УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 30.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция №3.

Тема: Архитектура операционной системы

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

Функциональная сложность операционной системы неизбежно приводит к сложности ее архитектуры, под которой понимают структурную организацию ОС на основе различных программных модулей. Обычно в состав ОС входят исполняемые и объектные модули стандартных для данной ОС форматов, библиотеки разных типов, модули исходного текста программ, программные

модули специального формата (например, загрузчик ОС, драйверы ввода-вывода), конфигурационные файлы, файлы документации, модули справочной системы и т. д.

Большинство современных операционных систем представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС.

Ядро и вспомогательные модули ОС

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

ядро — модули, выполняющие основные функции ОС;

модули, выполняющие вспомогательные функции ОС.

Модули ядра выполняют такие базовые функции ОС, как управление процессами, памятью, устройствами ввода-вывода и т. п. Ядро составляет сердцевину операционной системы, без него ОС является полностью неработоспособной и не сможет выполнить ни одну из своих функций.

В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса, такие как переключение контекстов, загрузка/выгрузка станиц, обработка прерываний. Эти функции недоступны для приложений. Другой класс функций ядра служит для поддержки приложений, создавая для них так называемую прикладную программную среду. Приложения могут обращаться к ядру с запросами — системными вызовами — для выполнения тех или иных действий, например для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т. д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования — API.

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями операционной системы, поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в оперативной памяти, то есть являются резидентными.

Ядро является движущей силой всех вычислительных процессов в компьютерной системе, и крах ядра равносилен краху всей системы. Поэтому разработчики операционной системы уделяют особое внимание надежности кодов ядра, в результате процесс их отладки может растягиваться на многие месяцы.

Обычно ядро оформляется в виде программного модуля некоторого специального формата, отличающегося от формата пользовательских приложений.

ПРИМЕЧАНИЕ

Термин «ядро» в разных ОС трактуется по-разному. Одним из определяющих свойств ядра является работа в привилегированном режиме. Этот вопрос будет рассмотрен в следующем разделе.

Остальные модули ОС выполняют весьма полезные, но менее обязательные функции. Например, к таким вспомогательным модулям могут быть отнесены

программы архивирования данных на магнитной ленте, дефрагментации диска, текстового редактора. Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур.

Поскольку некоторые компоненты ОС оформлены как обычные приложения, то есть в виде исполняемых модулей стандартного для данной ОС формата, то часто бывает очень сложно провести четкую грань между операционной системой и приложениями (рис. 3.1).

Решение о том, является ли какая-либо программа частью ОС или! нет, принимает 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 определенное время как пользовательское приложение, а потом стать частью ОС, или наоборот. Ярким примером такого изменения статуса программы является Web-браузер компании Microsoft, который сначала поставлялся как отдельное приложение, затем стал частью операционных систем Windows NT 4.0 и Windows 95/98, а сегодня существует большая вероятность того, что по решению суда этот браузер снова превратится в самостоятельное приложение.

Рис. 3.1. Нечеткость границы между ОС и приложениями

Вспомогательные модули ОС обычно подразделяются на следующие группы:

утилиты — программы, решающие отдельные задачи управления и сопровождения компьютерной системы, такие, например, как программы сжатия дисков, архивирования данных на магнитную ленту;

системные обрабатывающие программы — текстовые или графические редакторы, компиляторы, компоновщики, отладчики;

программы предоставления пользователю дополнительных услуг — специальный вариант пользовательского интерфейса, калькулятор и даже игры;

библиотеки процедур различного назначения, упрощающие разработку приложений, например библиотека математических функций, функций ввода-вывода и т. д.

Как и обычные приложения, для выполнения своих задач утилиты, обрабатывающие программы и библиотеки ОС, обращаются к функциям ядра посредством системных вызовов (рис. 3.2).

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

Рис. 3.2. Взаимодействие между ядром и вспомогательными модулями ОС

Модули ОС, оформленные в виде утилит, системных обрабатывающих программ и библиотек, обычно загружаются в оперативную память только на время выполнения своих функций, то есть являются транзитными. Постоянно в оперативной памяти располагаются только самые необходимые коды ОС, составляющие ее ядро. Такая организация ОС экономит оперативную память компьютера.

Важным свойством архитектуры ОС, основанной на ядре, является возможность защиты кодов и данных операционной системы за счет выполнения функций ядра в привилегированном режиме.

Ядро в привилегированном режиме

Для надежного управления ходом выполнения приложений операционная система должна иметь по отношению к приложениям определенные привилегии. Иначе некорректно работающее приложение может вмешаться в работу ОС и, например, разрушить часть ее кодов. Все усилия разработчиков операционной системы окажутся напрасными, если их решения воплощены в незащищенные от приложений модули системы, какими бы элегантными и эффективными эти решения ни были. Операционная система должна обладать исключительными полномочиями также для того, чтобы играть роль арбитра в споре приложений за ресурсы компьютера в мультипрограммном режиме. Ни одно приложение не должно иметь возможности без ведома ОС получать дополнительную область памяти, занимать процессор дольше разрешенного операционной системой

периода времени, непосредственно управлять совместно используемыми внешними устройствами.

Обеспечить привилегии операционной системе невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум два режима работы — пользовательский режим (user mode) и привилегированный режим, который также называют режимом ядра (kernel mode), или режимом супервизора (supervisor mode). Подразумевается, что операционная система или некоторые ее части работают в привилегированном режиме, а приложения — в пользовательском режиме.

Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро становится той частью ОС, которая работает в привилегированном режиме (рис. 3.3). Иногда это свойство — работа в привилегированном режиме — служит основным определением понятия «ядро».

Рис. 3.3. Архитектура операционной системы с ядром в привилегированном режиме

Приложения ставятся в подчиненное положение за счет запрета 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), тогда как другие запрещается выполнять только при определенных условиях. Например, инструкции ввода-вывода могут быть запрещены приложениям при доступе к контроллеру жесткого диска, который хранит данные, общие для ОС и всех приложений, но разрешены при доступе к последовательному порту, который выделен в монопольное владение для определенного приложения. Важно, что условия разрешения выполнения критичных инструкций находятся под полным контролем ОС и этот контроль обеспечивается за счет набора инструкций, безусловно запрещенных для пользовательского режима.

Аналогичным образом обеспечиваются привилегии ОС при доступе к памяти. Например, выполнение инструкции доступа к памяти для приложения разрешается, если инструкция обращается к области памяти, отведенной данному приложению операционной системой, и запрещается при обращении к областям памяти, занимаемым ОС или другими приложениями. Полный контроль ОС над доступом к памяти достигается за счет того, что инструкция или инструкции конфигурирования механизмов защиты памяти (например, изменения ключей

защиты памяти в мэйнфреймах IBM или указателя таблицы дескрипторов памяти в процессорах Pentium) разрешается выполнять только в привилегированном режиме.

Очень важно, что механизмы защиты памяти используются операционной системой не только для защиты своих областей памяти от приложений, но и для защиты областей памяти, выделенных ОС какому-либо приложению, от остальных приложений. Говорят, что каждое приложение работает в своем адресном пространстве. Это свойство позволяет локализовать некорректно работающее приложение в собственной области памяти, так что его ошибки не оказывают влияния на остальные приложения и операционную систему.

Между количеством уровней привилегий, реализуемых аппаратно, и количеством уровней привилегий, поддерживаемых ОС, нет прямого соответствия. Так, на базе четырех уровней, обеспечиваемых процессорами компании Intel, операционная система OS/2 строит трехуровневую систему привилегий, а операционные системы Windows NT, UNIX и некоторые другие ограничиваются двухуровневой системой.

С другой стороны, если аппаратура поддерживает хотя бы два уровня привилегий, то ОС может на этой основе создать программным способом сколь угодно развитую систему защиты.

Эта система может, например, поддерживать несколько уровней привилегий, образующих иерархию. Наличие нескольких уровней привилегий позволяет более тонко распределять полномочия как между модулями операционной системы, так и между самими приложениями. Появление внутри операционной системы более привилегированных и менее привилегированных частей позволяет повысить устойчивость ОС к внутренним ошибкам программных кодов, так как такие ошибки будут распространяться только внутри модулей с определенным уровнем привилегий. Дифференциация привилегий в среде прикладных модулей позволяет строить сложные прикладные комплексы, в которых часть более привилегированных модулей может, например, получать доступ к данным менее привилегированных модулей и управлять их выполнением.

На основе двух режимов привилегий процессора ОС может построить сложную систему индивидуальной защиты ресурсов, примером которой является типичная система защиты файлов и каталогов. Такая система позволяет задать для любого пользователя определенные права доступа к каждому из файлов и каталогов.

Повышение устойчивости операционной системы, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления выполнения системных вызовов. Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского 'режима в- привилегированный, а при возврате к приложению — переключение из привилегированного режима в пользовательский (рис. 3.4). Во всех типах процессоров из-за дополнительной двукратной задержки переключения переход на процедуру со сменой режима выполняется медленнее, чем вызов процедуры без смены режима.

Рис. 3.4. Смена режимов при выполнении системного вызова к привилегированному ядру

Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической. Ее используют многие популярные операционные системы, в том числе многочисленные версии UNIX, VAX VMS, IBM OS/390, OS/2, и с определенными модификациями — Windows NT.

В некоторых случаях разработчики ОС отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, известная специализированная операционная система NetWare компании Novell использует привилегированный режим процессоров Intel x86/ Pentium как для работы ядра, так и для работы своих специфических приложений — загружаемых модулей NLM (рис. 3.5). При таком построении ОС обращения приложений к ядру выполняются быстрее, так как нет переключения режимов, однако при этом отсутствует надежная аппаратная защита памяти, занимаемой модулями ОС, от некорректно работающего приложения. Разработчики NetWare пошли на такое потенциальное снижение надежности своей операционной системы, поскольку ограниченный набор ее специализированных приложений позволяет компенсировать этот архитектурный недостаток за счет тщательной отладки каждого приложения.

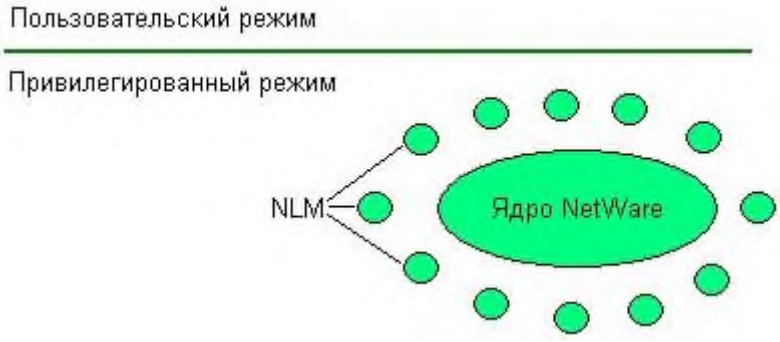

Рис. 3.5. Упрощенная архитектура операционной системы NetWare

В одном режиме работают также ядро и приложения тех операционных систем, 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. Операционная система 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. Архитектура MS-DOS соответствует архитектуре ОС, приведенной на рис. 3.2. Некорректно написанные приложения вполне могли разрушить основные модули MS-DOS, что иногда и происходило, но область использования MS-DOS (и многих подобных ей ранних операционных систем для персональных компьютеров, таких как MSX, CP/M) и не предъявляла высоких требований к надежности ОС.

ПРИМЕЧАНИЕ

Появление в более поздних версиях процессоров Intel (начиная с 80286) возможности работы в привилегированном режиме не было использовано разработчиками MS-DOS. Эта ОС всегда работает на процессорах данного типа в так называемом реальном режиме, в котором эмулируется процессор 8086/88. Не следует считать, что реальный режим является синонимом пользовательского режима, а привилегированный режим — его альтернативой. Реальный режим был реализован только для совместимости поздних моделей процессоров с ранней моделью 8086/88 и альтернативой ему является защищенный режим работы процессора, в котором становятся доступными все особенности процессоров поздних моделей, в том числе и работа на одном из четырех уровней привилегий.

Многослойная структура ОС

Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев: нижний слой образует аппаратура, промежуточный — ядро, а утилиты, обрабатывающие программы и приложения, составляют верхний слой системы (рис. 3.6). Слоистую структуру вычислительной системы принято изображать в виде системы концентрических окружностей, иллюстрируя тот факт, что 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. 3.6. Трехслойная схема вычислительной системы

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе и программных. В соответствии с этим подходом система состоит из иерархии слоев. Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций,

которые образуют межслойный интерфейс (рис. 3.7). На основе функций нижележащего слоя следующий (вверх по иерархии) слой строит свои функции — более сложные и более мощные, которые, в свою очередь, оказываются примитивами для создания еще более мощных функций вышележащего слоя. Строгие правила касаются только взаимодействия между слоями системы, а между модулями внутри слоя связи могут быть произвольными. Отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться к другому модулю своего слоя, либо обратиться за помощью к нижележащему слою через межслойный интерфейс.

Такая организация системы имеет много достоинств. Она существенно упрощает разработку системы, так как позволяет сначала определить «сверху вниз» функции слоев и межслойные интерфейсы, а затем при детальной реализации постепенно наращивать мощность функций слоев, двигаясь «снизу вверх». Кроме того, при модернизации системы можно изменять модули внутри слоя без необходимости производить какие-либо изменения в остальных слоях, если при этих внутренних изменениях межслойные интерфейсы остаются в силе.

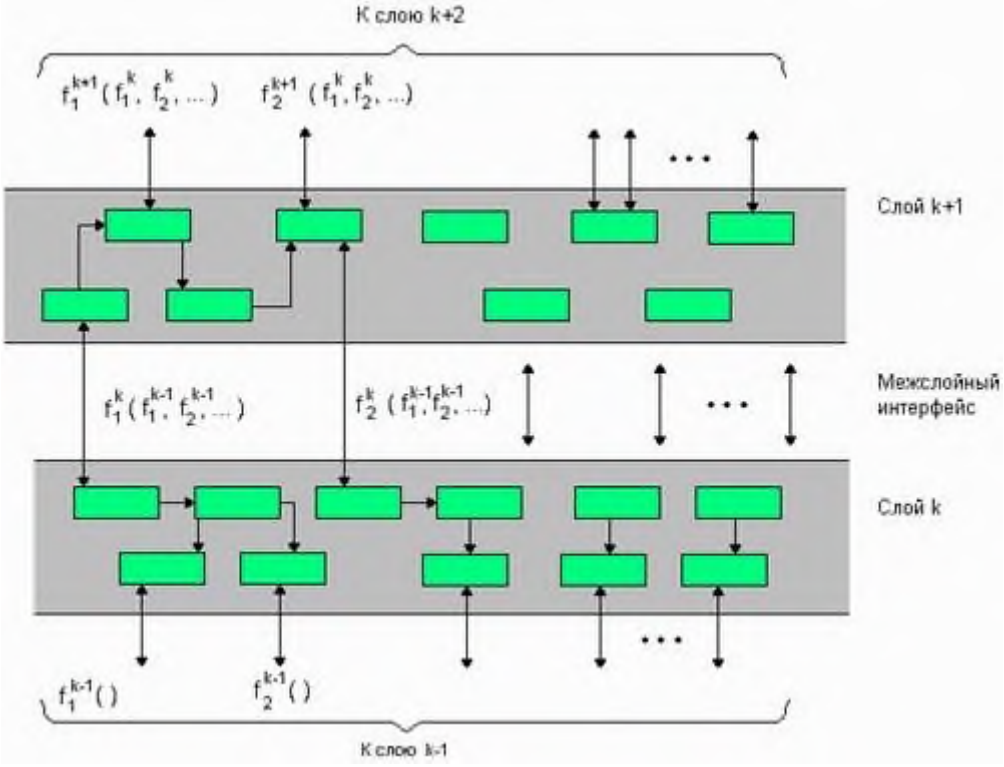

Рис. 3.7. Концепция многослойного взаимодействия

Поскольку ядро представляет собой сложный многофункциональный комплекс, то многослойный подход обычно распространяется и на структуру ядра.

Ядро может состоять из следующих слоев:

Средства аппаратной поддержки ОС. До сих пор об операционной системе говорилось как о комплексе программ, но, вообще говоря, часть функций ОС может выполняться и аппаратными средствами. Поэтому иногда можно встретить определение операционной системы как совокупности программных и аппаратных средств, что и отражено на рис. 3.8. К операционной системе относят, естественно, не все аппаратные устройства компьютера, а только средства аппаратной поддержки ОС, то есть те, которые прямо участвуют в организации вычислительных процессов: средства поддержки привилегированного режима,

систему прерываний, средства переключения контекстов процессов, средства защиты областей памяти и т. п.

Машинно-зависимые компоненты ОС. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС. Примером экранирующего слоя может служить слой HAL операционной системы Windows NT.

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 не принимают решений о распределении ресурсов — они только отрабатывают принятые «наверху» решения, что и дает повод называть их исполнительными механизмами для модулей верхних слоев. Например, решение о том, что в данный момент нужно прервать выполнение текущего процесса А и начать выполнение процесса В, 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 В.

Менеджеры ресурсов. Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на данном слое работают менеджеры (называемые также диспетчерами) процессов, ввода-вывода, файловой системы и оперативной памяти. Разбиение на менеджеры может быть и несколько иным, например менеджер файловой системы иногда объединяют с менеджером ввода-вывода, а функции управления доступом пользователей к системе в целом и ее отдельным объектам поручают отдельному менеджеру безопасности. Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений. Например, менеджер виртуальной памяти управляет перемещением страниц из оперативной памяти на диск и обратно. Менеджер должен отслеживать интенсивность обращений к страницам, время пребывания их в памяти, состояния процессов, использующих данные, и многие другие параметры, на основании которых он время от времени принимает решения о том, какие страницы необходимо выгрузить и какие — загрузить. Для исполнения принятых решений менеджер обращается к нижележащему слою базовых механизмов с запросами о загрузке (выгрузке) конкретных страниц. Внутри слоя менеджеров существуют тесные взаимные связи, отражающие тот факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам — процессору, области памяти, возможно, к определенному файлу или устройству ввода-вывода. Например, при создании процесса менеджер процессов обращается к менеджеру памяти, который должен выделить процессу определенную область памяти для его кодов и данных.

Интерфейс системных вызовов. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными

утилитами, образуя прикладной программный интерфейс операционной системы. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Например, в операционной системе UNIX с помощью системного вызова fd = open("/doc/a.txt", 0_RDONLY) приложение открывает файл a.txt, хранящийся в каталоге /doc, а с помощью системного вызова read(fd, buffer, count) читает из этого файла в область своего адресного пространства, имеющую имя buffer, некоторое количество байт. Для осуществления таких комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов, причем для выполнения одного системного вызова может понадобиться несколько таких обращений.

Рис. 3.8. Многослойная структура ядра ОС

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними может быть и иным. В системах, предназначенных для аппаратных платформ одного типа, например ОС NetWare, слой машинно-зависимых модулей обычно не выделяется, сливаясь со слоем базовых механизмов и, частично, со слоем менеджеров ресурсов. Не всегда оформляются в отдельный слой базовые механизмы — в этом случае менеджеры ресурсов не только планируют использование ресурсов, но и самостоятельно реализуют свои планы.

Возможна и противоположная картина, когда ядро состоит из большего количества слоев. Например, менеджеры ресурсов, составляя определенный слой ядра, в свою очередь, могут обладать многослойной структурой. Прежде всего это относится к менеджеру ввода-вывода, нижний слой которого составляют драйверы устройств, например драйвер жесткого диска или драйвер сетевого адаптера, а верхние слои — драйверы файловых систем или протоколов сетевых служб, имеющие дело с логической организацией информации.

Способ взаимодействия слоев в реальной ОС также может отклоняться от описанной выше схемы. Для ускорения работы ядра в некоторых случаях происходит непосредственное обращение с верхнего слоя к функциям нижних слоев, минуя промежуточные. Типичным примером такого «неправильного» взаимодействия является начальная стадия обработки системного вызова. На

многих аппаратных платформах для реализации системного вызова используется инструкция программного прерывания. Этим приложение фактически вызывает модуль первичной обработки прерываний, который находится в слое базовых механизмов, а уже этот модуль вызывает нужную функцию из слоя системных вызовов. Сами функции системных вызовов также иногда нарушают субординацию иерархических слоев, обращаясь прямо к базовым механизмам ядра.

Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению работы ядра за счет дополнительных накладных расходов на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и логичность системы. Обычно операционные системы, прошедшие долгий путь эволюционного развития, например многие версии UNIX, имеют неупорядоченное ядро с небольшим числом четко выделенных слоев, а у сравнительно «молодых» операционных систем, таких как Windows NT, ядро разделено на большее число слоев и их взаимодействие формализовано в гораздо большей степени.

Аппаратная зависимость и переносимость ОС

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

Типовые средства аппаратной поддержки ОС

Четкой границы между программной и аппаратной реализацией функций ОС не существует — решение о том, какие функции ОС будут выполняться программно, а какие аппаратно, принимается разработчиками аппаратного и программного обеспечения компьютера. Тем не менее практически все современные аппаратные платформы имеют некоторый типичный набор средств аппаратной поддержки ОС, в который входят следующие компоненты:

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

Средства поддержки привилегированного режима обычно основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора. Этот регистр содержит некоторые признаки, определяющие режимы работы процессора, в том числе и признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения слова состояния

машины в результате прерывания или выполнения привилегированной команды. Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро-пользователь) или четыре (например, ядро- супервизор- выполнение- пользователь у платформы VAX или 0-1-2-3 у процессоров Intel x86/Pentium). В обязанности средств поддержки привилегированного режима входит выполнение проверки допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности.

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

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

Переключение контекста выполняется по определенным командам процессора, например по команде перехода на новую задачу. Такая команда вызывает автоматическую загрузку данных из сохраненного контекста в регистры процессора, после чего процесс продолжается с прерванного ранее места.

Система прерываний позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, быстро переходить с одной программы на другую. Механизм прерываний нужен для того, чтобы оповестить процессор о возникновении в вычислительной системе некоторого непредсказуемого события или события, которое не синхронизировано с циклом работы процессора. Примерами таких событий могут служить завершение операции ввода-вывода внешним устройством (например, запись блока данных контроллером диска), некорректное завершение арифметической операции (например, переполнение регистра), истечение интервала астрономического времени. При возникновении условий прерывания его источник (контроллер внешнего устройства, таймер, арифметический блок процессора и т. п.) выставляет определенный электрический сигнал. Этот сигнал прерывает выполнение процессором последовательности команд, задаваемой исполняемым кодом, и вызывает автоматический переход на заранее определенную процедуру, называемую процедурой обработки прерываний. В большинстве моделей процессоров отрабатываемый аппаратурой переход на

процедуру обработки прерываний сопровождается заменой слова состояния машины (или даже всего контекста процесса), что позволяет одновременно с переходом по нужному адресу выполнить переход в привилегированный режим. После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода.

Прерывания играют важнейшую роль в работе любой операционной системы, являясь ее движущей силой. Действительно, большая часть действий ОС инициируется прерываниями различного типа. Даже системные вызовы от приложений выполняются на многих аппаратных платформах с помощью специальной инструкции прерывания, вызывающей переход к выполнению соответствующих процедур ядра (например, инструкция int в процессорах Intel или SVC в мэйнфреймах IBM).

Системный таймер, часто реализуемый в виде быстродействующего регистра-счетчика, необходим операционной системе для выдержки интервалов времени. Для этого в регистр таймера программно загружается значение требуемого интервала в условных единицах, из которого затем автоматически с определенной частотой начинает вычитаться по единице. Частота «тиков» таймера, как правило, тесно связана с частотой тактового генератора процессора. (Не следует путать таймер ни с тактовым генератором, который вырабатывает сигналы, синхронизирующие все операции в компьютере, ни с системными часами — работающей на батареях электронной схеме, — которые ведут независимый отсчет времени и календарной даты.) При достижении нулевого значения счетчика таймер инициирует прерывание, которое обрабатывается процедурой операционной системы. Прерывания от системного таймера используются ОС в первую очередь для слежения за тем, как отдельные процессы расходуют время процессора. Например, в системе разделения времени при обработке очередного прерывания от таймера планировщик процессов может принудительно передать управление другому процессу, если данный процесс исчерпал выделенный ему квант времени.

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

Машинно-зависимые компоненты ОС

Одна и та же операционная система не может без каких-либо изменений устанавливаться на компьютерах, отличающихся типом процессора или/и способом организации всей аппаратуры. В модулях ядра ОС не могут не отразиться такие особенности аппаратной платформы, как количество типов прерываний и формат таблицы ссылок на процедуры обработки прерываний, состав регистров общего назначения и системных регистров, состояние которых

нужно сохранять в контексте процесса, особенности подключения внешних устройств и многие другие.

Однако опыт разработки операционных систем показывает: ядро можно спроектировать таким образом, что только часть модулей будут машинно-зависимыми, а остальные не будут зависеть от особенностей аппаратной платформы. В хорошо структурированном ядре машинно-зависимые модули локализованы и образуют программный слой, естественно примыкающий к слою аппаратуры, как это и показано на рис. 3.8. Такая локализация машинно-зависимых модулей существенно упрощает перенос операционной системы на другую аппаратную платформу.

Объем машинно-зависимых компонентов ОС зависит от того, насколько велики отличия в аппаратных платформах, для которых разрабатывается ОС. Например, ОС, построенная на 32-битовых адресах, для переноса на машину с 16-битовыми адресами должна быть практически переписана заново. Одно из наиболее очевидных отличий — несовпадение системы команд процессоров — преодолевается достаточно просто. Операционная система программируется на языке высокого уровня, а затем соответствующим компилятором вырабатывается код для конкретного типа процессора. Однако во многих случаях различия в организации аппаратуры компьютера лежат гораздо глубже и преодолеть их таким образом не удается. Например, однопроцессорный и двухпроцессорный компьютеры требуют применения в ОС совершенно разных алгоритмов распределения процессорного времени. Аналогично отсутствие аппаратной поддержки виртуальной памяти приводит к принципиальному различию в реализации подсистемы управления памятью. В таких случаях не обойтись без внесения в код операционной системы специфики аппаратной платформы, для которой эта ОС предназначается.

Для уменьшения количества машинно-зависимых модулей производители операционных систем обычно ограничивают универсальность машинно-независимых модулей. Это означает, что их независимость носит условный характер и распространяется только на несколько типов процессоров и созданных на основе этих процессоров аппаратных платформ. По этому пути пошли, например, разработчики ОС Windows NT, ограничив количество типов процессоров для своей системы четырьмя и поставляя различные варианты кодов ядра для однопроцессорных и многопроцессорных компьютеров.

Особое место среди модулей ядра занимают низкоуровневые драйверы внешних устройств. С одной стороны эти драйверы, как и высокоуровневые драйверы, входят в состав менеджера ввода-вывода, то есть принадлежат слою ядра, занимающему достаточно высокое место в иерархии слоев. С другой стороны, низкоуровневые драйверы отражают все особенности управляемых внешних устройств, поэтому их можно отнести и к слою машинно-зависимых модулей. Такая двойственность низкоуровневых драйверов еще раз подтверждает схематичность модели ядра со строгой иерархией слоев.

Для компьютеров на основе процессоров Intel x86/Pentium разработка экранирующего машинно-зависимого слоя ОС несколько упрощается за счет встроенной в постоянную память компьютера базовой системы ввода-вывода — BIOS. BIOS содержит драйверы для всех устройств, входящих в базовую конфигурацию компьютера: жестких и гибких дисков, клавиатуры, дисплея и т. д.

Эти драйверы выполняют весьма примитивные операции с управляемыми устройствами, например чтение группы секторов данных с определенной дорожки диска, но за счет этих операций экранируются различия аппаратных платформ персональных компьютеров и серверов на процессорах Intel разных производителей. Разработчики операционной системы могут пользоваться слоем драйверов BIOS как частью машинно-зависимого слоя ОС, а могут и заменить все или часть драйверов BIOS компонентами ОС.

Переносимость операционной системы

Если код операционной системы может быть сравнительно легко перенесен с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа на аппаратную платформу другого типа, то такую ОС называют переносимой (portable), или мобильной.

Хотя ОС часто описываются либо как переносимые, либо как непереносимые, мобильность — это не бинарное состояние, а понятие степени. Вопрос не в том, может ли быть система перенесена, а в том, насколько легко можно это сделать. Для того чтобы обеспечить свойство мобильности ОС, разработчики должны следовать следующим правилам.

Большая часть кода должна быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить систему. Такими языками являются стандартизованные языки высокого уровня. Большинство переносимых ОС написано на языке С, который имеет много особенностей, полезных для разработки кодов операционной системы, и компиляторы которого широко доступны. Программа, написанная на ассемблере, является переносимой только в тех случаях, когда перенос операционной системы планируется на компьютер, обладающий той же системой команд. В остальных случаях ассемблер используется только для тех непереносимых частей системы, которые должны непосредственно взаимодействовать с аппаратурой (например, обработчик прерываний), или для частей, которые требуют максимальной скорости (например, целочисленная арифметика повышенной точности).

Объем машинно-зависимых частей кода, которые непосредственно взаимодействуют с аппаратными средствами, должен быть по возможности минимизирован. Так, например, следует всячески избегать прямого манипулирования регистрами и другими аппаратными средствами процессора. Для уменьшения аппаратной зависимости разработчики ОС должны также исключить возможность использования по умолчанию стандартных конфигураций аппаратуры или их характеристик. Аппаратно-зависимые параметры можно «спрятать» в программно- задаваемые данные абстрактного типа. Для осуществления всех необходимых действий по управлению аппаратурой, представленной этими параметрами, должен быть написан набор аппаратно-зависимых функций. Каждый раз, когда какому-либо модулю ОС требуется выполнить некоторое действие, связанное с аппаратурой, он манипулирует абстрактными данными, используя соответствующую функцию из имеющегося набора. Когда ОС переносится, то изменяются только эти данные и функции, которые ими манипулируют. Например, в ОС Windows NT диспетчер прерываний преобразует аппаратные уровни прерываний конкретного типа

процессора в стандартный набор уровней прерываний IRQL, с которыми работают остальные модули операционной системы. Поэтому при переносе Windows NT на новую платформу нужно переписать, в частности, те коды диспетчера прерываний, которые занимаются отображением уровней прерывания на абстрактные уровни IRQL, а те модули ОС, которые пользуются этими абстрактными уровнями, изменений не потребуют.

Аппаратно-зависимый код должен быть надежно изолирован в нескольких модулях, а не быть распределен по всей системе. Изоляции подлежат все части ОС, которые отражают специфику как процессора, так и аппаратной платформы в целом. Низкоуровневые компоненты ОС, имеющие доступ к процессорно - зависимым структурам данных и регистрам, должны быть оформлены в виде компактных модулей, которые могут быть заменены аналогичными модулями для других процессоров. Для снятия платформенной зависимости, возникающей из-за различий между компьютерами разных производителей, построенными на одном и том же процессоре (например, MIPS R4000), должен быть введен хорошо локализованный программный слой машинно-зависимых функций.

В идеале слой машинно-зависимых компонентов ядра полностью экранирует остальную часть ОС от конкретных деталей аппаратной платформы (кэши, контроллеры прерываний ввода-вывода и т. п.), по крайней мере для того набора платформ, который поддерживает данная ОС. В результате происходит подмена реальной аппаратуры некой унифицированной виртуальной машиной, одинаковой для всех вариантов аппаратной платформы. Все слои операционной системы, которые лежат выше слоя машинно-зависимых компонентов, могут быть написаны для управления именно этой виртуальной аппаратурой. Таким образом, у разработчиков появляется возможность создавать один вариант машинно-независимой части ОС (включая компоненты ядра, утилиты, системные обрабатывающие программы) для всего набора поддерживаемых платформ (рис. 3.9).

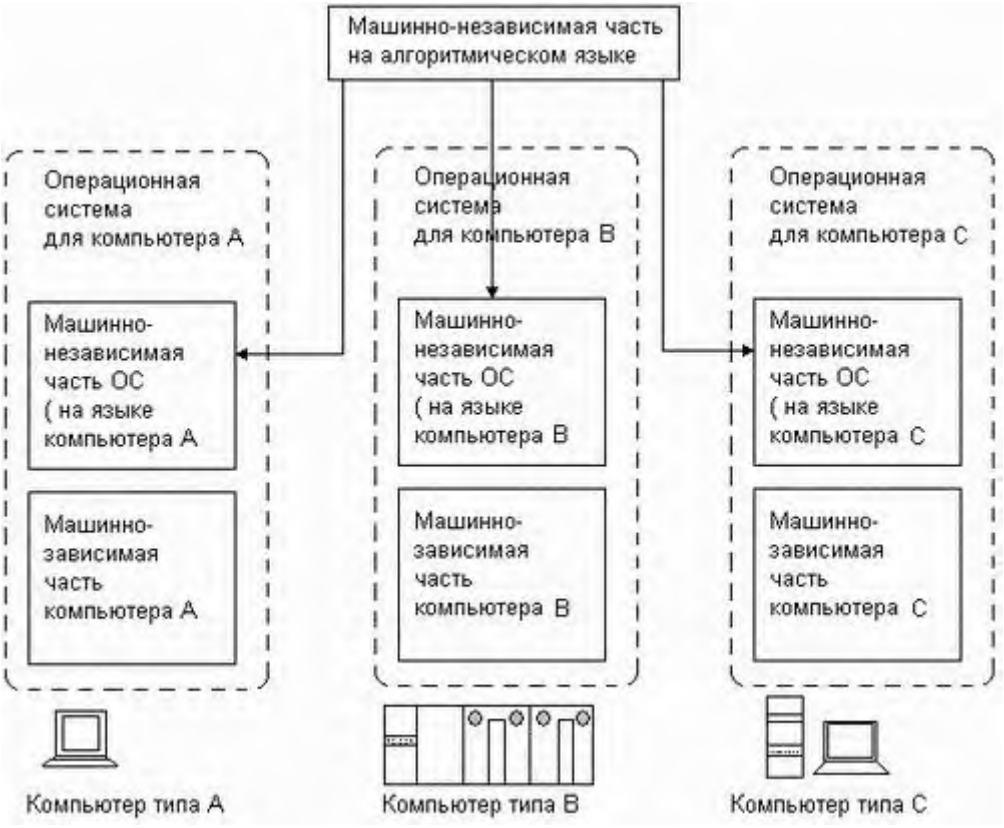

Рис. 3.9. Перенос операционной системы на разные аппаратные платформы

Микроядерная архитектура

Концепция

Микроядерная архитектура является альтернативой классическому способу построения операционной системы. Под классической архитектурой в данном случае понимается рассмотренная выше структурная организация ОС, в соответствии с которой все основные функции операционной системы, составляющие многослойное ядро, выполняются в привилегированном режиме. При этом некоторые вспомогательные функции ОС оформляются в виде приложений и выполняются в пользовательском режиме наряду с обычными пользовательскими программами (становясь системными утилитами или обрабатывающими программами). Каждое приложение пользовательского режима работает в собственном адресном пространстве и защищено тем самым от какого-либо вмешательства других приложений. Код ядра, выполняемый в привилегированном режиме, имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций.

Суть микроядерной архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром (рис. 3.10). Микроядро защищено от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие базовые (но не все!) функции ядра по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений и управлению устройствами ввода-вывода, связанные с загрузкой или чтением регистров устройств. Набор функций

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

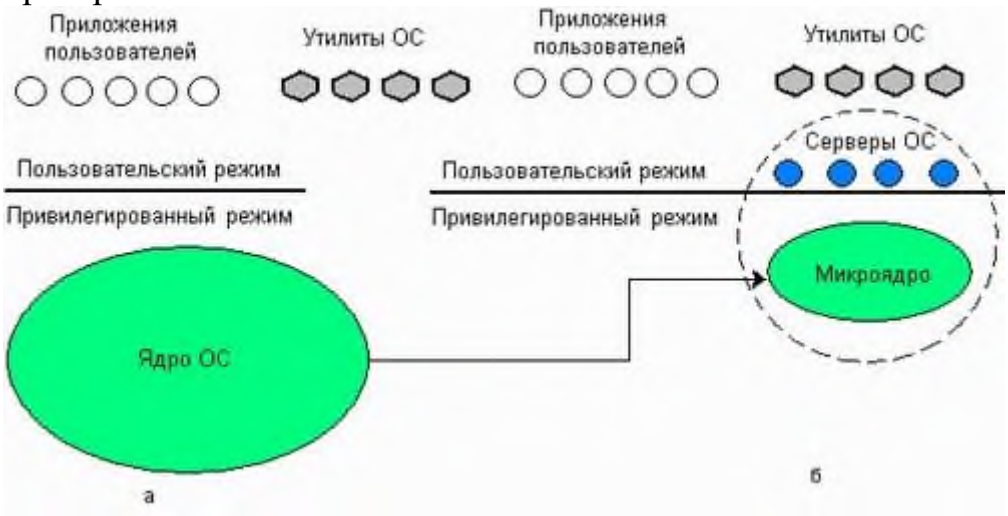

Рис. 3.10. Перенос основного объема функций ядра в пользовательское пространство

Все остальные более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме. 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 «периферийными» модулями, работающими в пользовательском режиме.

Работающие в пользовательском режиме менеджеры ресурсов имеют принципиальные отличия от традиционных утилит и обрабатывающих программ операционной системы, хотя при микроядерной архитектуре все эти программные компоненты также оформлены в виде приложений. Утилиты и обрабатывающие программы вызываются в основном пользователями. Ситуации, когда одному приложению требуется выполнение функции (процедуры) другого приложения, возникают крайне редко. Поэтому в операционных системах с классической архитектурой отсутствует механизм, с помощью которого одно приложение могло бы вызвать функции другого.

Совсем другая ситуация возникает, когда в форме приложения оформляется часть операционной системы. По определению, основным назначением такого приложения является обслуживание запросов других приложений, например создание процесса, выделение памяти, проверка прав доступа к ресурсу и т. д. Именно поэтому менеджеры ресурсов, вынесенные в пользовательский режим, называются серверами ОС, то есть модулями, основным назначением которых является обслуживание запросов локальных приложений и других модулей ОС. Очевидно, что для реализации микроядерной архитектуры необходимым условием является наличие в операционной системе 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. 3.11). Клиент, которым может быть

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

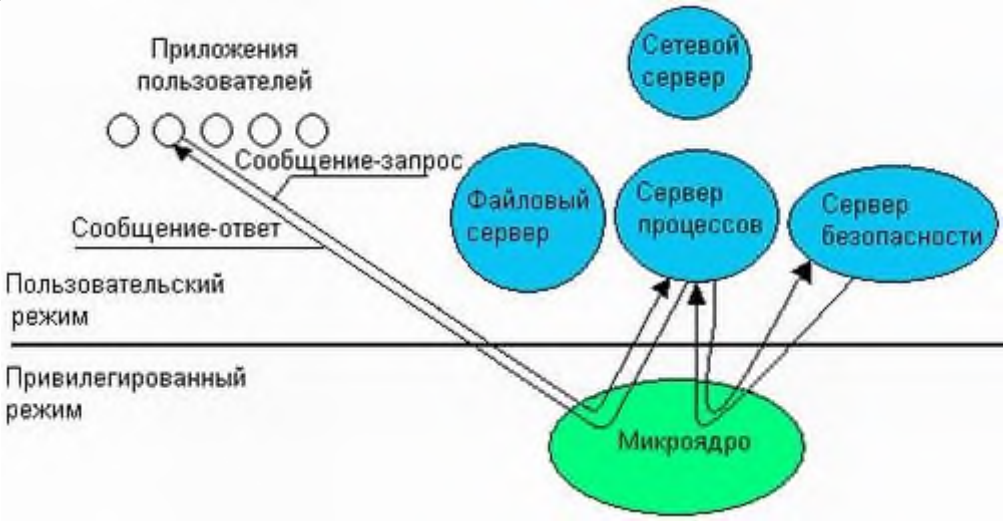

Рис. 3.11. Реализация системного вызова в микроядерной архитектуре

Преимущества и недостатки микроядерной архитектуры

Операционные системы, 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, и это является основным недостатком микроядерной архитектуры.

Высокая степень переносимости обусловлена тем, что весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.

Расширяемость присуща микроядерной ОС в очень высокой степени. В традиционных системах даже при наличии многослойной структуры нелегко удалить один слой и поменять его на другой по причине множественности и размытости интерфейсов между слоями. Добавление новых функций и изменение существующих требует хорошего знания операционной системы и больших затрат времени. В то же время ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС. Добавление новой подсистемы требует разработки нового приложения, что никак

не затрагивает целостность микроядра. Микроядерная структура позволяет не только добавлять, но и сокращать число компонентов операционной системы, 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. Обычно традиционные операционные системы позволяют динамически добавлять в ядро или удалять из ядра только драйверы внешних устройств — ввиду частых изменений в конфигурации подключенных к компьютеру внешних устройств подсистема ввода-вывода ядра допускает загрузку и выгрузку драйверов «на ходу», но для этого она разрабатывается особым образом (например, среда STREAMS в UNIX или менеджер ввода-вывода в Windows NT). 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 не нужные 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 операционной системы, что не наблюдается в традиционной ОС, где все модули ядра могут влиять друг на друга. И если отдельный сервер терпит крах, то он может быть перезапущен без останова или повреждения остальных серверов ОС. 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.

Модель с микроядром хорошо подходит для поддержки распределенных вычислений, так как использует механизмы, аналогичные сетевым: взаимодействие клиентов и серверов путем обмена сообщениями. Серверы микроядерной ОС могут работать как на одном, так и на разных компьютерах. В этом случае при получении сообщения от приложения микроядро может обработать его самостоятельно и передать локальному серверу или же переслать по сети микроядру, работающему на другом компьютере. Переход к распределенной обработке требует минимальных изменений в работе операционной системы — просто локальный транспорт заменяется на сетевой.

Производительность. При классической организации ОС (рис. 3.12, а) выполнение системного вызова сопровождается двумя переключениями режимов, а при микроядерной организации (рис. 3.12, 6) — четырьмя. Таким образом, операционная система 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. 3.12. Смена режимов при выполнении системного вызова

Серьезность этого недостатка хорошо иллюстрирует история развития Windows NT. В версиях 3.1 и 3.5 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 схемой. Однако очень скоро разработчики Windows NT поняли, что такой механизм обращений к часто используемым функциям графического интерфейса существенно замедляет работу приложений и делает данную операционную систему уязвимой в условиях острой конкуренции. В результате в версию Windows NT 4.0 были внесены существенные изменения — все перечисленные выше модули были перенесены в ядро, что отдалило эту ОС от идеальной микроядерной архитектуры, но зато резко повысило ее производительность.

Этот пример иллюстрирует главную проблему, с которой сталкиваются разработчики операционной системы, решившие применить микроядерный подход, — что включать в микроядро, а что выносить в пользовательское пространство. В идеальном случае микроядро может состоять только из средств передачи сообщений, средств взаимодействия с аппаратурой, в том числе средств доступа к механизмам привилегированной защиты. Однако многие разработчики не всегда жестко придерживаются принципа минимизации функций ядра, часто жертвуя этим ради повышения производительности. В результате реализации ОС образуют некоторый спектр, на одном краю которого находятся системы с минимально возможным микроядром, а на другом — системы, подобные Windows NT, в которых микроядро выполняет достаточно большой объем функций.

Контрольные вопросы

1. Какие из приведенных ниже терминов являются синонимами?

привилегированный режим;
защищенный режим;
режим супервизора;
пользовательский режим;
реальный режим;
режим ядра

2. Можно ли, анализируя двоичный код программы, сделать вывод о невозможности ее выполнения в пользовательском режиме?

3. В чем состоят отличия в работе процессора в привилегированном и пользовательском режимах? ;

4. В идеале микроядерная архитектура ОС требует размещения в микроядре только тех компонентов ОС, которые не могут выполняться в пользовательском режиме. Что заставляет разработчиков операционных систем отходить от этого принципа и расширять ядро за счет перенесения в него функций, которые могли бы быть реализованы в виде процессов-серверов?

5. Какие этапы включает разработка варианта мобильной ОС для новой аппаратной платформы?

6. Опишите порядок взаимодействия приложений с ОС, имеющей микроядерную архитектуру.

7. Какими этапами отличается выполнение системного вызова в микроядерной ОС и ОС с монолитным ядром?

8. Может ли программа, эмулируемая на «чужом» процессоре, выполняться быстрее, чем на «родном»?