Лекция 1: Архитектура, назначение и функции операционных систем.

1.1. Понятие операционной системы. Виртуальные машины.

Современный компьютер – сложнейшая аппаратно-программная система. Написание программ для компьютера, их отладка и последующее выполнение представляет собой сложную трудоемкую задачу. Основная причина этого – огромная разница между тем, что удобно для людей, и тем, что удобно для компьютеров. Компьютер понимает только свой, машинный язык (назовем его Я0), а для человека наиболее удобен разговорный или хотя бы язык описания алгоритмов – алгоритмический язык. Проблему можно решить двумя способами. Оба способа связаны с разработкой команд, которые были бы более удобны для человека, чем встроенные машинные команды компьютера. Эти новые команды в совокупности формируют некоторый язык, который назовем Я1.

Упомянутые два способа решения проблемы различаются тем, каким образом компьютер будет выполнять программы, написанные на языке Я1. Первый способ – замена каждой команды языка Я1 на эквивалентный набор команд в языке Я0. В этом случае компьютер выполняет новую программу, написанную на языке Я0, вместо программы, написанной на языке Я1. Эта технология называется трансляцией.

Второй способ – написание программы на языке Я0, которая берет программы, написанные на языке Я1, в качестве входных данных, рассматривает каждую команду по очереди и сразу выполняет эквивалентный набор команд языка Я0. Эта технология не требует составления новой программы на Я0. Она называется интерпретацией, а программа, которая осуществляет интерпретацию, называется интерпретатором.

В подобной ситуации проще представить себе существование гипотетического компьютера или виртуальной машины, для которой машинным языком является язык Я1, чем думать о трансляции и интерпретации. Назовем такую виртуальную машину М1, а виртуальную машину с языком Я0 – М0. Для виртуальных машин можно будет писать программы, как будто они (машины) действительно существуют.

Очевидно, можно пойти дальше – создать еще набор команд, который в большей степени ориентирован на человека и в меньшей степени на компьютер, чем Я1. Этот набор формирует язык Я2 и, соответственно, виртуальную машину М2. Так можно продолжать до тех пор, пока не дойдем до подходящего нам языка уровня n.

Большинство современных компьютеров состоит из двух и более уровней. Уровень 0 – аппаратное обеспечение машины. Электронные схемы этого уровня выполняют программы, написанные на языке уровня 1. Следующий уровень – микроархитектурный уровень.

На этом уровне можно видеть совокупности 8 или 32 (иногда и больше) регистров, которые формируют локальную память и АЛУ (арифметико-логическое устройство). Регистры вместе с АЛУ формируют тракт данных, по которому поступают данные. Основная операция этого тракта заключается в следующем. Выбирается один или два регистра, АЛУ производит над ними какую-то операцию, а результат помещается в один из этих регистров. На некоторых машинах работа тракта контролируется особой программой, которая называется микропрограммой. В других машинах такой контроль выполняется аппаратным обеспечением.

Следующий (второй) уровень составляет уровень архитектуры системы команд. Команды используют регистры и другие возможности аппаратуры. Команды формируют уровень ISA (Instruction Set Architecture), называемый машинным языком. Обычно машинный язык содержит от 50 до 300 команд, служащих преимущественно для перемещения данных по компьютеру, выполнения арифметических операций и сравнения величин.

Следующий (третий) уровень обычно – гибридный. Большинство команд в его языке есть также и на уровне архитектуры системы команд. У этого уровня есть некоторые дополнительные особенности: набор новых команд, другая организация памяти, способность выполнять две и более программы одновременно и некоторые другие. С течением времени набор таких команд существенно расширился. В нем появились так называемые макросы операционной системы или вызовы супервизора, называемые теперь системными вызовами.

Новые средства, появившиеся на третьем уровне, выполняются интерпретатором, который работает на втором уровне. Этот интерпретатор был когда-то назван операционной системой. Команды третьего уровня, идентичные командам второго уровня, выполняются микропрограммой или аппаратным обеспечением, но не операционной системой. Иными словами, одна часть команд третьего уровня интерпретируется операционной системой, а другая часть – микропрограммой. Вот почему этот уровень операционной системы считается гибридным.

Операционная система была создана для того, чтобы автоматизировать работу оператора и скрыть от пользователя сложности общения с аппаратурой, предоставив ему более удобную систему команд. Нижние три уровня (с нулевого по второй) конструируются не для того, чтобы с ними работал обычный программист. Они изначально предназначены для работы интерпретаторов и трансляторов, поддерживающих более высокие уровни. Эти трансляторы и интерпретаторы составляются системными программистами, которые специализируются на разработке и построении новых виртуальных машин.

Над операционной системой (ОС) расположены остальные системные программы. Здесь находятся интерпретатор команд (оболочка), компиляторы, редакторы и т.д. Подобные программы не являются частью ОС (иногда оболочку пользователи считают операционной системой). Под операционной системой обычно понимается то программное обеспечение, которое запускается в режиме ядра или, как еще его называют, режиме супервизора. Она защищена от вмешательства пользователя с помощью специальных аппаратных средств.

Четвертый уровень представляет собой символическую форму одного из языков низкого уровня (обычно ассемблер). На этом уровне можно писать программы в приемлемой для человека форме. Эти программы сначала транслируются на язык уровня 1, 2 или 3, а затем интерпретируются соответствующей виртуальной или фактически существующей (физической) машиной.

Уровни с пятого и выше предназначены для прикладных программистов, решающих конкретные задачи на языках высокого уровня (C, C++, C#, VBA и др.). Компиляторы и редакторы этих уровней запускаются в пользовательском режиме. На еще более высоких уровнях располагаются прикладные программы пользователей.

Большинство пользователей компьютеров имеют опыт общения с операционной системой, по крайней мере, в той степени, чтобы эффективно выполнять свои текущие задачи. Однако они испытывают затруднения при попытке дать определение операционной системе. В известной степени проблема связана с тем, что операционные системы выполняют две основные, но практически не связанные между собой функции: расширение возможностей компьютера и управление его ресурсами.

С точки зрения пользователя ОС выполняет функцию расширенной машины или виртуальной машины, в которой легче программировать и легче работать, чем непосредственно с аппаратным обеспечением, составляющим реальный компьютер. Операционная система не только устраняет необходимость работы непосредственно с дисками и предоставляет простой, ориентированный на работу с файлами интерфейс, но и скрывает множество неприятной

работы с прерываниями, счетчиками времени, организацией памяти и другими компонентами низкого уровня.

Однако концепция, рассматривающая операционную систему прежде всего как удобный интерфейс пользователя, – это взгляд сверху вниз. Альтернативный взгляд, снизу вверх, дает представление об операционной системе как о механизме, присутствующем в компьютере для управления всеми компонентами этой сложнейшей системы. В соответствии с этим подходом работа операционной системы заключается в обеспечении организованного и контролируемого распределения процессоров, памяти, дисков, принтеров, устройств ввода-вывода, датчиков времени и т.п. между различными программами, конкурирующими за право их использовать.

1.2. Операционная система, среда и операционная оболочка

Операционные системы (ОС) в современном их понимании (их назначении и сущности) появились значительно позже первых компьютеров (правда, по всей видимости, и исчезнут в этой сущности в компьютерах будущего). Почему и когда появились ОС? Считается, что первая цифровая вычислительная машина ENIAC (Electronic Numerical Integrator and Computer) была создана в 1946 году по проекту "Проект PX" Министерства обороны США. На реализацию проекта затрачено 500 тыс. долларов. Компьютер содержал 18000 электронных ламп, массу всякой электроники, включал в себя 12 десятиразрядных сумматоров, а для ускорения некоторых арифметических операций имел умножитель и "делитель-извлекатель" квадратного корня. Программирование сводилось к связыванию различных блоков проводами. Конечно, никакого программного обеспечения и тем более операционных систем тогда еще не существовало [10, 13].

Интенсивное создание различных моделей ЭВМ относится к началу 50-х годов прошлого века. В эти годы одни и те же группы людей участвовали и в проектировании, и в создании, и в программировании, и в эксплуатации ЭВМ. Программирование осуществлялось исключительно на машинном языке (а затем на ассемблере), не было никакого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм. Операционные системы еще не появились, а все задачи организации вычислительного процесса решались вручную каждым программистом с примитивного пульта управления ЭВМ.

С появлением полупроводниковых элементов вычислительные возможности компьютеров существенно выросли. Наряду с этим заметно прогрессировали достижения в области автоматизации программирования и организации вычислительных работ. Появились алгоритмические языки (алгол, фортран, кобол) и системное программное обеспечение (трансляторы, редакторы связи, загрузчики и др.). Выполнение программ усложнилось и включало в себя следующие основные действия:

- загрузка нужного транслятора (установка нужных МЛ и др.);
- запуск транслятора и получение программы в машинных кодах;
- связывание программы с библиотечными подпрограммами;
- загрузка программы в оперативную память;
- запуск программы;
- вывод результатов работы программы на печатающее или другое периферийное устройство.

Для организации эффективной загрузки всех средств компьютера в штаты вычислительных центров ввели должности специально обученных операторов, профессионально выполнявших работу по организации вычислительного процесса для всех пользователей этого центра. Однако, как бы ни был подготовлен оператор, ему тяжело

состязаться в производительности с работой устройств компьютера. И поэтому большую часть времени дорогостоящий процессор простаивал, а следовательно, использование компьютеров не было эффективным.

С целью исключения простоев были предприняты попытки разработки специальных программ – мониторов, прообразов первых операционных систем, которые осуществляли автоматический переход от задания к заданию. Считается, что первую операционную систему создала в 1952 году для своих компьютеров IBM-701 исследовательская лаборатория фирмы General Motors [9]. В 1955 году эта фирма и North American Aviation совместно разработали ОС для компьютера IBM-704.

В конце 50-х годов прошлого века ведущие фирмы изготовители поставляли операционные системы со следующими характеристиками:

- пакетная обработка одного потока задач;
- наличие стандартных программ ввода-вывода;
- возможности автоматического перехода от программы к программе;
- средства восстановления после ошибок, обеспечивающие автоматическую "очистку" компьютера в случае аварийного завершения очередной задачи и позволяющие запускать следующую задачу при минимальном вмешательстве оператора;
- языки управления заданиями, предоставляющие пользователям возможность описывать свои задания и ресурсы, требуемые для их выполнения.

Пакет представляет собой набор (колоду) перфокарт, организованную специальным образом (задание, программы, данные). Для ускорения работы он мог переноситься на магнитную ленту или диск. Это позволяло сократить простои дорогой аппаратуры. Надо сказать, что в настоящее время в связи с прогрессом микроэлектронных технологий и методологий программирования значительно снизилась стоимость аппаратных и программных средств компьютерной техники. Поэтому сейчас основное внимание уделяется тому, чтобы сделать работу пользователей и программистов более эффективной, поскольку затраты труда квалифицированных специалистов сейчас представляют собой гораздо большую долю общей стоимости вычислительных систем, чем аппаратные и программные средства компьютеров.

Расположение операционной системы в иерархической структуре программного и аппаратного обеспечения компьютера можно представить, как показано на рис. 1.1.

Рис. 1.1. Иерархическая структура программно-аппаратных средств компьютера

Самый нижний уровень содержит различные устройства компьютера, состоящие из микросхем, проводников, источников питания, электронно-лучевых трубок и т.п. Этот уровень можно разделить на подуровни, например контроллеры устройств, а затем сами устройства. Возможно деление и на большее число уровней. Выше расположен микроархитектурный уровень, на котором физические устройства рассматриваются как отдельные функциональные единицы.

На микроархитектурном уровне находятся внутренние регистры центрального процессора (их может быть несколько) и арифметико-логические устройства со средствами управления ими. На этом уровне реализуется выполнение машинных команд. В процессе выполнения команд используются регистры процессора и устройств, а также другие возможности аппаратуры. Команды, видимые для работающего на ассемблере программиста, формируют уровень ISA (Instruction Set Architecture – архитектура системы команд), часто называемый машинным языком.

Операционная система предназначена для того, чтобы скрыть все эти сложности. Конечный пользователь обычно не интересуется деталями устройства аппаратного обеспечения компьютера. Компьютер ему видится как набор приложений. Приложение может быть написано программистом на каком-либо языке программирования. Для упрощения этой работы программист использует набор системных программ, некоторые из которых называются утилитами. С их помощью реализуются часто используемые функции, которые помогают работать с файлами, управлять устройствами ввода-вывода и т.п. Программист применяет эти средства при разработке программ, а приложения во время выполнения обращаются к утилитам для выполнения определенных функций. Наиболее важной из системных программ является операционная система, которая освобождает программиста от необходимости глубокого знания устройства компьютера и представляет ему удобный интерфейс для его использования. Операционная система выступает в роли посредника, облегчая программисту, пользователям и программным приложениям доступ к различным службам и возможностям компьютера [10].

Таким образом, операционная система – это набор программ, контролирующих работу прикладных программ и системных приложений и исполняющих роль интерфейса между пользователями, программистами, прикладными программами, системными приложениями и аппаратным обеспечением компьютера.

Образно можно сказать, что аппаратура компьютера предоставляет "сырую" вычислительную мощность, а задача операционной системы заключается в том, чтобы сделать использование этой вычислительной мощности доступным и по возможности удобным для пользователя. Программист может не знать детали управления конкретными ресурсами (например, диском) компьютера и должен обращаться к операционной системе с соответствующими вызовами, чтобы получить от нее необходимые сервисы и функции. Этот набор сервисов и функций и представляет собой операционную среду, в которой выполняются прикладные программы.

Таким образом, операционная среда – это программная среда, образуемая операционной системой, определяющая интерфейс прикладного программирования (API) как множество системных функций и сервисов (системных вызовов), которые предоставляются прикладным программам. Операционная среда может включать несколько интерфейсов прикладного программирования. Кроме основной операционной среды, называемой естественной (native), могут быть организованы путем эмуляции (моделирования) дополнительные программные среды, позволяющие выполнять приложения, которые рассчитаны на другие операционные системы и даже другие компьютеры.

Еще одно важное понятие, связанное с операционной системой, относится к реализации пользовательских интерфейсов. Как правило, любая операционная система обеспечивает удобную работу пользователя за счет средств пользовательского интерфейса. Эти средства могут быть неотъемлемой частью операционной среды (например, графический интерфейс Windows или текстовый интерфейс командной строки MS DOS), а могут быть реализованы отдельной системной программой – оболочкой операционной системы (например, Norton Commander для MS DOS). В общем случае под оболочкой операционной системы понимается

часть операционной среды, определяющая интерфейс пользователя, его реализацию (текстовый, графический и т.п.), командные и сервисные возможности пользователя по управлению прикладными программами и компьютером.

Перейдем к рассмотрению эволюции операционных систем.

1.3. Эволюция операционных систем

Рассматривая эволюцию ОС, следует иметь в виду, что разница во времени реализации некоторых принципов организации отдельных операционных систем до их общего признания, а также терминологическая неопределенность не позволяют дать точную хронологию развития ОС. Однако сейчас уже достаточно точно можно определить основные вехи на пути эволюции операционных систем.

Существуют также различные подходы к определению поколений ОС. Известно разделение ОС на поколения в соответствии с поколениями вычислительных машин и систем [5, 9, 10, 13]. Такое деление нельзя считать полностью удовлетворительным, так как развитие методов организации ОС в рамках одного поколения ЭВМ, как показал опыт их создания, происходит в достаточно широком диапазоне. Другая точка зрения не связывает поколение ОС с соответствующими поколениями ЭВМ. Так, например, известно определение поколений ОС по уровням входного языка ЭВМ, режимам использования центральных процессоров, формам эксплуатации систем и т.п. [5, 13].

Видимо, наиболее целесообразным следует считать выделение этапов развития ОС в рамках отдельных поколений ЭВМ и ВС.

Первым этапом развития системного программного обеспечения можно считать использование библиотечных программ, стандартных и служебных подпрограмм и макрокоманд. Концепция библиотек подпрограмм является наиболее ранней и восходит к 1949 году [4, 17]. С появлением библиотек получили развитие автоматические средства их сопровождения – программы-загрузчики и редакторы связей. Эти средства применялись в ЭВМ первого поколения, когда операционных систем как таковых еще не существовало.

Стремление устранить несоответствие между производительностью процессоров и скоростью работы электромеханических устройств ввода-вывода, с одной стороны, и использование достаточно быстродействующих накопителей на магнитных лентах и барабанах (НМЛ и НМБ), а затем на магнитных дисках (НМД), с другой стороны, привело к необходимости решения задач буферизации и блокирования-деблокирования данных. Возникли специальные программы методов доступа, которые вносились в объекты модулей редакторов связей (впоследствии стали использоваться принципы полибуферизации). Для поддержания работоспособности и облегчения процессов эксплуатации машин создавались диагностические программы. Таким образом было создано базовое системное программное обеспечение.

С улучшением характеристик ЭВМ и ростом их производительности стало ясно, что существующего базового программного обеспечения (ПО) недостаточно. Появились операционные системы ранней пакетной обработки – мониторы. В рамках системы пакетной обработки во время выполнения любой работы в пакете (трансляция, сборка, выполнение готовой программы) никакая часть системного ПО не находилась в оперативной памяти, так как вся память предоставлялась текущей работе. Затем появились мониторные системы, в которых оперативная память делилась на три области: фиксированная область мониторной системы, область пользователя и область общей памяти (для хранения данных, которыми могут обмениваться объектные модули).

Началось интенсивное развитие методов управления данными, возникала такая важная функция ОС, как реализация ввода-вывода без участия центрального процесса – так называемый спулинг (от англ. SPOOL – Simultaneous Peripheral Operation on Line).

Появление новых аппаратных разработок (1959-1963 гг.) – систем прерываний, таймеров, каналов – стимулировало дальнейшее развитие ОС [4, 5, 9]. Возникли исполнительные системы, которые представляли собой набор программ для распределения ресурсов ЭВМ, связей с оператором, управления вычислительным процессом и управления вводом-выводом. Такие исполнительные системы позволили реализовать довольно эффективную по тому времени форму эксплуатации вычислительной системы – однопрограммную пакетную обработку. Эти системы давали пользователю такие средства, как контрольные точки, логические таймеры, возможность построения программ оверлейной структуры, обнаружение нарушений программами ограничений, принятых в системе, управление файлами, сбор учетной информации и др.

Однако однопрограммная пакетная обработка с ростом производительности ЭВМ не могла обеспечить экономически приемлемый уровень эксплуатации машин. Решением стало мультипрограммирование – способ организации вычислительного процесса, при котором в памяти компьютера находится несколько программ, попеременно выполняющихся одним процессором, причем для начала или продолжения счета по одной программе не требовалось завершения других. В мультипрограммной среде проблемы распределения ресурсов и защиты стали более острыми и трудноразрешимыми.

Теория построения операционных систем в этот период обогатилась рядом плодотворных идей. Появились различные формы мультипрограммных режимов работы, в том числе разделение времени – режим, обеспечивающий работу многотерминальной системы. Была создана и развита концепция виртуальной памяти, а затем и виртуальных машин. Режим разделения времени позволил пользователю интерактивно взаимодействовать со своими программами, как это было до появления систем пакетной обработки.

Одной из первых ОС, использующих эти новейшие решения, была операционная система MCP (главная управляющая программа), созданная фирмой Burroughs для своих компьютеров B5000 в 1963 году. В этой ОС были реализованы многие концепции и идеи, ставшие впоследствии стандартными для многих операционных систем:

- мультипрограммирование;
- мультипроцессорная обработка;
- виртуальная память;
- возможность отладки программ на исходном языке;
- написание операционной системы на языке высокого уровня.

Известной системой разделения времени того периода стала система CTSS (Compatible Time Sharing System) – совместимая система разделения времени, разработанная в Массачусетском технологическом институте (1963 год) для компьютера IBM-7094 [37]. Эта система была использована для разработки в этом же институте совместно с Bell Labs и General Electric системы разделения времени следующего поколения MULTICS (Multiplexed Information And Computing Service). Примечательно, что эта ОС была написана в основном на языке высокого уровня EPL (первая версия языка PL/1 фирма IBM).

Одним из важнейших событий в истории операционных систем считается появление в 1964 году семейства компьютеров под названием System/360 фирмы IBM, а позже – System/370 [11]. Это было первой в мире реализацией концепции семейства программно и информационно

совместимых компьютеров, ставшей впоследствии стандартной для всех фирм компьютерной отрасли.

Нужно отметить, что основной формой использования ЭВМ, как в системах разделения времени, так и в системах пакетной обработки, стал многотерминальный режим. При этом не только оператор, но и все пользователи получали возможность формулировать свои задания и управлять их выполнением со своего терминала. Поскольку терминальные комплексы скоро стало возможным размещать на значительных расстояниях от компьютера (благодаря модемным телефонным соединениям), появились системы удаленного ввода заданий и телеобработки данных. В ОС добавились модули, реализующие протоколы связи [10, 13].

К этому времени произошло существенное изменение в распределении функций между аппаратными и программными средствами компьютера. Операционная система становится "неотъемлемой частью ЭВМ", как бы продолжением аппаратуры. В процессорах появился привилегированный (Супервизор в OS/360) и пользовательский (Задача в OS/360) режимы работы, мощная система прерываний, защита памяти, специальные регистры для быстрого переключения программ, средства поддержки виртуальной памяти и др.

В начале 70-х годов появились первые сетевые ОС, которые позволили не только рассредоточить пользователей, как в системах телеобработки данных, но и организовать распределенное хранение и обработку данных между компьютерами, соединенных электрическими связями. Известен проект ARPANET МО США. В 1974 году IBM объявила о создании собственной сетевой архитектуры SNA для своих мэйнфреймов, обеспечивающей взаимодействие типа "терминал-терминал", "терминал-компьютер", "компьютер-компьютер". В Европе активно разрабатывалась технология построения сетей с коммутацией пакетов на основе протоколов X.25.

К середине 70-х годов наряду с мэйнфреймами широкое распространение получили мини-компьютеры (PDP-11, Nova, HP). Архитектура мини-компьютеров была значительно проще, многие функции мультипрограммных ОС мэйнфреймов были усечены. Операционные системы мини-ЭВМ стали делать специализированными (RSX-11M – разделение времени, RT-11 – ОС реального времени) и не всегда многопользовательскими.

Важной вехой в истории мини-компьютеров и вообще в истории операционных систем явилось создание ОС UNIX. Написал эту систему Кен Томпсон (Ken Thompson), один из специалистов по компьютерам в BELL Labs, работавший над проектом MULTICS. Собственно, его UNIX – это усеченная однопользовательская версия системы MULTICS. Первоначальное название этой системы – UNICS (UNiplexed Information and Computing Service – примитивная информационная и компьютерная служба). Так в шутку была названа эта система, поскольку MULTICS (MULTiplexed Information and Computing Service) – мультиплексная информационная и компьютерная служба. С середины 70-х годов началось массовое использование ОС UNIX, написанной на 90% на языке С. Широкое распространение С-компиляторов сделало UNIX уникальной переносимой ОС, а поскольку она поставлялась вместе с исходными кодами, она стала первой открытой операционной системой. Гибкость, элегантность, мощные функциональные возможности и открытость позволили ей занять прочные позиции во всех классах компьютеров – от персональных до супер-ЭВМ.

Доступность мини-компьютеров послужила стимулом для создания локальных сетей. В простейших ЛВС компьютеры соединялись через последовательные порты. Первое сетевое приложение для ОС UNIX – программа UUCP (Unix to Unix Copy Program) – появилось в 1976 году.

Дальнейшее развитие сетевых систем со стеком протоколов TCP/IP: в 1983 году он был принят МО США в качестве стандарта и использован в сети ARPANET. В этом же году ARPANET

разделилась на MILNET (для военного ведомства США) и новую ARPANET, которую стали называть Internet.

Все восьмидесятые годы характерны появлением все более совершенных версий UNIX: Sun OS, HP-UX, Irix, AIX и др. Для решения проблемы их совместимости были приняты стандарты POSIX и XPG, определяющие интерфейсы этих систем для приложений.

Еще одним знаменательным событием для истории операционных систем было появление в начале 80-х годов персональных компьютеров. Они послужили мощным толчком для распределения локальных сетей, в результате поддержка сетевых функций стала для ОС ПК необходимым условием. Однако и дружественный интерфейс, и сетевые функции появились у ОС ПК не сразу [13].

Наиболее популярной версией ОС раннего этапа развития персональных компьютеров была MS-DOS компании Microsoft – однопрограммная, однопользовательская ОС с интерфейсом командной строки. Многие функции, обеспечивающие удобство работы пользователю, в этой ОС предоставлялись дополнительными программами – оболочкой Norton Commander, PC Tools и др. Наибольшее влияние на развитие программного обеспечения ПК оказала операционная среда Windows, первая версия которой появилась в 1985 году. Сетевые функции также реализовались с помощью сетевых оболочек и появились в MS-DOS версии 3.1. В это же время появились сетевые продукты Microsoft – MS-NET, а позже – LAN Manager, Windows for Workgroup, а затем и Windows NT.

Другим путем пошла компания Novell: ее продукт NetWare – операционная система со встроенными сетевыми функциями. ОС NetWare распространялась как операционная система для центрального сервера локальной сети и за счет специализации функций файл-сервера обеспечивала высокую скорость удаленного доступа к файлам и повышенную безопасность данных. Однако эта ОС имела специфический программный интерфейс (API), что затрудняло разработку приложений.

В 1987 году появилась первая многозадачная ОС для ПК – OS/2, разработанная Microsoft совместно с IBM. Эта была хорошо продуманная система с виртуальной памятью, графическим интерфейсом и возможностью выполнять DOS-приложения. Для нее были созданы и получили распространение сетевые оболочки LAN Manager (Microsoft) и LAN Server (IBM). Эти оболочки уступали по производительности файловому серверу NetWare и потребляли больше аппаратных ресурсов, но имели важные достоинства. Они позволяли выполнять на сервере любые программы, разработанные для OS/2, MS-DOS и Windows, кроме того, можно было использовать компьютер, на котором они работали, в качестве рабочей станции. Неудачная рыночная судьба OS/2 не позволила системам LAN-Manager и LAN-Server захватить заметную долю рынка, но принципы работы этих сетевых систем во многом нашли свое воплощение в ОС 90-х годов – MS Windows NT.

В 80-е годы были приняты основные стандарты на коммуникационные технологии для локальных сетей: в 1980 г. – Ethernet, в 1985 г. – Token Ring, в конце 80-х –FDDI (Fiber Distributed Data Interface), распределенный интерфейс передачи данных по волоконно-оптическим каналам, двойное кольцо с маркером. Это позволило обеспечить совместимость сетевых ОС на нижних уровнях, а также стандартизировать операционные системы с драйверами сетевых адаптеров.

Для ПК применялись не только специально разработанные для них ОС (MS-Dos, NetWare, OS/2), но и адаптировались уже существующие ОС, в частности UNIX. Наиболее известной системой этого типа была версия UNIX компании Santa Cruz Operation (SCO UNIX).

В 90-е годы практически все операционные системы, занимающие заметное место на рынке, стали сетевыми. Сетевые функции встраиваются в ядро ОС, являясь ее неотъемлемой

частью. В ОС используются средства мультиплексирования нескольких стеков протоколов, за счет которого компьютеры могут поддерживать одновременную работу с разнородными серверами и клиентами. Появились специализированные ОС, например, сетевая ОС IOS компании Cisco System, работающая в маршрутизаторах. Во второй половине 90-х годов все производители ОС усилили поддержку средств работы с интерфейсами. Кроме стека протоколов TCP/IP в комплект поставки начали включать утилиты, реализующие популярные сервисы Интернета: telnet, ftp, DNS, Web и др.

Особое внимание уделялось в последнем десятилетии и уделяется в настоящее время корпоративным сетевым операционным системам. Это одна из наиболее важных задач в обозримом будущем. Корпоративные ОС должны хорошо и устойчиво работать в крупных сетях, которые характерны для крупных организаций (предприятий, банков и т.п.), имеющих отделения во многих городах и, возможно, в разных странах. Корпоративная ОС должна без проблем взаимодействовать с ОС разного типа и работать на различных аппаратных платформах. Сейчас определились лидеры в классе корпоративных ОС – это MS Windows 2000/2003, UNIX и Linux-системы, а также Novell NetWare 6.5.

1.4. **Назначение, состав и функции ОС**

В настоящее время существует большое количество различных типов операционных систем, отличающихся областями применения, аппаратными платформами, способами реализации и др. Назначение операционных систем можно разделить на четыре основные составляющие [5, 10, 13.].

1. Организация (обеспечение) удобного интерфейса между приложениями и пользователями, с одной стороны, и аппаратурой компьютера – с другой. Вместо реальной аппаратуры компьютера ОС представляет пользователю расширенную виртуальную машину, с которой удобнее работать и которую легче программировать. Вот список основных сервисов, предоставляемых типичными операционными системами.

Разработка программ: ОС представляет программисту разнообразные инструменты разработки приложений: редакторы, отладчики и т.п. Ему не обязательно знать, как функционируют различные электронные и электромеханические узлы и устройства компьютера. Часто пользователь не знает даже системы команд процессора, поскольку он может обойтись мощными высокоуровневыми функциями, которые представляет ОС.

Исполнение программ. Для запуска программы нужно выполнить ряд действий: загрузить в основную память программу и данные, инициализировать устройства ввода-вывода и файлы, подготовить другие ресурсы. ОС выполняет всю эту рутинную работу вместо пользователя.

Доступ к устройствам ввода-вывода. Для управления каждым устройством используется свой набор команд. ОС предоставляет пользователю единообразный интерфейс, который скрывает все эти детали и обеспечивает программисту доступ к устройствам ввода-вывода с помощью простых команд чтения и записи. Если бы программист работал непосредственно с аппаратурой компьютера, то для организации, например, чтения блока данных с диска ему пришлось бы использовать более десятка команд с указанием множества параметров. После завершения обмена программист должен был бы предусмотреть еще более сложный анализ результата выполненной операции.

Контролируемый доступ к файлам. При работе с файлами управление со стороны ОС предполагает не только глубокий учет природы устройства ввода-вывода, но и знание структур данных, записанных в файлах. Многопользовательские ОС, кроме того, обеспечивают механизм защиты при обращении к файлам.

Системный доступ. ОС управляет доступом к совместно используемой или общедоступной вычислительной системе в целом, а также к отдельным системным ресурсам. Она обеспечивает защиту ресурсов и данных от несанкционированного использования и разрешает конфликтные ситуации.

Обнаружение ошибок и их обработка. При работе компьютерной системы могут происходить разнообразные сбои за счет внутренних и внешних ошибок в аппаратном обеспечении, различного рода программных ошибок (переполнение, попытка обращения к ячейке памяти, доступ к которой запрещен и др.). В каждом случае ОС выполняет действия, минимизирующие влияние ошибки на работу приложения (от простого сообщения об ошибке до аварийной остановки программы).

Учет использования ресурсов. Хорошая ОС имеет средства учета использования различных ресурсов и отображения параметров производительности вычислительной системы. Эта информация важна для настройки (оптимизации) вычислительной системы с целью повышения ее производительности.

В результате реальная машина, способная выполнить только небольшой набор элементарных действий (машинных команд), с помощью операционной системы превращается в виртуальную машину, выполняющую широкий набор гораздо более мощных функций. Виртуальная машина тоже управляется командами, но уже командами более высокого уровня, например: удалить файл с определенным именем, запустить на выполнение прикладную программу, повысить приоритет задачи, вывести текст файла на печать и т.д. Таким образом, назначение ОС состоит в предоставлении пользователю (программисту) некоторой расширенной виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальный компьютер, систему или сеть.

2. Организация эффективного использования ресурсов компьютера. ОС не только представляет пользователям и программистам удобный интерфейс к аппаратным средствам компьютера, но и является своеобразным диспетчером ресурсов компьютера. К числу основных ресурсов современных вычислительных систем относятся процессоры, основная память, таймеры, наборы данных, диски, накопители на МЛ, принтеры, сетевые устройства и др. Эти ресурсы распределяются операционной системой между выполняемыми программами. В отличие от программы, которая является статическим объектом, выполняемая программа – это динамический объект, он называется процессом и является базовым понятием современных ОС.

Управление ресурсами вычислительной системы с целью наиболее эффективного их использования является вторым назначением операционной системы. Критерии эффективности, в соответствии с которыми ОС организует управление ресурсами компьютера, могут быть различными. Например, в одних системах важен такой критерий, как пропускная способность вычислительной систем, в других – время ее реакции. Зачастую ОС должны удовлетворять нескольким, противоречащим друг другу критериям, что доставляет разработчикам серьезные трудности.

Управление ресурсами включает решение ряда общих, не зависящих от типа ресурса задач:

1. планирование ресурса – определение, какому процессу, когда и в каком качестве (если ресурс может выделяться частями) следует выделить данный ресурс;
2. удовлетворение запросов на ресурсы – выделение ресурса процессам;
3. отслеживание состояния и учет использования ресурса – поддержание оперативной информации о занятости ресурса и распределенной его доли;

4. разрешение конфликтов между процессами, претендующими на один и тот же ресурс.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, особенности которых, в конечном счете, определяют облик ОС в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Таким образом, управление ресурсами составляют важное назначение ОС. В отличие от функций расширенной виртуальной машины большинство функций управления ресурсами выполняются операционной системой автоматически и прикладному программисту недоступны.

3. Облегчение процессов эксплуатации аппаратных и программных средств вычислительной системы. Ряд операционных систем имеет в своем составе наборы служебных программ, обеспечивающие резервное копирование, архивацию данных, проверку, очистку и дефрагментацию дисковых устройств и др.

Кроме того, современные ОС имеют достаточно большой набор средств и способов диагностики и восстановления работоспособности системы. Сюда относятся:

- диагностические программы для выявления ошибок в конфигурации ОС;
- средства восстановления последней работоспособной конфигурации;
- средства восстановления поврежденных и пропавших системных файлов и др.

Следует отметить еще одно назначение ОС.

4. Возможность развития. Современные ОС организуются таким образом, что допускают эффективную разработку, тестирование и внедрение новых системных функций, не прерывая процесса нормального функционирования вычислительной системы. Большинство операционных систем постоянно развиваются (нагляден пример Windows). Происходит это в силу следующих причин.

Обновление и возникновение новых видов аппаратного обеспечения. Например, ранние версии ОС UNIX и OS/2 не использовали механизмы страничной организации памяти (что это такое, мы рассмотрим позже), потому, что они работали на машинах, не обеспеченных соответствующими аппаратными средствами.

Новые сервисы. Для удовлетворения пользователей или нужд системных администраторов ОС должны постоянно предоставлять новые возможности. Например, может потребоваться добавить новые инструменты для контроля или оценки производительности, новые средства ввода-вывода данных (речевой ввод). Другой пример – поддержка новых приложений, использующих окна на экране дисплея.

Исправления. В каждой ОС есть ошибки. Время от времени они обнаруживаются и исправляются. Отсюда постоянные появления новых версий и редакций ОС. Необходимость регулярных изменений накладывает определенные требования на организацию операционных систем. Очевидно, что эти системы (как, впрочем, и другие сложные программы системы) должны иметь модульную структуру с четко определенными межмодульными связями (интерфейсами). Важную роль играет хорошая и полная документированность системы.

Перейдем к рассмотрению состава компонентов и функций ОС. Современные операционные системы содержат сотни и тысячи модулей (например, W2000 содержит 29 млн строк исходного кода на языке С). Функции ОС обычно группируются либо в соответствии с типами локальных ресурсов, которыми управляет ОС, либо в соответствии со специфическими задачами, применимыми ко всем ресурсам. Совокупности модулей, выполняющих такие группы функций, образуют подсистемы операционной системы.

Наиболее важными подсистемами управления ресурсами являются подсистемы управления процессами, памятью, файлами и внешними устройствами, а подсистемами, общими для всех ресурсов, являются подсистемы пользовательского интерфейса, защиты данных и администрирования.

Управление процессами. Подсистема управления процессами непосредственно влияет на функционирование вычислительной системы. Для каждой выполняемой программы ОС организует один или более процессов. Каждый такой процесс представляется в ОС информационной структурой (таблицей, дескриптором, контекстом процессора), содержащей данные о потребностях процесса в ресурсах, а также о фактически выделенных ему ресурсах (область оперативной памяти, количество процессорного времени, файлы, устройства ввода-вывода и др.). Кроме того, в этой информационной структуре хранятся данные, характеризующие историю пребывания процесса в системе: текущее состояние (активное или заблокированное), приоритет, состояние регистров, программного счетчика и др.

В современных мультипрограммных ОС может существовать одновременно несколько процессов, порожденных по инициативе пользователей и их приложений, а также инициированных ОС для выполнения своих функций (системные процессы). Поскольку процессы могут одновременно претендовать на одни и те же ресурсы, подсистема управления процессами планирует очередность выполнения процессов, обеспечивает их необходимыми ресурсами, обеспечивает взаимодействие и синхронизацию процессов.

Управление памятью. Подсистема управления памятью производит распределение физической памяти между всеми существующими в системе процессами, загрузку и удаление программных кодов и данных процессов в отведенные им области памяти, настройку адресно-зависимых частей кодов процесса на физические адреса выделенной области, а также защиту областей памяти каждого процесса. Стратегия управления памятью складывается из стратегий выборки, размещения и замещения блока программы или данных в основной памяти. Соответственно используются различные алгоритмы, определяющие, когда загрузить очередной блок в память (по запросу или с упреждением), в какое место памяти его поместить и какой блок программы или данных удалить из основной памяти, чтобы освободить место для размещения новых блоков.

Одним из наиболее популярных способов управления памятью в современных ОС является виртуальная память. Реализация механизма виртуальной памяти позволяет программисту считать, что в его распоряжении имеется однородная оперативная память, объем которой ограничивается только возможностями адресации, предоставляемыми системой программирования.

Важная функция управления памятью – защита памяти. Нарушения защиты памяти связаны с обращениями процессов к участкам памяти, выделенной другим процессам прикладных программ или программ самой ОС. Средства защиты памяти должны пресекать такие попытки доступа путем аварийного завершения программы-нарушителя.

Управление файлами. Функции управления файлами сосредоточены в файловой системе ОС. Операционная система виртуализирует отдельный набор данных, хранящихся на внешнем накопителе, в виде файла – простой неструктурированной последовательности байтов, имеющих символьное имя. Для удобства работы с данными файлы группируются в каталоги, которые, в свою очередь, образуют группы – каталоги более высокого уровня. Файловая система преобразует символьные имена файлов, с которыми работает пользователь или программист, в физические адреса данных на дисках, организует совместный доступ к файлам, защищает их от несанкционированного доступа.

Управление внешними устройствами. Функции управления внешними устройствами возлагаются на подсистему управления внешними устройствами, называемую также подсистемой ввода-вывода. Она является интерфейсом между ядром компьютера и всеми подключенными к нему устройствами. Спектр этих устройств очень обширен (принтеры, сканеры, мониторы, модемы, манипуляторы, сетевые адаптеры, АЦП разного рода и др.), сотни моделей этих устройств отличаются набором и последовательностью команд, используемых для обмена информацией с процессором и другими деталями.

Программа, управляющая конкретной моделью внешнего устройства и учитывающая все его особенности, называется драйвером. Наличие большого количества подходящих драйверов во многом определяет успех ОС на рынке. Созданием драйверов занимаются как разработчики ОС, так и компании, выпускающие внешние устройства. ОС должна поддерживать четко определенный интерфейс между драйверами и остальными частями ОС. Тогда разработчики компаний-производителей устройств ввода-вывода могут поставлять вместе со своими устройствами драйверы для конкретной операционной системы.

Защита данных и администрирование. Безопасность данных вычислительной системы обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок программного обеспечения, а также средствами защиты от несанкционированного доступа. Для каждого пользователя системы обязательна процедура логического входа, в процессе которой ОС убеждается, что в систему входит пользователь, разрешенный административной службой. Администратор вычислительной системы определяет и ограничивает возможности пользователей в выполнении тех или иных действий, т.е. определяет их права по обращению и использованию ресурсов системы.

Важным средством защиты являются функции аудита ОС, заключающегося в фиксации всех событий, от которых зависит безопасность системы. Поддержка отказоустойчивости вычислительной системы реализуется на основе резервирования (дисковые RAID-массивы, резервные принтеры и другие устройства, иногда резервирование центральных процессоров, в ранних ОС – дуальные и дуплексные системы, системы с мажоритарным органом и др.). Вообще обеспечение отказоустойчивости системы – одна из важнейших обязанностей системного администратора, который для этого использует ряд специальных средств и инструментов [7, 10, 13].

Интерфейс прикладного программирования. Прикладные программисты используют в своих приложениях обращения к операционной системе, когда для выполнения тех или иных действий им требуется особый статус, которым обладает только ОС. Возможности операционной системы доступны программисту в виде набора функций, который называется интерфейсом прикладного программирования (Application Programming Interface, API). Приложения обращаются к функциям API с помощью системных вызовов. Способ, которым приложение получает услуги операционной системы, очень похож на вызов подпрограмм.

Способ реализации системных вызовов зависит от структурной организации ОС, особенностей аппаратной платформы и языка программирования.

В ОС UNIX системные вызовы почти идентичны библиотечным процедурам. Ситуация в Windows иная (более подробно это рассмотрим далее).

Пользовательский интерфейс. ОС обеспечивает удобный интерфейс не только для прикладных программ, но и для пользователя (программиста, администратора). В ранних ОС интерфейс сводился к языку управления заданиями и не требовал терминала. Команды языка управления заданиями набивались на перфокарты, а результаты выполнения задания выводились на печатающее устройство.

Современные ОС поддерживают развитые функции пользовательского интерфейса для интерактивной работы за терминалами двух типов: алфавитно-цифрового и графического. При работе за алфавитно-цифровым терминалом пользователь имеет в своем распоряжении систему команд, развитость которой отражает функциональные возможности данной ОС. Обычно командный язык ОС позволяет запускать и останавливать приложения, выполнять различные операции с каталогами и файлами, получать информацию о состоянии ОС, администрировать систему. Команды могут вводиться не только в интерактивном режиме с терминала, но и считываться из так называемого командного файла, содержащего некоторую последовательность команд.

Программный модуль ОС, ответственный за чтение отдельных команд или же последовательности команд из командного файла, иногда называют командным интерпретатором (в MS-DOS – командным процессором).

Вычислительные системы, управляемые из командной строки, например UNIX-системы, имеют командный интерпретатор, называемый оболочкой (Shell). Она, собственно, не входит в состав ОС, но пользуется многими функциями операционной системы. Когда какой-либо пользователь входит в систему, запускается оболочка. Стандартным терминалом для нее является монитор с клавиатурой. Оболочка начинает работу с печати приглашения (prompt) – знака доллара (или иного знака), говорящего пользователю, что оболочка ожидает ввода команды (аналогично управляется MS-DOS). Если теперь пользователь напечатает какую-либо команду, оболочка создает системный вызов и ОС выполнит эту команду. После завершения оболочка опять печатает приглашение и пытается прочесть следующую входную строку.

Ввод команд может быть упрощен, если операционная система поддерживает графический пользовательский интерфейс. В этом случае пользователь выбирает на экране нужный пункт меню или графический символ (так это происходит, например, в ОС Windows).

1.5. Архитектура операционной системы

Под архитектурой операционной системы понимают структурную и функциональную организацию ОС на основе некоторой совокупности программных модулей. В состав ОС входят исполняемые и объектные модули стандартных для данной ОС форматов, программные модули специального формата (например, загрузчик ОС, драйверы ввода-вывода), конфигурационные файлы, файлы документации, модули справочной системы и т.д.

На архитектуру ранних операционных систем обращалось мало внимания: во-первых, ни у кого не было опыта в разработке больших программных систем, а во-вторых, проблема взаимозависимости и взаимодействия модулей недооценивалась. В подобных монолитных ОС почти все процедуры могли вызывать одна другую. Такое отсутствие структуры было несовместимо с расширением операционных систем. Первая версия ОС OS/360 была создана коллективом из 5000 человек за 5 лет и содержала более 1 млн строк кода. Разработанная несколько позже операционная система Mastics содержала к 1975 году уже 20 млн строк [17]. Стало ясно, что разработка таких систем должна вестись на основе модульного программирования.

Большинство современных ОС представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой унифицированной архитектуры ОС не существует, но известны универсальные подходы к структурированию ОС. Принципиально важными универсальными подходами к разработке архитектуры ОС являются [5, 10, 13, 17]:

- модульная организация;
- функциональная избыточность;

- функциональная избирательность;
- параметрическая универсальность;
- концепция многоуровневой иерархической вычислительной системы, по которой ОС представляется многослойной структурой;
- разделение модулей на две группы по функциям: ядро – модули, выполняющие основные функции ОС, и модули, выполняющие вспомогательные функции ОС;
- разделение модулей ОС на две группы по размещению в памяти вычислительной системы: резидентные, постоянно находящиеся в оперативной памяти, и транзитные, загружаемые в оперативную память только на время выполнения своих функций;
- реализация двух режимов работы вычислительной системы: привилегированного режима (режима ядра – Kernel mode), или режима супервизора (supervisor mode), и пользовательского режима (user mode), или режима задачи (task mode);
- ограничение функций ядра (а следовательно, и количества модулей ядра) до минимального количества необходимых самых важных функций.

Первые ОС разрабатывались как монолитные системы без четко выраженной структуры (рис. 1.2).

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

Однако даже такие монолитные системы могут быть немного структурированными. При обращении к системным вызовам, поддерживаемым ОС, параметры помещаются в строго определенные места, такие как регистры или стек, а затем выполняется специальная команда прерывания, известная как вызов ядра или вызов супервизора. Эта команда переключает машину из режима пользователя в режим ядра, называемый также режимом супервизора, и передает управление ОС. Затем ОС проверяет параметры вызова, для того чтобы определить, какой системный вызов должен быть выполнен. После этого ОС индексирует таблицу, содержащую ссылки на процедуры, и вызывает соответствующую процедуру.

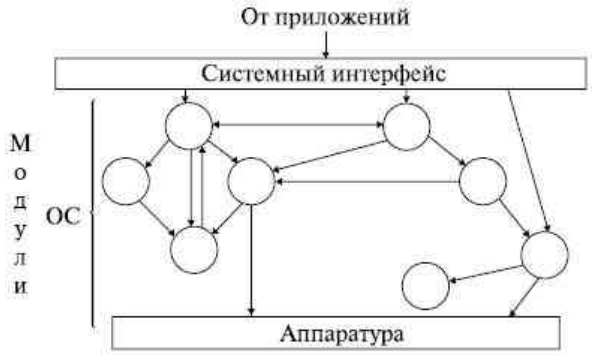

Рис. 1.2. Монолитная архитектура

Такая организация ОС предполагает следующую структуру [13]:

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

В этой модели для каждого системного вызова имеется одна сервисная процедура. Утилиты выполняют функции, которые нужны нескольким сервисным процедурам. Это деление процедур на три слоя показано на рис. 1.3.

Классической считается архитектура ОС, основанная на концепции иерархической многоуровневой машины, привилегированном ядре и пользовательском режиме работы транзитных модулей. Модули ядра выполняют базовые функции ОС: управление процессами, памятью, устройствами ввода-вывода и т.п. Ядро составляет сердцевину ОС, без которой она является полностью неработоспособной и не может выполнить ни одну из своих функций. В ядре решаются внутрисистемные задачи организации вычислительного процесса, недоступные для приложения.

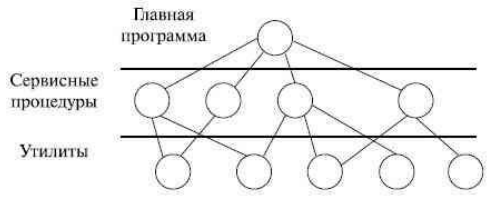

Рис. 1.3. Структурированная архитектура

Особый 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, открытие и чтение файла, получение системного времени, вывода информации на дисплей и т.д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования – API (Application Programming Interface).

Для обеспечения высокой скорости работы ОС модули ядра (по крайней мере, большая их часть) являются резидентными и работают в привилегированном режиме (Kernel mode). Этот режим, во-первых, должен обезопасить работу самой ОС от вмешательства приложений, и, во-вторых, должен обеспечить возможность работы модулей ядра с полным набором машинных инструкций, позволяющих собственно ядру выполнять управление ресурсами компьютера, в частности, переключение процессора с задачи на задачу, управлением устройствами ввода-вывода, распределением и защитой памяти и др.

Остальные модули ОС выполняют не столь важные функции, как ядро, и являются транзитными. Например, это могут быть программы архивирования данных, дефрагментации диска, сжатия дисков, очистки дисков и т.п.

Вспомогательные модули обычно подразделяются на группы:

- утилиты – программы, выполняющие отдельные задачи управления и сопровождения вычислительной системы;
- системные обрабатывающие программы – текстовые и графические редакторы (Paint, Imaging в Windows 2000), компиляторы и др.;
- программы предоставления пользователю дополнительных услуг (специальный вариант пользовательского интерфейса, калькулятор, игры, средства мультимедиа Windows 2000);
- библиотеки процедур различного назначения, упрощения разработки приложений, например, библиотека функций ввода-вывода, библиотека математических функций и т.п.

Эти модули ОС оформляются как обычные приложения, обращаются к функциям ядра посредством системных вызовов и выполняются в пользовательском режиме (user mode). В

этом режиме запрещается выполнение некоторых команд, которые связаны с функциями ядра ОС (управление ресурсами, распределение и защита памяти и т.п.).

В концепции многоуровневой (многослойной) иерархической машины структура ОС также представляется рядом слоев. При такой организации каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс. На основе этих функций следующий верхний по иерархии слой строит свои функции – более сложные и более мощные и т.д. Такая организация системы существенно упрощает ее разработку, т.к. позволяет сначала "сверху вниз" определить функции слоев и межслойные интерфейсы, а при детальной реализации, двигаясь "снизу вверх", – наращивать мощность функции слоев. Кроме того, модули каждого слоя можно изменять без необходимости изменений в других слоях (но не меняя межслойных интерфейсов!).

Многослойная структура ядра ОС может быть представлена, например, вариантом, показанным на рис. 1.4.

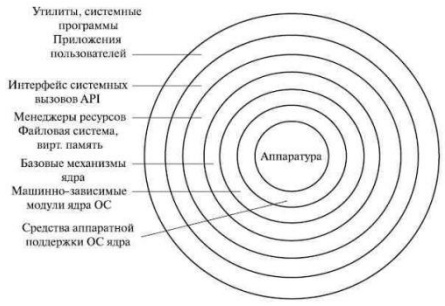

Рис. 1.4. Многослойная структура ОС

В данной схеме выделены следующие слои.

1. Средства аппаратной поддержки ОС. Значительная часть функций ОС может выполняться аппаратными средствами [10]. Чисто программные ОС сейчас не существуют. Как правило, в современных системах всегда есть средства аппаратной поддержки ОС, которые прямо участвуют в организации вычислительных процессов. К ним относятся: система прерываний, средства поддержки привилегированного режима, средства поддержки виртуальной памяти, системный таймер, средства переключения контекстов процессов (информация о состоянии процесса в момент его приостановки), средства защиты памяти и др.
2. Машинно-зависимые модули ОС. Этот слой образует модули, в которых отражается специфика аппаратной платформы компьютера. Назначение этого слоя – "экранирование" вышележащих слоев ОС от особенностей аппаратуры (например, Windows 2000 – это слой HAL (Hardware Abstraction Layer), уровень аппаратных абстракций).
3. Базовые механизмы ядра. Этот слой модулей выполняет наиболее примитивные операции ядра: программное переключение контекстов процессов, диспетчерскую прерываний, перемещение страниц между основной памятью и диском и т.п. Модули этого слоя не принимают решений о распределении ресурсов, а только обрабатывают решения, принятые модулями вышележащих уровней. Поэтому их часто называют исполнительными механизмами для модулей верхних слоев ОС.
4. Менеджеры ресурсов. Модули этого слоя выполняют стратегические задачи по управлению ресурсами вычислительной системы. Это менеджеры (диспетчеры) процессов ввода-вывода, оперативной памяти и файловой системы. Каждый

менеджер ведет учет свободных и используемых ресурсов и планирует их распределение в соответствии запросами приложений.

5. Интерфейс системных вызовов. Это верхний слой ядра ОС, взаимодействующий с приложениями и системными утилитами, он образует прикладной программный интерфейс ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной компактной форме, без указания деталей их физического расположения.

Повышение устойчивости ОС обеспечивается переходом ядра в привилегированный режим. При этом происходит некоторое замедление выполнения системных вызовов. Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению – обратное переключение. За счет этого возникает дополнительная задержка в обработке системного вызова (рис. 1.5). Однако такое решение стало классическим и используется во многих ОС (UNIX, VAX, VMS, IBM OS/390, OS/2 и др.).

Рис. 1.5. Обработка системного вызова

Многослойная классическая многоуровневая архитектура ОС не лишена своих проблем. Дело в том, что значительные изменения одного из уровней могут иметь трудно предвидимое влияние на смежные уровни. Кроме того, многочисленные взаимодействия между соседними уровнями усложняют обеспечение безопасности. Поэтому, как альтернатива классическому варианту архитектуры ОС, часто используется микроядерная архитектура ОС.

Суть этой архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. Микроядро защищено от остальных частей ОС и приложений. В его состав входят машинно-зависимые модули, а также модули, выполняющие базовые механизмы обычного ядра. Все остальные более высокоуровневые функции ядра оформляются как модули, работающие в пользовательском режиме. Так, менеджеры ресурсов, являющиеся неотъемлемой частью обычного ядра, становятся "периферийными" модулями, работающими в пользовательском режиме. Таким образом, в архитектуре с микроядром традиционное расположение уровней по вертикали заменяется горизонтальным. Это можно представить, как показано на рис. 1.6.

Внешние по отношению к микроядру компоненты ОС реализуются как обслуживающие процессы. Между собой они взаимодействуют как равноправные партнеры с помощью обмена сообщениями, которые передаются через микроядро. Поскольку назначением этих компонентов ОС является обслуживание запросов приложений пользователей, утилит и системных обрабатывающих программ, менеджеры ресурсов, вынесенные в пользовательский режим, называются серверами ОС, т.е. модулями, основным назначением которых является обслуживание запросов локальных приложений и других модулей ОС.

Рис. 1.6. Переход к микроядерной архитектуре

Схематично механизм обращений к функциям ОС, оформленным в виде серверов, выглядит, как показано на рис. 1.7.

Рис. 1.7. Клиент-серверная архитектура

Схема смены режимов при выполнении системного вызова в ОС с микроядерной архитектурой выглядит, как показано на рис. 1.8. Из рисунка ясно, что выполнение системного вызова сопровождается четырьмя переключениями режимов (4 t), в то время как в классической архитектуре – двумя. Следовательно, производительность ОС с микроядерной архитектурой при прочих равных условиях будет ниже, чем у ОС с классическим ядром.

Рис. 1.8. Обработка системного вызова в микроядерной архитектуре

В то же время признаны следующие достоинства микроядерной архитектуры [17]:

- единообразные интерфейсы;
- простота расширяемости;
- высокая гибкость;
- возможность переносимости;
- высокая надежность;
- поддержка распределенных систем;
- поддержка объектно-ориентированных ОС.

По многим источникам вопрос масштабов потери производительности в микроядерных ОС является спорным. Многое зависит от размеров и функциональных возможностей микроядра. Избирательное увеличение функциональности микроядра приводит к снижению количества переключений между режимами системы, а также переключений адресных пространств процессов.

Может быть, это покажется парадоксальным, но есть и такой подход к микроядерной ОС, как уменьшение микроядра.

Для возможности представления о размерах микроядер операционных систем в ряде источников [17] приводятся такие данные:

- типичное микроядро первого поколения – 300 Кбайт кода и 140 интерфейсов системных вызовов;
- микроядро ОС L4 (второе поколение) – 12 Кбайт кода и 7 интерфейсов системных вызовов.
-

В современных операционных системах различают следующие виды ядер.

1. Наноядро (НЯ). Крайне упрощённое и минимальное ядро, выполняет лишь одну задачу – обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки посылает информацию о результатах обработки вышележащему программному обеспечению. НЯ используются для виртуализации аппаратного обеспечения реальных компьютеров или для реализации механизма гипервизора.
2. Микроядро (МЯ) предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием. Большая часть работы осуществляется с помощью специальных пользовательских процессов, называемых сервисами. В микроядерной операционной системе можно, не прерывая ее работы, загружать и выгружать новые драйверы, файловые системы и т. д. Микроядерными являются ядра ОС Minix и GNU Hurd и ядро систем семейства BSD. Классическим примером микроядерной системы является Symbian OS. Это пример распространенной и отработанной микроядерной (а начиная с версии Symbian OS v8.1, и наноядерной) операционной системы.
3. Экзоядро (ЭЯ) предоставляет лишь набор сервисов для взаимодействия между приложениями, а также необходимый минимум функций, связанных с защитой: выделение и высвобождение ресурсов, контроль прав доступа и т. д. ЭЯ не занимается предоставлением абстракций для физических ресурсов – эти функции выносятся в библиотеку пользовательского уровня (так называемую libOS). В отличие от микроядра ОС, базирующиеся на ЭЯ, обеспечивают большую эффективность за счет отсутствия необходимости в переключении между процессами при каждом обращении к оборудованию.
4. Монолитное ядро (МнЯ) предоставляет широкий набор абстракций оборудования. Все части ядра работают в одном адресном пространстве. МнЯ требуют перекомпиляции при изменении состава оборудования. Компоненты операционной системы являются не самостоятельными модулями, а составными частями одной программы. МнЯ более производительно, чем микроядро, поскольку работает как один большой процесс. МнЯ является большинство Unix-систем и Linux. Монолитность ядер усложняет отладку, понимание кода ядра, добавление новых функций и возможностей, удаление ненужного, унаследованного от предыдущих

версий кода. "Разбухание" кода монолитных ядер также повышает требования к объёму оперативной памяти.

5. Модульное ядро (Мод. Я) – современная, усовершенствованная модификация архитектуры МЯ. В отличие от "классических" МнЯ, модульные ядра не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера. Вместо этого они предоставляют тот или иной механизм подгрузки модулей, поддерживающих то или иное аппаратное обеспечение (например, драйверов). Подгрузка модулей может быть как динамической, так и статической (при перезагрузке ОС после переконфигурирования системы). Мод. Я удобнее для разработки, чем традиционные монолитные ядра. Они предоставляют программный интерфейс (API) для связывания модулей с ядром, для обеспечения динамической подгрузки и выгрузки модулей. Не все части ядра могут быть сделаны модулями. Некоторые части ядра всегда обязаны присутствовать в оперативной памяти и должны быть жёстко "вшиты" в ядро.
6. Гибридное ядро (ГЯ) – модифицированные микроядра, позволяющие для ускорения работы запускать "несущественные" части в пространстве ядра. Имеют "гибридные" достоинства и недостатки. Примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные 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 использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.
7. Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре Windows NT. Хотя Windows NT часто называют микроядерной операционной системой, это не совсем так. Микроядро NT слишком велико (более 1 Мбайт), чтобы носить приставку "микро". Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром.

1.6. Классификация операционных систем

Все многообразие существующих (и ныне не использующихся) ОС можно классифицировать по множеству различных признаков. Остановимся на основных классификационных признаках.

1. По назначению ОС делятся на универсальные и специализированные. Специализированные ОС, как правило, работают с фиксированным набором программ (функциональных задач). Применение таких систем обусловлено невозможностью использования универсальной ОС по соображениям эффективности, надежности, защищенности и т.п., а также вследствие специфики решаемых задач [10].

Универсальные ОС рассчитаны на решение любых задач пользователей, но, как правило, форма эксплуатации вычислительной системы может предъявлять особые требования к ОС, т.е. к элементам ее специализации.

2. По способу загрузки можно выделить загружаемые ОС (большинство) и системы, постоянно находящиеся в памяти вычислительной системы. Последние, как правило, специализированные и используются для управления работой специализированных устройств (например, в БЦВМ баллистической ракеты или спутника, научных приборах, автоматических устройствах различного назначения и др.).
3. По особенностям алгоритмов управления ресурсами. Главным ресурсом системы является процессор, поэтому дадим классификацию по алгоритмам управления процессором, хотя можно, конечно, классифицировать ОС по алгоритмам управления памятью, устройствами ввода-вывода и.т.д.
    - Поддержка многозадачности (многопрограммности). По числу одновременно выполняемых задач ОС делятся на 2 класса: однопрограммные (однозадачные) – например, MS-DOS, MSX, и многопрограммные (многозадачные) – например, ОС ЕС ЭВМ, OS/360, OS/2, UNIX, Windows разных версий.

Однопрограммные ОС предоставляют пользователю виртуальную машину, делая более простым и удобным процесс взаимодействия пользователя с компьютером. Они также имеют средства управления файлами, периферийными устройствами и средства общения с пользователем. Многозадачные ОС, кроме того, управляют разделением совместно используемых ресурсов (процессор, память, файлы и т.д.), это позволяет значительно повысить эффективность вычислительной системы.

- Поддержка многопользовательского режима. По числу одновременно работающих пользователей ОС делятся: на однопользовательские (MS-DOS, Windows 3x, ранние версии OS/2) и многопользовательские (UNIX, Windows NT/2000/2003/XP/Vista).

Главное отличие многопользовательских систем от однопользовательских – наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует заметить, что может быть однопользовательская мультипрограммная система.

- Виды многопрограммной работы. Специфику ОС во многом определяет способ распределения времени между несколькими одновременно существующими в системе процессами (или потоками). По этому признаку можно выделить 2 группы алгоритмов: не вытесняющая многопрограммность (Windows 3.x, NetWare) и вытесняющая многопрограммность (Windows 2000/2003/XP, OS/2, Unix).

В первом случае активный процесс выполняется до тех пор, пока он сам не отдает управление операционной системе. Во втором случае решение о переключении процессов принимает операционная система. Возможен и такой режим многопрограммности, когда ОС разделяет процессорное время между отдельными ветвями (потоками, волокнами) одного процесса.

- Многопроцессорная обработка. Важное свойство ОС – отсутствие или наличие средств поддержки многопроцессорной обработки. По этому признаку можно выделить ОС без поддержки мультипроцессирования (Windows 3.x, Windows 95) и с поддержкой мультипроцессирования (Solaris, OS/2, UNIX, Windows NT/2000/2003/XP).

Многопроцессорные ОС классифицируются по способу организации вычислительного процесса на асимметричные ОС (выполняются на одном процессоре, распределяя прикладные задачи по остальным процессорам) и симметричные ОС (децентрализованная система).

4. По области использования и форме эксплуатации. Обычно здесь выделяют три типа в соответствии с использованными при их разработке критериями эффективности:
    - системы пакетной обработки (OS/360, ОС ЕС);
    - системы разделения времени (UNIX, VMS);
    - системы реального времени (QNX, RT/11).

Первые предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Критерий создания таких ОС – максимальная пропуская способность при хорошей загрузке всех ресурсов компьютера. В таких системах пользователь отстранен от компьютера.

Системы разделения времени обеспечивают удобство и эффективность работы пользователя, который имеет терминал и может вести диалог со своей программой.

Системы реального времени предназначены для управления техническими объектами (станок, спутник, технологический процесс, например доменный и т.п.), где существует предельное время на выполнение программ, управляющих объектом.

5. По аппаратной платформе (типу вычислительной техники), для которой они предназначаются, операционные системы делят на следующие группы.
    - Операционные системы для смарт-карт. Некоторые из них могут управлять только одной операцией, например, электронным платежом. Некоторые смарт-карты являются JAVA-ориентированным и содержат интерпретатор виртуальной машины JAVA. Апплеты JAVA загружаются на карту и выполняются JVM-интерпретатором. Некоторые из таких карт могут одновременно управлять несколькими апплетами JAVA, что приводит к многозадачности и необходимости планирования.
    - Встроенные операционные системы. Управляют карманными компьютерами (lialm OS, Windows CE – Consumer Electronics – бытовая техника), мобильными телефонами, телевизорами, микроволновыми печами и т.п.
    - Операционные системы для персональных компьютеров, например, Windows 9.x, Windows XP, Linux, Mac OSX и др.
    - Операционные системы мини-ЭВМ, например, RT-11 для PDP-11 – ОС реального времени, RSX-11 M для PDP-11 – ОС разделения времени, UNIX для PDP-7.
    - Операционные системы мэйнфреймов (больших машин), например, OS/390, происходящая от OS/360 (IBM). Обычно ОС мэйнфреймов предполагает одновременно три вида обслуживания: пакетную обработку, обработку транзакций (например, работа с БД, бронирование авиабилетов, процесс работы в банках) и разделение времени.
    - Серверные операционные системы, например, UNIX, Windows 2000, Linux. Область применения – ЛВС, региональные сети, Intranet, Internet.
    - Кластерные операционные системы. Кластер – слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений и представляющихся пользователю единой системной, например, Windows 2000 Cluster Server, Windows 2008 Server, Sun Cluster (базовая ОС – Solaris).

1.7. Эффективность и требования, предъявляемые к ОС

К операционным системам современных компьютеров предъявляется ряд требований. Главным требованием является выполнение основных функций эффективного управления ресурсами и обеспечения удобного интерфейса для пользователя и прикладных программ. Современная ОС должна поддерживать мультипрограммную обработку, виртуальную память, свопинг, развитый интерфейс пользователя (многооконный графический, аудио -, менюориентированный и т.д.), высокую степень защиты, удобство работы, а также выполнять многие другие необходимые функции и услуги. Кроме этих требований функциональной полноты, к ОС предъявляется ряд важных эксплуатационных требований.

1. Эффективность. Под эффективностью вообще любой технической (да и не только технической) системы понимается степень соответствия системы своему назначению, которая оценивается некоторым множеством показателей эффективности [10].

Поскольку ОС представляет собой сложную программную систему, она использует для собственных нужд значительную часть ресурсов компьютера. Часто эффективность ОС оценивают ее производительностью (пропускной способностью) – количеством задач пользователей, выполняемых за некоторый промежуток времени, временем реакции на запрос пользователя и др.

На все эти показатели эффективности ОС влияет много различных факторов, среди которых основными являются архитектура ОС, многообразие ее функций, качество программного кода, аппаратная платформа (компьютер) и др.

2. Надежность и отказоустойчивость. Операционная система должна быть, по меньшей мере, так же надежна, как компьютер, на котором она работает. Система должна быть защищена как от внутренних, так и от внешних сбоев и отказов. В случае ошибки в программе или аппаратуре система должна обнаружить ошибку и попытаться исправить положение или, по крайней мере, постараться свести к минимуму ущерб, нанесенный этой ошибкой пользователям.

Надежность и отказоустойчивость ОС, прежде всего, определяются архитектурными решениями, положенными в ее основу, а также отлаженностью программного кода (основные отказы и сбои ОС в основном обусловлены программными ошибками в ее модулях). Кроме того, важно, чтобы компьютер имел резервные дисковые массивы, источники бесперебойного питания и др., а также программную поддержку этих средств.

3. Безопасность (защищенность). Ни один пользователь не хочет, чтобы другие пользователи ему мешали. ОС должна защищать пользователей и от воздействия чужих ошибок, и от попыток злонамеренного вмешательства (несанкционированного доступа). С этой целью в ОС как минимум должны быть средства аутентификации – определения легальности пользователей, авторизации – предоставления легальным пользователям установленных им прав доступа к ресурсам, и аудита – фиксации всех потенциально опасных для системы событий.

Свойства безопасности особенно важны для сетевых ОС. В таких ОС к задаче контроля доступа добавляется задача защиты данных, передаваемых по сети.

4. Предсказуемость. Требования, которые пользователь может предъявить к системе, в большинстве случаев непредсказуемы. В то же время пользователь предпочитает, чтобы обслуживание не очень сильно менялось в течение предположительного времени. В частности, запуская свою программу в системе, пользователь должен иметь основанное на опыте работы с этой программной приблизительное представление, когда ему ожидать выдачи результатов.

5. Расширяемость. В отличие от аппаратных средств компьютера полезная жизнь операционных систем измеряется десятками лет. Примером может служить ОС UNIX, да и MS-DOS. Операционные системы изменяются со временем, как правило, за счет приобретения новых свойств, например, поддержки новых типов внешних устройств или новых сетевых технологий. Если программный код модулей ОС написан таким образом, что дополнения и изменения могут вноситься без нарушения целостности системы, то такую ОС называют расширяемой. Операционная система может быть расширяемой, если при ее создании руководствовались принципами модульности, функциональной избыточности, функциональной избирательности и параметрической универсальности.
6. Переносимость. В идеальном случае код ОС должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которые различаются не только типом процессора, но и способом организации всей аппаратуры компьютера) одного типа на аппаратную платформу другого типа. Переносимые ОС имеют несколько вариантов реализации для разных платформ, такое свойство ОС называется также многоплатформенностью. Достигается это свойство за счет того, что основная часть ОС пишется на языке высокого уровня (например С, С++ и др.) и может быть легко перенесена на другой компьютер (машинно-независимая часть), а некоторая меньшая часть ОС (программы ядра) является машинно-зависимой и разрабатывается на машинном языке другого компьютера.
7. Совместимость. Существует несколько "долгоживущих" популярных ОС (разновидности UNIX, MS-DOS, Windows3.x, Windows NT, OS/2), для которых наработана широкая номенклатура приложений. Для пользователя, переходящего с одной ОС на другую, очень привлекательна возможность – выполнить свои приложения в новой операционной системе. Если ОС имеет средства для выполнения прикладных программ, написанных для других операционных систем, то она совместима с этими системами. Следует различать совместимость на уровне двоичных кодов и совместимость на уровне исходных текстов. Кроме того, понятие совместимости включает также поддержку пользовательских интерфейсов других ОС.
8. Удобство. Средства ОС должны быть простыми и гибкими, а логика ее работы ясна пользователю. Современные ОС ориентированы на обеспечение пользователю максимально возможного удобства при работе с ними. Необходимым условием этого стало наличие у ОС графического пользовательского интерфейса и всевозможных мастеров – программ, автоматизирующих активизацию функций ОС, подключение периферийных устройств, установку, настройку и эксплуатацию самой ОС.
9. Масштабируемость. Если ОС позволяет управлять компьютером с различным числом процессов, обеспечивая линейное (или почти такое) возрастание производительности при увеличении числа процессоров, то такая ОС является масштабируемой. В масштабируемой ОС реализуется симметричная многопроцессорная обработка. С масштабируемостью связано понятие кластеризации – объединения в систему двух (и более) многопроцессорных компьютеров. Правда, кластеризация направлена не столько на масштабируемость, сколько на обеспечение высокой готовности системы.

Следует заметить, что в зависимости от области применения конкретной операционной системы может изменяться и состав предъявляемых к ней требований.

Производители могут предлагать свои ОС в различных, различающихся ценой и производительностью, конфигурациях. Например, Microsoft продает [10]:

- Windows 2003 Server (до 4-х процессоров) – для малого и среднего бизнеса;
- Windows 2003 Advanced Server (до 8 процессоров, 2-узловой кластер) – для средних и крупных предприятий;
- Windows 2003 DataCenter Server (16-32 процессора, 4-узловой кластер) – для особо крупных предприятий.

1.8. Совместимость и множественные прикладные среды

В то время как многие архитектурные особенности ОС непосредственно касаются только системных программистов, концепция множественных прикладных (операционных) средств непосредственно связана с нуждами конечных пользователей – возможностью операционной системы выполнять приложения, написанные для других операционных систем. Такое свойство операционной системы называется совместимостью.

Совместимость приложений может быть на двоичном уровне и на уровне исходных текстов [13]. Приложения обычно хранятся в ОС в виде исполняемых файлов, содержащих двоичные образы кодов и данных. Двоичная совместимость достигается в том случае, если можно взять исполняемую программу и запустить ее на выполнение в среде другой ОС.

Совместимость на уровне исходных текстов требует наличие соответствующего компилятора в составе программного обеспечения компьютера, на котором предполагается выполнить данное приложение, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция исходных текстов приложения в новый исполняемый модуль.

Совместимость на уровне исходных текстов важна в основном для разработчиков приложений, в распоряжении которых эти исходные тексты имеются. Но для конечных пользователей практическое значение имеет только двоичная совместимость, так как только в этом случае они могут использовать один и тот же продукт в различных операционных системах и на различных машинах.

Вид возможной совместимости зависит от многих факторов. Самый главный из них – архитектура процессора. Если процессор применяет тот же набор команд (возможно, с добавлениями, как в случае IBM PC: стандартный набор + мультимедиа + графика + потоковые) и тот же диапазон адресов, то двоичная совместимость может быть достигнута достаточно просто. Для этого необходимо соблюдение следующих условий:

- API, который использует приложение, должен поддерживаться данной ОС;
- внутренняя структура исполняемого фала приложения должна соответствовать структуре исполняемых фалов данной ОС.

Если процессоры имеют разную архитектуру, то, кроме перечисленных условий, необходимо организовать эмуляцию двоичного кода. Например, широко используется эмуляция команд процессора Intel на процессоре Motorola 680x0 компьютера Macintosh. Программный эмулятор в этом случае последовательно выбирает двоичную инструкцию процессора Intel и выполняет эквивалентную подпрограмму, написанную в инструкциях процессора Motorola. Так как у процессора Motorola нет в точности таких же регистров, флагов, внутреннего АЛУ и др., как в процессорах Intel, он должен также имитировать (эмулировать) все эти элементы с использованием своих регистров или памяти.

Это простая, но очень медленная работа, поскольку одна команда Intel выполняется значительно быстрее, чем эмулирующая ее последовательность команд процессора Motorola.

Выходом в таких случаях является применение так называемых прикладных программных сред или операционных сред. Одной из составляющих такой среды является набор функций интерфейса прикладного программирования API, который ОС предоставляет своим приложениям. Для сокращения времени на выполнение чужих программ прикладные среды имитируют обращение к библиотечным функциям.

Эффективность этого подхода связана с тем, что большинство сегодняшних программ работает под управлением GUI (графических интерфейсов пользователя) типа Windows, MAC или UNIX Motif, при этом приложения тратят 60-80% времени на выполнение функций GUI и других библиотечных вызовов ОС. Именно это свойство приложений позволяет прикладным средам компенсировать большие затраты времени, потраченные на покомандное эмулирование программ. Тщательно спроектированная программная прикладная среда имеет в своем составе библиотеки, имитирующие библиотеки GUI, но написанные на "родном" коде. Таким образом, достигается существенное ускорение выполнения программ с API другой операционной системы. Иначе такой подход называют трансляцией – для того, чтобы отличить его от более медленного процесса эмулирования по одной команде за раз.

Например, для Windows-программы, работающей на Macintosh, при интерпретации команд процессора Intel производительность может быть очень низкой. Но когда производится вызов функции GUI, открытие окна и др., модуль ОС, реализующий прикладную среду Windows, может перехватить этот вызов и перенаправить его на перекомпилированную для процессора Motorola 680x0 подпрограмму открытия окна. В результате на таких участках кода скорость работы программы может достичь (а, возможно, и превзойти) скорость работы на своём родном процессоре.

Чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь обеспечивать совместимость API. Концепции, положенные в основу разных ОС, могут входить в противоречия друг с другом. Например, в одной ОС приложению может быть разрешено управлять устройствами ввода-вывода, в другой – эти действия являются прерогативой ОС.

Каждая ОС имеет свои собственные механизмы защиты ресурсов, свои алгоритмы обработки ошибок и исключительных ситуаций, особую структуру процессора и схему управления памятью, свою семантику доступа к файлам и графический пользовательский интерфейс. Для обеспечения совместимости необходимо организовать бесконфликтное сосуществование в рамках одной ОС нескольких способов управления ресурсами компьютера.

Существуют различные варианты построения множественных прикладных сред, отличающиеся как особенностями архитектурных решений, так и функциональными возможностями, обеспечивающими разную степень переносимости приложений. Один из наиболее очевидных вариантов реализации множественных преклонных сред основывается на стандартной многоуровневой структуре ОС.

На рис. 1.9 ОС OS1 поддерживает кроме своих "родных" приложений приложения операционных систем OS2 и OS3. Для этого в её составе имеются специальные приложения, прикладные программные среды, которые транслируют интерфейсы "чужих" операционных систем API OS2 и API OS3 в интерфейс своей "родной" ОС – API OS1. Так, например, в случае если бы в качестве OS2 выступала ОС UNIX, а в качестве OS1 – OS/2, для выполнения системного вызова создания процесса fork () в UNIX-приложении программная среда должна обращаться к ядру операционной системы OS/2 с системным вызовом DOS ExecPgm ().

Рис. 1.9. Организация множественных прикладных сред

К сожалению, поведение почти всех функций, составляющих API одной ОС, как правило, существенно отличается от поведения соответствующих функций другой ОС. Например, чтобы функция создания процесса в OS/2 Dos ExecPgm () полностью соответствовала функции создания процесса fork () в UNIX-подобных системах, её нужно было бы изменить и прописать новую функциональность: поддержку возможности копирования адресного пространства родительского процесса в пространство процесса-потомка [17].

Еще один способ построения множественных прикладных сред основан на микроядерном подходе. При этом очень важно отметить базовое, общее для всех прикладных сред отличие механизмов операционной системы от специфических для каждой из прикладных сред высокоуровневых функций, решающих стратегические задачи. В соответствии с микроядерной архитектурой все функции ОС реализуются микроядром и серверами пользовательского режима. Важно, что прикладная среда оформляется в виде отдельного сервера пользовательского режима и не включает базовых механизмов.

Приложения, используя API , обращаются с системными вызовами к соответствующей прикладной среде через микроядро. Прикладная среда обрабатывает запрос, выполняет его (возможно, обращаясь для этого за помощью к базовым функциям микроядра) и отсылает приложению результат. В ходе выполнения запроса прикладной среде приходится, в свою очередь, обращаться к базовым механизмам ОС, реализуемым микроядром и другими серверами ОС.

Такому подходу к конструированию множественных прикладных сред присущи все достоинства и недостатки микро ядерной архитектуры, в частности:

- очень просто можно добавлять и исключать прикладные среды, что является следствием хорошей расширяемости микро ядерных ОС;
- при отказе одной из прикладных сред остальные сохраняют работоспособность, что способствует надежности и стабильности системы в целом;
- низкая производительность микроядерных ОС сказывается на скорости работы прикладных средств, а значит, и на скорости работы приложений.

В итоге следует отметить, что создание в рамках одной ОС нескольких прикладных средств для выполнения приложений различных ОС представляет собой путь, который позволяет иметь единственную версию программы и переносить ее между различными операционными системами. Множественные прикладные среды обеспечивают совместимость на двоичном уровне данной ОС с приложениями, написанными для других ОС.

1.9. Виртуальные машины как современный подход к реализации множественных прикладных сред

Понятие "монитор виртуальных машин" (МВМ) возникло в конце 60-х годов как программный уровень абстракции, разделявший аппаратную платформу на несколько

виртуальных машин. Каждая из этих виртуальных машин (ВМ) была настолько похожа на базовую физическую машину, что существующее программное обеспечение могло выполняться на ней в неизменном виде. В то время вычислительные задачи общего характера решались на дорогих мэйнфреймах (типа IBM/360), и пользователи высоко оценили способность МВМ распределять дефицитные ресурсы среди нескольких приложений.

В 80-90-е годы существенно снизилась стоимость компьютерного оборудования и появились эффективные многозадачные ОС, что уменьшило ценность МВМ в глазах пользователей. Мэйнфреймы уступили место мини-компьютерам, а затем ПК, и нужда в МВМ отпала. В результате из компьютерной архитектуры попросту исчезли аппаратные средства для их эффективной реализации. К концу 80-х в науке и на производстве МВМ воспринимались не иначе как исторический курьез [10].

Сегодня МВМ – снова в центре внимания. Корпорации Intel, AMD, Sun Microsystems и IBM создают стратегии виртуализации, в научных лабораториях и университетах для решения проблем мобильности, обеспечения безопасности и управляемости развиваются подходы, основанные на виртуальных машинах. Что же произошло между отставкой МВМ и их возрождением?

В 90-е годы исследователи из Стэндфордского университета начали изучать возможность применения ВМ для преодоления ограничений оборудования и операционных систем. Проблемы возникли у компьютеров с массовой параллельной обработкой (Massively Parallel Processing, MPP), которые плохо поддавались программированию и не могли выполнять имеющиеся ОС. Исследователи обнаружили, что с помощью виртуальных машин можно сделать эту неудобную архитектуру достаточно похожей на существующие платформы, чтобы использовать преимущества готовых ОС. Из этого проекта вышли люди и идеи, ставшие золотым фондом компании VMware (www.vmware.com), первого поставщика МВМ для компьютеров массового применения.

Как ни странно, развитие современных ОС и снижение стоимости оборудования привели к появлению проблем, которые исследователи надеялись решить с помощью МВМ. Дешевизна оборудования способствовала быстрому распространению компьютеров, но они часто бывали недогруженными, требовали дополнительных площадей и усилий по обслуживанию. А следствиями роста функциональных возможностей ОС стали их неустойчивость и уязвимость.

Чтобы уменьшить влияние системных аварий и защититься от взломов, системные администраторы вновь обратились к однозадачной вычислительной модели (с одним приложением на одной машине). Это привело к дополнительным расходам, вызванным повышенными требованиями к оборудованию. Перенос приложений с разных физических машин на ВМ и консолидация этих ВМ на немногих физических платформах позволили повысить эффективность использования оборудования, снизить затраты на управление и производственные площади. Таким образом, способность МВМ к мультиплексированию аппаратных средств – на этот раз во имя консолидации серверов и организации коммунальных вычислений – снова возродила их к жизни.

В настоящее время МВМ стал не столько средством организации многозадачности, каким он был когда-то задуман, сколько решением проблем обеспечения безопасности, мобильности и надежности. Во многих отношениях МВМ дает создателям операционных систем возможность развития функциональности, невозможной в нынешних сложных ОС. Такие функции, как миграция и защита, намного удобнее реализовать на уровне МВМ, поддерживающих обратную совместимость при развертывании инновационных решений в области операционных систем при сохранении предыдущих достижений.

Виртуализация – развивающаяся технология. В общих словах, виртуализация позволяет отделить ПО от нижележащей аппаратной инфраструктуры. Фактически она разрывает связь между определенным набором программ и конкретным компьютером. Монитор виртуальных машин отделяет программное обеспечение от оборудования и формирует промежуточный уровень между ПО, выполняемым виртуальными машинами, и аппаратными средствами. Этот уровень позволяет МВМ полностью контролировать использование аппаратных ресурсов гостевыми операционными системами (GuestOS), которые выполняются на ВМ.

МВМ создает унифицированное представление базовых аппаратных средств, благодаря чему физические машины различных поставщиков с разными подсистемами ввода-вывода выглядят одинаково и ВМ выполняются на любом доступном оборудовании. Не заботясь об отдельных машинах с их тесными взаимосвязями между аппаратными средствами и программным обеспечением, администраторы могут рассматривать оборудование просто как пул ресурсов для оказания любых услуг по требованию.

Благодаря полной инкапсуляции состояния ПО на ВМ монитор МВМ может отобразить ВМ на любые доступные аппаратные ресурсы и даже перенести с одной физической машины на другую. Задача балансировки нагрузки в группе машин становится тривиальной, и появляются надежные способы борьбы с отказами оборудования и наращивания системы. Если нужно отключить отказавший компьютер или ввести в строй новый, МВМ способен соответствующим образом перераспределить виртуальные машины. Виртуальную машину легко тиражировать, что позволяет администраторам по мере необходимости оперативно предоставлять новые услуги.

Инкапсуляция также означает, что администратор может в любой момент приостановить или возобновить работу ВМ, а также сохранить текущее состояние виртуальной машины либо вернуть ее к предыдущему состоянию. Располагая возможностью универсальной отмены, удается легко справиться с авариями и ошибками конфигурации. Инкапсуляция является основой обобщенной модели мобильности, поскольку приостановленную ВМ можно копировать по сети, сохранять и транспортировать на сменных носителях.

МВМ играет роль посредника во всех взаимодействиях между ВМ и базовым оборудованием, поддерживая выполнение множества виртуальных машин на единой аппаратной платформе и обеспечивая их надежную изоляцию. МВМ позволяет собрать группу ВМ с низкими потребностями в ресурсах на отдельном компьютере, снизив затраты на аппаратные средства и потребность в производственных площадях.

Полная изоляция также важна для надежности и обеспечения безопасности. Приложения, которые раньше выполнялись на одной машине, теперь можно распределить по разным ВМ. Если одно из них в результате ошибки вызовет аварию ОС, другие приложения будут от нее изолированы и продолжат работу. Если же одному из приложений угрожает внешнее нападение, атака будет локализована в пределах "скомпрометированной" ВМ. Таким образом, МВМ – это инструмент реструктуризации системы для повышения ее устойчивости и безопасности, не требующий дополнительных площадей и усилий по администрированию, которые необходимы при выполнении приложений на отдельных физических машинах.

МВМ должен связать аппаратный интерфейс с ВМ, сохранив полный контроль над базовой машиной и процедурами взаимодействия с ее аппаратными средствами. Для достижения этой цели существуют разные методы, основанные на определенных технических компромиссах. При поиске таких компромиссов принимаются во внимание основные требования к МВМ: совместимость, производительность и простота. Совместимость важна потому, что главное достоинство МВМ – способность выполнять унаследованные приложения. Производительность определяет величину накладных расходов на виртуализацию – программы

на ВМ должны выполняться с той же скоростью, что и на реальной машине. Простота необходима, поскольку отказ МВМ приведет к отказу всех ВМ, выполняющихся на компьютере. В частности, для надежной изоляции требуется, чтобы МВМ был свободен от ошибок, которые злоумышленники могут использовать для разрушения системы.

Вместо того чтобы заниматься сложной переработкой кода гостевой операционной системы, можно внести некоторые изменения в основную операционную систему, изменив некоторые наиболее "мешающие" части ядра. Подобный подход называется паравиртуализацией [10]. Ясно, что в этом случае адаптировать ядро ОС может только автор, и, например, Microsoft не проявляет желания адаптировать популярное ядро Windows 2000 к реалиям конкретных виртуальных машин.

При паравиртуализации разработчик МВМ переопределяет интерфейс виртуальной машины, заменяя непригодное для виртуализации подмножество исходной системы команд более удобными и эффективными эквивалентами. Заметим, что хотя ОС нужно портировать для выполнения на таких ВМ, большинство обычных приложений могут выполняться в неизменном виде.

Самый большой недостаток паравиртуализации – несовместимость. Любая операционная система, предназначенная для выполнения под управлением паравиртуализованного монитора МВМ, должна быть портирована в эту архитектуру, для чего нужно договариваться о сотрудничестве с поставщиками ОС. Кроме того, нельзя использовать унаследованные операционные системы, а существующие машины не удается легко заменить виртуальными.

Чтобы добиться высокой производительности и совместимости при виртуализации архитектуры x86, компания VMware разработала новый метод виртуализации, который объединяет традиционное прямое выполнение с быстрой трансляцией двоичного кода "на лету". В большинстве современных ОС режимы работы процессора при выполнении обычных прикладных программ легко поддаются виртуализации, а следовательно, их можно виртуализировать посредством прямого выполнения. Непригодные для виртуализации привилегированные режимы может выполнять транслятор двоичного кода, исправляя "неудобные" команды x86. В результате получается высокопроизводительная виртуальная машина, которая полностью соответствует оборудованию и поддерживает полную совместимость ПО.

Преобразованный код очень похож на результаты паравиртуализации. Обычные команды выполняются в неизменном виде, а команды, нуждающиеся в специальной обработке (такие как POPF и команды чтения регистров сегмента кода), транслятор заменяет последовательностями команд, которые подобны требующимся для выполнения на паравиртуализованной виртуальной машине. Однако есть важное различие: вместо того, чтобы изменять исходный код операционной системы или приложений, транслятор двоичного кода изменяет код при его выполнении в первый раз.

Хотя трансляция двоичного кода требует некоторых дополнительных расходов, при нормальных рабочих нагрузках они незначительны. Транслятор обрабатывает лишь часть кода, и скорость выполнения программ становится сопоставимой со скоростью прямого выполнения – как только заполнится кэш-память трассировки.

Трансляция двоичного кода также помогает оптимизировать прямое выполнение. Например, если при прямом выполнении привилегированного кода часто происходит перехват команд, это может привести к существенным дополнительным расходам, поскольку при каждом перехвате управление передается от виртуальной машины к монитору и обратно. Трансляция кода может устранить многие из таких перехватов, что приведет к снижению

накладных расходов на виртуализацию. Это особенно верно для центральных процессоров с длинными конвейерами команд, в частности, для современного семейства x86, в котором перехват связан с высокими дополнительными расходами.

1.10. Эффекты виртуализации

Экспертиза современных продуктов и недавние исследования раскрывают некоторые интересные возможности развития МВМ и требования, которые они предъявят к технологиям виртуализации.

Администраторы центра данных могут с единой консоли быстро вводить в действие ВМ и управлять тысячами виртуальных машин, выполняющихся на сотнях физических серверов. Вместо того чтобы конфигурировать отдельные компьютеры, администраторы будут создавать по имеющимся шаблонам новые экземпляры виртуальных серверов и отображать их на физические ресурсы в соответствии с политиками администрирования. Уйдет в прошлое взгляд на компьютер как на средство предоставления конкретных услуг. Администраторы будут рассматривать компьютеры просто как часть пула универсальных аппаратных ресурсов (примером тому может служить виртуальный центр VMware VirtualCenter).

Отображение виртуальных машин на аппаратные ресурсы очень динамично. Возможности миграции работающих ВМ (подобные тем, которые обеспечивает технология VMotion компании Vmware) позволяют ВМ быстро перемещаться между физическими машинами в соответствии с потребностями центра данных. МВМ сможет справляться с такими традиционными проблемами, как отказ оборудования, за счет простого перемещения ВМ с отказавшего компьютера на исправный. Возможность перемещения работающих ВМ облегчит решение аппаратных проблем, таких как планирование профилактического обслуживания, окончание срока действия лизингового договора и модернизация оборудования: администраторы станут устранять эти проблемы без перерывов в работе.

Еще недавно нормой являлась ручная миграция, но сейчас уже распространены инфраструктуры виртуальных машин, которые автоматически выполняют балансировку нагрузки, прогнозируют отказы аппаратных средств и соответствующим образом перемещают ВМ, создают их и уничтожают в соответствии со спросом на конкретные услуги.

Решение проблем на уровне МВМ положительно сказывается на всех программах, выполняющихся на ВМ, независимо от их возраста (унаследованная или новейшая) и поставщиков. Независимость от ОС избавляет от необходимости покупать и обслуживать избыточную инфраструктуру. Например, из нескольких версий ПО службы поддержки или резервного копирования останется лишь одна – та, которая работает на уровне МВМ.

Виртуальные машины сильно изменили отношение к компьютерам. Уже сейчас простые пользователи умеют легко создавать, копировать и совместно использовать ВМ. Модели их применения значительно отличаются от привычных, сложившихся в условиях вычислительной среды с ограниченной доступностью аппаратных средств. А разработчики ПО могут применять такие продукты, как VMware Workstation, чтобы легко установить компьютерную сеть для тестирования или создать собственный набор испытательных машин для каждой цели.

Повышенная мобильность ВМ значительно изменила способы их применения. Такие проекты, как Collective и Internet Suspend/Resume, демонстрируют возможность перемещения всей вычислительной среды пользователя по локальной и территориально-распределенной сети. Доступность высокоемких недорогих сменных носителей, например, жестких дисков USB, означает, что потребитель может захватить свою вычислительную среду с собой, куда бы он ни направлялся.

Динамический характер компьютерной среды на базе ВМ требует и более динамичной топологии сети. Виртуальные коммутаторы, виртуальные брандмауэры и оверлейные сети становятся неотъемлемой частью будущего, в котором логическая вычислительная среда отделится от своего физического местоположения.

Виртуализация обеспечивает высокий уровень работоспособности и безопасности благодаря нескольким ключевым возможностям.

Локализация неисправностей. Большинство отказов приложений происходят из-за ошибок ПО. Виртуализация обеспечивает логическое разделение виртуальных разделов, поэтому программный сбой в одном разделе никак не влияет на работу приложения в другом разделе. Логическое разделение также позволяет защищаться от внешних атак, что повышает безопасность консолидированных сред.

Гибкая обработка отказов. Виртуальные разделы можно настроить так, чтобы обеспечить автоматическую обработку отказов для одного или нескольких приложений. Благодаря средствам обеспечения высокой степени работоспособности, заложенным сейчас в платформы на базе процессоров Intel® Itanium® 2 и Intel® Xeon™ MP, требуемый уровень услуг часто можно обеспечить, предусмотрев аварийный раздел на той же платформе, где работает основное приложение. Если требуется еще более высокий уровень работоспособности, аварийный раздел можно разместить на отдельной платформе.

Разные уровни безопасности. Для каждой виртуальной машины можно установить разные настройки безопасности. Это позволит IT-организациям обеспечить высокий уровень контроля за конечными пользователями, а также гибкое распределение административных привилегий.

МВМ имеют мощный потенциал для реструктуризации существующих программных систем в целях повышения уровня защиты, а также облегчают развитие новых подходов к построению безопасных систем. Сегодняшние ОС не обеспечивают надежной изоляции, оставляя машину почти беззащитной. Перемещение механизмов защиты за пределы ВМ (чтобы они выполнялись параллельно с ОС, но были изолированы от нее) позволяет сохранить их функциональные возможности и повысить устойчивость к нападениям.

Размещение средств безопасности за пределами ВМ – привлекательный способ изоляции сети. Доступ к сети предоставляется ВМ после проверки, гарантирующей, что она, с одной стороны, не представляет угрозы, а с другой – неуязвима для нападения. Управление доступом к сети на уровне ВМ превращает виртуальную машину в мощный инструмент борьбы с распространением злонамеренного кода.

Мониторы МВМ особенно интересны в плане управления многочисленными группами программ с различными уровнями безопасности. Благодаря отделению ПО от оборудования ВМ обеспечивают максимальную гибкость при поиске компромисса между производительностью, обратной совместимостью и степенью защиты. Изоляция программного комплекса в целом упрощает его защиту. В сегодняшних ОС почти невозможно судить о безопасности отдельного приложения, поскольку процессы плохо изолированы от друг друга. Таким образом, безопасность приложения зависит от безопасности всех остальных приложений на машине.

Гибкость управления ресурсами, которую обеспечивают МВМ, может сделать системы более стойкими к нападениям. Возможность быстро тиражировать ВМ и динамически адаптироваться к большим рабочим нагрузкам станет основой мощного инструмента, позволяющего справиться с нарастающими перегрузками из-за внезапного наплыва посетителей на Web-сайте или атаки типа "отказ в обслуживании".

Модель распространения программных продуктов на основе ВМ потребует от поставщиков ПО корректировки лицензионных соглашений. Лицензии на эксплуатацию на конкретном процессоре или физической машине не приживутся в новых условиях, в отличие от лицензий на число пользователей или неограниченных корпоративных лицензий. Пользователи и системные администраторы будут отдавать предпочтение операционным средам, которые легко и без особых затрат распространяются в виде виртуальных машин.

Возрождение МВМ существенно изменило представления разработчиков программных и аппаратных средств о структурировании сложных компьютерных систем и управлении ими. Кроме того, МВМ обеспечивают обратную совместимость при развертывании инновационных решений в области операционных систем, которые позволяют решать современные задачи, сохраняя предыдущие достижения. Эта их способность станет ключевой при решении грядущих компьютерных проблем.

Виртуализация предоставляет также преимущества для сред разработки и тестирования ПО. Различные этапы цикла создания ПО, включая получение рабочей версии, можно выполнять в разных виртуальных разделах одной и той же платформы. Это поможет повысить степень полезного использования аппаратного обеспечения и упростить управление жизненным циклом. Во многих случаях IT-организации получат возможность тестировать новые и модернизированные решения на имеющихся рабочих платформах, не прерывая производственный процесс. Это не только упрощает миграцию, но также позволяет сократить расходы, устранив необходимость дублирования вычислительной среды.

Освобождая разработчиков и пользователей от ресурсных ограничений и недостатков интерфейса, виртуальные машины снижают уязвимость системы, повышают мобильность программного обеспечения и эксплуатационную гибкость аппаратной платформы.

Компьютерные системы существуют и продолжают развиваться благодаря тому, что разработаны по законам иерархии и имеют хорошо определенные интерфейсы, отделяющие друг от друга уровни абстракции. Использование таких интерфейсов облегчает независимую разработку аппаратных и программных подсистем силами разных групп специалистов. Абстракции скрывают детали реализации нижнего уровня, уменьшая сложность процесса проектирования.

Подсистемы и компоненты, разработанные по спецификациям разных интерфейсов, не способны взаимодействовать друг с другом. Например, приложения, распространяемые в двоичных кодах, привязаны к определенной ISA и зависят от конкретного интерфейса к операционной системе. Несовместимость интерфейсов может стать сдерживающим фактором, особенно в мире компьютерных сетей, в котором свободное перемещение программ столь же необходимо, как и перемещение данных.

Виртуализация позволяет обойти эту несовместимость. Виртуализация системы или компонента (например, процессора, памяти или устройства ввода/вывода) на конкретном уровне абстракции отображает его интерфейс и видимые ресурсы на интерфейс и ресурсы реальной системы. Следовательно, реальная система выступает в роли другой, виртуальной системы или даже нескольких виртуальных систем.

В отличие от абстракции, виртуализация не всегда нацелена на упрощение или сокрытие деталей. Например, при отображении виртуальных дисков на реальный программные средства виртуализации используют абстракцию файла как промежуточный шаг. Операция записи на виртуальный диск преобразуется в операцию записи в файл (и следовательно, в операцию записи на реальный диск). Отметим, что в данном случае никакого абстрагирования не происходит – уровень детализации интерфейса виртуального диска (адресация секторов и дорожек) ничем не отличается от уровня детализации реального диска.