**№ занятия: 3. Практическое занятие № 1. Тема: Архитектура операционных систем.**

**Введение.**

На данном практическом занятии мы изучим основные модели операционных систем: монолитные операционные системы, многоуровневые системы, микроядерные и экзоядерные ОС. Выявим их область применения, недостатки и преимущества. Познакомимся с технологиями виртуализации. Во второй части занятия рассмотрим наиболее общий подход к структуризации операционной системы. Данный подход заключается в разделение всех ее модулей на две группы:

1) ядро – модули, выполняющие основные функции ОС;

2) вспомогательные модули ОС.

**Вопрос 1. Модели операционных систем.**

**Монолитные операционные системы**. Монолитная операционная система является самой распространенной системой. Данная операционная система работает как единая программа в режиме ядра. Для построения исполняемого файла монолитной системы необходимо сначала скомпилировать все отдельные процедуры (или файлы, содержащие процедуры), а затем связать их все вместе, воспользовавшись системным компоновщиком. Здесь, по существу, полностью отсутствует сокрытие деталей реализации – каждая процедура видна любой другой процедуре.

Тем не менее, даже такие монолитные системы могут иметь некоторую структуру. Такая организация предполагает следующую базовую структуру операционной системы:

* основная программа, которая вызывает требуемую служебную процедуру;
* набор служебных процедур, выполняющих системные вызовы;
* набор вспомогательных процедур, содействующих работе служебных процедур.

В этой модели для каждого системного вызова имеется одна ответственная за него служебная процедура, которая его и выполняет. Вспомогательные процедуры выполняют действия, необходимые нескольким служебным процедурам, в частности извлечение данных из пользовательских программ.

Таким образом, процедуры делятся на три уровня, которые показаны на рис. 1.

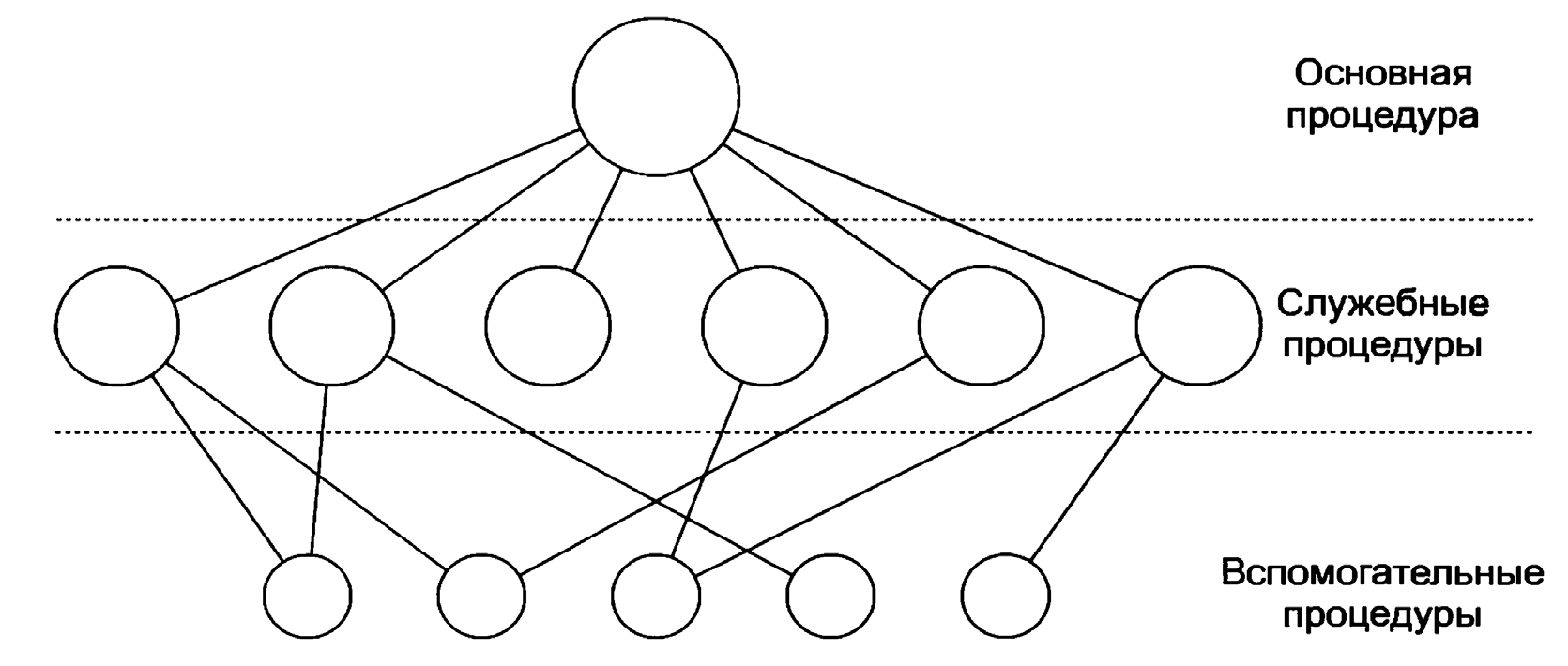
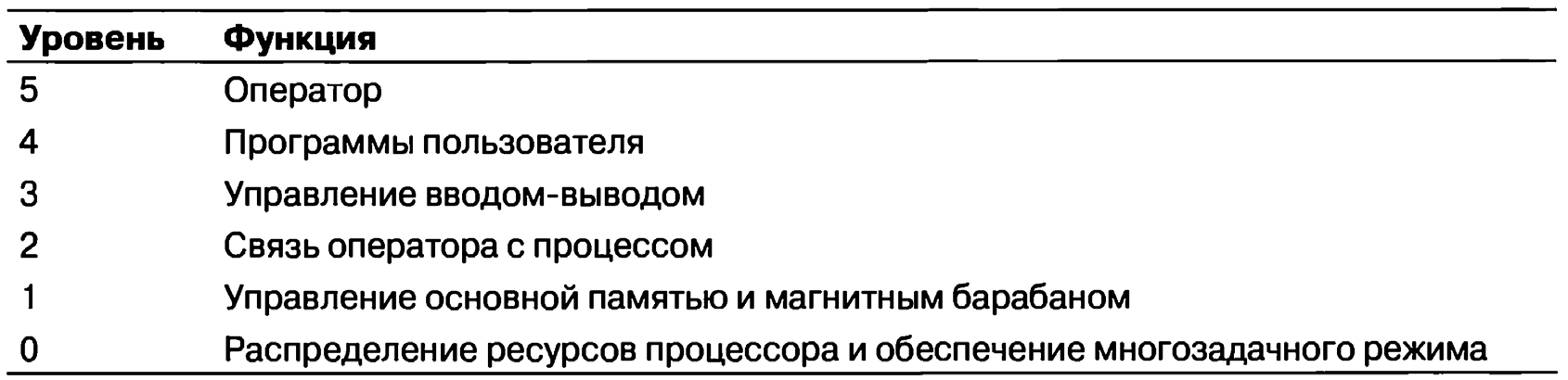


Рис. 1. Простая структурированная модель монолитной системы

**Многоуровневые системы**. Обобщением подхода, показанного на рис. 1, является организация ОС в виде иерархии уровней, каждый из которых является надстройкой над нижележащим уровнем. Первой системой, построенной таким образом, была система *THE*, созданная в *Technische Hogeschool Eindhoven* в Голландии *Э. Дейкстрой* и его студентами в 1968 году. Система *THE* была простой пакетной системой для голландского компьютера *Electrologica X8*, имевшего память 32 Кбайт 27-разрядных слов.

Рис. 2. Система THE.

Как показано в табл. 1, у системы было шесть уровней. Уровень 0 занимался распределением ресурса процессора (процессорного времени), переключением между процессами при возникновении прерываний или истечении времени таймера. Над уровнем 0 система состояла из последовательных процессов, каждый из которых мог быть запрограммирован без учета того, что несколько процессов были запущены на одном процессоре. Иными словами, уровень 0 обеспечивал основу многозадачности центрального процессора.

*Таблица 1. Структура операционной системы THE.*

Уровень 1 управлял памятью. Он выделял процессам пространство в основной памяти и на магнитном барабане емкостью 512 К слов, который использовался для хранения частей процесса (страниц), не умещавшихся в оперативной памяти. На уровнях выше первого процессы не должны были беспокоиться о том, где именно они находятся, в памяти или на барабане; доставкой страниц в память по мере надобности занималось программное обеспечение уровня 1.

Уровень 2 управлял связью каждого процесса с консолью оператора (то есть с пользователем). Над этим уровнем каждый процесс фактически имел свою собственную консоль оператора.

Уровень 3 управлял устройствами ввода-вывода и буферизацией информационных потоков в обоих направлениях. Над третьим уровнем каждый процесс мог работать с абстрактными устройствами ввода-вывода, имеющими определенные свойства.

На уровне 4 работали пользовательские программы, которым не надо было заботиться о процессах, памяти, консоли или управлении вводом-выводом.

Процесс системного оператора размещался на уровне 5.

**Микроядра**. При использовании многоуровневого подхода разработчикам необходимо выбрать, где провести границу между режимами ядра и пользователя. Традиционно все уровни входили в ядро, но это необязательно. Существуют очень весомые аргументы в пользу того, чтобы в режиме ядра выполнялось как можно меньше процессов, поскольку ошибки в ядре могут вызвать немедленный сбой системы.

Замысел, положенный в основу конструкции микроядра, направлен на достижение высокой надежности за счет разбиения операционной системы на небольшие, вполне определенные модули. Только один из них – микроядро – запускается в режиме ядра, а все остальные запускаются в виде относительно слабо наделенных полномочиями обычных пользовательских процессов.

В частности, если запустить каждый драйвер устройства и файловую систему как отдельные пользовательские процессы, то ошибка в одном из них может вызвать отказ соответствующего компонента, но не сможет вызвать сбой всей системы.

Таким образом, ошибка в драйвере звукового устройства приведет к искажению или пропаданию звука, но не вызовет зависание компьютера. В отличие от этого в монолитной системе, где все драйверы находятся в ядре, некорректный драйвер звукового устройства может запросто сослаться на неверный адрес памяти и привести систему к немедленной вынужденной остановке.

Было разработано и получило распространение множество различных микроядер.

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

Первое поколение:

* Микроядро Mach от университета Карнеги — Меллон (CMU).
* Микроядро ОС ChorusOS от института INRIA.

Второе поколение:

* Микроядро из ОС Minix от Эндрю Таненбаума (свободный университет Амстердама).
* L3 от немецкого ученого Йохена Лидтке .
* L4/x86 от Йохена Лидтке.

Третье поколение

* seL4 от фирмы NICTA.
* Coyotos от фирмы «The EROS Group, LLC».
* NOVA.

Особенно часто они используются в приложениях, работающих в реальном масштабе времени в промышленных устройствах, авионике и военной технике, которые выполняют особо важные задачи и должны отвечать очень высоким требованиям надежности.

Идея, имеющая некоторое отношение к использованию минимального ядра, заключается в том, чтобы помещать в ядро исполнительный механизм, а не политику. Чтобы пояснить эту мысль, рассмотрим планирование выполнения процессов. Относительно простой алгоритм планирования заключается в назначении каждому процессу приоритета с последующим запуском ядром готового к выполнению процесса с наиболее высоким приоритетом. Механизм, который находится в ядре, предназначен для поиска и запуска процесса с наибольшим приоритетом.

Политика, заключающаяся в назначении процессам приоритетов, должна быть реализована процессами, работающими в пользовательском режиме.

Таким образом, политика и механизм могут быть разобщены, а ядро может быть уменьшено в размерах.

**Клиент-серверная модель**. Небольшая вариация идеи микроядер выражается в обособлении двух классов процессов: серверов, каждый из которых предоставляет какую-нибудь службу, и клиентов, которые пользуются этими службами. Эта модель известна как клиент-серверная. Достаточно часто самый нижний уровень представлен микроядром, но это не обязательно. Вся суть заключается в наличии клиентских процессов и серверных процессов.

Связь между клиентами и серверами часто организуется с помощью передачи сообщений. Чтобы воспользоваться службой, клиентский процесс составляет сообщение, в котором говорится, что именно ему нужно, и отправляет его соответствующей службе. Затем служба выполняет определенную работу и отправляет обратно ответ. Если клиент и сервер запущены на одной и той же машине, то можно провести определенную оптимизацию, но концептуально здесь идет речь о передаче сообщений.

Очевидным развитием этой идеи будет запуск клиентов и серверов на разных компьютерах, соединенных локальной или глобальной сетью, как показано на рис. 3.

Таким образом, клиент-серверная модель является абстракцией, которая может быть использована как для отдельно взятой машины, так и для машин, объединенных в сеть.

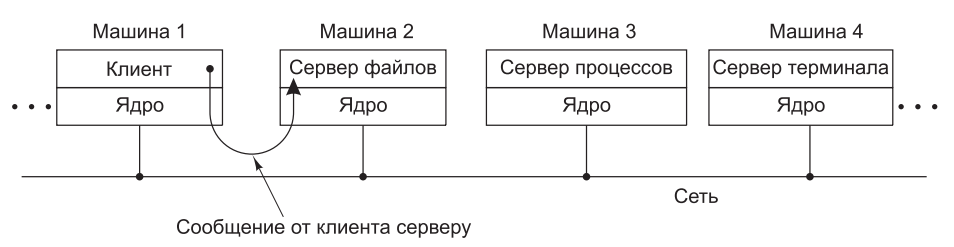
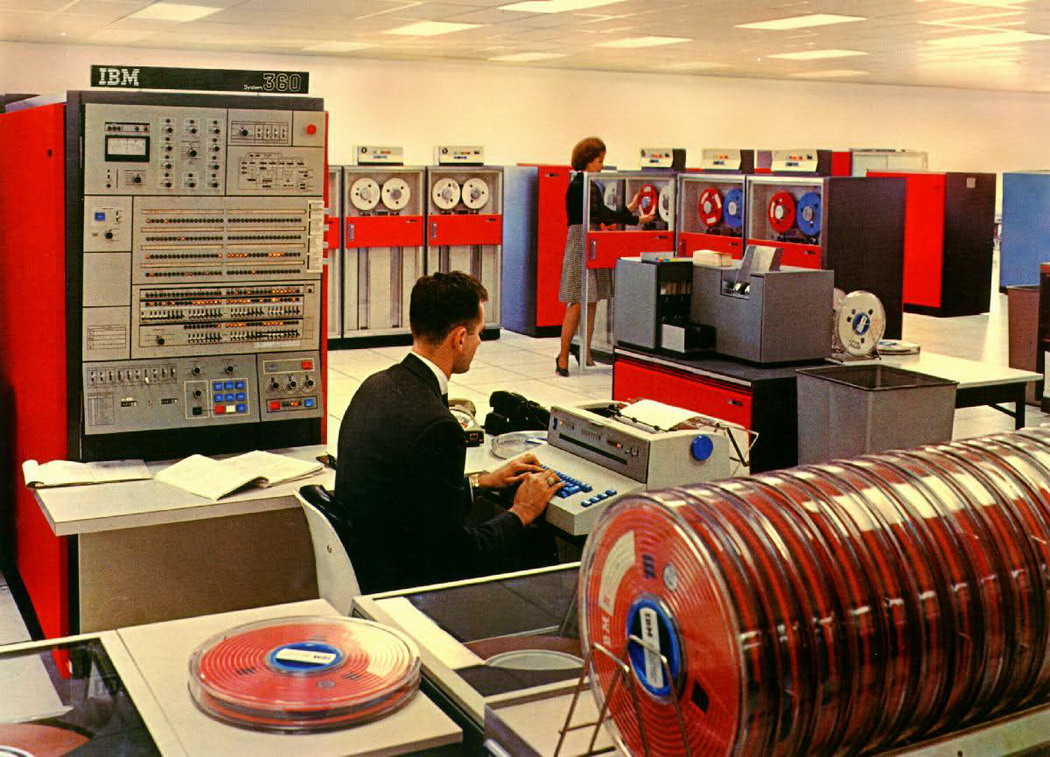


Рис. 3. Клиент-серверная модель, реализованная с помощью сети

**Виртуальные машины.**

Первые выпуски OS/360 были системами исключительно пакетной обработки. Но многие пользователи машин IBM/360 хотели получить возможность интерактивной работы с использованием терминала, поэтому различные группы разработчиков как в самой корпорации IBM, так и за ее пределами решили написать для этой машины системы с разделением времени. Позже была выпущена официальная система раз деления времени — TSS/360, и когда она наконец-то дошла до потребителей, то была настолько громоздкой и медлительной, что под нее было переоборудовано всего лишь несколько вычислительных центров. В конечном счете от этого проекта отказались, после того как на него уже было потрачено 50 млн долларов.

Рис. 4. IBM System/360 (S/360) — семейство компьютеров класса мейнфреймов, которое было анонсировано 7 апреля 1964 года. 

IBM System/360 послужила образцом для разработки ЕС ЭВМ. ЕС ЭВМ (Единая система электронных вычислительных машин) — советская серия компьютеров. Аналоги серий System/360 и System/370 фирмы IBM, выпускавшихся в США c 1964 года. Программно и аппаратно (аппаратно — только на уровне интерфейса внешних устройств) совместимы со своими американскими прообразами. Активно эксплуатировались в СССР и странах СЭВ с 1971 по 1990 годы, после чего стали выходить из эксплуатации, и примерно к 2000 практически исчезли.

**VM/370**  
Группа из научного центра IBM Scientific Center в Кембридже (Массачусетс) разработала совершенно другую систему, которую IBM в конечном итоге приняла как законченный продукт. Эта система, первоначально называвшаяся CP/CMS, а позже переименованная в VM/370 (Seawright and MacKinnon, 1979), была основана на следующем проницательном наблюдении: система с разделением времени обеспечивает, во-первых, многозадачность, а во-вторых, расширенную машину с более удобным интерфейсом, чем у простого оборудования.

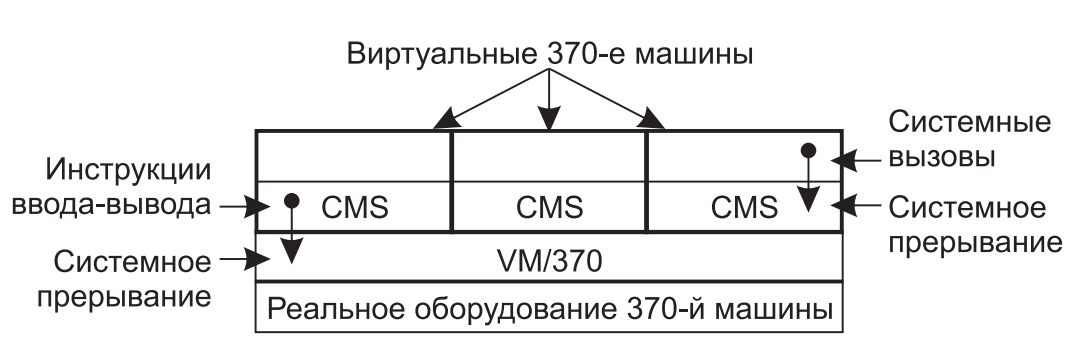
Сущность VM/370 заключается в полном разделении этих двух функций. Основа системы, известная как **монитор виртуальных машин**, запускается непосредственно на обычном оборудовании и обеспечивает многозадачность, предоставляя верхнему уровню не одну, а несколько виртуальных машин (рис. 5). Но, в отличие от всех других операционных систем, эти виртуальные машины не являются машинами с расширенной архитектурой. Они не поддерживают файлы и другие полезные свойства. Вместо этого они являются точной копией исходной аппаратуры, включающей режим ядра и пользователя, устройства ввода-вывода, прерывания и все остальное, что есть у настоящей машины.

Рис. 5. Структура VM/370 с тремя запущенными системами CMS

Поскольку каждая виртуальная машина идентична настоящему оборудованию, на каждой из них способна работать любая операционная система, которая может быть запущена непосредственно на самом оборудовании. На разных виртуальных машинах могут быть запущены разные операционные системы, как это часто и происходит на самом деле. Изначально на системах VM/370 пользователи запускали в своих виртуальных машинах OS/360 или одну из других больших операционных систем пакетной обработки или обработки транзакций, в то время как другие запускали однопользовательскую интерактивную систему CMS (Conversational Monitor System — система диалоговой обработки) для пользователей системы разделения времени.

Когда программа под управлением операционной системы CMS выполняет системный вызов, он перехватывается в системное прерывание операционной системы на своей собственной виртуальной машине, а не на VM/370, как это было бы при ее запуске на реальной, а не на виртуальной машине. Затем CMS выдает обычные команды ввода-вывода для чтения своего виртуального диска или другие команды, которые могут ей понадобиться для выполнения этого вызова. Эти команды ввода-вывода перехватываются VM/370, которая выполняет их в рамках моделирования реального оборудования.

При полном разделении функций многозадачности и предоставления машины с расширенной архитектурой каждая из составляющих может быть намного проще, гибче и удобнее для обслуживания.

В своем современном перерождении z/VM обычно используется для запуска нескольких полноценных операционных систем, а не упрощенных, однопользовательских систем вроде CMS. Например, на машинах zSeries можно запустить одну или несколько виртуальных машин Linux, а наряду с ними — обычные операционные системы IBM.

IBM System z (более раннее название — IBM eServer zSeries) — бренд, созданный компанией IBM для обозначения линейки мейнфреймов.

**Виртуальная машина**.

Хотя в IBM виртуальные машины используются уже четыре десятилетия и ряд других компаний, включая Oracle и Hewlett-Packard, недавно добавили поддержку виртуальных машин к своим высокопроизводительным промышленным серверам, тем не менее, по большому счету, идея виртуализации в мире персональных компьютеров до конца 1990 годов практически игнорировалась. Но сейчас сочетание новых потребностей, нового программного обеспечения и новых технологий придало этой теме особую актуальность. Сначала о потребностях. Многие компании традиционно запускали свои почтовые серверы, веб-серверы, FTP-серверы и все остальные серверы на отдельных компьютерах, иногда имеющих различные операционные системы. Виртуализация рассматривается ими как способ запуска всех этих серверов на одной и той же машине с возможностью избежать при этом отказа всех серверов при отказе одного из них.

Виртуализация популярна также в мире веб-хостинга. Без нее клиенты вынуждены выбирать между общим хостингом (который дает им только учетную запись на веб-сервере, но не позволяет управлять его программным обеспечением) и выделенным хостингом (который предоставляет им их собственную, очень гибкую, но не оправдывающую затрат машину при небольших или средних по объему веб-сайтах). Когда компания, предоставляющая услуги веб-хостинга (хостинг-провайдер), сдает в аренду виртуальные машины, на одной физической машине может быть запущено множество виртуальных машин, каждая из которых превращается в полноценную машину. Клиенты, арендовавшие виртуальную машину, могут запускать на ней какие угодно операционную систему и программное обеспечение, но за часть стоимости выделенного сервера (поскольку та же самая физическая машина одновременно поддерживает множество виртуальных машин).

Другой вариант использования виртуализации предназначен для конечных пользователей, которым необходима возможность одновременного запуска двух или более операционных систем, например Windows и Linux, поскольку некоторые из любимых ими приложений работают под управлением только одной из этих операционных систем.

Такая ситуация показана на рис. 6, а, при этом термин «монитор виртуальной машины» заменен на «гипервизор первого типа» (type 1 hypervisor), который широко используется в наши дни из-за краткости при наборе по сравнению с первым вариантом.

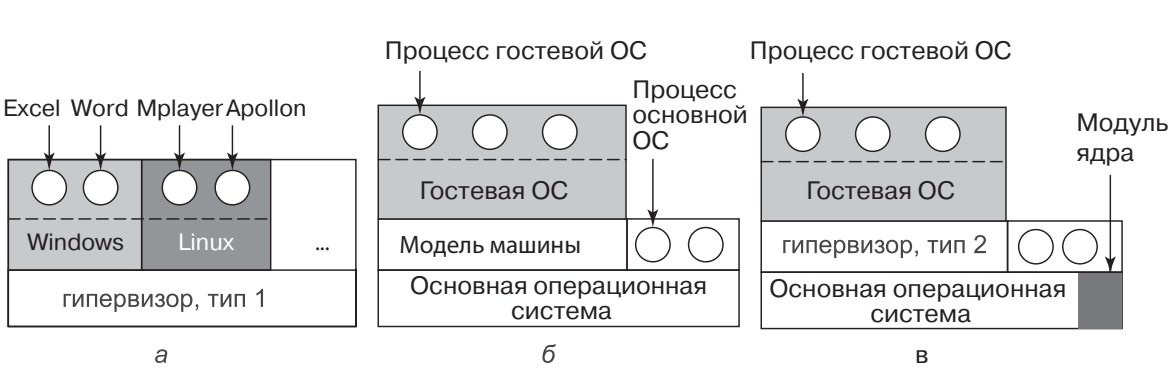


Рис. 6. Гипервизор: а — тип 1; б — чистый гипервизор, тип 2; в — практический гипервизор, тип 2

Привлекательность виртуальных машин сомнениям не подвергалась, проблема заключалась в их реализации. Чтобы запустить на компьютере программное обеспечение виртуальных машин, его центральный процессор должен быть готов к работе в этом режиме.

Проблема заключается в следующем. Когда операционная система, запущенная на виртуальной машине (в режиме пользователя), выполняет привилегированные инструкции, например изменение слова состояния программы — PSW или операцию ввода-вывода, необходимо, чтобы оборудование осуществило перехват данных инструкций и вызов монитора виртуальных машин, который выполнит их программную эмуляцию. На некоторых центральных процессорах, особенно на Pentium, его предшественниках и их клонах, попытки выполнения привилегированных инструкций в режиме пользователя просто игнорируются. Эта особенность исключает создание виртуальных машин на таком оборудовании, чем объясняется недостаточный интерес к ним в мире x86. Конечно, существовали интерпретаторы для Pentium, такие как Bochs, которые запускались на этом процессоре, но при потере производительности обычно в один-два порядка они не подходили для серьезной работы.

В 1990-х годах и в первые годы нового тысячелетия был реализован ряд научно-исследовательских проектов, в частности Disco в Стэнфорде и Xen в Кембридже. Эти исследования привели к появлению нескольких коммерческих продуктов (например, VMware Workstation и Xen), и интерес к виртуальным машинам снова вырос. Сегодня в число популярных гипервизоров кроме VMware и Xen входят KVM (для ядра Linux), VirtualBox (от Oracle) и Hyper-V (от Microsoft).

Некоторые из этих ранних исследовательских проектов улучшили производительность по сравнению с интерпретаторами типа Bochs путем трансляции блоков кода на лету, сохранения их во внутреннем кэше и повторного использования результата трансляции в случае их нового исполнения. Это существенно повысило производительность и привело к созданию того, что сейчас называется моделями машин (machine simulators) (рис. 6, б). Но хотя эта технология, известная как двоичная трансляция (binary translation), помогла улучшить ситуацию, получившиеся системы, несмотря на то что они неплохо подходили для публикаций на академических конференциях, по-прежнему не отличались быстротой для использования в коммерческих средах, где производительность имеет весьма большое значение.

Следующим шагом в улучшении производительности стало добавление модуля ядра (рис. 6, в) для выполнения ряда трудоемких задач. Сложившаяся сейчас практика показывает, что все коммерчески доступные гипервизоры, такие как VMwareWorkstation, используют эту гибридную стратегию (а также имеют множество других усовершенствований).

На практике действительным различием между гипервизорами типа 1 и типа 2 является то, что в типе 2 для создания процессов, сохранения файлов и т. д. используется основная операционная система (host operating system) и ее файловая система. Гипервизор типа 1 не имеет основной поддержки и должен выполнять все эти функции самостоятельно.

**Экзоядра**. Вместо клонирования настоящей машины, как это делается в виртуальных машинах, существует иная стратегия, которая заключается в их разделении, иными словами, в предоставлении каждому пользователю подмножества ресурсов. При этом одна виртуальная машина может получить дисковые блоки от 0 до 1023, другая – от 1024 до 2047, и т. д.

Самый нижний уровень, работающий в режиме ядра, – это программа под названием экзоядро. Его задача состоит в распределении ресурсов между виртуальными машинами и затем в отслеживании попыток их использования, чтобы ни одна из машин не пыталась использовать чужие ресурсы.

Преимущество схемы экзоядра заключается в том, что она исключает уровень отображения. При других методах работы каждая виртуальная машина считает, что она имеет свой собственный диск с нумерацией блоков от 0 до некоторого максимума. Поэтому монитор виртуальных машин должен вести таблицы преобразования адресов на диске (и всех других ресурсов). При использовании экзоядра необходимость в таком переназначении отпадает. Экзоядру нужно лишь отслеживать, какой виртуальной машине, какие ресурсы были переданы. Такой подход имеет еще одно преимущество: он отделяет многозадачность (в экзоядре) от пользовательской операционной системы (в пространстве пользователя) с меньшими затратами, так как для этого ему необходимо всего лишь не допускать вмешательства одной виртуальной машины в работу другой.

**Вопрос 2. Ядро и вспомогательные модули**

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

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

Модули ядра выполняют базовые функции ОС, связанные с управлением процессами, памятью, устройствами ввода-вывода и т. п. Именно ядро занимается переключением контекстов, загрузкой/выгрузкой страниц, обработкой прерываний. Непосредственное выполнение такого рода действий недоступно для приложений. При необходимости они могут обращаться к ядру с системными вызовами, используя для этого имеющийся в их распоряжении интерфейс прикладного программирования – *API*.

Функции, отнесенные в ведение ядра, являются наиболее часто используемыми функциями операционной системы, поэтому скорость их выполнения определяет производительность 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.

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

Вспомогательные модули ОС выполняют весьма полезные, но менее обязательные функции. Например, к таким 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. 7).

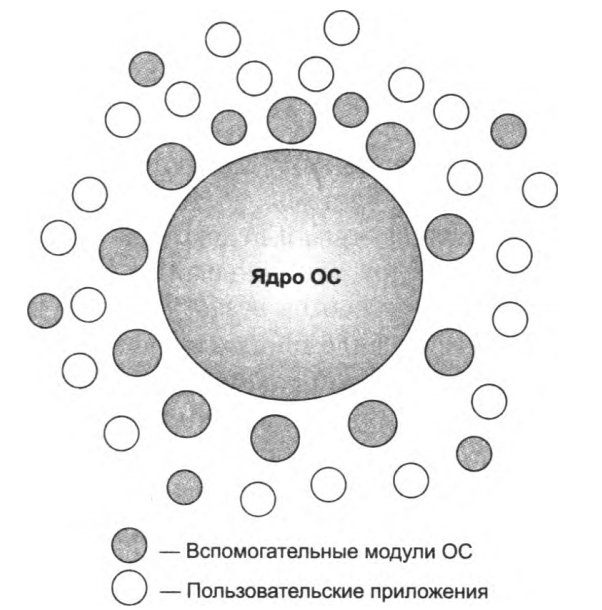


Рис. 7. Нечеткость границы между ОС и приложениями.

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. 8).

В некоторых случаях каждое исправление ядра может потребовать его полной перекомпиляции.

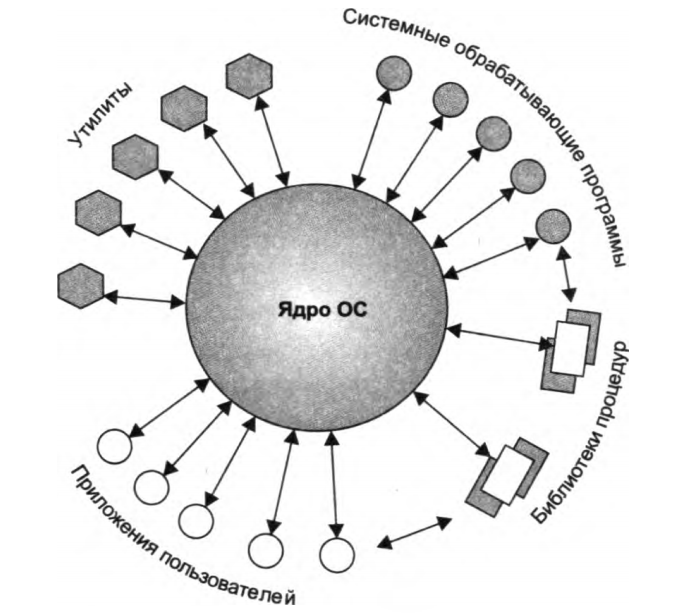


Рис. 8. Взаимодействие между ядром и вспомогательными модулями ОС

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

Так как основные функции ОС выполняются ядром, то чаще всего именно ядро становится той частью ОС, которая работает в привилегированном режиме (рис. 9). Иногда это свойство – работа в привилегированном режиме – служит основным определением понятия «ядро».

Приложения ставятся в подчиненное положение за счет запрета для них выполнения в пользовательском режиме некоторых критичных команд (инструкций), связанных с переключением процессора с задачи на задачу, управлением устройствами ввода-вывода, доступом к механизмам распределения и защиты памяти. Важно, что условия разрешения выполнения критичных команд находятся под полным контролем ОС, и этот контроль обеспечивается за счет набора команд, безусловно запрещенных для пользовательского режима.

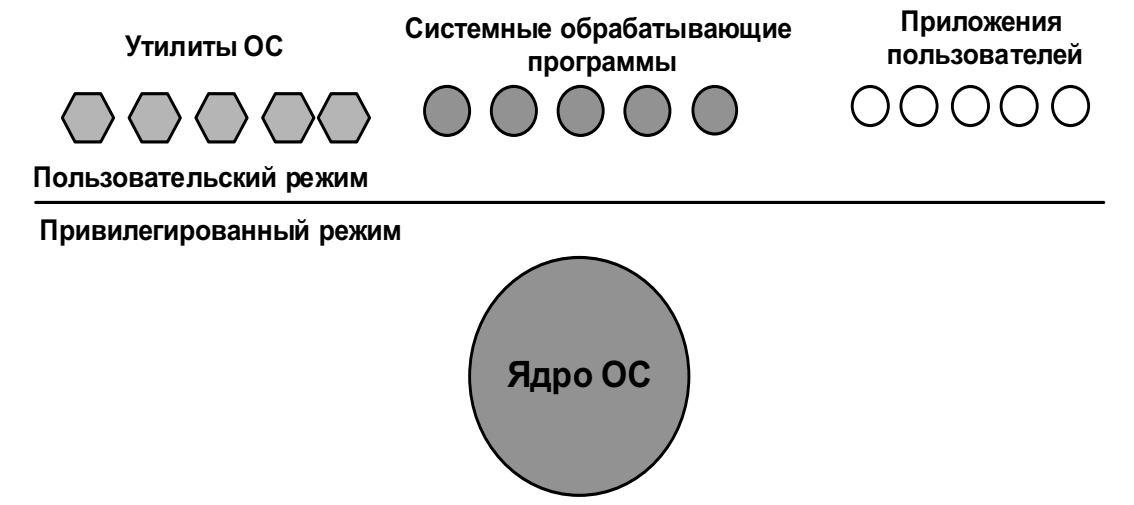
Аналогичным образом обеспечиваются привилегии ОС при доступе к памяти. Очень важно, что механизмы защиты памяти используются операционной системой не только для защиты своих областей памяти от приложений, но и для защиты областей памяти, выделенных ОС какому-либо приложению, от остальных приложений. Между количеством уровней привилегий, реализуемых аппаратно, и количеством уровней привилегий, поддерживаемых ОС, нет прямого соответствия.

Рис. 9. Архитектура операционной системы с ядром в привилегированном режиме.

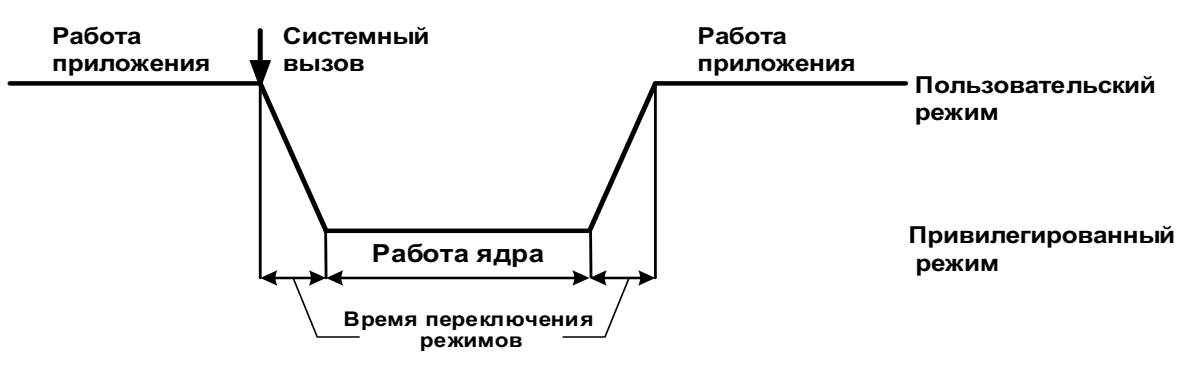
Повышение устойчивости операционной системы, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления выполнения системных вызовов. Системный вызов инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению – переключение из привилегированного режима в пользовательский (рис. 9).

Рис. 10. Смена режимов при выполнении системного вызова к привилегированному ядру.

Во всех типах процессоров из-за дополнительной двукратной задержки переключения переход на процедуру со сменой режима выполняется медленнее, чем вызов процедуры без смены режима.

Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической. Ее используют большинство популярных операционных систем, в том числе семейства ОС *Windows NT, Unix/Linux, Mac OS X*, операционные системы мэйнфреймов.

Однако в некоторых случаях разработчики ОС отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, в известных специализированных ОС *NetWare 3.x* и *4.x* компании *Novell* привилегированный режим процессоров использовались как для работы ядра, так и для работы специфических приложений – загружаемых модулей *NLM* (рис. 10).

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.

Аналогичный подход используется в самой популярной ОС маршрутизаторов – *Cisco IOS* (*Internetwork Operating System*). Эта специализированная ОС также состоит из ядра и приложений, которые выполняются в одном и том же режиме процессора. При этом время переключения системы между ядром и приложениями сокращается, а это очень важно для системы реального времени *IOS*, главной задачей которой является обработка поступающих в маршрутизатор по входным интерфейсам пакетов с минимальными задержками.

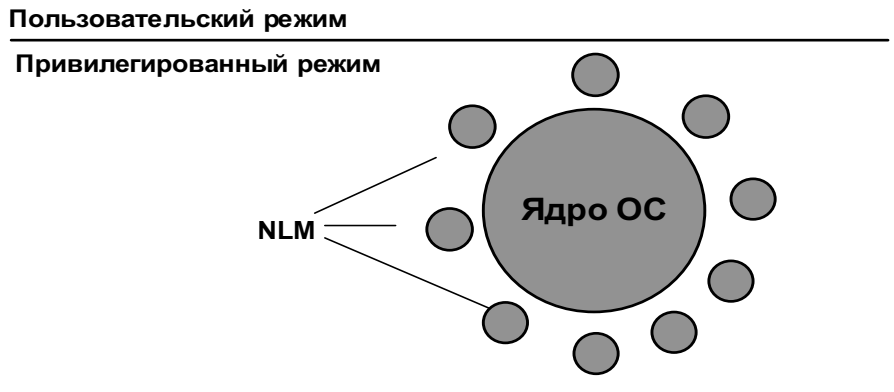


Рис. 11. Упрощенная архитектура операционной системы NetWare

**Многослойная структура ОС**. Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как состоящую из трех иерархически расположенных слоев: нижний слой образует аппаратура, промежуточный – ядро, а утилиты, обрабатывающие программы и приложения, составляют верхний слой системы (рис. 11).

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. 12. Трехслойная схема вычислительной системы

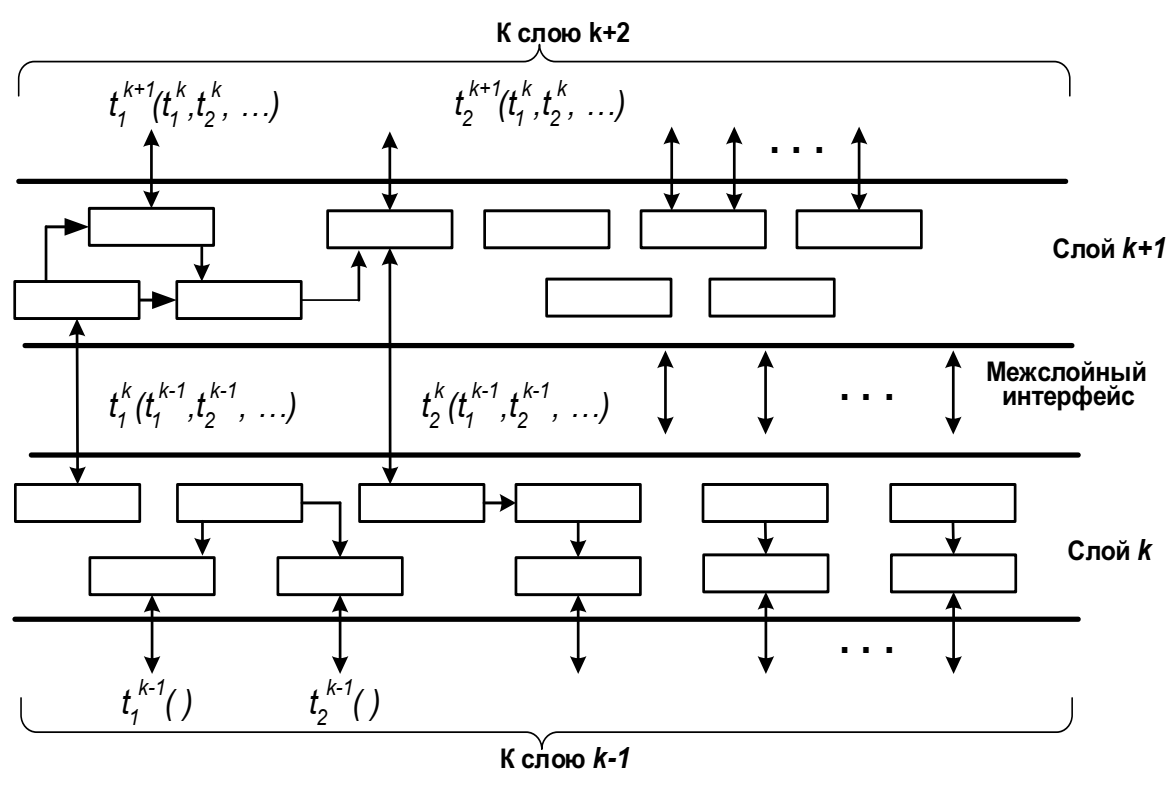
Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе программных. В соответствии с этим подходом система состоит из иерархии слоев. Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс (рис. 13).

Рис. 13. Концепция многослойного взаимодействия

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 возможности 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. Поскольку ядро представляет собой сложный многофункциональный комплекс, то многослойный подход обычно распространяется и на структуру ядра. Ядро может состоять из следующих слоев (рис. 14.).

 Рис. 14. Многослойная структура ядра ОС

**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. 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, что позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС. *Linux, Unix, Mac OS*, операционные системы семейства *Windows NT* – во всех этих ОС имеется четко определенный слой программных модулей, экранирующих особенности аппаратуры.

**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 «на верху» 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 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. *API*-функции, 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/а.txt", О\_RDОNLY*) приложение открывает файл *a.txt*, хранящийся в каталоге /*doc*, а с помощью системного вызова read (*fd, buffer,’count*) читает некоторое количество байтов из этого файла в область своего адресного пространства, имеющую имя *buffer*. 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.

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними может быть иным. В системах, предназначенных для аппаратных платформ одного типа, слой машинно-зависимых модулей может сливаться 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 схемы. Для ускорения работы ядра в некоторых случаях происходит непосредственное обращение с верхнего слоя к функциям нижних слоев, минуя промежуточные.

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.

**Заключение.**

Операционные системы можно рассматривать с двух точек зрения: в качестве менеджеров ресурсов и в качестве расширенных машин. С точки зрения менеджера ресурсов работа операционных систем заключается в эффективном управлении различными частями системы. С точки зрения расширенной машины работа операционных систем состоит в предоставлении пользователям абстракций, более удобных в использовании по сравнению с реальным компьютером. В число таких абстракций включаются процессы, адресные пространства и файлы.

Операционные системы могут иметь различную структуру. Наиболее распространенными являются монолитная система, многоуровневая система, микроядро, клиент-серверная система, виртуальная машина и экзоядро.