Раздел 1 Вычислительная система.

Тема 1.2 Программное обеспечение.

Лекция №4 Архитектурные особенности модели микропроцессорной системы. Монолитные, микроядерные, смешанные системы. Характеристики.

1. ***Монолитные системы*** (монолитное ядро, *monolithic kernel*). Монолитная система – это схема операционной системы, при которой все ее компоненты являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путем непосредственного вызова процедур.

Для построения монолитной системы необходимо скомпилировать все отдельные процедуры, а затем связать их в единый объектный файл с помощью компоновщика. В связи с тем, что монолитная система представляет собой единую программу, то перекомпиляция является единственным способом добавления новых компонентов и исключения неиспользуемых. Присутствие лишних компонентов нежелательно, так как система полностью располагается в оперативной памяти, кроме того, исключение ненужных компонентов повышает ее надежность.

При обращении к системным вызовам, поддерживаемым операционной системой, параметры помещаются в строго определенные места – *регистры* или *стек*, после чего выполняется специальная команда прерывания, известная как *вызов ядраа*. Эта команда переключает машину из режима пользователя в режим ядра и передает управление операционной системе. Такая организация операционной системы предполагает следующую структуру (рис. 2.6):

– главная программа, вызывающая требуемую служебную процедуру;
– набор служебных процедур, выполняющих системные вызовы;
– набор утилит, обслуживающих служебные процедуры.

В этой модели для каждого системного вызова имеется одна служебная процедура. Утилиты выполняют функции, которые нужны нескольким служебным процедурам. Монолитная структура – старейший способ организации операционных систем.

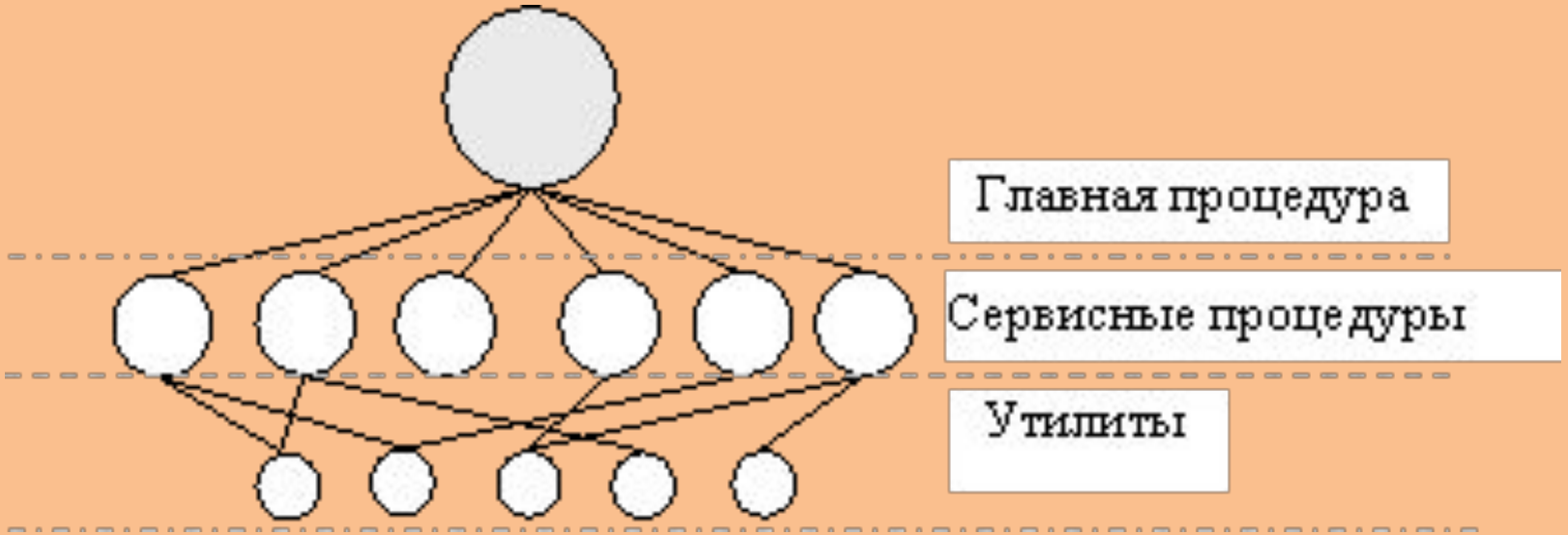

Рис. 2.6. Простая модель монолитной системы

2. ***Многоуровневые системы***. Обобщением подхода является организация операционных систем в виде иерархии уровней. Первой многоуровневой системой была система *THE*, созданная в *Technische Hogeschool Eindhoven* (Нидерланды) Э. Дейкстроем (E.W. *Dijkstra*) и его студентами в 1968 г. Она была простой пакетной системой для голландского компьютера *Electrologica X8,* память которого состояла из 32 Кб 27-разрядных слов. Система включала 6 уровней (рис. 2.7):

– уровень 0 занимался распределением времени процессора, переключая процессы при возникновении прерывания или при срабатывании таймера, т. е. обеспечивал базовую многозадачность процессора;

– уровень 1 управлял памятью, он выделял процессам пространство в оперативной памяти и на магнитном барабане объемом 512 Кб слов для тех частей процессов (страниц), которые не помещались в оперативной памяти;

– уровень 2 управлял связью между консолью оператора и процессами, процессы, расположенные выше этого уровня, имели свою собственную консоль оператора;

– уровень 3 управлял устройствами ввода-вывода и буферизовал потоки информации к ним и от них, поэтому любой процесс выше уровня 3, вместо того чтобы работать с конкретными устройствами, мог обращаться к абстрактным устройствам ввода-вывода, обладающим удобными для пользователя характеристиками;

– уровень 4 был предназначен для работы пользовательских программ, которым не нужно было заботиться ни о процессах, ни о памяти, ни о консоли, ни об управлении устройствами ввода-вывода;

– уровень 5 предназначался для процесса системного оператора

Уровни	**Функции**
5	Оператор
4	Программы пользователя
3	Управление вводом-выводом
2	Связь оператор-процесс
1	Управление памятью и барабаном
0	Распределение процессора и многозадачность

Дальнейшее обобщение многоуровневой концепции было сделано в операционной системе *MULTICS*, где уровни представляли серию концентрических колец, причем внутренние кольца являлись более привилегированными, чем внешние. Когда процедура внешнего кольца «хотела» вызвать процедуру кольца, лежащего внутри, она должна была выполнить эквивалент системного вызова, т. е. команду *TRAP*, параметры которой тщательно проверяются перед вызовом. Хотя операционная система *MULTICS* являлась частью адресного пространства каждого пользовательского процесса, аппаратура обеспечивала защиту данных на уровне сегментов памяти, разрешая или запрещая доступ к индивидуальным процедурам (сегментам памяти) для записи, чтения или выполнения.

В системе *THE* многоуровневая схема представляла собой конструкционное решение, и все части системы были связаны в один объектный файл, а в *MULTICS* механизм разделения колец действовал во время исполнения на аппаратном уровне. Преимущество подхода *MULTICS* заключалось в том, что его можно было расширить и на структуру пользовательских подсистем

3. ***Виртуальные машины***. Большое количество пользователей операционной системы пакетной обработки *OS/360* хотели работать в системе с разделением времени, поэтому программисты в корпорации *IBM* и в других центрах решили написать систему с разделением времени. Официальная операционная система с разделением времени *TSS/360* от *IBM* оказалась громоздкой и медленной. Группа из научного центра *IBM* в Кембридже, штат Массачусетс, разработала новую систему, которую *IBM* приняла как законченный продукт. В настоящее время она используется на оставшихся мэйнфреймах. Эта система, в оригинале называвшаяся *CP/CMS*, а позже переименованная в *VM/370*, была основана на двух разделяемых принципах (рис. 2.8):

– система с разделением времени обеспечивает многозадачность;

– система с разделением времени обеспечивает расширенную машину с более удобным интерфейсом, чем тот, что предоставляется оборудованием напрямую.

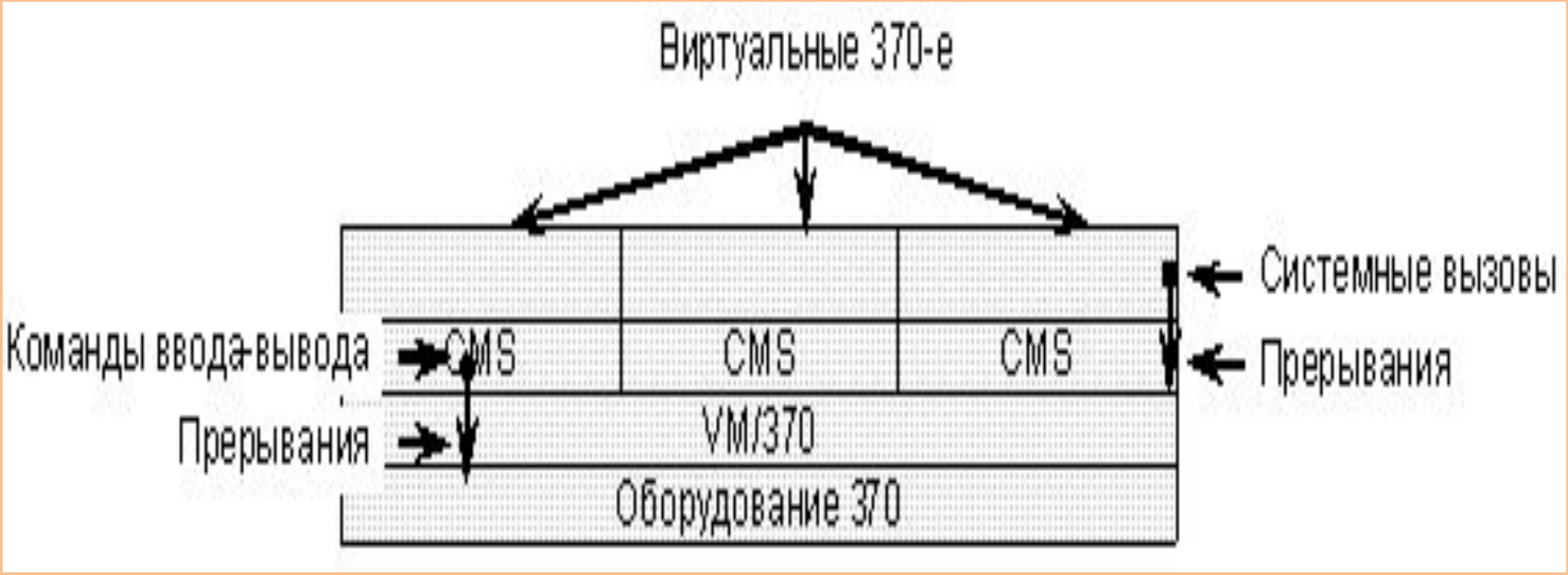

Рис. 2.8. Структура VM/370 с системой CMS

Сердце системы – *монитор виртуальной машины* работает с оборудованием и обеспечивает многозадачность, предоставляя верхнему слою не одну, а несколько виртуальных машин. В отличие от других операционных систем, эти виртуальные машины не являются расширенными, так как они не поддерживают файлы и прочие удобства, а представляют собой точные копии аппаратуры, включая режимы ядра и пользователя, ввод-вывод данных, прерывания и пр., присутствующей на реальном компьютере.

Когда операционная система *CMS* выполняет системный вызов, он прерывает операционную систему на своей собственной виртуальной машине, а не на *VM/370*, как произошло бы, если бы вызов работал на реальной машине. Затем операционная система *CMS* выдает обычные команды ввода-вывода для чтения своего виртуального диска или другие команды, которые могут понадобиться для выполнения вызова. Эти команды перехватываются *VM/370*, которая выполняет их в рамках моделирования реального оборудования. При полном разделении функций многозадачности и предоставления расширенной машины каждая часть может быть намного проще, гибче и удобней для обслуживания.

В настоящее время идея виртуальной машины используется для работы старых программ, написанных для системы *MS-DOS* на *Pentium* (или на других 32-разрядных процессорах *Intel*). При разработке компьютера *Pentium* и его программного обеспечения компании *Intel* и *Microsoft* учитывали потребность в работе старых программ на новом оборудовании. Поэтому корпорация *Intel* создала на процессоре *Pentium* режим виртуального процессора 8086. В этом режиме машина действует как 8086 (которая с точки зрения программного обеспечения идентична 8088), включая 16-разрядную адресацию памяти с ограничением объема памяти в 1 Мб. Такой режим используется системой *Windows* и другими операционными системами для запуска программ *MS-DOS*. Программы запускаются в режиме виртуального процессора 8086. Пока они выполняют обычные команды, они работают напрямую с оборудованием. Но когда программа пытается обратиться по прерыванию к операционной системе, чтобы сделать системный вызов, или пытается напрямую осуществить ввод-вывод данных, происходит прерывание с переключением на монитор виртуальной машины

4. ***Экзоядро***. В этой системе реализован принцип обеспечения каждого пользователя абсолютной копией реального компьютера, но с подмножеством ресурсов. Например, одна виртуальная машина может получить блоки на диске с номерами от 0 до 1023, следующая – от 1024 до 2047 и т. д. На нижнем уровне в режиме ядра работает программа – *экзоядро* (*exokernel*), в задачу которой входит распределение ресурсов для виртуальных машин и проверка их использования (отслеживание попыток машин использовать чужой ресурс). Каждая виртуальная машина на уровне пользователя может работать с собственной операционной системой, как на *VM/370* или на виртуальных машинах *8086*-х для *Pentium*, с той разницей, что каждая машина ограничена набором ресурсов, которые она запросила и которые ей были предоставлены.

Преимущества подхода:

– отделяется многозадачность (в экзоядре) от операционных систем пользователя (в пространстве пользователя) с меньшими затратами, так как для этого экзоядру необходимо только не допускать вмешательства одной виртуальной машины в работу другой;

– можно обойтись без уровня отображения, так как экзоядру нужно только хранить запись о том, какой виртуальной машине выделен данный ресурс (при других подходах виртуальная машина считает, что она использует свой собственный диск с нумерацией блоков от 0 до некоторого максимума, поэтому монитор виртуальной машины должен поддерживать таблицы преобразования адресов на диске).

5. ***Ядро в привилегированном режиме***. Операционная система должна иметь по отношению к приложениям определенные привилегии, иначе некорректно работающее приложение может вмешаться в работу операционной системы и разрушить часть ее кодов. Операционная система должна обладать исключительными полномочиями в распределении ресурсов для приложений в мультипрограммном режиме. Обеспечить привилегии невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум два режима работы –*пользовательский* (*user mode*) и *привилегированный* (режим ядра, *kernel mode* или режим супервизора, *supervisor mode*). Так как ядро выполняет все основные функции операционной системы, то чаще всего именно оно работает в привилегированном режиме (рис. 2.9).

Приложения ставятся в подчиненное положение за счет запрета выполнения в пользовательском режиме некоторых критичных команд, связанных с переключением процессора с задачи на задачу. Выполнение некоторых инструкций в пользовательском режиме запрещается безусловно или только при определенных условиях.

Условия разрешения выполнения критичных инструкций находятся под полным контролем операционной системы. Полный контроль операционной системы над доступом к памяти достигается за счет того, что инструкции конфигурирования механизмов защиты памяти разрешается выполнять только в привилегированном режиме.

Каждое приложение работает в своем адресном пространстве, что позволяет локализовать некорректно работающее приложение таким образом, чтобы его ошибки не оказывали влияния на другие приложения и операционную систему.

Повышение устойчивости операционной системы, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления выполнения системных вызовов (рис. 2.10).

Архитектура операционной системы, основанная на привилегированном ядре и приложениях пользовательского режима, стала классической. Ее используют многие популярные операционные системы: версии *UNIX, VAX VMS, IBM OS/390, OS/2* и *Windows NT* (с определенными модификациями).

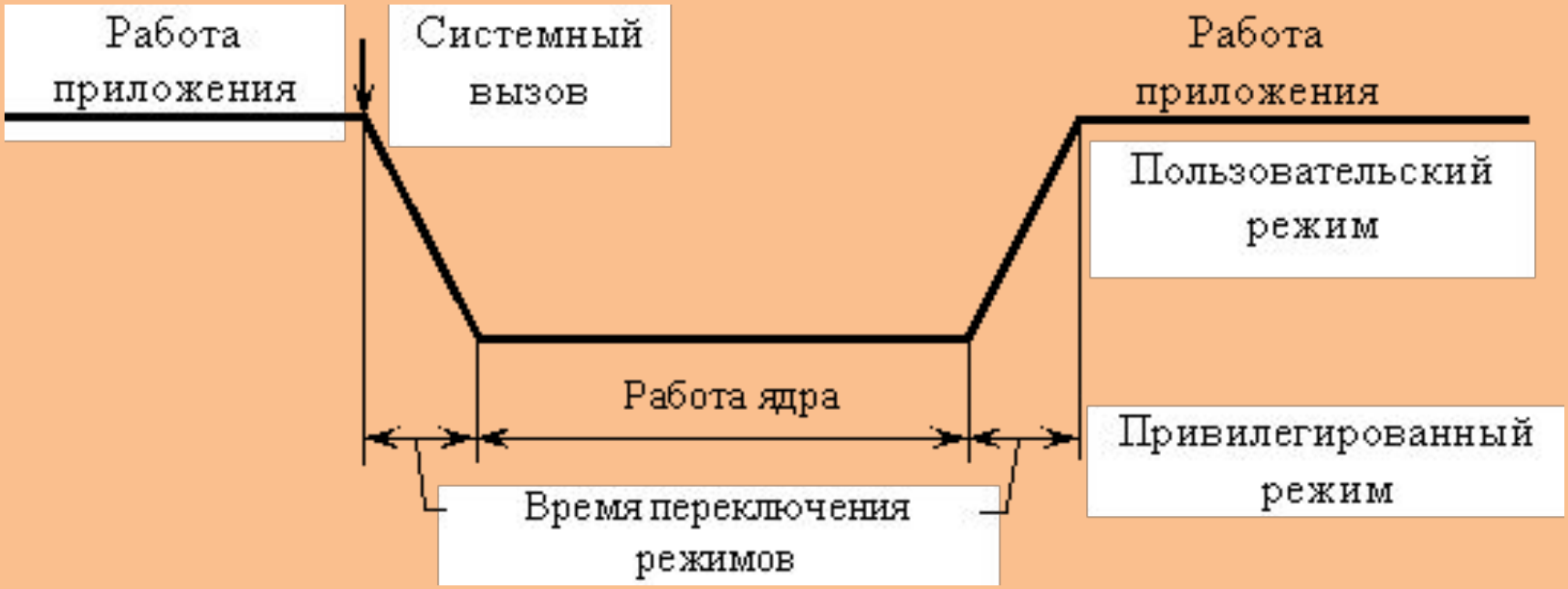

Рис. 2.10. Смена режимов при выполнении системного вызова к привилегированному ядру

В некоторых случаях разработчики операционных систем отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Например, операционная система *NetWare* компании *Novell* использует привилегированный режим процессоров *Intel x86/Pentium*для работы ядра и для работы приложений – загружаемых модулей *NLM* (рис. 2.11).

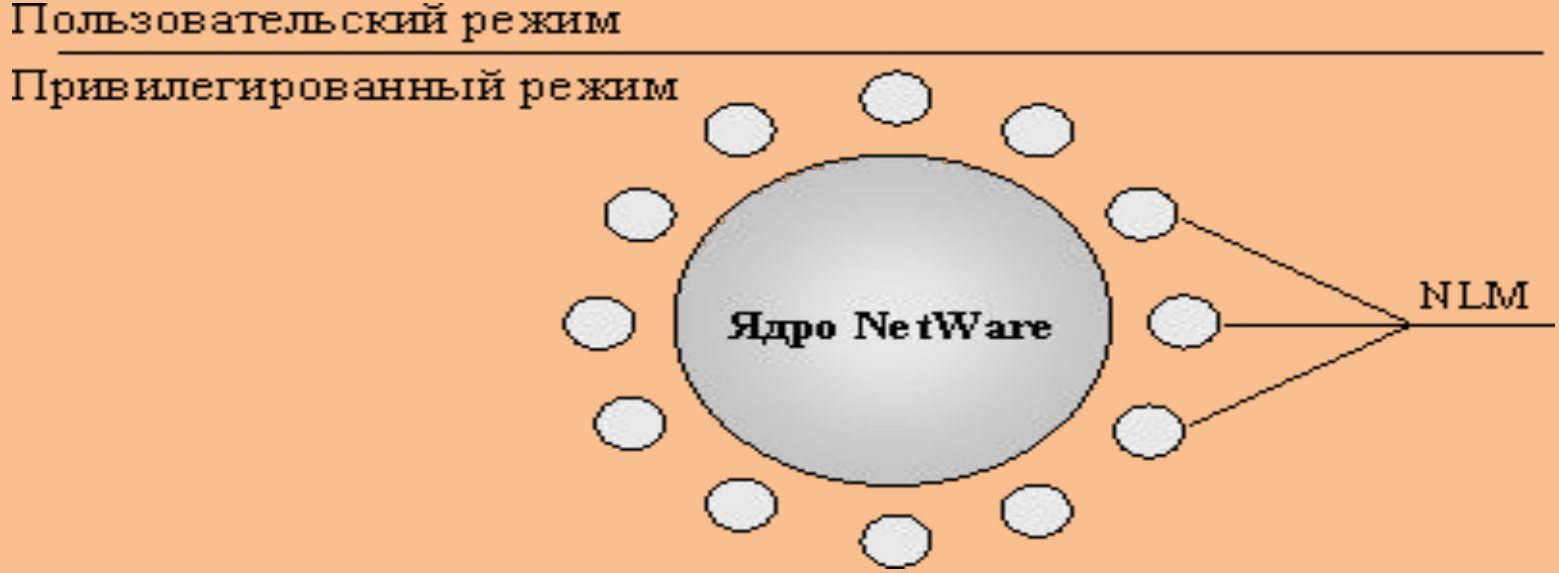

Рис. 2.11. Упрощенная архитектура операционной системы *NetWare*

Разработчиками *MS-DOS* не было использовано появление в более поздних версиях процессоров *Intel* (начиная с 80286-го) возможности работы в привилегированном режиме. Эта операционная система работала на процессорах данного типа в *реальном режиме*, в котором эмулировался процессор8086/88. Реальный режим был реализован только для совместимости поздних моделей процессоров с ранней моделью 8086/88, и альтернативой ему явился защищенный режим работы процессора, в котором стали доступными все особенности процессоров поздних моделей, в том числе и работа на одном из четырех уровней привилегий

6. ***Многослойная структура операционной системы***. Вычислительную систему, работающую под управлением операционной системы на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев: нижний слой образует аппаратура, промежуточный – ядро, верхний – утилиты, обрабатывающие программы и приложения (рис. 2.12). При такой организации приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра.

Рис. 2.12. Трехслойная схема вычислительной системы

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа. В соответствии с этим подходом система состоит из иерархии слоев, каждый слой обслуживает вышележащий, выполняя для него набор функций, которые образуют межслойный интерфейс. На основе функций нижележащего слоя вышележащий слой строит свои более сложные функции.

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».

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

Ядро может состоять из следующих слоев (рис. 2.13).

● *Средства аппаратной поддержки операционной системы* – средства, которые прямо участвуют в организации вычислительных процессов: средства поддержки привилегированного режима, система прерываний, средства переключения контекстов процессов, средства защиты областей памяти и т. п.

● *Машинно-зависимые компоненты операционной системы* – образуются программными модулями, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры, что позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной операционной системой. Пример: экранирующий слой *HAL* операционной системы *Windows NT*.

● *Слой базовых механизмов ядра* – выполняет самые примитивные операции ядра: программное переключение контекстов процессов; диспетчеризацию прерываний; перемещение страниц из памяти на диск и обратно и т. п. Модули слоя не принимают решений о распределении ресурсов, они только отрабатывают принятые «наверху» решения, поэтому их называют исполнительными механизмами для модулей верхних слоев. Например, решение о прерывании выполнения текущего процесса *A* и начале выполнения процесса *B* принимает менеджер процессов вышележащего слоя, а слой базовых механизмов получает директиву о том, что нужно выполнить переключение с контекста текущего процесса на контекст процесса *B*.

● *Менеджеры ресурсов* – слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на данном слое работают менеджеры (диспетчеры) процессов, ввода-вывода, файловой системы и оперативной памяти. Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений. Для исполнения принятых решений менеджер обращается к нижележащему слою базовых механизмов с конкретными запросами. Внутри слоя менеджеров существуют тесные взаимные связи, так как для выполнения процессу может потребоваться доступ одновременно к нескольким ресурсам: процессору, памяти, определенному файлу или устройству ввода-вывода и пр. 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* – самый верхний слой ядра, который 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.

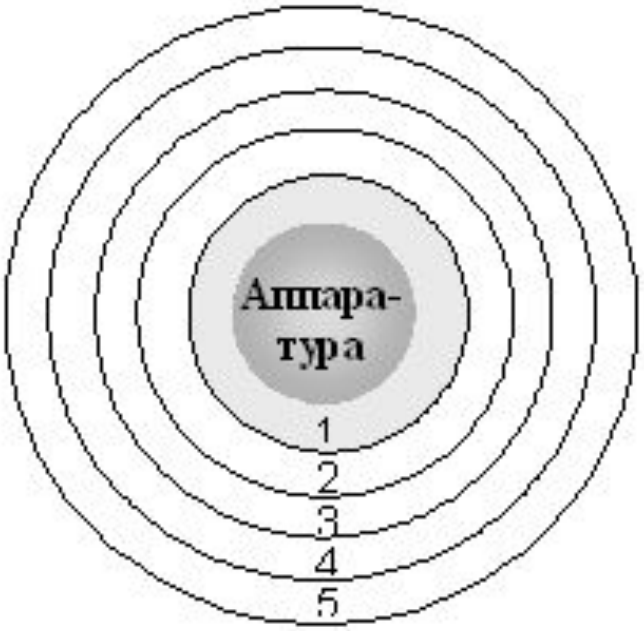

Рис. 2.13. Многослойная структура ядра ОС: *1* – средства аппаратной поддержки ОС; *2* – машинно-зависимые модули; *3* – базовые механизмы ядра; *4* – менеджеры ресурсов; *5* – интерфейс системных вызовов

Приведенное разбиение ядра операционной системы на слои является условным.

При выборе количества слоев учитывается, что увеличение числа слоев приводит к некоторому замедлению работы ядра за счет дополнительного межслойного взаимодействия; 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 операционных систем нового поколения, например *Windows NT*, ядро разделено на большее число слоев, взаимодействие которых формализовано в большей степени

7. ***Микроядерная архитектура***.

Микроядерная архитектура является альтернативой классическому многослойному способу построения операционных систем и состоит в перенесении значительной части системного кода на уровень пользователя и одновременной минимизацией ядра. Подход, когда большинство составляющих операционной системы являются самостоятельными программами, называется *микроядерной архитектурой* (*microkernel architecture*). Взаимодействие между программами операционной системы обеспечивает специальный модуль ядра – *микроядро*. Микроядро работает в привилегированном режиме и обеспечивает взаимодействие между программами, планирование использования процессора, первичную обработку прерываний, операции ввода-вывода и базовое управление памятью. Остальные компоненты системы взаимодействуют друг с другом путем передачи сообщений через микроядро (рис. 2.14).

Микроядро защищено от остальных частей операционной системы и приложений. Более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме (рис. 2.15). Основным назначением такого приложения является обслуживание запросов других приложений, например, создание процесса, выделение памяти, проверка прав доступа к ресурсу и т. п.

Рис. 2.14. Микроядерная архитектура операционной системы

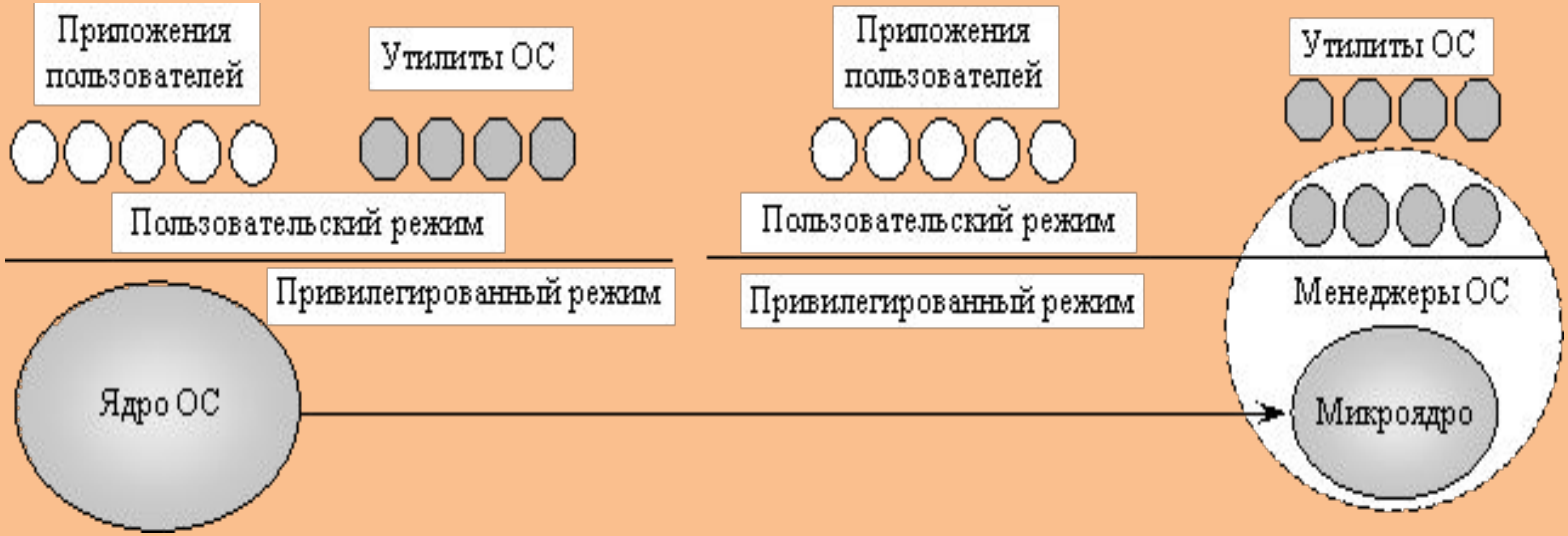

Рис. 2.15. Перенос основного объема функций ядра в пользовательское пространство

8. ***Модель клиент-сервер***.

В развитии современных операционных систем наблюдается тенденция дальнейшего переноса кода в верхние уровни и минимизации ядра, что достигается перекладыванием выполнения большинства задач операционной системы на средства пользовательских процессов. Менеджеры ресурсов, вынесенные в пользовательский режим, называются *серверами операционных систем*. Получая запрос на какую-либо операцию, например, чтение блока файла, пользовательский процесс (обслуживаемый или клиентский) посылает запрос серверному (обслуживающему) процессу, который его обрабатывает и высылает назад ответ (рис. 2.16).

Рис. 2.16. Модель клиент-сервер

Если происходит ошибка на файловом сервере, может разрушиться только служба обработки файловых запросов, что обычно не приводит к остановке всей машины.

Другое преимущество модели клиент-сервер заключается в ее простой адаптации к использованию в распределенных системах (рис. 2.17).

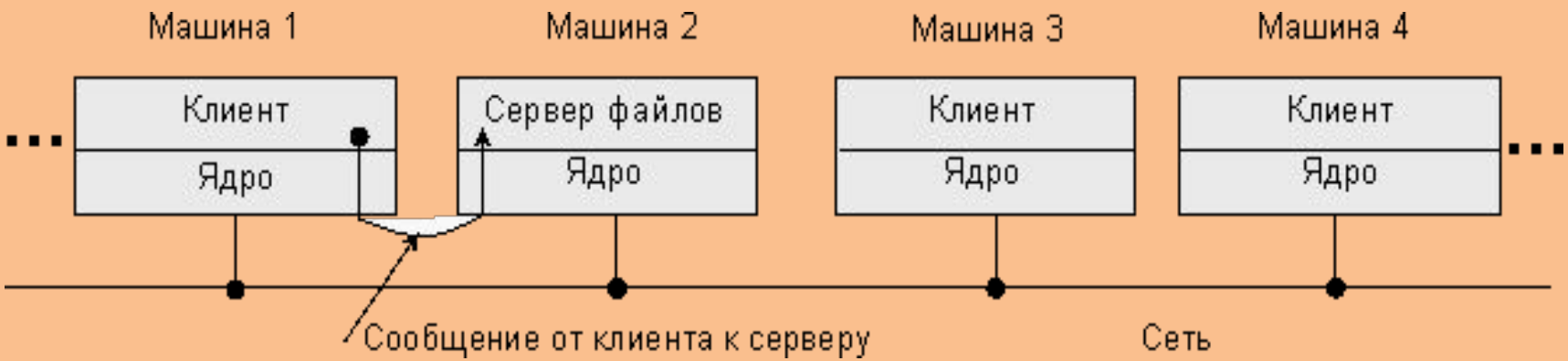

Рис. 2.17. Модель клиент-сервер в распределенной системе

Если клиент общается с сервером, посылая ему сообщения, клиенту не нужно знать, обрабатывается ли его сообщение локально на его собственной машине или оно было послано по сети серверу на удаленной машине.

9. ***Смешанные системы***.

Все рассмотренные подходы к построению операционных систем имеют свои достоинства и недостатки. В большинстве случаев современные операционные системы используют различные комбинации этих подходов. Ниже приведены три примера.

Ядро операционной системы *Linux* представляет собой монолитную систему с элементами микроядерной архитектуры. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра – модулей. В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться любые экспортируемые ядром функции.

Операционные системы *4.4BSD* и *MkLinux* основаны на микроядре *Mach*. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляются монолитным ядром. Данный подход сформировался в результате попыток использования преимущества микроядерной архитектуры, сохраняя по возможности отлаженный код монолитного ядра.

Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре *Windows NT*. Микроядро *Windows NT* велико (более 1 Мб). Компоненты ядра *Windows NT* располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как в микроядерных операционных системах. В то же время все компоненты работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно монолитным операционным системам.