ГЛАВА 2

ПРИНЦИПЫ ПОСТРОЕНИЯ И ФУНКЦИОНИРОВАНИЯ ОПЕРАЦИОННЫХ СИСТЕМ В ТРАДИЦИОННЫХ ВС

ВВЕДЕНИЕ

Данный раздел посвящен описанию принципов построения и структурной организации операционных систем (ОС) в традиционных вычислительных системах. Дана классификация поколений ОС, их функциональные особенности и современные виды. Показано, что основой любой современной вычислительной системы является операционная система, от которой зависит как функционирование вычислительной установки в целом, так и эффективность прохождения работ через нее.

В разделе рассматриваются следующие вопросы:

- Что такое операционная система ?
- История развития операционных систем.
- Концепция построения операционных систем (процесс, файлы, вызов, ядро).
- Классификация операционных систем и их структурные особенности.
- Понятие генерации, инсталляции, инициализации ОС.
- Ядро операционной системы, структура, состав, взаимодействие.
- Общая схема загрузки операционной системы.
- Общая схема функционирования операционной системы (на примере MS DOS)

2.1. ЧТО ТАКОЕ ОПЕРАЦИОННАЯ СИСТЕМА?

Операционная система ЭВМ или отдельной информационно - вычислительной системы представляет собой совокупность программ, выполняющих две основных функции:

Эффективное управление ресурсами системы, обеспечение (в случае необходимости) их распределения между несколькими пользователями и контроль за выделением ресурсов для одновременного выполнения многих задач;

Предоставление набора услуг, обеспечивающего пользователю удобный интерфейс, который приспособлен к его нуждам лучше, чем интерфейс физической машины; этим интерфейсом является интерфейс "виртуальная машина", который предоставляет набор средств для управления информацией и реализацией прикладного программного обеспечения.

Программы, выполняемые вычислительной системой, по функциональному назначению подразделяются на два класса: прикладные и системные. Операционные системы составляют важную часть программ второго класса.

Операционная система *— программная среда, обеспечивающая поддержку работы всех программ, их взаимодействие с аппаратными средствами вычислительной системы, а также предоставляющая пользователю возможности общего управления вычислительной системой и использования ресурсов ВС.*

Операционные системы различаются архитектурой, возможностями, требуемыми ресурсами для их функционирования, набором сервисных функций, ориентацией на типы используемых микропроцессоров и другими особенностями. Развитие операционных систем неразрывно связано и в значительной степени определялось развитием технического обеспечения вычислительных систем, функциональными и структурными особенностями компонентов ВС и внедрением новых принципов организации вычислительного процесса на аппаратном и программном уровне.

2.2. ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Поколения операционных систем:

— НУЛЕВОЕ ПОКОЛЕНИЕ. Для первых цифровых вычислительных машин не существовало операционных систем. Пользователь получал возможность полностью распоряжаться вычислительной системой и ее ресурсами на строго определенное время. Это были машины значительных размеров, сделанные на ненадежных электронных лампах. Программы, написанные на машинном языке, работали на "голой" машине без какого либо промежуточного системного обеспечения. Пользователи вводили свои программы, написанные на машинном языке, в компьютер с помощью машинных носителей (перфокарт, перфолент или, в самых первых компьютерах, с помощью коммутационной панели), загружали программы, ждали их выполнения и уносили распечатки с выходными данными. В некотором смысле пользователи-программисты выполняли функции операционной системы, так как они управляли всеми деталями действий машины и эффективность использования ее ресурсов целиком зависела от их действий. Прямое взаимодействие с машиной (пошаговое выполнение команд, непосредственный контроль и изменение состояния ячеек памяти) было главным средством отладки программы. Программисты должны были сами писать программы выполнения процедур ввода/вывода для своих программ. Печать символа, например, требовала много машинных команд, так как в компьютере не было нужной функции. Если этого не было в вашей программе, то этого не было нигде. Отсутствовало также управление памятью. Каждая программа имела доступ ко всему пространству памяти — она пользовалась тем, что было ей нужно, а остаток не был задействован.

Загрузка системы включала в себя запись программ в двоичном коде с помощью переключателей на передней панели в память машины, что занимало очень много времени и не гарантировало отсутствие ошибок в управлении. Такая работа

оказалась крайне неэффективной, так как дорогостоящее оборудование использовалось неэкономно.

Первыми программными системными средствами, повышающими эффективность работы пользователя на вычислительной установке, а отсюда и работу программиста, были, с одной стороны, средства автоматизации разработки программ (ассемблеры, компиляторы, отладчики), а с другой ── подпрограммы ввода/вывода.

— *ПЕРВОЕ ПОКОЛЕНИЕ.* В середине 50-х годов в компьютерах стала использоваться новая элементная база ── транзисторы. Это сделало машины более надежными, и сфера их применения расширилась. Однако принцип работы практически не изменился: программист или профессиональный оператор загружал программу, ждал пока она будет выполнена, затем снова ждал, когда будут напечатаны результаты ── только потом загружалась следующая программа. Стало ясно, что такое использование дорогого быстродействующего оборудования является крайне неэффективным. Одна из проблем, которую должны были решить операционные системы, такова: ввод и вывод данных занимали время, но при этом не полностью использовались вычислительные возможности центрального процессора (ЦП) т.к. скорости ЦП и устройств ввода/вывода несоизмеримы. Ожидая, пока произойдет ввод/вывод, ЦП обычно работал вхолостую, растрачивая свой ресурс.

Первым решением проблемы повышения эффективности использования ЦП была простая пакетная обработка. При таком подходе перфокарты многих программ считывались дополнительным низкоскоростным компьютером на магнитную ленту, подготавливая входную очередь заданий — пакет. ЦП обрабатывал программы пакета одну за другой и выводил результаты на другую магнитную ленту. Так как накопители на магнитной ленте были быстрее, чем устройство перфоввода и принтеры, то положение дел улучшилось. Проблема простоя ЦП во время ввода/вывода была решена за счет ускорения этого процесса.

К концу 50-х годов появились первые пакетные мониторы. Эти программы создавались с целью выполнения всей последовательности работ (организации данных и выполнения прикладных программ), подготовленных заранее, и автоматизации процесса перехода от одной работы к другой. Основной функцией такой системы было управление ресурсами: памятью, процессором, вводом/выводом. Автоматизация управления ресурсами ВС вызвала появление функций защиты ВС при выполнении работ от ошибок:

- ограничение времени доступа к процессору, чтобы устранить блокирование всей работы из-за зацикливания в одной программе;
- надзор за вводом/выводом, чтобы избежать циклического обращения к периферийным устройствам по ошибке;

- защита зоны памяти, постоянно занимаемой монитором, от ошибок адресации в пользовательских программах.

Обеспечение указанных функций привело к расширению возможностей машин: измерение и учет времени, ограничения на использование некоторых инструкций, защита памяти.

Использование пакетных мониторов существенно повысило производительность вычислительных систем и уменьшило непроизводительные затраты машинного времени при смене заданий. Однако это повышение в значительной степени ограничивалось тем, что при выполнении операций ввода/вывода простаивал процессор. Стали использоваться два спаренных компьютера. Программы выполнялись на одном из них (главном), оснащенном устройством с повышенной скоростью передачи данных на магнитную ленту и считывания данных с магнитной ленты. Вспомогательный компьютер служил для записи данных с перфокарт на ленту и вывода результатов вычислений с магнитной ленты на печать.

Операционная система этих систем состояла из команд, расположенных между программами, подлежащими считыванию, и резидентной управляющей программы для интерпретации этих команд написанных с использованием JCL (язык управления заданиями).

Предварительное планирование очередности работ позволяло работать обеим машинам параллельно и тем самым более полно использовать возможности главного процессора. Недостаток такой организации — негибкость системы (временные ограничения для ввода заданий, долгое ожидание результатов). Описанные системы последовательной пакетной обработки получили широкое распространение в начале 60-х годов.

— *ВТОРОЕ ПОКОЛЕНИЕ.* В период с 1960 по 1970 г. значительный прогресс в области технологии материалов и в разработке принципов функционирования операционных систем позволил устранить недостатки систем пакетной обработки и перейти к системам с коллективным доступом.

Введение автономных специализированных процессоров для передачи информации (каналов, или устройств обмена) позволило освободить центральный процессор от управления вводом/выводом.

Мультипрограммирование, связанное с разделением памяти сразу для нескольких работ, повысило производительность центрального процессора за счет одновременной работы процессора и нескольких устройств ввода/вывода.

Для равномерной загрузки оборудования в таких системах требуется тщательный подбор задач, выполняемых в режиме мультипрограммирования. В большинстве случаев программы не будут подходить друг другу, и невозможно обеспечить равномерность загрузки.. Однако при запуске более чем двух программ одновременно ЦП может работать с пользой большую часть времени.

Конечно, функционирование ВС в многозадачном режиме требует использования более сложного системного программного обеспечения

(операционной системы), чтобы следить за тем, где расположена та или иная программа, в какой стадии выполнения и, совместно с аппаратным обеспечением, предотвращать взаимодействие между программами.

Управление памятью в первых мультипрограммных системах было достаточно простым: следить, чтобы каждая программа занимала только выделенную ей часть памяти и оставалась там вплоть до завершения и вывода.

Однако случалось, что программа была слишком большой, чтобы уместиться в отведенной ей части, или даже настолько большой, что не умещалась во всей памяти машины. В этом случае программы должны были делиться на оверлеи — отдельные части программы, которые вызывали друг друга, но это создавало дополнительные трудности программисту, так как именно ему приходилось делить программу на части подходящего размера и следить за бесконфликтным вызовом одной программы другой.

Лучшим решением этой проблемы была виртуальная память, где доступное пространство памяти ВС для каждой программы выглядело больше, чем имеющийся в действительности объем реальной оперативной памяти. Программа могла быть поделена на страницы (обычно одинакового размера). Те страницы программы, которые были активны (выполнялись команды или осуществлялся доступ к данным), сохранялись в оперативной памяти, тогда как те, которые не использовались, временно записывались на быстродействующий диск. Этот процесс полностью управлялся операционной системой и был незаметен программисту, который мог писать программы, занимающие больше памяти, чем было физически доступно.

Разработчики также создали библиотеку стандартных программ ввода/вывода, чтобы пользователям не приходилось создавать их с нуля. После этого операционные системы имели собственные функции вывода символа на экран и выполняли другие задачи ввода/вывода. Было логичнее просто использовать эти функции, чем каждый раз повторять их в каждой программе. Так родилась идея системных функций, которые до сих пор являются частью большинства операционных систем.

Хотя мультипрограммирование заставляло ЦП работать большую часть времени, оно никак не помогало решить другую проблему: эффективное использование всей вычислительной установки в целом. Программист обычно приносил программу в виде колоды перфокарт в машинный зал и отдавал их оператору. Затем нужно было ждать, когда она выполниться, что занимало иногда несколько часов т.к. задача выполнялась в пакете. И, наконец, программист забирал распечатки с результатами из машинного зала и смотрел, что получилось. Времени обычно хватало на 2-3 таких прогона в день, так что отладка была очень длительной процедурой (поразительно, что какое-то программное обеспечение все-таки было написано).

Решением проблемы нехватки времени было введение систем разделения времени. При разделении времени несколько терминалов связывались с одним ЦП, и каждый получал серию маленьких промежутков (квантов) времени ЦП. Так как

человек обычно работает медленнее процессора, то пользователь не замечал, что между нажатием "1" и "s" при набивке строки "1s" ЦП обслуживает еще нескольких пользователей. Результатом явилась интерактивная система, где кажется, что компьютер отзывается сразу после ввода. Это режим кажущегося совмещения. Основной целью функционирования систем с разделением времени является улучшение работы пользователя. В наиболее развитых системах с разделением времени результаты пользователю отправлялись по мере их получения, что еще больше усиливало эффект "индивидуального" использования ресурсов ВС пользователем.

Первой широко используемой системой с разделением времени была операционная система UNIX, которая с тех пор получила широкое распространение и была перенесена на множество различных машин.

В среде с разделением времени операционная система имеет другие приоритеты, чем пакетная система. Она все еще остается мультипрограммной системой, но акцент делает не на оптимизацию использования ЦП, а на минимизации времени отклика. Это позволяет пользователю нормально работать и видеть результаты через достаточно короткое время. Естественно, что при подключении к системе слишком большого количества пользователей, время отклика будет увеличивается и характеристики системы будут ухудшаться. Система должна попытаться удовлетворить всех пользователей в равной мере, снабдив каждого из них одинаковой частью машинного времени.

Задача распределения времени ЦП в системах с разделением времени намного более сложна, чем в пакетных системах. Она решается специальной частью операционной системы, называемой планировщиком. Принципы, заложенные в планировщике, являются важными факторами в характеристике системы. В системе с разделением времени каждая программа работает в течение короткого отрезка времени: говорят, что программа получает квант времени. Когда промежуток времени одной программы заканчивается, начинает работать другая программа, несмотря на то, что первая программа не завершена. Это переключение и принципы выбора следующей программы для исполнения определяется приоритетной системой и применяемыми в ВС дисциплинами обслуживания заявок: каждый процесс либо является приоритетным, либо отключается и замещается другим.

Заметим, что когда мы говорим о двух программах, работающих одновременно в системе с разделением времени, мы не имеем в виду точную одновременность, так как ЦП отдает квант времени только одному пользователю. Пользователям кажется, что много программ работают параллельно, но в действительности ЦП запускает программу пользователя А на несколько миллисекунд, затем программу пользователя Б, и так далее, до нескольких десятков. Затем он возвращается, чтобы дать пользователю А следующий квант времени. Только быстрота этого "карусельного" переключения времени ЦП создает иллюзию одновременности для пользователя.

Операционная система с разделением времени должна также управлять ресурсами. Как и в реальном мире, ресурсы — это то, чего всегда не хватает, то

есть то, что нужно делить между многими пользователями. Примером ресурсов могут служить принтеры и память. Операционная система поддерживает порядок, осуществляя вывод программ пользователей на принтер и распределяя память так эффективно и справедливо, как это возможно.

Метод обращения с памятью весьма сложен. Один из общих подходов состоит в том, чтобы поделить каждую программу пользователя на сегменты. Сегменты могут соответствовать логическим частям программы, например один сегмент ── для команд, один сегмент — для данных и один — для стека. Сегменты могут заноситься в память и удаляться из нее по необходимости. Они также могут быть защищены так, что только одна программа будет иметь к ним доступ. Или. наоборот, они могут быть совместными, делая группу данных доступной многим пользователям.

Так как в системе с разделением времени имеется множество пользователей, то возникают дополнительные сложности. Должна быть соблюдена конфиденциальность отдельных программ, так чтобы пользователь не мог случайно или преднамеренно изменить чужую программу либо данные или, что еще хуже, разрушить саму систему. Желательно, чтобы система имела простейший метод контроля. Должно регистрироваться, например, как долго каждый пользователь работал с системой и какие действия он совершал. Именно при разработке систем с разделением времени были введены понятия прерывания и логических устройств. Прерывания и логика обработки прерываний позволило перейти к внедрению операционных систем с принципиально новыми функциональными возможностями.

— *ТРЕТЬЕ ПОКОЛЕНИЕ.* На 70-е годы приходится начало расцвета операционных систем. В это время были заложены принципы совместимости операционных систем "снизу вверх". Разрабатываются многорежимные операционные системы, предназначенные для разных конфигураций вычислительных систем. Однако для этих систем характерна труднодоступность пользователя к аппаратным ресурсам вычислительной системы.

70-е годы отмечены двумя событиями:

- Появлением микропроцессоров и постоянным ростом их возможностей, которые позволили обеспечить значительные вычислительные мощности при относительно низкой стоимости.
- Развитием техники передачи данных (телеобработки) и постоянно растущей степени интеграции средств связи в информационно- вычислительные системы.

Одновременно с этим опыт эксплуатации вычислительных систем сформировал новые требования пользователей к ним:

- возможность разделения ресурсов, оправданное с экономической точки зрения, и доступ к удаленной информации (ресурсы и информация могут находиться в географически разных местах);

- возможность наилучшего соответствия и адаптации вычислительной системы к структуре прикладных задач, приводящая к децентрализации системы и географическому распределению ее элементов;
- возможность объединения прикладных задач (ранее разъединенных) в единую совокупность обьектов на единых системных принципах использования и взаимодействия .

В результате появились новые типы вычислительных систем :

— Сети телеобработки данных, позволяющие организовать взаимосвязь между существующими системами, обмен данными между ними и доступ к удаленным устройствам информационного обслуживания.

— Локальные сети ЭВМ с высокоскоростными каналами передачи информации (10 Мбит/с), географически сконцентрированные и выполняющие функции:

- систем управления;
- систем связи и управления документацией (учрежденческих систем), ориентированных на обмен информацией, создание и хранение документов;
- систем общего пользования, предназначенных, в частности, для создания программного обеспечения.

В тоже время предполагаемое развитие сетей и персональных компьютеров не исключает существование больших централизованных систем с доступом в режиме разделения времени, которые остаются экономически оправданными для многочисленных применений. Задачи вычислительной математики в ряде специальных областей требуют развития вычислительных систем очень большой мощности, включающих специализированные мультипроцессоры, для которых должна быть разработана своя архитектура операционных систем.

В 1965 году фирма Bell Telephone Laboratories, объединив свои усилия с компанией General Electric и проектом MAC Массачусетского технологического института, приступили к разработке новой операционной системы, получившей название Multics. Перед системой Multics были поставлены задачи — обеспечить одновременный доступ к ресурсам ЭВМ большого количества пользователей, обеспечить достаточную скорость вычислений и хранение данных и дать возможность пользователям в случае необходимости совместно использовать данные. Многие разработчики, в последствии принявшие участие в создании ранних редакций системы UNIX, участвовали в работе над системой Multics в фирме Bell Telephone Laboratories. Хотя первая версия системы Multics и была запущена в 1969 году на ЭВМ GE 645, она не обеспечивала выполнение главных вычислительных задач, для решения которых она предназначалась, и не было даже ясно, когда цели разработки будут достигнуты. Поэтому фирма Bell Laboratories прекратила свое участие в проекте.

По окончании работы над проектом Multics сотрудники Исследовательского центра по информатике фирмы Bell Telephone Laboratories остались без "достаточно интерактивного вычислительного средства". Пытаясь усовершенствовать среду программирования, Кен Томпсон, Дэннис Ричи и другие набросали на бумаге проект файловой системы, получивший позднее дальнейшее развитие в ранней

версии файловой системы UNIX. Томпсоном были написаны программы, имитирующие поведение предложенной файловой системы в режиме подкачки данных по запросу, им было даже создано простейшее ядро операционной системы для ЭВМ GE 645. Томпсон получил возможность изучить машину PDP-7, однако условия разработки программ потребовали использования кросс-ассемблера для трансляции программы на машине с системой GECOS и использования перфоленты для ввода в PDP-7. Для того, чтобы улучшить условия разработки, Томпсон и Ричи выполнили на PDP-7 свой проект системы, включивший первую версию файловой системы UNIX, подсистему управления процессами и небольшой набор утилит. В конце концов, новая система больше не нуждалась в поддержке со стороны системы GECOS в качестве операционной среды разработки и могла поддерживать себя сама. Новая система получила название UNIX, по сходству с Multics его придумал еще один сотрудник Исследовательского центра по информатике Брайан Керниган.

Несмотря на то, что эта ранняя версия системы UNIX уже была многообещающей, она не могла реализовать свой потенциал до тех пор, пока не получила применение в реальном проекте. Так, для того, чтобы обеспечить функционирование системы обработки текстов для патентного отдела фирмы Bell Telephone Laboratories, в 1971 году система UNIX была перенесена на ЭВМ PDP-11. Система отличалась небольшим объемом: 16 Кбайт для системы, 8 Кбайт для программ пользователей, обслуживала диск объемом 512 Кбайт и отводила под каждый файл не более 64 Кбайт. После своего первого успеха Томпсон собрался было написать для новой системы транслятор с Фортрана, но вместо этого занялся языком Би (B), предшественником которого явился язык BCPL. Би был языком интерпретирующего типа со всеми недостатками, присущими подобным языкам, поэтому Ричи переделал его в новую разновидность, получившую название Си (C) и разрешающую генерировать машинный код, объявлять типы данных и определять структуру данных. В 1973 году система была написана заново на Си. Это был шаг, неслыханный для того времени, но имевший огромный резонанс среди широкого круга пользователей. Количество машин фирмы Bell Telephone Laboratories, на которых была инсталирована система, возросло до 25, в результате чего была создана группа по системному сопровождению UNIX внутри фирмы.

В то время корпорация AT&T не могла заниматься продажей компьютерных продуктов в связи с соответствующим соглашением, подписанным ею с федеральным правительством в 1956 году, и распространяла систему UNIX среди университетов, которым она была нужна в учебных целях. Следуя букве соглашения, корпорация AT&T не рекламировала, не продавала и не сопровождала систему. Несмотря на это, популярность системы устойчиво росла. В 1974 году Томпсон и Ричи опубликовали статью, описывающую систему UNIX, в журнале Communications of the ACM, что дало еще один импульс к распространению системы. К 1977 году количество машин, на которых функционировала система UNIX, увеличилось до 500, при чем 125 из них работали в университетах. Система UNIX завоевала популярность среди телефонных компаний, поскольку

обеспечивала хорошие условия для разработки программ, обслуживала работу в сети в режиме диалога и работу в реальном масштабе времени (с помощью системы MERT). Помимо университетов, лицензии на систему UNIX были переданы коммерческим организациям. В 1977 году корпорация Interactive Systems стала первой организацией, получившей права на перепродажу системы UNIX с надбавкой к цене за дополнительные услуги, которые заключались в адаптации системы к функционированию в автоматизированных системах управления учрежденческой деятельностью. 1977 год также был отмечен "переносом" системы UNIX на машину, отличную от PDP (благодаря чему стал возможен запуск системы на другой машине без изменений или с небольшими изменениями), а именно на Interdata 8/32.

С ростом популярности микропроцессоров другие компании стали переносить систему UNIX на новые машины, однако ее простота и ясность побудили многих разработчиков к самостоятельному развитию системы, в результате чего было создано несколько вариантов базисной системы. За период между 1977 и 1982 годом фирма Bell Laboratories объединила несколько вариантов, разработанных в корпорации AT&T, в один, получивший коммерческое название UNIX версия III. В дальнейшем фирма Bell Telephone Laboratories добавила в версию III несколько новых особенностей, назвав новый продукт UNIX версия V, и эта версия стала официально распространяться корпорацией AT&T с января 1983 года. В то же время сотрудники Калифорнийского университета в Бэркли разработали вариант системы UNIX, получивший название BSD 4.3 для машин серии VAX и отличающийся некоторыми новыми, интересными особенностями.

К началу 1984 года система UNIX была уже инсталлирована приблизительно на 100000 машин по всему миру, причем на машинах с широким диапазоном вычислительных возможностей ── от микропроцессоров до больших ЭВМ — и разных изготовителей. Ни о какой другой операционной системе нельзя было бы сказать того же. Популярность и успех системы UNIX объяснялись несколькими причинами:

Система написана на языке высокого уровня, благодаря чему ее легко читать, понимать, изменять и переносить на другие машины. По оценкам, сделанным Ричи, первый вариант системы на Си имел на 20-40 % больший объем и работал медленнее по сравнению с вариантом на ассемблере, однако преимущества использования языка высокого уровня намного перевешивают недостатки:

- Наличие довольно простого пользовательского интерфейса, в котором имеется возможность предоставлять все необходимые пользователю услуги.
- Наличие элементарных средств, позволяющих создавать сложные программы из более простых.
- Наличие иерархической файловой системы, легкой в сопровождении и эффективной в работе.
- Обеспечение согласования форматов в файлах, работа с последовательным потоком байтов, благодаря чему облегчается чтение прикладных программ.

- Наличие простого, последовательного интерфейса с периферийными устройствами.
- Система является многопользовательской, многозадачной; каждый пользователь может одновременно выполнять несколько процессов.
- Архитектура машины скрыта от пользователя, благодаря этому облегчен процесс написания программ, работающих на различных конфигурациях аппаратных средств.
- Простота и последовательность вообще отличают систему UNIX и объясняют большинство из вышеприведенных доводов в ее пользу.

Хотя операционная система и большинство команд написаны на Си, система UNIX поддерживает ряд других языков, таких как Фортран, Бейсик, Паскаль, Ада, Кобол, Лисп и Пролог. Система UNIX может поддерживать любой язык программирования, для которого имеется компилятор или интерпретатор, и обеспечивать системный интерфейс, устанавливающий соответствие между пользовательскими запросами к операционной системе, и набором запросов, принятых в UNIX.

— *ЧЕТВЕРТОЕ ПОКОЛЕНИЕ.* В конце 70-х годов технология изготовления БИС понизила стоимость и размеры одного транзистора до таких пределов, что стало целесообразным производить персональные компьютеры. Это вызвало поворот к ситуации с одной программой и одним пользователем, как во времена первых компьютеров, и возможность полного владения и управления ресурсами системы. Первый массовый рынок персональных компьютеров включал такие модели, как IMSAI 8080, Radio Shach TRS-80 Model I, Apple II и позднее IBM PC.

Операционные системы персональных компьютеров (ПК) могли быть достаточно простыми. Им нужно было обслуживать действие только одной программы, так что их основной задачей стало управление файлами и обеспечение ряда полезных для программиста функций. Для первого поколения ПК пользователем был программист, работавший на языке ассемблера, так как первые машины не имели возможностей для вызова средствами языков высокого уровня (обычно BASIC) системных функций, встроенных в операционную систему. Первыми операционными системами на оснащенных процессорами Intel компьютерах были CP/M и — с появлением IBM PC — MS-DOS. MS-DOS до сих пор наиболее распространенная операционная система и замечательно сохранилась за десятилетие — достаточно большое время в компьютерном мире.

Однако становилось все более ясно, что однозадачный персональный компьютер не является окончательным вариантом. Операционная система, которая может загружать и запускать только одну программу, ограничивает производительность труда пользователей.

Разработчики попытались обеспечить ограниченные формы мультипрограммности в MS-DOS с помощью использования TSR — программ (программ которые завершаются и остаются резидентными в памяти), но это

решение далеко от совершенства. С одной стороны, TSR── программы не обеспечивают реальную многозадачность: если вы выведете TSR — программу на экран, то программа, с которой вы работали до того, остановится. С другой стороны, TSR — программы в связи с ограничениями в системе прерываний конфликтуют друг с другом, так как архитектура MS-DOS не создавалась для их использования.

Термин "многозадачность" аналогичен термину " мультипрограммирование". Оба относятся к системе, в которой несколько программ могут находиться в памяти и выполняться одновременно. Однако "задача" означает не совсем то же самое, что "программа", как мы увидим позднее, когда будем рассматривать процессы и цепочки.

Можно запустить версию многозадачной многопользовательской операционной системы на персональном компьютере, используя такие системы как Unix или Xenix. Но они в действительности используют персональный компьютер как миникомпьютер, создавая систему с разделением времени и принося с этим всю оболочку многопользовательской системы. Однако такие системы не будут оптимальными с позиций одного пользователя.

Многие компании предлагали операционные системы, которые имели некоторые черты многозадачной системы, например, DESQView фирмы QuarterDeck Office System, Windows фирмы Microsoft и PC-MOS фирмы Software Link. Среди систем такого класса следует выделить операционную систему OS/2, имеющую ряд отличительных особенностей, позволяющих определить ее такими фирмами как IBM и Microsoft как наиболее перспективную ОС.

Таким образом, OS/2 является основной однопользовательской многозадачной операционной системой, разработанной специально для персонального компьютера. 80-е годы это период интенсивной разработки и распространения дружественных операционных систем.

— *ПЯТОЕ ПОКОЛЕНИЕ*. Современные операционные системы обеспечивают дружественный пользовательский интерфейс для различных категорий пользователей. Операционные системы реализуют такие элементы человеко-машинного взаимодействия, как многооконный интерфейс с пользователем, переключение между программами (и их окнами) и т.д. Операционные системы содержат мощный набор графических примитивов, обеспечивающих стандартизацию работы на различных типах графических устройств.

Таким образом, в течение последних 30 лет был разработан ряд удачных операционных систем, некоторые из них приведены в таблице табл. 2.1.

2.3. КОНЦЕПЦИЯ ПОСТРОЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Часть операционной системы, постоянно находящаяся в основной памяти, называется *ядром системы или резидентной* частью операционной системы.

Программы, вызываемые в основную память для выполнения определенных функций и не хранящиеся там постоянно, называются *транзитными программами* или просто *транзитами*.

Транзиты вызываются в участок основной памяти, который называется транзитной областью управляющей программы. Вся область основной памяти, занимаемая ядром и транзитами, называется областью управляющей программы. Остальная часть основной памяти образует область проблемных программ. Разделение управляющей программы на ядро и транзиты позволяет минимизировать область управляющей программы и, соответственно, увеличить область памяти выделяемой для проблемных программ.

В силу того, что транзиты вызываются на одно и тоже место основной памяти, перекрывая друг друга, они иногда называются еще перекрывающимися программами или программами с запланированным перекрытием.

Операционная система спроектирована для работы в различных режимах. Поэтому на вход системы может одновременно поступать большое количество заявок, выполнить которые сразу невозможно просто из-за того, что нет достаточного количества физических ресурсов. К физическим ресурсам системы относятся центральный процессор, место в основной и вспомогательной памяти, отдельные программы, устройства управления вводом/выводом, каналы, таймер, консоль оператора. Все заявки, поступающие на вход системы, делятся на независимые ***задания***, являющиеся внешней единицей работы для операционной системы. Любая выполняемая работа в системе, называется процессом, являющимся динамическим объектом системы, которому она выделяет ресурсы.

Табл. 2.1

Операционная система	Год разработки	Разработчик
Atlas	1959	Манчестерский университет, Англия
CTSS	1963	Масачусетский технологический институт (MTI) , США
OS/360 (1)	1964	IBM, США
THE	1967	Эйндховенский технологический университет, Голландия
RC 4000	1970	Университет города Орхуса, Дания
OS/360 (21)	1970	IBM, США
UNIX (1)	1972	Bell Laboratories, США
Multics	1975	Bell Laboratories, MTI, США

MVS	1976	IBM, США
VMS	1980	Digital Equipment Corp., США
CP/M	1983	Digital Research Inc., США
MS-DOS	1983	Microsoft Corp., США
UNIX (7.5)	1983	Bell Laboratories, США
UNIX (V)	1985	Bell Laboratories, США
OS/2 Warp	1994	IBM, США
WINDOWS NT	1993-94	Microsoft Corp., США

Выполнение пользовательских процессов в операционной системе осуществляется на двух уровнях: уровне пользователя и уровне ядра. Когда процесс производит обращение к операционной системе, режим выполнения процесса переключается с режима задачи (пользовательского) на режим ядра: операционная система пытается обслужить запрос пользователя, возвращая код ошибки в случае неудачного завершения операции. Даже если пользователь не нуждается в каких-либо определенных услугах операционной системы и не обращается к ней с запросами, система еще выполняет учетные операции, связанные с пользовательским процессом, обрабатывает прерывания, планирует процессы, управляет распределением памяти и т.д. Большинство вычислительных систем разнообразной архитектуры (и соответствующие им операционные системы) поддерживают большее число уровней, чем указано здесь, однако уже двух режимов: режима задачи и режима ядра, вполне достаточно для операционной системы.

Основные различия между этими двумя режимами:

- В режиме задачи процессы имеют доступ только к своим собственным инструкциям и данным, но не к инструкциям и данным ядра (либо других процессов).
- В режиме ядра процессам уже доступны адресные пространства ядра и пользователей. Например, виртуальное адресное пространство процесса может быть поделено на адреса, доступные только в режиме ядра, и на адреса, доступные в любом режиме.

Некоторые машинные команды являются привилегированными и вызывают возникновение ошибок при попытке их использования в режиме задачи. Например, в машинном языке может быть команда, управляющая регистром состояния процессора; процессам, выполняющимся в режиме задачи, она недоступна.

Проще говоря, любое взаимодействие с аппаратурой описывается в терминах режима ядра и режима задачи и протекает одинаково для всех пользовательских программ, выполняющихся в этих режимах. Операционная система хранит внутренние записи о каждом процессе, выполняющемся в системе. На Рисунке 2.1. показано это разделение: ядро делит процессы A, B, C и D, расположенные вдоль

горизонтальной оси, аппаратные средства вводят различия между режимами выполнения, расположенными по вертикали. Несмотря на то, что система функционирует в одном из двух режимов, ядро действует от имени пользовательского процесса. Ядро не является какой-то особой совокупностью процессов, выполняющихся параллельно с пользовательскими, оно само выступает составной частью любого пользовательского процесса.

Рис. 2.1. Процессы и режимы их выполнения

2.4. КЛАССИФИКАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ И ИХ СТРУКТУРНЫЕ ОСОБЕННОСТИ

Различают следующие типы операционных систем:

- Однопрограммные ОС
- Многопрограммные ОС
- Однопроцессорные ОС
- Многопроцессорные ОС
- Распределенные ОС
- Виртуальные ОС

Система работает в *однопрограммном* режиме если в системе находится одна или несколько задач, но все ресурсы системы отданы одной задаче, и она не может быть прервана другой. Задача возвращает ресурсы только при нормальном или аварийном завершении.

Система работает в *многопрограммном* режиме, если в ней находится несколько задач в разной стадии исполнения, и каждая из них может быть прервана другой с последующим возвратом.

Многопрограммный режим в однопроцессорной системе — это организация вычислительного процесса с планированием во времени. При этом операционная система обеспечивает выполнение активизированных процессов на одном и том же оборудовании, разделяя (синхронизируя) их работу во времени. В функции такой операционной системы также входит защита влияния одного процесса на другой. Основные заботы о синхронизации взаимодействующих процессов возложены на программиста.

Многопрограммный режим в многопроцессорной системе — это планирование во времени и в пространстве. В этом режиме операционная система, выполняя одну из своих основных функций (обеспечение эффективности работы ресурсов), должна распределять активизированные процессы по процессорам (в пространстве) и синхронизировать их работу во времени. В том случае, если количество задач больше количества процессоров, задача планирования, диспетчеризации усложняется. В связи со сложностью решения задач распределения процессов по процессорам, их решение выполняется или до активизации процессов в многопроцессорной системе (статическое планирование), или решаются простыми способами, не дающими оптимального решения.

В настоящее время применение систем массового распараллеливания и распределенных систем обработки информации, когда вычислительная система может в большинстве случаев выделить столько вычислительных ресурсов, сколько требуется, временную координату планирования можно исключить, что значительно облегчает задачи операционной системы по планированию и организации вычислительных процессов.

- ***Режимы эксплуатации вычислительной системы***

Бурный рост потребностей в высокопроизводительных и надежных средствах вычислений привел к появлению и развитию вычислительных систем (ВС). Пока рано говорить о достаточно полной и точной классификации ВС, так как развитие их структур и способов функционирования продолжается, и почти каждая новая разработка ВС вносит новые идеи, заставляющие изменять принятую классификацию. Тем не менее, можно указать некоторые, уже установившиеся признаки, по которым удается классифицировать существующие и проектируемые ВС (рис.2.2).

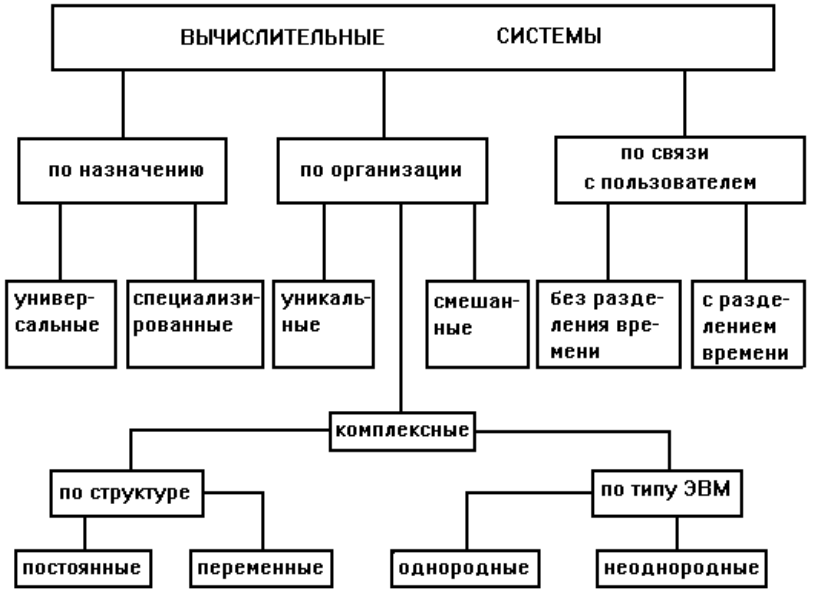

Рис.2.2

В приведенной классификации не нашли отражения способы использования ВС при решении задач и организации вычислительного процесса в системе машин. Первый опыт эксплуатации ВС показал, что большие возможности этих систем могут быть полностью раскрыты и использованы только при применении эффективных методов и средств организации работы. Обзор литературы по применению вычислительной техники позволяет обобщить различные формы эксплуатации ВС и представить их в виде определенной классификационной схемы (рис. 2.3). При этом следует заметить, что в настоящее время наибольшее внимание уделяется организации работ в параллельных системах.

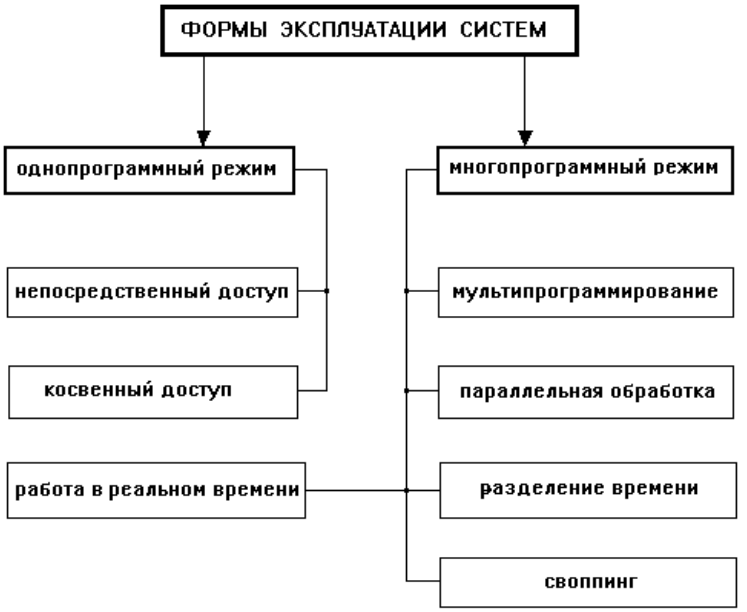

Рис. 2.3

- *Однопрограммный режим работы*

Как уже отмечалось, вычислительная система (машина) работает в однопрограммном режиме, если в системе находится одна или несколько задач, но все рессурсы отданы одной задаче, и она не может быть прервана для выполнения другой.

В начальной стадии применения ЭВМ пользователь сам работал за пультом управления, осуществляя ввод своей программы, ее запуск и наблюдал за выходом результатов по мере их получения.

Время реакции пользователя при непосредственном доступе велико по сравнению со временем реакции машины, что чрезвычайно невыгодно с точки зрения эффективного использования мощности последней. Действительно, пользователь определенное время набирает с клавиатуры информацию, или обдумывает свое очередное действие, в это время ЭВМ простаивает. При пакетной обработке доступ к ЭВМ осуществляется косвенно: посредством управляющей программы, обеспечивающей переход от одной задачи к другой и контроль за их выполнением. При этом, значительно сокращаются непроизводительные потери

машинного времени, повышается эффективность использования ресурсов ЭВМ, но, в то же время, полностью исключается возможность диалога человек — машина.

При пакетном режиме предусматривается последовательная обработка задач. Однако в некоторый момент времени только одна из задач пакета выполняется и занимает все ресурсы, а остальные задачи (программы) пакета находятся в состоянии ожидания своей очереди. Во время выполнения одной из программ пакета запрещается ее прерывание с целью перехода к другой программе пакета. Переход к очередной программе разрешается либо после завершения текущей программы, либо в случае ее аварийного останова.

Для внесения малейшего изменения в программу необходимо ожидать ввода в машину очередного пакета задач. Таким образом, при косвенном доступе к ЭВМ время реакции машины в подавляющем большинстве случаев превышает время реакции пользователя, а это снижает эффективность работы пользователя.

При пакетной обработке задач в однопрограммном режиме функционирования ВС возникает необходимость выбора оптимальной последовательности выполнения задач с заданными сроками реализации на одном обслуживающем устройстве. В случае, если возможности ВС являются постоянными, а заявки (задачи пакета), которые должны быть обслужены, уже имеются в наличии, необходимо определить оптимальную, в некотором смысле, очередность обслуживания заявок .

- *Многопрограммный режим работы*

Система работает в многопрограммном режиме, если в системе находится несколько задач на разной стадии выполнения, и каждая из них может быть прервана другой с последующим восстановлением.

Основной проблемой многопрограммной работы является организация защиты программ от взаимного влияния как на уровне оперативной памяти, так и на разных уровнях внешней памяти. Необходимость организации защиты возникает вследствии того, что различные программы пишутся независимо друг от друга в расчете на единоличное использование ресурсов системы. Это создает опасность взаимного влияния программ друг на друга и должно быть учтено при реализации системы.

Разделение аппаратных и программных ресурсов системы ставит сложные задачи управления перед СУПЕРВИЗОРом, которые он решает благодаря наличию специальных алгоритмов распределения рессурсов системы.

Если количество задач превышает количество процессоров, то возникает проблема определения (планирование) очередности их решения во времени. Планирование во времени — это решение множества задач на ограниченных ресурсах. При организации (планировании) работы параллельной вычислительной ВС и распределении задач между множеством процессоров (ресурсов) планирование осуществляется в пространстве и во времени (рис. 2.4).

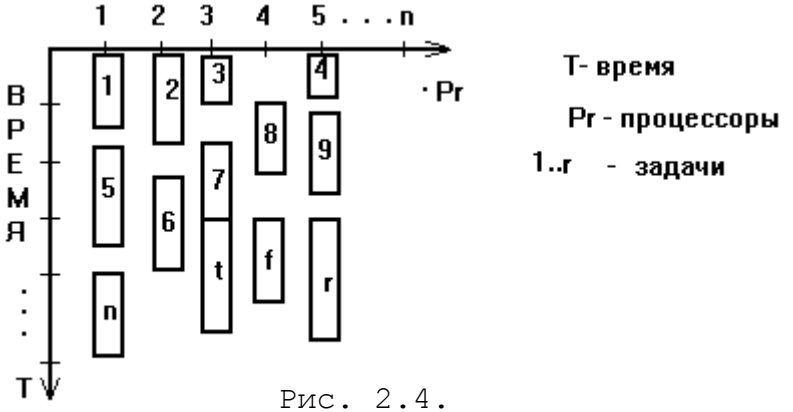

Рис. 2.4.

Как только количество процессоров становится больше, чем количество независимо решаемых задач, надобность во временной координате планирования пропадает.

Принято различать пять способов реализации многопрограммного режима работы:

- Классическое мультипрограммирование;
- Параллельная обработка;
- Разделение времени;
- Свопинг.

Классическое мультипрограммирование — это режим истинного совмещения, когда параллельно исполняемые задачи занимают различное оборудование.

Режим мультипрограммирования характеризуется тем, что правила перехода от программы к программе устанавливаются из соображений достижения максимального использования ресурсов машины путем более широкого использования совмещения их действий. Дорогостоящий ЦП все-таки простаивает, когда данные вводятся или выводятся, несмотря на то, что процесс ввода/вывода все время ускоряется. Тогда было предложено другое решение — поместить в память одновременно несколько программ. Память поделена на участки, и каждая программа располагается в своем участке. Теперь, при выполнении ввода/вывода для одной программы, вторая программа может производить вычисления, т.е. процессор переключается на выполнение другой программы (рис. 2.5). Этот подход, при котором несколько программ делят память и по очереди используют ЦП и другие ресурсы, работающие автономно, называется мультипрограммированием.

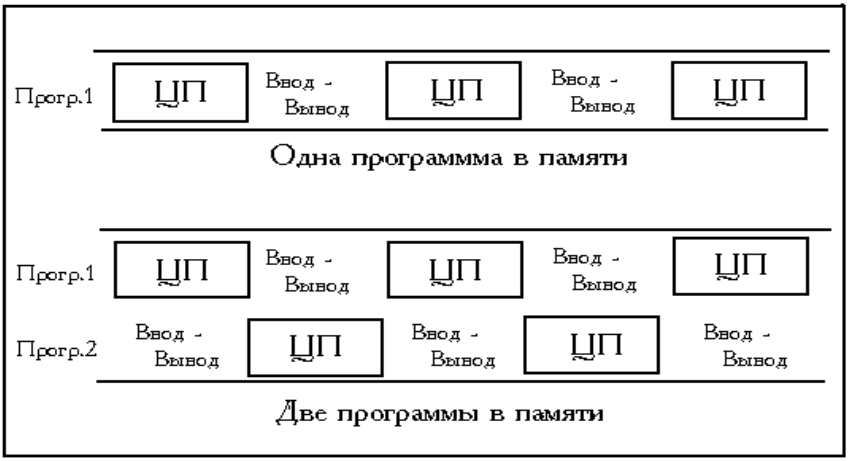

Рис. 2.5. Многозадачный и последовательный подход

При мультипрограммировании улучшение качества обслуживания пользователя непосредственно не предусматривается. Может случится так, что одна из программ, захватив один из ресурсов, например процессор, "заблокирует" выполнение других программ. Кроме того, мультипрограммирование, в чистом виде, не совместимо с непосредственным доступом, так как это один из видов пакетной обработки задач, то есть все пользователи получают результаты одновременно, как и в режиме косвенного доступа. Тем не менее, общее время обработки пакета задач при мультипрограммировании оказывается меньше, чем при последовательной пакетной обработке, за счет лучшего использования оборудования.

Производительность системы, работающей в мультипрограммном режиме, во многом зависит от состава группы программ, исполняемых совместно. Действительно, почти неизбежно возникновение отдельных моментов времен, когда все программы ожидают ввод или вывод данных, а центральное устройство простаивает. Отсюда следует задача целенаправленного формирования пакета программ с целью сокращения непроизводительных простоев оборудования.

На рис. 2.6.б показан пример выполнения трех задач в мультипрограммном режиме. При этом необходимо учитывать приоритеты задач при выборе задачи, которой необходимо дать предпочтение в случае множественных требований к некоторым ресурсам.

При уменьшении абсолютного времени решения, время решения отдельных заявок может увеличиваться по сравнению с однопрограммным режимом работы. рис. 2.6.б

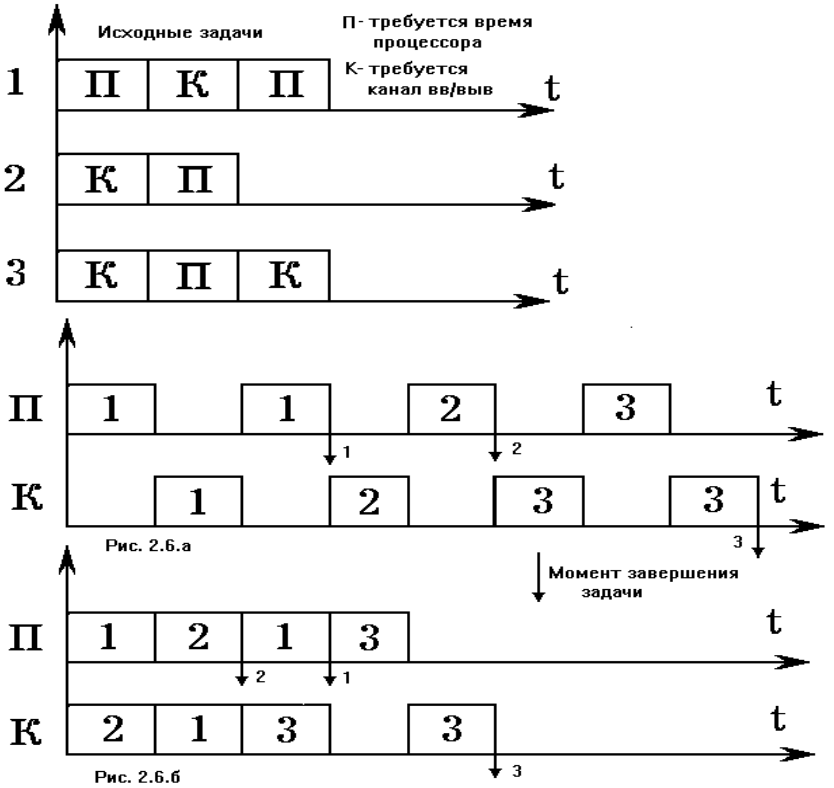

Рис. 2.6.а

Рис. 2.6.б

Если количество однотипных устройств увеличить, то и количество задач, обрабатываемых одновременно, будет увеличиваться. Для такой системы вводится критерий — степень мультипрограммирования. Степень мультипрограммирования, определяет количество заявок при которых система работает наиболее эффективно, а дальнейшее увеличение количества заявок не приводит к повышению эффективности работы вычислительной системы (рис. 2.6.б).

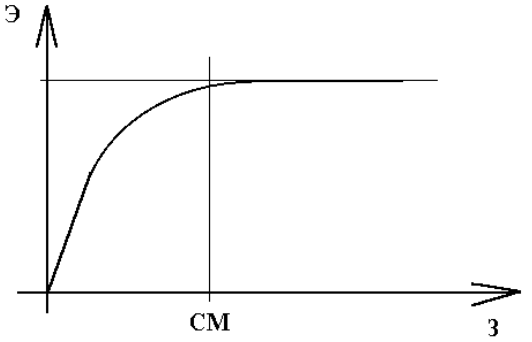

Рис.2.6.б. Э — эффективность, З — задачи,
СМ — степень мультипрограммирования

Под режимом *параллельной обработки* данных нескольких задач понимается такой многопрограммный режим, в котором переход от одной задачи к другой происходит через достаточно короткие промежутки времени (кванты), сравнимые с тактом машины, чтобы создать у пользователя впечатление одновременности исполнения нескольких программ (режим кажущегося совмещения).

Основной целью данного режима является улучшение обслуживания пользователей, выражающееся в том, что у последних создается впечатление отсутствия очереди, так как решение их задач не прерывается на длительные отрезки времени. Кроме того, если пользователю предоставляются некоторые средства прямого доступа (хотя бы для вывода информации), то это впечатление еще более усиливается выдачей результатов по мере их получения. Параллельная обработка основана на значительном отличии времени реакции пользователя и машины вследствии высокой скорости работы последней.

Однако, в общем случае, время ответа при этом режиме не оптимально, так как требуется тонкий механизм учета приоритета задач, учитываемый при определении очередности их обработки. Наибольшее применение получили алгоритмы, основанные на принципе сканирования очереди. Время выполнения пакета задач при параллельной обработке значительно превышает время их выполнения при единоличном использовании машины. Тем не менее, параллельная обработка задач, при которой пользователь может получать результаты, не ожидая конца обслуживания всего пакета, уменьшает время ожидания ответа по сравнению с режимом последовательной обработки.

В большинстве случаев параллельная обработка сочетается с мультипрограммированием, что обеспечивает наличие двух типов прерываний, вызывающих переход к другой программе: естественное прерывание во время ожидания ввода/вывода (мультипрограммирование); вынужденное прерывание в конце отрезка времени, отводимого каждой программе (параллельная обработка).

Разделение времени — наиболее развитая форма многопрограммной работы, совмещающая мультипрограммирование с непосредственным доступом и обеспечением доступа некоторых привилегированных пользователей к ресурсам системы. Время ответа в такой системе близко к тому, которое было при единоличном использовании машины.

Как и при параллельной работе, задачи выполняются поочередно в течение некоторого отрезка времени (кванта), по завершении которого происходит вынужденное прерывание. Длительность этих отрезков времени (квантов) не является, как правило, фиксированной, а изменяется в зависимости от рассматриваемой задачи, а также от конкретных условий эксплуатации в момент передачи управления от одного пользователя другому. Выбор пользователя, задаче которого будет передано управление, и выделение этой задаче кванта времени осуществляет управляющая программа, являющаяся составной частью программ — диспетчеров, предназначенных для работы с разделением времени. Диспетчер функционирует в соответствии с выбранной для данной системы дисциплиной обслуживания заявок, которые будут рассмотрены ниже.

Существует не менее интересный способ реализации многопрограммного режима работы на однопроцессорной машине — "свопинг". Он используется при недостаточной памяти, для хранения всех параллельно выполняемых программ.

Свопинг характеризуется тем, что после определенного времени (кванта) программы пользователей, находящиеся в оперативной памяти, переписываются на внешний носитель информации (диск) в специально выделенные для этого своп-файлы, т.е. сохраняется состояние этих задач. Следующая совокупность параллельно выполняемых программ занимает освободившееся место в оперативной памяти, и система исполняет их в многопрограммном режиме до завершения следующего кванта.

- *Формы взаимодействия человека с машиной*

Учитывая вышесказанное, можно выделить следующие формы взаимодействия пользователя с машиной:

1) Непосредственный доступ (НД)
2) Косвенный доступ (Косв.Д)
3) Коллективный доступ (КД)

Непосредственный доступ (НД) подразумевает, что все ресурсы вычислительной системы находятся в полном распоряжении пользователя.

Косвенный доступ (Косв.Д) — режим работы пользователя с вычислительной системой с помощью управляющей программы (косвенно), для которой пользователь должен подготовить информацию, определяющую ее действия для выполнения его работы.

Коллективный доступ (КД) подразумевает работу многих пользователей с ресурсами системы, т.е. разделение ресурсов (оборудования, времени, программ,

данных) между многими. В этом режиме, как правило, пользователь общается с ЭВМ посредством терминального устройства, имеющего ограниченные возможности по вводу и выводу информации и отсутствие вычислительных.

В настоящее время принципы коллективного доступа значительно усилились за счет использования интеллектуальных терминалов — персональных ЭВМ. Сочетание вычислительных возможностей персональной ЭВМ и использование, в случае необходимости, всей вычислительной мощности вычислительной среды, к которой этот терминал подключен, определяет практически неограниченные возможности выполнения задач пользователя. Такую форму взаимодействия (НД1) можно определить как непосредственный доступ на новом качественном уровне.(рис. 2.7) Несомненно эта форма взаимодействия человека с машиной (или лучше с вычислительной средой) требует совершенно новых принципов организации вычислительной системы и ее функционирования.

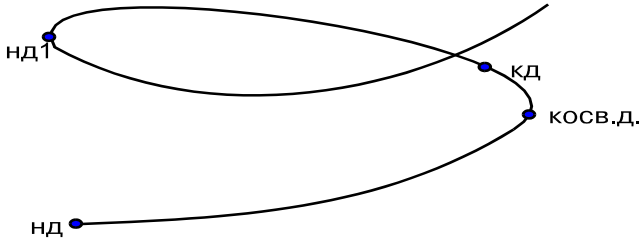

Рис. 2.7. Эволюционное развитие формы взаимодействия человек — машина

Для достижения большей производительности ВС в систему включают несколько независимых процессоров, взаимодействующих через оперативную память или специальный канал связи. Подобная организация ВС привела к мультипроцессорным вычислительным системам, способным параллельно выполнять несколько независимых программ или несколько независимых ветвей одной программы. Высокие требования к надежности и живучести вычислительной системы предопределили наличие в ней как минимум двух процессоров и возможность двухмаршрутной связи между модулями системы. Свойства мультипроцессорных вычислительных систем отражают две основные тенденции современного развития вычислительной техники:

- Модульный принцип построения технического обеспечения, при котором каждое устройство выполняется в виде независимого модуля, что позволяет: во первых, наращивать структуру и, во-вторых, достигается возможность сделать систему менее уязвимой к отказам в силу взаимозаменяемости однотипных модулей.
- Распараллеливание программ (выделение параллельных участков) и параллельное выполнение независимых ветвей одной программы или выполнение полностью независимых программ, что позволяет уменьшить

общее время реализации задач и повысить эффективность работы оборудования.

Второе свойство отличает мультипроцессорные системы от мультипрограммных, в памяти которых хранится несколько программ, совмещаются работы внешних и центральных устройств и выполнение отдельных операций центральных устройств. Основная цель мультипрограммирования ── наиболее полное использование оборудования ── лучше всего достигается при непрерывном функционировании в системе большого количества программ, когда несмотря на прерывания, и, следовательно, некоторое увеличение времени выполнения одной программы, время решения многих задач уменьшается по сравнению с временем их решения на машине без совмещения операций.

Многопроцессорная архитектура включает в себя два и более ЦП, совместно использующих общую память и периферийные устройства (рис. 2.8), располагая большими возможностями в увеличении производительности системы, связанными с одновременным исполнением процессов на разных ЦП. Каждый ЦП функционирует независимо от других, но все они работают с одним и тем же ядром операционной системы. Поведение процессов в такой системе ничем не отличается от поведения в однопроцессорной системе ── с сохранением семантики обращения к каждой системной функции — но при этом они могут открыто перемещаться с одного процессора на другой. Хотя, к сожалению, это не приводит к снижению затрат процессорного времени, связанного с выполнением процесса.

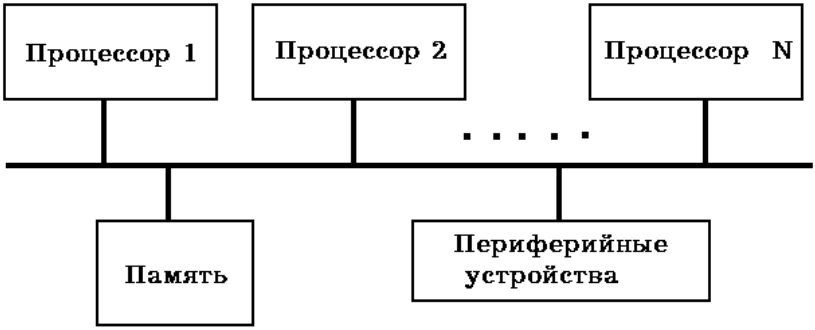

Рис.2.8. Многопроцессорная конфигурация

Отдельные многопроцессорные системы называются системами с присоединенными процессорами, поскольку в них периферийные устройства доступны не для всех процессоров.

Параллельная работа нескольких процессоров в режиме ядра при выполнению различных процессов создает ряд проблем, связанных с сохранением целостности

данных и решаемых благодаря использованию соответствующих механизмов защиты

Если вопрос об опасности возникновения нарушения целостности оставить открытым, как бы редко подобные нарушения ни случались, ядро утратит свою неуязвимость и его поведение станет непредсказуемым. Избежать этого можно тремя способами:

- Исполнять все критические операции на одном процессоре, опираясь на стандартные методы сохранения целостности данных в однопроцессорной системе;
- Регламентировать доступ к критическим участкам программы, используя элементы блокирования ресурсов;
- Устранить конкуренцию за использование структур данных путем соответствующей переделки алгоритмов.

Главный процессор несет ответственность за обработку всех обращений к операционной системе и всех прерываний. Подчиненные процессоры ведают выполнением процессов в режиме задачи и информируют главный процессор о всех производимых обращениях к системным функциям.

Мультипроцессорная система, в отличии от мультипрограммной, позволяет уменьшить время решения каждой отдельной задачи за счет одновременного выполнения независимых участков ее программы и достигнуть высокого коэффициента использования оборудования при непрерывной обработке большого количества программ.

Распределенные операционные системы представляют собой совокупность вычислительных средств и периферийных устройств, объединенных между собой с обеспечением возможности взаимного обмена информацией.

Свойства распределенных операционных систем:

- Отсутствие общей памяти приводит к тому, что нельзя определить общее состояние системы с помощью множества совместных переменных, а невозможность совместного обращения к памяти и различия в задержке передач сообщений приводят к тому, что при определении состояния какого-либо элемента системы из двух различных точек можно получить разные результаты, в зависимости от порядка предшествующих событий;
- Распределенная система распределяет выполняемые работы в узлах системы, исходя из соображений повышения пропускной способности всей системы;
- Распределенные системы имеют высокий уровень организации параллельных вычислений.

Два фактора увеличивают вероятность отказов отдельных элементов (по сравнению с централизованными системами):

1. Большое число элементов системы.
2. Надежность систем связи обычно меньше, чем надежность информационных систем.

Однако надежность всей системы в целом в распределенных операционных системах весьма высока за счет специфических свойств, позволяющих исключить

или ослабить эффект отказа отдельных элементов системы (исправление вышедших из строя элементов, переконфигурация системы и т.д.).

Архитектура распределенной системы представлена на рис. 2.9. Каждый компьютер, показанный на рисунке, является автономным модулем (вычислительным узлом), состоящим из ЦП, памяти и периферийных устройств. Соответствие модели не нарушается даже несмотря на то, что компьютер располагает только локальной файловой системой. В случае необходимости, он должен иметь периферийные устройства для связи с другими машинами, а все необходимые ему файлы могут располагаться и на ином компьютере. Физическая память, доступная каждой машине, не зависит от процессов, выполняемых на других машинах. Этой особенностью распределенные системы отличаются от сильносвязанных многопроцессорных систем. Соответственно, и ядро системы на каждой машине функционирует независимо от внешних условий эксплуатации распределенной среды.

Распределенные системы традиционно делятся на следующие категории:

- *периферийные системы*, представляющие собой группы машин, отличающихся ярко выраженной общностью и связанных с одной (обычно более крупной) машиной. Периферийные процессоры делят свою нагрузку с центральным процессором и переадресовывают ему все обращения к операционной системе. Цель периферийной системы состоит в увеличении общей производительности сети и в предоставлении возможности выделения процессора одному процессу в операционной системе. Система запускается как отдельный модуль; в отличие от других моделей распределенных систем, периферийные системы не обладают реальной автономией, за исключением случаев, связанных с диспетчеризацией процессов и распределением локальной памяти.

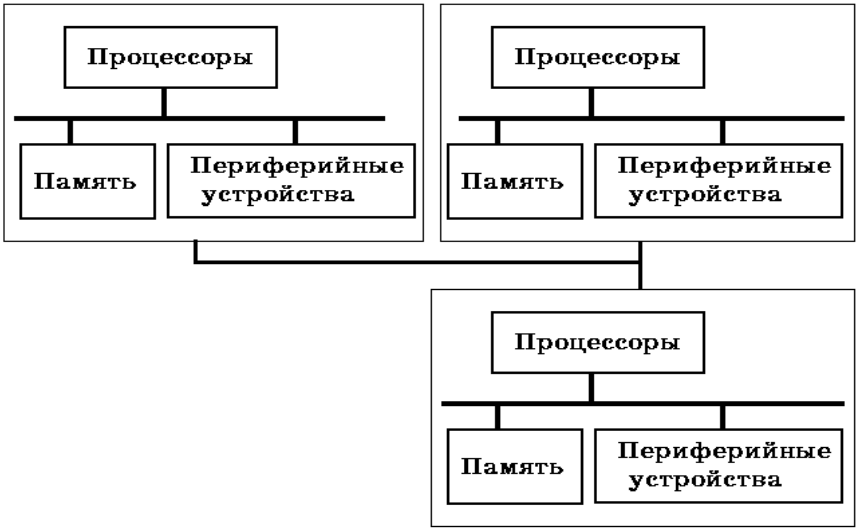

Рис. 2.9. Модель системы с распределенной архитектурой

- *распределенные системы* типа "Newcastle", позволяющие осуществлять дистанционную связь по именам удаленных файлов в библиотеке Удаленные файлы имеют спецификацию (составное имя), которая в указании пути поиска содержит специальные символы или дополнительную компоненту имени, предшествующую корню файловой системы. Реализация этого метода не предполагает внесения изменений в ядро системы, вследствие этого он более прост, чем другие методы, но менее гибок.
- *абсолютно "прозрачные" распределенные системы*, в которых для обращения к файлам, расположенным на других машинах, достаточно указания их стандартных составных имен; распознавание этих файлов как удаленных входит в обязанности ядра. Маршруты поиска файлов, указанные в их составных именах, пересекают машинные границы в точках монтирования, сколько бы таких точек ни было сформировано при монтировании файловых систем на дисках.

Виртуальная операционная система характеризуется обеспечением многопрограммного режима работы, когда каждый пользователь может работать в собственной операционной среде.

2.5. ПОНЯТИЕ ГЕНЕРАЦИИ, ИНСТАЛЛЯЦИИ, ИНИЦИАЛИЗАЦИИ

При покупке вычислительной системы приобретается и программное обеспечение для выполнения тех функций, ради которых эта ВС и покупалась. Как правило, в это ПО включены все необходимые программы для полной поддержки всех соответствующих частей ВС. Таким образом, покупается исходный вариант операционной системы или *дистрибутив.* В него входят различные варианты драйверов для различных аппаратных компонент, а также программы для обеспечения различных режимов работы ВС. Это драйверы CD-ROMa, сетевых Ethernet карт, SCSI-устройств, Sound Blastera, принтера и т.д. Причем в состав дистрибутива могут входить варианты драйверов для поддержки различных локальных шин (VESA, PCI) и других аппаратных платформ. Кроме этого, дистрибутив имеет все необходимые программы, поддерживающие работу вычислительной системы в различных режимах работы, например, однопрограммном или многопрограммном, многопроцессорном, сетевом.

Для нормального функционирования вычислительной системы в реальных условиях эксплуатации совокупность программ всего исходного ПО не нужна и из всего их множества выбирается некое подмножество программ, обеспечивающих работу вычислительной установки в установленных пользователем режимах и на имеющемся оборудовании. Процесс отбора из дистрибутива тех программных модулей, которые будут использоваться называется *генерацией* операционной системы. Процесс генерации выполняется либо с использованием специального "языка генерации", либо с помощью организации диалога пользователя с системой в результате которого определяются желания и требования пользователя к функционированию системы, а система в соответствии с этим определяет совокупность необходимых программ, поддерживающих эти требования.

Место, где находится резиденция системы, называется " резидентным" томом, а сама система "резиденцией". Из всех программных модулей в резидентном томе часть используется по мере необходимости, а некоторые из них обеспечивают базовое функционирование ВС и составляют *резидентные* программы, находящиеся в системной области оперативной памяти. Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Некоторые программы, связанные с выполнением функций ОС, но не находящиеся постоянно в оперативной памяти называются *транзитными*. Эти программы вызываются в память по мере необходимости. Кроме этого, пользователь может по своему желанию определить некоторые программы как резидентные. Все остальные программы являются транзитами, вызываются динамически и могут затирать друг друга.

При функционировании вычислительной системы пользователь может менять свои представления о ее функционировании. Кроме этого может меняться и состав оборудования. Процесс локальной настройки ОС по желанию пользователя называется *инсталляцией*. Признаком, по которому можно отличить инсталляцию от генерации является то, что, если при настройке операционной системы не

используется дистрибутив — это инсталляция, если требуется дистрибутив — генерация.

При загрузке ОС необходимо выполнить связывание, размещение всех частей, входящих в ядро супервизора, т.е. размещение резидентных программ на своих местах, формирование специальных системных структур данных в области данных операционной системы, формирование постоянной области, активизацию процессов для начала работы операционной системы. Этот процесс называется *инициализацией* операционной системы.

2.6. КОНЦЕПЦИЯ РЕСУРСОВ В СОВРЕМЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Под **ресурсом** абстрактно можно понимать "нечто" (реально существующее), которое может понадобиться для выполнения программы (задачи, процесса). Все ресурсы можно разделить условно на две категории:

Физические устройства, входящие в состав современной ЭВМ.

Внутренние ресурсы пользовательских программ (данные, подпрограммы и т. д.).

В качестве физических устройств можем рассматривать следующие:

* оперативная память (или ОЗУ), где хранятся данные, программы, различные системные объекты управления, векторы прерываний, информация о внешних устройствах, таблицы оверлеев, системные буферы, блоки управления активизированных процессов (РСВ) и т.д.
* процессор(ы), который выполняет работу, определяемую операционной системой (выполняет обработку данных);
* клавиатура как устройство ввода;
* принтер как устройство печати (вывода);
* терминал как устройство вывода информации;
* магнитные, жесткие или гибкие диски как устройства ввода/вывода.

Чаще всего при описании ресурсов рассматривают процессор и оперативную память (ОП). ОП выделяется пользователю для записи данных, программ, размещения блоков управления процессами и так далее.

Если ОП достаточно велика для того, чтобы разместить все необходимые программы, данные и вспомогательные таблицы, то особых затруднений в размещении и использовании пространства памяти нет.

Иначе обстоит дело с процессором. Это самое динамичное, с точки зрения управления и планирования его работой, устройство.

Под **выделением процессора** понимается выделение его времени для обработки различных задач, находящихся в системе.

Наибольшую трудность для ОС может представлять планирование процессов для выполнения на процессоре. Для этого существует множество дисциплин управления очередями, но обязательным является использование приоритетов,

которое позволяет отсортировать имеющиеся в системе и готовые к выполнению процессы.

Наиболее приоритетный процесс и получает в свое распоряжение время процессора (квант) для выполнения. Разработчики ОС стремятся достигнуть максимально возможной загрузки процессора (наиболее эффективной).

К внутренним ресурсам системы можно также отнести различные данные (константы, переменные), подпрограммы или даже целые программы (библиотеки программ, используемые пользовательскими программами в процессе выполнения).

В многопользовательских или многопроцессорных системах к некоторым программам часто обращаются или они могут использоваться несколькими процессами одновременно. Поэтому для повышения эффективности работы системы стремятся выделить из этих программ ядро — наиболее используемые данные и процедуры программы, — для использования этими процессами.

Некоторый ресурс может потребоваться сразу нескольким процессам, однако, в любой момент времени он может быть выделен только одному из них. В таком случае остальные процессы выстраиваются в очередь (отсортированную по приоритету) на выделение ресурса. Когда процесс освободит ресурс, он может быть выделен первому процессу в очереди ожидающих, а если такового нет, то ресурс считается свободным.

Такая задача согласования работы нескольких процессов при использовании общего ресурса называется *задачей взаимного исключения*, т.е. первый процесс в очереди ожидания к общему ресурсу получает его, а остальным доступ к ресурсу запрещен.

2.7. ПРОЦЕСС — ОСНОВА ФУНКЦИОНИРОВАНИЯ ВС

Процесс — это теоретическое понятие, на основании которого можно описать то, что происходит в системе при выполнении некоторых действий (программы).

Для пользователей знать, что происходит в системе при выполнении программы, не обязательно — наглядно показывает результат. Однако, для разработчиков систем необходимо знать, что происходит при выполнении той или иной операции, т.е. изменение состояния машины на физическом уровне. Понятие процесса, его развитие во времени и в пространстве в распределенных системах обработки информации помогают создать некоторую условную модель выполнения программы.

Состояние машины определяется состоянием процессора (содержимым адресуемых и внутренних регистров) и состоянием памяти (содержимым ячеек памяти). Это состояние меняется при выполнении процессором некоторых операций. Выполнение операции — это некоторое действие, результатом которого является переход за конечное время из начального состояния машины (до выполнения операции) в конечное (после ее выполнения). В глобальном масштабе такой операцией может быть и вся программа. Однако, на таком уровне мы не

можем ввести понятие процесса. Тогда рассмотрим каждую такую операцию как неделимую, т.е. выполняемую процессором за один шаг (единицу времени, машинный такт).

С этой точки зрения программу можно рассматривать как последовательность элементарных (неделимых) операций. Будем отслеживать выполнение программы в течение времени T (от начала до конца выполнения). За это время процессор успеет выполнить N операций (N сколь угодно большое), которые будем идентифицировать по номеру (1,..,N).

В таких элементарных операциях можно выделить два момента: начало операции — H и конец операции — K. В эти моменты, времена которых обозначили соответственно tH и tK, будем отмечать состояния машины. Эти два момента считаются *событиями*. Таким образом, события позволяют отмечать изменения состояния машины.

При выполнении программы имеем последовательность операций 1,2,..,N, причем tH(1)<tK(1)=<tH(2)<...<tK(N) (хотя конец предыдущей операции и начало следующей — это в принципе одно и то же). Такая последовательность и называется *последовательным процессом* (или просто *процессом*). События H(1), K(1), H(2),..., K(N) образуют *временной след* (трассу или историю) процесса.

Таким образом, можно выделить три наиболее употребляемых определения процесса.

Процесс :

* это любая выполняемая работа в системе;
* это динамический объект системы, которому она выделяет ресурсы;
* это траектория процессора в адресном пространстве ВС.

2.7.1. Классификация процессов

По способу создания процессы делятся на:

- Системные процессы. Они создаются при загрузке системы, имеют оверлейную или динамически-последовательную структуру программ.
- Проблемные (пользовательские) процессы. Создаются при активизации работы (задачи пользователя) операционной системой. Под активизацией процесса здесь понимается начало выполнения программы, подчиненной этому процессу, т.е. начало процесса.

Пользовательский процесс может быть активизирован только другим процессом, для этого и существуют системные процессы, активизируемые ОС при запуске.

По способу существования процессы делятся на:

* Последовательные. Это процессы, которые выполняются друг за другом, т.е. когда закончится выполнение первого процесса, начинается второй и т.д.
* Параллельные. Процессы, которые могут выполняться одновременно. *Параллельные процессы* могут быть:

Независимые, т.е. выполняют разные операции и не имеют общих частей.

Асинхронные — это процессы, которые должны синхронизироваться и взаимодействовать друг с другом. Взаимодействие — это передача данных одним процессом другому. Также процессы могут иметь общие части — процедуры, данные, требуемые (но еще не выделенные) ресурсы. При этом возникает задача взаимного исключения.

При инициализации ядра и системы активизируются процессы, подчиненные ОС — системные процессы, обеспечивающие выполнение основных функции ОС.

Особое значение для функционирования вычислительной системы имеет процесс, связанный с обработкой прерываний. Он является первым процессом, который активизируется при загрузке и инициализации ядра.

Должен быть активизирован процесс администратора памяти (содержащийся в системном файле msdos.sys) — это процесс, который управляет эффективным распределением памяти для размещения различных программ. Также необходима активизация процесса для работы с внешними устройствами и управления данными (управления файловой системой).

Выделяется отдельно процесс, подчиненный часам, который активизируется на аппаратном уровне.

2.7.2. Состояния процессов

Во время своего существования процесс может находиться в четырех состояниях: ***подготовленном, готовом, активном и заблокированном*** (рис.2.10). В каждый конкретный момент времени процесс находится в одном из этих состояний, которое фиксируется в блоке управления процессом (PCB).

Процесс находится в **подготовленном** состоянии, если он находится в системе (как правило на внешнем носителе информации), но ему не выделены ресурсы системы.

Процесс находится в **готовом** состоянии или активизирован, если ему уже выделены ресурсы (программа, необходимая для выполнения, находится в оперативной памяти), однако ему не выделено время процессора для выполнения (процессор занят другим процессом).

Процесс, которому выделяется время процессора, переводиться в **активное** состояние или состояние выполнения.

Во время выполнения процесса ему могут понадобиться дополнительные ресурсы, но для их получения необходимо некоторое время. Т.е. процесс должен ожидать наступления некоторого события или окончания выполнения конкретной операции (например, ввода/вывода для получения/выдачи данных). Такой процесс переводится в **заблокированное** состояние.

Процессы при поступлении в систему находятся в подготовленном состоянии и накапливаются во входных очередях заданий. Следует различать процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и процессы, находящиеся в подготовленном состоянии после загрузки операционной системы. Процессы первого вида, как

правило, принадлежат пользователям, а второго вида — ОС. Такие процессы находятся в подготовленном состоянии до тех пор, пока для выполнения функций системы они не будут активизированы. Пользовательские процессы активизируются или делается попытка их активизации при наличии в системе ресурсов. Попытка активизации процесса производится системой при наличии в системе оперативной памяти, достаточной для размещении программы, подчиненной процессу.

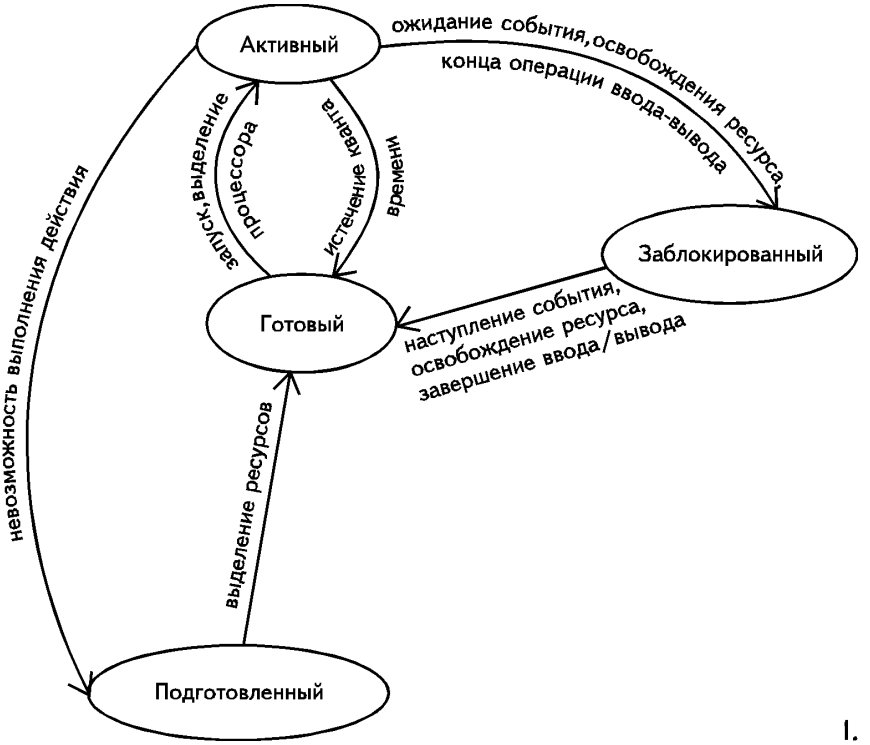

Рис. 2.10. Переход процесса из состояния в состояние

Поэтому в системе может находиться одновременно довольно много подготовленных и готовых процессов, для которых организуются очереди (одна для готовых и одна для подготовленных процессов). Дисциплины управления ими могут быть различными и выбор дисциплины обслуживания определяется требованиями, предъявляемыми к системе планирования.

Если система определила необходимость активизации процесса и выделяет нужные ему ресурсы, кроме времени процессора, то она переводит его в готовое состояние. При этом система определяет (если может) имеются ли в системе ресурсы, необходимые для выполнения данного процесса. Процесс, который переводится в готовое состояние, должен иметь больший приоритет, чем процессы, требующие таких же, как и он, типов ресурсов (т.е. он находится в начале очереди подготовленных процессов). При этом сортировку по приоритетам процессов в очереди и планирование перехода в готовое состояние (следя за освобождением ресурсов) должен осуществлять планировщик данной очереди.

Если процессор освободился, то первый (наиболее приоритетный) процесс из очереди готовых процессов получает время процессора и переходит в активное состояние. Выделение времени процессора процессу осуществляет диспетчер.

Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.

Процесс, который требует дополнительных ресурсов, кроме времени процессора, либо выполнения некоторой операции другим процессом, переводится в заблокированное состояние и ожидает наступления события, которое он потребовал. Как только произошло ожидаемое событие, процесс переводится в готовое состояние.

Если процесс требует действия или ресурса, которых в данный момент операционная система не может выполнить, он переводится в подготовленное состояние и считается приостановленным.

Задача ОС заключается в слежении за состоянием всех процессов, находящихся внутри системы, особенно в активизированном состоянии, поскольку в активном состоянии процесс является монополистом по отношению к процессору. Поэтому ему и выделяется квант времени, чтобы процесс не блокировал процессор или не занимал его на длительное время, иначе могут возникнуть тупиковые ситуации. В некоторых ОС процесс, требующий наименьшего времени обслуживания на процессоре, имеет наибольший приоритет. Тогда быстрые заявки быстрее покинут систему, а процессы, требующие на обслуживание довольно большого кванта времени при большой загрузке системы вообще могут не обслужиться. С этой точки зрения наиболее приоритетными оказываются системные процессы, так как в ОС все заложено для их быстрого выполнения.

Приоритеты программ обработки прерываний зависят от употребляемости соответствующих ресурсов. Наибольший приоритет имеет программа обработки прерывания по таймеру (вызывается 18,2 раза в секунду) и наименьший — прерывание по вводу/выводу. Пользовательские процессы имеют различные приоритеты в зависимости от порождающих их процессов и от времени выполнения на процессоре. Также необходимо следить за очередью блокированных процессов, с тем, чтобы среди них не было "бесконечно" ожидающих процессов. Вполне возможно, что событие, которое они ожидают, вообще может не произойти — тогда их надо вывести из системы. Также может оказаться, что блокированный процесс имеет слишком низкий приоритет и он может бесконечно долго находиться

в режиме бесконечного откладывания в очереди блокированных процессов — тогда необходима система динамического, адаптивного изменения приоритетов.

В общем случае, система должна следить за процессами в очередях готовых и заблокированных процессов, чтобы не было бесконечно ожидающих процессов или процессов, монопольно удерживающих выделенные им ресурсы. Из сказанного следует, что эффективность системы управления процессами в значительной степени влияет на характеристики ВС, связанные с пропускной способностью. Плохая организация управления процессами может привести к тому, что некоторые процессы могут так и остаться в подготовленном состоянии.

2.7.3. Блок управления процессом (PCB — process control block)

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью **блока управления процессом** (PCB). **Вход в процесс** (фиксация системой процесса) — это создание его блока управления (PCB), а **выход из процесса** — это его уничтожение, т. е. уничтожение его блока управления.

Таким образом для каждого активизированного процесса система создает PCB, в котором в сжатом виде содержится информация о процессе, используемая при управлении. PCB — это системная структура данных, содержащая определенные сведения о процессе и имеющая следующие поля:

1. Уникальный индентификатор процесса (имя)
2. Текущее состояние процесса.
3. Приоритет процесса.
4. Указатели участка памяти выделенного программе, подчиненной данному процессу.
5. Указатели выделенных ему ресурсов.
6. Область сохранения регистров.
7. Права процесса (список разрешенных операций)
8. Связи зависимости в иерархии процессов (список дочерних процессов, имя родительского процесса)
9. Пусковой адрес программы, подчиненной данному процессу.

Когда ОС переключает процессор с процесса на процесс, она использует области сохранения регистров в PCB для запоминания информации, необходимой для рестарта (повторного запуска) каждого процесса с точки прерывания, когда он в следующий раз получит в свое распоряжение процессор. Количество процессов в системе ограничено и определяется самой системой, пользователем во время генерации ОС или при загрузке. Неудачное определение количества одновременно исполняемых программ может привести к снижению полезной эффективности работы системы, т.к. переключение процессов требует выполнения дополнительных операций по сохранению и восстановлению состояния процессов. Блоки управления системных процессов создаются при загрузке системы. Это

необходимо, чтобы система выполняла свои функции достаточно быстро, и время реакции ОС было минимальным. Однако, количество блоков управления системными процессами меньше, чем количество самих системных процессов. Это связано с тем, что структура ОС имеет либо оверлейную, либо динамически ── последовательную структуру иерархического типа, и нет необходимости создавать для программ, которые никогда не будут находиться одновременно в оперативной памяти, отдельные РСВ. При такой организации легко учитывать приоритеты системных процессов, выстроив их по приоритетам заранее при инициализации системы. Блоки управления проблемными (пользовательскими) процессами создаются в процессе активизации процессов динамически. Все РСВ находятся в выделенной системной области памяти.

В каждом РСВ есть поле состояния процесса. Все блоки управления системными процессами располагаются в порядке убывания приоритетов и находятся в системной области памяти. Если приоритеты системных блоков можно определить заранее, то для проблемных процессов необходима таблица приоритетов проблемных программ. Каждый блок РСВ имеет стандартную структуру, фиксированный размер, точку входа, содержит указанную выше информацию и дополнительную информацию для синхронизации процессов. Для синхронизации в РСВ имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

В цепочке связи указывается адрес РСВ вызываемого (поле 1) и вызывающего (поле 2) процесса.

В цепочке ожидания, в поле 3 указывается адрес РСВ вызываемого процесса, если вызываемый процесс занят. В поле 4 занятого процесса находится число процессов, которые ожидают данный.

Если процесс А пытается вызвать процесс В, а у процесса В в РСВ занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания РСВ процесса А, а в поле счетчика ожидания РСВ процесса В добавляется 1. Как только процесс В завершает выполнение своих функций, он передает управление вызывающему процессу следующим образом: В проверяет состояние своего счетчика ожидания, и, если счетчик больше 0, то среди РСВ других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 РСВ которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передается этому процессу.

Пример: Имеем связь процессов X и В, и процесс А вызвал процесс В (рис.2.11).

Процесс X Процесс B

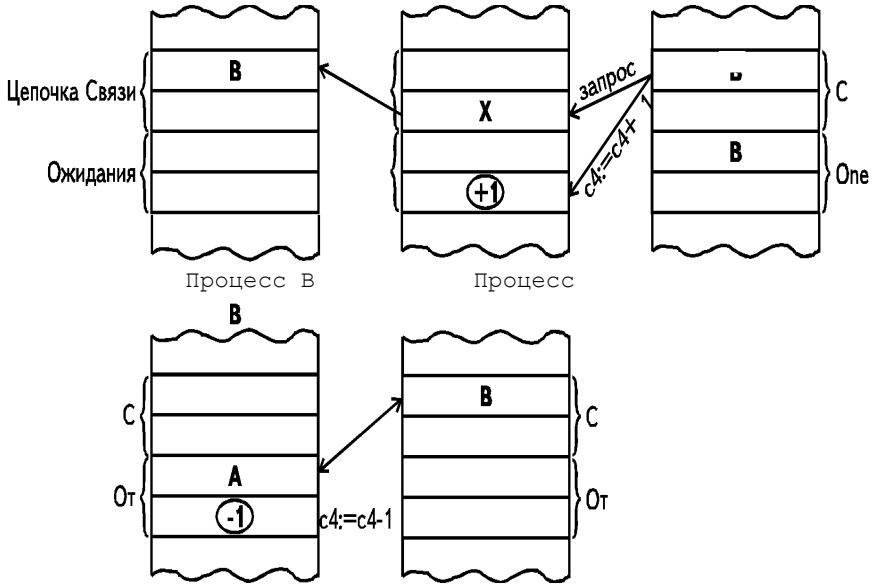

Рис. 2.11. Состояние после выполнения процесса X

2.7.4. Операции над процессами

Над процессами можно производить следующие операции: создание; уничтожение; возобновление; изменение приоритета; блокирование; пробуждение; запуск (выбор).

1. **Создание** процесса включает в себя:
   а) присвоение имени процессу,
   б) включение этого имени в список имен процессов, известных системе,
   в) определение начального приоритета,
   г) формирование блока управления процессом,
   д) выделение начальных ресурсов.
2. **Уничтожение** процесса означает его удаление из системы. Ресурсы, выделенные этому процессу, возвращаются системе, имя в системных списках стирается и блок управления процессом освобождается.
3. **Изменение приоритета** означает просто модификацию значения приоритета в блоке управления данным процессом. В основном увеличивают приоритет у

процессов, когда предполагают, что они будут находиться в состоянии бесконечного откладывания или уменьшают приоритет процессу, надолго захватившему процессор.

4. **Запуск** (выбор) процесса, осуществляемый планировщиком, который выделяет процессу время процессора.
5. **Приостановленный** процесс не может продолжить свое выполнение до тех пор, пока его не активизирует любой другой процесс (кратковременное исключение определенных процессов в периоды пиковой нагрузки). В случае длительной приостановки процесса, его ресурсы должны быть освобождены. Решение об освобождении ресурса зависит от его природы. Основная память должна освобождаться немедленно, а вот принтер может и не быть освобожден. Приостановку процесса обычно производят в следующих случаях:
   а) если система работает ненадежно и может отказать, то текущие процессы надо приостановить, исправить ошибку и затем активизировать приостановленные процессы;
   б) если промежуточные результаты работы процесса вызвали сомнение, то нужно его приостановить, найти ошибку и запустить либо сначала, либо с контрольной точки;
   в) если ВС перегружена, что вызвало снижение ее эффективности;
   г) если для продолжения нормального выполнения процессу необходимы ресурсы, которые ВС не может ему предоставить.

   Инициатором приостановки может быть либо сам процесс, либо другой процесс. В однопроцессорной ВС при однопрограммном режиме работы выполняющийся процесс может приостановить сам себя или быть принудительно остановиться с пульта управления без возможности восстановления, т.к. ни один другой процесс не может выполняться одновременно с ним. В мультипроцессорной системе или при многопрограммном режиме работы любой выполняющийся процесс может быть приостановлен другим процессом.
6. **Возобновление** (или активизация) процесса ── операция подготовки процесса к повторному запуску выполняемая операционной системой, с той точки, в которой он был приостановлен.

2.7.5. Иерархия процессов

Процесс может породить новый процесс. При этом первый, порождающий процесс, называется **родительским**, а второй, порожденный процесс, — **дочерним**. Для создания некоторого процесса необходим только один родительский процесс (рис. 2.12.).

Таким образом, в системе создается иерархическая структура процессов. На самом верху иерархии (рис. 2.12) находятся системные процессы (процесс А на рисунке), которые в свою очередь порождают дочерние процессы и т.д. Получаем некоторое дерево процессов, которое может расти вниз и иметь столько ярусов,

сколько позволяет система. Причем у дочерних процессов может быть только один родительский, но у родительского может быть несколько дочерних.

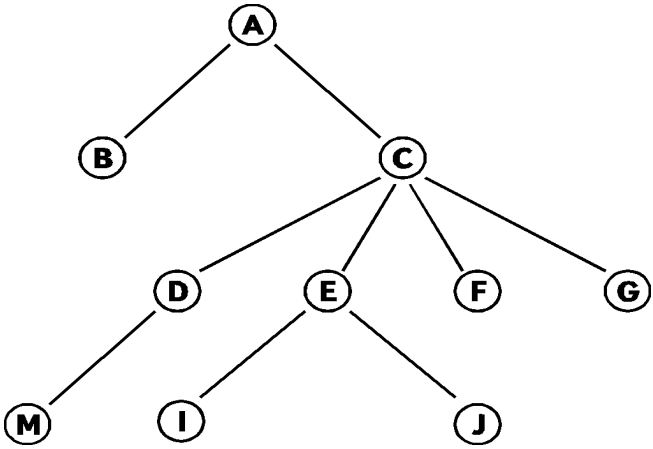

Рис. 2.12. Иерархия процессов

Уничтожение процессов на нижнем ярусе иерархии не вызывает затруднений, т.к. они не имеют дочерних процессов. При уничтожении родительского процесса в некоторых системах удаляются и дочерние процессы, а в некоторых дочерние процессы начинают существовать независимо от родительского, уничтоженного. При создании процесса в системе UNIX текущий процесс разветвляется на два независимых параллельных процесса: родительский и дочерний. Они не имеют общей первичной памяти, но могут совместно использовать все открытые файлы. Для дочерних процессов создаются копии всех сегментов данных, в которые разрешена запись. Создается процесс примитивом fork, значение которого позволяет затем узнать какой из процессов является дочерним, а какой — родительским. Родительский процесс может быть уничтожен раньше дочерних в двух случаях:

1. его выполнение не зависит от результатов, формируемых в дочерних процессах, и он имеет все ресурсы, которые необходимы для его завершения. Тогда родительский процесс завершается независимо от завершения дочерних.
2. ОС считает, что он создает тупиковую ситуацию и удаляет его из системы. Однако, обычно родительский процесс ожидает завершения дочерних, получает от них результаты и завершается сам.

2.7.6. Прерывание. Переключение контекста процесса

Прерывание — это нарушение последовательности выполнения действий (команд), т.е. после текущего действия (команды) выполняется не следующее (команда), а некоторое другое действие.

При появлении сигнала прерывания управление передается ОС, которая запоминает состояние прерванного процесса в области сохранения регистров PCB. Далее ОС анализирует тип прерывания и передает управление соответствующей программе обработки этого прерывания. Инициатором прерывания может быть выполняющийся процесс или оно может быть вызвано некоторым событием, связанным или даже не связанным с этим процессом.

Рассмотрим механизм передачи управления **программе обработки прерывания (IH)**. Как было сказано выше, ОС запоминает состояние прерванного процесса и передает управление IH. Эта операция называется **переключением контекста**. При реализации переключения используются **слова состояния программы (PSW)**, с помощью которых осуществляется управление порядком выполнения команд. В **PSW** содержится информация относительно состояния процесса, обеспечивающая продолжение прерванной программы на момент прерывания.

Существуют три типа PSW: текущее, новое и старое PSW (рис.2.13.). Адрес следующей команды (активной программы), подлежащей выполнению, содержится в текущем PSW, в котором также указываются и типы прерываний, разрешенных и запрещенных в данный момент.

Процессор реагирует только на разрешенные прерывания, а запрещенные игнорирует или задерживает их выполнение. Адрес следующей команды, хранящийся в **текущем PSW**, в командном цикле передается счетчику команд, а адрес новой команды записывается в PSW (по первым двум битам команды можно определить ее длину). Кроме этого, в PSW находятся: признак результата предыдущей команды, поля масок некоторых прерываний (например, каналов и прерываний исключительных ситуаций).

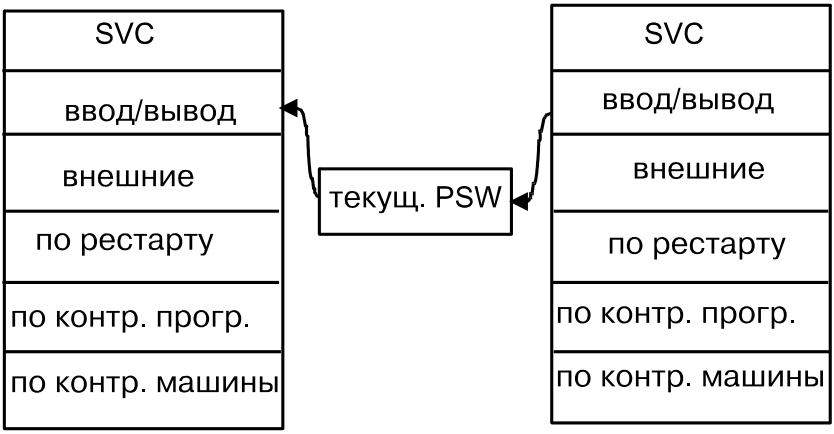

Рис. 2.13

В **новых PSW** содержатся адреса, по которым резидентно размещаются программы обработки прерываний. При возникновении разрешенного прерывания, в одном из **старых PSW** (соответствующем типу прерывания) система сохраняет содержимое текущего PSW — адрес следующей команды данного процесса, которая должна выполниться по окончании обработки прерывания и передаче управления данному прерванному процессу (т.е. адрес команды, которая следует за текущей в данном процессе и которая должна была бы выполниться при отсутствии сигнала прерывания). Таким образом, обеспечивается корректное возвращение в прерванный процесс после обработки прерывания.

Одно из новых PSW, которое содержит адрес обработчика прерывания, соответствующего поступившему сигналу прерывания, записывается в текущее PSW, и происходит переход на соответствующую программу обработки прерывания. Таким образом с помощью механизма смены PSW организуется вход в прерывающую программу. Старые PSW могут храниться в постоянной области памяти вместе с векторами прерываний; в стеке той программы, которая прерывается или в PCB прерванного процесса. Если старое PSW находится в постоянной области памяти, то количество выделенных для этого PSW равно количеству классов прерываний. В этом случае глубина прерываний определяется количеством старых PSW.

При выполнении процедуры выхода из прерывания, ОС может передать управление процессору для продолжения выполнения прерванного процесса, если ОС не допускает перехвата процессора более приоритетному процессу (первому в

очереди готовых процессов), тогда данный процесс переводится в готовое состояние.

2.7.7. Ядро операционной системы

Операции, связанные с процессами, входят в базовое обеспечение машины. Они включаются в комплекс используемых средств с помощью программ и микропрограмм, совокупность которых составляет **ядро ОС**. Таким образом, все операции, связанные с процессами, осуществляются под управлением ядра ОС, которое представляет лишь небольшую часть кода ОС в целом. Поскольку эти программы часто используются, то резидентно размещаются в ОП. Другие же части ОС перемещаются в ОП по мере необходимости. Ядро ОС скрывает от пользователя частные особенности физической машины, предоставляя ему все необходимое для организации вычислений:

- само понятие процесса, а значит и операции над ним, механизмы выделения времени физическим процессам;
- примитивы синхронизации, реализованные ядром, которые скрывают от пользователя физические механизмы перестановки контекста при реализации операций, связанных с прерываниями.

Выполнение программ ядра может осуществляться двумя способами:

1. вызовом примитива управления процессами (создание, уничтожение, синхронизация и т.д.); эти примитивы реализованы в виде обращений к супервизору;
2. прерыванием: программы обработки прерываний составляют часть ядра, т.к. они непосредственно связаны с операциями синхронизации и изолированы от высших уровней управления.

Таким образом, во всех случаях вход в ядро предусматривает сохранение слова состояния (при переключении контекста в PSW) и регистров процессора (в PCB), который вызывает супервизор или обрабатывает прерывание.

2.7.8. Функции ядра ОС

Ядро ОС содержит программы для реализации следующих функций:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- диспетчеризацию заданий, процессов и ресурсов;
- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействия между процессами;
- манипулирование PCB;
- поддержка операций ввода/вывода;

- поддержка распределения и перераспределения памяти;
- поддержка работы файловой системы;
- поддержка механизма вызова── возврата при обращении к процедурам;
- поддержка определенных функций по ведению учета работы машины;

Одна из самых важных функций, реализованная в ядре — обработка прерываний.

В систему поступает постоянный поток прерываний и требуется быстрая реакция на них с тем, чтобы эффективно использовались ресурсы системы, и пользователь как можно быстрее получал ответ (на запрос в виде прерывания). При обработке (дешифрации) текущего прерывания ядро запрещает другие прерывания и разрешает их только после завершения обработки текущего. При большой интенсивности прерываний может сложиться такая ситуация, когда ядро будет блокировать прерывания в течение значительной части времени, т.е. не будет иметь возможности эффективно реагировать на прерывания. Поэтому, ядро ОС обычно осуществляет лишь минимально возможную предварительную обработку каждого прерывания, а затем передает это прерывание на дальнейшую обработку соответствующему системному процессу, после начала работы которого ядро могло бы реагировать на последующие прерывания. Таким образом улучшается реакция системы на прерывания (сигнал прерывания).

2.7.9. Синхронизация процессов

Активизированные в системе процессы могут выполняться параллельно, если им для выполнения не требуется одинаковых ресурсов. Однако, это довольно редкая ситуация в системе. В общем случае несколько процессов могут выполняться параллельно лишь некоторое малое время. В остальное время они синхронизируются или взаимодействуют. **Взаимодействие** — это передача данных одного процесса другому. **Синхронизация** необходима для корректности передачи данных или для обращения к **общему ресурсу** (ОР).

Синхронизация *при передаче данных* заключается в следующем: пока процесс — передатчик не сформирует передаваемые данные и не перешлет их, процесс — приемник не должен выполнять никаких действий, т.е. должен ожидать пересылки данных, и тогда он может продолжить свое выполнение. Процесс — приемник может и сам создать процесс — передатчик, например, при возникновении необходимости ввода данных. При этом процесс — приемник блокируется и передает управление дочернему процессу, по окончании выполнения которого он может продолжить (возобновить) свое выполнение. При реализации такого вида синхронизации необходимо определить порядок предшествования во времени для некоторые точек трасс процессов и определить условие, разрешающее переход некоторых точек трасс на определенные процессы. Эти выделенные точки называются **точками синхронизации** (когда два выполнявшихся параллельно

процесса подошли к точке, в которой необходимо определить, кто из них передатчик и кто приемник; либо точка фиксирует момент, когда создается и запускается дочерний процесс (передается управление от родительского).

Синхронизация *при обращении к ОР* необходима, чтобы исключить одновременный доступ к нему сразу нескольких процессов. Здесь возникает задача взаимного исключения, когда только один процесс может в данный момент "захватить" ОР, а остальные должны ожидать. **Критический участок (КУ)** — это часть процесса, где происходит обращение к ОР. Процесс, который в данный момент обладает ОР, находится в КУ. В период нахождения в КУ процесс является монополистом по отношению к ОР. Поэтому необходимо, чтобы процесс как можно быстрее проходил свой КУ и не блокировался в нем, иначе может сложиться ситуация, когда несколько (очевидно, наименее приоритетных) процессов будут бесконечно долго ожидать выделения ОР. При выходе процесса из КУ, вход туда может быть разрешен одному из ожидающих в очереди к ОР процессов. Это также своего рода синхронизация. Таким образом, процессы, обращающиеся к ОР могут выполняться лишь последовательно.

2.7.10. Примитивы синхронизации

Для синхронизации процессов используются примитивы. Некоторые из них (в основном, семафоры и мониторы) реализованы в ядре (для синхронизации ввода/вывода, переключения контекста (переход по прерыванию) и т.д.).

Примитивы синхронизации могут быть реализованы следующим образом:

1. Самые простые примитивы — **флажки** (примитивы взаимоисключения). Если флажок равен 0, т.е. условие ложно, то процесс не может войти в КУ, т.к. там уже находится другой процесс; если флажок равен 1 (условие истинно), то процесс входит в КУ, обнуляя флажок (если имеется очередь процессов к ОР, то в КУ входит наиболее приоритетный из них). На основе флажков реализованы алгоритмы синхронизации Деккера. Флажки — программная реализация решения задачи взаимного исключения. Это самый низкий уровень.
2. **Команда test_and_set** — аппаратное средство синхронизации (более высокий уровень). Это специальная команда, выполняющая вместе (неделимо) 3 действия:
   - чтение значения переменой;
   - запись ее значения в область сохранения;
   - установка нового значения этой переменной;
3. **Семафор** — некоторая (защищенная) переменная, значение которой можно считывать и изменять только при помощи специальных операций P и V.

   Преимущество по сравнению с test_and_set ── разделение действий на отдельные.

Операция **P(S)**: проверяет значение семафора S; если S>0, то S:=S-1, иначе (S=0) ждать (по S) .

Операция **V(S)**: проверяет, есть ли процессы, приостановленные по данному семафору; если есть, то разрешить одному из них продолжить свое выполнение (войти в КУ); если нет, то S:=S+1.

Операция P должна стоять до входа в КУ , а операция V ── после выхода из КУ.

Недостатки:

1) громоздкость — в каждом процессе каждый КУ должен окаймляться операциями P и V;
2) невозможность решения целого ряда задач с помощью таких примитивов.

4. **Монитор** — примитив высокого уровня. Здесь "забор" ставится вокруг ОР, что удобнее (чем семафоры), т.к. ресурсов в несколько раз меньше, чем процессов (их КУ) и ставить "заборы" вокруг КУ слишком громоздко. Можно сказать, что монитор — это некоторый программный модуль, в котором объединены ОР и подпрограммы для работы с ними. При обращении к ОР, т.е. соответствующей процедуре монитора для работы с ОР, осуществляется вход в монитор. В каждый конкретный момент времени может выполняться только одна процедура монитора — это и есть решение задачи взаимного исключения. Если процесс обратился к процедуре монитора, а другой процесс уже находится внутри монитора, то обратившийся процесс блокируется и переводится во внешнюю очередь процессов ── блокированных по входу в монитор. Процесс, вошедший в монитор (т.е. выполняющий его процедуру), также может быть блокирован им, если не выполняется некоторое *условие* X для выполнения процесса. Условие X в мониторе ── это некоторая переменная типа condition. С ней связывается некоторая внутренняя очередь процессов, блокированных по условию X. Внутренняя очередь приоритетнее внешней. Для блокирования процесса по переменной X и помещения его во внутреннюю очередь по этой переменной используется операция монитора WAIT(X). При выполнении операции SIGNAL(X) из очереди, связанной с переменной X, извлекается и разблокируется первый процесс. Если внутренняя очередь пуста, то в монитор разрешается войти первому процессу из внешней очереди.

2.7.11. Классические задачи, связанные с доступом к ОР

* *Производители — потребители. Понятие кольцевого буфера*

Рассмотрим пару "производитель — потребитель" (рис. 2.14).

Рис. 2.14

В качестве ОР выступает некоторый **буфер** , куда заносятся и откуда считываются данные. Один **процесс**—источник или **производитель**, генерирует информацию, которую другой **процесс**—получатель или **потребитель** использует. Это взаимодействие производится при помощи буфера. Процесс—производитель генерирует данные либо осуществляет некоторые вычисления и результат заносит в буфер, откуда процесс—потребитель считывает данные и печатает их. Если оба процесса работают с примерно одинаковыми скоростями, то проблем передачи/приема практически не существует; в противном случае (резко различные скорости) возникают трудности синхронизации.

Если процесс—производитель работает быстрее, то он может несколько раз перезаписывать данные в буфере, прежде чем потребитель считает их. Происходит потеря информации, чего нельзя допустить. Тогда процесс—производитель должен дожидаться, пока потребитель не считает данные, только тогда он сможет снова заносить данные в буфер. При этом не будет потери информации (скорость обмена процесса—потребителя). Если же процесс—потребитель быстрее, то он будет несколько раз считывать одни и те же данные до замены их производителем. Имеет место дублирование информации. В этом случае потребитель должен дожидаться замены информации в буфере производителем.

В ОС предусматривается выделение некоторого количества ячеек памяти для использования в качестве буфера передач. Этот буфер можно представить в виде массива заданного размера. Процесс — производитель помещает передаваемые данные в последовательные элементы этого массива. Потребитель считывает данные в порядке их поступления. Производитель может опережать потребителя на несколько шагов. Со временем, процесс—производитель заполнит последний элемент выделенного буфера—массива. Когда он сформирует очередные данные для передачи, то должен будет возвратиться к началу буфера и продолжить запись, начиная с первого элемента буфера. Такой массив работает как замкнутое кольцо и носит название **кольцевого буфера**. Так как размер буфера ограничен, процесс—производитель может столкнуться с тем, что все элементы массива заняты. В этом случае производитель должен подождать, пока потребитель не считает и не освободит хотя бы один элемент массива. Может возникнуть ситуация, когда потребитель хотел бы прочитать данные, а массив пуст. Тогда потребитель должен будет ожидать, пока производитель не начнет помещать данные в буфер. Таким образом, такой буфер хорошо использовать для передачи данных от процесса—производителя процессу—потребителю, если они имеют различные скорости. Примером реализации такой схемы является буфер клавиатуры ПК.

* *Читатели—писатели*

Процессы—писатели записывают информацию в некоторый ресурс, например, порт (рис. 2.15).

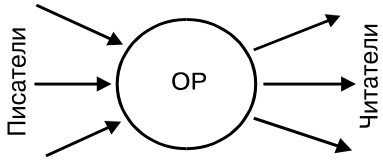

Рис. 2.15

Процессы—читатели должны считывать эту информацию, но записывать не имеют права. Может сложиться ситуация, когда имеется много процессов—писателей, которые поочередно записывают информацию в порт. При этом процессы—читатели могут находиться в ситуации бесконечного откладывания, т.е. ожидания, пока процессы—писатели закончат запись и читатели получат доступ к ОР (порту) для считывания. Поэтому, при появлении нового процесса—писателя приоритет отдается имеющимся в системе процессам—читателям. Однако, теперь уже процессы—читатели могут надолго заблокировать доступ к ОР писателям. В этом случае, при появлении новых процессов—читателей, больший приоритет имеют находящиеся в системе процессы—писатели.

2.7.12. Тупики. Тупиковые ситуации и методы борьбы с ними

Тупик — это:

1. ситуация, из которой система не может выйти;
2. ситуация, когда процесс ждет события, которое никогда не произойдет.

Примером из жизни может служить транспортная пробка на перекрестке.

Второй пример: круговая цепь ожидания (рис. 2.16).

Здесь ресурс 2 выделен процессу 1, а ресурс 1 — процессу 2. В какой-то момент процесс 1 затребовал дополнительно ресурс 1, а процесс 2 — ресурс 2. Естественно, ни один из этих ресурсов не может быть выделен затребовавшему его процессу. Получаем тупиковую ситуацию, при которой ни один из двух процессов не может закончить свое выполнение, пока не будет освобожден требуемый ресурс, чего в данном случае никогда не сможет произойти. Выход заключается в удалении одного из процессов попавших в тупиковую ситуацию из системы.

Тупик ***неизбежен***, если присутствует четыре условия его возникновения:

1. Условие *взаимного исключения*. Процесс обладает монопольным правом владения ресурсами во время всего существования процесса.

2. Условие *ожидания*. Процесс, обладая монопольным правом, пытается захватить новые ресурсы.
3. Условие *перераспределения*. Ресурс нельзя отнять до полного завершения процесса.
4. Условие: *круговая цепь ожидания*. Каждый из процессов цепочки требует дополнительный ресурс, который уже выделен процессу данной цепочки.

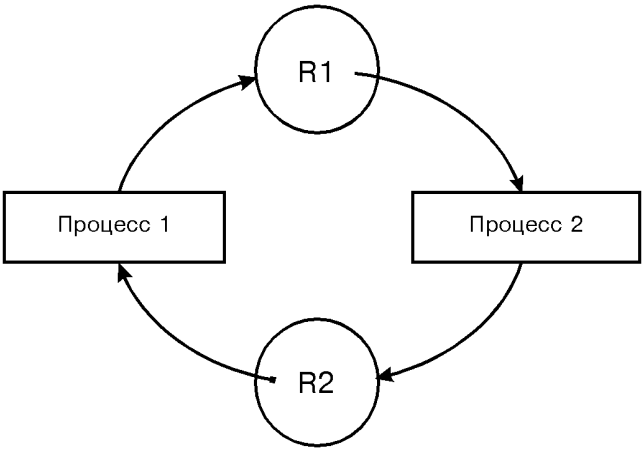

Рис. 2.16

* *Тупики в системе спулинга*

При вводе заданий используется режим **спулинга** — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера. Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.

В таких системах хорошим выходом было бы неполное заполнение буфера (на 75-80%) и передача после этого управления печатающему устройству. По освобождении буфера можно его снова неполностью заполнить, затем снова перейти к распечатке и т.д.

Во многих современных ОС принтер начинает распечатку до заполнения буфера (по мере заполнения). При этом часть буфера (распечатанная) освобождается и туда можно заносить новую информацию. В таких системах вероятность возникновения тупиковой ситуации значительно меньше.

2.7.13. Способы борьбы с тупиками

Последствия тупиков равносильны ситуации бесконечного откладывания. Тупики и ситуация бесконечного откладывания ── это ситуации равносильные "потери процесса".

Можно выделить 4 стратегии борьбы с тупиками:

1. **Предотвращение тупика**. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций. Однако этот метод часто приводит к нерациональному использованию ресурсов.
2. **Обход тупика**. Этот метод предусматривает менее жесткие ограничения, чем первый и обеспечивает лучшее использование ресурсов. Метод учитывает возможность возникновения тупиков, и при увеличении вероятности тупиковой ситуации принимаются меры по обходу тупика.
3. **Обнаружение тупиков**. Требуется установить сам факт возникновения тупиковой ситуации, причем точно определить те процессы и ресурсы, которые в нее включены.
4. **Восстановление после тупиков**. Необходимо устранить тупиковую ситуацию, чтобы система смогла продолжить работу, а процессы, попавшие в тупиковую ситуацию, смогли завершиться и освободить занимаемые ими ресурсы. Восстановление — это серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что вся или почти вся проделанная процессами работа теряется.

2.7.13.1. Предотвращение тупиков

Возникновение тупика невозможно, если нарушается одно из четырех необходимых условий возникновения тупиковых ситуаций. Первое условие при этом можно и не нарушать, т.е. считаем что процесс может обладать монопольным правом владения ресурсами. Нарушая остальные три условия получаем следующие три способа предотвращения тупиков:

I способ. Процесс запрашивает и получает все ресурсы сразу. Если процесс не имеет возможности получить все требуемые ресурсы сразу, т.е. некоторые из них на данный момент заняты, то он должен ожидать освобождения требуемых

ресурсов. При этом он не должен удерживать за собой какие-либо ресурсы. Нарушается условие "ожидания дополнительных ресурсов" (2-е) и тупиковая ситуация невозможна. При этом способе имеем простой процессов, ожидающих выделения ресурсов, и работу вхолостую значительной части ресурсов. Можно прибегнуть к разделению программы на несколько программных шагов, работающих независимо друг от друга. При этом выделение ресурсов можно осуществлять для каждого шага программы, однако увеличиваются расходы на этапе проектирования прикладных программ.

Этот способ имеет два существенных недостатка:

1). Снижается эффективность работы системы;

2). Усиливается вероятность бесконечного откладывания для всех процессов.

II способ. Если процесс, удерживающий за собой определенные ресурсы, затребовал дополнительные и получил отказ в их получении, он должен освободить все ранее полученные ресурсы. При необходимости процесс должен будет запросить их снова вместе с дополнительными. Здесь нарушается условие "перераспределения" (3-е). При этом способе каждый процесс может несколько раз запрашивать, получать и отдавать ресурсы системе. При этом система будет выполнять массу лишней работы — происходит деградация системы с точки зрения полезной работы. Недостатки этого способа:

1. Если процесс в течение некоторого времени удерживал ресурсы, а затем освободил их, он может потерять имевшуюся информацию.
2. Здесь также возможно бесконечное откладывание процессов — процесс запрашивает ресурсы, получает их, однако не может завершиться, т.к. требуются дополнительные ресурсы; далее он, не завершившись, отдает имеющиеся у него ресурсы системе; затем он снова запрашивает ресурсы и т.д.

Имеем бесконечную цепочку откладывания завершения процесса.

III способ. Стратегия линейности. Все ресурсы выстроены в порядке присвоенных приоритетов и захват новых ресурсов может быть выполнен только в порядке возрастания приоритетов. При этом нарушается условие "круговая цепь ожидания" (4-е).

Трудности и недостатки данного способа:

1. Поскольку запрос ресурсов осуществляется в порядке возрастания приоритетов, а они назначаются при установке машины, то в случае введения новых типов ресурсов может возникнуть необходимость переработки существующих программ и систем;
2. Приоритеты ресурсов должны отражать нормальный порядок, в котором большинство заданий используют ресурсы. Однако, если задание требует ресурсы не в этом порядке, то оно будет захватывать и удерживать некоторые ресурсы задолго до того, как появится необходимость их использования — теряется эффективность.
3. Способ не предоставляет удобства пользователям.

2.7.13.2. Обход тупиков

Алгоритм "банкира".

При использовании алгоритма "банкира" подразумевается, что :

1) системе заранее известно количество имеющихся ресурсов;
2) система знает, сколько ресурсов может максимально потребоваться процессу;
3) число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
4) система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
5) процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется **надежным**, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени. Иначе состояние считается **ненадежным**. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Пример решения задачи выделения ресурсов по алгоритму "банкира".

Имеем 6 ресурсов и 6 процессов (рис .2.17.). В таблице показано состояние занятости ресурсов на данный момент (1 свободный ресурс).

Процесс	Выделенное число ресурсов	Требуемое число ресурсов
A	2	3
B	1	3
C	0	2
D	1	2
E	1	4
F	0	5

Рис. 2.17

Эта таблица отражает надежное состояние, т.к. существует такая последовательность выделений — освобождений ресурсов, при которой все процессы за конечное время будут завершены.

Недостатки алгоритма банкира:

1) Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.
2) Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.
3) Число работающих пользователей (процессов) остается постоянным.
4) Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.
5) Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.
6) Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако, для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены).

Графический метод обхода тупика (рис. 2.18).

Имеются 2 процесса P1 и P2 и каждому из них для выполнения требуются по 2 ресурса: Д — диск , П — процессор. Если P1 захватил процессор , а P2 — диск, то возникает тупиковая ситуация. Однако, если P1 захватил П и Д и завершил свое выполнение, то P2 захватывает эти же ресурсы и заканчивает выполнение — беступиковое взаимодействие.

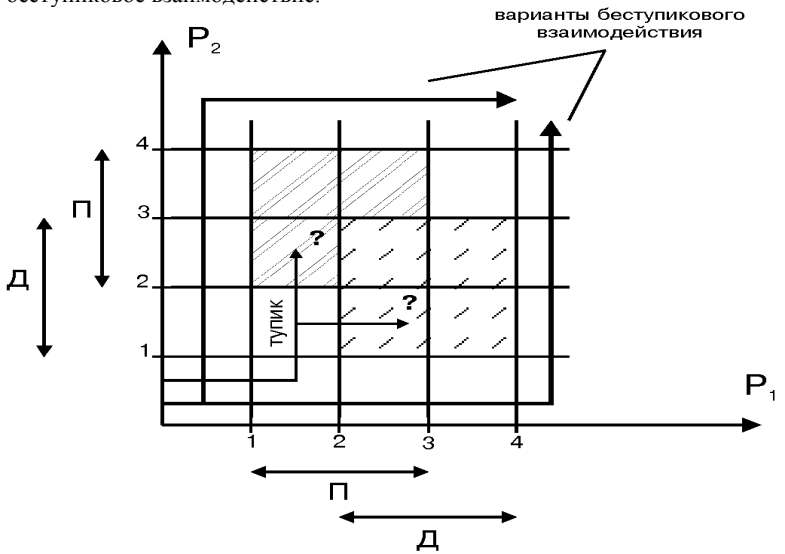

Рис. 2.18

2.7.13.3. Обнаружение тупиков

Обнаружение тупика — это установление факта того, что возникла тупиковая ситуация и определение процессов и ресурсов, вовлеченных в тупиковую ситуацию. Для обнаружения тупиковой ситуации используют операцию **редукции графа**. Эта операция выполняется системой, когда есть подозрение, что какие-то процессы находятся в тупике. При этом определяется, какие процессы могут успешно завершиться (по графу) и освободить ресурсы. Если запросы ресурсов для некоторого процесса могут быть удовлетворены, то граф можно редуцировать на этот процесс. При этом процесс удаляется из графа, включая все дуги от него к требуемым ресурсам и от выделенных ресурсов к нему, т.е. эти ресурсы могут считаться свободными и выделяться другим процессам. Если граф можно редуцировать на все процессы, значит, тупиковой ситуации нет, а если этого сделать нельзя, то все "нередуцируемые" процессы образуют группу процессов, находящихся в тупиковой ситуации.

Пример редуцируемого графа , содержащего 5 процессов и 4 типа ресурсов.

Примечание к рис.2.19.: дуга от процесса к ресурсу (кругу) ⟶ требование ресурса этого типа; тип ресурса - кружок, а сами ресурсы - квадратики внутри круга; P ⟶ обозначение процесса; R ⟶ типа ресурса; дуга от ресурса (квадрата) к процессу ⟶ ресурс выделен процессу.

На рис. 2.19 показан граф беступикового взаимодействия.

* *Восстановление после тупиков.*

Систему, оказавшуюся в тупике, необходимо вывести из него, нарушив одно или несколько необходимых условий его существования. При этом несколько процессов потеряют (полностью или частично) проделанную работу. Однако, это дешевле, чем оставлять систему в тупиковой ситуации. Возникают следующие сложности восстановления системы:

1) В первый момент может быть неочевидно, что система попала в тупиковую ситуацию.
2) В большинстве систем нет достаточно эффективных средств, позволяющих приостановить процесс на неопределенно долгое время, вывести его из системы и возобновить впоследствии. Некоторые процессы (например, процессы реального времени) должны работать непрерывно и не допускают приостановки и последующего восстановления.
3) Если в системе существуют эффективные средства приостановки/возобновления, то их использование требует значительных затрат машинного времени и внимания высококвалифицированного оператора.

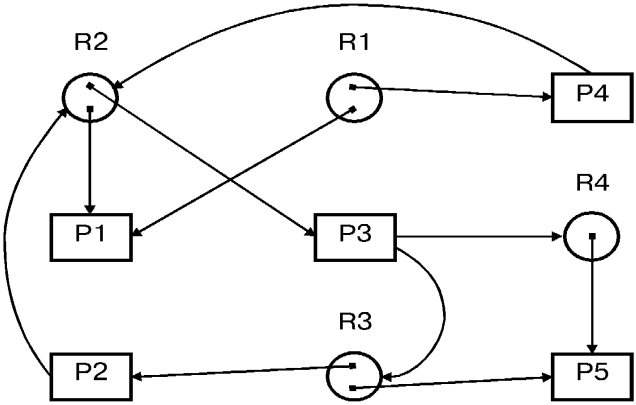

Рис. 2.19

1) Восстановление после тупика небольших масштабов требует и небольшого количества работы; крупный тупик может потребовать довольно большого объема работ по его исключению из системы.

В современных системах восстановление обычно выполняют путем принудительного вывода одного или нескольких процессов из системы, чтобы их ресурсы могли быть использованы. Тогда эти процессы теряются вместе с информацией в них, однако, оставшиеся процессы получают возможность завершить свою работу. Т.е. в этом случае несколько процессов просто убиваются.

При удалении процессов могут возникнуть проблемы с приоритетностью (удаляются обычно наименее приоритетные процессы):

1) Процессы могут не иметь конкретных приоритетов.
2) Значения приоритетов могут оказаться неправильными или могут быть нарушены (например, для бесконечно откладывающегося процесса увеличивается приоритет).
3) Трудность в оптимальном определении, какие из процессов вывести из системы.

Самый эффективный способ восстановления после тупиков — **механизм приостановки/возобновления**. Он позволяет кратковременно переводить процессы в состояние ожидания, а затем активизировать ожидающие процессы, причем без потери информации.

Часто при тупиковых ситуациях требуется перезапуск (рестарт) системы. При этом существует **перезапуск сначала** (вся полученная до рестарта информация теряется) или **с контрольной точки** (теряется только информация после контрольной точки). Система не определяет контрольных точек — их должен предусмотреть программист (для конкретного процесса).

2.8. МЕХАНИЗМ ПРЕРЫВАНИЙ — ОСНОВА УПРАВЛЕНИЯ ПРОЦЕССАМИ

В этой части рассмотрена концепция прерываний — одного из основных механизмов организации многопрограммных режимов работы и обмена информацией с внешними устройствами. Конкретная реализация системы обработки прерываний рассмотрена на примере операционной системы MS-DOS, реализованной на компьютерах IBM PC XT/AT. Описаны общие вопросы организации системы прерываний, даны основные понятия и определения, обсуждаются проблемы и различные методы реализации аппаратных и программных средств поддержки механизма прерываний.

Практически любая вычислительная система, кроме одного или нескольких процессоров имеет, в своем составе множество устройств, обрабатывающих данные параллельно с процессором, что повышает общую производительность системы. Это внешние накопители, устройства коммутации, дополнительные специализированные процессоры, таймеры и прочие устройства. При такой организации системы возникает проблема реализации взаимодействия между процессором и периферийными устройствами.

2.8.1. Взаимодействие с внешними устройствами

Рассмотрим такое взаимодействие подробнее. Устройство может *независимо* от процессора выполнять операции ввода/вывода, а иногда и помещать результат в оперативную память (используя прямой доступ к памяти). Если процессору необходимо принять или передать данные при помощи внешнего устройства, он посылает устройству запрос на выполнение определенных действий. Если устройство готово его обработать, оно выполняет программу ввода/вывода и удовлетворяет запрос, а процессор получает результат его работы. Таким образом, процессору необходимо активизировать устройство, назначить ему работу и получить результат (либо информацию о завершении обработки). Рассмотрим возможные варианты организации такого обмена:

1. Процессор передает устройству необходимую для работы информацию, а затем *постоянно* проверяет состояние устройства до появления в нем признака окончания обработки, после чего (если необходимо) принимает результат. Такой режим называют **режимом непосредственного (программного) опроса** *(Polling mode)*.
2. Процессор помещает необходимые устройству данные в определенное место, откуда устройство их считывает, затем выполняет необходимую работу и располагает результат также в определенном месте, из которого процессор сможет их получить. Это развитие режима непосредственного опроса, исключающее прямое взаимодействие процессора с устройством, получило название **системы с почтовым ящиком** *(Mailbox system)*.

3. Процессор посылает устройству запрос и выполняет процесс, не связанный с окончанием работы внешнего устройства. Устройство выполняет свои функции, а после формирования результата посылает процессору специальный сигнал, сообщающий о том, что обработка закончена. Процессор обращается к устройству и получает необходимую информацию. Именно такой метод реализуется при помощи **системы прерываний** *(Interrupt system)*.

Эти методы различны по объему требуемых для их реализации ресурсов, по времени получения результата взаимодействия процессора и периферийного устройства и по временным затратам на ожидание взаимодействия. Так, режим непосредственного опроса требует минимального количества ресурсов и позволяет получить результат наиболее быстрым образом. Однако, на ожидание готовности устройства процессор тратит довольно много времени, особенно, если устройство имеет невысокую скорость обработки данных (для некоторых устройств ввода/вывода, например, для печатающих устройств, скорость обработки информации в тысячи и миллионы раз ниже, чем у процессора), а это значит, что во время взаимодействия процессор большую часть времени тратит зря.

Системы с "почтовыми ящиками" позволяют организовать более независимую работу процессора и внешнего устройства, но это увеличивает время получения результата и затраты на дополнительное оборудование.

Прерывания сводят время бесполезного ожидания к нулю, но реализация системы прерываний требует наибольших затрат оборудования, к тому же необходимость переключения выполняемой в процессоре задачи на программу взаимодействия с устройством требует дополнительного времени.

Именно поэтому, в вычислительных системах одновременно могут использоваться различные методы организации взаимодействия процессоров и вспомогательных устройств в зависимости от требований, предъявляемых к такому взаимодействию. Одни устройства могут использовать прерывания, другие — непосредственную связь с процессором или "почтовые ящики". Существуют также комбинации этих методов. Например, разновидностью метода непосредственного опроса является режим, в котором процессор, выполняя основную работу, *прерывается* сигналом таймера через определенные промежутки времени. При этом он проверяет состояние всех подключенных к нему устройств, отмечая произошедшие со времени последнего опроса изменения, и, если таковые присутствуют, проверяет, имеются ли в устройстве необходимые ему данные и принимает имеющуюся для него информацию. Завершив опрос, процессор продолжает выполнение прерванного процесса. Аналогичный вариант возможен и при просмотре "почтовых ящиков". Однако, наиболее часто используется механизм прерываний, особенно в многозадачных системах, так как с точки зрения затраченного времени этот режим наиболее экономный.

Механизм прерываний используется не только при взаимодействии процессора с внешними устройствами. Ведь в качестве ресурса могут выступать не только внешние устройства, но и процессор по отношению к процессам, а также

операционная система по отношению к программе пользователя. Поэтому прерывания являются основой функционирования ОС (особенно многозадачной).

2.8.2. Обработка прерываний

В общем случае под **прерыванием** понимают событие, требующее от системы прекращения работы активного процесса и перехода к обработке другого задания. Вычислительная система, обладающая возможностью использования прерываний, должна содержать целый комплекс программно-аппаратных средств, обеспечивающих регистрацию сигнала прерывания, приостановку активного процесса, переход к обработке прерывания и восстановление прерванного процесса. Совокупность действий, выполняемых после регистрации сигнала прерывания, называется **обработкой прерывания**. **Система обработки прерываний** — комплекс программно-аппаратных средств, обеспечивающих приостановку активного процесса и переход к обработке прерывания.

Рассмотрим работу системы прерываний подробней. В реальной жизни нам очень часто приходится сталкиваться с аналогами прерываний, реализованных в вычислительных системах. К примеру, предположим, что вы читаете книгу, и в это время раздается телефонный звонок. Тогда вы откладываете книгу, подходите к телефону, отвечаете на звонок, а затем возвращаетесь к прерванному на самом интересном месте занятию. В этот момент вам необходимо определить, с какого места продолжить чтение. Вернемся к тому моменту, когда вы отложили книгу. Если вы ожидали важного звонка, то вы могли закрыть книгу, даже не запомнив места, на котором остановились. В другом случае вы могли бы оставить пометку карандашом в том месте, где вас *прервал* телефонный звонок, а на нужной странице положить закладку. Но если книга вас весьма заинтересовала, то вы могли бы не сразу реагировать на звонок, а дочитать до конца предложения, абзаца, а то и до конца "самого интересного места".

Аналогичным образом работают прерывания в вычислительных системах. Когда процессор получает сигнал прерывания, он должен прервать выполняемый в нем процесс. При этом возможны несколько вариантов реакции процессора на сигнал прерывания:

1. Прерывание возможно только после определенных команд, при этом необходимо сохранение минимального количества информации о состоянии системы.
2. Прерывание возможно после завершения любой команды процессора.
3. Прерывание возможно после завершения очередного такта процессора. При этом требуется сохранение большого объема информации.

Большинство процессоров построены таким образом, что сигнал прерывания проверяется ими только после завершения выполнения текущей команды выполняющегося процесса. Однако, в системе могут существовать прерывания, для которых невозможно ожидание завершения очередной команды, например, если возникает ошибка в самой команде. Поэтому в системе могут использоваться

одновременно несколько из перечисленных способов для обработки различных прерываний.

Обнаружив сигнал прерывания, процессор должен запомнить состояние прерванного процесса для того, чтобы продолжить его выполнение после обработки прерывания. После этого в процессор загружается новый процесс, выполняющий обработку прерывания. Эта процедура получила название **переключения контекста** или **переключения состояния**. Она является одной из важнейших в организации работы операционной системы, причем ее реализация требует поддержки и выполнения некоторых функций на аппаратном уровне.

Каждый процессор оснащен аппаратурой для управления последовательностью выполнения команд. Это специальный регистр, постоянно хранящий адрес следующей команды. Он называется **словом состояния программы** *(Program Status Word, PSW),* иначе называемый также **словом состояния процесса** *(Process Status Word, PSW),* **счетчиком программы** *(Program Counter, PC),* **счетчиком адреса программы** *(Program Address Counter, PAC),* **указателем команды** *(Instruction Pointer, IP).* Как минимум, такой регистр должен содержать адрес следующей команды (или последовательности команд), а в дополнение к этому — различную информацию, которую система связывает с процессом.

При переключении контекста процессор сохраняет содержимое PSW, затем помещает в него новое значение, соответствующее процессу, называемому **обработчиком прерывания** *(Interrupt Handler, IH),* который и выполняет обработку самого прерывания. После завершения обработки прерывания в PSW восстанавливается содержимое сохраненного ранее PSW.

В процессе обработки прерывания можно выделить следующие **фазы прерывания** (рис. 2.20):

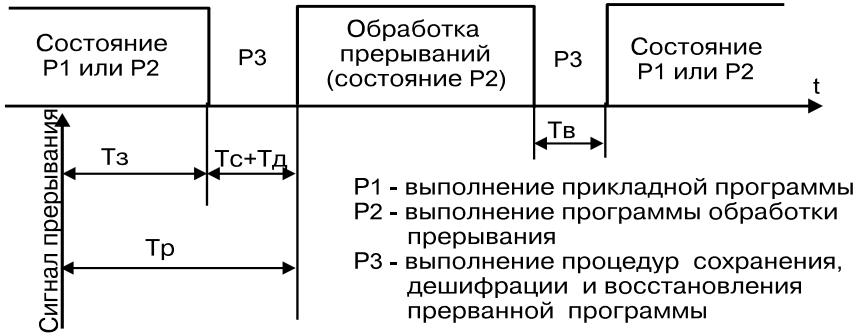

Рис. 2.20. Фазы прерывания

1. $T_з$ — **время задержки** между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания (смотри выше).

2. $T_с$ — **время сохранения** необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.
3. $T_д$ — **время дешифрации** сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.
4. $T_в$ — **время восстановления** прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется **временем реакции системы на сигнал прерывания** ($T_р$).

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях **запросов на прерывания**. Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание. Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

1. Полноупорядоченная схема**.** Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить *все* регистры запросов на прерывания с более высоким приоритетом.
2. Частично упорядоченная схема**.** Все прерывания делятся на **классы** и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.

2.8.3. Классы прерываний

По своему назначению, причине возникновения прерывания делятся на различные **классы**. Традиционно выделяют следующие классы:

1. *Прерывания от схем контроля машины*. Возникают при обнаружении сбоев в работе аппаратуры, например, при несовпадении четности в микросхемах памяти.
2. *Внешние прерывания*. Возбуждаются сигналами запросов на прерывание от различных внешних устройств: таймера, клавиатуры, другого процессора и пр.
3. *Прерывания по вводу/выводу*. Инициируются аппаратурой ввода/вывода при изменении состояния каналов ввода/вывода, а также при завершении операций ввода/вывода.
4. *Прерывания по обращению к супервизору*. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции

операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.

5. *Программные прерывания*. Возникают при выполнении **команды вызова прерывания** либо при обнаружении ошибки в выполняемой команде, например, при арифметическом переполнении.

В последнее время принято прерывания 4 и 5 классов объединять в один класс программных прерываний, причем, в зависимости от источника, вызвавшего прерывание, среди них выделяют такие подтипы:

- прерывание вызванное исполнением процессором *команды перехода к подпрограмме обработки прерывания*
- прерывания, возникающие в результате *исключительной* (аварийной) *ситуации* в процессоре (деление на "0", переполнение и т.д.).

В связи с многообразием различных ВС и их постоянным развитием меняется и организация системы прерываний. Так, с появлением виртуальной памяти, появился класс страничных прерываний, который можно отнести и к классу исключительных ситуаций в процессоре; в системах с кэш-памятью существуют прерывания подкачки страниц в кэш-память и т.д.

Во время выполнения обработчика одного из прерываний возможно поступление другого сигнала прерывания, поэтому система должна использовать определенную *дисциплину обслуживания заявок* на прерывания. Обычно различным классам либо отдельным прерываниям присваиваются различные *абсолютные приоритеты*, т.о. при поступлении запроса с высшим приоритетом текущий обработчик снимается, а его место занимает обработчик вновь поступившего прерывания. Количество уровней вложенности прерываний называют **глубиной системы прерываний**. Обработчики прерываний либо дисциплина обслуживания заявок на прерывание должны иметь также специальные средства для случая, когда при обработке прерывания возникает запрос на обработку прерывания от того же источника.

2.9. РЕАЛИЗАЦИЯ МЕХАНИЗМА ПРЕРЫВАНИЙ В КОМПЬЮТЕРАХ ТИПА IBM PC

В компьютерах семейства IBM PC, построенных на основе микропроцессора Intel 80x86 и использующих операционную систему MS-DOS, механизм прерываний реализуется как при помощи аппаратных, так и при помощи программных средств.

Микропроцессор поддерживает обработку 256 различных прерываний, каждое из которых идентифицируется номером в диапазоне 00h-FFh. Сегментные адреса обработчиков прерываний, называемые **векторами прерываний**, содержатся в **таблице векторов прерываний**, которая располагается в начале оперативной памяти (по адресу 0000:0000). Каждый вектор прерывания имеет размер 4 байта,

таким образом, таблица векторов занимает 1024 байта. Для определения адреса обработчика любого прерывания нужно номер прерывания умножить на 4.

В защищенном режиме старшие модели семейства процессоров вместо таблицы векторов используют **таблицу дескрипторов прерываний,** сходную по своему назначению с таблицей векторов.

Операционная система не контролирует содержимое таблицы векторов, хотя и имеются средства для чтения и изменения векторов. При помощи функций MS-DOS значение векторов можно изменить, поместив нужный адрес непосредственно в таблицу векторов прерываний. В этом случае контроль за содержимым векторов и их соответствием размещению обработчиков в памяти полностью ложится на программиста, а последствия в случае ошибки могут быть самыми непредсказуемыми.

Активизация обработчика прерывания может произойти в результате возникновения следующих ситуаций:

- возникновение сигнала внутреннего прерывания внутри микросхемы процессора (**внутренние** или **логические прерывания**)
- поступление в процессор сигнала запроса на прерывание от внешнего (по отношению к процессору) устройства (**аппаратные прерывания**)
- выполнение процессором команды INT вызова процедуры обработки прерывания (**программные прерывания**)

2.9.1. Внутренние прерывания

Для обслуживания внутренних прерываний микропроцессоры i8086 использовали 5 первых векторов прерываний (00h-04h). Прерывания с номерами 05h-31h фирма Intel зарезервировала для использования в дальнейших разработках, однако фирма IBM при создании IBM PC использовала эти вектора по своему усмотрению. В последующих реализациях процессоров эти же вектора были задействованы для обработки новых внутренних прерываний, в результате чего стали возможны конфликты. Поскольку все добавленные прерывания используются в защищенном режиме процессора, то для избежания конфликтов при переходе в защищенный режим контроллер прерываний перепрограммируется таким образом, что при поступлении сигнала на линию IRQ0 вызывается процедура обработки с номером, отличным от 08h (обычно 50h или 60h). Для последующих линий IRQ вызываются процедуры с последующими (относительно базового для IRQ0) номерами.

Сигналы внутренних прерываний возникают при нарушениях в работе самого микропроцессора либо при ошибках в выполняемых командах и формируются внутри схемы микропроцессора при возникновении одной из ситуаций, указанных в Табл. 2.2.

2.9.2. Аппаратные прерывания

Аппаратные прерывания инициируются различными устройствами компьютера, внешними по отношению к процессору (системный таймер, клавиатура, контроллер

диска и пр.). Эти устройства формируют сигналы запросов на прерывания (IRQ), поступающие на **контроллер прерываний**.

2.9.2.1. Программируемый контроллер прерываний (ПКП)

Программируемый контроллер прерываний *(Programmable Interrupt Controller, PIC),* реализованный на микросхеме i8259, предназначен для поддержки аппаратных прерываний от восьми различных устройств. Каждое устройство имеет собственный приоритет при обслуживании заявок на прерывания. Кроме того, сигнал запроса прерывания может быть **замаскирован**, т.е. при появлении этого сигнала контроллер прерываний его игнорирует. Для увеличения количества обслуживаемых внешних устройств, микросхемы контроллеров можно каскадировать, подключая выходы дополнительных микросхем ко входам основной.

Контроллер прерываний можно запрограммировать для работы в нескольких режимах:

1. Режим фиксированных приоритетов *(Fixed Priority, Fully Nested Mode).* В этом режиме запросы прерываний имеют жесткие приоритеты от 0 (высший) до 7 (низший) и обрабатываются в соответствии с этими приоритетами. Если запрос с меньшим приоритетом возникает во время обработки запроса с более высоким приоритетом, то он игнорируется. BIOS при включении питания и при перезагрузке программирует контроллер прерываний на работу именно в этом режиме. В дальнейшем, режим при необходимости, можно изменить программным способом.

Табл. 2.2. Внутренние прерывания процессоров i80x86

Исключительная ситуация	Номер прерывания
Деление на 0	00h
Завершение выполнения очередной команды процессора при установленном флаге TF (используется в режиме отладки программ)	01h
Особая ситуация в режиме отладки (80386+)	01h
Выполнение команды INTO при установленном флаге OF (используется при обнаружении переполнения)	04h
Выход проверяемой величины за пределы диапазона при выполнении команды BOUND (80186+)	05h
Попытка выполнения недопустимой операции (например	06h

Исключительная ситуация	Номер прерывания
привилегированной команды в реальном режиме) (80286+)	
Попытка выполнения команды сопроцессора при отсутствии последнего (80286+)	07h
Возникновение двух исключительных ситуаций в одной команде (80286+)	08h
Попытка сопроцессора получить доступ к памяти за границами сегмента (80286,80386)	09h
Попытка переключиться на TSS, содержащий неверную информацию (80286+)	0Ah
Обращение к сегменту, выгруженному на диск из оперативной памяти (80286+)	0Bh
Переполнение стека (80286+)	0Ch
Отказ общей защиты (80286+)	0Dh
Ошибка обращения к странице виртуальной памяти (80386+)	0Eh
Обращение по неправильно выровненному адресу памяти (486+)	11h

1. Режим с автоматическим сдвигом приоритетов *(Automatic Rotation)*. В этом режиме после обработки прерывания данному уровню присваивается минимальный приоритет, а все остальные приоритеты изменяются циклически таким образом, что запросы следующего за только что обработанным уровнем имеют наивысший, а предыдущего — наинизший приоритет. Например, после обработки запроса уровня 6 приоритеты устанавливаются следующим образом: уровень 7 имеет приоритет 0, уровень 8 — приоритет 1 и т.д., уровень 5 — приоритет 6 и уровень 6 — приоритет 7. Таким образом, в этом режиме всем запросам дается возможность получить обработку.
2. Режим с программно-управляемым сдвигом приоритетов *(Specific Rotation)*. Приоритеты можно изменять так же, как и в режиме автоматического сдвига, однако для этого необходимо подать специальную команду, в которой указывается номер уровня, которому нужно присвоить максимальный приоритет.
3. Режим автоматического завершения обработки прерывания *(Automatic End of Interrupt)*. В этом режиме, в отличие от остальных, обработчик прерывания *не должен* при своем завершении посылать контроллеру специальный **сигнал завершения обработки аппаратного прерывания** *(End Of Interrupt, EOI),* который разрешает обработку прерываний с текущим и более низкими приоритетами. В данном режиме эти прерывания разрешаются в момент начала

обработки прерывания. Режим используется редко, т.к. все обработчики прерываний должны быть **повторно входимыми (реентерабельными)** в случае, если до завершения обработки возникнет запрос на прерывание от того же источника.

4. Режим специальной маски *(Special Mask Mode)*. В данном режиме можно замаскировать отдельные уровни прерываний, изменив приоритетное упорядочение обработки запросов. После отмены этого режима восстанавливается прежний порядок обработки прерываний.
5. Режим опроса *(Polling Mode)*. В этом режиме обработка прерываний не производится автоматически, сигналы запроса прерываний лишь фиксируются во внутренних регистрах контроллера. Процедуру обработки прерывания необходимо вызывать "вручную", в соответствии с хранящимся в контроллере номером уровня с максимальным приоритетом, по которому имеется запрос.

Кроме этого, имеется возможность программировать номер процедуры прерывания, которая будет вызываться при возникновении сигнала на первой линии IRQ (при этом номера процедур для остальных IRQ будут иметь последующие значения). Эта возможность используется при переходе в защищенный режим процессора, т.к. в этом режиме прерывания с номерами 05h-11h используются для обработки критических ситуаций (логические прерывания). Чтобы исключить возможные конфликты, аппаратные прерывания обрабатываются процедурами с большими номерами, например, OS/2 v1.x перемещает обработчики прерываний от устройств, подключенных к IRQ0-IRQ7, на процедуры с номерами 50h-57h. Кроме этого, указанную процедуру применяют в тех случаях, когда необходимо контролировать выполнение обработчиков прерываний, т.к. перепрограммирование контроллера усложняет перехват прерываний.

Приведем пример такой программы:

```
IDEAL
MODEL TINY
CODESEG
      StartupCode
      JMP INIT          ; переход к инициализации резидента
IRQ0:                   ; подпрограмма обработки прерывания таймера
      STI               ; разрешение прерываний
      INT 08h  ; вызов оригинального обработчика
       IRET             ; возврат из обработчика
IRQ1:                   ; подпрограмма обработки прерывания клавиа;туры
       STI
       INT 09h
       IRET
IRQ2:           ; подпрограмма обработки запроса по IRQ2
       STI
```

```
        INT 0Ah
        IRET
IRQ3:           ; обработка прерывания от COM2,COM4
        STI
        INT 0Bh
        IRET
IRQ4:           ; обработка прерывания от COM1,COM3
        STI
        INT 0Ch
        IRET
IRQ5:           ; подпрограмма обработки запроса по IRQ5
        STI
        INT 0Dh
        IRET
IRQ6:           ; подпрограмма обработки запроса по IRQ6
        STI
        INT 0Eh
        IRET
IRQ7:           ; подпрограмма обработки запроса по IRQ7
        STI
        INT 0Fh
        IRET
        INIT:
        MOV AX,2560h   ;занесение новых значений в таблицу векторов
                                ;    25 — номер функции
                                ;    60 — номер прерывания
        MOV BX,OFFSET IRQ       ; адрес начала таблицы адресов новых
                                    ;   обработчиков прерываний
A1:     MOV DX,[BX]     ; значение очередного элемента таблицы
                        ;  адресов новых обработчиков прерываний
        INT 21h         ; вызов функции MS-DOS
        ADD BX,2                ; изменение индекса в таблице
        INC AL          ; увеличение номера прерывания
        CMP AL,67h
        JNG A1
        CLI                     ; запрет прерываний
        MOV AL,0FFh  ; процедура перепрограммирования контроллера
        OUT 21h,AL
        MOV AL,11h
        OUT 20h,AL
                MOV AL,60h  ; номер первой процедуры обработки прерываний
        OUT 21h,AL
        MOV AL,4
```

```
        OUT 21h,AL
        MOV AL,1
        OUT 21h,AL
        MOV AL,0
        OUT 21h,AL
        OUT 20h,AL
        STI                             ; разрешение прерываний
        MOV AX,3100h            ; завершение резидентной программы
        MOV DX,OFFSET INIT      ; вычисление размера
                                ; резидентной части
        MOV CL,4                        ;  в параграфах
        SHR DX,CL
        INC DX
        INT 21h                 ; конец исполняемого кода
        ; таблица адресов новых обработчиков прерываний
IRQ   DW IRQ0,IRQ1,IRQ2,IRQ3,IRQ4,IRQ5,IRQ6,IRQ7
ENDS
END @StartUp
```

Данная программа программирует контроллер таким образом, что появление сигналов на линиях IRQ0-IRQ7 вызывает соответственно процедуры INT60h-INT67h. Новые обработчики прерываний в данном примере не выполняют никаких полезных функций — они просто вызывают старые процедуры обработки, однако их нетрудно наполнить нужным содержанием.

Так как после выполнения этой процедуры вместо прерываний INT08h-INT0Fh будут вызываться процедуры INT60h-INT67h, предварительно необходимо занести в таблицу векторов адреса новых обработчиков прерываний, которые уже должны находиться в памяти. Таблицу векторов можно изменить, непосредственно записав нужные значения в память по адресам, начиная с 0000:0180 (60h*4=180h), или используя функцию 25h прерывания 21h, предоставляемую MS-DOS, как и сделано в приведенном примере. При вызове прерывания необходимо указать следующие параметры:

- в регистре AH — номер функции
- в регистре AL — номер изменяемого вектора
- в регистрах DS:DX — адрес нового обработчика

Программирование контроллера заключается в посылке определенных кодов в управляющие регистры ПКП (с номерами 20h и 21h). Среди этих кодов указывается и номер вектора, который контроллер в дальнейшем будет выставлять на шину при требовании прерывания процессора.

Программа состоит из двух частей: резидентной и инициализирующей. При запуске программы выполняется код начиная от метки INIT, за которой в цикле в таблицу векторов заносятся новые значения при помощи функции MS-DOS. Далее следует процедура программирования контроллера, которая обрамлена командами

CLI и STI для того, чтобы исключить обработку прерываний до завершения перепрограммирования.

Программа завершается при помощи функции 31h прерывания 21h, при этом часть программы, расположенная перед меткой INIT, остается в памяти после завершения программы. Резидентная часть содержит новые обработчики прерываний, каждый из них начинается меткой IRQx (соответственно обрабатываемому запросу). Адреса обработчиков помещаются в таблицу, находящуюся в конце программы, для удобства их использования при занесении в таблицу векторов.

Функция 31h требует указания в регистре DX размера резидентной части в параграфах. Для этого размер резидента в байтах (определяемый выражением OFFSET INIT) делится на 16 при помощи сдвига вправо на 4 разряда и округляется в большую сторону.

Данная программа предполагает, что вектора 60h-67h не используются другими программами, в противном случае возникнет конфликт между программами, совместно использующими эти вектора прерываний, и прерывания будут обрабатываться некорректно.

Схема взаимодействия внешних устройств, ПКП и центрального процессора показана на рис.2.21. В компьютерах класса IBM PC и PC/XT к контроллеру подключено 8 устройств, для обработки прерываний от которых используются подпрограммы, на которые указывают вектора 08h..0Fh (табл.2.3).

В связи с увеличением количества внешних устройств, требующих поддержки аппаратными прерываниями, компьютеры IBM PC/AT и последующие модели оснащаются двумя контроллерами прерываний. Дополнительный контроллер, называемый **ведомым**, подключается ко входу IRQ2 основного контроллера, который называют **ведущим** (рис.2.22.). На входы ведомого ПКП подключены линии IRQ8-IRQ15, причем линия IRQ8 имеет максимальный приоритет в ведомом контроллере, а IRQ15 — минимальный. В результате такого каскадирования запросы от устройств, подключенных к ведомому контроллеру, обрабатываются по двухуровневой системе приоритетов: младший уровень — внутри ведомого ПКП, а старший — внутри ведущего. Поскольку выход ведомого контроллера подключен к линии IRQ2, приоритеты линий IRQ8-IRQ15 расположены между приоритетами IRQ1-IRQ3. Устройства, имеющие возможность обслуживания аппаратными прерываниями в компьютерах IBM PC/AT, перечислены в таблице 2.4.

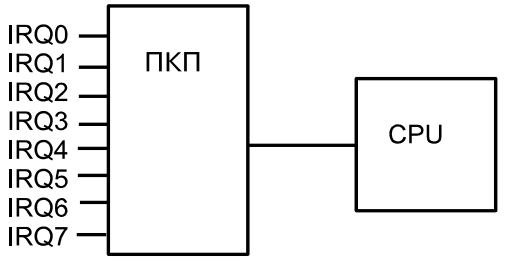

Рис. 2.21 Общая схема соединения источников прерываний, ПКП и микропроцессора в IBM PC и IBM PC/XT

Табл. 2.3. Аппаратные прерывания IBM PC/XT

Линия запроса	Вектор прерывания	Подключенное устройство
IRQ0	08h	таймер
IRQ1	09h	клавиатура
IRQ2	0Ah	резерв
IRQ3	0Bh	COM2,COM4
IRQ4	0Ch	COM1,COM3
IRQ5	0Dh	жесткий диск
IRQ6	0Eh	гибкий диск
IRQ7	0Fh	LPT1

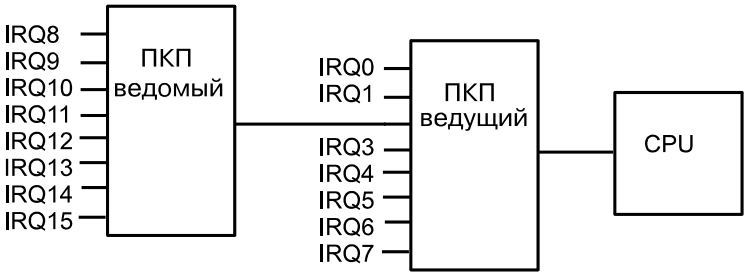

Рис. 2.22. Общая схема соединения источников прерываний, ПКП и микропроцессора в IBM PC/AT и последующих моделях

Рассмотрим внутреннюю структуру контроллера. В нем можно выделить следующие основные узлы (рис. 2.23):

- регистр запросов на прерывания *(Interrupt Requests Register, IRR)*. На входы регистра поступают сигналы с линий IRQ.
- регистр маскирования прерываний *(Interrupt Mask Register, IMR)*. Единица в разряде регистра блокирует прохождение сигнала по соответствующей линии запроса.
- регистр обрабатываемых запросов *(In-Service Register, ISR)*. Единица в разряде регистра указывает на то, что обрабатывается прерывание данного уровня.
- схему обработки приоритетов *(Priority Resolver)*. Определяет запрос с максимальным приоритетом.

Табл. 2.4. Аппаратные прерывания IBM PC AT

Линия запроса	Вектор прерывания	Подключенное устройство
IRQ0	08h	таймер
IRQ1	09h	клавиатура
IRQ2	0Ah	ведомый ПКП
IRQ8	70h	микросхема КМОП
IRQ9	71h	перенаправлено на int 09h
IRQ10	72h	резерв
IRQ11	73h	резерв
IRQ12	74h	резерв (мышь в PS/2)
IRQ13	75h	сопроцессор
IRQ14	76h	жесткий диск
IRQ15	77h	резерв
IRQ3	0Bh	COM2,COM4
IRQ4	0Ch	COM1,COM3
IRQ5	0Dh	LPT2
IRQ6	0Eh	гибкий диск
IRQ7	0Fh	LPT1

Сигнал от внешнего устройства по линии IRQ поступает на регистр запросов на прерывание, устанавливая в единицу соответствующий разряд *(1)*. Далее, если в

регистре маскирования прерываний разряд с тем же номером не установлен в "1", сигнал проходит в схему анализа приоритетов *(2)*. Если в данный момент не выполняется прерывание с более высоким приоритетом, сигнал запроса прерывания поступает на вход регистра обслуживаемых запросов, где разрешает установку соответствующего бита в "1" (но не устанавливает его), и одновременно по линии INT в микропроцессор *(3)*.

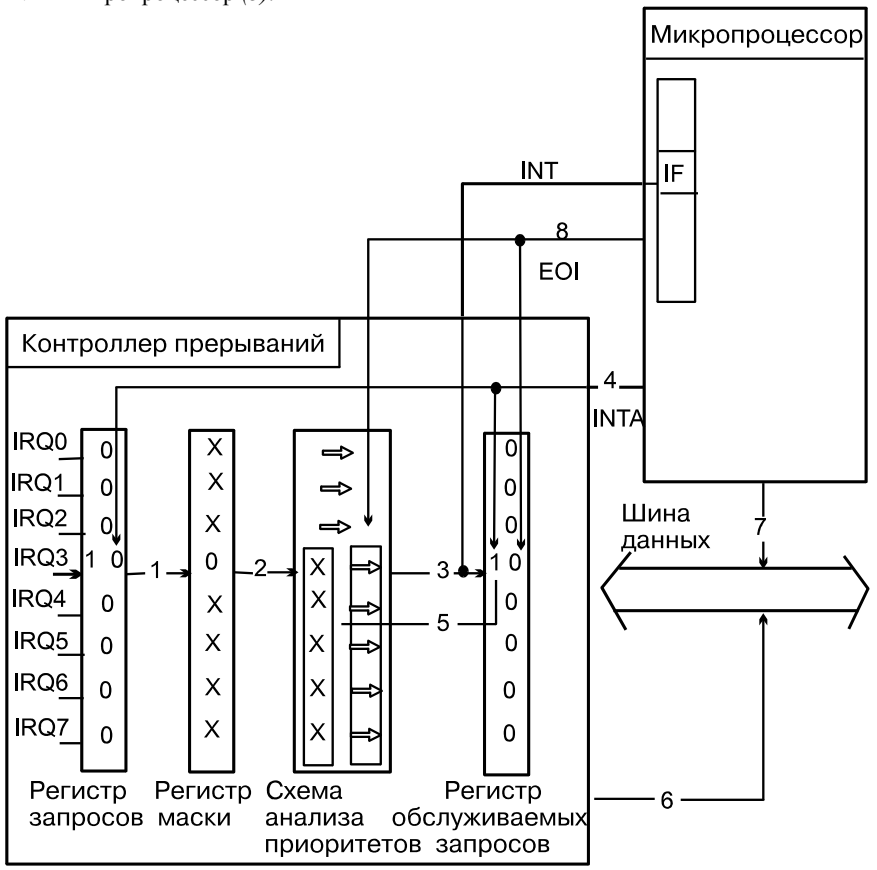

Рис. 2.23. Схема взаимодействия ПКП и микропроцессора

Микропроцессор реагирует на поступление сигнала INT только в том случае, если в регистре флагов установлен флаг IF разрешения прерываний (таким образом, сбросив флаг IF командой CLI, можно запретить все аппаратные прерывания, регистрируемые контроллером прерываний). Получив сигнал INT, микропроцессор

завершает обработку текущей команды и переходит к обработке прерывания, начинающееся с посылки контроллеру прерываний сигнала INTA *(INTerrupt Acknoledge*, подтверждение прерывания), который устанавливает разрешенный ранее бит регистра обслуживаемых запросов и сбрасывает бит регистра запросов *(4)*.

С этого момента прерывание переводится в разряд обслуживаемых, а на регистре запросов может регистрироваться и ждать обслуживания новый сигнал запроса прерывания, возможно, от того же устройства. После установки бита в регистре обслуживаемых прерываний, в схеме анализа приоритетов блокируются все прерывания, начиная с текущего уровня и ниже по цепочке приоритетов *(5)*. После получения второго сигнала подтверждения от процессора по линии *INTA,* ПКП выставляет на шину 8-битовый номер вектора прерывания *(6),* которое получило обслуживание (BIOS программирует ведущий контроллер на номера 0-7, а ведомый — на номера 8-15). Далее процессор запоминает в стеке прерываемой программы значения регистра флагов, счетчика команд (IP) и регистра адреса кодового сегмента (CS), затем сбрасывает флаг IF. В последние два регистра помещаются значения, хранящиеся в векторе прерывания, номер которого процессор читает на шине данных *(7)*. После этого начинается выполнение обработчика прерывания.

Для обеспечения корректной работы системы, обработчик прерывания должен сохранить содержимое всех регистров процессора с тем, чтобы при выходе из обработчика прерванная программа могла продолжить свое функционирование. Обычно процедура обработки прерывания сохраняет все изменяемые регистры в стеке прерываемой программы до первого их изменения, а затем восстанавливает их перед выходом. Из этого следует, что стек любой программы, которая может быть прервана, должен содержать достаточно свободного места для сохранения в нем адресов возврата и регистров процессора несколькими прерываниями, т.к. прерывания могут быть вложенными и очень редко создают свой стек, поэтому при вложенных прерываниях все обработчики могут сохранять данные в одном стеке программы.

Исходя из сказанного, можно считать, что значения регистров CS, IP и Flags, помещаемые в стек при инициализации прерывания, а также остальных регистров процессора, сохраняемых обработчиком, составляют PSW процессора i80x86 в реальном режиме. При этом старое PSW запоминается в стеке прерываемой программы, а новое PSW находится в векторе вызываемого прерывания и имеет сокращенный вид (в реальном режиме система выполняет только одну программу, прерывающим процессом может быть только обработчик прерывания, а он всегда выполняется с начала, поэтому новое PSW не содержит инициализирующих значений регистров). Новое PSW замещает старое в регистрах процессора при переключении контекста.

Обработка прерывания завершается после выполнения процессором **команды** *IRET* **возврата из обработчика прерывания**. Процессор восстанавливает значения сохраненных в стеке регистров, таким образом устанавливается флаг IF и

прерывания вновь разрешаются. Однако, многие обработчики прерываний содержат команду STI разрешения прерываний в своем теле, разрешая таким образом прерывания с более высоким приоритетом. При этом необходимо быть уверенным в том, что при возникновении вложенных прерываний не возникнут конфликты. Обычно это условие выполняется, и прерывания с высшим приоритетом (которые имеют большую важность для нормального функционирования системы) получают возможность быть обработанными.

Что касается прерываний данного и низшего уровней приоритетов, то они остаются заблокированными схемой анализа приоритетов, даже после завершения обработки прерывания. Для разрешения этих прерываний существует специальная **команда завершения прерывания** *(End Of Interrupt, EOI),* которая заключается в посылке определенного кода на управляющий регистр контроллера прерываний (8). Эта команда сбрасывает бит в регистре обслуживаемых прерываний и разрешает прохождение сигналов запроса прерываний через схему анализа приоритетов. Обычно такая команда завершает обработчик прерывания, однако она может располагаться в любом месте внутри обработчика. В последнем случае возможно вложенное прерывание того же уровня, что и обрабатываемое, т.е. будет выполняться та же самая процедура обработки, что обрабатывается в текущий момент. Такая процедура должна удовлетворять определенным требованиям: она должна быть повторно входимой (реентерабельной). Забота об этом ложится на программиста. Следует также помнить, что если команду STI в обработчике задавать не обязательно (т.к. при возврате из прерывания команда IRET восстанавливает из стека значение регистра флагов, сохраненное в момент вызова прерывания, т.е. с установленным флагом IF), то команду завершения прерывания (EOI) нужно подать обязательно, иначе будут запрещены прерывания на схеме анализа приоритетов, а текущее прерывание останется в стадии выполнения.

2.9.2.2. Немаскируемые прерывания

Существует одно аппаратное прерывание, отличное от всех остальных. Это **немаскируемое прерывание** *(Non-Maskable Interrupt, NMI).* Его отличие состоит в том, что хоть инициатором его и является внешнее устройство, сигнал запроса поступает в процессор, минуя контроллер прерываний. Поэтому такой запрос на прерывание невозможно замаскировать. Кроме этого, запрос по линии NMI вызывает немедленную реакцию процессора и имеет максимальный приоритет в системе, хотя обработчик NMI может быть прерван любым маскируемым прерыванием, если их разрешить командой STI. Это позволяет использовать данное прерывание в особо критических ситуациях: обычно оно используется при падении напряжения питания и при ошибках памяти, однако на некоторых моделях IBM — совместимых компьютеров NMI используется также для обслуживания сопроцессора, клавиатуры, каналов ввода/вывода, дисковых контроллеров, часов реального времени, таймеров, контроллеров прямого доступа к памяти и пр.

В процессорах i80286 и выше в случае, если во время обработки NMI возникнет повторный запрос на немаскируемое прерывание, он не прервет выполнение текущего обработчика, а запомнится и будет обработан после завершения текущего обработчика. Если повторных запросов во время выполнения обработчика будет несколько, то сохранится только первый, последующие будут проигнорированы. При возврате из обработчика командой IRET процессор не переходит на выполнение следующей команды, а пытается выполнить команду по тому же адресу, что привел к возникновению критической ситуации.

2.9.2.3. Программные прерывания

Программные прерывания инициируются специальной командой процессору INT n, где n указывает номер прерывания, обработчик которого необходимо вызвать. При этом в стек заносится содержимое регистров флагов, указателя команд и сегмента кода, а флаг разрешения прерывания сбрасывается (аналогично тому, как при обработке аппаратных прерываний).

Далее в процессор загружается новое PSW, значение которого содержится в векторе прерывания с номером n, и управление переходит к обработчику прерывания. Требования к обработчику программного прерывания такие же, как и к обработчику аппаратного прерывания, за исключением того, что команда завершения аппаратного прерывания (EOI) не требуется. Обработка прерывания завершается командой IRET, восстанавливающей из стека старое PSW.

Обработка программного прерывания является более мягким видом прерывания по отношению к прерываемой программе, нежели обработка аппаратного прерывания, т.к. аппаратное прерывание выполняется по требованию самой прерываемой программы, и она ожидает от этого какого-либо результата. Аппаратное же прерывание получает управление безо всякого ведома выполняющейся программы и зачастую не оказывает на нее никакого влияния.

2.9.2.4. Поддержка прерываний в BIOS

BIOS берет на себя основную работу по программированию периферийных устройств компьютера, в том числе и контроллера прерываний. Кроме этого BIOS устанавливает обработчики всех аппаратных прерываний, а также предоставляет множество других функций по управлению практически всеми устройствами компьютера при помощи программных прерываний. Адреса этих обработчиков BIOS заносит в таблицу векторов прерываний.

Все служебные функции BIOS реализованы в обработчиках программных прерываний и могут быть вызваны при помощи команды прерывания *INT* с указанием номера прерывания. Обычно одно прерывание содержит в себе не одну, а несколько функций, выполняющих сходные действия или работающих с одним устройством. Такой подход позволяет реализовать большое количество функций, используя гораздо меньшее количество прерываний, что очень важно, так как количество обслуживаемых системой прерываний ограничено. Список программных прерываний, предоставляемых BIOS, приведен в таблице 2.5.

2.9.2.5. Прерывания в MS-DOS

Система MS-DOS активно использует механизм прерываний. Все функции, предоставляемые ОС, как и функции BIOS, реализованы именно в виде программ обработки прерываний. Если некоторому процессу требуется сервис операционной системы, он должен вызвать программное прерывание (аналог обращения к супервизору), передав через регистры общего назначения микропроцессора необходимые параметры DOS и функции DOS, вызываемой процедуры.

Следуя терминологии справочного руководства MS-DOS, служебные процедуры операционной системы разделены на две категории: прерывания DOS имеют уникальные номера программных прерываний, а для вызова функции необходимо вызвать прерывание и конкретизировать номер функции, задав его в одном из регистров (обычно AX).

MS-DOS 3.00 предоставляет прерывания 20h-28h и 2Fh, пять из них (20h, 25h-27h, 2Fh) являются "настоящими" (содержащими одну процедуру обработчика) прерываниями DOS; все функции реализованы в обработчике прерывания 21h; три прерывания (22h-24h) содержат адреса процедур обработки исключительных ситуаций, которые могут настраиваться каждой программой. Существуют также недокументированные прерывания и служебные функции, частично документированные в последующих версиях MS-DOS. Новые версии MS-DOS привнесли также и новые прерывания (табл.2.6.):

Табл. 2.5. Служебные прерывания BIOS

Номер прерывания	Назначение
05h	Печать экрана
10h	Работа с видеоадаптером
11h	Получение списка подключенного оборудования
12h	Получение размера памяти
13h	Функции работы с дисками
14h	Работа с COM-портом
15h	Работа с кассетным магнитофоном и дополнительные функции BIOS
16h	Доступ к клавиатуре и буферу клавиатуры

17h	Функции работы с печатающим устройством
18h	Вызов недискового загрузчика (напр. ROM-BASIC в PC фирмы IBM)
19h	Вызов программы начальной загрузки
1Ah	Работа с часами реального времени
1Bh	Обработка нажатия Ctrl-Break
1Ch	Вызывается обработчиком прерывания таймера (IRQ0, INT 08h)
1Dh	Доступ к таблице параметров видеосистемы
1Eh	Доступ к таблице параметров дисков
1Fh	Доступ к области описания шрифта 8x8
4Ah	Обработчик "будильника" часов реального времени
FEh,FFh	Используются при возвращении в реальный режим i80286

Изменение векторов прерываний довольно часто используется при написании программ, резидентно остающихся в памяти. Однако, при написании таких программ следует уделять особое внимание реентерабельности, ведь, получая управление, резидентный процесс прерывает другой процесс, которым может быть и системный обработчик прерывания. Но прерывания MS-DOS не являются реентерабельными (реентерабельность прерываний BIOS умалчивается, то есть обычно проблем с функциями BIOS не возникает, но все же гарантии на этот счет нет), поэтому использовать функции MS-DOS можно только убедившись, что прерванный процесс не был системным. Для этого можно использовать функцию 34h прерывания 21h, которая документирована начиная с версии MS-DOS 5.0, но существует и в более ранних версиях. Функция возвращает в ES:BX адрес флага InDOS, сигнализирующего о выполнении системного процесса. Другой весьма полезной недокументированной функцией MS-DOS является функция 52h прерывания 21h, возвращающая в ES:BX адрес "списка списков" *(List of lists)* иначе называемого блоком переменных DOS *(DOS vars block),* в котором содержатся адреса управляющих таблиц MS-DOS и указатели на системные драйвера.

Табл. 2.6. Прерывания MS-DOS

Номер прерывания	Назначение прерывания
20h	Завершить программу
21h	Служебные функции MS-DOS
22h	Адрес процедуры, получающей управление после завершения программы
23h	Обработчик нажатия Ctrl-Break
24h	Обработчик критической ошибки (аппаратуры)
25h	Абсолютное чтение диска
26h	Абсолютная запись на диск

27h	Резидентное завершение программы
28h	Вызывается при ожидании ввода с клавиатуры
29h	Вывод символа на экран
2Bh—2Dh	Зарезервированы
2Fh	Мультиплексное прерывание
30h	Межсегментный переход на процедуру поддержки CP/M

Операционная система позволяет создавать новые, изменять, дополнять и полностью замещать любые, в том числе и собственные, обработчики прерываний. Это позволяет весьма гибко настроить систему для использования с различными устройствами и для оптимального выполнения определенных заданий. Хотя MS-DOS и предоставляет средства для работы с векторами и обработчиками прерываний, но и вектора прерываний, и процедуры обработки прерываний можно изменить, минуя операционную систему. Это способствует распространению в такой среде, как MS-DOS, компьютерных вирусов, а также создает проблемы при совместном использовании одного ресурса несколькими процессами.

2.9.2.6. Использование собственных обработчиков прерываний

Хотя BIOS и операционная система предоставляют весьма широкий спектр служебных процедур, нередко при использовании нестандартных устройств либо для расширения функциональности стандартных устройств или системных функций возникает необходимость в написании собственных обработчиков прерываний. Новый обработчик может либо полностью заменить системный, либо дополнить его. Второй вариант применяется чаще, т.к. он проще реализуется.

Существуют 3 основных способа дополнения имеющегося обработчика прерывания:

1. Выполнить дополнительные действия до вызова имеющегося обработчика
2. Выполнить дополнительные действия после вызова имеющегося обработчика
3. Выполнить дополнительные действия до и после вызова имеющегося обработчика

Для изменения (дополнения) процедуры обработки прерывания необходимо изменить адрес этого обработчика в таблице векторов прерываний. Это можно сделать непосредственно записав адрес нового кода в таблицу или при помощи функции MS-DOS. Если новый обработчик будет лишь дополнять старый, то предварительно необходимо сохранить адрес старого обработчика для того, чтобы иметь возможность его вызова. Получить этот адрес также можно двумя путями: прочитав его значение в таблице векторов либо вызвав функцию MS-DOS.

Предположим, мы хотим добавить свой код к обработке прерывания клавиатуры. Тогда изменение вектора прерывания может выглядеть следующим образом:

```
OldInt9h DD ? ; адрес старого обработчика прерывания
```

```
PROC NewInt9h          ; новый обработчик
        . . . . . .
ENDP
        . . . . . .
        MOV AX,0  ; занесение в ES сегмента таблицы векторов (0000)
        MOV ES,AX
        CLI
        MOV AX,[ES:9*4]              ; получение и сохранение адреса
        MOV [WORD PTR OldInt9h],AX   ;   старого обработчика
        MOV AX,[ES:9*4+2]
        MOV [WORD PTR OldInt9h+2],AX
        MOV AX,OFFSET NewInt9h       ; занесение нового адреса
        MOV [WORD PTR ES:9*4],AX
        MOV AX,SEG NewInt9h
        MOV [WORD PTR ES:9*4+2],AX
        STI
. . . . . .
```

В данном примере функции MS-DOS не используются, поэтому фрагмент кода по обращению к таблице векторов обрамлен командами CLI и STI для того, чтобы исключить возможность изменения таблицы векторов другим процессом в это же время.

Того же результата можно достичь, используя функции MS-DOS (INT 21h): 35h для получения и 25h для изменения вектора прерывания. Обе функции требуют задания в регистре AL номера вектора, функция 35h возвращает значение этого вектора (сегмент:смещение) в регистрах ES:BX, а 25h записывает в указанный вектор значение регистров DS:DX. Функции DOS гарантируют корректное использование таблицы векторов, поэтому команды CLI и STI использовать не нужно:

```
        . . . . . .
OldInt9h DD ? ; адрес старого обработчика прерывания
PROC NewInt9h          ; новый обработчик
        . . . . . .
ENDP
        . . . . . .
        MOV AX,3509h             ; получение и сохранение адреса
        INT 21h          ;   старого обработчика
        MOV [WORD PTR OldInt9h],BX
        MOV [WORD PTR OldInt9h+2],ES
        MOV AX,2509h                     ; занесение нового адреса
        MOV AX,SEG NewInt9h
        MOV DS,AX
        MOV DX,OFFSET NewInt9h
```

```
        INT 21h
        . . . . . .
```

Если дополнение к обработчику вызывается после системной процедуры, необходимо передать ей управление таким образом, чтобы после выполнения команды IRET системного обработчика управление перешло к добавленному нами коду. Для этого переход на старый обработчик выполняется командой CALL с атрибутом FAR. Хотя эта команда имеется во всех процессорах i80x86, она не реализована во многих Ассемблерах, поэтому ее необходимо задавать непосредственно ее машинным кодом: 9Ah <длинный адрес вызова> . Так как команда CALL сохраняет в стеке только регистры IP и CS, а IRET извлекает из стека еще и Flags, то перед вызовом CALL необходимо поместить в стек регистр флагов командой PUSHF.

С учетом сказанного, обработчик прерывания может иметь следующий вид:

```
PROC NewInt9h   ; новый обработчик
        PUSHF
        DB 9Ah  ; CALL FAR
OldInt9h DD ?   ; адрес старого обработчика прерывания
        . . . . . .                     ; добавленный код
        IRET
ENDP
```

Если же новый код добавляется перед вызовом системного обработчика, можно использовать команду JMP FAR, которую также нужно задавать ее машинным кодом: 0EAh <длинный адрес перехода>. При этом, после выполнения старого обработчика, команда IRET выполнит возврат в прерванную программу, а не в добавленный обработчик.

Рассмотрим пример расширения процедуры обработки прерывания клавиатуры с целью выполнения специальных действий при нажатии на клавишу ScrollLock:

```
IDEAL
MODEL TINY
CODESEG
        STARTUPCODE
        JUMP Init
PROC NewInt9h           ; новый обработчик
        PUSHF                   ; сохранение изменяемых регистров
        PUSH AX
        IN AL,60h               ; чтение порта клавиатуры
        CMP AL,70
        JNE _Jump
        . . . . . .     ; действия при нажатии ScrollLock
_Jump:
        POP AX          ; восстановление измененных регистров
```

```
        POPF
        DB 0EAh                 ; JMP FAR
OldInt9h DD ?           ; адрес старого обработчика прерывания
ENDP
Init:
        MOV AX,3509h                    ; получение и сохранение адреса
        INT 21h                         ;   старого обработчика
        MOV [WORD PTR OldInt9h],BX
        MOV [WORD PTR OldInt9h+2],ES
        MOV AX,2509h                    ; занесение нового адреса
        MOV DX,OFFSET NewInt9h
        INT 21h
        MOV AX,3100h            ; завершение резидентной программы
        MOV DX,OFFSET Init      ; вычисление размера резидентной
                        ; части
        MOV CL,4                ;   в параграфах
        SHR DX,CL
        INC DX
        INT 21h
ENDS
END @StartUp
```

Новый обработчик прерывания читает порт 60h, в котором при вызове INT 9 находится скан-код нажатой клавиши, и сравнивает его с кодом ScrollLock (70). Если была нажата клавиша ScrollLock, выполняются соответствующие действия, после чего командой JMP FAR осуществляется переход в системный обработчик, который и завершает обработку прерывания, поэтому в новой процедуре не используется команда IRET.

Необходимо помнить, что обработчик прерывания должен сохранять все изменяемые им регистры процессора и восстанавливать их при выходе, кроме тех случаев, когда через регистры передается результат обработки прерывания.

В случае, если новый обработчик полностью заменяет прежний, необходимо помнить, что при входе в процедуру обработки запрещаются все прерывания, поэтому по возможности их нужно разрешить командой STI. Кроме того, если новый обработчик обслуживает аппаратное прерывание, необходимо перед его завершением послать контроллеру прерываний команду EOI. Это делается следующим образом:

- при обработке запросов ведущего ПКП (IRQ0-IRQ7) — командами
  ```
  MOV AL,20h
  OUT 20h,AL
  ```
- при обработке запросов ведомого ПКП (IRQ8-IRQ15) — командами
  ```
  MOV AL,20h
  OUT 0A0h,AL
  ```

2.9.2.7. Реализация выполнения операций ввода/вывода в MS-DOS

- *Ввод/вывод через COM — порт*

Рассмотрим взаимодействие пользовательской программы, операционной системы и аппаратуры на примере выполнения ввода из внешнего устройства, подключенного к последовательному интерфейсу. В IBM PC такой интерфейс соответствует стандарту RS-232-C. Обычно в компьютере установлено 2 или 4 порта, обозначаемых COM1-COM4, однако порты могут формировать запросы на прерывания только по двум линиям: IRQ3 (COM2 и COM4) и IRQ4 (COM1 и COM3). Поэтому нельзя одновременно работать с портами COM1 и COM3 или COM2 и COM4, используя аппаратные прерывания (можно работать с одним портом в режиме программного опроса, а с другим — либо через прерывания, либо в режиме опроса).

Интерфейс RS-232-C реализован в виде семи регистров, базовый адрес (адрес первого регистра) для COM1 обычно 3F8h (адреса портов можно узнать в таблице параметров BIOS по адресу 0040:0000h):

Первый регистр (3F8h) используется для чтения при приеме и для записи при передаче данных, а также для установки скорости обмена COM-порта с подключенным к нему устройством.

Второй регистр (3F9h) управляет режимом выдачи сигнала запроса прерывания. Возможны следующие режимы:

1. прерывание возможно при готовности принимаемых данных
2. прерывание возможно после передачи последнего бита из буфера при передаче
3. прерывание возможно при обнаружении ошибки и при получении сигнала BREAK
4. прерывание возможно при изменении состояния входных линий COM-порта

Если произошло прерывание от COM-порта, то его причину можно узнать, прочитав третий регистр (по адресу 3FAh).

Четвертый регистр (3FBh) является управляющим: записанная в нем информация определяет длину передаваемых слов, количество стоп-бит, тип контроля.

Остальные регистры используются для управления линией и подключенным к COM-порту устройством.

MS-DOS предоставляет 2 функции для работы с COM-портом:

1. чтение символа из COM-порта — функция 03 прерывания 21h
2. запись символа в COM-порт — функция 04 прерывания 21h

Рассмотрим рисунок 2.24. Программа пользователя, желающая ввести символ из COM-порта, должна загрузить в регистр AH номер функции — 03 и подать *(1)* команду INT 21h. При этом текущее PSW сохраняется *(2)* в стеке, а его место замещает *(3)* адрес процедуры ввода MS-DOS, которая инициализирует *(4)* управляющий регистр. Далее система ожидает получения COM-портом символа, о чем оповещает сигнал на линии IRQ4, вызывающий *(5)* обработчик прерывания

0Ch. Для этого текущее PSW сохраняется *(6)* в стеке, его заменяет *(7)* новое PSW обработчика аппаратного прерывания, который определяет причину прерывания и читает *(8)* данные из регистра данных. После этого работа обработчика завершается *(9),* и управление возвращается *(10)* функции ввода, которая помещает введенный через COM-порт символ в регистр AL и возвращает *(11)* управление пользовательской программе, PSW которой восстанавливается *(12)* из стека. Выполняется *(13)* следующая за INT 21h команда, и программа может прочитать поступивший в порт символ в регистре AL.

- *Ввод данных с клавиатуры*

При нажатии или отпускании клавиши клавиатура посылает контроллеру прерываний по линии IRQ1 запрос на обработку прерывания и устанавливает в порте с номером 60h скан-код нажатой (отпущенной) клавиши. Если прерывание данного уровня не замаскировано и разрешено, контроллер посылает требование прерывания процессору. Процессор проверяет состояние флага IF и, если флаг не установлен, подтверждает запрос контроллера. Контроллер выставляет на шину данных номер вектора, который должен указывать на обработчик прерывания клавиатуры (в реальном режиме это обычно INT9), а процессор запоминает в стеке активной программы адрес возврата и регистр флагов, а затем помещает в регистры CS и IP значения, находящиеся в начале оперативной памяти в таблице векторов в указанном КПП векторе прерываний. После этого управление получает обработчик прерывания.

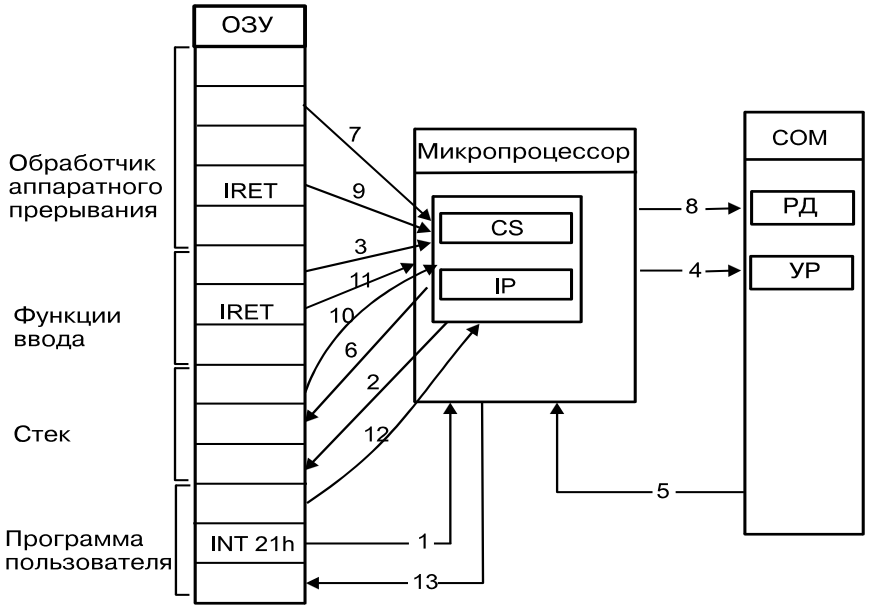

Рис. 2.24. Выполнение операции ввода из COM-порта

Процедура обработки прерывания читает из порта 60h скан-код и определяет, была ли клавиша нажата или отпущена, а также с какой именно клавишей было произведено это действие. Если действия производились с клавишей переключателем (Insert, CapsLock и т.п.) или со специальной клавишей (Ctrl, Shift, Alt), обработчик изменяет статус (нажата, отпущена) этой клавиши в специальном слове состояния клавиатуры, находящемся в области данных BIOS по адресу 0040:0017. Кроме этого, обработчик проверяет также, не была ли нажата специальная клавиша (или комбинация клавиш): PrtScr, SysRq, Pause, Ctrl-Alt-Del, Ctrl-Break, Ctrl-NumLock и др. — и, если это так, обработчик вызывает другое прерывание, обрабатывающее соответствующую ситуацию (например, 1Bh при нажатии Ctrl-Break).

В остальных случаях обработчик сопоставляет нажатую клавишу с состоянием клавиш—переключателей и формирует соответствующий код (ASCII для клавиш с символом ASCII или расширенный для не символьных клавиш и комбинаций различных клавиш с Shift, Ctrl и Alt). Этот код имеет длину 2 байта и содержит:

- для ASCII символа — его ASCII — код в младшем байте и скан-код клавиши в старшем
- для расширенного кода — его значение в старшем байте и 0 в младшем

Этот код заносится в специальный буфер клавиатуры, также находящийся в области данных BIOS (рис. 2.25). Буфер имеет дисциплину обслуживания FIFO и размер 32 байта, следовательно, он может сохранить значения 16-ти нажатий клавиш. Значения в буфер помещаются последовательно (сперва младший байт, за ним старший) и циклически, т.е. при достижении конца буфера значения помещаются в его начало. Для определения начала и конца очереди имеются указатели на начало (голову) и конец (хвост). Эти указатели хранятся по адресам 0040:001A и 0040:001C соответственно и содержат смещения от начала сегмента 0040:0000 (рис. 2.25.A).

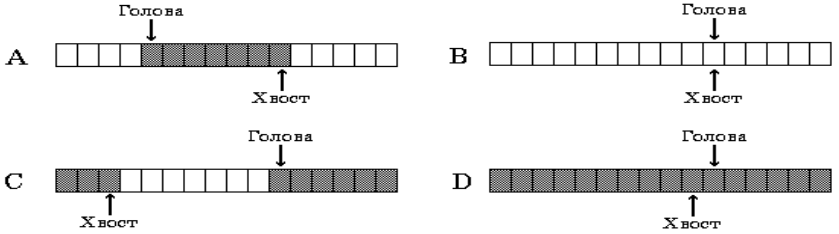

Рис. 2.25. Буфер клавиатуры

При записи нового значения в буфер указатель на хвост увеличивается на 2 и записываемое значение помещается по этому адресу. При чтении из буфера извлекается значение, находящееся по адресу, который содержится в указателе на начало, а затем этот указатель увеличивается на 2. Если значения указателей одинаковы, буфер считается пустым (рис 2.25.B). Если какой-либо указатель достигает конца буфера, он переходит на начало, т.о. указатель на хвост может иметь меньшее значение, чем указатель на голову (рис. 2.25.C). При достижении указателем на конец значения, на 2 меньшего указателя на начало, буфер считается заполненным и новое значение игнорируется (рис. 2.25.D).

Таким образом, программы, выполняющие ввод с клавиатуры, на самом деле обычно читают буфер клавиатуры. Значения, находящиеся в буфере, можно получить при помощи функций BIOS (INT 16h) или MS-DOS (INT 21h, функции 01, 06, 07, 08, 0Ah, 0Bh, 0Ch, 3Fh). Эти функции позволяют также получать состояние буфера и изменять его.

- *Ввод команд для выполнения*

Рассмотрим другой пример, демонстрирующий работу системы прерываний, а именно, отслеживание командным процессором вводимых пользователем с клавиатуры команд. При этом выполняется следующая последовательность действий:

Пользователь набирает на клавиатуре имя и параметры задания. При нажатии каждой клавиши клавиатура посылает запрос на прерывание, он обрабатывается

процедурой INT 9h, которая заносит значение введенного символа в буфер клавиатуры.

Командный процессор в это же время проверяет буфер клавиатуры на наличие в нем не введенных данных, и читает их по мере появления (при помощи функций MS-DOS). Так как эти процедуры выполняются гораздо быстрее, нежели ввод символов на клавиатуре, то большая часть процессорного времени при ожидании ввода тратится без пользы. Для повышения эффективности работы системы функции MS-DOS ввода с клавиатуры при ожидании вызывают специальное мультиплексное прерывание INT 28h, которое выполняет фоновые процессы (например, печать файлов на принтер командой PRINT).

Получив символ CR (Enter), который свидетельствует об окончании ввода, командный процессор проверяет его корректность и, в случае таковой, запускает команду или программу на выполнение при помощи соответствующего прерывания

2.10. ОБЩАЯ СХЕМА ЗАГРУЗКИ MS DOS

При включении компьютера процессор устанавливает состояние сброса, осуществляет контроль четности, устанавливает в регистре CS значение 0FFFFh, а в регистре IP — нуль. Таким образом, первая выполняемая команда находится по адресу FFFF:0, что является точкой входа в BIOS. При этом на аппаратном уровне выполняется тестирование устройств (POST-тест), заполнение векторов прерываний. BIOS проверяет порты компьютера для определения и инициализации внешних устройств. Затем BIOS создает в памяти (по адресу 0) таблицу прерываний и выполняет две операции: INT 11h (запрос списка присоединенного оборудования) и INT 12h (запрос размера физической памяти).

Далее BIOS определяет наличие OS DOS. Если обнаружена системная дискета, то BIOS выполняет прерывание INT 19h для доступа к первому сектору диска, который содержит блок начальной загрузки (Master boot), Считывается 0 дорожка 1 сектор цилиндра 0, определяется активный раздел диска (для Hard Drive), считывается первый сектор активного раздела, в начале которого находится адрес программы начальной загрузки операционной системы, она считывается и ей передается управление.

Для функционирования операционной системы MS-DOS в оперативную память программой начальной загрузки загружается обязательная часть MS-DOS: MSDOS.SYS, IO.SYS, COMMAND.COM.

- IO.SYS — обеспечивает логический интерфейс низкого уровня между системой и аппаратурой (через BIOS); содержит набор резидентных драйверов основных периферийных устройств. При инициализации программа определяет состояние всех устройств и оборудования, а также управляет операциями обмена между памятью и внешними устройствами.

- В MSDOS.SYS располагаются программы управления памятью, файлами, данными, внешними устройствами, заданиями, процессами. Это центральный компонент DOS, реализующий основные функции ОС ── управление ресурсами ПК и выполняемыми программами. Основу составляют обработчики прерываний верхнего уровня.
- COMMAND.COM — командный процессор, обеспечивающий интерфейс пользователя с вычислительной системой

Файл IO.SYS, загруженный с диска, обычно состоит из двух модулей: BIOS и SYSINIT. SYSINIT вызывается с помощью программы инициализации BIOS. Модуль определяет величину непрерывной памяти, доступной системе, и затем располагает SYSINIT в старшие адреса. Далее модуль переносит ядро системы DOS, MSDOS.SYS, из области ее начальной загрузки в область окончательного расположения.

Далее SYSINIT вызывает программу инициализации в модуле MSDOS.SYS. Ядро DOS инициализирует ее внутренние таблицы и рабочие области, устанавливает вектора прерываний по адресам с 20h по 2Fh и перебирает связанный список резидентных драйверов устройств, вызывая функцию инициализации для каждого из них.

Когда ядро DOS проинициализировано и все резидентные драйверы доступны, модуль SYSINIT может вызвать обычный файловый сервис системы MS-DOS и открыть файл CONFIG.SYS. Драйверы, в соответствии с директивами DEVICE=, указанными в CONFIG.SYS, последовательно загружаются в память, активизируются с помощью вызовов соответствующих модулей инициализации и заносятся в связанный список драйверов. Функции инициализации каждого из них сообщают SYSINIT размер памяти, отведенный под соответствующий драйвер.

После загрузки всех установленных драйверов, SYSINIT закрывает все дескрипторы файлов и открывает консоль, принтер и последовательный порт. Это позволяет замещать резидентные драйверы BIOS стандартных устройств. Модуль SYSINIT перемещает себя к старшим адресам памяти и сдвигает ядро системы DOS в строну младших адресов памяти в область окончательного расположения.

В конце своего выполнения SYSINIT вызывает системную функцию EXEC для загрузки интерпретатора командной строки, т.е. COMMAND.COM.

COMMAND.COM ── командный процессор, который обеспечивает интерфейс пользователя с установкой.

Система загружает две части программы Command.com в память:

1. резидентная часть — обрабатывает все ошибки дисковых операций ввода / вывода и управляет прерываниями:
   - INT 22h — адрес обработки завершения задачи;
   - INT 23h — адрес реакции на Ctr/Break;
   - INT 24h — адрес реакции на фатальную ошибку.

транзитная часть — загружается в самые старшие адреса памяти. Она содержит настраивающий загрузчик и предназначена для загрузки .COM или .EXE файлов с диска в память для выполнения. В результате получаем окончательное распределение памяти. В зависимости от модификации персонального компьютера и состава его периферийного оборудования, распределение адресного пространства может несколько различаться. Тем не менее, размещение основных компонентов системы довольно строго унифицировано. Типичная схема использования адресного пространства приведена в табл. 2.7.

Первые 640 Кбайт адресного пространства с адресами 00000h до 9FFFFh отводится под основную оперативную память, которая еще называют стандартной (Conventional). Начальный килобайт оперативной памяти занят векторами прерываний (256 векторов по 4 байта). Вслед за векторами прерываний располагается область данных BIOS, которая занимает адреса от 00400h до 004FFh. В этой области хранятся разнообразные данные, используемые программами BIOS в процессе управления периферийным оборудованием.

Так, здесь размещаются:

- входной буфер клавиатуры с системой указателей;
- адреса последовательных и параллельных портов;
- данные, характеризующие настройку видеосистемы (форма курсора, его текущее местоположение на экране, текущий видеорежим и пр.);
- ячейки для отсчета текущего времени; область межзадачных связей и т.д.

1 Кбайт	Векторы прерываний	00400h
256 байт	Область данных BIOS	00500h
512 байт	Область данных DOS	00700h
	IO.SYS и MSDOS.SYS	
	Загружаемые драйверы	
	Command.com (резидентная часть)	
	Свободная память для загружаемых прикладных и системных программ	A0000h
64 Кбайт	Графический буфер	B0000h
32 Кбайт	UMB	B8000h
32 Кбайт	Текстовый буфер	C0000h
64 Кбайт	ПЗУ — расширения BIOS	D0000h
64 Кбайт	UMB	E0000h
128 Кбайт	ПЗУ — расширения BIOS	1000000h
64 Кбайт	HMA	10FFF0h
До 15Мбайт	XMS	

Табл. 2.7. Окончательное распределение адресного пространства

Область данных BIOS заполняется информацией в процессе начальной загрузки компьютера и динамически модифицируется системой по мере необходимости.

В области памяти, начиная с 500h, содержатся некоторые системные данные MS-DOS. Вслед за областью данных MS-DOS располагается собственно операционная система, загружаемая из файлов IO.SYS и MSDOS.SYS (IBMBIO.COM и IBMDOS.COM для системы PC-DOS).

Вся оставшаяся память до границы 640 Кбайт называется транзитной областью. Она свободна для загрузки любых системных или прикладных программ. Оставшиеся 384 Кбайта выше 640 Кбайт адресного пространства называются старшей памятью (Upper Memory). Первоначально они были предназначены для размещения постоянных запоминающих устройств (ПЗУ). Практически под ПЗУ занята только часть адресов. Поэтому часть старшей памяти оказывается свободной. Эти участки используются для адресации к добавочным блокам оперативной памяти, которые носят название блоков старшей памяти (Upper Memory Blocks, UMB). Для поддержки старшей памяти используется драйвер HIMEM.SYS, который позволяет эффективно использовать UMB, загружая в них устанавливаемые драйверы устройств, а также резидентные программы. Загрузка системных программ в UMB освобождает от них стандартную память, увеличивая ее транзитную область. Наряду со стандартной памятью (640 Кбайт) используется расширенная память объемом до 15 Мбайт.

В транзитную часть грузится программа диалога с пользователем, после чего система готова к восприятию команд пользователя.

Задания фиксируются в кольцевом буфере, по <CR> интерпретатор начинает выполнение поступившего задания (выполнение передается MS-DOS.SYS).

Если найден файл с расширением .COM или .EXE, то COMMAND.COM, используя системную функцию EXEC, осуществляет загрузку и выполнение найденного файла. Если в области транзитных программ достаточно свободной памяти, то EXEC выделяет блок памяти под новую программу, строит PSP по его базовому адресу, а затем считывает программу в память непосредственно за PSP. В конце своей работы EXEC устанавливает регистры сегмента и стека и передает управление программе.

При инициализации ядра и системы активизируются и исполняются следующие процессы операционной системы: администратор памяти, программа работы с ВУ, программа поддержки файловой системы и процесс, подчиненный таймеру. После запуска системы управление передается внутренним процедурам COMMAND.COM, среди которых можно выделить программу системного ввода,

анализирующую входной поток заданий. Программа работает в режиме сторожа; входной поток заданий накапливается во входных очередях (первый уровень классического планирования процессов). Как только система определяет, что у нее есть ресурсы (наиболее важно наличие ОП), операционная система загружает планировщик как транзитную программу, которая анализирует наличие остальных ресурсов, инициализируя очередь процессов к процессору.

После завершения выполнения транзитной программы вызывается специальная функция завершения, которая в свою очередь освобождает память от транзитной программы и возвращает управление программе, вызвавшей загрузку транзитной программы (Command.com).

При загрузке COM — программы DOS устанавливает в четырех сегментных регистрах адрес первого байта PSP. Затем устанавливает указатель стека на конец сегмента объемом 64Кбайт (FFFFh). В вершину стека заносится нулевое слово. В командный указатель помещается 100h (размер PSP). После этого управление передается по адресу регистровой пары CS:IP. Этот адрес является началом выполняемой COM-программы. При выходе из программы команда RET заносит в регистр IP нулевое слово. В этом случаи в CS:IP получается адрес первого байта PSP, где находится команда INT 20h. При этом управление передается в резидентную часть Command.com.

Программа в формате EXE состоит из двух частей:

1. заголовка — записи, содержащей информацию по управлению и настройке программы;
2. загрузочного модуля.

После инициализации регистры CS и SS содержат правильные адреса сегментов, а регистр DS (и ES) должен быть настроен в программе на собственный сегмент данных. При завершении программы команда RET заносит в регистр IP нулевое значение. В регистровой паре CS:IP получается адрес первого байта PSP. После выполнения команды управление передается в DOS.

MS-DOS отводит префиксу область в 256 байт в начале блока памяти, выделяемого транзитной программе (Табл. 2.8.) Префикс имеет несколько связей с MS-DOS, которые могут использоваться транзитной программой; кроме того, определенную информацию записывает в него MS-DOS как для собственных целей, так и для передачи транзитной программе, которая в случае необходимости может эту информацию использовать.

Системное прерывание int 20h
Сегмент, конец выделенного блока
Зарезервировано
Длинный вызов диспетчера функций MS-DOS

Предыдущие содержание вектора прерывания обработки завершения (int 22h)
Предыдущее содержание вектора прерывания Ctr-C (int 23h)
Предыдущее содержание вектора прерывания обработки критической ошибки (int 24h)
Зарезервировано
Сегментный адрес блока окружения
Зарезервировано
Стандартный блок управления файлами № 1
Стандартный блок управления файлами № 2 (перекрывается если FCB 1 открыт)
Хвост команды и стандартная область пересылки диска

Табл. 2.87. Структура PSP

2.11. ОСНОВЫ ФУНКЦИОНИРОВАНИЯ МНОГОПРОГРАММНЫХ ОПЕРАЦИОННЫХ СИСТЕМ (СТРУКТУРА, СОСТАВ, ВЗАИМОДЕЙСТВИЕ)

Дальнейшее рассмотрение ядра операционной системы будет основано на детальном описании операционной системы UNIX и OS/2.

2.11.1. Принципы функционирования UNIX

Ядро UNIX и взаимодействие задач

На рис. 2.26 изображена архитектура верхнего уровня системы UNIX. Технические средства, показанные в центре диаграммы, выполняют функции, обеспечивающие функционирование операционной системы.

Операционная система взаимодействует с аппаратурой непосредственно, обеспечивая обслуживание программ и их независимость от деталей аппаратной конфигурации. Если представить систему состоящей из пластов, в ней можно выделить системное ядро, изолированное от пользовательских программ. Поскольку программы не зависят от аппаратуры, их легко переносить из одной системы UNIX в другую, функционирующую на другом комплексе технических средств, если только в этих программах не подразумевается работа с конкретным оборудованием. Например, программы, рассчитанные на определенный размер машинного слова, гораздо труднее переводить на другие машины по сравнению с программами, не требующими подобных установок.

Программы, подобные командному процессору shell и редакторам (ed и vi), и показанные на внешнем по отношению к ядру, слое, взаимодействуют с ядром при помощи хорошо определенного набора обращений к операционной системе. Обращения к операционной системе принуждают ядро к выполнению различных операций, которых требует вызывающая программа, и обеспечивают обмен

данными между ядром и программой. Некоторые из программ, приведенных на рисунке, в стандартных конфигурациях системы известны как команды, однако, на одном уровне с ними могут располагаться и доступные пользователю программы, такие как программа a.out, стандартное имя для исполняемого файла, созданного компилятором с языка Си. Другие прикладные программы располагаются выше указанных программ на верхнем уровне, как это показано на рисунке. Например, стандартный компилятор с языка Си, **cc**, располагается на самом внешнем слое: он вызывает препроцессор для Си, ассемблер и загрузчик (компоновщик), т.е. отдельные программы предыдущего уровня. Хотя на рисунке приведена двухуровневая иерархия прикладных программ, пользователь может расширить иерархическую структуру на столько уровней, сколько необходимо. В самом деле, стиль программирования, принятый в системе UNIX, допускает разработку комбинации программ, выполняющих одну и ту же общую задачу.

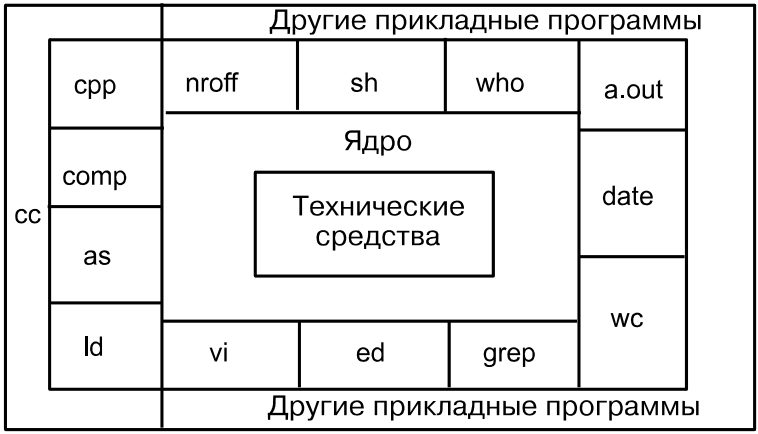

Рис. 2.26. Архитектура системы UNIX

Многие прикладные подсистемы и программы, составляющие верхний уровень системы, такие как командный процессор **shell**, редакторы, SCCS (система обработки исходных текстов программ) и пакеты программ подготовки документации, постепенно становятся синонимом понятия "система UNIX". Однако, все они пользуются услугами программ нижних уровней и в конечном счете, ядра с помощью набора обращений к операционной системе. Набор обращений к операционной системе вместе с реализующими их внутренними алгоритмами составляют "тело" ядра. Короче говоря, ядро реализует функции, на которых основывается выполнение всех прикладных программ в системе UNIX, и им же определяются эти функции.

На рис. 2.26. уровень ядра операционной системы изображен непосредственно под уровнем прикладных программ пользователя. Выполняя различные элементарные операции по запросам пользовательских процессов, ядро обеспечивает функционирование пользовательского интерфейса, описанного выше.

Среди функций ядра можно отметить:

- Управление выполнением процессов посредством их создания, завершения или приостановки и организации взаимодействия между ними.
- Планирование очередности предоставления выполняющимся процессам времени центрального процессора (диспетчеризация). Процессы работают с центральным процессором в режиме разделения времени: центральный процессор выполняет процесс, по завершении отсчитываемого ядром кванта времени процесс приостанавливается и ядро активизирует выполнение другого процесса. Позднее ядро запускает приостановленный процесс.
- Выделение выполняемому процессу оперативной памяти. Ядро операционной системы дает процессам возможность совместно использовать участки адресного пространства на определенных условиях, защищая при этом адресное пространство, выделенное процессу, от вмешательства извне. Если системе требуется свободная память, ядро освобождает память, временно выгружая процесс на внешние запоминающие устройства, которые называют устройствами выгрузки. Если ядро выгружает процессы на устройства выгрузки целиком, такая реализация системы UNIX называется системой со свопингом (подкачкой); если же на устройство выгрузки выводятся страницы памяти, такая система называется системой с замещением страниц.
- Выделение внешней памяти с целью обеспечения эффективного хранения информации и выборка данных пользователя. Именно в процессе реализации этой функции создается файловая система. Ядро выделяет внешнюю память под пользовательские файлы, мобилизует неиспользуемую память, структурирует файловую систему в форме, доступной для понимания, и защищает пользовательские файлы от несанкционированного доступа.
- Управление доступом процессов к периферийным устройствам, таким как терминалы, ленточные устройства, дисководы и сетевое оборудование.

Выполнение ядром своих функций довольно очевидно. Например, оно узнает, что данный файл является обычным файлом или устройством, но скрывает это различие от пользовательских процессов. Так же оно, форматируя информацию файла для внутреннего хранения, защищает внутренний формат от пользовательских процессов, возвращая им неотформатированный поток байтов. Наконец, ядро реализует ряд необходимых функций по обеспечению выполнения процессов пользовательского уровня, за исключением функций, которые могут быть реализованы на самом пользовательском уровне. Например, ядро выполняет действия, необходимые shell'у как интерпретатору команд: оно позволяет процессору shell читать вводимые с терминала данные, динамически порождать процессы, синхронизировать выполнение процессов, открывать каналы и переадресовывать ввод/вывод. Пользователи могут разрабатывать свои версии

командного процессора shell с тем, чтобы привести рабочую среду в соответствие со своими требованиями, не затрагивая других пользователей. Такие программы пользуются теми же услугами ядра, что и стандартный процессор shell.

Система UNIX позволяет таким устройства, как внешние устройства ввода/вывода и системные часы, асинхронно прерывать работу центрального процессора. По получении сигнала прерывания ядро операционной системы сохраняет свой текущий контекст (застывший образ выполняемого процесса), устанавливает причину прерывания и обрабатывает прерывание. После того, как прерывание будет обработано ядром, прерванный контекст восстановится и работа продолжится так, как будто ничего не случилось. Устройствам обычно приписываются приоритеты в соответствии с очередностью обработки прерываний. В процессе обработки прерываний ядро учитывает их приоритеты и блокирует обслуживание прерывания с низким приоритетом на время обработки прерывания с более высоким приоритетом.

Особые ситуации связаны с возникновением незапланированных событий, вызванных процессом, таких как недопустимая адресация, задание привилегированных команд, деление на ноль и т.д. Они отличаются от прерываний, которые вызываются событиями, внешними по отношению к процессу. Особые ситуации возникают непосредственно при выполнении команды, и система, обработав особую ситуацию, пытается перезапустить команду; считается, что прерывания возникают между выполнением двух команд, при этом система после обработки прерывания продолжает выполнение процесса уже начиная со следующей команды. Для обработки прерываний и особых ситуаций в системе UNIX используется один и тот же механизм.

Ядро иногда обязано предупреждать возникновение прерываний во время критических действий, могущих в случае прерывания запортить информацию. Например, во время обработки списка с указателями возникновение прерывания от диска для ядра нежелательно, т.к. при обработке прерывания можно испортить указатели. Обычно имеется ряд привилегированных команд, устанавливающих уровень прерывания процессора в слове состояния процессора. Установка уровня прерывания на определенное значение отсекает прерывания этого и более низких уровней, разрешая обработку только прерываний с более высоким приоритетом. На рис. 2.27. показана последовательность уровней прерывания. Если ядро игнорирует прерывания от диска, в этом случае игнорируются и все остальные прерывания, кроме прерываний от часов и машинных сбоев.

Рис. 2.27. Стандартные уровни прерываний

Ядро постоянно располагается в оперативной памяти, наряду с выполняющимся в данный момент процессом (или частью его, по меньшей мере). В процессе компиляции программа—компилятор генерирует последовательность адресов, являющихся адресами переменных и информационных структур, а также адресами инструкций и функций. Компилятор генерирует адреса для виртуальной машины так, словно на физической машине не будет выполняться параллельно с транслируемой ни одна другая программа.

Когда программа запускается на выполнение, ядро выделяет для нее место в оперативной памяти, при этом совпадение виртуальных адресов, сгенерированных компилятором, с физическими адресами совсем необязательно. Ядро, взаимодействуя с аппаратными средствами, транслирует виртуальные адреса в физические, т.е. отображает адреса, сгенерированные компилятором, в физические, машинные адреса. Такое отображение опирается на возможности аппаратных средств, поэтому компоненты системы UNIX, занимающиеся им, являются машинно — зависимыми. Например, отдельные вычислительные машины имеют специальное оборудование для подкачки выгруженных страниц памяти.

Обращения к операционной системе позволяют процессам производить операции, которые иначе не выполняются. В дополнение к обработке подобных обращений ядро операционной системы осуществляет общие учетные операции, управляет планированием процессов, распределением памяти и защитой процессов в оперативной памяти, обслуживает прерывания, управляет файлами и устройствами и обрабатывает особые ситуации, возникающие в системе. В функции ядра системы UNIX намеренно не включены многие функции, являющиеся частью других операционных систем, поскольку набор обращений к системе позволяет процессам выполнять все необходимые операции на пользовательском уровне.

Обе сущности, файлы и процессы, являются центральными понятиями модели операционной системы UNIX. На рис. 2.28. представлена блок-схема ядра системы, отражающая состав модулей, из которых состоит ядро, и их взаимосвязи друг с другом. В частности, на ней слева изображена файловая подсистема, а справа — подсистема управления процессами, две главные компоненты ядра. Эта схема дает логическое представление о ядре, хотя в действительности в структуре ядра имеются отклонения от модели, поскольку отдельные модули испытывают внутреннее воздействие со стороны других модулей.

Схема на рис. 2.28. имеет три уровня: уровень пользователя, уровень ядра и уровень аппаратуры. Обращения к операционной системе и библиотеки составляют границу между пользовательскими программами и ядром, проведенную на рис. 2.27. Обращения к операционной системе выглядят так же, как обычные вызовы функций в программах на языке Си, и библиотеки устанавливают соответствие между этими вызовами функций и элементарными системными операциями. При этом программы на ассемблере могут обращаться к операционной системе непосредственно, без использования библиотеки системных вызовов. Программы часто обращаются к другим библиотекам, таким как библиотека стандартных подпрограмм ввода/вывода, достигая тем самым более полного использования системных услуг. Для этого во время компиляции библиотеки связываются с программами и частично включаются в программу пользователя.

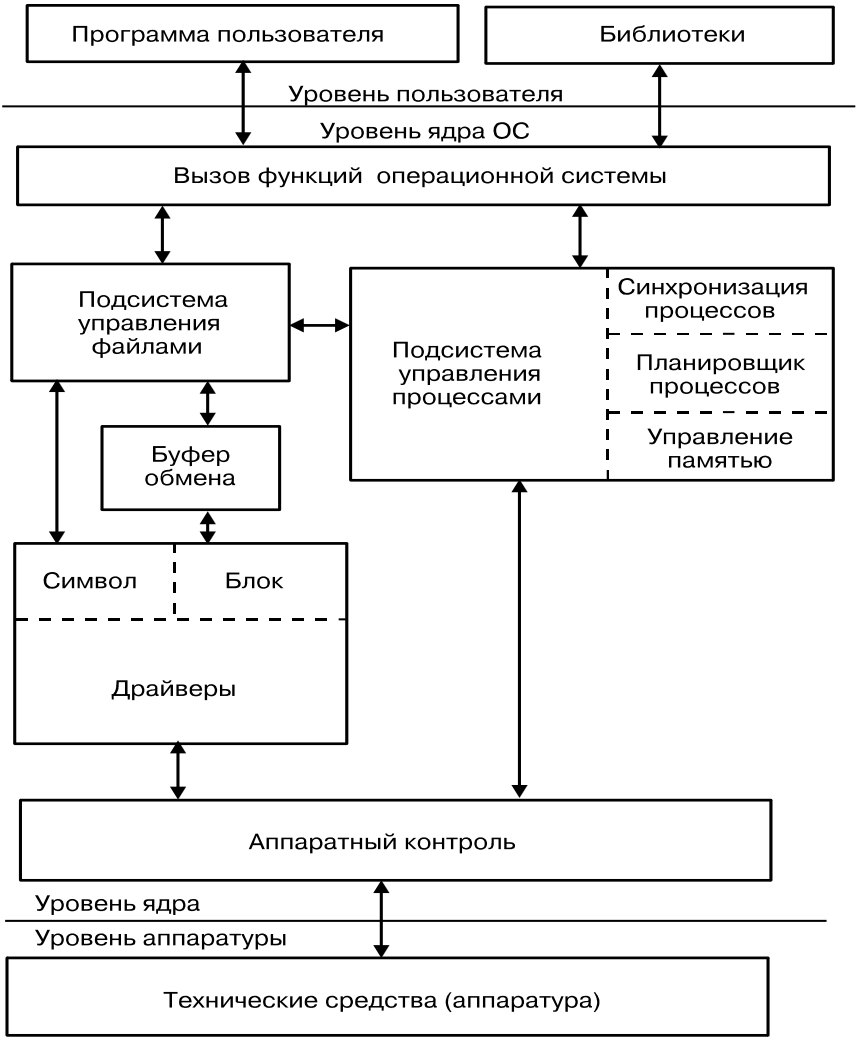

Рис. 2.28. Блок — схема ядра операционной системы

Совокупность обращений к операционной системе разделена на те обращения, которые взаимодействуют с подсистемой управления файлами и те, которые взаимодействуют с подсистемой управления процессами. Файловая подсистема

управляет файлами, размещает записи файлов, управляет свободным пространством, доступом к файлам и поиском данных для пользователей. Процессы взаимодействуют с подсистемой управления файлами, используя при этом совокупность специальных обращений к операционной системе, таких как open (для того, чтобы открыть файл на чтение или запись), close, read, write, stat (запросить атрибуты файла), chown (изменить запись с информацией о владельце файла) и chmod (изменить права доступа к файлу).

Подсистема управления файлами обращается к данным, которые хранятся в файле, используя буферный механизм, управляющий потоком данных между ядром и устройствами внешней памяти. Буферный механизм, взаимодействуя с драйверами устройств ввода/вывода блоками, инициирует передачу данных к ядру и обратно. Драйверы устройств являются такими модулями в составе ядра, которые управляют работой периферийных устройств. Устройства ввода/вывода блоками относятся программами пользователя к типу запоминающих устройств с произвольной выборкой; их драйверы построены таким образом, что все остальные компоненты системы воспринимают эти устройства как запоминающие устройства с произвольной выборкой. Например, драйвер запоминающего устройства на магнитной ленте позволяет ядру системы воспринимать это устройство как запоминающее устройство с произвольной выборкой. Подсистема управления файлами также непосредственно взаимодействует с драйверами устройств " неструктурированного" ввода/вывода без вмешательства буферного механизма. К устройствам неструктурированного ввода/вывода, иногда именуемым устройствами посимвольного ввода/вывода (текстовыми), относятся устройства, отличные от устройств ввода/вывода блоками.

Подсистема управления процессами отвечает за синхронизацию процессов, взаимодействие процессов, распределение памяти и планирование выполнения процессов. Подсистема управления файлами и подсистема управления процессами взаимодействуют между собой, когда файл загружается в память на выполнение: подсистема управления процессами читает в память исполняемые файлы перед тем, как их выполнить.

Примерами обращений к операционной системе, используемых при управлении процессами, могут служить fork (создание нового процесса), exec (наложение образа программы на выполняемый процесс), exit (завершение выполнения процесса), wait (синхронизация продолжения выполнения основного процесса с моментом выхода из порожденного процесса), brk (управление размером памяти, выделенной процессу) и signal (управление реакцией процесса на возникновение экстраординарных событий).

Модуль распределения памяти контролирует выделение памяти процессам. Если в какой-то момент система испытывает недостаток в физической памяти для запуска всех процессов, ядро пересылает процессы между основной и внешней памятью с тем, чтобы все процессы имели возможность выполняться. Существует два способа управления распределением памяти: выгрузка (подкачка) и замещение страниц. Программу подкачки иногда называют планировщиком, т.к. она

"планирует" выделение памяти процессам и оказывает влияние на работу планировщика центрального процессора.

Модуль "планировщик" распределяет между процессами время центрального процессора. Он планирует очередность выполнения процессов до тех пор, пока они добровольно не освободят центральный процессор, дождавшись выделения ресурса, или до тех пор, пока ядро системы не выгрузит их после того, как их время выполнения превысит заранее определенный квант времени. Планировщик выбирает на выполнение готовый к запуску процесс с наивысшим приоритетом; выполнение предыдущего процесса (приостановленного) будет продолжено тогда, когда его приоритет будет наивысшим среди приоритетов всех готовых к запуску процессов. Существует несколько форм взаимодействия процессов между собой: от асинхронного обмена сигналами о событиях до синхронного обмена сообщениями.

Наконец, аппаратный контроль отвечает за обработку прерываний и за связь с машиной. Такие устройства, как диски и терминалы, могут прерывать работу центрального процессора во время выполнения процесса. При этом ядро системы после обработки прерывания может возобновить выполнение прерванного процесса. Прерывания обрабатываются не самими процессами, а специальными функциями ядра системы, перечисленными в контексте выполняемого процесса.

Ядро системы идентифицирует каждый процесс по его номеру, который называется идентификатором процесса (PID). Нулевой процесс является особенным процессом, который создается "вручную" в результате загрузки системы; после порождения нового процесса (процесс 1) нулевой процесс становится процессом подкачки. Процесс 1, известный под именем init, является предком любого другого процесса в системе и связан с каждым процессом особым образом.

Пользователь, транслируя исходный текст программы, создает исполняемый файл, который состоит из нескольких частей:

- набора "заголовков", описывающих атрибуты файла,
- текста программы,
- представления на машинном языке данных, имеющих начальные значения при запуске программы на выполнение, и указания на то, сколько пространства памяти ядро системы выделит под неинициализированные данные, так называемые bss (*) (ядро устанавливает их в 0 в момент запуска),
- других секций, таких как информация символических таблиц.

Ядро загружает исполняемый файл в память при выполнении системной операции exec, при этом загруженный процесс состоит по меньшей мере из трех частей, так называемых областей: текста, данных и стека. Области текста и данных корреспондируют с секциями текста и bss-данных исполняемого файла, а область стека создается автоматически и ее размер динамически устанавливается ядром системы во время выполнения. Стек состоит из логических записей активации, помещаемых в стек при вызове функции и выталкиваемых из стека при возврате управления в вызвавшую процедуру; специальный регистр, именуемый указателем вершины стека, показывает текущую глубину стека. Запись активации включает параметры передаваемые функции, ее локальные переменные, а также данные,

необходимые для восстановления предыдущей записи активации, в том числе значения счетчика команд и указателя вершины стека в момент вызова функции.

Текст программы включает последовательности команд, управляющие увеличением стека, а ядро системы выделяет, если нужно, место под стек.

Поскольку процесс в системе UNIX может выполняться в двух режимах, режиме ядра или режиме задачи, он пользуется в каждом из этих режимов отдельным стеком. Стек задачи содержит аргументы, локальные переменные и другую информацию относительно функций, выполняемых в режиме задачи. Слева на Рисунке 2.29. показан стек задачи для процесса, связанного с выполнением системной операции write в программе copy. Процедура запуска процесса (включенная в библиотеку) обратилась к функции main с передачей ей двух параметров, поместив в стек задачи запись 1; в записи 1 есть место для двух локальных переменных функции main. Функция main затем вызывает функцию copy с передачей ей двух параметров, old и new, и помещает в стек задачи запись 2; в записи 2 есть место для локальной переменной count. Наконец, процесс активизирует системную операцию write, вызвав библиотечную функцию с тем же именем. Каждой системной операции соответствует точка входа в библиотеке системных операций; библиотека системных операций написана на языке Ассемблера и включает специальные команды прерывания, которые, выполняясь, порождают "прерывание", вызывающее переключение аппаратуры в режим ядра. Процесс ищет в библиотеке точку входа, соответствующую отдельной системной операции, подобно тому, как он вызывает любую из функций, создавая при этом для библиотечной функции запись активации. Когда процесс выполняет специальную инструкцию, он переключается в режим ядра, выполняет операции ядра и использует стек ядра.

Рис. 2.29. Стеки задачи и ядра для программы копирования.

Стек ядра содержит записи активации для функций, выполняющихся в режиме ядра. Элементы функций и данных в стеке ядра соответствуют функциям и данным, относящимся к ядру, но не к программе пользователя, тем не менее, конструкция стека ядра подобна конструкции стека задачи. Стек ядра для процесса пуст, если процесс выполняется в режиме задачи. Справа на Рисунке 2.30 представлен стек ядра для процесса выполнения системной операции write в программе copy.

Каждому процессу соответствует точка входа в таблице процессов ядра, кроме того, каждому процессу выделяется часть оперативной памяти, отведенная под задачу пользователя. Таблица процессов включает в себя указатели на промежуточную таблицу областей процессов, точки входа в которую служат в качестве указателей на собственно таблицу областей. Областью называется непрерывная зона адресного пространства, выделяемая процессу для размещения текста, данных и стека. Точки входа в таблицу областей описывают атрибуты области, как, например, хранятся ли в области текст программы или данные, закрытая ли эта область или же совместно используемая, и где конкретно в памяти размещается содержимое области. Внешний уровень косвенной адресации (через промежуточную таблицу областей, используемых процессами, к собственно таблице областей) позволяет независимым процессам совместно использовать области. Когда процесс запускает системную операцию exec, ядро системы выделяет области под ее текст, данные и стек, освобождая старые области, которые использовались процессом. Если процесс запускает операцию fork, ядро удваивает размер адресного пространства старого процесса, позволяя процессам совместно использовать области, когда это возможно, и, с другой стороны, производя физическое копирование. Если процесс запускает операцию exit, ядро освобождает области, которые использовались процессом. На рис. 2.31 изображены информационные структуры, связанные с запуском процесса. Таблица процессов ссылается на промежуточную таблицу областей, используемых процессом, в которой содержатся указатели на записи в собственно таблице областей, соответствующие областям для текста, данных и стека процесса.

Запись в таблице процессов и часть адресного пространства задачи, выделенная процессу, содержат управляющую информацию и данные о состоянии процесса. Это адресное пространство является расширением соответствующей записи в таблице процессов. В качестве полей в таблице процессов выступают:

- поле состояния,
- идентификаторы, которые характеризуют пользователя, являющегося владельцем процесса (код пользователя или UID),
- значение дескриптора события, когда процесс приостановлен (находится в состоянии "сна").

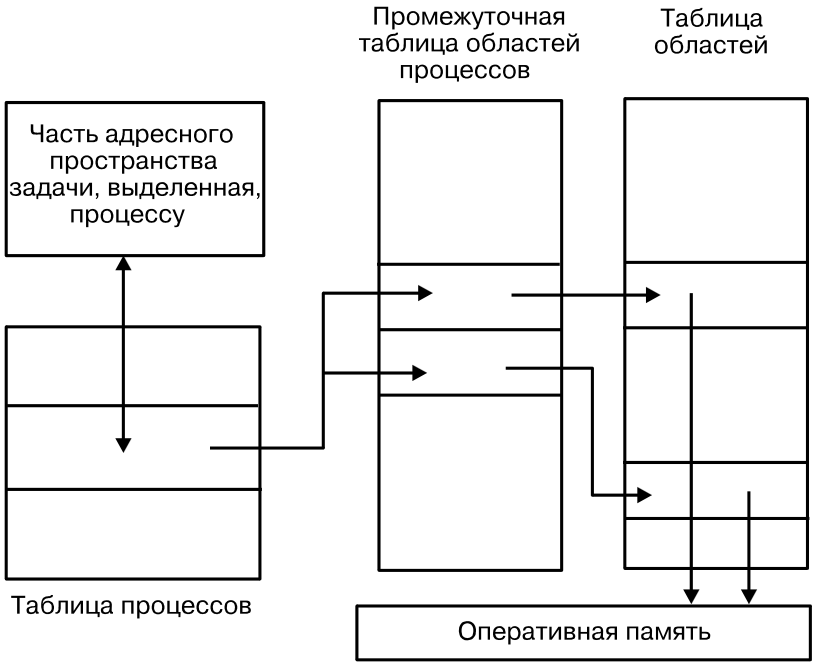

Рис. 2.30. Информационные структуры для процессов

Адресное пространство задачи, выделенное процессу, содержит описывающую процесс информацию, доступ к которой должен обеспечиваться только во время выполнения процесса.

Важными полями являются:

- указатель на позицию в таблице процессов, соответствующую текущему процессу,
- параметры текущей системной операции, возвращаемые значения и коды ошибок,
- дескрипторы файла для всех открытых файлов,
- внутренние параметры ввода/вывода,
- текущий каталог и текущий корень,
- границы файлов и процесса.

Ядро системы имеет непосредственный доступ к полям адресного пространства задачи, выделенного выполняемому процессу, но не имеет доступ к соответствующим полям других процессов. С точки зрения внутреннего алгоритма, при обращении к адресному пространству задачи, выделенному выполняемому

процессу, ядро ссылается на структурную переменную u, и, когда запускается на выполнение другой процесс, ядро перенастраивает виртуальные адреса таким образом, чтобы структурная переменная u указывала бы на адресное пространство задачи для нового процесса. В системной реализации предусмотрено облегчение идентификации текущего процесса благодаря наличию указателя на соответствующую запись в таблице процессов из адресного пространства задачи.

Большинство информационных структур ядра размещается в таблицах фиксированного размера, а не в динамически выделенной памяти. Преимущество такого подхода состоит в том, что программа ядра проста, но в ней ограничивается число элементов информационной структуры до значения, предварительно заданного при генерации системы. Если во время функционирования системы число элементов информационной структуры ядра выйдет за указанное значение, ядро не сможет динамически выделить место для новых элементов и должно сообщить об ошибке пользователю, сделавшему запрос. Если, с другой стороны, ядро сгенерировано таким образом, что выход за границы табличного пространства будет маловероятен, дополнительное табличное пространство может не понадобиться, поскольку оно не может быть использовано для других целей. Как бы то ни было, простота алгоритмов ядра представляется более важной, чем сжатие последних байтов оперативной памяти. Обычно в алгоритмах для поиска свободных мест в таблицах используются несложные циклы и этот метод более понятен и иногда более эффективен по сравнению с более сложными схемами выделения памяти.

Несмотря на то, что ядро работает в контексте процесса, отображение виртуальных адресов, связанных с ядром, осуществляется независимо от всех процессов. Программы и структуры данных ядра резидентны в системе и совместно используются всеми процессами. При запуске системы происходит загрузка программ ядра в память с установкой соответствующих таблиц и регистров для отображения виртуальных адресов ядра в физические. Таблицы страниц для ядра имеют структуру, аналогичную структуре таблицы страниц, связанной с процессом, а механизмы отображения виртуальных адресов ядра похожи на механизмы, используемые для отображения пользовательских адресов. На многих машинах виртуальное адресное пространство процесса разбивается на несколько классов, в том числе системный и пользовательский, и каждый класс имеет свои собственные таблицы страниц. При работе в режиме ядра система разрешает доступ к адресам ядра, при работе же в режиме задачи такого рода доступ запрещен. Поэтому, когда в результате прерывания или выполнения системной функции происходит переход из режима задачи в режим ядра, операционная система по договоренности с техническими средствами разрешает ссылки на адреса ядра, а при возврате в режим ядра эти ссылки уже запрещены. В других машинах можно менять преобразование виртуальных адресов, загружая специальные регистры во время работы в режиме ядра.

На рис. 2.31 приведен пример, в котором виртуальные адреса от 0 до 4М-1 принадлежат ядру, а начиная с 4М — процессу. Имеются две группы регистров

управления памятью, одна для адресов ядра и одна для адресов процесса, причем каждой группе соответствует таблица страниц, хранящая номера физических страниц со ссылкой на адреса виртуальных страниц. Адресные ссылки с использованием группы регистров ядра допускаются системой только в режиме ядра; следовательно, для перехода между режимом ядра и режимом задачи требуется только, чтобы система разрешила или запретила адресные ссылки с использованием группы регистров ядра.

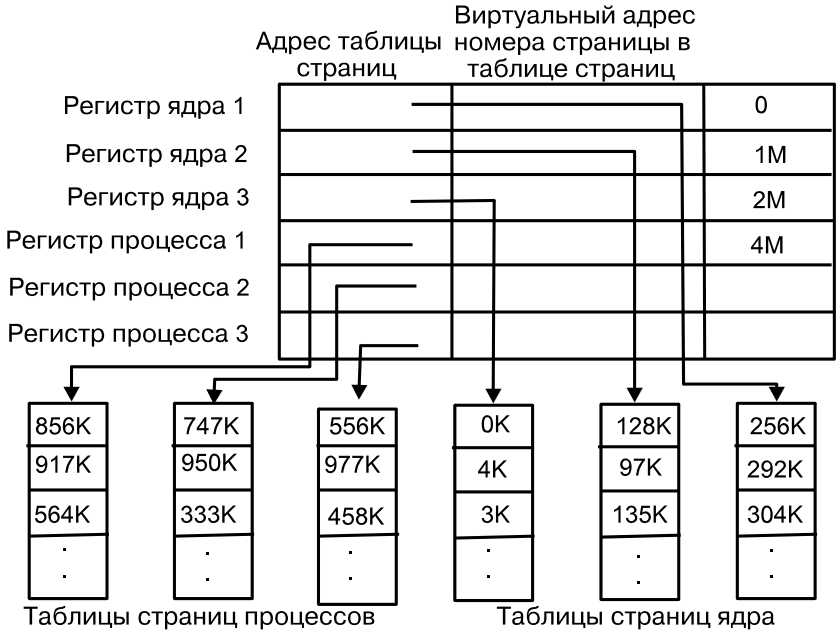

Рис. 2.31. Переключение режима работы с непривилегированного (режима задачи) на привилегированный (режим ядра)

В некоторых системах ядро загружается в память таким образом, что большая часть виртуальных адресов ядра совпадает с физическими адресами, и функция преобразования виртуальных адресов в физические превращается в функцию тождественности. Работа с пространством процесса, тем не менее, требует, чтобы преобразование виртуальных адресов в физические производилось ядром.

Запись в таблице процессов и часть адресного пространства задачи, выделенная процессу, содержат управляющую информацию и данные о состоянии процесса.

Это адресное пространство является расширением соответствующей записи в таблице процессов. В качестве полей в таблице процессов выступают:

- поле состояния,
- идентификаторы, которые характеризуют пользователя, являющегося владельцем процесса (код пользователя или UID),
- значение дескриптора события, когда процесс приостановлен (находится в состоянии "сна").

Запись в таблице процессов и часть адресного пространства задачи, выделенная процессу, содержат управляющую информацию и данные о состоянии процесса. Это адресное пространство является расширением соответствующей записи в таблице процессов. В качестве полей в таблице процессов выступают:

- поле состояния,
- идентификаторы, которые характеризуют пользователя, являющегося владельцем процесса (код пользователя или UID),
- значение дескриптора события, когда процесс приостановлен (находится в состоянии "сна").

2.11.2. Принципы функционирования OS/2. Ядро OS/2 и взаимодействие задач

Способность запуска двух или более программ одновременно является несомненно наиболее интересной особенностью OS/2. Обычно одна программа работает на переднем плане, взаимодействуя с пользователем. В это же время другие незаметно работают в фоновом режиме. Кроме того, как мы увидим, части одной программы также могут работать одновременно.

Планировщик определяет, какой из нескольких различных задач предоставить время ЦП. Каковы цели планировщика OS/2? Системы пакетной обработки пытаются максимизировать время использования и производительность ЦП, в то время как системы с разделением времени пытаются уменьшить время отклика и обеспечить справедливое распределение времени между различными пользователями. OS/2 пытается оптимизировать время отклика для главного процесса, видимого пользователю, и, в то же время обеспечить адекватное распределение времени между различными фоновыми процессами. Она делает это при помощи так называемого "приоритетного квантующего времени" планировщика.

Приоритетность планировщика заключается в том, что различным задачам присваиваются разные приоритеты: чем выше приоритет задачи, тем чаще она будет работать. Существует три главных класса приоритетов: критический по времени, нормальный и простаивающий. Программы обычно запускаются с нормальным приоритетом. Внутри каждого класса OS/2 различает 32 приоритетных уровня. Фоновые задачи имеют самый высокий приоритет, другие задачи получают более низкий. Операционная система может изменять приоритет

задачи во время выполнения; обычно, если задача не работала в течение некоторого времени, ее приоритет поднимается.

Для большинства приложений программисту не нужно иметь дело с приоритетами. Однако, в некоторых ситуациях бывает необходимо изменить приоритет задачи, и OS/2 предусматривает системную функцию для этой цели.

Планировщик называется приоритетным потому, что он прервет программу до ее окончания для запуска другой. Он называется квантующим время потому, что он делит время на кванты равной длины и распределяет их между различными задачами на основе их приоритета.

В действительности, OS/2 обеспечивает три различных вида многозадачности. Одновременно могут работать различные экранные группы, различные процессы и различные цепочки. Рассмотрим каждый из них.

- *Экранные группы*

Экранные группы (называемые также сеансами) являются формой многозадачности OS/2, доступной не только программисту, но и пользователю. По существу, экранная группа представляет собой виртуальный компьютер, у него есть свое собственное изображение экрана и своя собственная воображаемая клавиатура (или манипулятор типа "мышь", если он подключен). Переключения между экранными группами аналогично переходу от одного компьютера к другому.

Экранная группа, использующая в данный момент экран и клавиатуру, является главным сеансом, остальные называются фоновыми сеансами.

Конечно, различные экранные группы должны разделять ресурсы. Работающие в фоновом режиме не могут (обычно) иметь доступ к экрану, пока пользователь не сделает их главными. Программы, работающие в отдельных экранных группах, не могут одновременно иметь доступ к одному и тому же периферийному устройству или участку памяти. Однако, что зависит от пользователя и программиста, прикладные программы, действующие в различных экранных группах, могут работать так, как если бы они имели целый компьютер. OS/2 дает гарантию того, что они не будут взаимодействовать друг с другом.

Возможно запустить одну экранную группу из другой без вызова программы управления сеансами. Это делается при помощи команды START.

- *Процессы*

Второй формой многозадачности OS/2 являются п р о ц е с с ы. С точки зрения программиста процесс похож на программу, хотя, существуют различия. Один процесс может вызвать выполнение другого; первый процесс называется родительским, а второй ── дочерним. Дочерний процесс — это знакомый аспект программирования в системе Unix. Процессы отличаются от оверлеев тем, что родительский и дочерний процессы могут одновременно находиться в памяти и

могут выполняться параллельно. Существует различие между словами "программа" и "процесс".

Программа — это по существу, .EXE──файл (или, возможно, несколько .EXE ── файлов).

Процесс — это выполнимый код плюс ресурсы, используемые процессом, такие как память, файлы и устройства ввода/вывода. Чтобы понять различия, представьте, что одна и та же программа или участок кода может выполняться двумя или более процессами одновременно. Если код написан должным образом, то нет причин, чтобы один процесс узнал о другом, выполняющем тот же код.

Процесс может быть фоновым. Этот процесс не будет иметь собственного экрана, так что мы не сможем увидеть вывод результатов. Для запуска фонового процесса используется команда DETACH.

Каждый процесс имеет свой собственный уникальный идентифицирующий номер. Номера присваиваются последовательно.

В фоновом процессе могут быть начаты параллельно несколько различных программ, хотя одновременно могут работать и несколько версий одной программы. Фоновый процесс не имеет (обычно) доступа к экрану и клавиатуре. Так что если произойдет ошибка, вы никогда не узнаете об этом. Также важно, чтобы программы, работающие в фоновом процессе, могли останавливать себя. Если по какой-либо причине она зависнет, то она зависнет навсегда, так как пользователь не может остановить ее извне. Даже выход из всей экранной группы не может уничтожить этот процесс.

- *Цепочки*

При реализации третьего вида многозадачности в OS/2 используются цепочки. Программисты на Си могут представлять цепочки как функции; они могут работать одновременно с другими функциями в программе. Цепочки в листинге программы выглядят очень похоже на функции и во многих случаях ведут себя, как они. Когда процесс начинается, он состоит из одной цепочки, затем он может начать новые цепочки, используя вызов API. Эти цепочки могут затем выполняться параллельно с первым.

Цепочки полезны для уменьшения потери времени ЦП во время ожидания ввода/вывода. Одна цепочка предназначается для ожидания ввода/вывода, пока другая выполняет вычисления или еще что-нибудь. Однако, существует множество других применений цепочек.

На рисунке 2.32 показана взаимосвязь экранных групп, процессов и цепочек. Каждая экранная группа может содержать один или больше действующих процессов, и любой процесс может содержать одну или несколько работающих цепочек.

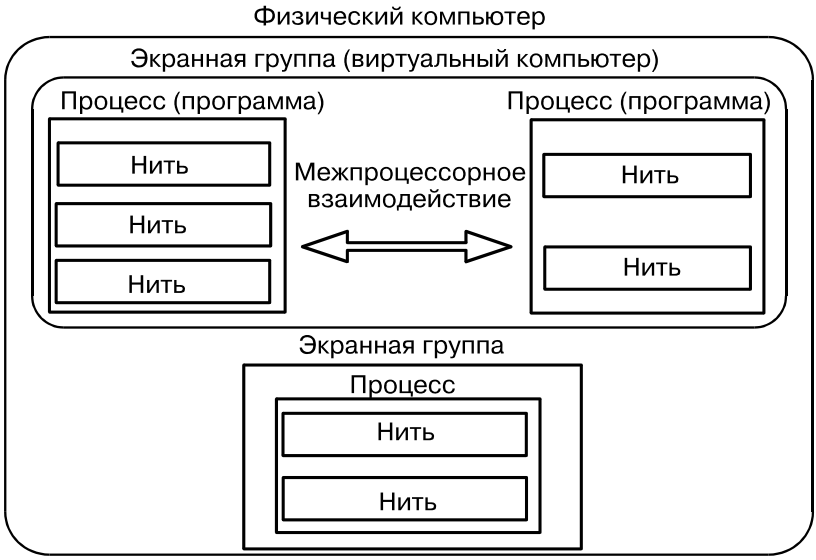

Рис.2.32. Экранные группы, процессы и цепочки.

- *Синхронизация процессов*

При одновременной работе нескольких процессов или цепочек возникает возможность, что два таких объекта (мы будем называть их задачами) начнут взаимодействовать друг с другом. Например, предположим, что две задачи имеют доступ к буферу данных, и одна должна заполнять буфер, перед тем как другая будет его считывать. Если задачи не будут должным образом синхронизированы, первая задача начнет считывать содержимое буфера до того, как другая закончит запись, что может привести к печальным последствиям.

Для предотвращения таких проблем в операционной системе OS/2 реализовано синхронизирующее устройство, называемое семафор. Как видно из названия, семафор останавливает задачу в определенном месте, пока другая задача не заставит семафор показывать, что ресурсы доступны. Существуют два вида семафоров: RAM — семафоры, используемые в основном между цепочками, и системные семафоры, используемые между процессами.

- *Связь между процессами.*

Цепочки, которые похожи на Си — функции, могут связываться почти так же, как и обычные функции; наиболее важный метод — с использованием глобальных

переменных. С другой стороны, отдельные процессы, которые по существу, являются отдельными программами, требуют другого подхода при организации связи. В OS/2 возможны различные связи между процессами, включая общую память, каналы и очереди.

Общая память — это сегмент памяти, который известен двум или более отдельным процессам; каждый из них может считывать и писать туда. Один процесс использует вызов API для создания сегмента общей памяти; остальные процессы используют другие вызовы для нахождения этого сегмента и таким образом получают доступ к нему.

Один процесс записывает данные в канал, где они доступны для чтения другим процессам. Однако, данные, посланные в канал, в действительности остаются в оперативной памяти, что делает общение с каналом более быстрым, чем с файлами.

Очереди — это более сложные структуры, в которых множество сообщений (возможно, достаточно больших) может храниться и различными способами передаваться различным процессам.

- *Управление памятью*

Проблема, присущая MS-DOS — это ограниченный объем оперативной памяти, к которой могут обращаться программы — 640 Кбайт. Эти ограничения вызваны архитектурой микропроцессоров 8088 и 8086, используемых в IBM PC/XT: эти микропроцессоры могут адресовать только 1 Мбайт памяти. В дальнейшем объем доступной оперативной памяти еще уменьшается, так как адреса между 640 Кбайт и 1 Мбайт необходимы для работы видео — адаптера и для BIOS ПЗУ. Процессоры i80286, i80386, i80486, Pentium позволяют адресовать гораздо больший объем памяти, но MS-DOS не способна использовать память более 640Кб.

Были попытки обойти это ограничение MS-DOS на объем доступной оперативной памяти, такие как стандарты памяти EMS и EEMS и их расширения. Они открывают подобие "окон" в дополнительной памяти, но это процесс сложный и неэффективный.

OS/2 ликвидирует 640-килобайтное ограничение памяти. Когда OS/2 управляет памятью программы, данные могут в действительности превышать объем физически доступной памяти.

Программы делятся на сегменты, и, если не хватает пространства в ОЗУ для полной программы, OS/2 "сбрасывает" некоторые сегменты из памяти на диск, где они остаются, пока не понадобятся. Во многих случаях программист может игнорировать предел объема физической памяти в машине.

Когда программа запущена, операционная система заботится о переносе соответствующих сегментов в и из памяти. Когда не хватает памяти для всех

одновременно работающих задач, система сохраняет на диске те сегменты, к которым не было обращений в течение долгого времени. Это называется алгоритм LRU (по "самому давнему использованию").

Аппаратура процессоров 80386 и выше переводит ссылки на виртуальные адреса в физические адреса. Каждый выполнимый процесс снабжается таблицей, которая содержит информацию об используемых сегментах, такую как их расположение и длина. Операционная система изменяет эти таблицы, чтобы описать загружаемый процесс, а аппаратура использует их для перевода ссылок на виртуальные адреса в адреса физической памяти. Если необходимого участка нет в памяти, соответствующий сегмент загружается с диска в память.

Эта деятельность незаметна для программиста, которому (обычно) не нужно знать, находится ли данный сегмент в памяти или на диске.

В языке Си компилятор создает сегменты, необходимые для кода программы, для стека и для переменных, определенных в программе. Однако, программист может создать другие сегменты данных, к которым можно обращаться независимо от программы, которые могут быть общими для различных процессов и которые могут освобождаться, когда необходимость в них отпадет. API OS/2 обеспечивает обширный набор функций для создания, использования и освобождения сегментов.

Версии OS/2 используют возможности процессора 80386 и выше, которые позволяют непосредственно адресовать 4 Гбайт физической памяти и до 64 Тбайт виртуальной памяти (1 Тбайт = 1024 Гбайт).

- *Интерфейс программиста*

Как может программист обращаться к функциям, встроенным в операционную систему? В MS-DOS это в основном связано с методом, называемым "прерывание INT 21", где программа вызывает прерывание номер 21H, которое затем вызывает переход к соответствующей функции, в зависимости от содержимого регистра AX. Эта система была недоступна для языков высокого уровня, пока в них не ввели специальную библиотеку функций для соответствующей загрузки регистров ЦП и обеспечения управления прерываниями. Этот процесс был грубым и неэффективным, так как программист должен был поместить значения в переменные, представляющие системные регистры, а затем библиотечная функция должна была взять эти значения из переменных и поместить их в регистры перед вызовом прерывания 21H .

В противоположность этому OS/2 использует очень ясный принцип доступа к системным функциям. Вместо использования программных прерываний доступ к функциям осуществляется непосредственно.

Параметры нужно теперь помещать не в регистры, а в стек, как это делается в других функциях. Вызовы могут делаться непосредственно из языков высокого уровня и даже проще, чем из языка ассемблера, а процесс проходит быстрее и эффективнее, чем в MS-DOS .

Эти системные функции и путь доступа к ним называется API (интерфейс прикладных программ). Существует более 200 функций API. Они обеспечивают ввод/вывод, контролирующие процессы, управление взаимодействием между процессами и решение многих других задач.

- *Доступ к аппаратуре*

В MS-DOS обычен непосредственный доступ к аппаратному обеспечению без обращения к операционной системе: делаются обращения к ПЗУ, BIOS или посылаются данные непосредственно в порты ввода/вывода. В OS/2 это не так. Так как несколько процессов могут попытаться использовать одно и то же устройство одновременно, операционная система должна контролировать запросы к устройствам, чтобы гарантировать отсутствие взаимодействия между процессами. Эта ситуация показана на рисунке 2.33. OS/2 содержит очень эффективные программы контроля доступа к периферии, так что потери производительности невелики по сравнению с непосредственным доступом.

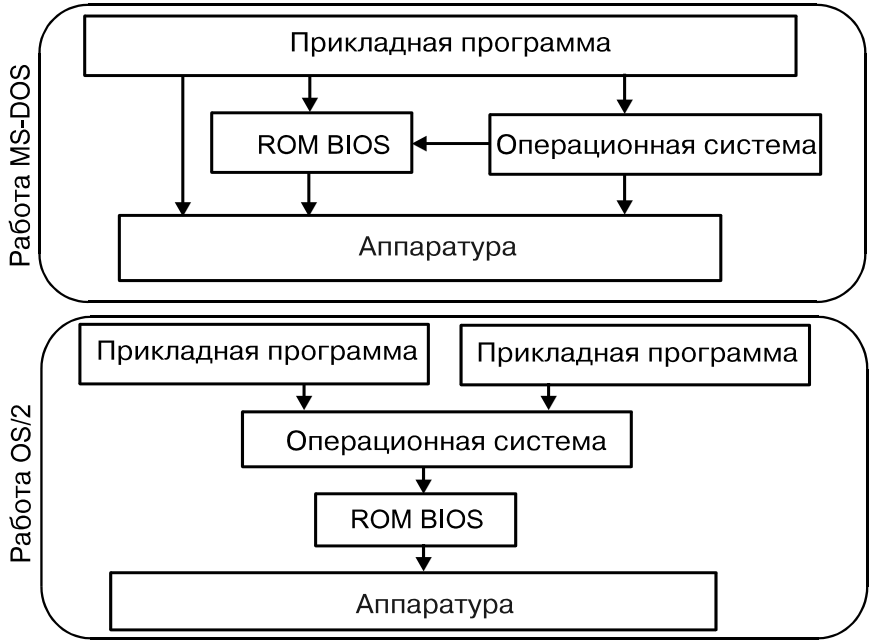

Рис.2.33. Сравнение доступа программ к аппаратуре

После установки на вычислительную машину операционной системы OS/2 возможен запуск программы для MS-DOS и Windows. Обратная совместимость — важный элемент в реализации OS/2. OS/2 обеспечивает для программ MS-DOS блок, который называется по-разному: "блок совместимости", "блок реального режима" или "блок 3.x". Этот блок является в действительности одним из двух режимов работы OS/2, позволяющих одной старой программе MS-DOS работать вместе с несколькими новыми программами, написанными специально для OS/2. Программы MS-DOS работают в реальном режиме (режим процессора 80386, который эмулирует процессоры 8088 и 8086), тогда как программы, работающие в нормальной среде OS/2, называются программами защищенного режима, так как в этом режиме память и ресурсы одной программы защищены от других программ. Большинство программ, работающих под управлением MS-DOS, могут работать в блоке совместимости, но есть, однако, некоторые, которые не могут этого делать.

- *Режим совместимости с MS DOS*

Аналогично программам MS-DOS реализован принцип функционирования программ для Windows в WinOS2 (код Windows адаптированный для работы с OS/2).

- *Семейство API*

Написание программ, работающих как в MS-DOS, так и в защищенном режиме OS/2 (без использования блока совместимости), избавляет от необходимости писать две версии программы, предназначенные для работы в двух разных системах. Разработчики предусмотрели для этой ситуации нечто, называемое "семейством API". Это "семейство" включает четыре возможных среды: MS-DOS, WinOS2, блок совместимости OS/2 и защищенный режим OS/2. Программы семейства API написаны с использованием подмножества функций API OS/2. Это приводит к совместимости между этими средами, но ограничивает возможности программы. Она не может (так как должна работать в MS-DOS) участвовать в многозадачности и использовать взаимодействие между процессами и другие полезные особенности OS/2.

- *Управляющие программы устройств*

В резидентных программах MS-DOS (или TSR — программах) программисту обычно необходимо перехватывать поток символов с клавиатуры до того, как они достигнут главной программы. Таким образом, когда нажата особая ("горячая") клавиша, программа может обратиться к TSR — программе. Подготовка такого перехвата весьма сложна и включает изменение векторов прерываний и выполнение других действий, последствия которых трудно предсказать.

OS/2 систематизировала перехват данных, поступающих от различных устройств ввода/вывода. Программа через функции API просит операционную систему установить системную функцию, называемую "управляющей программой устройств". Эта управляющая программа перехватывает поток символов, идущих от устройства, так что основная программа может управлять ими и предпринимать соответствующие действия.

Много различных программ могут одновременно устанавливать эти управляющие программы, и операционная система гарантирует, что они не будут взаимодействовать друг с другом.

- *Динамическая компоновка*

OS/2 позволяет динамически связывать функции с программой во время работы, а не во время компоновки. Это помогает управлению памятью, так как функции динамической компоновки не нужно располагать в памяти вместе с главной программой. Они могут быть загружены позднее, когда понадобятся, и удалены после использования, освобождая таким образом память. Использование памяти оптимизируется, так как различные программы могут совместно использовать одни и те же функции динамической компоновки.

- *Кольца защиты*

Процессоры 80386, 80486 и Pentium позволяют программам работать в нескольких уровнях защиты. Программы более высоких уровней (с меньшими номерами) имеют доступ к программам более низких уровней (с большими номерами), но не наоборот. Ядро OS/2 занимает высший уровень — кольцо 0, а прикладные программы — низший уровень — кольцо 3. Это проиллюстрировано на рис. 2.34.

Обычно программы работают полностью в кольце 3. Можно написать драйверы устройств так, чтобы они использовали кольцо 2.

- *Организация памяти*

Подход программиста к использованию оперативной памяти в OS/2 и в MS-DOS сильно различается. В MS-DOS обычно предполагается, что программа и ее данные всегда занимают одни и те же адреса памяти, куда они были загружены первоначально, оставляя всю другую память свободной.

Программа таким образом может изменять свой собственный код и осуществлять абсолютную адресацию памяти. В OS/2 программа или ее данные могут быть сброшены на диск и записаны снова в другое место памяти. К тому же участки памяти, не занятые программой, могут содержать другие программы, так что свободной памяти не остается, кроме той, которая специально получена от операционной системы. Таким образом, программист должен "играть по правилам

большего пространства" и делать меньше предположений о том, где объекты находятся в памяти, чем он мог бы это делать в MS-DOS.

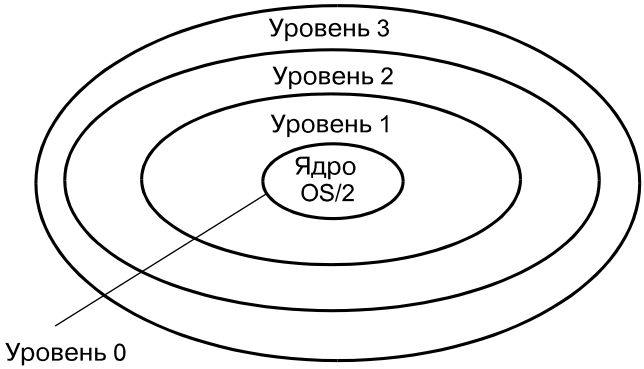

Рис. 2.34. Кольца защиты

- *Обработка ошибок*

Важной частью реализации OS/2 является последовательный и легкий способ возврата сообщений об ошибках из функций в программу. Возвращаемые значения всех функций OS/2 одинаковы: значение 0, если нет ошибки, и положительное число, если ошибка есть. Для всех функций используется один и тот же список номеров ошибок, что упрощает анализ ошибки. Номер ошибки возвращается в регистре AX; для программистов на Си функция сама возвращает номер ошибки, который может быть легко проанализирован программой.

- *Независимые программы*

Одним из преимуществ OS/2 является возможность запуска независимых программ в новой экранной группе (или с помощью программы управления представлениями). Существует много случаев, когда запуск двух или более программ одновременно приносит пользу. Подобные действия обеспечиваются запуском нескольких программ одновременно в различных экранных группах, выбранных с помощью программы управления сеансами или как различных задач программы управления представлениями.

- *Отдельные цепочки*

Использование новых особенностей OS/2 состоит в создании программ с отдельными цепочками. Это позволяет программе выполнять несколько действий одновременно. Например, при запуске программа может считывать данные инициализации с диска и, в то же время печатать сообщение для пользователя. В программе обработки текста, одна цепочка может вычислять, как форматировать страницу, другая — получать текст пользователя с клавиатуры, а третья — ждать доступа к диску.

Каковы преимущества этого вида многозадачности? В целом программа будет работать быстрее, чем раньше. Вместо того, чтобы ждать, пока пользователь закончит печатать сообщение, программа использует одну цепочку для ожидания ввода пользователя и, в это же время выполняет с помощью других цепочек другие задачи, такие как вычисление или ввод/вывод с диска.

Также бывают ситуации, в которых, не достигая преимущества в быстродействии, цепочки облегчают концептуализацию проблемы или написание определенного вида программы. Если цепочка однажды запущена, она поддерживается независимо, что точно соответствует работе с независимыми объектами структур данных.

Различные цепочки сохраняют память. Если много цепочек делают одно и то же одновременно, они могут использовать один код — нет необходимости писать отдельный код для каждого цепочки.

- *Отдельные процессы*

Отдельные процессы применяются, когда одновременно присутствуют две или больше главные задачи. Например, в интегральной программе, чтобы все три задачи работали одновременно. Вместо использования независимых программ, которые должны быть взаимосвязаны, проще использовать дочерние процессы одной управляющей программы. При этом процессы могут координировать свою деятельность и обмениваться данными.

- *Резидентные программы*

В OS/2 программы типа TSR могут выполнять задачи, пока работает главная программа, а не ожидать нажатия "горячей" клавиши. Например, программа — корректор, включенная нажатием "горячей" клавиши, может обращаться к словарю, пока пользователь продолжает печатать текст в редакторе.

В MS-DOS количество TSR — программ, которые можно использовать одновременно, было ограничено доступной памятью. В OS/2 доступен больший объем памяти и, следовательно, большее число более мощных TSR — программ может быть одновременно загружено в ОЗУ. Также исключается проблема взаимодействия между TSR— программами. Система трактует TSR — программы

как любую другую программу, работающую в многозадачной среде, а управляющие программы устройств систематизируют использование "горячих" клавиш.

2.11.3. Сетевая операционная система Windows NT

Windows NT (далее именуемая NT) — многозадачная среда для распределенных систем обработки информации, функционирующая как на высокопроизводительных хост — ЭВМ, так и на отдельных рабочих станциях и обладающая встроенными средствами для работы с сетями. Любая из рабочих станций в сети под управлением NT может одновременно выступать в роли сервера файлов или печати, являться членом кластера процессоров, выполняющих общую информационную задачу, работать в режиме однопользовательского рабочего места.

В NT реализованы многие достаточно известные концепции, являвшиеся до сих пор спецификой лишь больших ЭВМ, и при этом использованы новейшие информационные технологии, что отражено в названии системы. "Большие возможности для дешевой ЭВМ" — так можно сформулировать девиз ОС.

* *Основные особенности системы*

При запуске программы на исполнение ОС порождает процесс. Процессом называется единица управления и потребления ресурсов системы. Как уже упоминалось, NT является многозадачной ОС. Это означает, что процесс не может монопольно захватить вычислительные ресурсы компьютера — их распределение является функцией ОС. Например, ОС централизованно выделяет каждому из процессов память, выйти за границы которой они самостоятельно не могут. Этим исключается возможность наложения или перекрытия процессов.

В памяти компьютера могут одновременно находиться несколько процессов, многозадачная ОС должна предусматривать процедуру выбора одного из них и запуска его на исполнение. В отличие от Windows, где переключение процессов происходит по инициативе пользователя, в NT их смена осуществляется принудительно через определенные промежутки времени.

* *Компоненты NT*

Все процессы в NT подразделяются на системные и пользовательские. Системные процессы имеют непосредственный доступ к аппаратным ресурсам и делятся на два класса:

- исполнительный модуль, обеспечивающий базовые операции ОС, необходимые для работы остальных ее компонентов (процессов);
- расширения привилегированного режима — процессы, тесно взаимодействующие с аппаратнозависимыми компонентами ОС.

Пользовательские процессы, требующие поддержки средств системного уровня, также подразделяются на два класса:

- защищенные подсистемы различных пользовательских интерфейсов;
- прикладные программы пользователей.

Разделение процессов на типы и классы тесно связано с аппаратными возможностями микропроцессора, называемыми кольцами защиты.

* *Кольца защиты*

Пользователям UNIX известны системный и пользовательский режимы выполнения процесса. В пользовательском режиме исполняется ограниченный набор аппаратных инструкций микропроцессора, тогда как в системном могут быть использованы любые инструкции. Переход в системный режим происходит в результате выполнения системного вызова.

Современные микропроцессоры имеют более двух уровней привилегий исполнения инструкций (например, в микропроцессоре Intel 80x86 их четыре). Эти уровни называются кольцами защиты. Разработчики NT разместили в нулевом кольце защиты только самую важную часть кода ОС — исполнительный модуль. Остальные компоненты ОС обладают меньшими привилегиями. Это отличает NT, скажем, от NetWare 3.x, где создаваемые пользователем загружаемые модули (NLM) могут исполняться даже в нулевом кольце, нарушая при этом защиту данных.

* *Исполнительный модуль*

Исполнительный модуль (Executive) осуществляет базовые операции нижнего уровня: управление процессами и памятью, диспетчеризацию ресурсов системы, обслуживание прерываний и обеспечение межпроцессорной связи для остальных компонентов ОС и для пользовательских задач. Каждому процессу в ОС присваивается определенный приоритет. Задачей исполнительного модуля является выбор процесса с наивысшим приоритетом и его запуск.

* *Микроядро*

Одним из достоинств UNIX традиционно считается компактное ядро системы, которое упрощает процесс переноса на новую компьютерную платформу. Во многих современных ОС ядро выросло настолько, что потребовалось выделить в нем микроядро, содержащее аппаратно──зависимую часть кода.

Одно из технологических новшеств NT — объектно - ориентированная структура микроядра, объектами которого являются ресурсы. Это означает, что микроядро взаимодействует с любыми ресурсами при помощи единого набора низкоуровневых вызовов. Для сравнения : в традиционных системах ядро при управлении драйверами использовало специфические для каждого из них команды.

В новой ОС ядро и программные модули, управляющие отдельными ресурсами, имеют общий канал обмена информацией, использующий механизм

сообщений. Получив сообщение, каждый модуль управления ресурсом выполняет специфические действия, например, вывод на терминал или в файл.

В результате удалось достичь высокой эффективности работы, централизации управления и гибкости функционирования. Для каждого модуля используется документированный программный интерфейс, что упрощает разработку программных средств. В большинстве случаев разработчики стремились обеспечить максимальную совместимость этих интерфейсов с уже существующими.

* *Расширения привилегированного режима*

Расширения привилегированного режима обеспечивают высокоуровневый сервис для прочих компонентов системы. Процессы этого класса управляют ресурсами ОС и включают менеджер памяти, планировщик процессов, монитор безопасности, драйверы устройств, файловую систему и сетевые службы.

Реализация ОС в виде совокупности процессов облегчает не только модификацию существующих модулей, но и дальнейшее расширение функций ОС. Например, управление сетевыми службами встроено как расширение привилегированного режима. В UNIX для этой цели обычно используются так называемые "процессы — демоны", выполняемые в пользовательском режиме и, следовательно, менее эффективные и защищенные.

* *Потоковые драйверы*

Динамически загружаемые драйверы внешних устройств имеют двухуровневую структуру. Драйверы верхнего уровня являются аппаратно-независимыми, в то время как нижний уровень, организованный в виде специальных библиотек, привязан к конкретным устройствам.

Для написания нового драйвера обычно необходимо лишь внести небольшие изменения в модуль нижнего уровня, причем драйверы пишутся не на языке Ассемблера, а на Си. Драйверы устройств могут динамически загружаться в процессе работы системы.

Драйверы реализованы как потоковые (Streams). Это позволяет направить информацию с выхода одного драйвера на вход другого, связав их в цепочки и наладив совместную последовательную работу.

Можно организовать программные мультиплексоры, направив информацию с выхода одного драйвера на вход нескольких драйверов другого уровня. Потоковые драйверы широко применяются и в UNIX для организации работы с сетями и терминалами.

* *Файловая система*

Файловая система построена по принципу драйвера высокого уровня, что позволяет в рамках одной ОС поддерживать разные способы организации файлов на внешних устройствах, например, CD-ROM; DOS-совместимую таблицу расположения файлов FAT; совместимую с OS/2 высокоскоростную файловую систему HPFS и собственную файловую систему NTFS.

Файловая система NTFS позволяет работать со сверхбольшими информационными томами и имеет средства исправления ошибок и замены дефектных секторов. Специальный механизм отслеживает и фиксирует все действия, выполняемые над накопителями, поэтому в случае сбоя целостность информации восстанавливается автоматически.

* *Модель клиент — сервер*

Принципиально важным в NT является то, что в основу взаимодействия прикладной программы и ОС заложена модель клиент — сервер.

Все обращения пользовательской программы (клиента) к ОС переадресуются ядром для обработки специальной программе (серверу), называемой защищенной подсистемой. При этом используется механизм, аналогичный вызову удаленной процедуры (Remote Procedure Call — RPC), что позволяет легко перейти от взаимодействия между процессами в пределах одной машины к распределенной системе.

Разрабатывая свою защищенную подсистему, программист имеет все преимущества пользовательского режима — стандартные компиляторы и мощные отладчики. Полностью отлаженная программа запускается в защищенном режиме.

* *Защищенные подсистемы*

Важным компонентом NT являются защищенные пользовательские подсистемы, выполняемые в непривилегированном пользовательском режиме. В виде таких подсистем реализованы интерфейсы прикладных программ (Application Programming Interface — API), причем одновременно поддерживаются три типа интерфейсов: собственный 32-разрядный, совместимый с 16-разрядным интерфейсом DOS и Windows 3.x; интерфейс для программ OS/2; POSIX — интерфейс UNIX — программ.

Благодаря наличию нескольких интерфейсов в NT можно запускать программы, разработанные для разных ОС. Интерфейс в стандарте POSIX обеспечивает на уровне исходных текстов совместимость со многими приложениями, работающими под UNIX. Аналогичная защищенная подсистема позволяет запускать прикладные программы, созданные для OS/2, например, SQL Server.

Новыми по сравнению с Windows, средствами управления ресурсами в NT являются семафоры, разделяемая память, именованные программные каналы (named pipes). Среди дополнительных графических возможностей — поддержка

кривых Безье и различных геометрических преобразований, средства аппаратнонезависимого управления цветом.

* *Организация памяти*

Модель памяти в NT использует 32-разрядную линейную адресацию (без сегментов), разделение и защиту как пользовательских, так и системных процессов. Применение страничной подкачки файлов (paging) позволяет загрузить на исполнение файл, размер которого превосходит объем ОЗУ компьютера. Процессы, находящиеся в ОЗУ, но не выполняющиеся в данный момент, могут быть временно перемещены на диск в область подкачки, в качестве которой используется специальный файл PAGEFILE.SYS. Новшеством является возможность размещения этого файла на нескольких дисках.

Один из недостатков традиционных ОС — необходимость избыточного копирования информационных массивов в различных буферах операционной системы. В NT исполнительный модуль, файловая система и драйверы устройств используют разделяемый пул программных буферов, что позволяет обойтись без дополнительной пересылки данных в процессе ввода/вывода.

* *Защита информации*

Встроенный в ядро NT монитор защиты (Security Monitor) обеспечивает единый механизм проверки прав доступа к любым ресурсам системы. Тем самым исключается возможность несанкционированного доступа к информации и обеспечивается высокая степень ее защиты.

NT не только контролирует доступ к любым ресурсам, но и протоколирует в различных журналах все действия администратора и пользователей, направленные на нарушение защиты.

* *Группа пользователей*

В NT имеется учетная база, содержащая имена пользователей и рабочих групп (к которым принадлежит тот или иной пользователь), а также пароли. База ведется специальным администратором. Перед началом работы системе необходимо сообщить свое имя и пароль, которые регистрируются в учетной базе, в противном случае доступ в систему не предоставляется.

Очень важна для NT концепция домена. Домен — это объединение нескольких компьютеров, с общей учетной базой пользователей и единой стратегией обеспечения безопасности. Пользователь может зарегистрироваться на любом из компьютеров домена.

Все пользователи разбиты на группы, различающиеся привилегиями доступа к системе. Наиболее широкими полномочиями обладают группы администраторов. Однако, принадлежность к этой группе не дает права доступа к любой информации

на диске, как в UNIX. Если пользователь снял право доступа администратора к своим файлам, то последний не сможет их использовать.

Учетная база пользователей находится в ведении администратора домена.

В UNIX право пользователя на запуск той или иной программы определяется атрибутами доступа к соответствующему исполняемому файлу. В NT эта концепция расширена за счет разделения программ на персональные (personal) и общего пользования (common). Персональная программа появляется на экране в виде пиктограммы только у тех пользователей, которые имеют к ней доступ, для остальных она остается невидимой.

* *Сетевые средства*

Составной частью продукта Windows NT Advanced Server является известный пользователям Mickosoft Lan Manager. NT позволяет с помощью разнообразных сетевых транспортных протоколов связываться с приложениями, работающими в средах DOS, Windows, OS/2 и Mac, а через LAN Manager/X — с UNIX.

* *Требования к аппаратуре*

ОС выпускается в двух версиях (собственно Windows NT и Windows Advanced Server для серверов доменов) и может работать на компьютерах с микропроцессорами Intel 80x86, RISK — процессорами R4000 и R4400, и в любом случае, ОС необходима машина с внушительной конфигурацией аппаратных средств. Необходимо иметь 12-16 Мбайт оперативной памяти и 80-100 Мбайт дискового пространства : 30 Мбайт занимает собственно ОС, 20 Мбайт рекомендуется зарезервировать для файла подкачки страниц (его размер должен в 1,5 раза превышать емкость ОЗУ), средства разработки и отладки программ (SDK) занимает 35 Мбайт. У RISK ── компьютеров эти потребности в полтора раза выше. Кроме того, система поставляется только на CD-ROM, так что необходим соответствующий накопитель.