была улучшена, вскоре она будет доступна в новой версии Windows 10. Эта ФС защи-

щает данные от искажения, для чего использует контрольные суммы для метаданных, а также может использовать контрольные суммы для данных файла. Во время чтения или записи файла, система проверяет контрольную сумму, чтобы убедиться в ее правильности. Таким образом обнаруживаются искаженные данные в режиме реального времени.

ReFS более современна и поддерживает гораздо большие объемы и более длинные имена файлов, чем NTFS. В долгосрочной перспективе это важные улучшения. В ФС NTFS имя файла ограничено 255 символами, в ReFS имя файла может содержать до 32 768 символов. Windows 10 позволяет отключить ограничение на предел символов для ФС NTFS, но он всегда отключается на томах ReFS.

NTFS имеет теоретический максимальный объем 16 Эбайт, а у ReFS теоретический максимальный объем 262 144 Эбайт [26].

В системах Linux можно обнаружить больший выбор ФС. К основным локальным ФС Linux относят:

1) ext2 (*Second Extended FS* — вторая расширенная файловая система) [27] — появилась в апреле 1992 г., это была первая ФС, изготовленная специально под нужды Linux. Разработана Реми Кардом с целью преодолеть ограничения файловой системы в Minix;
2) ext3 (*Third Extended FS* — третья расширенная файловая система) — разработана Стивеном Твиди в 1999 г., включена в основное ядро Linux в ноябре 2001 г. Является надстройкой над ext2, в нее добавлено журналирование файловых операций. Существует возможность на лету (без перезагрузки) конвертировать ФС ext2 в ФС ext3. В сравнении с другими новейшими разработками, обладает более скромным размером пространства, до 4 Тбайт ($4 \cdot 2^{40}$ байт) для 32-разрядных систем. В данный момент является наиболее стабильной и поддерживаемой ФС в среде Linux;
3) ext4 (*Fourth Extended FS*) — это попытка создать 64-битную ext3, способную поддерживать больший размер ФС (1 Эбайт). В нее также были добавлены возможности — выделение пространства не отдельными блоками, а экстентами (непрерывными последовательностями блоков), увеличение диапазона и точности временных меток, дефрагментация на лету и пр. Обеспечивается прямая совместимость и частично обратная совместимость с системой ext3 [28];
4) XFS — ФС с поддержкой журнализации, созданная компанией Silicon Graphics в 1993 г. для ОС IRIX, впоследствии была пере-

несена в Linux. С 2001 г. код XFS распространяется по GPL. Эта ФС оптимизирована для работы с файлами большого размера (максимальный допустимый размер файла — 9 млн Тбайт) [29];

5) JFS (*Journaled File System* — журналируемая ФС) — производительная 64-битная файловая система от IBM со встроенной поддержкой журнализации. Используется в основном в серверах фирмы IBM. Существуют версии для ОС AIX, Linux и OS/2. Распространяется по лицензии GPL. Включена в состав ядра Linux в начале 2000-х гг. Достоинство системы — неплохая масштабируемость, недостаток не особо активная поддержка на протяжении всего жизненного цикла. Максимальные размеры: файла — 4 Пбайта; файловой системы — 32 Пбайта [30];
6) Reiserfs и Reiserfs4 — файловая система ReiserFS была разработана специально для ОС Linux Гансом Райзером и компанией Namesys. Она стала первой журналируемой ФС. Это эффективная ФС, которая очень быстро работает с небольшими файлами (таких системных файлов достаточно много). В основе структуры этой системы лежат деревья. Максимальный размер файла составляет 1 Эбайт, а количество файлов, которые находятся на одном разделе, — около 4 млрд. Позволяет упаковывать «хвосты» файлов [31]. К сожалению, в связи с печальной судьбой ее создателя, у этой ФС нет будущего;
7) BTRFS (*B-Tree File system*) — ФС для Unix-подобных ОС, основана на технике «Copy on Write» (CoW), которая призвана обеспечить легкость масштабирования ФС, ее высокую надежность и сохранность данных, гибкость настроек и простоту администрирования, при этом сохраняется высокая скорость работы ФС. BTRFS — это молодая и современная ФС, которая призвана решать широкий круг задач. Она активно развивается, поэтому некоторые ее характеристики значительно отличаются от версии к версии. Максимальный размер файла, поддерживаемый данной ФС, равен 2^{64} байт [32];
8) ZFS (*Zettabyte File System*) — ФС, изначально созданная в Sun Microsystems (сейчас принадлежит Oracle) для ОС Open Solaris в ноябре 2005 г. Особенности ZFS включают: пулы хранения, копирование при записи, снимки ФС, контроль целостности данных и автоматическое восстановление (scrubbing), максимальный объем файла 16 Эбайт и максимум в $256 \cdot 10^{15}$ Збайт хранилища

без ограничения на количество ФС или файлов. ZFS лицензирована под Common Development and Distribution License (CDDL). Описывается как последнее слово файловых систем [33]. ZFS стабильная, быстрая, безопасная, с заделом на будущее.

Сравнение ФС, работающих в ОС Linux приведено в таблице.

Сравнение ФС, работающих в ОС Linux

ФС	Размер блока	Максим. размер ФС	Максим. размер файла
Ext3FS	От 1 до 4 Кбайт	4 Тбайта	2 Тбайта
XFS	От 512 байт до 64 Кбайт	18 тыс. Пбайт	9 млн. Тбайт
JFS	512, 1024, 2048, 4096 байт	От 4 Пбайт (при 512-байтных блоках) до 32 Пбайт (при 4-Кбайтных блоках)	От 512 Тбайт (при 512-байтных блоках) до 4 Пбайт (при 4-Кбайтных блоках)
ReiserFS	Предполагалась до 64 Кбайт (пока фиксирован, 4 Кбайта)	9 000 Пбайт	1 Эбайт

Тип ФС в Linux можно узнать несколькими способами. Один из них — использование команды df (выдает сведения о смонтированных ФС). Пример ввода этой команды в окне терминала Ubuntu приведен на рис. 7.7. В данном случае есть раздел с четвертой расширенной ФС (ext4), также на диске обнаружен раздел, работающий под Windows (vfat).

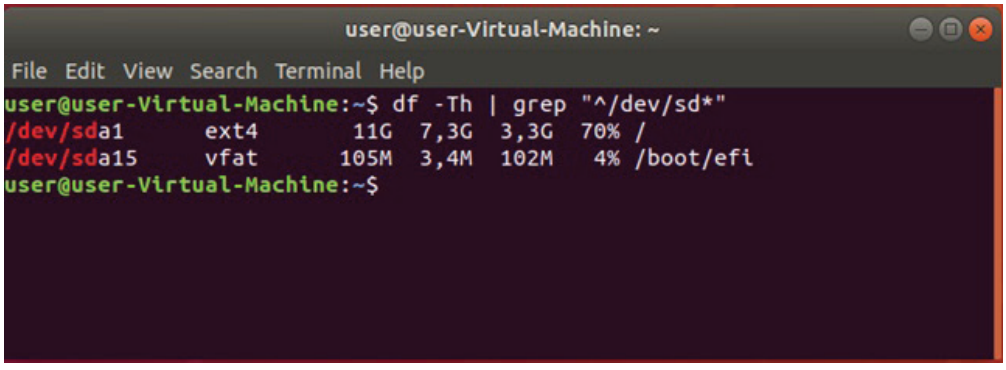


Рис. 7.7. Работа команды df

Примеры файловых систем

Организация ФС FAT

FAT и NTFS — это основные ФС, работающие с ОС Windows.

Рассмотрим сначала ФС FAT, т. к. она появилась раньше, и попытаемся понять, почему Microsoft постепенно отказалась от этой системы и разработала новую — NTFS.

Название системы получено от ее основной системной структуры — FAT (*File Allocation Table* — таблица размещения файлов). Эта таблица содержит информацию о размещении файлов на диске. Вся остальная информация о файлах — их атрибуты — содержатся в каталогах.

Версии существуют следующие: FAT12, FAT16, FAT32, где 12, 16, 32 — это разрядность элементов в таблице FAT. Логический раздел, отформатированный под ФС FAT12 (FAT16), состоит из следующих областей (рис. 7.8):

1) загрузочный сектор содержит программу начальной загрузки ОС (если этот раздел активный);
2) основная копия FAT содержит информацию о размещении файлов и каталогов на диске;
3) резервная копия FAT;
4) корневой каталог занимает фиксированную область размером в 32 сектора (16 Кбайт), что позволяет хранить 512 записей о файлах и каталогах, поскольку каждая запись каталога состоит из 32 байт (это ограничение снято в ФС FAT32);
5) область данных — предназначена для размещения всех файлов и всех каталогов (кроме корневого).

ФС FAT поддерживает типы файлов: обычный файл и каталог. ФС распределяет память только из области данных (рис. 7.8), причем в качестве минимальной единицы дискового пространства используется кластер.

Рассмотрим структуру таблицы FAT. В таблице каждому кластеру на диске отведена одна ячейка (один элемент). Элементы пронумерованы, нумерация сквозная и соответствует логическим номерам кластеров.

Первые два кластера на диске выделены под системную информацию, поэтому для размещения данных выделяются кластеры начиная со второго.

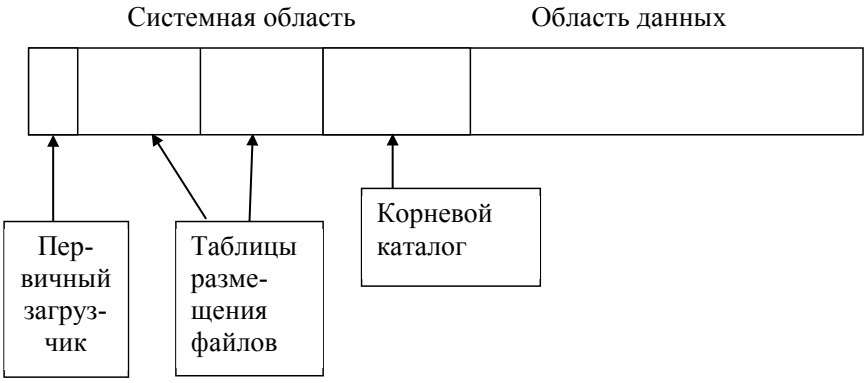


Рис. 7.8. Структура логического диска, отформатированного под FAT16

Элемент в таблице FAT может принимать следующие значения, характеризующие состояния связанного с ним кластера:

1) кластер свободен (не выделен ни одному файлу);
2) кластер используется файлом и не является последним кластером файла, в этом случае элемент таблицы содержит номер следующего кластера файла;
3) последний кластер файла;
4) дефектный кластер;
5) резервный кластер.

В исходном состоянии (после форматирования) все кластеры ФС свободны и все элементы таблицы FAT принимают значение «кластер свободен». При размещении файла, ОС просматривает FAT и ищет первый свободный элемент. Найдя такой элемент, ОС записывает в соответствующий кластер порцию данных, а номер соответствующего элемента фиксирует в каталоге. Если размер файла больше размера кластера, то продолжается просмотр таблицы для поиска очередного свободного элемента таблицы. После его обнаружения, номер этого элемента заносится в номер предыдущего элемента, и т. д., таким образом получается структура, известная как связный список. Так происходит до тех пор, пока не закончатся все данные для файла. В этом случае в последний элемент списка элементов FAT заносится признак конца файла.

При обращении к существующему файлу выполняется обратная процедура. Элемент каталога, содержащий информацию о файле, содержит номер первого кластера файла. В FAT элемент для этого кла-

стера содержит адрес следующего кластера. Такая цепочка (прохождение по списку) будет продолжаться, пока не будет достигнут последний кластер файла, элемент FAT которого содержит специальный код, означающий конец файла (*End-of-File* — EOF).

Предположим, что элемент каталога для файла «Письмо.docx» показывает, что файл начинается в кластере 5. Найдем кластер 5 на рис. 7.9. Данный элемент содержит число 6, поэтому продолжение файла нужно искать в кластере 6. Согласно FAT, из шестого кластера нужно перейти в десятый, а из десятого — в одиннадцатый. В элементе FAT для одиннадцатого кластера находится маркер конца файла. Таким образом, этот файл занимает 4 кластера, что должно согласовываться с размером файла, указанным в элементе каталога.

2	3	4	5	6	7	8	9	10	11	12	13
0	0	EOF	6	10	8	EOF	0	11	EOF	15	…

Рис. 7.9. Пример таблицы размещения файлов (FAT)

Файл «Письмо.docx» является примером фрагментированного файла.

Размер таблицы FAT и разрядность используемых в ней элементов определяются версией ФС. Можно подсчитать максимальную емкость диска, поддерживаемую той или иной версией этой ФС, по формуле

$$2^{\text{разрядность}} \cdot \text{размер кластера}, \qquad (7.1)$$

где $2^{\text{разрядность}}$ — определяет максимально возможный номер кластера (исходя из размерности элемента таблицы FAT).

Например, для системы FAT16 этот расчет максимального объема дискового пространства следующий (учитывая, что максимально возможный размер кластера равен 64 Кбайта):

$$V = 2^{16} \cdot 64 \text{ Кбайт} = 2^{16} \cdot 2^{6} \cdot 2^{10} = 2^{32} = 2^{2} \cdot 2^{30} = 4 \text{ Гб}. \qquad (7.2)$$

Итак, FAT16 может быть установлена в разделе, размер которого не более 4 Гбайта.

При удалении файла из ФС FAT, на первом этапе он просто перерегистрируется в другой каталог с именем «Корзина». В первый байт этого дубликата записи заносится специальный символ, являющийся индикатором того, что данные из файла подлежат удалению. По запросу пользователя или переполнению Корзины производится фактиче-

ское удаление данных с диска, при этом все элементы таблицы FAT, отведенные под файл, обнуляются, а с течением времени выделяются для размещения другого файла.

ФС FAT просуществовала в Windows довольно длительный период времени. Если оценивать FAT по приведенным ранее критериям, то можно отметить следующее:

1) скорость доступа к данным не высока, т. к. данные располагаются отдельно от адресной информации, кроме того, необходимо формировать список кластеров по таблице размещения файлов;
2) объем системной информации — по этому критерию FAT можно считать одной из наиболее эффективных, т. к. данный объем небольшой, т. е. больше места можно использовать для хранения пользовательских данных при том же размере ФС;
3) склонность к фрагментации высокая, т. к. пространство выделяется отдельными кластерами в разные моменты времени (например, когда редактируется файл, в него вводится новая информация);
4) максимально возможный размер файла по современным требованиям недостаточный (см. расчет для FAT16, хотя в FAT32 он значительно больше), в ранних версиях также существовало ограничение на размер корневого каталога.

Кроме того, современные ОС — это многопользовательские ОС, т. е. в ФС этих ОС должна быть предусмотрена защита файлов одного пользователя от несанкционированного доступа другого пользователя, чего не было предусмотрено при разработке данной ФС.

Программный код, поддерживающий функционирование такой ФС, достаточно эффективен и хорошо отлажен, сбои в работе крайне редки, но существуют внешние причины (например, перебои с электроэнергией), которые могут привести к потере данных, а свойством восстанавливаемости (о котором речь пойдет дальше) данная ФС не обладает. Поэтому для Windows была разработана другая ФС — это NTFS (*New Technology File System* — ФС новой технологии).

Файловая система NTFS

Как большинство ФС, NTFS делит все полезное место на кластеры. NTFS поддерживает размеры кластеров в пределах от 512 байт до 64 Кбайт, при этом стандартом считается кластер размером 4 Кбайта.

В NTFS вместо понятия «логический диск» используется понятие «том». Основной системной структуры тома NTFS является главная таблица файлов (*Master File Table*). MFT содержит одну или несколько записей для каждого файла, включая запись для себя самой. Каждая запись MFT имеет фиксированную длину, которая зависит от объема диска, обычно это записи в 1, 2 или 4 Кбайта. Для большинства современных дисков, размер записи MFT равен 2 Кбайта, он и считается размером по умолчанию [34].

Том условно разделен на две части: начальные 12 % диска отведены под MFT-зону — пространство, в котором размещается метафайл MFT. Запись каких-либо пользовательских данных в эту область невозможна. MFT-зона не дополняется никакими данными, даже если есть свободное место для того, чтобы служебный файл MFT не фрагментировался при своем росте. Остальные 88 % диска отведены для хранения файлов.

«ФС NTFS представляет собой выдающееся достижение структуризации: каждый элемент системы представляет собой файл — даже служебная информация» [35]. Хранение всего в виде файлов облегчает ФС поиск и управление данными, а доступ к каждому отдельному файлу может быть ограничен за счет использования дескриптора защиты (структура данных, хранящая информацию о правах доступа различных пользователей). В случае повреждения участка диска, выделенного для хранения системных данных, NTFS может переместить эти файлы метаданных и вся информация на диске по-прежнему будет доступной.

Первые 16 файлов NTFS (метафайлы, или файлы метаданных) носят служебный характер. Каждый из них отвечает за определенный аспект работы системы. Преимущество этого модульного подхода заключается в его гибкости, в отличие от NTFS, например, в ФС FAT физическое повреждение области, где размещены таблицы FAT, фатально для работы всего диска.

Метафайлы находятся корневом каталоге NTFS диска, их имена начинаются с символа «$»; следует заметить, что получить какую-либо информацию о них стандартными средствами практически невозможно.

Структура тома с установленной NTFS показана на рис. 7.10.

Рис. 7.10. Структура тома с NTFS

Все файлы на томе NTFS однозначно определяются номером строки, выделенной для хранения атрибутов файла, в MFT.

Весь том в NTFS представляется последовательностью кластеров. Порядковый номер кластера в томе NTFS называется логическим номером кластера (*Logical Cluster Number*, *LCN*). Любой файл в NTFS состоит из последовательности кластеров, при этом порядковый номер кластера внутри файла называется виртуальным номером кластера (*Virtual Cluster Number*, *VCN*). Базовая единица распределения дискового пространства в NTFS — отрезок. **Отрезок** — это непрерывная область кластеров, выделяемая для размещения файла. В качестве адреса отрезка NTFS использует логический номер его первого кластера, а также количество кластеров в отрезке (LCN, k). Часть файла, раз-

мещенная в отрезке и начинающаяся с кластера с виртуальным номером VCN, характеризуется адресом, который представлен тройкой чисел: (VCN, LCN, k). Для хранения номера кластера в NTFS используется 64 разряда, что дает возможность поддерживать тома и файлы размером до 2^{64} кластеров. Если размер кластера, например, 4 Кбайта, то появляется возможность использовать тома и файлы, состоящие из $(64 \cdot 2^{70})$ байт.

В MFT имеется запись для каждого файла на диске, включая запись для самой себя. Файловая запись MFT содержит либо все атрибуты данного файла, либо адреса отрезков, которые определяют, где на диске расположены значения нерезидентных атрибутов файла.

Первой записью в MFT является запись для самой себя. Вторая запись соответствует файлу, находящемуся в середине диска и содержащему копию первых 16 записей MFT. Эта неполная копия MFT используется для поиска метафайлов в том случае, если часть файла MFT по какой-либо причине испорчена.

После того как файловая запись для MFT найдена, NTFS считывает адреса ее атрибута данных и сохраняет это в ОП. Данная информация говорит NTFS, где на диске находятся отрезки, составляющие MFT. Затем NTFS находит записи для остальных метафайлов и открывает эти файлы. Далее NTFS выполняет операцию восстановления ФС (описана далее). Теперь том готов к использованию.

В процессе работы NTFS записывает информацию в еще один важный метафайл — журнал транзакций (его имя –"$LogFile"). В журнале транзакций регистрируются все операции, влияющие на изменение структуры тома. Журнал транзакций используется при восстановлении тома NTFS после сбоя системы (см. гл. 8).

Одна из записей MFT зарезервирована для корневого каталога. Получив первый запрос на открытие некоторого файла, NTFS начинает поиск этого файла с записи в MFT, соответствующей корневому каталогу. После того как файл найден, NTFS запоминает номер его записи в таблице, чтобы при последующих операциях с данным файлом обращаться к этой записи напрямую.

Схема занятости дискового пространства под том NTFS хранится в файле битовой карты (*bitmap file*). Этот файл представляет собой последовательность битов, где каждый бит отражает состояние соответствующего (с тем же логическим номером) кластера на диске: значение бита 0 — кластер свободен, 1 — кластер занят.

Еще один важный системный файл — загрузочный файл (*boot file*), который хранит код начального загрузчика Windows (если схема загрузки BIOS — MBR).

Есть метафайл, содержащий таблицу определения атрибутов (*attribute definition table*), в которой задаются типы атрибутов, поддерживаемые в этой системе.

В NTFS данные файла могут быть целиком размещены в записи таблицы MFT, если файл такого размера, который это позволяет (включаются в запись как атрибут данных). Если размер файла больше размера записи MFT, в запись помещаются определенные атрибуты файла для хранения остальных атрибутов, в т. ч. данных, выделяется один или несколько отрезков в области данных. Часть файла, размещенная в записи MFT, называется резидентной частью, а те части, которые размещены в области данных, — нерезидентными. При этом адреса отрезков, содержащих нерезидентные части файла, размещаются в атрибутах резидентной части. Сортировка может осуществляться только по резидентным атрибутам. Некоторые системные файлы являются полностью резидентными, а другие имеют и нерезидентные части, которые располагаются после MFT в области данных.

Каждый файл и каталог на томе NTFS также представлен набором атрибутов, два из которых — это его имя и данные. Часть атрибутов системные, они определяются структурой тома NTFS. Системные атрибуты имеют стандартные имена и коды типа и определенный формат. Часть атрибутов может быть определена пользователями. Здесь также применяется два вида хранения — резидентное (в записи MFT) и нерезидентное — в области данных.

В качестве системных атрибутов для примера можно указать следующие:

1) File Name (имя файла) — этот атрибут содержит длинное имя файла в формате Unicode, а также номер входа в таблице MFT для каталога, где он зарегистрирован; если такой файл зарегистрирован в нескольких каталогах (несколько жестких ссылок), то у него будет несколько атрибутов типа File Name; данный атрибут всегда должен быть резидентным;
2) Version (версия) — атрибут содержит номер последней версии файла;
3) Security Descriptor (дескриптор безопасности) — этот атрибут содержит информацию о защите файла: список прав доступа и поле

аудита, которое определяет, какого рода операции над этим файлом нужно регистрировать;

4) Data (данные) — содержит обычные данные файла;
5) Index Root (корень индекса) — корень B-дерева, используемого для поиска файлов в каталоге;
6) Index Allocation (размещение индекса) — нерезидентные части индексного списка B-дерева;
7) Standard Information (стандартная информация) — этот атрибут хранит всю остальную стандартную информацию о файле, например, время создания файла, время обновления, стандартные атрибуты (только для чтения, скрытый системный, архивный, сжатый, шифрованный) и др.

В зависимости от способа размещения, файлы NTFS делят на небольшие, большие, очень большие и сверхбольшие.

Небольшие файлы (*small*). Если файл имеет небольшой размер, то он может целиком располагаться внутри одной записи MFT, имеющей, например, размер 2 Кбайта. Небольшие файлы NTFS состоят по крайней мере из следующих атрибутов (рис. 7.11):

1) стандартная информация (SI — Standard Information);
2) имя файла (FN — File Name);
3) данные (Data);
4) дескриптор безопасности (SD — Security descriptor).

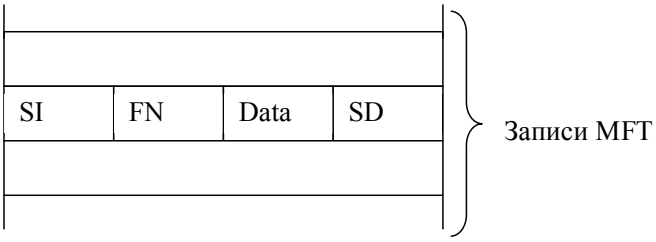


Рис. 7.11. Небольшой файл в NTFS

Большинство файлов имеет больший размер, чем может уместиться в записи MFT, кроме того, у файла может быть достаточное количество атрибутов, однако обычно файлы размером менее 1500 байт помещаются внутри записи MFT (размером 2 Кбайта).

Большие файлы (*large*). Если данные файла не помещаются в одну запись MFT, то это отражено в заголовке атрибута данных (с име-

нем Data) как специальный признак. В таком случае атрибут данных содержит адресную информацию отрезков в виде (LCN, VCN, k) (рис. 7.12).

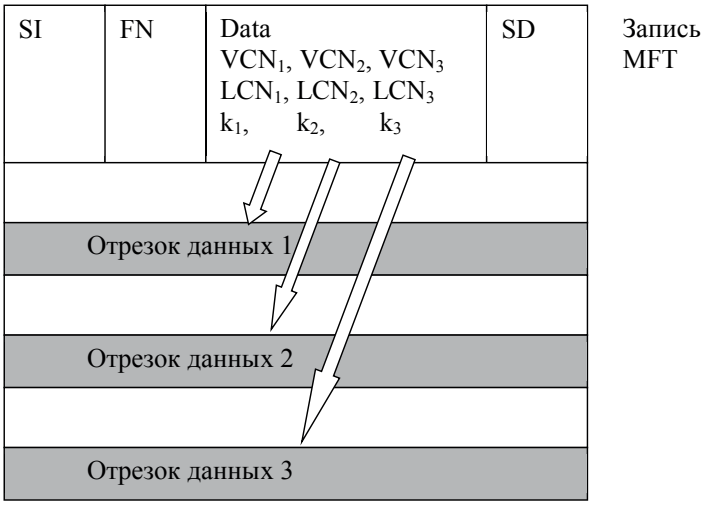


Рис. 7.12. Организация большого файла в NTFS

Очень большие файлы (*huge*). Если файл очень большого размера — такой, что его атрибут данных, хранящий адреса отрезков данных, не помещается в одной записи, то для хранения адресов отрезков выделяется дополнительная запись в MFT, а ее номер указывается в основной записи файла. Этот номер содержится в атрибуте Attribute List.

Сверхбольшие файлы (*extremely huge*). Для сверхбольших файлов в атрибуте Attribute List можно указать несколько дополнительных записей MFT, выделенных для хранения адресов отрезков. В этом случае можно также использовать двойную косвенную адресацию, когда один нерезидентный атрибут ссылается на несколько других нерезидентных атрибутов.

Каждый каталог NTFS также представлен в виде одной (или нескольких) записи в таблице MFT. Записи каталогов вместо атрибута данных содержат атрибуты с именем *Index Root*. Эти атрибуты (индексы) содержат списки файлов, зарегистрированных в соответствующих каталогах. Индексы обеспечивают возможность сортировки файлов по определенному атрибуту, что значительно ускоряет поиск. В зависимости от формы хранения списка файлов различают небольшие и большие каталоги.

Небольшие каталоги. Если количество файлов в каталоге невелико, то их список может быть резидентным в записи в MFT (рис. 7.13). Для резидентного хранения списка используется уже упоминавшийся атрибут с именем *Index Root.* Этот индексный атрибут, как правило, содержит имя файла и номер записи в MFT, содержащей основные атрибуты файла.

SI	FN	IR <a.bat, 27>, <c.sys, 92> <zyx, N> <…><### >	SD

Рис. 7.13. Организация небольшого каталога в NTFS:
— признак конца файлов

Большие каталоги. Каталог со времен растет, все больше объектов в нем регистрируется, и этот список может потребовать нерезидентной формы хранения. При этом начало списка всегда резидентно в основной записи каталога в таблице MFT. Имена объектов (файлов и подкаталогов) резидентной части списка являются узлами структуры, известной как B-дерево (двоичное дерево). Остальные части списка файлов размещаются в отрезках в области данных. Для их поиска используется атрибут с именем *Index Allocation,* содержащий адреса этих нерезидентных отрезков. Одни части списков являются листьями дерева, а другие — промежуточными узлами, содержащими наряду с именами файлов атрибут *Index Allocation,* указывающий на списки более низких уровней.

Поиск в таком каталоге уникального имени файла (номера основной записи о файле в MFT) по его символьному имени происходит следующим образом. Сначала символьное имя файла сравнивается с именем первого узла в резидентной части индекса. Если искомое имя меньше, то начинается поиск в первой нерезидентной группе, для этого из атрибута Index Allocation извлекается адрес отрезка (VCN_1, LCN_1, k_1), в котором хранятся имена файлов первой группы. Среди имен этой группы поиск осуществляется прямым перебором. При совпадении поискового имени и имени в каталоге извлекается номер основной записи о файле в MFT, остальные атрибуты файла извлекаются уже из этой записи. Если же искомое имя больше имени первого узла резидентной части индекса, то оно сравнивается с именем второго узла, и если это имя меньше, то описанная выше про-

цедура перебора и сравнения применяется ко второй нерезидентной группе имен, и т. д.

В результате уменьшается количество операций просмотра и сравнения, а значит, ускоряется поиск, а с ним и производительность работы файловой системы.

Контрольные вопросы

1. Что такое файловая система и из чего она состоит?
2. Приведите примеры современных ФС.
3. Определите, какие ФС и в каких разделах дисков установлены на вашем ПК.
4. Какое минимальное пространство можно выделить под размещение отдельного файла?
5. Зачем введено понятие кластера (блока) в файловых системах?
6. Какие вы знаете структуры данных, которые используются в файловых системах?
7. Рассчитайте максимальный размер раздела, поддерживаемый ФС FAT32.
8. Оцените эффективность ФС NTFS, используя стандартные критерии оценки файловых систем. В чем преимущество ФС NTFS по сравнению с ФС FAT?
9. В виде каких единиц выделяется дисковое пространство в ФС NTFS, и какими — в FAT? В чем достоинства и недостатки каждого из способов?
10. Какие метаданные хранятся на любом из томов с ФС NTFS?

8. Отказоустойчивость дисковых систем и восстанавливаемость файловых систем

Диски и ФС, используемые для упорядоченного хранения данных на дисках, часто представляют собой последний островок стабильности, на котором находит спасение пользователь после неожиданного краха системы, разрушившего результаты его труда, полученные за последние нескольких минут или даже часов, но не сохраненные на диске. Однако диски также могут отказывать, например, по причине нарушения магнитных свойств отдельных областей поверхности. В данном разделе будут рассмотрены методы, которые повышают устойчивость вычислительной системы к отказам дисков за счет использования избыточных дисков и специальных алгоритмов управления массивами таких дисков.

Другой причиной недоступности данных после сбоя системы может стать нарушение целостности служебной информации ФС, произошедшее из-за незавершенности операций по изменению этой информации при крахе системы. Примером такого нарушения может служить несоответствие между адресной информацией файла и фактическим размещением его данных. Для борьбы с этим явлением применяются так называемые восстанавливаемые ФС, которые обладают определенной степенью устойчивости к сбоям и отказам компьютера (при сохранении работоспособности диска, на котором расположена данная ФС).

Комплексное применение отказоустойчивых дисковых массивов и восстанавливаемых ФС существенно повышают такой важный показатель вычислительной системы, как общая надежность.

Восстанавливаемость файловых систем

«**Восстанавливаемость ФС** — это свойство, которое гарантирует, что в случае отказа питания или краха системы, когда все данные в оперативной памяти безвозвратно теряются, все начатые файловые операции будут либо успешно завершены, либо отменены безо всяких отрицательных последствий для работоспособности ФС, т. е. в любой момент времени ФС будет находиться в устойчивом состоянии» [5].

Любая операция над файлом (создание, удаление, запись, чтение и т. д.) или модификация структуры каталогов может быть представлена в виде некоторой последовательности подопераций. ФС переходит в неустойчивое состояние тогда, когда не все подоперации выполняемой файловой операции завершены, а выполнена только их часть.

Как правило, свойство восстанавливаемости основано на использовании концепции атомарной транзакции (*atomic transaction*). «**Атомарная транзакция** — это метод обработки изменений в данных, при котором сбои в работе системы не нарушают корректности или целостности этих данных» [4]. Суть атомарной транзакции заключается в том, что определенные операции над данными, именно они называются транзакциями, выполняются по принципу «все или ничего». Транзакцией объявляется операция, изменяющая данные ФС (обычно системные данные) или структуру каталогов тома. Отдельные изменения на диске, составляющие транзакцию, выполняются атомарно (т. е. неделимо): в ходе транзакции на диск должны быть внесены все изменения, если это не так, то ту часть изменений, которые уже сделаны, нужно отменить. Такая отмена называется откатом (*rollback*). После отката ФС возвращается в исходное устойчивое состояние, в котором она была до начала транзакции.

Модель атомарной транзакции пришла из бизнеса. Пусть, например, идет переговорный процесс двух фирм о покупке-продаже некоторого товара. В процессе переговоров условия договора могут многократно меняться, уточняться. Пока договор еще не подписан обеими сторонами, каждая из них может от него отказаться, но после подписания контракта сделка (транзакция) должна быть выполнена от начала и до конца. Если же контракт не подписан, то любые действия, которые были уже проделаны, отменяются или объявляются недействительными.

Для обеспечения восстанавливаемости ФС, в системе существует и ведется журнал транзакций, в нем производится упреждающее протоколирование транзакций. Оно заключается в том, что перед изменением какого-либо блока данных на диске или в дисковом кэше, в журнале производится запись, где отмечается, какая транзакция делает изменения, какой файл и блок изменяются и каковы старое и новое значения изменяемого блока (чтобы можно было как повторить изменение, так и откатить его). Только после успешной регистрации всех подопераций в журнале делаются изменения в исходных блоках.

Если транзакция прерывается, то информация журнала регистрации используется для приведения файлов, каталогов и служебных данных ФС в исходное состояние, т. е. производится откат. Если транзакция фиксируется, то и об этом делается запись в журнал регистрации, а новые значения измененных данных сохраняются в журнале еще некоторое время, чтобы сделать возможным повторение транзакции, если это потребуется.

Действия, не модифицирующие данные (чтение, поиск), не протоколируются. Записываются действия, необходимые как для повтора подопераций, так и для ее отката.

Пусть к ФС поступает запрос на выполнение той или иной операции ввода-вывода. Эта операция включает несколько шагов, связанных с созданием, уничтожением и модификацией объектов ФС. Если все подоперации были благополучно завершены, то транзакция считается завершенной. Это действие фиксируется в журнале, появляется запись фиксации транзакции. Если же одна или более подопераций не успели выполниться из-за сбоя питания или краха ОС, то для обеспечения целостности ФС, все измененные в рамках транзакции данные должны быть возвращены точно в то состояние, в котором они находились до начала выполнения транзакции.

При восстановлении часто возникают ситуации, когда система пытается отменить транзакцию, которая уже была отменена или вообще не выполнялась. При повторении же некоторой транзакции может оказаться, что она уже была выполнена. Учитывая это, необходимо так определить подоперации транзакции, чтобы многократное выполнение каждой из этих подопераций не имело никакого добавочного эффекта по сравнению с первым выполнением этой подоперации. Такое свойство операции называется идемпотентностью (*idempoyency*). Примером идемпотентной операции может служить, например, многократное присвоение переменной некоторого значения.

Восстанавливаемость NTFS

ФС NTFS является восстанавливаемой; следует отметить, что восстанавливаемость поддерживается только для системных данных, т. е. для файлов метаданных и каталогов. Сохранность данных из пользовательских файлов, с которыми вы работали в момент сбоя, не гарантируется.

Журнал регистрации транзакций в NTFS имеет две части: область рестарта и область протоколирования (рис. 8.1):

1) область рестарта содержит информацию о том, с какого места необходимо будет начать читать журнал транзакций для проведения процедуры восстановления системы после сбоя или краха ОС;
2) область протоколирования содержит записи обо всех изменениях в системных данных ФС, произошедших в результате выполнения транзакций в течение некоторого достаточно большого периода. Все записи имеют логический последовательный номер LSN (*Logical Sequence Number*). Записи о подоперациях одной операции, объявленной транзакцией, объединены в связный список. Запись в область протоколирования происходит циклически: если выделенная под область протоколирования память заполнена, то новые записи начинают замещать наиболее старые.

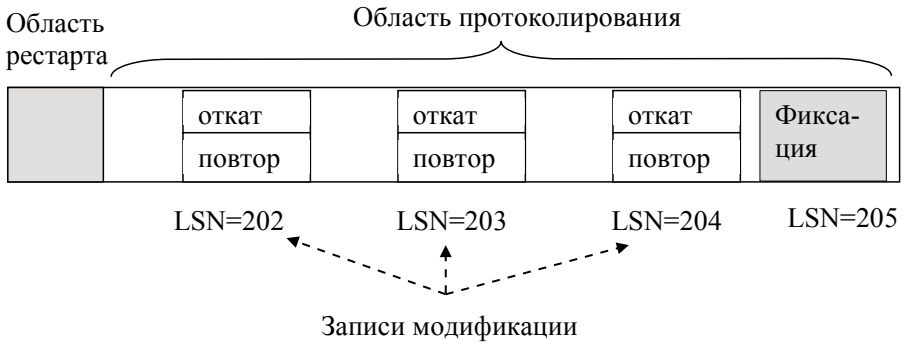


Рис. 8.1. Структура журнала транзакций в ФС NTFS

Различают следующие типы записей в журнале транзакций: запись модификации, запись контрольной точки, запись фиксации транзакции, запись таблицы модификации, запись таблицы модифицированных страниц.

Каждая запись модификации соответствует одной подоперации, модифицирующей системные данные. Эта запись составлена из двух частей: первой части — с инструкцией, благодаря которой система может повторить эту подоперацию, и второй части — с инструкцией для ее отмены.

Пусть, например, регистрируется операция-транзакция создания файла text1.docx, которая состоит из трех подопераций (см. рис. 8.1). В журнале соответствующая служба создаст три записи модификации (табл. 8.1).

Таблица 8.1

Записи модификации при создании файла

Запись модификации	Информация для повторения транзакции	Информация для отката транзакции
LSN=202	Выделить и инициировать запись для нового файла text1.docx в таблице MFT	Удалить запись о файле text1.docx из таблицы MFT
LSN=203	Добавить имя файла в индекс (каталог)	Исключить имя файла из индекса (каталога)
LSN=204	Установить биты с 3-го по 9-й в битовой карте в 1	Обнулить биты с 3-го 9-й в битовой карте

ФС NTFS все действия с журналом транзакций выполняются специальной службой LFS (*Log File System*). Она создает в журнале новые записи, осуществляет запись самого журнала на диск, при восстановлении считывает записи в прямом и обратном порядке и выполняет ряд других действий над записями в журнале.

Прежде чем выполнить любую транзакцию, NTFS вызывает службу журнала транзакций LFS для регистрации всех подопераций в журнале транзакций. Когда все подоперации транзакции выполнены, с помощью службы LFS транзакция фиксируется. Это выражается в том, что в журнал заносится специальный вид записи — запись фиксации транзакции.

NTFS создает новые системные таблицы: модифицированных страниц и незавершенных транзакций, делается это на основании записей журнала транзакций. Системные таблицы представляют собой новую форму представления информации, удобную для проведения процесса восстановления.

Запись контрольной точки в журнал выполняется регулярно, что позволяет держать в актуальном состоянии области рестарта, и систе-

ма при необходимости может достаточно точно определить, с какой записи в журнале начинать процедуру восстановления.

Процесс восстановления ФС состоит из следующих шагов:

1) извлечения области рестарта из файла журнала транзакций и идентификации номера последней по времени записи о контрольной точке;
2) чтения записи контрольной точки, создания таблицы незавершенных транзакций и таблицы модифицированных страниц;
3) анализа таблицы модифицированных страниц, определения номера первой по времени записи модификации страницы;
4) чтения журнала транзакций в прямом направлении, начиная с первой записи модификации, найденной при анализе таблицы модифицированных страниц. При этом система повторно выполняет все зафиксированные транзакции, в результате чего устраняются все несоответствия в ФС, которые были вызваны потерями модифицированных страниц во время сбоя, произошедшего в системе;
5) анализа таблицы незавершенных (незафиксированных) транзакций, определение номера самой поздней подоперации, выполненной в рамках такого типа транзакции;
6) чтения журнала транзакций в обратном направлении, отката незавершенных транзакций. Благодаря тому что все подоперации одной транзакции образуют связный список, система осуществляет быстрый переход от записи к записи и извлечение из них нужной для отмены изменений информации.

Операция отмены также объявляется транзакцией, т. к. она связана с модификацией системных данных ФС, она также протоколируется в журнале транзакций. По отношению к ней применимы те же процедуры повторения или отката.

Избыточные дисковые массивы RAID

В основе средств обеспечения отказоустойчивости дисковой памяти лежит принцип избыточности, который основан на использовании дисковых подсистем RAID — *Redundant Array of Inexpensive Disks* (с англ. «избыточный массив недорогих дисков»). Создателями RAID являются

Д. Паттерсон (David A. Patterson), Г. Гибсон (Garth A. Gibson) и Р. Катц (Randy H. Katz). Впервые RAID были использованы в 1987 г. [36].

Основная идея технологии состоит в том, что для хранения данных используют несколько дисков даже в тех случаях, когда для таких данных хватило бы объема одного диска. Совместное использование нескольких централизованно управляемых дисков позволяет получить новые свойства, которые отсутствуют у каждого отдельного диска.

RAID-массив может быть организован двумя различными способами: первый — программный, когда в состав ОС включается специальный драйвер, обеспечивающий отказоустойчивость на базе нескольких обычных дисковых устройств, которые управляются обычными контроллерами; второй способ аппаратный, в этом случае технология RAID полностью реализуется аппаратными средствами, в таких системах есть специальный аппаратный контроллер.

Для пользователей и прикладных программ дисковый массив RAID представляется единым логическим устройством. Такое устройство может обладать различными качествами в зависимости от алгоритмов централизованного управления и способов размещения информации на дисках. Массивы RAID создаются с двумя разными целями: создать отказоустойчивую систему и повысить производительность ее работы.

Калифорнийский университет в Беркли представил следующие уровни спецификации RAID, которые были приняты как стандарт де-факто [37]:

1) RAID-1 — зеркальный дисковый массив;
2) RAID-2 — массивы, в которых использован код Хемминга;
3) RAID-3 и RAID-4 — дисковые массивы с чередованием и выделенным диском четности;
4) RAID-5 — дисковый массив с чередованием и отсутствием выделенного диска четности.

В современных RAID-контроллерах предоставлены дополнительные уровни спецификации RAID:

1) RAID-0 — дисковый массив повышенной производительности с чередованием, без отказоустойчивости.;
2) RAID-6 — дисковый массив с чередованием, использующий две контрольные суммы, вычисляемые двумя независимыми способами;
3) RAID-10 — массив RAID-0, построенный из массивов RAID-1;
4) RAID-01 — массив RAID-1, построенный из массивов RAID-0 (имеет низкую отказоустойчивость);

5) RAID-1E (зеркало из трех устройств), RAID-50 (массив RAID-0 из массивов RAID-5), RAID-05 (RAID-5 из RAID-0), RAID-60 (RAID-0 из RAID-6) и различные другие.

Существует спецификация, изданная негосударственной международной организацией SNIA (*Storage Networking Industry Association*), в которой описаны все уровни RAID — «Common RAID Disk Data Format Specification» [38].

При оценке эффективности RAID-массивов чаще всего используются следующие критерии:

1) степень избыточности хранимой информации (иногда вместо этого используют стоимость хранения единицы информации);
2) производительность операций чтения и записи;
3) степень отказоустойчивости.

Следует отметить, что первый критерий тесно связан с третьим, т. к. свойство отказоустойчивости тесно связано с избыточностью хранимой информации: где нет избыточности, не возникает и отказоустойчивости.

Далее рассмотрены основные уровни RAID.

Уровень RAID-0

В логическом устройстве RAID-0 (рис. 8.2), при выполнении операции записи, данные расщепляются на блоки и передаются параллельно на все диски: первый блок данных записывается на первый диск, второй — на второй и т. д. Различные варианты реализации технологии RAID-0 могут отличаться размерами блоков данных, например, в Windows такие блоки равны 64 Кбайтам. При чтении контроллер (или драйвер) мультиплексирует блоки данных, поступающие со всех дисков, и передает их источнику запроса.

Уровень RAID-0 не обладает избыточностью данных, из этого следует, что у данной конфигурации отсутствует отказоустойчивость. По сравнению с отдельным диском, в котором данные записываются и считываются последовательно, производительность дисковой конфигурации RAID-0 значительно выше за счет возможности одновременной записи (чтения) на все диски массива. Цель создания таких систем — повышение производительности системы.

Достоинство системы состоит в том, что скорость считывания файлов увеличивается в n раз, где n — количество дисков; при этом такая оп-

тимальная производительность достигается только для больших запросов, когда фрагменты файла находятся на каждом из дисков. Недостатки ее следующие:

1) существует угроза потери всех данных, если при записи произойдет сбой. Если один из дисков выйдет из строя, то восстанавливать придется все диски массива;
2) сложность увеличения емкости системы. Если объем логического устройства нужно увеличить, то необходимо полностью перераспределить информацию по всему изменившемуся набору дисков;
3) увеличивается вероятность потери данных — если вероятность отказа одного диска равна p, то вероятность выхода из строя массива RAID-0 из двух дисков рассчитывается как $(2p - p \cdot p)$. Таким образом, если вероятность отказа одного диска за год равна 1 %, то вероятность отказа массива RAID-0 из двух дисков составляет 1,99 %, т. е. практически в два раза больше.

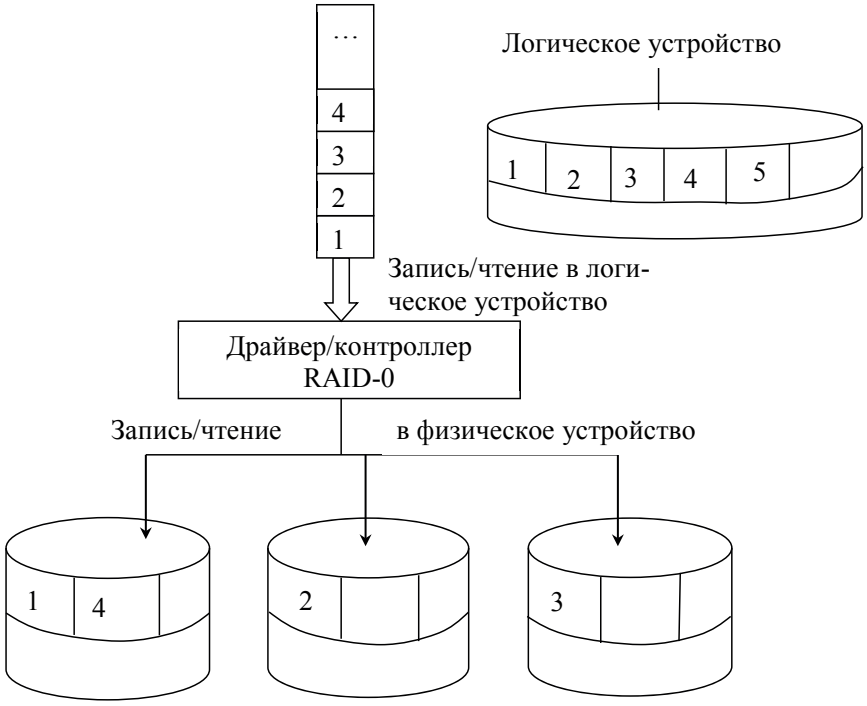


Рис. 8.2. Организация массива RAID-0

Уровень RAID-1

Уровень RAID-1 (рис. 8.3) реализует подход, называемый зеркалированием (*mirroring*).

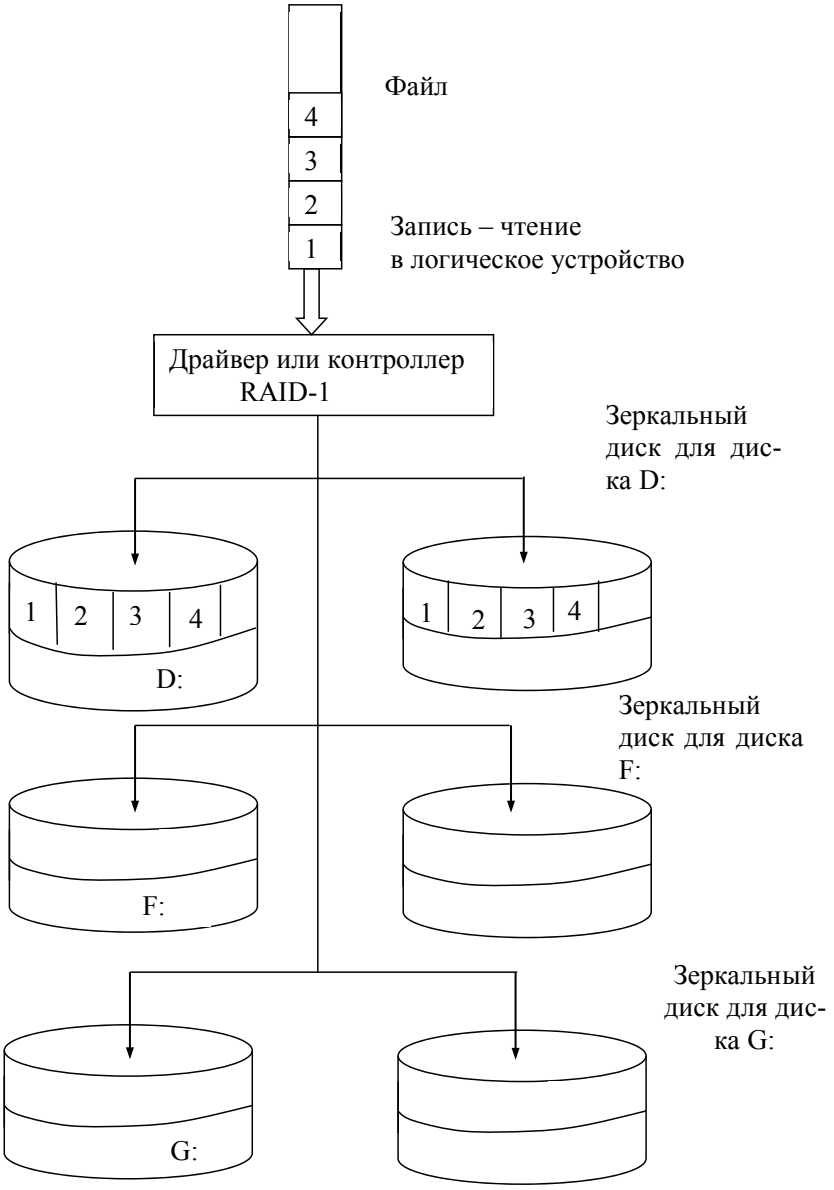


Рис. 8.3. Организация массива RAID-1

Логическое устройство образуется на основе одной или нескольких пар дисков, в которых один диск является основным, а второй диск (называемый зеркальным) дублирует информацию, находящуюся на основном диске. Если основной диск выходит из строя, на зеркальном остаются данные, тем самым поддерживается повышенная отказоустойчивость всего логического устройства. Это обеспечивается избыточностью, данные хранятся в двух экземплярах, в результате дисковое пространство эффективно используется лишь на 50 %.

При внесении изменений в данные, расположенные на логическом устройстве RAID-1, контроллер (или драйвер) массива дисков одинаково модифицирует и основной, и зеркальный диски, при этом дублирование операций происходит прозрачно для пользователя и приложений. Удвоение количества операций записи снижает, хотя и не очень значительно, производительность дисковой подсистемы, поэтому в случае аппаратного RAID вместе с дублированием дисков дублируются и их контроллеры. Это дублирование, называемое еще дуплексированием (*duplexing*), вместе с повышением скорости операций записи усиливает надежность системы — данные на зеркальном диске останутся доступными не только в случае выхода из строя одного из дисков, но и одного из контроллеров.

Некоторые современные дисковые контроллеры (например, SCSI-контроллеры) могут ускорять выполнение операций чтения с дисков, входящих в зеркальный набор. При высокой интенсивности ввода-вывода, контроллер распределяет нагрузку между двумя дисками так, что две операции чтения могут выполняться одновременно. В результате распараллеливания работы, время выполнения операции чтения может быть снижено в два раза. Таким образом, некоторое снижение производительности, возникающее при выполнении операций записи, компенсируется повышением скорости выполнения операций чтения.

Достоинства уровня:

1) имеет высокую надежность — работает до тех пор, пока функционирует хотя бы один диск в массиве. Вероятность выхода из строя сразу двух дисков равна произведению вероятностей отказа каждого диска, т. е. значительно ниже вероятности выхода из строя отдельного диска;
2) обеспечивает приемлемую скорость записи (такую же, как и без дублирования) и выигрыш по скорости чтения при распараллеливании запросов.

Недостаток его — в этой системе максимальная избыточность, что приводит к тому, что пользователь получает в распоряжение объем одного диска по цене двух.

Уровни RAID-2, RAID-3, RAID-4

На уровне RAID-2 данные расщепляются побитно, при этом первый бит записывается на первый диск, второй бит — на второй диск и т. д. Отказоустойчивость реализуется в RAID-2 путем применения для кодирования данных корректирующего кода Хэмминга, который обеспечивает исправление однократных ошибок и обнаружение двукратных ошибок. Избыточность обеспечивается за счет нескольких дополнительных дисков, используемых для записи кода коррекции ошибок.

Достоинством массива RAID-2 является повышение скорости дисковых операций по сравнению с производительностью одного диска.

Недостатком массива RAID-2 является то, что минимальное количество дисков, при котором имеет смысл его использовать, — 7; только начиная с этого количества для него требуется меньше дисков, чем для RAID-1 (4 диска с данными, 3 диска с кодами коррекции ошибок), в дальнейшем избыточность уменьшается по экспоненте.

RAID-2 обеспечивает высокую производительность и надежность, но из-за высокой стоимости его реализации он применяется в основном в мейнфреймах и суперкомпьютерах.

В массивах RAID-3 используется расщепление (*stripping*) данных на массиве дисков с выделением отдельного диска для хранения блока четности.

В массиве RAID-3 из *N* дисков, данные разбиваются на части размером меньше сектора (разбиваются на байты или блоки) и распределяются по *N*-1 дискам. В RAID-2 для хранения контрольной информации применялся *N*-1 диск, благодаря чему осуществлялась коррекция ошибок на лету. Большинство пользователей использует просто восстановление информации в случае ее повреждения, для чего достаточно данных, которые помещаются на один выделенный жесткий диск.

Основное отличие технологии RAID-3 от RAID-2 — невозможность коррекции ошибок на лету.

Минимальное количество дисков, создающих конфигурацию RAID-3, равно трем, при этом избыточность максимальна и равна

33 %. При увеличении числа дисков в конфигурации, степень избыточности снижается.

Уровень RAID-3 позволяет выполнять одновременное чтение или запись данных на несколько дисков для файлов с длинными записями.

Достоинства уровня следующие:

1) высокая скорость чтения и записи данных;
2) минимальное количество дисков для создания массива равно трем.

Недостатки его:

1) массив этого типа хорош только для однозадачной работы с большими файлами, т. к. время доступа к отдельному сектору, разбитому по дискам, равно максимальному из интервалов доступа к секторам каждого из дисков. Для блоков малого размера время доступа намного больше времени чтения;
2) большая нагрузка на контрольный диск, и, как следствие, его надежность сильно падает по сравнению с дисками, хранящими данные.

Организация RAID-4 подобна RAID-3 за исключением того, что данные распределяются блоками, при этом обмен может происходить независимо с каждым диском. Как и в RAID-3, для хранения контрольной информации используется один дополнительный диск. Эта конфигурация удобна для файлов с короткими записями и большой частотой операций чтения по сравнению с операциями записи, поскольку в таком случае возможно одновременное выполнение нескольких операций чтения.

Среди широко распространенных систем хранения RAID-4 применяется на устройствах компании NetApp (*NetApp FAS*), где его недостатки успешно устранены за счет работы дисков в специальном режиме групповой записи, определяемом используемой на устройствах внутренней ФС WAFL.

Уровень RAID-5

Основным недостатком уровней RAID от 2-го до 4-го является невозможность производить параллельные операции записи, т. к. для хранения информации о четности используются отдельные контрольные диски. RAID-5 не имеет этого недостатка (рис. 8.4). Блоки данных и контрольные суммы циклически записываются на все диски мас-

сива. Под контрольными суммами подразумевается результат операции «xor» (исключающее ИЛИ). Особенность операции «xor» состоит в том, что она дает возможность заменить любой операнд результатом и получить при этом недостающий операнд.

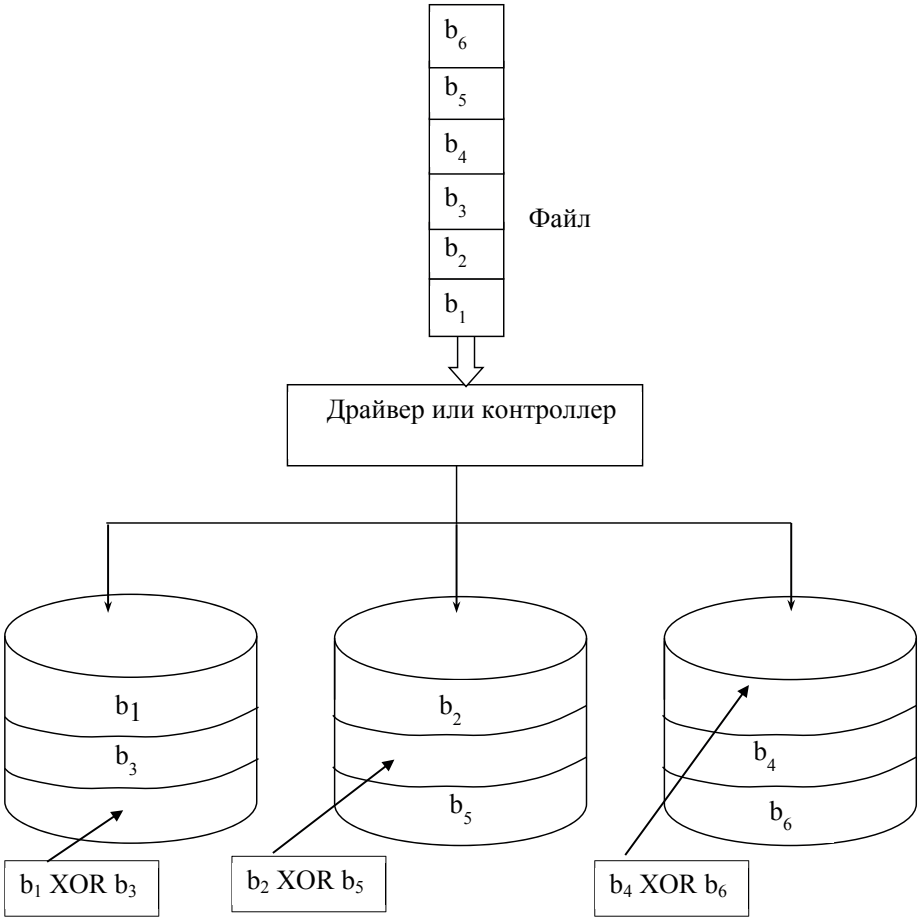


Рис. 8.4. Организация массива RAID-5

Покажем на примере. Пусть *a*, *b*, *c* — три диска RAID-массива и верно: *a xor b* = *c*. В таком случае если *a* откажет, то его можно восстановить, применив *c xor b*. Это является истинным вне зависимости от количества операндов. Именно данное свойство обеспечивает отказоустойчивость в случае с RAID-5. Для хранения результата «*xor*» требуется всего 1 диск, размер которого равен размеру любого другого диска в RAID.

В этой конфигурации минимальное количество используемых дисков равно трем.

Достоинства RAID-5:

1) получил широкое распространение благодаря своей экономичности. Самая большая избыточность в случае применения 3 дисков — 33 %. Для 4 дисков она 25 %, для 5 дисков — 20 % и т. д.;
2) обеспечивает высокую скорость чтения — выигрыш достигается за счет независимых потоков данных с нескольких дисков массива, которые могут обрабатываться параллельно.

Недостатки уровня:

1) производительность RAID-5 заметно ниже при записи в произвольном порядке;
2) при выходе из строя одного диска, надежность тома сразу снижается до уровня RAID-0 с соответствующим количеством дисков (n-1) — т. е. в n-1 раз ниже надежности одного диска — данное состояние называется критическим (*degrade*, или *critical*). Для возвращения массива к нормальной работе требуется длительный процесс восстановления, связанный с ощутимой потерей производительности и повышенным риском.

В табл. 8.2 приведены характеристики для основных конфигураций избыточных дисковых массивов.

Таблица 8.2

Сравнение характеристик уровней RAID

Уровень RAID	Избыточность	Отказоустойчивость	Скорость чтения	Скорость записи
RAID-0	Нет	Нет	Повышенная	Повышенная
RAID-1	50 %	Есть	Повышенная	Пониженная (в варианте без дуплексирования)
RAID-3, RAID-4, RAID-5	До 33 %	Есть	Повышенная	Пониженная (в разной степени)

Другие уровни RAID

RAID-6 похож на RAID-5, но имеет более высокую степень надежности — 3 диска данных и 2 диска контроля четности. Основан на кодах Рида — Соломона и обеспечивает работоспособность после одновременного выхода из строя любых двух дисков. Обычно использование

RAID-6 вызывает падение производительности дисковой группы примерно от 10 до 15 % по сравнению с RAID-5, что вызвано бóльшим объемом работы для контроллера (более сложный алгоритм расчета контрольных сумм), а также необходимостью читать и перезаписывать больше дисковых блоков при записи каждого блока.

Помимо базовых уровней RAID-0–RAID-6, описанных в спецификации SNIA [38], существуют комбинированные уровни с названиями вида «RAID α+β» или «RAID-αβ», что обычно означает «RAID-β, составленный из нескольких RAID-α» (иногда производители интерпретируют это по-своему). Например: RAID-10 (или RAID 1+0) — это RAID-0, составленный из нескольких (или хотя бы двух) RAID-1 (зеркалированных пар); RAID-51 — это RAID-1, зеркалирующий два RAID-5.

Комбинированные уровни наследуют преимущества и недостатки своих «родителей»: появление чередования в уровне RAID 5+0 не добавляет этой конфигурации надежности, зато повышает производительность. Уровень RAID 1+5 очень надежный, но непроизводительный и неэкономичный: полезная емкость тома меньше половины суммарной емкости дисков.

Контрольные вопросы

1. Какие вы знаете восстанавливаемые ФС? Что обеспечивает свойство восстанавливаемости ФС?
2. Что подразумевает свойство идемпотентности?
3. Какие записи модификации будут сделаны в журнале при удалении файла в ФС NTFS?
4. Какие типы RAID существуют?
5. Для чего используются RAID-массивы?
6. Какая технология RAID обеспечивает максимальную безопасность данных? Какую технологию RAID следует использовать для более быстрого доступа к данным?
7. Что предполагает технология RAID-01?

9. Кэширование* данных

Еще одной технологией, широко используемой в работе дисковой подсистемы, является технология кэширования. Следует также заметить, что само понятие кэширования значительно шире.

Кэширование данных

Кэширование — это способ совместного функционирования устройств памяти двух типов, которые отличаются временем доступа и стоимостью хранения единицы данных, суть которого заключается в том, что за счет динамического копирования в быструю память наиболее часто используемых данных из медленной памяти уменьшается среднее время доступа к данным, а также экономится пространство в более дорогой быстродействующей памяти.

Основным свойством кэш-памяти является ее прозрачность для программ и пользователей — перемещение данных из памяти одного типа в память другого типа осуществляется без участия этих пользователей или программ системными средствами.

Кэш-памятью, или кэшем, часто называют одно из устройств, участвующих в процессе кэширования, а именно — быструю память. Это устройство обычно более дорогое и менее объемное.

Кэширование — это универсальный метод, предназначенный для ускорения доступа к различным типам памяти. Если кэширование применяется для уменьшения среднего времени доступа к ОП, то в качестве кэша используют регистровую память. Если кэширование используется для ускорения доступа к данным на диске, то в этом случае роль кэш-памяти выполняют буферы в ОП, в которые помещаются

---

* Нормативное написание «*кеширование*», в профессиональной среде пишут «кэширование», «кэш». (*Прим. ред.*)

наиболее активно используемые данные. Если вспомнить иерархию устройств памяти (см. рис. 5.1), то в качестве кэша для одного из слоев выступает всегда выше прилегающий к нему слой.

Виртуальная память также может считаться реализацией принципа кэширования, при этом ОП выступает в роли кэша по отношению к дисковой памяти. В таком случае кэширование используется не для уменьшения времени доступа к данным, а для того, чтобы подменить дисковой основную память, тем самым увеличить ее емкость. В результате этого наиболее активно используемые данные постоянно находятся в ОП, остальная информация перемещается на диск, где и хранится в более объемной и менее дорогостоящей памяти.

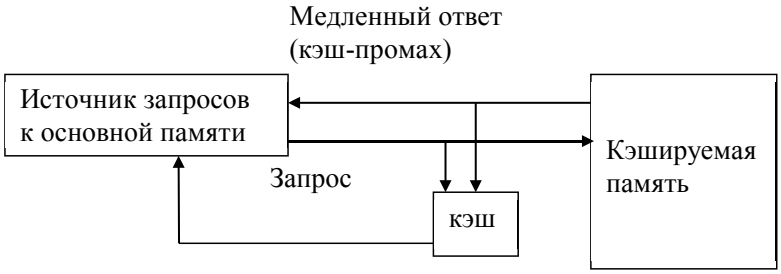
Схема кэширования

Рассмотрим одну из возможных схем кэширования (рис. 9.1). Содержимое кэш-памяти представляет собой совокупность записей обо всех загруженных в нее элементов данных из кэшируемой памяти. Каждая запись включает в себя:

1) значение элемента данных;
2) адрес, который этот элемент данных имеет в кэшируемой памяти;
3) дополнительную информацию, которая используется для реализации алгоритма замещения данных в кэше и обычно включает признак модификации и признак действительности данных.

При каждом обращении к кэшируемой памяти, по физическому адресу просматривается содержимое кэш-памяти в целях определения, не находятся ли там нужные данные. Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — по взятому из запроса значению поля адреса в кэшируемой памяти. Далее возможен один из двух вариантов развития событий:

1) если данные обнаруживаются в кэш-памяти, т. е. произошло кэш-попадание (*cache-hit*), они считываются из нее и результат передается источнику запроса;
2) если нужные данные отсутствуют в кэш-памяти, т. е. произошел кэш-промах (*cache-miss*), они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память.



Структура кэш-памяти

Адрес данных в памяти	Данные	Управляющая информация

Рис. 9.1. Схема реализации кэширования

Интуитивно понятно, что эффективность кэширования зависит от вероятности попадания в кэш. Покажем это путем нахождения зависимости среднего времени доступа к основной памяти от вероятности кэш-попаданий.

Пусть имеется:

1) основное запоминающее устройство со средним временем доступа к данным t_1;
2) кэш-память, имеющая время доступа t_2 (очевидно, что $t_2 < t_1$).

Пусть t — среднее время доступа к данным в системе с кэш-памятью, а p — вероятность кэш-попадания. По формуле полной вероятности имеем:

$$t = t_1 (p — 1) + t_2 p + t_1. \qquad (9.1)$$

Среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности попадания в кэш и изменяется от среднего времени доступа в основное запоминающее устройство (кэшируемую память) (t_1 при $p = 0$) до среднего времени доступа непосредственно в кэш-память t_2 при $p = 1$. Очевидно, что использование кэш-памяти дает выигрыш только при высокой вероятности кэш-попадания.

Вероятность обнаружения данных в кэше зависит от многих факторов, основные из которых объем кэшируемой памяти, объем кэша, алгоритм замещения данных в кэше, особенности выполняемой программы, время работы программы, уровень мультипрограммирования и др. Нужно отметить, что в большинстве реализаций кэш-памяти процент кэш-попаданий оказывается достаточно высоким — свыше 90 %. Такое высокое значение кэш-попаданий обусловлено двумя свойствами данных: пространственной и временной локальности.

Временная локальность состоит в том, что если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время. Пространственная локальность — если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Благодаря свойству временной локальности можно спрогнозировать, что данные, недавно считанные из кэшируемой памяти, следует разместить в запоминающем устройстве быстрого доступа, т. к. они опять скоро понадобятся. В начале работы системы, когда кэш еще пуст, почти каждый запрос к кэшируемой памяти выполняется по длинному пути: просмотр кэша, обнаружение промаха, чтение данных из кэшируемой памяти, передача результата источнику запроса и запись данных в кэш. В течение работы, кэш заполняется и, в соответствии со свойством временной локальности, возрастает вероятность обращения к данным, которые уже были использованы ранее, а значит, содержатся в кэше и могут быть считаны значительно быстрее, чем из кэшируемой памяти.

Свойство пространственной локальности также используется для увеличения вероятности кэш-попадания: в соответствии с ним в кэш-память считывается не один элемент данных, к которому уже произошло обращение, а целый блок таких элементов; в него включаются те элементы данных, которые расположены в непосредственной близости с данным элементом. Поскольку при выполнении программы высока вероятность последовательного выполнения операторов, которые хранятся в соседних ячейках памяти, то имеет смысл загружать в кэш-память целый фрагмент программы. Это же относится и к данным: если программа ведет обработку некоторого массива или списка, то ее работу можно ускорить, загрузив в кэш весь массив или список данных.

Проблема согласования данных при кэшировании

В процессе работы содержимое кэш-памяти обновляется и данные из нее время от времени вытесняются. Вытеснение предполагает либо объявление свободной соответствующей области кэш-памяти, если вытесняемые данные за время нахождения в кэше не были изменены, либо, если данные были модифицированы, в дополнение к описанному осуществление записи данных в кэшируемую память. Алгоритм замещения данных в кэш-памяти существенно влияет на ее эффективность. Такой алгоритм должен быть максимально быстрым, чтобы не замедлять работу кэш-памяти, а обеспечивать максимальную вероятность кэш-попаданий. Поскольку невозможно предсказать ход выполнения вычислительного процесса, то гарантировать оптимальный результат нельзя, обычно применяют рациональные решения, которые незначительно замедляют работу кэша, который по технологии призван быть быстрым.

Наличие в компьютере двух копий данных: одна — в кэшируемой памяти, вторая — в кэше — приводит к необходимости решать проблему согласования данных. Если происходит запись в кэшируемую память по некоторому адресу, а содержимое этой ячейки уже находится в кэше, то такая запись в кэше становится недостоверной. Существуют два подхода к решению этой проблемы:

1) сквозная запись (*write through*) — при каждом запросе к кэшируемой памяти, в т. ч. и при записи, просматривается кэш. Если данные по запрашиваемому адресу отсутствуют, то запись выполняется только в кэшируемую память. Если же данные, к которым выполняется обращение, находятся в кэше, то запись выполняется одновременно в кэш и кэшируемую память;
2) обратная запись (*write back*) — при возникновении запроса к памяти выполняется просмотр кэша, и если запрашиваемых данных там нет, то запись выполняется только в кэшируемую память. В противном случае запись производится только в кэш-память, при этом в служебном поле данных делается специальная отметка (признак модификации). Эта отметка указывает на то, что при вытеснении этих данных из кэша (при сбросе кэша) необходимо записать их и в кэшируемую память, чтобы актуализировать ее состояние.

В некоторых алгоритмах замещения предусматривается первоочередная выгрузка модифицированных данных. Они могут выгружаться не только при освобождении места в кэш-памяти для новых данных, но и в фоновом режиме, т. е. когда система не очень загружена.

Схемы выполнения запросов в системах с кэш-памятью

Приведем в словесной форме обобщенный алгоритм работы кэш-памяти:

1) выполнить запрос к кэшируемой памяти;
2) выполнить просмотр кэша;
3) произошло кэш-попадание. Если да, то перейти к шагу 4, если нет, то — к шагу 8;
4) определить, какая операция. Если «чтение», то перейти к шагу 5, если «запись» — к шагу 7;
5) считать данные из кэша;
6) передать данные источнику запроса. Закончить алгоритм;
7) определить режим согласования. Если «сквозная запись», то произвести запись в кэшируемую память и в кэш, иначе (т. е. режим «обратной записи») — произвести запись в кэш и установить признак модификации. Закончить алгоритм;
8) определить, какая операция. Если «запись», то произвести запись в кэшируемую память и закончить алгоритм; если «чтение» — перейти к шагу 9;
9) произвести чтение из кэшируемой памяти;
10) определить, есть ли свободное пространство в кэше. Если да, то перейти к шагу 13, иначе — к шагу 11;
11) осуществить выбор данных на выгрузку и объявить соответствующий элемент кэша свободным;
12) определить, были ли выгружаемые данные модифицированы. Если да, то копировать выгружаемые данные в кэшируемую память;
13) копировать данные, считанные из кэшируемой памяти в кэш;
14) передать данные источнику запроса. Закончить алгоритм.

Из приведенного алгоритма видно, что, когда выполняется запись, кэш просматривается только для согласования содержимого кэша

и кэшируемой памяти. Если происходит промах, то запросы на запись не вызывают никаких изменений содержимого кэша.

В большинстве реализаций кэш-памяти, при возникновении запроса, сначала просматривается кэш, а затем, если произошел промах, выполняется обращение к кэшируемой памяти. Однако существует и другая схема работы кэша: поиск в кэше и кэшируемой памяти начинается одновременно, а затем, в зависимости от результата просмотра кэша, операция в кэшируемой памяти либо продолжается, либо прерывается.

При выполнении запросов к кэшируемой памяти, во многих вычислительных системах используется двухуровневое кэширование. Кэш первого уровня имеет меньший объем и более высокое быстродействие, чем кэш второго уровня. Кэш второго уровня играет роль кэшируемой памяти по отношению к кэшу первого уровня.

На рис. 9.2 показана схема выполнения запроса на чтение в системе с двухуровневым кэшем. Сначала делается попытка обнаружить данные в кэше первого уровня. Если произошел промах, поиск продолжается в кэше второго уровня. Если же нужные данные отсутствуют и здесь, то происходит считывание данных из кэшируемой памяти. Понятно, что время доступа к данным оказывается минимальным, когда кэш-попадание происходит уже на первом уровне, несколько большим — при обнаружении данных на втором уровне и обычным временем доступа к кэшируемой памяти, если нужных данных нет ни в том, ни в другом кэше. При считывании данных из кэшируемой памяти происходит их запись в кэш второго уровня, а если данные считываются из кэша второго уровня, то они записываются в кэш первого уровня.

При работе такой иерархической организованной памяти необходимо обеспечить непротиворечивость данных на всех уровнях. Кэши разных уровней согласуют данные разными способами. Кэш первого уровня может использовать сквозную запись, а кэш второго уровня — обратную запись. (Именно такая комбинация алгоритмов согласования применена в процессоре Pentium при одном из возможных вариантов работы.)

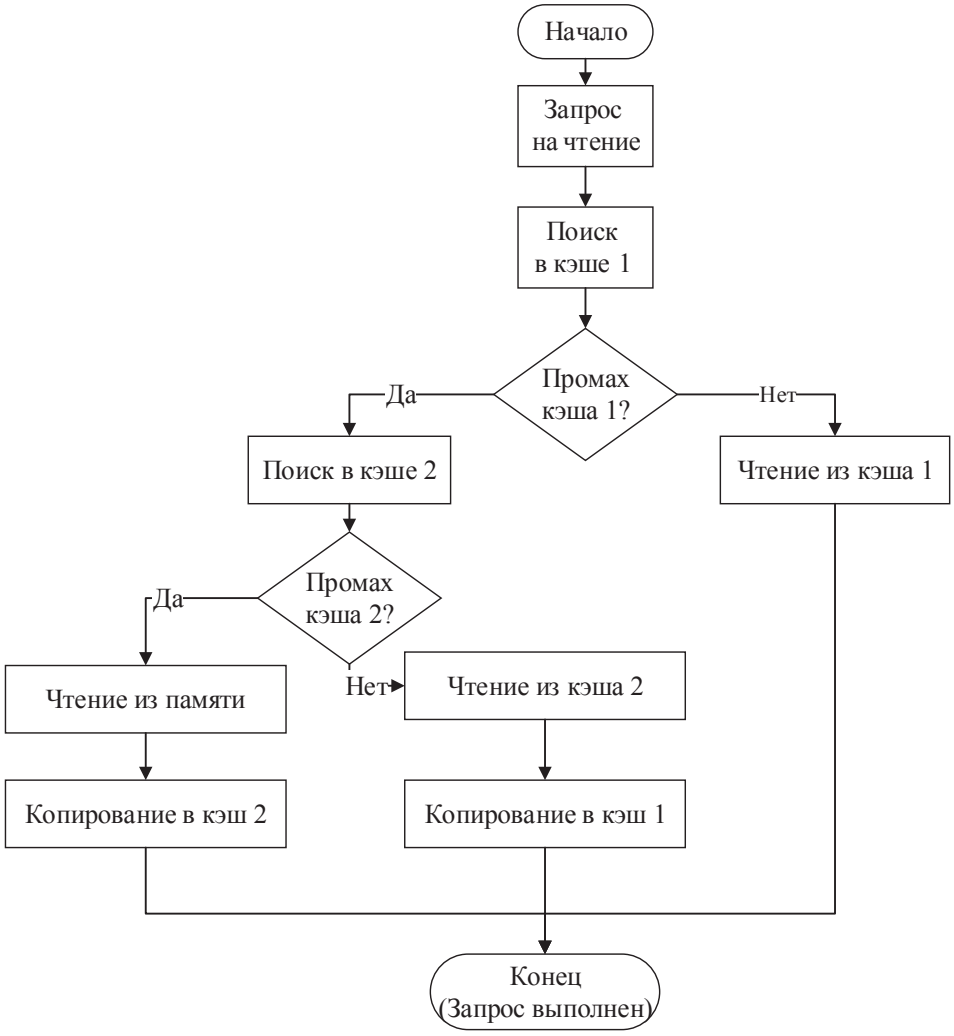


Рис. 9.2. Блок-схема алгоритма выполнения запроса на чтение в системе с двухуровневым кэшем

Контрольные вопросы

1. Нарисовать блок-схему алгоритма функционирования кэш-памяти.
2. Чем объясняется эффективность существующей технологии кэширования?

10. Архитектура операционной системы: основные концепции

Архитектура операционной системы

Любая сложная система должна иметь понятную и рациональную структуру, для программной системы это значит, что она должна иметь модульную структуру, т. е. быть составлена из модулей, которые имеют отдельное функциональное назначение и между которыми четко определены интерфейсы. Ясное представление о назначении каждого модуля упростит в будущем работу по модификации и развитию системы. Все разработчики знают, что сложную программу без хорошей структуры чаще проще разработать заново, чем понять и модернизировать.

Функциональная сложность ОС неизбежно приводит к сложности ее архитектуры и многообразию форматов файлов ее составляющих. Обычно в состав ОС входят:

1) исполняемые и объектные модули стандартных для данной ОС форматов;
2) библиотеки разных типов;
3) модули исходного текста программ;
4) программные модули специального формата (например, загрузчик ОС, драйверы ввода-вывода);
5) конфигурационные файлы;
6) файлы документации;
7) модули справочной системы
8) и т. д.

Большинство современных ОС — хорошо структурированные модульные системы, легко модифицируемые за счет документированных интерфейсов, способные к переносу на новые аппаратные платформы. Единой архитектуры ОС не существует, но существуют универсальные подходы к разработке архитектуры и кодированию ОС.

Ядро и вспомогательные модули ОС

Наиболее общим подходом к структуризации ОС является разделение всех ее модулей на две группы:

1) ядро — модули, выполняющие основные функции ОС;
2) модули, выполняющие вспомогательные функции ОС.

В модулях ядра реализованы базовые функции, которые должна выполнять ОС. Ядро — это сердце ОС, без него она полностью неработоспособна, не сможет выполнят свои функции.

Ядро выполняет следующий набор функций:

1) решает внутрисистемные задачи организации вычислительного процесса;
2) осуществляет поддержку работы приложений.

Вводится понятие интерфейса прикладного программирования — API (*Application Program Interface*) — это те функции ОС, которые вызываются приложениями.

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями ОС, поэтому от скорости их выполнения зависит производительность всей системы. Для увеличения производительности, все модули ядра, по крайней мере большая их часть, должны постоянно находиться в ОП, т. е. эти модули являются резидентными (не должны откачиваться на диск работающей системой виртуальной памяти).

Ядро является движущей силой всех процессов в компьютере, крах ядра приводит к краху всей системы в целом, поэтому разработчиками ОС особое внимание уделяется созданию кодов ядра.

Отличительные свойства ядра

Ядро обычно имеет специальный формат, который его отличает от пользовательских приложений.

Остальные модули, входящие в состав ОС, выполняют полезные, но не такие жизненно важные функции. К таким вспомогательным модулям могут быть отнесены программы проверки файловых систем, создания архивов данных, дефрагментации диска и т. п. Вспомогательные модули ОС либо являются приложениями, либо могут быть под-

ключены в виде библиотек. Общий вид системы, имеющей архитектуру на основе ядра, представлен на рис. 10.1.

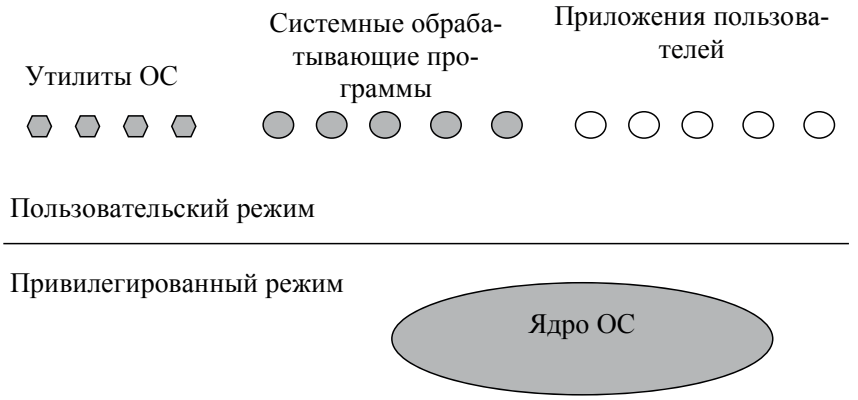


Рис. 10.1. ОС на основе ядра

Поскольку некоторые компоненты ОС работают как обычные приложения и оформлены в виде исполняемых модулей, то часто бывает сложно определить, какие модули относятся к ОС, а какие можно считать обычными приложениями. Решение о том, включать ли некоторое приложение в состав ОС или нет, принимает сам производитель ОС. Среди многих факторов, способных повлиять на это решение, немаловажными являются перспективы того, будет ли программа иметь массовый спрос у потенциальных пользователей данной ОС.

Некоторая программа может существовать довольно долго как обычное приложение, а со временем стать частью ОС или, наоборот, быть выведенной из нее. В качестве примера подобного явления можно привести Internet Explorer, Web-браузер компании Microsoft, который долгое время поставлялся как отдельное приложение, а затем вошел в состав ОС Windows.

Однако между утилитами и системными обрабатывающими программами нет четкого разделения. Обычно под утилитами понимают программы, решающие отдельные системные задачи по управлению и сопровождению компьютерной системы: к таким программам можно отнести программы по работе с дисками (архиваторы, дефрагментаторы, программы по разбиению диска и т. п.). В качестве системных обрабатывающих программ обычно рассматривают текстовые или графические редакторы, трансляторы, отладчики и т. п.

Приложения, утилиты, обрабатывающие программы и библиотеки ОС для выполнения своих задач обращаются к функциям ядра с помощью системных вызовов.

Разбиение ОС на ядро и модули приложений обеспечивает достаточно простую расширяемость ОС. Чтобы ввести новую системную функцию высокого уровня, нужно просто разработать новое приложение и установить его в системе, при этом не потребуется модификации базовых функций, сосредоточенных в ядре системы. Изменение функций ядра оказывается значительно сложнее, причем уровень сложности определяется структурной организацией этого ядра.

Утилиты ОС, системные обрабатывающие программы и библиотеки обычно находятся в ОП только во время выполнения своих функций, они являются транзитными. Резидентно в ОП находятся только необходимые коды ОС, которые и заключены в ее ядре. Такой способ функционирования, поддерживаемый архитектурой ОС, экономит ресурс ОП компьютера.

Важным свойством данной архитектуры ОС является возможность защиты кодов и данных ОС за счет того, что функции ядра выполняются в привилегированном режиме. Для управления выполнением приложений, ОС должна иметь определенные привилегии по отношению к этим приложениям. Это делается по следующим причинам:

1) некорректно работающее приложение может вмешаться в работу ОС и разрушить часть ее кодов. Все усилия разработчиков ОС окажутся напрасными, если их решения воплощены в незащищенные от приложений модули системы, какими бы элегантными и эффективными эти решения ни были;
2) ОС должна обладать исключительными полномочиями для того, чтобы играть роль арбитра в споре приложений за ресурсы компьютера в мультипрограммном режиме.

Обеспечить привилегии ОС не представляется возможным без использования специальных средств аппаратной поддержки**.** Аппаратура компьютера должна поддерживать как минимум два режима работы:

1) пользовательский режим (*user mode*);
2) привилегированный режим, который также называют режимом ядра (*kernel mode*).

Предполагается, что ОС или некоторые ее части работают в привилегированном режиме, а обычные пользовательские приложения — в пользовательском. Как было сказано, именно ядро выполняет основ-

ные функции ОС, поэтому ядро и становится той частью ОС, которая выполняется в привилегированном режиме (рис. 10.1). Иногда работа в привилегированном режиме становится тем основным свойством, по которому определяют модули ядра.

Приложения ставятся в подчиненное положение за счет запрета:

1) выполнения в пользовательском режиме некоторых критичных команд, связанных с переключением процессора с задачи на задачу, управлением устройствами ввода-вывода, доступом к механизмам распределения и защиты памяти. Выполнение некоторых инструкций в пользовательском режиме запрещается безусловно (очевидно, что к таким инструкциям относится инструкция перехода в привилегированный режим). Часть инструкций запрещается выполнять только при определенных условиях;
2) доступа к определенным областям памяти (точнее, приложению разрешено обращаться по адресам, принадлежащим к его адресному пространству).

Следует заметить, что ОС использует механизмы защиты памяти не только для защиты системных областей памяти от доступа приложений, но и для защиты адресного пространства приложения от доступа других приложений. Это свойство позволяет изолировать некорректно работающее приложение в собственном адресном пространстве, и его ошибки не окажут влияния на работу остальных приложений и самой ОС.

Между количеством уровней привилегий, поддерживаемых процессором, и количеством уровней привилегий, используемых ОС, нет прямого соответствия. Например, на базе 4 уровней, поддерживаемых процессорами Intel, такая ОС, как OS/2, строила 3-уровневую систему привилегий, а ОС семейства Windows, различные версии Linux и некоторые другие ОС ограничиваются 2-уровневой системой.

Если процессор поддерживает хотя бы два уровня привилегий, то ОС способна на этой базе создать программно достаточно развитую систему защиты. Повышение устойчивости ОС за счет перевода ядра на работу в привилегированный режим приводит к некоторому замедлению выполнения системных вызовов. Системный вызов в этом случае осуществляет переключение процессора из пользовательского режима в привилегированный, а при обратном переходе — к выполнению приложения — происходит переключение из привилегированного режима в пользовательский (рис. 10.2).

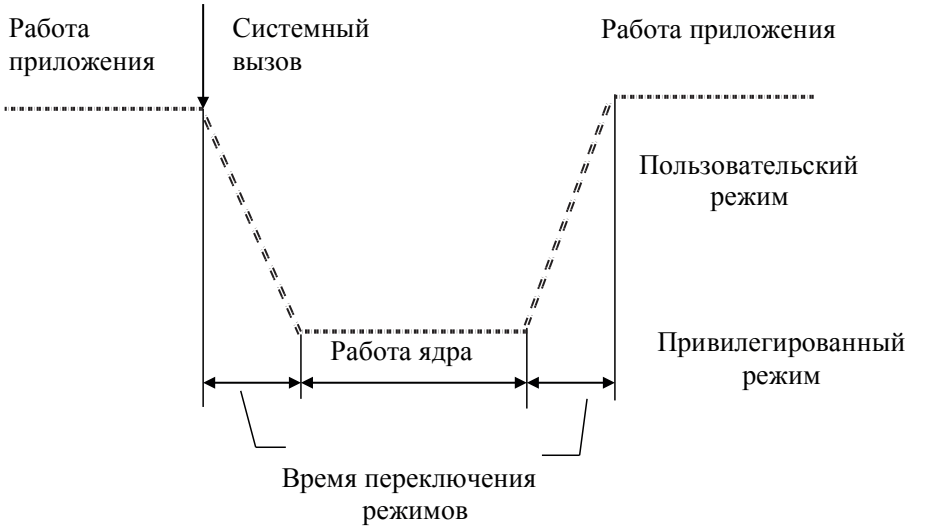


Рис. 10.2. Смена режимов при выполнении системного вызова к ядру

Из-за дополнительной двукратной задержки, вызов процедуры, сопровождающийся сменой режима, выполняется медленнее, чем вызов процедуры без необходимости смены режима.

В одном режиме работают также ядро и приложения тех ОС, которые разработаны для процессоров, вообще не поддерживающих привилегированного режима работы. Наиболее популярным процессором такого типа был процессор Intel 8088/86, послуживший основой для персональных компьютеров компании IBM. ОС MS-DOS, разработанная компанией Microsoft для этих компьютеров, состояла из двух модулей MSDOS. SYS и IO. SYS, составлявших ядро системы (хотя название «ядро» для этих модулей не употреблялось, по своей сути они им являлись), к которым с системными вызовами обращались командный интерпретатор COMMAND. COM, системные утилиты и приложения. Некорректно написанные приложения вполне могли разрушить основные модули MS-DOS, что иногда и происходило, но область использования MS-DOS (и многих подобных ей ранних ОС для персональных компьютеров, таких как MSX, CP/M) и не предъявляла высоких требований к надежности ОС.

В современных ОС различают следующие виды ядер [39].

1. Наноядро (НЯ) — крайне простое и минимальное по размеру ядро, на которое возложена одна задача — обработка аппаратных прерываний, которые генерируют устройства компьютера.

По окончании обработки информация о результатах посылается вышележащему ПО.

2. Микроядро (МЯ) предоставляет элементарные функции по управлению процессами и небольшое множество абстракций, предназначенных для работы с аппаратурой. Основная часть работы проделывается с помощью пользовательских процессов, которые называются сервисами. Будет подробно рассмотрено далее.
3. Экзоядро (ЭЯ) предоставляет множество сервисов, осуществляющих взаимодействия между приложениями, а также минимальный набор функций, обеспечивающих защиту, в т. ч. захват и освобождение ресурсов, контроль предоставляемых прав доступа и т. д. ЭЯ не осуществляет предоставление абстракций для ресурсов — эти функции вынесены в специальную библиотеку пользовательского уровня. В отличие от микроядерных архитектур, данная архитектура обеспечивает большую эффективность за счет того, что нет необходимости при каждом обращении к оборудованию переключаться между процессами.
4. Монолитное ядро (МнЯ) предоставляет большой выбор абстракций оборудования. Все части такого ядра работают в едином адресном пространстве. Такие ядра требуют перекомпиляции, если изменяется состав оборудования. Компоненты ОС представляют собой несамостоятельные модули, а являются частями единой программы. МнЯ более производительно, чем, например, микроядро, т. к. работает как единый процесс. Подобная архитектура характерна для большинства Unix- и Linux-систем. Монолитность ядер усложняет отладку, понимание кода ядра, его расширение, удаление устаревшего ненужного кода. Разрастание кода монолитных ядер повышает требования к объему ОП.
5. Модульное ядро (МодЯ) — современная модифицированная архитектура на основе МнЯ. В отличие от классической архитектуры МнЯ, эти ядра не требуют перекомпиляции ядра, если изменится состав аппаратуры компьютера. Вместо этого они предоставляют механизм загрузки дополнительных модулей, которые поддерживают новое аппаратное обеспечение (например, подгрузки драйверов). Подобная подгрузка может быть динамической или статической (происходит при перезагрузке ОС после изменения состава системы). МодЯ проще в разработке, чем монолитные ядра. Они предоставляют программный интерфейс

(API) для обращения модулей к ядру, для осуществления процедуры динамической загрузки и выгрузки модулей. Не все части ядра могут быть оформлены в виде модулей, некоторые должны всегда находиться в ОП и должны быть постоянной частью этого ядра.

6. Гибридное ядро (ГЯ) — модифицированное микроядро, которое позволяет ускорять работу, запуская некоторые жизненно не очень важные части в пространстве ядра. Таким ядрам присущи свои достоинства и недостатки. Примером подобного гибридного подхода может служить возможность запуска ОС, у которой монолитное ядро, под управлением микроядра. Так устроены BSD версии 4.4 и MkLinux, которые основаны на микроядре Mach. Микроядро управляет виртуальной памятью и обеспечивает работу низкоуровневых драйверов. Все остальные функции, в число которых входит поддержка интерфейса с прикладными программами, выполняются монолитным ядром. Данный подход пытается сочетать в себе преимущества микроядерной архитектуры с возможностью использовать хорошо отлаженный код монолитного ядра.

Наиболее удачно сочетание микроядерной архитектуры и элементов монолитного ядра реализовано в ядре Windows версий NT. Хотя Windows версий NT часто называют микроядерными ОС, это не совсем так. Ядро NT имеет слишком большой размер (более 1 Мбайт), чтобы называться микроядром. Часть компонентов ядра Windows NT вытесняются из ОП и взаимодействуют друг с другом с помощью сообщений, как и происходит в микроядерных ОС. Одновременно все части кода ядра работают в одном адресном пространстве и используют общие структуры данных, что характерно для ОС с монолитным ядром.

Иерархический (многослойный) подход при построении ядра

Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев: нижний слой образован аппаратурой, промежуточный — ядром, а утилиты, обрабатывающие програм-

мы и приложения, составляют верхний слой системы (рис. 10.3). Слоистую структуру вычислительной системы принято изображать в виде системы концентрических окружностей, поскольку каждый слой может взаимодействовать только со смежными слоями. Действительно, при такой организации ОС, приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра.

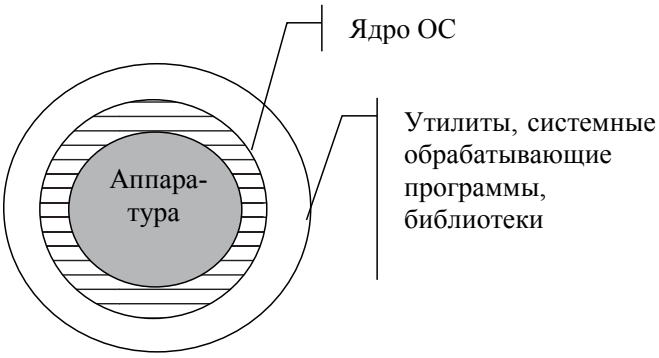


Рис. 10.3. Трехслойная структура вычислительной системы

Многослойный подход является универсальным при создании архитектур сложных систем любого типа, включая программные системы. Этот подход заключается в следующем:

1) система состоит из множества иерархически связанных слоев (рис. 10.4);
2) каждый вышележащий слой обращается за обслуживанием к лежащему под ним слою, вызывая предоставляемые этим слоем функции, которые образуют межслойный интерфейс. На основе функций межслойного интерфейса следующий (вверх по иерархии) слой строит свои функции, которые являются более сложными и более мощными и которые по той же схеме становятся примитивами для создания функций лежащего над ним слоя;
3) строгие правила относятся только к способу взаимодействия между слоями системы, а внутри слоя связи между модулями могут быть произвольными. Каждый модуль может выполнить свою работу одним из 3 способов: самостоятельно, вызвать функцию другого модуля своего слоя, обратиться к нижележащему слою через межслойный интерфейс.

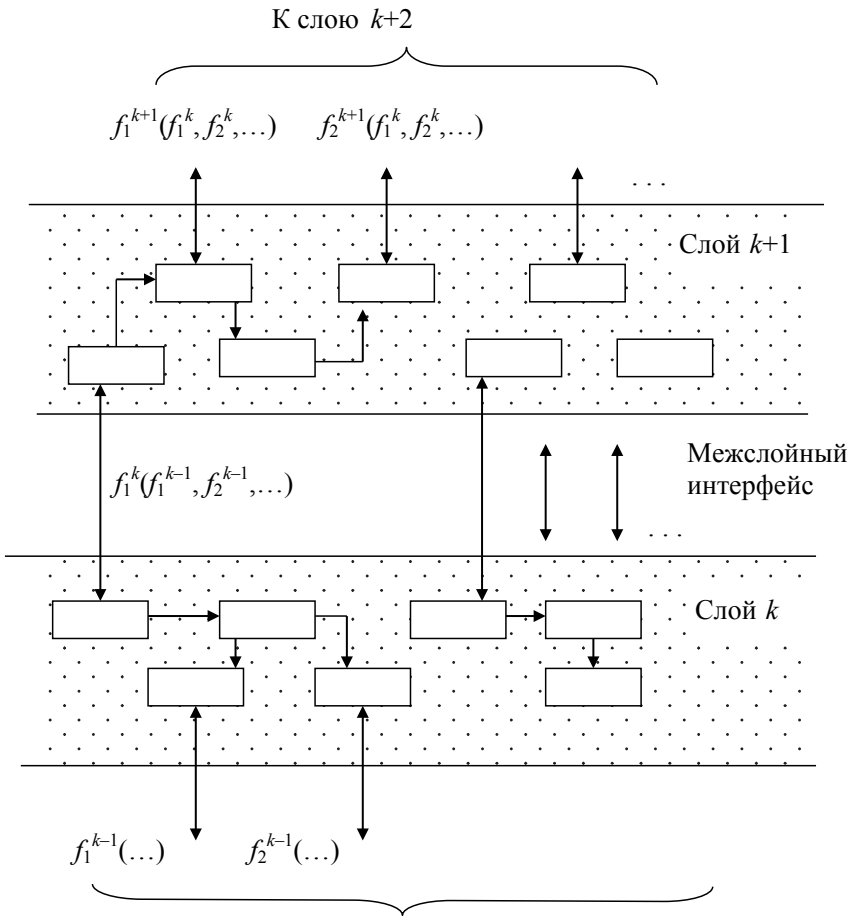


Рис. 10.4. Концепция многослойного взаимодействия

Такая структура системы имеет следующие достоинства:

1) значительно упрощает разработку системы, т. к. позволяет на начальном этапе сверху вниз определить функции, которые закрепляются за слоями и межслойные интерфейсы, а затем, на этапе детальной реализации, наращивать мощность функций слоев, осуществляя движение снизу вверх;
2) при модернизации системы существует возможность изменения модулей внутри одного слоя без необходимости изменять модули в остальных слоях, если при этих внутрислойных изменениях межслойные интерфейсы остаются неизменными.

Ядро может состоять из следующих слоев.

1. Средства аппаратной поддержки ОС. Уже упоминалось, что часть функций ОС выполняется непосредственно аппаратурой. К этому слою относят ту часть аппаратуры, которая непосредственно участвует в организации вычислительного процесса: средства поддержки привилегированного режима, систему прерываний, средства переключения контекстов процессов, средства защиты областей памяти и т. п.
2. Машинно зависимые компоненты ОС. Этот слой образуют программные модули, содержимое которых зависит от специфики аппаратной платформы ПК. Идеально, если слой полностью экранирует все вышележащие слои ядра от особенностей аппаратной платформы. Это дает возможность разрабатывать вышележащие слои, используя машинно независимые компоненты, которые одинаковы для разных типов аппаратных платформ, поддерживаемых данной ОС. Примером такого экранирующего слоя может служить слой HAL (*Hardware Abstraction Layer* — слой абстрагирования от оборудования) в ОС Windows.
3. 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 данного слоя сами не принимают решений, управляющих распределением ресурсов, они реализуют те решения, которые приняты вышележащими слоями, что и оправдывает их название исполнительных механизмов. Например, решение о том, что в данный момент нужно прервать выполнение текущего процесса *A* и начать выполнение процесса *B*, принимается менеджером процессов на вышележащем слое, а слою базовых механизмов передается только директива о том, что нужно выполнить переключение с контекста текущего процесса на контекст процесса *B*.
4. Менеджеры ресурсов. Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на данном слое функционируют такие менеджеры (иногда называются диспетчерами), как менеджер процессов, подсистема ввода-вывода, файловая система и менеджер распределения ОП. Каждый из менеджеров учитывает занятость ресурсов опре-

деленного типа и планирует их распределение в соответствии с поступившими запросами от приложений. Для исполнения принятых решений менеджер обращается к нижележащему слою базовых механизмов с запросами на реализацию принятых им решений. Внутри слоя менеджеров существуют тесные взаимные связи для координации действий.

5. Интерфейс системных вызовов. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Для реализации на практике таких действий, системные вызовы обычно обращаются за помощью к функциям нижележащего слоя менеджеров ресурсов. Нужно учитывать, что для выполнения одного системного вызова может понадобиться несколько обращений к нижележащему слою.

Описанное разбиение ядра на слои является необязательным к реализации. В реальной ОС количество слоев и распределение функций между ними может отличаться от описанного. Способ взаимодействия слоев также может быть несколько иным. Иногда, для повышения производительности ядра, строгие правила взаимодействия только с соседними слоями нарушаются и происходят обращения через слой (или несколько слоев).

Выбор количества и назначения слоев ядра является достаточно сложным: увеличение количества слоев ведет к замедлению работы ядра за счет дополнительных временных задержек на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и структурированность системы.

Обычно давно работающие ОС, например версии UNIX, имеют ядра с небольшим количеством нечетко выделенных слоев, а у ОС недавних разработок, таких как Windows, ядро разделено на достаточно большее количество слоев, причем их взаимодействие формализовано в высокой степени.

Микроядерная архитектура

Одной из альтернатив классической архитектуры ОС на основе монолитного или модульного ядра является микроядерная архитектура. Рассмотрим подробно эту архитектуру.

В привилегированном режиме в этом случае работает только небольшая часть ОС, которая и является микроядром (рис. 10.5). Оно защищено от остальных частей ОС и приложений. В состав самого микроядра обычно включаются машинно зависимые модули и часть модулей, выполняющих базовые функции (например, управление процессами, обработка прерываний, организация виртуальной памяти, обмен сообщениями и с устройствами ввода-вывода и т. п.).

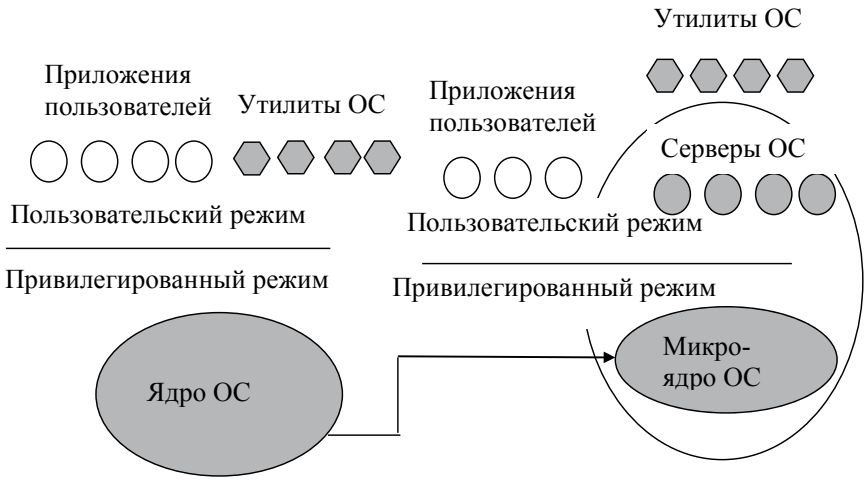


Рис. 10.5. Архитектура ОС:
а — на основе ядра; *б* — на основе микроядра

Все остальные более высокоуровневые функции ядра оформляются в виде приложений (серверов), работающих в пользовательском режиме, поэтому такая технология носит название клиент-серверной. Однозначного решения о том, какие из системных функций нужно оставить в привилегированном режиме, а какие — перенести в пользовательский, не существует. В общем случае многие менеджеры ресурсов, являющиеся неотъемлемыми частями обычного ядра — файловая система, подсистемы управления виртуальной памятью и процессами,

менеджер безопасности и т. п., — становятся серверными модулями, работающими в пользовательском режиме.

Менеджеры ресурсов, которые работают в пользовательском режиме, имеют принципиальные отличия от традиционных утилит и обрабатывающих программ ОС, однако при микроядерной архитектуре они оформляются также в виде приложений.

В ОС с классической архитектурой утилиты и обрабатывающие программы вызываются в основном пользователями и практически никогда другими приложениями, поэтому в ОС с классической архитектурой не предусмотрен механизм, с помощью которого одно приложение могло бы вызывать функции другого.

Совсем другая ситуация возникает, когда в форме приложения оформляется только часть функций ОС. Основной задачей такого приложения будет являться обслуживание запросов, поступающих от других приложений: создание процессов, выделение памяти, проверка прав доступа и т. д. Благодаря своему функционалу — выполнению сервисных функций, менеджеры ресурсов, работающие в пользовательском режиме, называют серверами ОС. Очевидно, что для реализации микроядерной архитектуры необходимым условием является наличие в ОС удобного и эффективного способа вызова процедур одного процесса из другого. Поддержка такого механизма и является одной из главных задач микроядра.

Схематично механизм обращения к функциям ОС, оформленным в виде серверов, выглядит следующим образом (рис. 10.6). 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, т. к. 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.

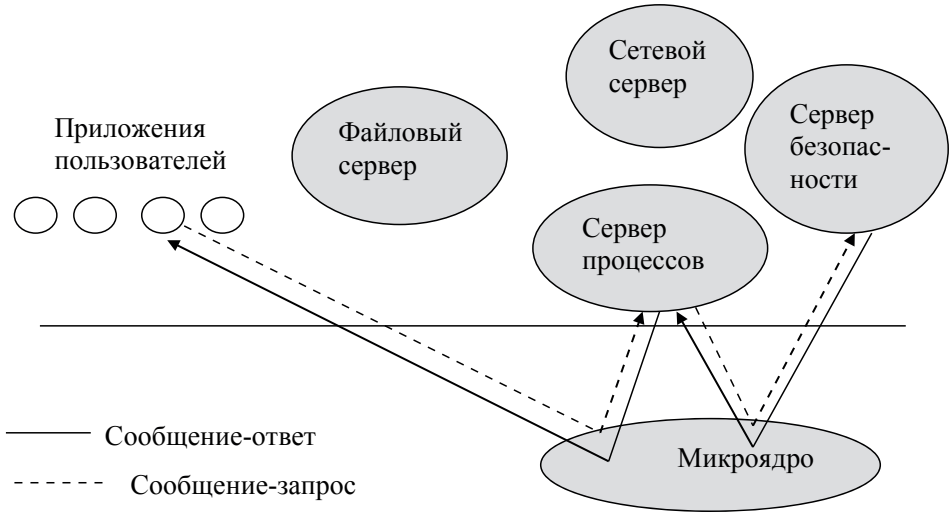


Рис. 10.6. Реализация системного вызова при микроядерной архитектуре

Достоинства и недостатки использования микроядерной архитектуры

ОС, основанные на концепции микроядра, в высокой степени удовлетворяют большинству требований, предъявляемых к современным ОС, обладая переносимостью, расширяемостью, надежностью и создавая хорошие предпосылки для поддержки распределенных приложений. За эти достоинства приходится платить снижением производительности, что и является основным недостатком микроядерной архитектуры.

Рассмотрим, какие из перечисленных ранее требований к современным ОС, обеспечиваются за счет использования при проектировании ОС микроядерной концепции.

Свойство переносимости обеспечивается тем, что весь машинно зависимый код заключен в самом микроядре и для переноса системы на новую платформу требуется изменение именно этого небольшого кода.

Расширяемость также хорошо поддерживается микроядерной архитектурой. В классических системах даже при использовании многослойной (иерархической) структуры достаточно трудно удалить один слой или заменить его на другой, как правило, по причине существования множества таких интерфейсов и отсутствия строгих правил их оформления. Добавление новых функций и модификация существу-

ющих требует знания всех этих интерфейсов и больших временных затрат. В то же время при микроядерной структуре существует ограниченный набор интерфейсов микроядра, что позволяет достаточно легко модифицировать и наращивать функции ОС. Добавление новой подсистемы, поддерживающей новую технологию или новый тип устройств, требует разработки и установки нового приложения, а это никак не затрагивает код самого микроядра.

Микроядерная структура позволяет не только добавлять, но и сокращать число компонентов ОС, что также бывает очень полезно. Например, не всем пользователям нужны средства безопасности или поддержки распределенных вычислений, а удаление их из традиционного ядра чаще всего невозможно. При микроядерном подходе конфигурируемость ОС не вызывает никаких проблем и не требует особых мер — достаточно изменить файл с настройками начальной конфигурации системы или же остановить ненужные больше серверы в ходе работы обычными для остановки приложений средствами.

Использование микроядерной модели повышает надежность ОС. Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти и таким образом защищен от других серверов ОС, 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. Более того, поскольку серверы выполняются в пользовательском режиме, они не имеют непосредственного доступа к аппаратуре и не могут модифицировать память, в которой хранится и работает микроядро. Другим потенциальным источником повышения надежности ОС является уменьшенный объем кода микроядра по сравнению с традиционным ядром — это снижает вероятность появления ошибок программирования (которые могут быть фатальны для функционирования ОС, поскольку код работает в привилегированном режиме).

Микроядерная архитектура способна поддерживать распределенные вычисления за счет того, что использует механизмы, подобные сетевым: клиенты взаимодействуют с серверами посредством обмена сообщениями. Переход к распределенной обработке требует минимальных изменений в работе микроядерной ОС, в таком случае локальный обмен заменяется на сетевой.

Единственным недостатком, который мешает повсеместному распространению данной архитектуры, является снижение производитель-

ности в системах, основанных на этой концепции. При классической организации ОС (рис. 10.7, *а*) выполнение любого системного вызова ядром сопровождается двумя переключениями режимов процессором, а при микроядерной организации (рис. 10.7, *б*) таких переключений четыре. 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.

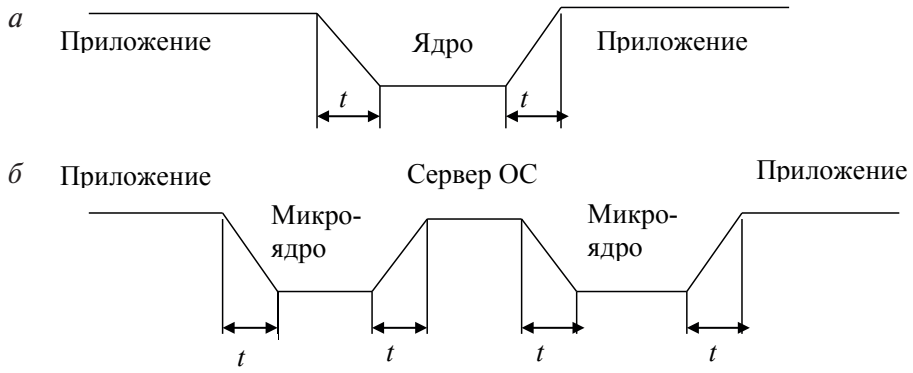


Рис. 10.7. Временные диаграммы системных вызовов:
а — классическая архитектура ОС; *б* — микроядерная архитектура ОС

Критичность данного недостатка отражена в истории развития Windows (версий NT). В ранних версиях диспетчер окон, графическая библиотека и высокоуровневые драйверы графических устройств входили в состав сервера пользовательского режима и вызов функций этих модулей осуществлялся в соответствии с микроядерной концепцией. Однако достаточно быстро разработчики этой ОС поняли, что такой механизм обращений к часто вызываемым функциям графического интерфейса существенно тормозит работу приложений, а это приводит к отставанию данной ОС в условиях острой конкуренции. В результате, начиная с версии Windows NT 4.0, в архитектуру ОС были внесены значительные изменения: все функции графического интерфейса были перенесены в ядро, это привело к отказу от идеальной микроядерной архитектуры, но значительно повысило ее производительность.

Пример Windows NT иллюстрирует главную проблему, которую должны решить разработчики ОС — сторонники микроядерного под-

хода: что включать в микроядро, а что должно работать в виде приложений в пользовательском режиме. В идеале микроядро должно состоять только из средств передачи сообщений и средств взаимодействия с аппаратурой. Однако приходится отказываться от принципа минимизации функций ядра ради повышения производительности ОС в целом.

Поколения микроядер

Микроядра условно делят на поколения. Микроядра разных поколений [41] отличаются устройством и технологическими решениями.

Первое поколение:

1) микроядро Mach от университета Карнеги — Меллон (CMU);
2) микроядро ОС ChorusOS от института INRIA.

Второе поколение:

1) микроядро из ОС Minix от Эндрю Таненбаума (свободный университет Амстердама);
2) L3 от Йохена Лидтке;
3) L4/x86 от Йохена Лидтке.

Третье поколение:

1) seL4 от фирмы NICTA;
2) Coyotos от фирмы The EROS Group, LLC;
3) NOVA.

Архитектура Windows NT

Windows NT можно условно разделить на две части: ту, которая работает в пользовательском режиме (часто говорят о защищенных подсистемах Windows NT), и ту, которая работает в привилегированном режиме (называемую исполнительной системой Windows NT). Структура Windows NT изображена рис. 10.8.

Четыре основных процесса пользовательского режима таковы:

1) фиксированные (или реализованные на аппаратном уровне) вспомогательные системные процессы, такие как процесс входа в систему и администратор сеансов — Session Manager, которые не входят в службы Windows;

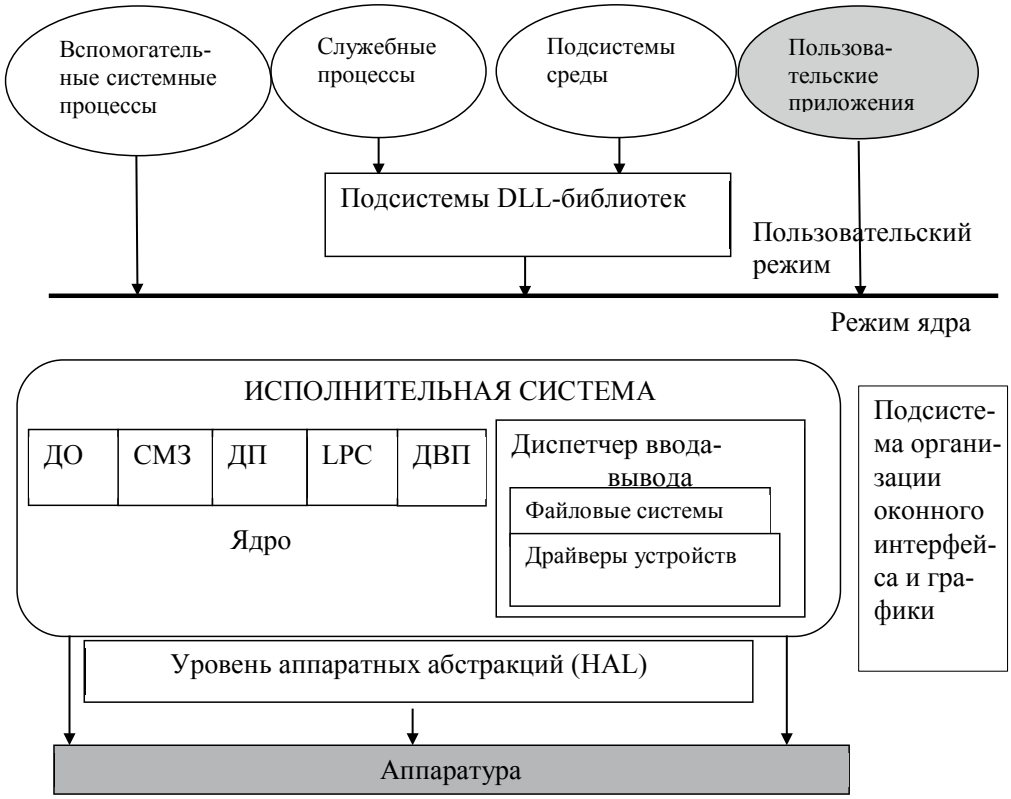


Рис. 10.8. Архитектура ОС Windows NT

2) служебные процессы, реализующие такие службы Windows, как диспетчер задач (*Task Scheduler*) и спулер печати (*Print Spooler*). Как правило, от служб требуется, чтобы они работали независимо от входов пользователей в систему. Многие серверные приложения Windows, например, Microsoft SQL Server и Microsoft Exchange Server, также включают компоненты, работающие как службы;
3) пользовательские приложения, которые могут относиться к одному из следующих типов — для 32- или 64-разрядной версии Windows, для 16-разрядной версии Windows 3.1, для 16-разрядной версии MS-DOS или для 32- или 64-разрядной версии POSIX. Следует учесть, что 16-разрядные приложения работают только в 32-разрядной версии Windows;
4) серверные процессы подсистемы окружения, которые реализуют часть поддержки среды ОС или специализированную часть,

представляемую пользователю и программисту. Изначально в Windows NT были включены три подсистемы среды — Windows, POSIX и OS/2. Подсистемы POSIX и OS/2 последний раз поставлялись в версии Windows 2000. Выпуски клиентской версии Windows Ultimate и Enterprise, а также все серверные версии включают поддержку для усовершенствованной подсистемы POSIX, которая называется подсистемой для приложений на основе Unix (Unix-based Applications, SUA) [40].

При работе под управлением Windows пользовательские приложения не вызывают имеющиеся в ОС Windows службы напрямую, а проходят через одну или несколько подсистем динамически подключаемых библиотек (dynamic-link libraries, DLL).

Исполнительную систему NT называют двигателем данной ОС, эта система способна поддерживать любое число серверных процессов. Серверы обеспечивают исполнительной системе NT пользовательские и программные интерфейсы и предоставляют среды для выполнения различных приложений.

К драйверам устройств относятся как аппаратные драйверы устройств, которые переводят вызовы функций ввода-вывода в запросы ввода-вывода конкретного аппаратного устройства, так и неаппаратные драйверы устройств, такие как драйверы файловой системы и сети.

Уровень аппаратных абстракций (*hardware abstraction layer*, *HAL*) является уровнем кода, который изолирует ядро, драйверы устройств и остальную исполняющую систему Windows от аппаратных различий конкретных платформ (таких как различия между материнскими платами).

Система организации многооконного интерфейса и графики реализует функции графического пользовательского интерфейса (*graphical user interface*, *GUI*), более известные как имеющиеся в Windows USER- и GDI-функции, предназначенные для работы с окнами, элементами управления пользовательского интерфейса и графикой.

Исполнительная подсистема

Исполнительная система NT (*NT executive*) — это та часть Windows NT, которая работает в режиме ядра и которая по сути является законченной ОС, если не принимать во внимание пользовательский графи-

ческий интерфейс. Исполнительная система состоит из компонентов, каждый из них реализует функции двух разных типов: системные сервисы, которые могут быть использованы подсистемами среды и компонентами исполнительной системы, и внутренние сервисы, которые доступны для использования только компонентами самой исполнительной системы.

Компоненты исполнительной системы работают независимо друг от друга, для этого каждый компонент создает свои собственные структуры данных и работает с ними. Интерфейсы между компонентами тщательно проработаны и находятся под полным контролем, поэтому можно удалить определенный компонент и заменить другим, который работает иначе. Если новый компонент корректно разработан и поддерживает все внутренние интерфейсы, то ОС будет работать, как прежде. Сопровождение ОС упрощается, поскольку компоненты исполнительной системы NT взаимодействуют предсказуемым образом.

Ниже перечислены различные компоненты исполнительной системы и их области ответственности:

1) диспетчер объектов (ДО) — создает, поддерживает и уничтожает объекты исполнительной системы NT — абстрактные типы данных, представляющие системные ресурсы;
2) справочный монитор защиты (СМЗ) — реализует политику безопасности на локальном компьютере; оберегает ресурсы ОС, обеспечивая защиту объектов и аудит во время выполнения;
3) диспетчер процессов (ДП) — создает и завершает процессы и потоки. Кроме того, приостанавливает и возобновляет исполнение потоков, хранит и выдает информацию о процессах и потоках NT;
4) средство локального вызова процедур (LPC) — осуществляет передачу сообщений между клиентскими и серверными процессами, которые запущены на данном локальном компьютере;
5) диспетчер виртуальной памяти (ДВП) — реализует виртуальную память — схему управления памятью, которая предоставляет каждому процессу большое собственное адресное пространство и защищает это пространство от других процессов;
6) ядро — реагирует на прерывания и исключения, направляет потоки на выполнение, выполняет межпроцессорную синхронизацию и предоставляет набор элементарных объектов и интерфейсов, используемый остальными частями исполнительной системы NT для реализации объектов более высокого уровня;

7) система ввода-вывода — состоит из группы компонентов, отвечающих за выполнение ввода-вывода на разнообразные устройства. В систему ввода-вывода входят следующие подкомпоненты:
   - диспетчер ввода-вывода — реализует средства ввода-вывода, не зависящие от типа устройства, и устанавливает модель для ввода-вывода исполнительной системы NT;
   - файловые системы — специальные драйверы, которые принимают запросы файлового ввода-вывода и транслируют их в запросы, адресованные конкретному устройству;
   - драйверы устройств исполнительной системы NT — низкоуровневые драйверы, напрямую работающие с оборудованием для записи вывода или считывания ввода с физических устройств или с сети;
   - диспетчер кэша — ускоряет файловый ввод-вывод, сохраняет последнюю информацию, считанную с диска, в системной памяти; диспетчер кэша использует средство подкачки страниц диспетчера виртуальной памяти для автоматической записи информации на диск в фоновом режиме.

Контрольные вопросы

2. Какие отличительные черты ядра ОС?
3. Какие типы ядер ОС обычно выделяют? Какой из них наиболее часто используется в современных версиях ОС?
4. В чем суть и преимущества многослойного подхода при проектировании архитектуры системы?
5. В чем заключаются достоинства и недостатки микроядерных архитектур?
6. Какие части выделяют в архитектуре современных версий ОС Windows?
7. Можно ли утверждать, что современные версии ОС Windows являются микроядерными?

11. Подсистема безопасности

Если рассматривают вопросы безопасности компьютерных систем, то обычно выделяют проблемы двух типов: безопасность локальной системы и сетевая безопасность. К безопасности локальной системы относят все проблемы защиты данных, хранящихся и обрабатывающихся компьютером, который рассматривается как автономная, или локальная, система. Эти проблемы решаются средствами ОС и установленных приложений, а также аппаратными средствами компьютера. В зону ответственности сетевой безопасности включены все вопросы, связанные со взаимодействием устройств в сети — это, например, защита данных при передаче их по линиям связи, а также защита от несанкционированного доступа в сеть. И хотя проблемы локальной и сетевой безопасности трудно отделить друг от друга, совершенно очевидно, что сетевая безопасность имеет свою специфику.

Автономно работающий компьютер можно эффективно защитить от внешних покушений разнообразными способами, например, просто запереть на замок клавиатуру или снять жесткий накопитель и поместить его в сейф. Компьютер, работающий в сети, нельзя изолировать от внешнего мира, он общается с другими компьютерами, находящимися вблизи от него или на значительном расстоянии; обеспечение безопасности в сети является сложной комплексной задачей. Вход стороннего пользователя в ваш компьютер является штатной ситуацией, поскольку вы работаете в сети. Для обеспечения безопасности в данном случае необходимо сделать этот вход контролируемым, что означает, что для каждого пользователя сети должны быть однозначно определены его права на доступ к различным ресурсам и выполнению системных действий на каждом из компьютеров, подключенных к сети.

В сетях существует ряд дополнительных опасностей, связанных с самим процессом передачи, которые представляют собой возможный перехват и анализ сообщений, передаваемых по сети, а также создание ложного трафика.

Вопросы сетевой безопасности приобретают особо важное значение сейчас, потому что большинство корпоративных сетей активно использует публичные каналы. Поставщики услуг публичных сетей редко обеспечивают защиту пользовательских данных при их транспортировке по своим магистралям, возлагая на пользователей заботы по их конфиденциальности, целостности и доступности.

Основные понятия и определения

Безопасная информационная система — это система, которая удовлетворяет трем основным требованиям:

1) защищает данные от несанкционированного доступа;
2) всегда готова предоставить их своим пользователям;
3) надежно хранит информацию и гарантирует неизменность данных.

Другими словами, безопасная система обладает свойствами конфиденциальности, доступности и целостности.

Конфиденциальность (*confidentiality*) — гарантия того, что защищаемые данные будут доступны только тем пользователям, которым системой разрешен данный тип доступа (такие пользователи называются авторизованными в системе);

Доступность (*availability*) — гарантия того, что авторизованные пользователи всегда будут получать доступ к данным.

Целостность (*integrity*) — гарантия сохранности данными правильных значений, которая обеспечивается запретом для неавторизованных пользователей каким-либо образом изменять, модифицировать, разрушать или создавать данные.

Требования безопасности могут меняться в зависимости от назначения системы, типа данных и типа существующих угроз. Невозможно представить информационную систему, для которой были бы неважны свойства целостности и доступности, но свойство конфиденциальности, при определенных условиях, не является обязательным. Например, в случае публикации информации в интернете на веб-сервере (ваша цель — сделать информацию доступной для наибольшего числа пользователей) конфиденциальность не требуется. При этом требования целостности и доступности являются актуальными. Действительно, если вы не предпримете специальных мер по обеспечению

целостности данных, злоумышленник может изменить данные на вашем сервере и нанести этим ущерб вашему предприятию. Не менее важным в данном примере является и обеспечение доступности данных. Затратив немалые средства на создание и поддержание сервера в интернете, предприятие вправе рассчитывать на отдачу — увеличение числа клиентов, количества продаж и т. д. Однако существует вероятность того, что злоумышленник предпримет атаку, в результате которой помещенные на сервер данные станут недоступными для тех, кому они предназначались.

Возможно связать понятия конфиденциальности, доступности и целостности с другими ресурсами вычислительной сети, с такими как внешние устройства или приложения. Эти три свойства безопасной системы называют CIA-триадой [42]. Впервые данный принцип был изложен в статье «Защита информации в компьютерных системах», написанной Зальцером и Шредером в 1974 г. и опубликованной в «Communications of the ACM». Эта триада в современном мире несколько устарела. Стали появляться расширения триады. Например, ФСБ в своей методичке по персональным данным, указав триаду как основные характеристики безопасности, уточняет: «в дополнение к перечисленным выше основным характеристикам безопасности могут рассматриваться также и другие характеристики безопасности. В частности, к таким характеристикам относятся неотказуемость, учетность (иногда в качестве синонима используется термин «подконтрольность»), аутентичность (иногда в качестве синонима используется термин «достоверность») и адекватность» [43].

ОЭСР (Организация экономического сотрудничества и развития) в 1992 г. предложила свои 9 принципов безопасности — Awareness (осведомленность), Responsibility (ответственность), Response (ответ), Ethics (этика), Democracy (демократия), Risk Assessment (оценка рисков), Security Design and Implementation (проектирование и применение безопасности), Security Management и Reassessment (управление безопасностью и переоценка).

В 2002 г. Дон Паркер предложил свой «Паркеровский гексагон», который к триаде добавлял еще 3 характеристики — владение-контроль (possession-control), аутентичность (достоверность) и полезность (utility). Паркер приводил такой пример владения-контроля.

Представьте, что вор украл у вас запечатанный конверт с банковскими картами и PIN-кодами к ним. Даже если вор не открыл этот кон-

верт и не нарушил тем самым его конфиденциальность, это все равно должно вызывать беспокойство владельца конверта, который потерял над ним контроль.

На тему полезности Паркер тоже приводил жизненную ситуацию.

Допустим, вы зашифровали свой жесткий диск и забыли пароль (ключ). Для данных на диске сохраняется конфиденциальность, целостность, доступность, достоверность и контроль, но... вы не можете ими воспользоваться. Это и есть нарушение полезности.

NIST (National Institute of Standards and Technology) в 2004 г. пошел еще дальше и предложил свою модель из 33 элементов или, как написано в SP800–27 «Engineering Principles for Information Technology Security (A Baseline for Achieving Security)», принципов [44].

Приведем еще несколько часто используемых терминов.

Угроза (безопасности информации) — совокупность условий и факторов, создающих потенциальную или реально существующую опасность нарушения безопасности информации [45]. Реализованная угроза называется атакой.

Уязвимость (информационной системы), *брешь* — свойство информационной системы, обусловливающее возможность реализации угроз безопасности обрабатываемой в ней информации [45]. Уязвимости классифицируются по множеству признаков. Один из самых важных признаков — вред, который можно нанести системе, используя уязвимость. Чаще всего под уязвимостью понимают конкретную ошибку, допущенную при проектировании или кодировании системы.

Риск — вероятностная оценка величины возможного ущерба, который может понести владелец информационного ресурса в результате успешно проведенной атаки [45]. Значение риска тем выше, чем более уязвимой является существующая система безопасности и чем выше вероятность реализации атаки.

Классификация угроз

Угрозы могут исходить как от легальных пользователей сети, так и от внешних злоумышленников. В последнее время в статистике нарушений безопасности зафиксирован резкий сдвиг от внешних к внутренним угрозам [46]. Примерно две трети от общего числа всех наи-

более серьезных инцидентов, связанных с безопасностью, составляют нарушения со стороны легальных пользователей сетей: сотрудников и клиентов предприятий, студентов, имеющих доступ к сети учебного заведения, и др. Вместе с тем внутренние атаки обычно наносят меньший ущерб, чем внешние.

Угрозы со стороны легальных пользователей делятся на умышленные и неумышленные.

К умышленным угрозам относятся, например, мониторинг системы в целях получения персональных данных других сотрудников (идентификаторов, паролей) или конфигурационных параметров оборудования. Это может быть также злонамеренное получение доступа к конфиденциальным данным, хранящимся на серверах и рабочих станциях сети предприятия, в целях их похищения, искажения или уничтожения; прямое вредительство — вывод из строя сетевого программного обеспечения и оборудования. Кроме того, к умышленным угрозам относится нарушение персоналом правил, регламентирующих работу пользователей в сети предприятия: посещение запрещенных сайтов, вынос за пределы предприятия съемных носителей, небрежное хранение паролей и другие подобные нарушения режима.

Однако не меньший материальный ущерб предприятию может быть нанесен в результате неумышленных нарушений персонала — ошибок, приводящих к повреждению сетевых устройств, данных, программного обеспечения.

Угрозы внешних злоумышленников являются умышленными и обычно квалифицируются как преступления. Среди внешних нарушителей безопасности встречаются люди, занимающиеся этой деятельностью профессионально или просто из хулиганских побуждений. Целью, которой руководствуются внешние злоумышленники, всегда является нанесение вреда предприятию. Это может быть, например, получение конфиденциальных данных, которые могут быть использованы для снятия денег с банковских счетов, или установление контроля над программно-аппаратными средствами сети для последующего их использования в атаках на сети других предприятий.

Классификация атак

Атаки на подсистему безопасности можно классифицировать по разным критериям. С точки зрения поставленных целей атаки можно разделить:

1) на DoS-атаки (*DoS — Denial of Service* — отказ в обслуживании) — приведение атакуемой системы (или одной из ее подсистем) в неработоспособное состояние за счет отвлечения ее ресурсов на обработку бессмысленных запросов;
2) несанкционированный доступ к данным или другим ресурсам атакуемой системы в целях похищения или разрушения этих данных;
3) захват контроля над системой, обычно в целях атаки на другие системы [3].

Наиболее опасны атаки третьего типа, т. к. система, уязвимая для данного типа атак, представляет проблему не только для ее владельца, но и для окружающих, особенно для тех, кто данному владельцу доверяет.

Основные типы атак на операционную систему

Атака «сканирование файловой системы» является одной из наиболее тривиальных, но в то же время одной из наиболее опасных. Суть атаки заключается в том, что злоумышленник просматривает ФС компьютера и пытается прочесть (или скопировать, или удалить) все файлы подряд. Если он не получает доступ к какому-то файлу или каталогу, то продолжает сканирование. Если объем ФС достаточно велик, рано или поздно обнаруживается хотя бы одна ошибка администратора. В результате этого злоумышленник получает доступ к информации, который должен быть ему запрещен. Данная атака может осуществляться специальной программой, которая выполняет вышеописанные действия в автоматическом режиме.

Несмотря на кажущуюся примитивность описанной атаки, защититься от нее не так просто. Если политика безопасности допускает анонимный или гостевой вход в систему, администраторам остается только надеяться, что права доступа ко всем файлам системы, число

которых может составлять сотни тысяч, определены абсолютно корректно. Если же анонимный и гостевой вход в систему запрещен, поддержание адекватной политики регистрации потенциально опасных событий (аудита) позволяет организовать эффективную защиту от этой угрозы. Впрочем, следует отметить, что поддержание адекватной политики аудита требует от администраторов системы определенного искусства. Кроме того, если данная атака осуществляется злоумышленником от имени другого пользователя, аудит совершенно неэффективен.

Атака «кража ключевой информации» в простейшем случае заключается в том, что злоумышленник подсматривает пароль, набираемый пользователем. То, что все современные ОС не высвечивают на экране вводимый пользователем пароль, несколько затрудняет эту задачу, но не делает ее невыполнимой. Известно, что для того чтобы восстановить набираемый пользователем пароль только по движениям рук на клавиатуре, достаточно всего несколько недель тренировок. Кроме того, достаточно часто встречается ситуация, когда пользователь ошибочно набирает пароль вместо своего имени, которое, в отличие от пароля, на экране высвечивается. Иногда пользователи, чтобы не забыть пароль, записывают его на бумагу, которую приклеивают к нижней части клавиатуры, к задней стенке системного блока или в какое-нибудь другое якобы укромное место. В этом случае пароль рано или поздно становится добычей злоумышленника. Особенно часто такие ситуации имеют место в случаях, когда политика безопасности требует от пользователей использовать длинные, трудные для запоминания пароли. Известны случаи, когда для кражи пароля злоумышленники скрытно фиксировали отпечатки пальцев пользователя на клавиатуре непосредственно после набора пароля. Наконец, если ключевая информация хранится пользователем на внешнем носителе, этот носитель может быть потерян и затем найден злоумышленником, а может быть просто украден.

Существует несколько способов проведения *атаки «подбор пароля»* [46]:

1) полный перебор (метод грубой силы, *bruteforce*) — самая простая (с технической точки зрения) атака на пароль — перебор всех комбинаций допустимых символов (начиная от односимвольных паролей). Современные вычислительные мощности позволяют перебрать все пароли длиной до пяти-шести символов за несколько

секунд. Важной характеристикой пароля, затрудняющей полный перебор, является его длина. Современный пароль должен иметь длину не менее 12 символов;

2) перебор в ограниченном диапазоне — известно, что многие пользователи, составляя пароль, используют символы, находящиеся в определенном диапазоне. Например, пароль, состоящий только из русских букв, или только из латинских букв, или только из цифр. Такой пароль значительно легче запомнить, однако задача противника, осуществляющего перебор, неимоверно упрощается;
3) атака по словарю — в качестве пароля очень часто выбирается какое-то слово. Программа автоматического перебора паролей проверяет слова, содержащиеся в заданном файле со словарем (существует огромное количество доступных словарей такого рода для разных языков). Словарь из двухсот тысяч слов проверяется такой программой за несколько секунд;
4) атака по персональному словарю — если атака по словарю и перебор паролей небольшой длины либо составленных из символов одной группы не помогает, злоумышленник может воспользоваться тем фактом, что для облегчения запоминания, многие пользователи выбирают в качестве пароля личные данные (номер сотового телефона, дату рождения, записанную наоборот, кличку собаки и т. д.);
5) сбор паролей, хранящихся в общедоступных местах — во многих организациях пароли создает и распределяет системный администратор, который использует приведенные выше правила. Пользователи обязаны пользоваться выданным им паролем. Однако, поскольку этот пароль сложно запомнить, он часто хранится под рукой в записанном виде. Нередки случаи, когда пароль записывается на стикер и приклеивается к монитору либо содержатся в записной книжке;
6) социальный инжиниринг — манипулирование людьми для проникновения в защищенные системы пользователя или организации. Если подобрать или украсть пароль не удается, можно попытаться обманом заставить пользователя отдать пароль самому. Классическая тактика социального инжиниринга — телефонный звонок жертве от имени того, кто имеет право знать запрашиваемую информацию. Например, злоумышленник может предста-

виться системным администратором и попросить сообщить пароль (или другие сведения) под убедительным предлогом;

7) фишинг — это процедура «выуживания» паролей случайных пользователей интернета. Обычно заключается в создании подставных сайтов, которые обманом вынуждают пользователя ввести свой пароль.

Атака «сборка мусора» заключается в следующем. Во многих ОС информация, уничтоженная пользователем, не уничтожается физически, а помечается как удаленная. С помощью специальных программных средств эта информация («мусор») может быть в дальнейшем восстановлена. Восстановив данную информацию, злоумышленник просматривает ее и получает интересующие его фрагменты. По окончании просмотра и копирования, вся эта информация вновь «уничтожается». В некоторых ОС, например, из семейства Windows, злоумышленнику даже не приходится использовать специальные программные средства — ему достаточно просто просмотреть «мусорную корзину» компьютера.

Сборка мусора может осуществляться не только на дисках компьютера, но и в ОП. В этом случае специальная программа, запущенная злоумышленником, выделяет себе всю или почти всю доступную ОП, просматривает ее содержимое и копирует фрагменты, содержащие заранее определенные ключевые слова. Если ОС не предусматривает очистку памяти при выделении, злоумышленник может получить таким образом много интересной для него информации, например, содержание области памяти, только что освобожденной текстовым редактором, в котором редактировался конфиденциальный документ.

Проводя *атаку «превышение полномочий»*, злоумышленник, используя ошибки в программном коде ОС и (или) в ее политике безопасности, получает права, превышающие те, которые были ему предоставлены в соответствии с политикой безопасности. Обычно удается запустить программу от имени другого пользователя или подменить динамически подгружаемую библиотеку. Реализация этой атаки представляет наибольшую опасность для тех ОС, в которых допускается временное повышение полномочий пользователя, например, в UNIX или Linux.

Атаки класса «отказ в обслуживании» (DoS-атаки) нацелены на полный или частичный вывод ОС из строя. Существуют следующие атаки данного класса:

1) захват ресурсов — программа захватывает все ресурсы компьютера, которые может получить. Например, программа присваивает себе наивысший приоритет и запускает бесконечный цикл;
2) бомбардировка трудновыполнимыми запросами — программа в бесконечном цикле направляет ОС запросы, выполнение которых требует больших ресурсов компьютера;
3) бомбардировка заведомо бессмысленными запросами — программа в бесконечном цикле генерирует и посылает ОС заведомо бессмысленные запросы, на которые та должна как-то ответить. ОС может с этим потоком запросов не справится, и происходит фатальная ошибка;
4) использование известных ошибок в программном обеспечении или администрировании ОС.

Системный подход к обеспечению безопасности

Для построения и поддержки работоспособности безопасной системы необходим системный подход. В соответствии с этим подходом, прежде всего необходимо осознать весь спектр возможных угроз для конкретной системы и для каждой из этих угроз продумать тактику ее отражения. В этой борьбе можно и нужно использовать самые разноплановые средства и приемы — морально-этические и законодательные, административные и психологические, защитные возможности программных и аппаратных средств сети.

К морально-этическим средствам защиты относят нормы, которые складываются в обществе по мере развития и распространения вычислительных средств. Необходимо внедрять в сознание людей, что действия, направленные на нарушение конфиденциальности, целостности и доступности чужих информационных ресурсов, кражу личной и коммерческой информации, аморальны по своей природе, сходны с разбоем и кражей имущества.

Законодательные средства защиты — это законодательные акты, нормативные акты и стандарты, которые регламентируют правила передачи, использования и обработки информации ограниченного доступа, а также вводят меры ответственности за нарушения этих правил. Правовая регламентация деятельности в области защиты информа-

ции имеет целью защиту информации, составляющей государственную тайну, обеспечение прав потребителей на получение качественных продуктов, защиту конституционных прав граждан на сохранение личной тайны, борьбу с организованной преступностью.

Административные меры — это действия, которые осуществляет руководство предприятия или организации для обеспечения информационной безопасности. К таким мерам относят правила работы сотрудников в форме должностные инструкций, предписывающих строгий порядок работы с конфиденциальной информацией, кроме того, режим работы сотрудников. К административным мерам также относятся правила приобретения предприятием средств безопасности. Представители администрации, которые несут ответственность за защиту информации, должны выяснить, насколько безопасным является использование продуктов, приобретенных у зарубежных поставщиков. Особенно это касается продуктов, связанных с шифрованием. В таких случаях желательно проверить наличие у продукта сертификата, выданного российскими тестирующими организациями.

Психологические меры безопасности могут играть значительную роль в укреплении безопасности системы. Пренебрежение учетом психологических моментов в неформальных процедурах, связанных с безопасностью, может привести к нарушениям защиты. Рассмотрим, например, сеть предприятия, в которой работает много удаленных пользователей. Время от времени пользователи должны менять пароли (обычная практика для предотвращения их подбора). В данной системе выбор паролей осуществляет администратор. В таких условиях злоумышленник может позвонить администратору по телефону и от имени легального пользователя попробовать получить пароль (о социальном инжиниринге уже упоминалось). При большом количестве удаленных пользователей не исключено, что такой простой психологический прием может сработать.

К физическим средствам защиты относятся экранирование помещений для защиты от излучения; проверка поставляемой аппаратуры на соответствие ее спецификациям и отсутствие аппаратных «жучков»; средства наружного наблюдения; устройства, блокирующие физический доступ к отдельным блокам компьютера; различные замки и другое оборудование, которые защищают помещения, где находятся носители информации, от незаконного проникновения, и др.

К техническим средствам информационной безопасности относят программное и аппаратное обеспечение. Такие средства решают широкий круг задач по защите системы, например, контроль доступа к защищаемым ресурсам, включающий процедуры аутентификации и авторизации пользователя (об этом речь пойдет дальше), аудит, шифрование информации, антивирусную защиту, контроль сетевого трафика и множество других задач. Технические средства безопасности могут быть либо встроены в программное (ОС и приложения) и аппаратное (компьютеры и коммуникационное оборудование) обеспечение, либо реализованы в виде отдельных продуктов, созданных специально для решения проблем безопасности.

Политика безопасности. Основные принципы

Важность и сложность проблемы обеспечения безопасности требует разработки политики информационной безопасности, которая подразумевает ответы на следующие вопросы: какую информацию защищать; какой ущерб может понести компания при потере или раскрытии определенных данных; кто или что может представлять собой угрозу безопасности системы, какого типа атаки могут быть предприняты; какие средства использовать для защиты каждого вида информации.

Политика безопасности (информации в организации) — совокупность документированных правил, процедур, практических приемов или руководящих принципов в области безопасности информации, которыми руководствуется организация в своей деятельности [47].

ОС называют защищенной, если она предусматривает средства защиты от основных классов угроз [47]. Защищенная ОС обязательно должна включать в свой состав средства разграничения доступа пользователей к ресурсам, средства проверки подлинности пользователя, который может работать в ней. Кроме того, защищенная ОС должна содержать средства противодействия попыткам вывода ее из строя, преднамеренным или случайным.

Если ОС предусматривает защиту не от всех основных классов угроз, а только частичную, то такую ОС называют частично защищенной.

Специалисты, ответственные за безопасность системы, формируя политику безопасности, должны следовать нескольким основным правилам. Одним из таких правил является предоставление каждому сотруднику предприятия того минимального уровня привилегий на доступ к данным, который необходим ему для выполнения его должностных обязанностей. Учитывая, что большая часть нарушений в области безопасности предприятий исходит именно от собственных сотрудников, важно ввести четкие ограничения для всех пользователей сети, не наделяя их излишними возможностями.

Еще одно правило — использование системного подхода к обеспечению безопасности, о чем уже говорилось. Следует также отметить, что, используя многоуровневую систему защиты, важно обеспечивать баланс надежности защиты всех уровней. Если в сети все сообщения шифруются, но ключи легкодоступны, то эффект от шифрования нулевой, или если на компьютерах установлена ФС, поддерживающая избирательный доступ на уровне отдельных файлов, но имеется возможность получить жесткий диск и установить его на другой машине, то все ваши старания окажутся напрасными.

Следующим универсальным правилом является использование средств, которые при отказе переходят в состояние максимальной защиты. Это касается самых различных средств безопасности. Если, например, автоматический пропускной пункт в какое-либо помещение ломается, то он должен фиксироваться в таком положении, чтобы ни один человек не мог пройти на защищаемую территорию. А если в сети имеется устройство, которое анализирует весь входной трафик и отбрасывает кадры с определенным, заранее заданным обратным адресом, то при отказе оно должно полностью блокировать вход в сеть. Неприемлемым следовало бы признать устройство, которое бы при отказе пропускало в сеть весь внешний трафик.

Правило единого контрольно-пропускного пункта — весь входящий во внутреннюю сеть и выходящий во внешнюю сеть трафик должен проходить через единственный узел сети, например, через межсетевой экран (*firewall*). Только это позволяет в достаточной степени контролировать трафик. В противном случае, когда в сети имеется множество пользовательских станций, имеющих независимый выход во внешнюю сеть, очень трудно скоординировать правила, ограничивающие права пользователей внутренней сети по доступу к серверам внешней сети и обратно — права внешних клиентов по доступу к ресурсам внутренней сети.

Правило баланса предполагаемого ущерба от успешной проведенной атаки и затрат на ее предотвращение состоит в следующем. Ни одна система безопасности не гарантирует защиту данных на уровне 100 %, поскольку является результатом компромисса между возможными рисками и возможными затратами. Определяя политику безопасности, администратор должен взвесить величину ущерба, которую может понести предприятие в результате нарушения защиты данных, и соотнести ее с величиной затрат, требуемых на обеспечение безопасности этих данных. В некоторых случаях можно отказаться от дорогостоящего межсетевого экрана в пользу стандартных средств фильтрации обычного маршрутизатора, в других же можно пойти на беспрецедентные затраты. Главное, чтобы принятое решение было обосновано экономически.

Основные функции подсистемы безопасности ОС

Опишем основные функции, выполняемые подсистемой безопасности:

1) разграничение доступа;
2) идентификацию, аутентификацию и авторизацию;
3) криптографические функции;
4) аудит;
5) управление политикой безопасности;
6) сетевые функции.

Подсистема безопасности никогда не состоит только из одного программного модуля. Функций у подсистемы достаточно много, и каждая из них реализуется одним или несколькими программными модулями. Некоторые функции реализуются непосредственно ядром ОС. Однако правило таково, что должен существовать четко определенный интерфейс между различными модулями подсистемы безопасности, которого придерживаются модули при взаимодействии для решения общих задач.

В некоторых ОС, таких как Windows, подсистема безопасности выделена в общей архитектуре, в других — как в UNIX, защитные функции распределены между различными элементами ОС. Обычно подсистема безопасности ОС допускает свое расширение дополнительными программными модулями.

Идентификация, аутентификация и авторизация

Идентификация (*identification*) субъекта доступа заключается в том, что субъект сообщает системе идентифицирующую информацию о себе в нужном формате (чаще всего логин и пароль) и таким образом определяет себя.

Аутентификация (*authentication*) — проверка подлинности пользователя, т. е. это процедура доказательства пользователем того, что он есть тот, за кого себя выдает, в частности, доказательство того, что именно ему принадлежит введенный им идентификатор. В процедуре аутентификации участвуют две стороны: одна из сторон стремится доказать свою аутентичность, предъявляя некоторые артефакты, другая сторона, называемая аутентификатором, проверяет предоставленные доказательства и принимает решение о подлинности. В качестве доказательства аутентичности используются самые разнообразные приемы:

1) аутентифицируемый может продемонстрировать знание некоего общего для обеих сторон секрета — слова (пароля) или факта (даты и места события, прозвища человека и т. п.);
2) аутентифицируемый может продемонстрировать, что он владеет неким уникальным предметом (физическим ключом), в качестве которого может выступать, например, электронная магнитная карта;
3) аутентифицируемый может доказать свою идентичность, используя собственные биохарактеристики — рисунок радужной оболочки глаза или отпечатки пальцев, которые предварительно были занесены в базу данных аутентификатора.

Легальность пользователя может устанавливаться по отношению к различным системам. Так, работая в сети, пользователь может проходить процедуру аутентификации и как локальный пользователь, который претендует на использование ресурсов только данного компьютера, и как пользователь сети, который хочет получить доступ ко всем сетевым ресурсам. При локальной аутентификации пользователь вводит свои идентификатор и пароль, которые автономно обрабатываются ОС, установленной на данном компьютере. При логическом входе в сеть данные о пользователе (идентификатор и пароль) передаются на сервер, который хранит учетные записи обо всех пользователях сети. Многие приложения имеют свои средства определения, являет-

ся ли пользователь законным, и тогда пользователю приходится проходить дополнительные этапы проверки.

Процесс *авторизации* субъекта доступа может начинаться только после успешной идентификации и аутентификации. При авторизации субъекта, ОС выполняет действия, необходимые для того, чтобы субъект мог начать работу в системе, наделяет его определенными правами и привилегиями в системе.

Криптографические функции

Криптографические функции базируются на шифровании. Шифрование — одно из базовых оснований для всех служб информационной безопасности, например, для службы аутентификации, для процесса разграничения доступа или безопасного хранения данных.

Любая процедура шифрования, которая превращает информацию из обычного читабельного вида в непонятный зашифрованный код, должна быть дополнена процедурой дешифрирования, которая, при условии применения ее к зашифрованному тексту, должна снова вернуть ему понятный вид.

Криптосистемой называется совокупность процедур шифрования и дешифрирования.

Криптостойкостью системы называется сложность алгоритма раскрытия шифрованного текста. Данная характеристика является одной из основных для криптосистемы.

Существует два класса криптосистем — симметричные и асимметричные. В симметричных схемах шифрования (классическая криптография) секретный ключ зашифровки совпадает с секретным ключом расшифровки. В асимметричных схемах шифрования (криптография с открытым ключом) открытый ключ зашифровки не совпадает с секретным ключом расшифровки.

Аудит

Аудит (*audit*) — фиксация в системном журнале событий, связанных с доступом к защищаемым системным ресурсам [47]. Подсистема аудита современных ОС позволяет определять перечень тех событий, которые интересуют администратора, при этом с помощью удобного графического интерфейса. Различные программные и аппаратные средства

обеспечивают возможность обнаружить и зафиксировать определенные события, нарушающие безопасность системы, любые попытки получить доступ к защищаемым ресурсам или удалить системные ресурсы. Аудит используется для того, чтобы обнаруживать даже неудачные атаки на компьютерные системы.

Никакая, даже очень сложная, система безопасности не гарантирует стопроцентную защиту, тогда последней надеждой в борьбе с нарушениями оказывается подсистема аудита. Необходимость включения в защищенную ОС функций аудита диктуется следующими обстоятельствами:

1) подсистема защиты ОС, не обладая интеллектом, не способна отличить случайные ошибки пользователей от злонамеренных действий. Например, то, что пользователь в процессе входа в систему ввел неправильный пароль, может означать как случайную ошибку при вводе пароля, так и попытку подбора пароля, но, если сообщение о подобном событии записано в журнал аудита, администратор, просматривая этот журнал, легко сможет установить, что же имело место на самом деле — ошибка легального пользователя или атака злоумышленника;
2) администраторы ОС должны иметь возможность получать информацию не только о текущем состоянии системы, но и о том, как она функционировала в недавнем прошлом. Журнал аудита дает такую возможность, накапливая информацию о важных событиях, связанных с безопасностью системы;
3) если администратор ОС обнаружил, что против системы проведена успешная атака, ему важно выяснить, когда была начата атака и каким образом она осуществлялась. При наличии в системе аудита, вся необходимая для этого информация может содержаться в журнале аудита.

Большинство экспертов по компьютерной безопасности сходятся во мнении, что привилегия работать с подсистемой аудита не должна предоставляться администраторам ОС. Другими словами, множество администраторов и множество аудиторов не должны пересекаться. При этом создается ситуация, когда администратор не может выполнять несанкционированные действия без того, чтобы это тут же не стало известно аудиторам, что существенно повышает защищенность системы от несанкционированных действий администратора.

Основные требования к подсистеме аудита ОС:

1) только ОС может добавлять записи в журнал аудита. Если предоставить это право некоторому пользователю, то он или она может скомпрометировать каких-либо других пользователей путем ввода в журнал аудита дополнительных записей;
2) запрет на редактирование и (или) удаление отдельных записей в журнале аудита (можно только, имея соответствующие права, очищать журнал полностью);
3) только пользователи-аудиторы, обладающие соответствующей привилегией, могут просматривать журнал аудита;
4) только пользователи-аудиторы могут очищать, полностью удаляя записи, журнал аудита. После этой процедуры в журнал автоматически вносится соответствующая запись с указанием времени ее осуществления и имени пользователя, очистившего журнал. ОС должна поддерживать возможность сохранения журнала в другом файле перед его очисткой;
5) в момент исчерпания журналом аудита места, выделенного для его размещения, ОС должна аварийно завершить свою работу. После ее перезагрузки работать в ней могут только пользователи-аудиторы. ОС сможет функционировать в обычном режиме лишь после очистки журнала аудита.

Под **политикой аудита** понимают совокупность правил, определяющих то, какие события и в какие моменты времени должны регистрироваться в журнале аудита. Для обеспечения надежной защиты ОС, в журнале аудита должны обязательно регистрироваться следующие события:

1) попытки входа-выхода из системы;
2) попытки изменения списка пользователей;
3) попытки изменения политики безопасности, в т. ч. и политики аудита.

При определении политики аудита не следует ограничиваться регистрацией событий только из перечисленных классов. Как правило, политика аудита в значительной степени определяется спецификой информации, хранимой и обрабатываемой в системе.

Управление политикой безопасности

Политика безопасности должна постоянно поддерживаться в актуальном состоянии, быстро реагировать на изменение условий функ-

ционирования системы с данной ОС, требований к защите информации, хранимой и обрабатываемой в системе, и т. д. Администратор системы управляет политикой безопасности используя соответствующих программные средства, встроенные в систему.

Разграничение доступа к объектам операционной системы

Понятие объекта, субъекта и метода доступа

Для понимания принципов и алгоритмов разграничения доступа введем основные понятия, нужные для этого.

Объектом доступа (**объектом**) называют любой элемент ОС, доступ к которому различных субъектов доступа может быть каким-то образом ограничен.

Методом доступа к объекту называется операция, которая может быть применена к некоторому объекту. Например, для файлов могут быть определены методы доступа «чтение», «запись» и «исполнение» (как в системах Linux).

Субъектом доступа называют любую сущность, которая способна выполнить операции над объектами (использовать по отношению к объектам доступа некоторые методы доступа). Иногда пользователей, а иногда запущенные ими процессы считают субъектами доступа (это имеет чисто теоретическое значение).

Другими словами, объект доступа — это то, к чему осуществляется доступ, субъект доступа — тот, кто осуществляет доступ, и метод доступа — это то, как осуществляется доступ.

Для объекта доступа может быть определен **владелец** — субъект, которому принадлежит данный объект и который несет ответственность за конфиденциальность содержащейся в объекте информации, а также за целостность и доступность объекта. Обычно владельцем объекта автоматически назначается субъект, создавший данный объект, в дальнейшем владелец объекта может быть изменен с использованием соответствующего метода доступа к объекту. Владелец объекта не может быть лишен некоторых прав на доступ к этому объекту. На владельца, как правило, возлагается ответственность за корректное ограничение прав доступа к данному объекту других субъектов.

Правом доступа к объекту называется право, предоставляемое для выполнения доступа к этому объекту с помощью некоторого метода или группы методов. В последнем случае право доступа дает субъекту возможность осуществлять доступ к объекту по любому методу из этой группы. Например, пользователь имеет право на чтение файла, если он может прочитать его содержимое, а на запись, если он или она может его редактировать.

Говорят, что субъект имеет некоторую **привилегию**, если он имеет право на доступ по некоторому методу или группе методов ко всем объектам ОС, поддерживающим данный метод доступа. Например, если субъект ОС Windows имеет привилегию отлаживать программы, он имеет право доступа ко всем объектам типа «процесс» и «поток» по группе методов, используемых отладчиками при отладке программы.

Разграничением доступа субъектов к объектам называется набор правил, определяющих для каждой тройки «субъект — метод — объект», разрешен ли доступ данного субъекта к данному объекту.

В некоторых ОС существует понятие «суперпользователь». **Суперпользователем** является пользователь, который может изменять правила разграничения доступа к объектам.

Правила разграничения доступа, действующие в ОС (более точно, в ФС, существующей в данной ОС), устанавливаются суперпользователем в соответствии с текущей политикой безопасности. Требования, которым они должны удовлетворять, следующие:

1) правила разграничения доступа должны соответствовать подобным административным нормам, принятым в организации, в которой работает данная компьютерная система. Например, если, согласно правилам организации, доступ пользователя к некоторой информации запрещен, то доступ к информации этого вида в компьютерной системе также должен быть ему запрещен;
2) правила разграничения доступа не должны допускать разрушающего воздействия субъектов доступа на хранящуюся информацию и саму ОС, которое выражается в несанкционированном изменении, удалении или другом воздействии на объекты;
3) любой объект доступа должен иметь владельца, другими словами, присутствие «ничьих» объектов недопустимо;
4) присутствие недоступных объектов, т. е. объектов, к которым не может обратиться ни один субъект доступа ни по одному ме-

тоду доступа, недопустимо. Недоступные объекты фактически бесполезно растрачивают аппаратные ресурсы компьютера;

5) утечка конфиденциальной информации недопустима.

Рассмотрим основные модели разграничения доступа.

Избирательное разграничение доступа (рис. 11.1). Система правил избирательного разграничения доступа (англ. *Discretionary Access Control*) формулируется следующим образом:

1) у каждого объекта есть владелец;
2) владелец объекта может произвольным образом ограничивать доступ других субъектов к данному объекту;
3) для каждой тройки «субъект — метод — объект» доступ определен однозначно;
4) существует хотя бы один привилегированный пользователь (суперпользователь), который может переопределить доступ к любому объекту (как правило, став его владельцем).

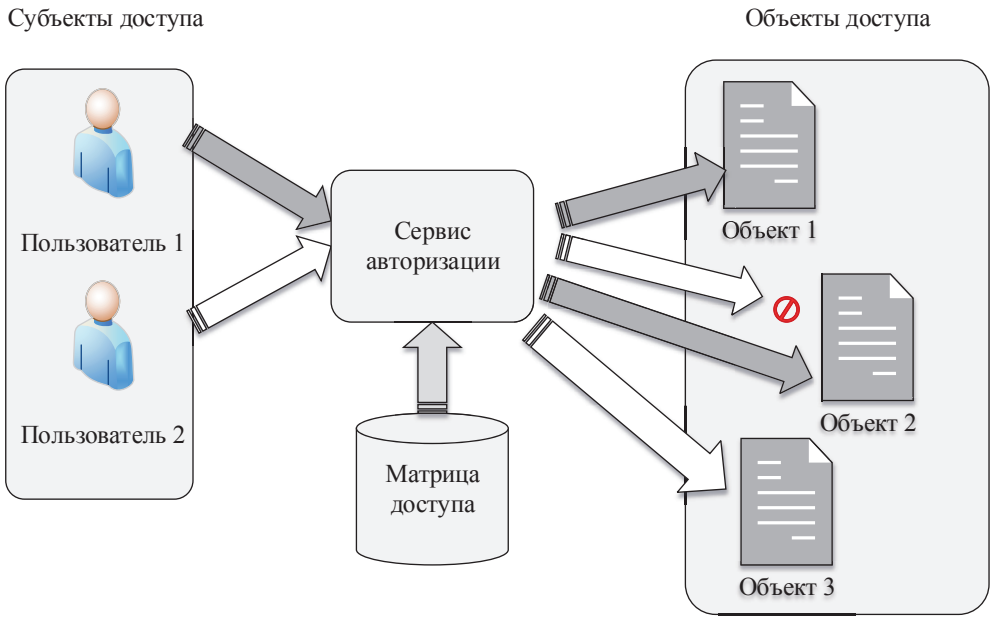


Рис. 11.1. Система избирательного разграничения доступа

Последнее требование введено для реализации механизма удаления потенциально недоступных объектов.

При создании объекта его владельцем назначается субъект, создавший данный объект. В дальнейшем это может быть изменено пользователем, у которого есть соответствующие привилегии в системе.

Для определения прав доступа субъектов к объектам, при избирательном разграничении доступа используется матрица доступа. Строки этой матрицы представляют собой объекты, столбцы — субъекты (или наоборот). В каждой ячейке матрицы хранится совокупность прав доступа, предоставленных данному субъекту на данный объект.

Поскольку матрица доступа очень велика, матрица доступа никогда не хранится в системе в явном виде. Для сокращения объема матрицы доступа часто в системах используется понятие группы как именованного объединения субъектов доступа. Права, предоставленные такой группе в целом, предоставляются и каждому члену группы.

На практике используется два способа кодирования строки матрицы доступа:

1) вектор доступа (UNIX, Linux) — вектор определенной длины, состоящий из нескольких подвекторов. Каждый подвектор описывает права доступа к данному объекту некоторого субъекта. С помощью такого вектора можно задать права доступа к объекту только ограниченного числа субъектов, что накладывает ограничения и на саму систему разграничения доступа;
2) список контроля доступа (*ACL — Access Control List* в Windows) — список переменной длины, элементами которого являются структуры, содержащие:
   - идентификатор субъекта;
   - права, предоставленные этому субъекту на данный объект;
   - различные флаги и атрибуты.

Можно рассматривать вектор доступа как список контроля доступа фиксированной длины, т. е. считать его частным случаем списка контроля доступа.

При создании нового объекта, владелец объекта может и должен определить права доступа различных субъектов к этому объекту. Если владелец объекта не сделал этого, то такому новому объекту должны быть назначены атрибуты защиты по умолчанию либо новый объект наследует атрибуты защиты от родительского объекта (каталога). Второй случай характерен для большинства современных файловых систем, т. к. легко реализуется в объектно ориентированной технологии.

Избирательное разграничение доступа — это наиболее распространенная схема разграничения доступа в современных ОС. Это обусловлено его сравнительной простотой реализации и достаточным удобством правил для пользователей системы. Однако степень защищенности ОС, подсистема защиты которой реализует это разграничение доступа, невысока. В большинстве стран запрещено хранить информацию, содержащую государственную тайну, в компьютерных системах, где реализовано только избирательное разграничение доступа.

Полномочное разграничение доступа. Полномочное, или мандатное, разграничение доступа (рис. 11.2) обычно применяется в совокупности с избирательным. Рассмотрим этот вариант.

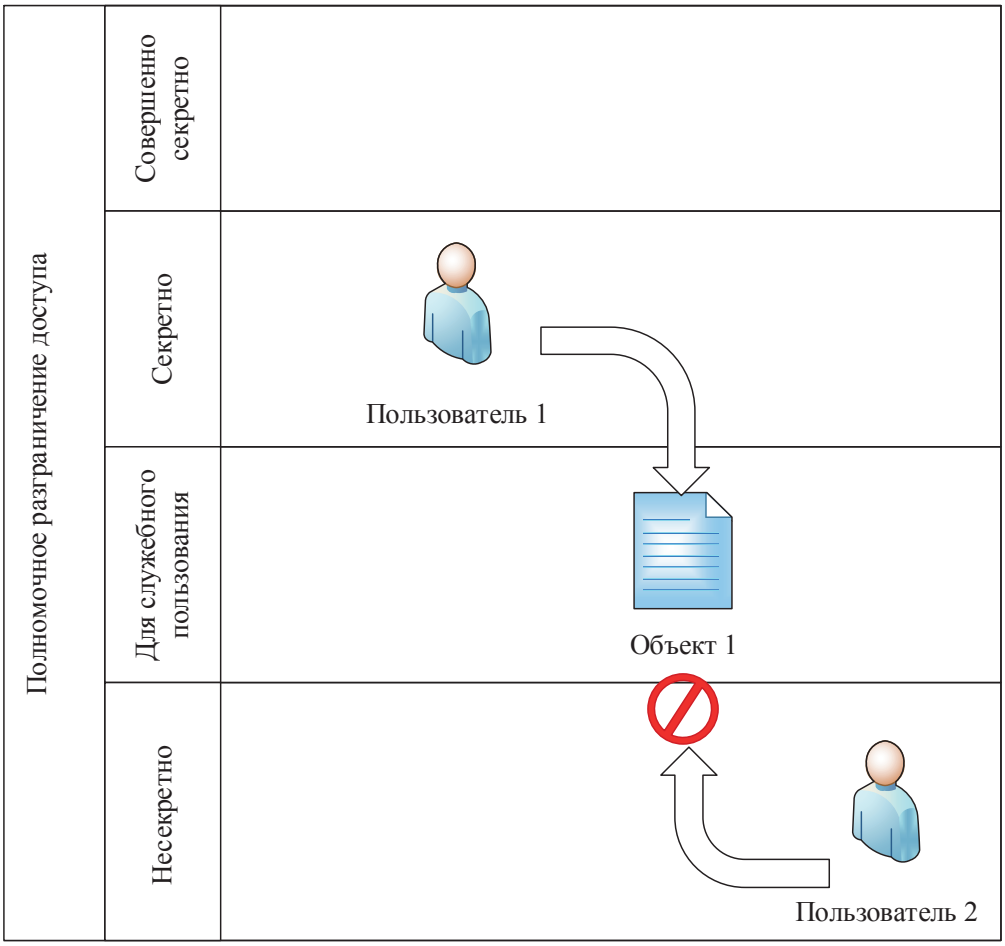


Рис. 11.2. Система полномочного разграничения доступа

В системе с полномочным разграничением доступа правила формулируются следующим образом:

1) выполняются все требования избирательного контроля доступа;
2) во всем множестве объектов доступа ОС выделяется подмножество объектов полномочного разграничения доступа. Каждый объект полномочного разграничения доступа имеет гриф секретности, обычно числовое значение. Чем больше значение грифа секретности, тем более секретный объект, и, наоборот, нулевое значение грифа секретности означает, что объект несекретен. Если объект не является объектом полномочного разграничения доступа, суперпользователь может обратиться к нему по любому методу, как в избирательной модели разграничения доступа;
3) каждый субъект доступа имеет уровень допуска (также имеет числовое значение). Чем выше числовое значение уровня допуска, тем более высокий уровень допуска имеет субъект. При нулевом значении уровня допуска, субъект не имеет допуска к объектам с полномочным разграничением допуска;
4) доступ субъекта к объекту запрещен независимо от состояния матрицы доступа, если:
   - объект является объектом полномочного разграничения доступа,
   - гриф секретности объекта строго выше уровня допуска субъекта, обращающегося к нему,
   - субъект открывает объект в режиме, допускающем чтение информации.

Объектами полномочного разграничения доступа обычно являются файлы. К объектам полномочного разграничения доступа следует относить файлы, в которых может храниться секретная информация, и не относить файлы, в которых секретная информация храниться не может (например, двоичные файлы).

В данной модели разграничения доступа, администраторам ОС, как правило, назначается максимальный уровень допуска.

Поскольку данная модель не дает ощутимых преимуществ по сравнению с предыдущей и в то же время существенно сложнее ее в технической реализации, на практике эта модель используется крайне редко.

Ролевое разграничение доступа (рис. 11.3). Ролевое разграничение доступа (англ. *Role Based Access Control*, *RBAC*) — развитие политики избирательного управления доступом, при этом права доступа субъек-

тов системы группируются по объектам с учетом специфики их применения, образуя роли. Применяется в основном в СУБД.

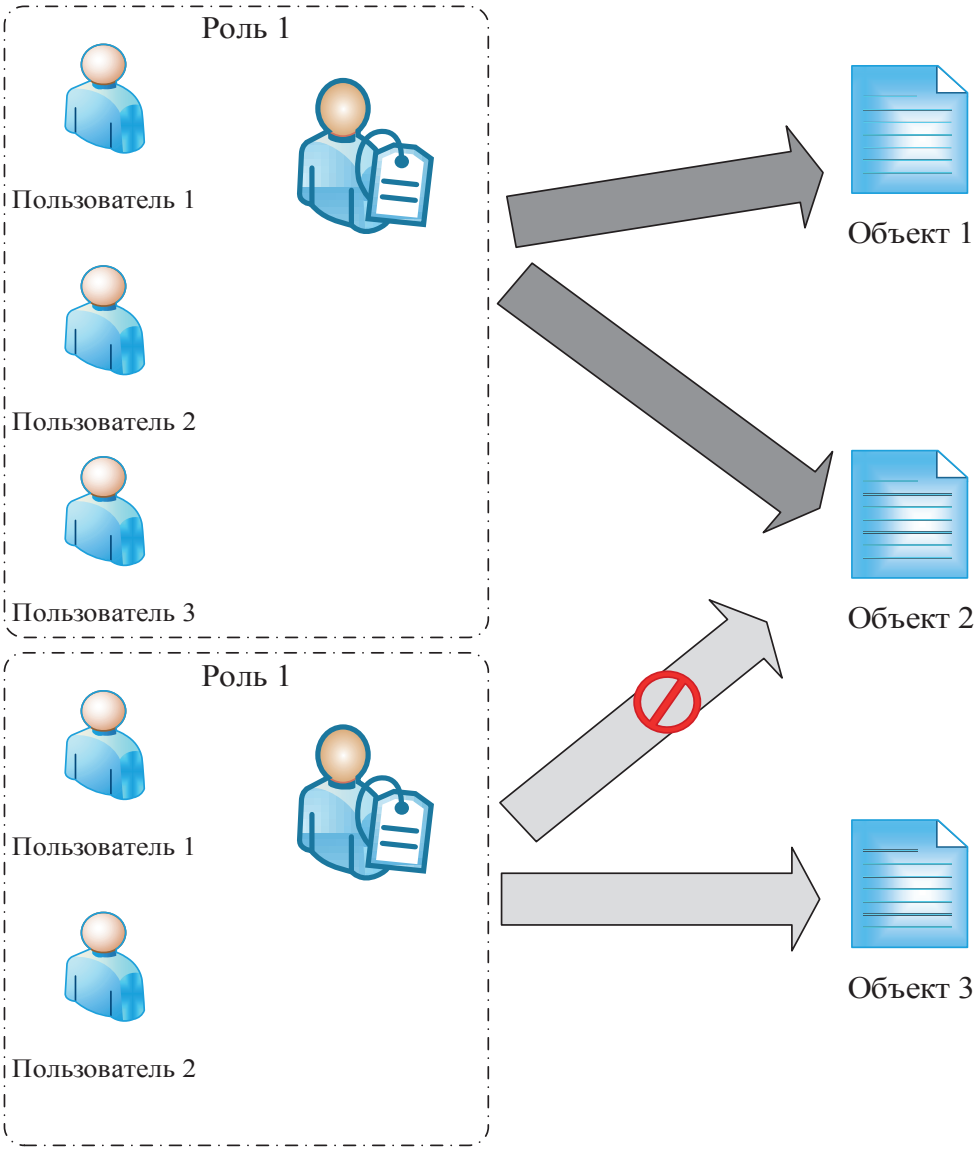


Рис. 11.3. Ролевое разграничение доступа

Классификация уровней защиты ОС

Пример требований к безопасности ОС сформулирован в издании Национального центра кибербезопасности (*NCSC — National Cybersecurity Center*) Министерства обороны США «Trusted Computer System Evaluation Criteria» (критерии оценки доверенных компьютерных систем), известном как «Оранжевая книга». Аналогом «Оранжевой книги» можно считать международный стандарт ISO/IEC 15408, который был опубликован в 2005 г. NCSC опубликовал также различные интерпретации «Оранжевой книги», разъясняющие положение этого документа применительно к условиям работы и компонентам системы. Так, издание «Trusted Network», или «Красная книга», — это интерпретация «Оранжевой книги» относительно сетевых компонент защищенной системы. «Голубая книга» интерпретирует требования «Оранжевой книги» к компонентам подсистем.

«Оранжевая книга» относится к оценочным стандартам, и речь в ней идет не о безопасных, а о доверенных системах. Под доверенной системой в стандарте понимается «система, использующая аппаратные и программные средства для обеспечения одновременной обработки информации разной категории секретности группой пользователей без нарушения прав доступа» [48]. Каких-либо абсолютных систем (в т. ч. и безопасных) в реальной жизни не существует, поэтому и было предложено оценивать лишь степень доверия, которое можно оказать той или иной системе. Безопасность и доверие оцениваются в данном стандарте с точки зрения управления доступом к информации, что и является средством обеспечения конфиденциальности, целостности и доступности.

В «Оранжевой книге» определены средства, наличие которых у ОС делает ее доверенной. Эти средства делятся на 4 уровня: *D*, *C*, *B* и *A*, из которых наивысшей безопасностью обладает уровень *A*. Разделы *C*, *B* и *A* иерархически разбиты на серии подуровней, называющихся классами: C_1, C_2, B_1, B_2, B_3 и A_1. Каждый уровень и класс ужесточает или дополняет требования, указанные в предшествующем уровне или классе.

Уровень *D* — минимальная защита, к системам этого уровня относят все, которые не удовлетворяют требованиям более высоких уровней.

Уровень *C* — дискреционное (избирательное) обеспечение секретности.

Класс C_1 — в ОС поддерживается избирательный контроль доступа. Пользователь, начинающий работать с системой, должен подтвердить свою подлинность (идентифицироваться и аутентифицироваться).

Базовым критерием является критерий C_2. Для соответствия уровню C_2 в ОС должны присутствовать следующие средства:

1) средство защищенной регистрации в системе требует, чтобы пользователь идентифицировал себя посредством ввода уникального идентификатора и пароля прежде, чем ему будет предоставлен доступ к системе;
2) избирательное разграничение доступа позволяет владельцу ресурса определять, кто имеет доступ к данному ресурсу и какие действия может с ним выполнять;
3) аудит обеспечивает возможность обнаружения и регистрации важных событий, имеющих отношение к попыткам нарушения безопасности или любой попытке создания, использования или разрушения системных ресурсов;
4) защита памяти предотвращает чтение информации, записанной кем-либо в память, после того как этот блок памяти был возвращен ОС. Перед повторным использованием память реинициализируется.

Уровень *B* — полномочное управление доступом. Системы, относящиеся к этому разделу, применяются тогда, когда предъявляются особые требования к безопасности, например, в военных организациях, где необходим еще более высокий уровень защиты, чем тот, что исходно обеспечивается уровнем C_2. В этом случае применяются ОС уровня защиты B_1, известного под названием «Полномочный контроль доступа» (*Mandatory Access Control*). Эти системы используют полномочное разграничение доступа. Критерий B_2 усиливает требования критерия B_1. Должны выполняться все требования критерия B_1. Подсистема защиты систем, удовлетворяющих критерию B_2, реализует формально определенную и четко документированную модель безопасности. Осуществляется поиск скрытых каналов утечки информации. Интерфейс подсистемы защиты четко и формально определен, его архитектура и реализация полностью документированы. Выдвигаются более жесткие требования к идентификации, аутентификации и авторизации пользователей. Существует также критерии B_3, но ничего не известно о наличии ОС, удовлетворяющих этому критерию.

Уровень *A* — верифицируемая безопасность. Содержит один класс A_1; неизвестно систем, удовлетворяющих этому классу.

Известно, что по классу C_2 была сертифицирована система Windows NT 4.0. В документации по этой ОС говорится о возможности усиления класса ее защиты до B_2.

Следует иметь в виду еще следующее: фактически сертификат NCSC описывает максимальный уровень защищенности, который может быть достигнут при использовании данного программного продукта.

За годы, прошедшие со времени разработки «Оранжевой книги» (1983), многие ее требования устарели, появился ряд новых требований к безопасности компьютерных систем, не отраженных в «Оранжевой книге». Это связано с тем, что за это время было обнаружено множество ранее неизвестных угроз безопасности компьютерных систем.

К основным недостаткам «Оранжевой книги» относятся следующие:

1) не рассматриваются криптографические средства защиты информации;
2) практически не рассматриваются вопросы обеспечения защиты системы от DoS-атак;
3) не уделяется внимания вопросам защиты системы от негативных воздействий вредоносного ПО;
4) недостаточно подробно рассматриваются вопросы взаимодействия нескольких экземпляров защищенных систем в локальной или глобальной вычислительной сети;
5) требования к средствам защиты от утечки конфиденциальной информации из защищенной системы ориентированы на хранение конфиденциальной информации в БД и мало приемлемы для защиты электронного документооборота.

Несмотря на ряд этих недостатков, следует подчеркнуть, что публикация «Оранжевой книги» без всякого преувеличения стала эпохальным событием в области информационной безопасности. Появился общепризнанный понятийный базис, без которого даже обсуждение проблем информационной безопасности было бы затруднительным.

Гостехкомиссия России выпускает руководящие документы (РД), которые становятся национальными стандартами в области информационной безопасности. В качестве стратегического направления Гостехкомиссия России выбрала ориентацию на «Общие критерии», важнейшие из них:

1) «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации» [49];

2) «Автоматизированные системы. Защита от несанкционированного доступа к информации. Классификация автоматизированных систем и требования по защите информации» [50].

В первом документе рассматриваются требования к обеспечению защищенности отдельных программно-аппаратных элементов защищенных компьютерных систем. Под **средствами вычислительной техники** понимаются не только аппаратные средства, но и «совокупность программных и технических элементов систем обработки данных, способных функционировать самостоятельно или в составе других систем» [49]. Установлено семь классов защищенности средств вычислительной техники. Самые низкие требования предъявляются к классу 7, самые высокие — к классу 1. Требования этих классов в основном соответствуют аналогичным требованиям «Оранжевой книги», при этом класс 7 соответствует классу D_1 «Оранжевой книги», класс 6 — классу C_1 и т. д., класс 1 — классу A_1. Базовому классу C_2 из «Оранжевой книги» соответствует класс 1Г российской классификации.

Контрольные вопросы

1. Как можно определить: атаку, угрозу, риск, уязвимость?
2. Какие основные атаки на ОС вам известны?
3. Что такое политика безопасности и на каких основных принципах она строится?
4. Какие основные функции должна выполнять подсистема безопасности ОС?
5. Что такое «социальный инжиниринг» и как он применяется на практике?
6. Что такое криптосистема? Как определить ее эффективность?
7. С какой целью используют аудит в подсистеме безопасности?
8. Какие типы средств защиты информации известны? К какому типу можно отнести те, что используют в подсистеме безопасности ОС?
9. Какими свойствами должна обладать информационная система, чтобы считаться безопасной?
10. Что такое «разграничение доступа» в ОС и какие разновидности разграничения доступа вы знаете? Какой из этих видов доступа наиболее часто используется в современных ОС и почему?

12. Вредоносное программное обеспечение

Все атаки на компьютерную систему осуществляются с помощью такого класса ПО, которое называется вредоносным (англ. *Maliciousn software*, *malware*). На рис. 12.1 показан экран браузера Chrome с сообщением об обнаруженном вредоносном ПО. Для того чтобы бороться с врагом, нужно знать его в лицо. Поэтому, понимая, что подробное описание этого класса программных продуктов и продуктов, которые защищают от него, требует отдельной книги (и, возможно, в нескольких частях), решено было включить небольшую главу, представляющую основные понятия из этой области.

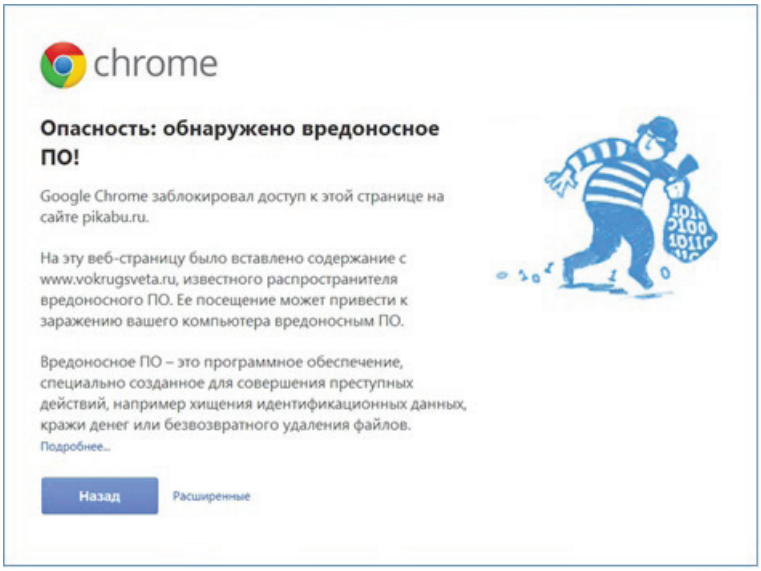


Рис. 12.1. Экран с сообщением об обнаружении вредоносного ПО

Вредоносная программа — программа, предназначенная для осуществления несанкционированного доступа к информации и (или) воздействия на информацию или ресурсы информационной системы [45].

Вредоносным считается любое программное обеспечение, которое пытается заразить компьютер или мобильное устройство. Злоумышленники используют его для самых разных целей, включая получение личных данных и паролей, похищение денежных средств, блокировку доступа к устройству для его владельца. Защититься от вредоносного ПО можно при помощи программного обеспечения, нацеленного на борьбу с этой угрозой.

Законодательные меры против киберпреступлений

Три статьи УК РФ рассматривают вопросы преступлений в сфере компьютерной безопасности: 272 ст. «Неправомерный доступ к компьютерной информации», 273 ст. «Создание, использование и распространение вредоносных программ для ЭВМ», 274 ст. «Нарушение правил эксплуатации ЭВМ, системы ЭВМ или их сети».

Ст. 272 УК РФ предусматривает ответственность за неправомерный доступ к охраняемой законом компьютерной информации, если это деяние повлекло уничтожение, блокирование, модификацию либо копирование компьютерной информации. Впервые в уголовном законодательстве Российской Федерации Федеральным законом от 7 дек. 2011 г. N 420-ФЗ «О внесении изменений в Уголовный кодекс Российской Федерации и отдельные законодательные акты Российской Федерации» в примечании к ст. 272 УК РФ дано понятие компьютерной информации как предмета преступления, к которому теперь относятся сведения (сообщения, данные), представленные в форме электрических сигналов, независимо от средств их хранения, обработки и передачи.

Неправомерный доступ к компьютерной информации — это незаконное либо не разрешенное собственником или иным ее законным владельцем использование возможности получения компьютерной информации. При этом под доступом понимается проникновение в ее источник с использованием средств (вещественных и интеллектуальных) компьютерной техники, позволяющее использовать полученную информацию (копировать, модифицировать, блокировать либо уничтожать ее).

В актуальной на сегодня редакции от 7 дек. 2011 г. ч. 1 ст. 273 УК РФ предусматривается ответственность за «создание, распространение или использование компьютерных программ либо иной компью-

терной информации, заведомо предназначенных для несанкционированного уничтожения, блокирования, модификации, копирования компьютерной информации или нейтрализации средств защиты компьютерной информации» [51].

В соответствии со ст. 274 УК РФ уголовная ответственность наступает за нарушение правил эксплуатации средств хранения, обработки или передачи компьютерной информации и информационно-телекоммуникационных сетей. Диспозиция ч. 1 ст. 274 УК РФ ранее предусматривала ответственность за «нарушение правил эксплуатации ЭВМ, системы ЭВМ или их сети лицом, имеющим доступ к ЭВМ, системе ЭВМ или их сети, повлекшее уничтожение, блокирование или модификацию охраняемой законом информации ЭВМ, если это деяние причинило существенный вред» [51].

Основные типы вредоносного ПО

Вредоносное ПО очень многообразно. Существует множество попыток его классификации. Учитывая, что «Лаборатория Касперского» имеет многолетний опыт борьбы с подобным ПО, приведем классификацию с их сайта.

Основные разновидности вредоносных программ следующие [52].

Вирус (англ. *Virus*) — исполняемый программный код или интерпретируемый набор инструкций, обладающий свойствами несанкционированного распространения и самовоспроизведения. Созданные дубликаты компьютерного вируса не всегда совпадают с оригиналом, но сохраняют способность к дальнейшему распространению и самовоспроизведению [45].

Червь (англ. *Worm*) является в некотором роде вирусом, т. к. создается на основе саморазмножающихся программ. Однако черви не могут заражать существующие файлы. Вместо этого червь поселяется в компьютере и существует отдельным объектом. Он ищет уязвимости в Сети или системе для дальнейшего распространения себя. Черви также могут подразделяться по способу заражения (электронная почта, мессенджеры, обмен файлами и пр.). Некоторые черви существуют в виде сохраненных на жестком диске файлов, а некоторые поселяются лишь в ОП компьютера.

Троянская программа (троян, или троянец, англ. *Trojan*) по своему действию является противоположностью вирусам и червям. Его предлагают загрузить под видом законного приложения, однако вместо заявленной функциональности он делает то, что нужно злоумышленникам. Троянские программы получили свое название от одноименного печально известного мифологического коня, т. к. под видом какой-либо полезной программы или утилиты в систему проникает деструктивный элемент. Трояны не самовоспроизводятся и не распространяются сами по себе. Однако, с увеличением вала информации и файлов в интернете, трояном стало достаточно легко заразиться.

Руткит (англ. *Rootkit*) в современном мире представляет собой особую часть вредоносных программ, разработанных специально, чтобы скрыть присутствие вредоносного кода и его действия от пользователя и установленного защитного программного обеспечения. Это возможно благодаря тесной интеграции руткита с ОС. Некоторые руткиты могут начать свою работу прежде, чем загрузится ОС. Таких называют буткитами.

Иногда дополнительно выделяют следующие типы вредоносного ПО, хотя многие из них являются представителями уже перечисленных классов (троянских программ или сетевых червей).

Программы-вымогатели (рис. 12.2) заражают компьютер, затем шифруют конфиденциальные данные, например, личные документы или фотографии, и требуют выкуп за их расшифровку. Если пользователь отказывается платить, данные удаляются.

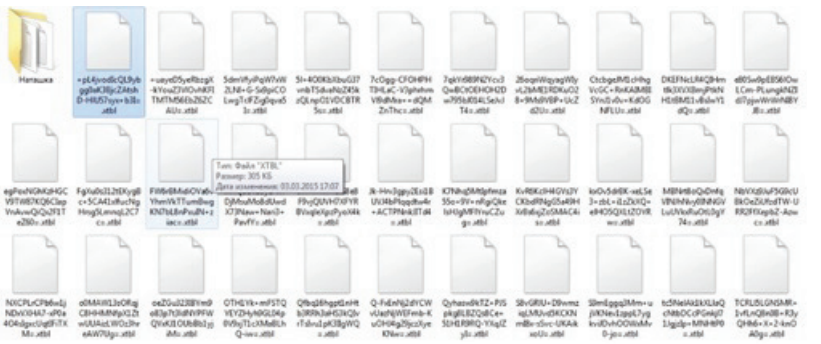


Рис. 12.2. Заражение программой-шифровальщиком (вымогателем)

Некоторые типы программ-вымогателей могут полностью заблокировать доступ к компьютеру. Они могут выдавать свои действия

за работу правоохранительных органов и обвинить пользователя в каких-либо противоправных поступках. При этом проблемы возникают не только из-за новых шифровальщиков. Прошло уже достаточно времени после эпидемии WannaCry, а этот зловред продолжает оставаться в TOP 10 наиболее распространенных семейств троянских программ-шифровальщиков. За третий квартал 2018 г. этот считавшийся побежденным WannaCry пытался атаковать 74 621 пользователя, т. е. за ним числилась почти треть (28,72 %) всех атак шифровальщиков [53].

DoS-программы реализуют атаку с одного компьютера с ведома пользователя. DDoS-программы (*Distributed DoS*) реализуют распределенные атаки с разных компьютеров, причем без ведома пользователя зараженного компьютера. Для этого DDoS-программа засылается любым способом на компьютер «жертв-посредников» и после запуска, в зависимости от текущей даты или по команде от «хозяина», начинает DoS-атаку на указанный сервер в Сети. Принцип самой, пожалуй, нашумевшей DOS-атаки под названием WinNuke был обнародован 7 мая 1997 г. Ее жертвами становились Windows-системы. Автор метода поместил его описание и исходный текст программы в несколько news-конференций. Ввиду его крайней простоты практически каждый человек мог вооружиться этим новейшим оружием. Очевидной первой жертвой стал www.microsoft.com. Данный сервер был выведен из строя на более чем двое суток.

Эксплойты — подвид вредоносных программ. Они содержат данные или исполняемый код, способный воспользоваться одной или несколькими уязвимостями в программном обеспечении на локальном или удаленном компьютере.

Вирусы

Компьютерный вирус — это самое старое вредоносное ПО, может быть, поэтому часто вместо понятия «вредоносное ПО» можно увидеть (услышать) «вирусное ПО». Днем рождения вируса считают 11 нояб. 1983 г., в это время калифорнийский студент Фред Коэн на занятиях по компьютерной безопасности занимался написанием программы, которая способна к саморазмножению и паразитическому распространению по сетям. Особой опасности программа Коэна не представляла, поскольку эксперимент был контролируемым и не имел далеко идущих целей.

Вирусы делятся на классы по среде обитания, а эти классы в свою очередь делятся на подклассы по способу заражения. Итак, по среде обитания вирусы делятся:

1) на файловые;
2) загрузочные;
3) макровирусы;
4) скриптовые.

Вирусом можно заразиться разными способами: от нажатия вредоносной ссылки или файла в неизвестном письме до заражения на вредоносном сайте. При этом вирус может выполнять множество разных задач, направленных в первую очередь на принесение вреда ОС. В настоящее время вирусы довольно редки, т. к. создатели такого ПО стараются держать свои программы и их распространение под контролем. В противном случае вирус довольно быстро попадает в руки антивирусных компаний.

Черви

Основные признаками, по которым черви различаются между собой:

1) способ распространения червя;
2) способ запуска копии червя на заражаемом компьютере;
3) методы внедрения в систему;
4) наличие некоторых отличительных характеристик, таких как полиморфизм, невидимость и т. д.

Большинство известных червей распространяется с помощью файлов: вложенного файла в электронное письмо, ссылки на зараженный файл на каком-либо интернет-ресурсе и т. д. Некоторые черви (так называемые «бесфайловые», или «пакетные» черви) распространяются в виде сетевых пакетов, проникают непосредственно в память компьютера и активируют свой код.

Для проникновения на удаленные компьютеры и запуска своей копии черви используют различные методы:

1) социальный инжиниринг (например, текст электронного письма, призывающий открыть вложенный файл);
2) недочеты в конфигурации сети (например, копирование на диск, открытый на полный доступ);
3) ошибки в службах безопасности ОС и приложений;
4) использование специального сборщика. Это небольшая программа, которая сама не является вредоносной. Сборщик забрасы-

вается на поражаемый компьютер, а потом по частям скачивает червя из Сети, собирает его и запускает. Поскольку червь проникает на компьютер по частям, антивирусные программы не могут его обнаружить.

Троянские программы

Принципиальное различие троянских программ от вирусов состоит в том, что вирус представляет собой самостоятельно размножающуюся программу, тогда как троянская программа не имеет возможности самостоятельно распространяется. Однако довольно часто встречаются гибриды — вирусы, вместе с которыми распространяются троянские программы.

Самый распространенный способ проникновения троянской программы — установка его самим пользователем, считающим, что перед ним нужная программа (собственно именно такому пути распространения трояны и обязаны своим названием). Особенно часто троянские программы встречаются в архивах на сайтах, распространяющих взломанное и нелицензионное программное обеспечение, а также на хакерских сайтах под видом дистрибутивов программ, генераторов серийных номеров либо хакерских утилит.

В последнее время наиболее часто такое вредоносное ПО маскируется под антивирусное. Скачивая некую антивирусную программу с некоторого сайта, вы заносите себе на компьютер троянскую программу. Если вы и не загружаете из Сети подозрительного содержимого, то все равно не можете считать себя в безопасности — троянские функции бывают встроены и в лицензионное программное обеспечение. Например, в сент. 2002 г. «Лаборатория Касперского» сообщила об обнаружении троянских функций в коммерческой программе, предназначенной для просмотра и редактирования графических изображений, — *Firehand Ember Millennium*.

Авторы троянских программ внимательно следят за новостями об обнаружении дыр в ОС и прикладных программах работы с электронной почтой, а также в интернет-браузерах. Цель этих действий — проникновение троянов в компьютеры через такие дыры в ОС либо их загрузка при просмотре web-страниц. Например, троян Trojan.JS. Scob и его модификации, написанные на языке JavaScript, размещались на взломанных серверах и, при дальнейшем просмотре страниц это-

го сервера пользователями, вызывали загрузку на их компьютеры троянских программ.

Разновидности троянских программ (по Касперскому)

Троянские программы классифицируются в соответствии с типом действий, выполняемых ими на компьютере.

Троянские утилиты удаленного администрирования (Backdoor). Троянские программы этого класса являются утилитами удаленного администрирования (управления) компьютеров. В общем они очень похожи на легальные утилиты того же направления (в составе Windows существует подобная программа — «Удаленный рабочий стол»). Единственное, что определяет их как вредоносные программы, — это их действия без ведома пользователя. Данная программа при установке и (или) загрузке не выдает никаких уведомлений. Таким образом, обладатель конкретной копии данного ПО может без ведома пользователя осуществлять операции разного рода.

Похитители паролей (*Trojan-PSW*). Такие программы занимаются тем, что воруют пароли. Проникнув на компьютер и установившись, троян сразу приступает к поиску файлов, содержащих соответствующую информацию. Кража паролей не единственная спецификация программ этого класса, они также могут красть информацию о системе, файлы, номера счетов, коды активации другого ПО и т. д.

Загрузчики (*Trojan-Downloader*). Эти трояны занимаются несанкционированной загрузкой другого вредоносного ПО на компьютер ничего не подозревающего пользователя. После загрузки программа либо инсталлируется, либо записывается трояном в автозагрузку (это в зависимости от возможностей ОС).

Установщики (*Trojan-Dropper*). Такие трояны устанавливают на компьютер-жертву программы, как правило, вредоносные. Анатомия троянов данного класса следующая: основной код, файлы. Основной код собственно и является трояном. Файлы — это программа, которую он должен установить. Троянская программа записывает ее в каталог (обычно временных файлов) и устанавливает. Установка происходит либо незаметно для пользователя, либо с выбросом сообщения об ошибке.

Троянские прокси-серверы (*Trojan-Proxy*) — семейство троянских программ, скрытно осуществляющих доступ к различным интернет-ресурсам обычно с целью рассылки спама.

Шпионские программы (*Trojan-Spy*). Эти троянские программы следят за пользователем: записывают информацию, набранную с клавиатуры, собирают снимки экрана и т. д. Представители таких программ — Log Writers или Key Loggers. Это тип троянов, копирующий всю информацию, вводимую с клавиатуры, и записывающий ее в файл, который впоследствии будет либо отправлен на определенный адрес электронной почты, либо просмотрен через FTP (*File Transfer Protocol*).

Интернет-кликеры (*Trojan-clicker*). Данное семейство троянских программ занимается организацией несанкционированных обращений к интернет-ресурсам путем отправления команд интернет-браузерам или подмены системных адресов ресурсов. Злоумышленники используют данные программы для следующих целей:

1) увеличения посещаемости каких-либо сайтов (для увеличения количества показов рекламы);
2) организации атаки на сервис;
3) привлечения потенциальных жертв для заражения вредоносным программным обеспечением.

Пример троянской программы (KeyPass)

Троян был обнаружен более чем в 20 странах — основной удар пришелся на Бразилию и Вьетнам, также имелись жертвы в Европе, Африке и Юго-Восточной Азии.

KeyPass шифрует все файлы (вне зависимости от их расширения) на локальных дисках и в сетевых папках, доступных с зараженного компьютера. Некоторые он при этом игнорирует — список таких файлов «зашит» в сам зловред. После шифрования файлы приобретают расширение *.KEYPASS; в каждую директорию, содержащую зашифрованные данные, записывается файл «!!! KEYPASS_DECRYPTION_INFO!!!.txt» с требованием выкупа. Содержимое файла показано на рис. 12.3.

Реализована очень простая схема: используется симметричный алгоритм шифрования AES-256 в режиме Cipher Feedback (CFB) с нулевым вектором инициализации и одним и тем же 32-байтовым ключом для всех файлов. Шифруется максимум 0x500 000 байтов (порядка 5 Мбайт) данных в начале каждого файла.

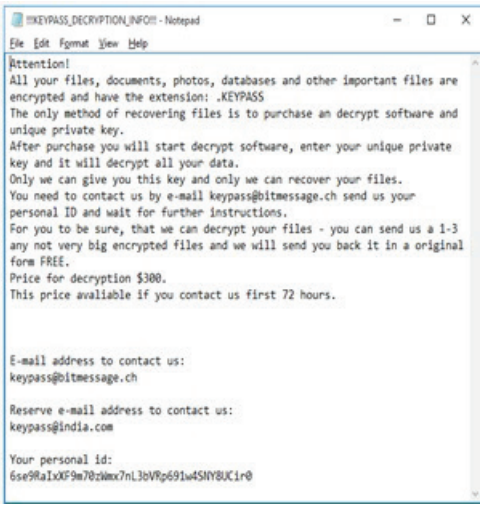


Рис. 12.3. Пример работы троянской программы KeyPass

Эксплойты

Эксплойты (пример обнаружения показан на рис. 12.4) вызываются ошибками в процессе разработки ПО, в результате которых в системе защиты программ оказываются уязвимости, которые успешно используются киберпреступниками для получения неограниченного доступа к самой программе, а через нее дальше — ко всему ПК.

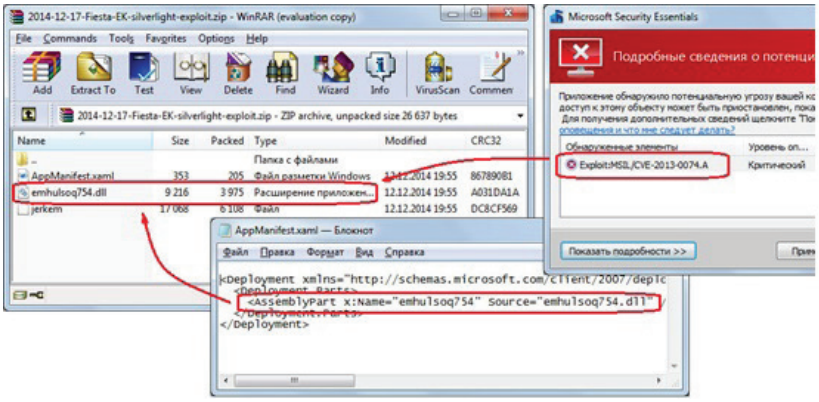


Рис. 12.4. PDF-эксплойт

Браузеры, наряду с Flash, Java и Microsoft Office, являются одними из самых подверженных атакам категорий ПО. Из-за того что эти про-

граммы установлены почти на всех ПК, их активно исследуют как эксперты по безопасности, так и хакеры, а разработчики браузеров вынуждены регулярно выпускать патчи для исправления уязвимостей.

Особую проблему представляют собой эксплойты неизвестных уязвимостей, обнаруженные и использованные киберпреступниками, так называемые «уязвимости нулевого дня». Может пройти много времени, прежде чем производители ПО узнают о наличии проблемы и устранят ее.

Наиболее известные наборы эксплойтов (эксплойт-киты):

1) *Angler* — один из самых сложных наборов на черном рынке. Это один из тех наборов эксплойтов, которые быстрее всего включают в свой арсенал недавно открытые уязвимости нулевого дня, а его вредоносные программы работают в ОП, без записи на жесткие диски жертв;
2) *Nuclear Pack* — поражает жертв за счет использования эксплойтов Java и Adobe PDF, кроме того, устанавливает Caphaw, который является известным банковским трояном;
3) *Neutrino* — набор от русскоязычных разработчиков, содержащий несколько эксплойтов Java. Neutrino стал известен тем, что владелец выставил его на продажу по цене 34 тыс. долл.;
4) *Blackhole Kit* — наиболее распространенная web-угроза в 2012 г., нацеленная на уязвимости в старых версиях браузеров Firefox, Chrome, Internet Explorer и Safari, а также многих популярных плагинов, таких как Adobe Flash, Adobe Acrobat и Java. После того как жертву заманили или перенаправили на подставную страницу, сложный скрипт определяет содержимое машины жертвы и загружает те эксплойты, для которых данный компьютер уязвим. Blackhole, в отличие от большинства других эксплойт-китов, даже удостоился отдельной статьи в «Википедии», хотя после ареста его разработчика сам набор практически не используется.

Признаки наличия вредоносного ПО на компьютере

Заподозрить заражение вредоносным ПО компьютера можно по следующим признакам:

1) отказ работы одной либо нескольких программ, особенно антивируса и брандмауэра;

2) появление всплывающих окон, содержащих рекламу;
3) время от времени появление окна удаленного соединения, которое не было инициировано пользователем;
4) при отсутствии активности пользователя на подключенном к интернету компьютере (пользователь ничего не скачивает, программы общений неактивны и т. д.), индикаторы подключения к сети продолжают показывать обмен информацией;
5) стартовая страница браузера постоянно меняется, а страница, указанная вами в роли стартовой, не сохраняется;
6) при попытке посетить сайты, куда пользователь раньше легко заходил (например, в поисковые системы), компьютер переадресовывает его на незнакомый сайт, часто содержащий рекламную информацию.

Правила, которых следует придерживаться, для снижения риска заражения

Риск заражения можно уменьшить, придерживаясь следующих правил:
1) не работать в системе с правами администратора; желательно работать с ограниченными правами, а для запуска программ, требующих больших прав, использовать пункт «Запустить от имени» в контекстном меню. В современных версиях ОС реализована возможность выполнения административных действий любым пользователем, но при этом система предупреждает о возможных последствиях и требует ввода административного пароля;
2) не загружать программ из непроверенных источников, прежде всего это относится к сайтам, распространяющим взломанное, нелицензионное программное обеспечение и хакерские утилиты;
3) без необходимости не допускать к своему компьютеру посторонних пользователей;
4) регулярно делать резервные копии важной информации и файлов;
5) пользоваться малораспространенными программами для работы в интернете или хотя бы не теми, что установлены по умолчанию (например, браузером Opera, Mozilla, Chrome). Этот подход имеет массу недостатков, но на уровне частных пользователей нередко оказывается самым действенным;

6) пользоваться нестандартными брандмауэрами;
7) переименовывать исполняемые файлы антивирусных программ и брандмауэров, а также сервисы, запускаемые ими;
8) пользоваться мониторами реестра, в которых необходимо включить слежение за разделами, управляющими автозапуском (обычно называются Run или RunOnce);
9) сделать снимок файлов в системных каталогах и при появлении новых попытаться определить, что это за файл и откуда он взялся, либо применять специальные программы — ревизоры диска, которые позволяют выявить новые подозрительные файлы, а также изменение размера существующих;
10) не запускать программ, полученных от неизвестных лиц;
11) включать на компьютере отображение всех расширений файлов и внимательно следить за полным именем файла. Троянская программа может встроить свой код в файл, имеющий двойное расширение (первое — обычное и безопасное, например, картинки gif, а второе — расширение исполняемого файла);
12) регулярно устанавливать обновления для ОС и используемых программ;
13) не разрешать браузеру запоминать пароли и не хранить их в слабо защищенных программах хранения паролей. В том случае, если вам удобнее не запоминать пароли, стоит подумать над установкой программы, которая сохраняет вводимые в нее записи в стойко зашифрованном виде.

Контрольные вопросы

1. Что такое вредоносное ПО?
2. Какое наказание можно понести за создание и использование вредоносного ПО?
3. Какие вы знаете основные классы вредоносного ПО? Какие особенности этих классов?
4. Какие основные признаки того, что компьютер заражен вредоносной программой?
5. Каких основных правил нужно придерживаться, чтобы свести риск заражения вредоносным ПО к минимуму?

Библиографический список

1. Введение в программные системы и их разработку. ИНТУИТ Национальный открытый университет : [сайт]. — URL: http://www.intuit.ru/studies/courses/3632/874/lecture/14291?page=2 (дата обращения: 01.06.2019). — Загл. с титул. экрана.
2. Прикладное программное обеспечение//ВикипедиЯ : [сайт]. — URL: https://ru.wikipedia.org/wiki/Прикладное_программное_обеспечение (дата обращения: 01.06.2019). — Загл. с титул. экрана.
3. Гордеев, А. В. Операционные системы: учебник для вузов / А. В. Гордеев. — 2-е изд.– Санкт-Петербург : Питер, 2009. — 416 с. — ISBN 978-5-94723-632-3.
4. Таненбаум, Э. Современные операционные системы : пер. с англ. / Э. Таненбаум, Х. Бос. — 4-е изд. — Санкт-Петербург : Питер (Серия «Классика computer science»), 2015. — 1120 с. — ISBN 978-5-496-01395-6 (в пер.).
5. Олифер, В. Г. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. — 2-е изд. — Санкт-Петербург : Питер, 2009. — 669 с. — ISBN 978-5-91180-528-9.
6. Brooks, Frederick Philips. The Mythical Man-Month / F. P. Brooks. — Addison Wesley : MA, United States, Longman Inc., 1995. — 321p. — ISBN 978-0-201-83595-3.
7. Хронология Windows//ВикипедиЯ : [сайт]. — URL: https://ru.wikipedia.org/wiki/Windows#/media/Файл: Windows_Updated_Family_Tree.png (дата обращения: 01.07.2019). — Загл. с титул. экрана.
8. Государство. Бизнес. ИТ//TAdviser : [сайт]. — URL: http://www.tadviser.ru/index.php/Продукт: Microsoft_Windows (дата обращения: 01.07.2019).). — Загл. с титул. экрана.
9. Стандартная общественная лицензия GNU // Операционная система GNU : [сайт]. — URL: https://www.gnu.org/licenses/gpl-3.0.ru.html (дата обращения: 01.07.2019). — Загл. с титул. экрана.
10. Red Hat : [сайт]. — URL: https://www.redhat.com/(дата обращения: 01.07.2019). — Загл. с титул. экрана.

11. Как Red Hat убила свой основной продукт и стала многомиллиардной корпорацией // Habr : [сайт]. — URL: https://habr.com/ru/company/redhatrussia/blog/351170/(дата обращения: 01.07.2019).). — Загл. с титул. экрана.
12. Debian : [сайт] — URL: https://ubuntu.com/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
13. Ubuntu : [сайт]. — URL: https://ubuntu.com/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
14. Колисниченко, Д. Н. Ubuntu 10. Краткое руководство пользователя / Д. Н. Колисниченко. — Санкт-Петербург : БХВ-Петербург, 2010. — 304 с. — ISBN 978-5-9775-0598-7.
15. FreeBSD : [сайт]. — URL: https://www.freebsd.org/ru/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
16. Apple : [сайт]. — URL: https://www.apple.com/ru/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
17. История macOS от System 1.0 1984 до macOS Mojave.//Я : [сайт]. — URL: https://yablyk.com/416385-illyustrirovannaya-istoriya-mac-os-x/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
18. Процессы в Windows//UINC : [сайт]. — URL: http://uinc.ru/articles/38/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
19. Иерархия памяти // ВикипедиЯ: [сайт]. — URL: https://ru.wikipedia.org/wiki/Иерархия_памяти (дата обращения: 01.06.2019). — Загл. с титул. экрана.
20. Интерфейсы подключения жестких дисков — IDE, SATA и другие//PC Information Guide : [сайт]. — URL: http://pc-information-guide.ru/zhestkij-disk/interfejsy-podklyucheniya-zhestkix-diskov-ide-sata-i-drugie.html (дата обращения: 01.08.2019). — Загл. с титул. экрана.
21. Hoffman, Chris. What is UEFI, And How Is It Different From BIOS?//Habr : [сайт]/C. Hoffman. — URL: https://habr.com/ru/company/redhatrussia/blog/351170/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
22. Mnemonic, Johny. Что такое UEFI. Как установить Windows 10 и более старые версии на компьютер с UEFI//F1Comp.ru : [сайт] / J. Mnemonic. — URL: https://f1comp.ru/zhelezo/chto-takoe-uefi-kak-ustanovit-uefi-na-windows-10-i-bolee-starye-versii/(дата обращения: 01.07.2019). — Загл. с титул. экрана.

23. Аршинов, Евгений. Ссылки в Windows символьные и не только // Habr : [сайт]/Е. Аршинов. — URL: https://habr.com/ru/post/50878 (дата обращения: 01.08.2019). — Загл. с титул. экрана.
24. Create Symbolic Links, Hard Links, and Directory Junction Points With MKLINK//TechJourney : [сайт]. — URL: https://techjourney.net/create-symbolic-links-hard-links-and-directory-junction-points-in-windows-with-mklink/(дата обращения: 01.08.2019). — Загл. с титул. экрана.
25. Типы файловых систем — в чем разница между FAT32, NTFS и extFAT // STARUS Recovery : [сайт]. — URL: https://www.starusrecovery.ru/articles/whats-is-the-difference-between-fat32-ntfs-exfat.html (дата обращения: 01.08.2019). — Загл. с титул. экрана.
26. Miroshnichenko, Michael. Сравнение файловых систем ReFS (Resilient File system) и NTFS // Hetman Software : [сайт] / M. Miroshnichenko. — URL: https://hetmanrecovery.com/ru/recovery_news/comparison-of-refs-resilient-file-system-and-ntfs-file-systems.htm (дата обращения: 01.08.2019). — Загл. с титул. экрана.
27. Poirier, Dave. The Second Extended File System : [сайт] / D. Poirier. — URL: http://www.nongnu.org/ext2-doc/ext2/(дата обращения: 01.08.2019). — Загл. с титул. экрана.
28. Джонс, М. Анатомия ext4/М. Джонс//IBM : [сайт] / М. Джонс. — URL: https://www.ibm.com/developerworks/ru/library/l-anatomy-ext4/(дата обращения: 01.07.2019). — Загл. с титул. экрана.
29. Файловая система XFS//Losst : [сайт]. — URL http://https://losst.ru/fajlovaya-sistema-xfs/(дата обращения: 01.08.2019). — Загл. с титул. экрана.
30. Journaled File System Technology for Linux//JFS for Linux : [сайт]. — URL http://jfs.sourceforge.net/(дата обращения: 01.08.2019). — Загл. с титул. экрана.
31. Файловая система ReiserFS//Losst : [сайт]. — URL: https://https://losst.ru/fajlovaya-sistema-reiserfs (дата обращения: 01.08.2019). — Загл. с титул. экрана.
32. BTRFS для самых маленьких//Habr : [сайт]. — URL: https://habr.com/ru/company/veeam/blog/458250/(дата обращения: 01.07.2019). — Загл. с титул. экрана.

33. ZFS: The last word in file systems//Sun Microsystems : [сайт]. — URL: http://web.archive.org/web/20060428092023/http://www.sun.com/2004–0914/feature (дата обращения: 01.07.2019). — Загл. с титул. экрана.
34. Кастер, Хелен. Основы Windows NT и NTFS / Х. Кастер. — Москва : Издательский отдел «Русская редакция», 1996. — 440 с. — ISBN 1-7502-0023-X.
35. Файловая система NTFS//iXBT.com : [сайт]. — URL: https://www.ixbt.com/storage/ntfs.html (дата обращения: 01.07.2019). — Загл. с титул. экрана.
36. Patterson, David A. A Case for Redundant Arrays of Inexpensive Disks (RAID)/David A. Patterson, Garth Gibson and Randy H. Katz. — ACM SIGMOD Record, 17 (3), July 1988. DOI: 10.1145/50202.50214. — URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87–391.pdf (дата обращения: 01.07.2019). — Загл. с титул. экрана.
37. RAID//ВикипедиЯ : [сайт]. — URL: https://ru.wikipedia.org/wiki/RAID (дата обращения: 01.08.2019). — Загл. с титул. экрана.
38. Common RAID Disk Data Format Specification//SNIA : [сайт]. — URL: https://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf (дата обращения: 01.08.2019). — Загл. с титул. экрана.
39. Ядро (операционной системы)//Академик : [сайт]. — URL: https://dic.academic.ru/dic.nsf/ruwiki/1219170 (дата обращения: 01.08.2019). — Загл. с титул. экрана.
40. Руссинович, М. Внутреннее устройство Microsoft Windows / М. Руссинович, Д. Соломон. — 6-е изд. — Санкт-Петербург : Питер, 2013. — 800 с. — ISBN 978-5-459-01730-4.
41. Микроядро//ВикипедиЯ : [сайт]. — URL: https://ru.wikipedia.org/wiki/Микроядро (дата обращения: 01.08.2019). — Загл. с титул. экрана.
42. Henderson, Anthony. The CIA Triad: Confidentiality, Integrity, Availability/A. Henderson//Pammore Institute — 25.03.2017. — URL: https://dic.academic.ru/dic.nsf/ruwiki/1219170 (дата обращения: 01.08.2019). — Загл. с титул. экрана.
43. Лукацкий, А. Триада «конфиденциальность, целостность, доступность» откуда она? / А. Лукацкий // SecurityLab.

ru : [сайт]. — 25.03.2017 — URL: https://www.securitylab.ru/blog/personal/Business_without_danger/24456.php (дата обращения: 01.08.2019). — Загл. с титул. экрана.
44. Stoneburner, Gary. Engineering Principles for Information Technology Security (A Baseline for Achieving Security). NIST Special Publication/Gary Stoneburner, Clark Hayden and Alexis Feringa//NIST – National Institute of Standards and Technology : [сайт]. 21.06.2004. – URL: https://www.nist.gov/publications/engineering-principles-it-security-baseline-achieving-security-revision (дата обращения: 01.08.2019). — Загл. с титул. экрана.
45. ГОСТ Р 50922–2006. Защита информации. Основные термины и определения//Техэксперт: Электронный фонд правовой и нормативно-технической документации : [сайт]. — URL: http://docs.cntd.ru/document/1200058320 (дата обращения: 01.08.2019). — Загл. с титул. экрана.
46. Угроза, атака, риск//Компьютерные сети : [сайт]. — URL: http://iptcp.net/ugroza-ataka-risk.html (дата обращения: 01.08.2019). — Загл. с титул. экрана.
47. Парольная защита//life-prog.ru : [сайт]. — 23.12.2013 — URL: https://life-prog.ru/1_2842_sposobi-ataki-na-parol-obespechenie-bezopasnosti-parolya.html (дата обращения: 01.08.2019). — Загл. с титул. экрана.
48. Критерии определения безопасности компьютерных систем//ВикипедиЯ : [сайт]. — URL: https://ru.wikipedia.org/wiki/Критерии_определения_безопасности_компьютерных_систем (дата обращения: 01.09.2019). — Загл. с титул. экрана.
49. Руководящий документ. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации//ФСТЭК России : [сайт]. — URL: https://fstec.ru/component/attachments/download/297 (дата обращения: 01.09.2019). — Загл. с титул. экрана.
50. Руководящий документ. Автоматизированные системы. Защита от несанкционированного доступа к информации. Классификация автоматизированных систем и требования по защите информации//ФСТЭК России : [сайт]. — URL: https://fstec.ru/component/attachments/download/296 (дата обращения: 01.09.2019). — Загл. с титул. экрана.

51. Методические рекомендации по осуществлению прокурорского надзора за исполнением законов при расследовании преступлений в сфере компьютерной информации//Адвокатское бюро «Домкины и партнеры» : [сайт]. — URL: https://www.advodom.ru/practice/272–273–274.php (дата обращения: 01.09.2019). — Загл. с титул. экрана.
52. Классификация вредоносных программ//kaspersky daily : [сайт]. — URL: https://www.kaspersky.ru/blog/klassifikaciya-vredonosnyx-programm/2200/(дата обращения: 01.09.2019). — Загл. с титул. экрана.
53. Панков, Н. WannaCry: Слухи о моей смерти сильно преувеличены / Н. Панков//kaspersky daily : [сайт]. — 23.11.2018 — URL: https://www.kaspersky.ru/blog/wannacry-is-still-alive/21729/(дата обращения: 01.09.2019). — Загл. с титул. экрана.
54. Что такое троянская программа?//kaspersky : [сайт]. — URL: https://www.kaspersky.ru/resource-center/threats/trojans (дата обращения: 01.09.2019). — Загл. с титул. экрана.

Учебное издание

Зверева Ольга Михайловна

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Редактор И. В. Меркурьева
Верстка О. П. Игнатьевой

Подписано в печать 23.11.2020. Формат 70×100/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 17,7.
Уч.-изд. л. 11,7. Тираж 100 экз. Заказ 231.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620083, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13
Факс: +7 (343) 358-93-06
http://print.urfu.ru


УНИВЕРСИТЕТСКОГО КНИГОИЗДАНИЯ
УНИВЕРСИТЕТСКАЯ КНИГА
2020


9 785799 631468