ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
Ульяновский государственный технический университет

Операционные системы:
Управление памятью

Составитель: ст. преп. П. С. Макаров

Ульяновск
2008

УДК 681.3(076)
ББК 32.97я7
О - 11

Одобрено секцией методических пособий научно-методического совета университета

Операционные системы: управление памятью : методические
О - 11 рекомендации / сост. : П. С. Макаров. – Ульяновск : УлГТУ, 2008. – 68 с.

В пособии представлена тема «Управление памятью» учебного курса «Операционные системы».

Курс предназначен и читается на факультете информационных систем и технологий УлГТУ для студентов 2 курса специальности 220100 «Электронные машины, комплексы, системы и сети» как основной курс.

Материалы курса представляют собой иллюстрации презентации с комментариями.

УДК 681.3(076)
ББК 32.97я7

ВВЕДЕНИЕ

Характерной особенностью нашего времени являются процессы информатизации фактически всех сфер человеческой деятельности, которые интенсивно развиваются. Развиваются и средства проведения занятий, методика проведения лекций. Все чаще и больше лекционная часть курса проходит с применением проекционного оборудования, когда студентам демонстрируются слайды, и преподаватель дает объяснения, используя данный материал.

В курсе «Операционные системы» также используются презентационные технологии. Именно поэтому в данном пособии выбран схожий подход: читателю предъявляется рисунок – слайд лекции – и даются комментарии к слайду. По схожему принципу построены учебные пособия на авторизированных учебных курсах компании Microsoft Windows.

В данной части курса читатель ознакомится с темой «Управление памятью операционных систем». *Операционная система* – это интерфейс между аппаратной частью и прикладными программами, с одной стороны, и пользователем ЭВМ – с другой. Это наиболее важная функция любого компьютера.

В начале происходит знакомство с понятием операционной системы в целом. Рассматривается эволюция операционных систем, их разновидности, классификация, назначение, а также их характеристики. Далее раскрывается тема «Память операционных систем». Подробно рассмотрена схема распределения памяти, а также типы адресации. Дальше читатель знакомиться с различными алгоритмами замещения памяти и уже потом – с особенностями Кэш-памяти.

Результатом изучения данной темы является ознакомление с операционными системами и их особенностями, знание сопутствующих понятий, а также умение управлять памятью ОС.

Эволюция операционных систем

Так как операционные системы появились и развивались в процессе конструирования компьютеров, то эти события тесно связаны.

Первое поколение (1945-1955)	
Электронные лампы и коммутационные панели	Последовательная обработка данных
Второе поколение (1955-1965)	
Транзисторы	Простые пакетные системы
Третье поколение (1965-1980)	
Интегральные схемы	Многозадачные пакетные системы
Четвертое поколение (с 1980 года по наши дни)	
Персональные компьютеры	Системы, работающие в режиме разделения времени

Рис. 1

Первый период (1945 – 1955)

Известно, что компьютер был изобретен английским математиком Чарльзом Бэбиджем в конце восемнадцатого века. Его «аналитическая машина» так и не смогла по-настоящему заработать, потому что технологии того времени не удовлетворяли требованиям по изготовлению деталей точной механики, которые были необходимы для вычислительной техники. Известно также, что этот компьютер не имел операционной системы.

Некоторый прогресс в создании цифровых вычислительных машин произошел после второй мировой войны. В середине 40-х были созданы первые ламповые вычислительные устройства. В то время одна и та же группа людей участвовала и в проектировании, и в эксплуатации, и в программировании вычислительной машины. Это была скорее научно-исследовательская работа в области вычислительной техники, а не использование компьютеров в качестве инструмента решения каких-либо практических задач из других прикладных областей. Программирование осуществлялось исключительно на машинном языке. Об операционных системах не было и речи, все задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Не было никакого другого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

Второй период (1955 – 1965)

С середины 50-х годов начался новый период в развитии вычислительной техники, связанный с появлением новой технической базы – *полупроводниковых элементов*. Компьютеры второго поколения стали более надежными, теперь они смогли непрерывно работать настолько долго, чтобы на них можно было возложить выполнение действительно практически важных задач. Именно в этот период произошло разделение персонала на программистов и операторов, эксплуатационщиков и разработчиков вычислительных машин.

В эти годы появились первые алгоритмические языки, а следовательно и первые системные программы – компиляторы. Стоимость процессорного времени возросла, что потребовало уменьшения непроизводительных затрат времени между запусками программ. Появились первые системы пакетной обработки, которые просто автоматизировали запуск одной программы за другой и тем самым увеличивали коэффициент загрузки процессора. Системы пакетной обработки явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило в виде колоды перфокарт, получила название *пакета заданий*.

Третий период (1965 – 1980)

Следующий важный период развития вычислительных машин относится к 1965-1980 годам. В это время в технической базе произошел переход от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам, что дало гораздо большие возможности новому, третьему поколению компьютеров.

Для этого периода характерно также создание семейств программно-совместимых машин. Первым семейством программно-совместимых машин, построенных на интегральных микросхемах, явилась серия машин IBM/360, построенное в начале 60-х годов. Это семейство значительно превосходило машины второго поколения по критерию цена/производительность. Вскоре идея программно-совместимых машин стала общепризнанной.

Программная совместимость требовала и совместимости операционных систем. Такие операционные системы должны были бы работать и на больших, и на малых вычислительных системах, с большим и с малым количеством разнообразной периферии, в коммерческой области и в области научных исследований. Операционные системы, построенные с намерением удовлетворить всем этим противоречивым требованиям, оказались чрезвычайно сложными «монстрами». Они состояли из многих миллионов ассемблерных строк, написанных тысячами программистов, и содержали

тысячи ошибок, вызывающих нескончаемый поток исправлений. В каждой новой версии операционной системы исправлялись одни ошибки и вносились другие.

Однако, несмотря на необозримые размеры и множество проблем, OS/360 и другие ей подобные операционные системы машин третьего поколения действительно удовлетворяли большинству требований потребителей. Важнейшим достижением ОС данного поколения явилась реализация мультипрограммирования. *Мультипрограммирование* – это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ. Пока одна программа выполняет операцию ввода-вывода, процессор не простаивает, как это происходило при последовательном выполнении программ (однопрограммный режим), а выполняет другую программу (многопрограммный режим). При этом каждая программа загружается в свой участок оперативной памяти, называемый разделом.

Другое нововведение – спулинг (spooling). *Спулинг* в то время определялся как способ организации вычислительного процесса, в соответствии с которым задания считывались с перфокарт на диск в том темпе, в котором они появлялись в помещении вычислительного центра, а затем, когда очередное задание завершалось, новое задание с диска загружалось в освободившийся раздел.

Наряду с мультипрограммной реализацией систем пакетной обработки появился новый тип ОС – *системы разделения времени*. Вариант мультипрограммирования, применяемый в системах разделения времени, нацелен на создание для каждого отдельного пользователя иллюзии единоличного использования вычислительной машины.

Четвертый период (1980 – настоящее время)

Следующий период в эволюции операционных систем связан с появлением больших интегральных схем (БИС). В эти годы произошло резкое возрастание степени интеграции и удешевление микросхем. Компьютер стал доступен отдельному человеку, и наступила эра персональных компьютеров. С точки зрения архитектуры персональные компьютеры ничем не отличались от класса миникомпьютеров типа PDP-11, но вот цена у них существенно отличалась. Если миникомпьютер дал возможность иметь собственную вычислительную машину отделу предприятия или университету, то персональный компьютер сделал это возможным для отдельного человека.

Компьютеры стали широко использоваться неспециалистами, что потребовало разработки «дружественного» программного обеспечения, это положило конец кастовости программистов.

На рынке операционных систем доминировали две системы: MS-DOS и UNIX. Однопрограммная однопользовательская ОС MS-DOS широко использовалась для компьютеров, построенных на базе микропроцессоров Intel 8088, а затем 80286, 80386 и 80486. Мультипрограммная

многопользовательская ОС UNIX доминировала в среде «не-интеловских» компьютеров, особенно построенных на базе высокопроизводительных RISC-процессоров.

В середине 80-х стали бурно развиваться сети персональных компьютеров, работающие под управлением сетевых или распределенных ОС.

В сетевых ОС пользователи должны быть осведомлены о наличии других компьютеров и должны делать логический вход в другой компьютер, чтобы воспользоваться его ресурсами, преимущественно файлами. Каждая машина в сети выполняет свою собственную локальную операционную систему, отличающуюся от ОС автономного компьютера наличием дополнительных средств, позволяющих компьютеру работать в сети. Сетевая ОС не имеет фундаментальных отличий от ОС однопроцессорного компьютера. Она обязательно содержит программную поддержку для сетевых интерфейсных устройств (драйвер сетевого адаптера), а также средства для удаленного входа в другие компьютеры сети и средства доступа к удаленным файлам, однако эти дополнения существенно не меняют структуру самой операционной системы.

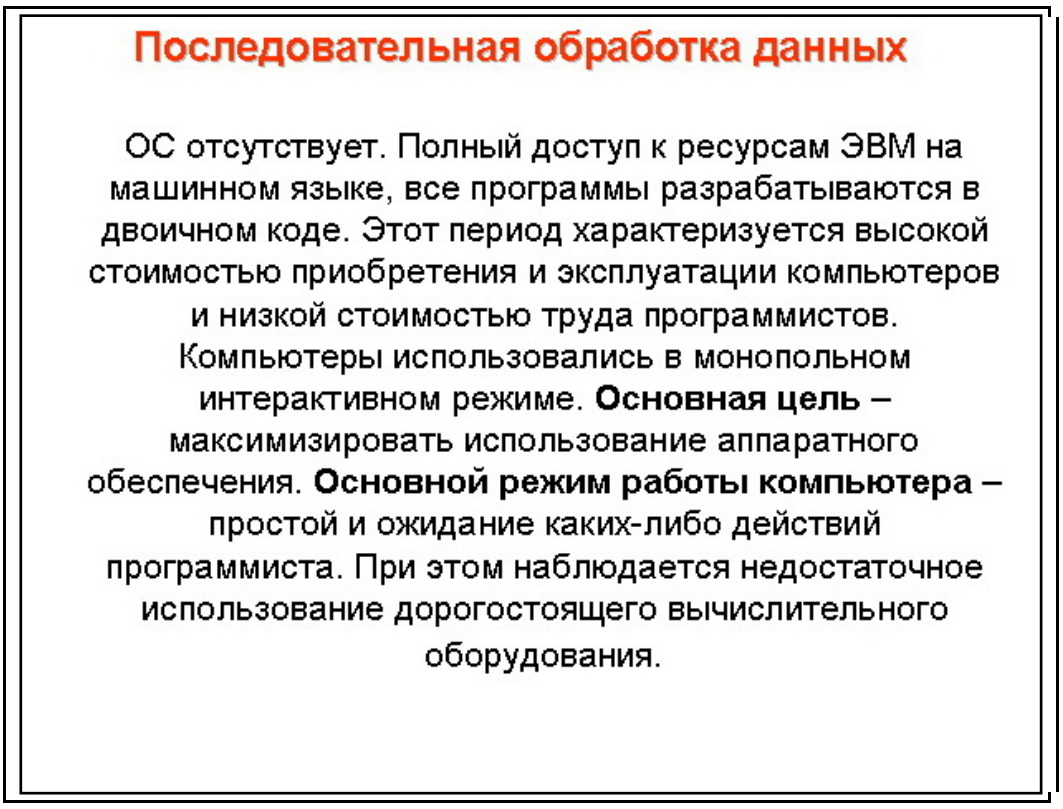

Рис. 2

ОС как система управления ресурсами

Идея о том, что ОС прежде всего система, обеспечивающая удобный интерфейс пользователям, соответствует рассмотрению сверху вниз. Другой взгляд, снизу вверх, дает представление об ОС как о некотором механизме, управляющем всеми частями сложной системы. Современные вычислительные системы состоят из процессоров, памяти, таймеров, дисков, накопителей на

магнитных лентах, сетевой коммуникационной аппаратуры, принтеров и других устройств. В соответствии со вторым подходом функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:

- планирование ресурса, а то есть определение, кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;
- отслеживание состояния ресурса, а то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов – какое количество ресурса уже распределено, а какое свободно.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, что в конечном счете и определяет их облик в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Так, например, алгоритм управления процессором в значительной степени определяет, является ли ОС системой разделения времени, системой пакетной обработки или системой реального времени.

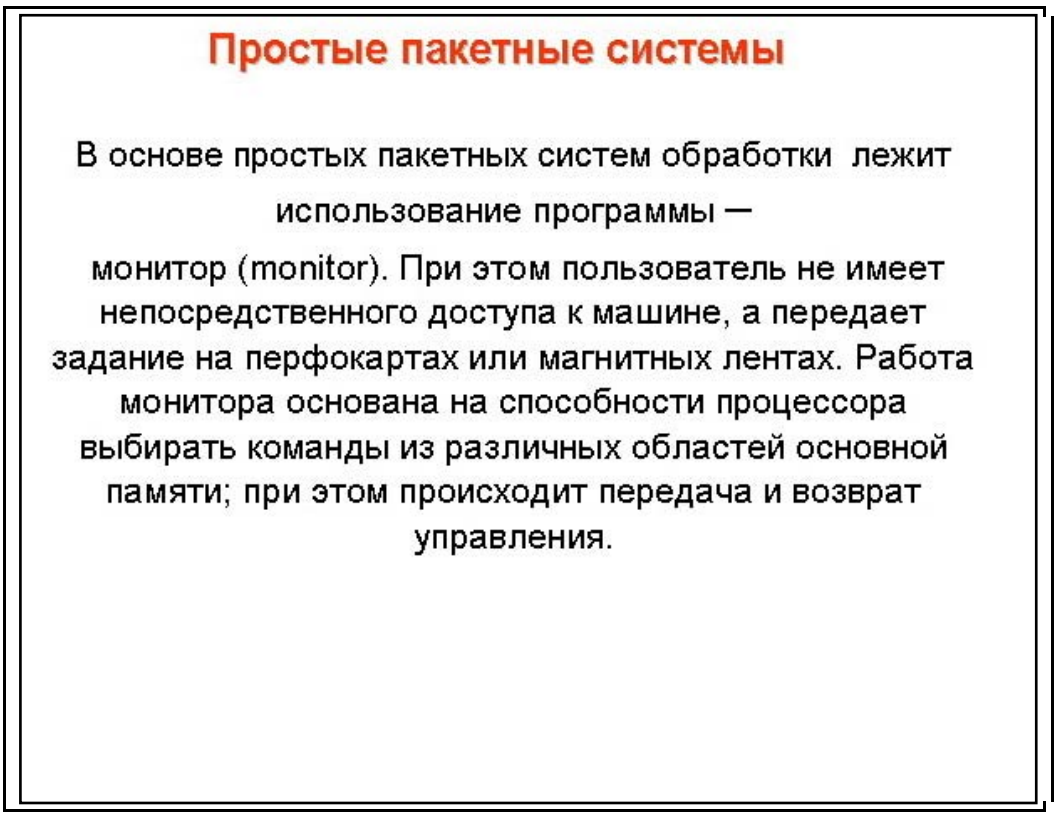

Рис. 3

ОС как расширенная машина

Использование большинства компьютеров на уровне машинного языка затруднительно, особенно это касается ввода-вывода. Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров таких, как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающих наличие и типы ошибок, которые, очевидно, надо анализировать. Даже если не входить в курс реальных проблем программирования ввода-вывода, ясно, что среди программистов нашлось бы немного желающих непосредственно заниматься программированием этих операций. При работе с диском программисту-пользователю достаточно представлять его в виде некоторого набора файлов, каждый из которых имеет имя. Работа с файлом заключается в его открытии, выполнении чтения или записи, а затем в закрытии файла. Вопросы, подобные таким, как следует ли при записи использовать усовершенствованную частотную модуляцию или в каком состоянии сейчас находится двигатель механизма перемещения считывающих головок, не должны волновать пользователя. Программа, которая скрывает от программиста все реалии аппаратуры и предоставляет возможность простого, удобного просмотра указанных файлов, чтения или записи – это, конечно, операционная система. Точно так же, как ОС ограждает программистов от аппаратуры дискового накопителя и предоставляет ему простой файловый интерфейс, операционная система берет на себя все малоприятные дела, связанные с обработкой прерываний, управлением таймерами и оперативной памятью, а также другие низкоуровневые проблемы. В каждом случае та абстрактная, воображаемая машина, с которой, благодаря операционной системе, теперь может иметь дело пользователь, гораздо проще и удобнее в обращении, чем реальная аппаратура, лежащая в основе этой абстрактной машины.

С этой точки зрения функцией ОС является предоставление пользователю некоторой расширенной или виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

Рис. 4

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, то есть решение максимального числа задач в единицу времени. Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие отличающиеся требования к ресурсам так, чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины; так, например, в мультипрограммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в таких ОС невозможно гарантировать выполнение того или иного задания в течение определенного периода времени. В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит только в случае, если активная задача сама отказывается от процессора, например, из-за необходимости выполнить операцию ввода-вывода. Поэтому одна задача может надолго занять процессор, что делает невозможным выполнение

интерактивных задач. Таким образом, взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат. Очевидно, что такой порядок снижает эффективность работы пользователя.

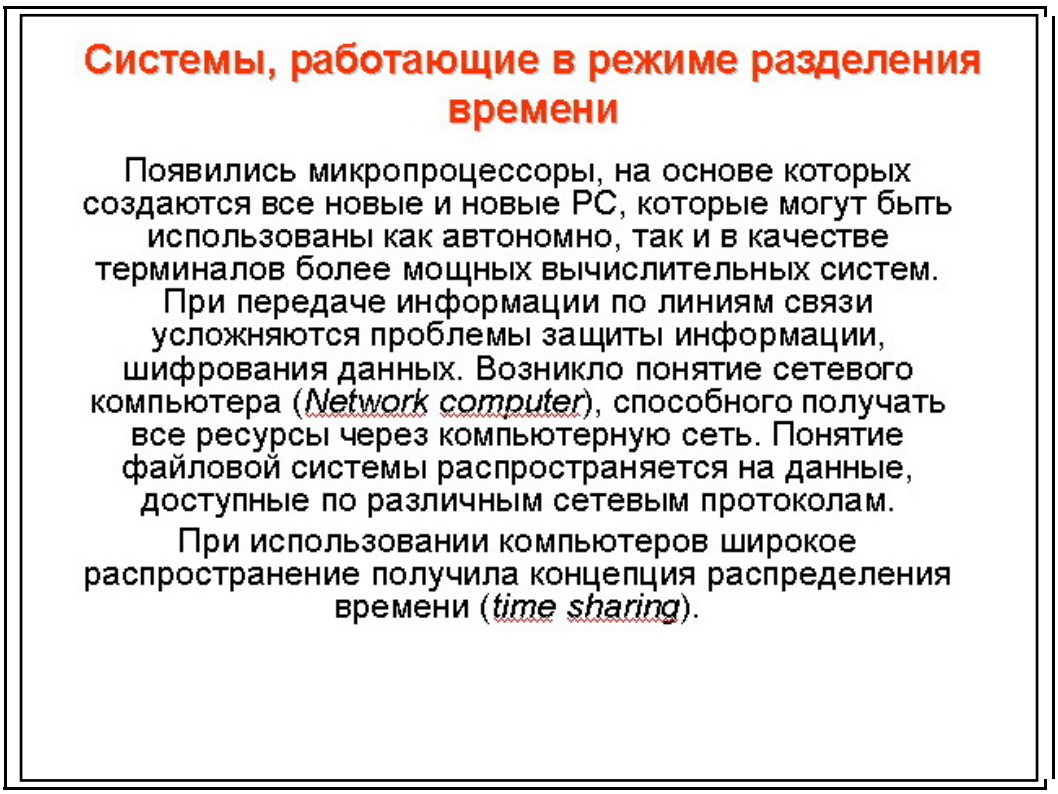

Рис. 5

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая «выгодна» системе, и, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу. Критерием эффективности

систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

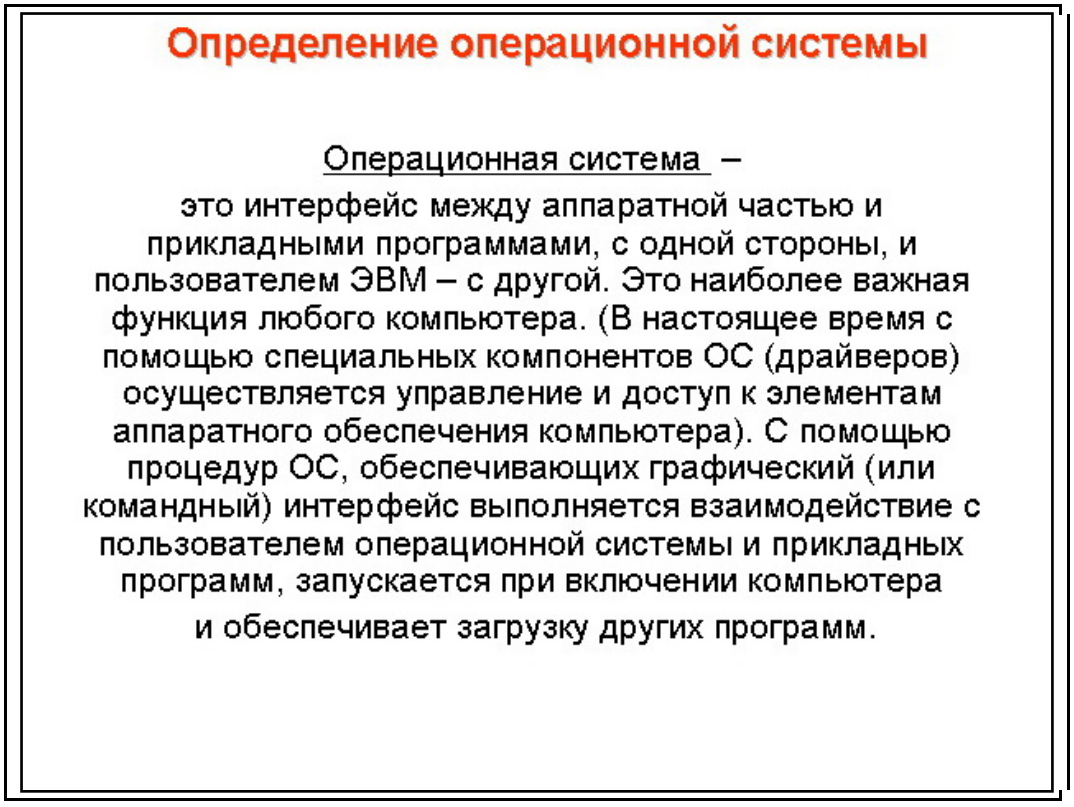

Рис. 6

Определение операционной системы

Операционная система в наибольшей степени определяет облик всей вычислительной системы в целом. Несмотря на это, пользователи, активно использующие вычислительную технику, зачастую испытывают затруднения при попытке дать определение операционной системе. Частично это связано с тем, что ОС выполняет две по существу малосвязанные функции: обеспечение пользователю-программисту удобств посредством предоставления для него расширенной машины и повышение эффективности использования компьютера путем рационального управления его ресурсами.

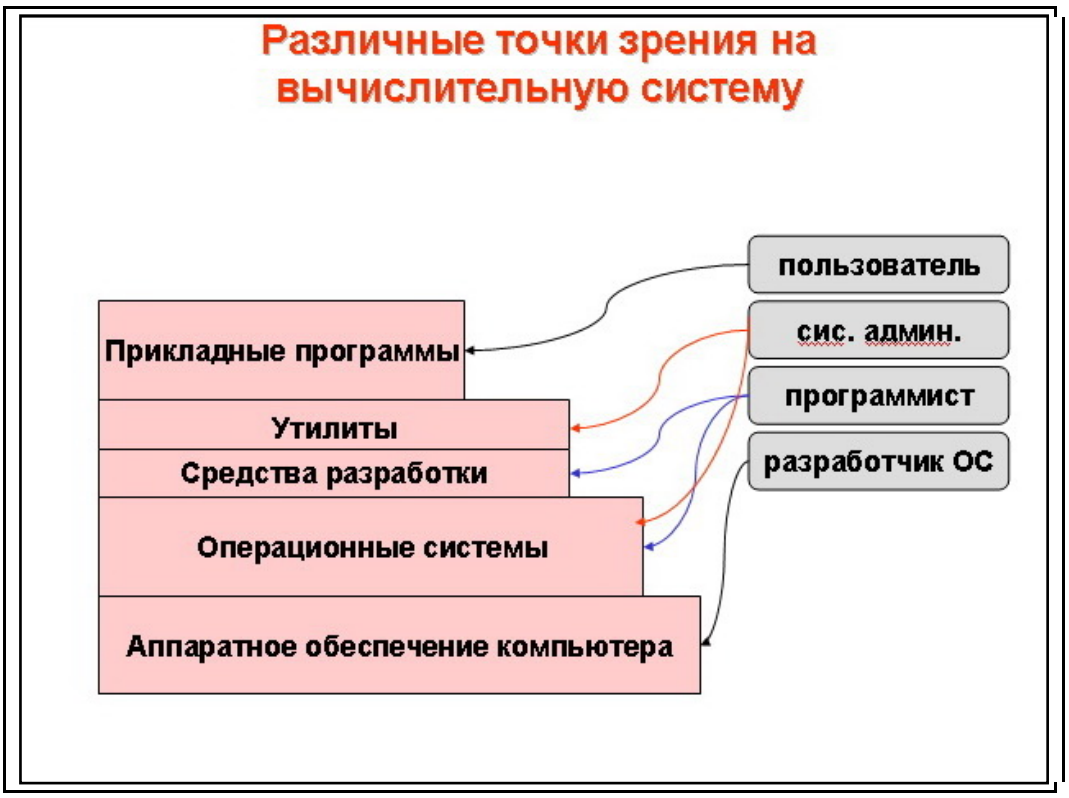

Рис. 7

Для того, чтобы успешно решать свои задачи, современный пользователь или даже прикладной программист может обойтись без досконального знания аппаратного устройства компьютера. Ему не обязательно быть в курсе того, как функционируют различные электронные блоки и электромеханические узлы компьютера. Кроме того, пользователь может не знать системы команд процессора. Пользователь-программист привык иметь дело с мощными высокоуровневыми функциями, которые ему предоставляет операционная система.

Операционная система избавляет программистов не только от необходимости напрямую работать с аппаратурой дискового накопителя, предоставляя им простой файловый интерфейс, но и берет на себя другие рутинные операции, связанные с управлением другими аппаратными устройствами компьютера: физической памятью, таймерами, принтерами и т. д.

Прикладному программисту возможности ОС доступны в виде набора функций, составляющих интерфейс прикладного программирования (API).

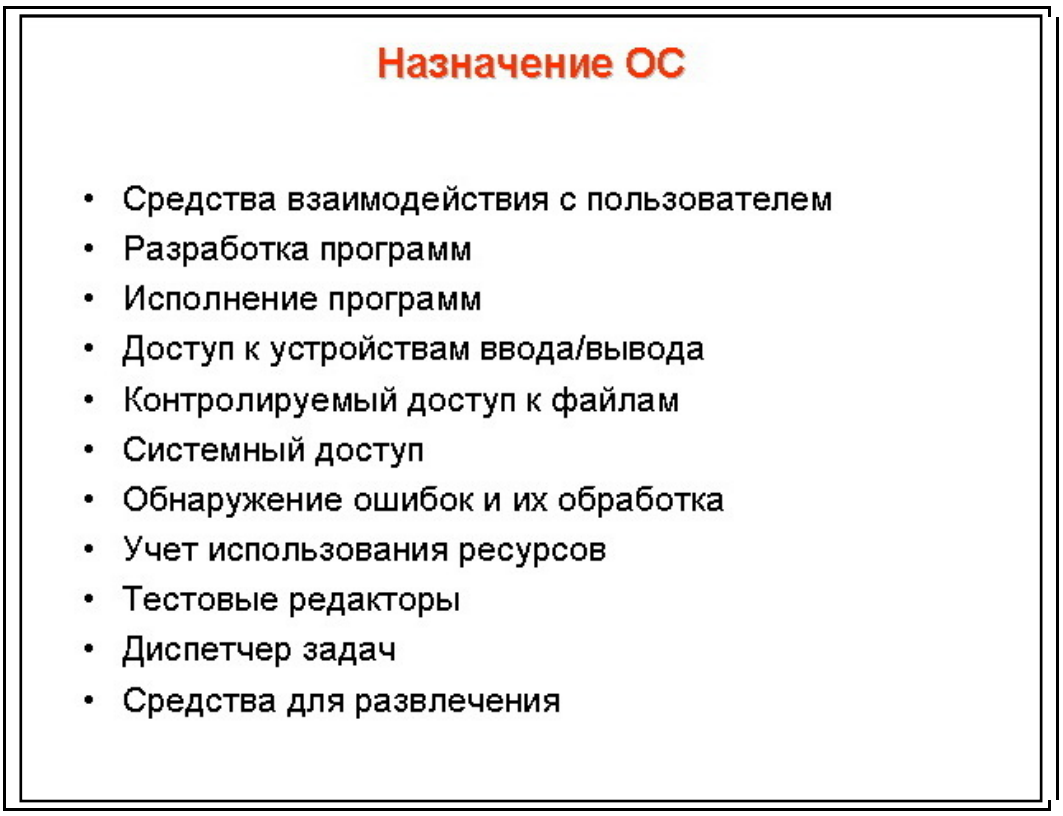

Рис. 8

Особенности аппаратных платформ

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. В любом случае специфика аппаратных средств, как правило, отражается на специфике операционных систем.

Очевидно, что ОС большой машины является более сложной и функциональной, чем ОС персонального компьютера. Так в ОС больших машин функции по планированию потока выполняемых задач, очевидно, реализуются путем использования сложных приоритетных дисциплин и требуют большей вычислительной мощности, чем в ОС персональных компьютеров. Аналогично обстоит дело и с другими функциями.

Сетевая ОС имеет в своем составе средства передачи сообщений между компьютерами по линиям связи, которые совершенно не нужны в автономной ОС. На основе этих сообщений сетевая ОС поддерживает разделение ресурсов компьютера между удаленными пользователями, подключенными к сети. Для поддержания функций передачи сообщений сетевые ОС содержат специальные программные компоненты, реализующие популярные коммуникационные протоколы, такие как IP, IPX, Ethernet и другие.

Многопроцессорные системы требуют от операционной системы особой организации, с помощью которой сама операционная система, а также

поддерживаемые ею приложения могли бы выполняться параллельно отдельными процессорами системы. Параллельная работа отдельных частей ОС создает дополнительные проблемы для разработчиков ОС, так как в этом случае гораздо сложнее обеспечить согласованный доступ отдельных процессов к общим системным таблицам, исключить эффект гонок и прочие нежелательные последствия асинхронного выполнения работ.

Другие требования предъявляются к операционным системам кластеров. *Кластер* – слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений и представляющихся пользователю единой системой. Наряду со специальной аппаратурой, для функционирования кластерных систем необходима и программная поддержка со стороны операционной системы, которая сводится, в основном, к синхронизации доступа к разделяемым ресурсам, обнаружению отказов и динамической реконфигурации системы. Одной из первых разработок в области кластерных технологий были решения компании Digital Equipment на базе компьютеров VAX. Недавно этой компанией заключено соглашение с корпорацией Microsoft о разработке кластерной технологии, использующей Windows NT. Несколько компаний предлагают кластеры на основе UNIX-машин.

Наряду с ОС, ориентированными на совершенно определенный тип аппаратной платформы, существуют операционные системы, специально разработанные таким образом, чтобы они могли быть легко перенесены с компьютера одного типа на компьютер другого типа, так называемые *мобильные* ОС. Наиболее ярким примером такой ОС является популярная система UNIX. В этих системах аппаратно-зависимые места тщательно локализованы, так что при переносе системы на новую платформу переписываются только они. Средством, облегчающим перенос остальной части ОС, является написание ее на машинно-независимом языке, например, на C, который и был разработан для программирования операционных систем.

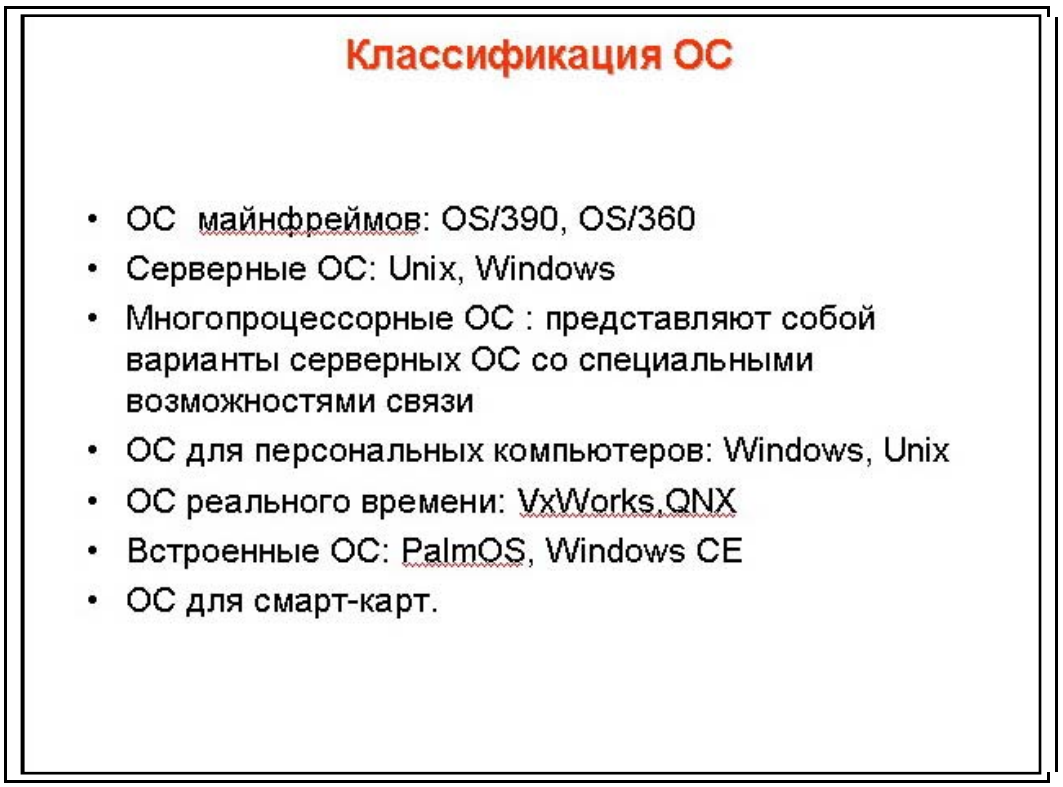

Рис. 9

Классификация ОС

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления, основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

Ниже приведена классификация ОС по нескольким наиболее основным признакам.

Особенности алгоритмов управления ресурсами

От эффективности алгоритмов управления локальными ресурсами компьютера во многом зависит эффективность всей сетевой ОС в целом. Поэтому, характеризуя сетевую ОС, часто приводят важнейшие особенности реализации функций ОС по управлению процессорами, памятью, внешними устройствами автономного компьютера. Так, например, в зависимости от особенностей использованного алгоритма управления процессором, операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многонитевую обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

Поддержка многозадачности

По числу одновременно выполняемых задач операционные системы могут быть разделены на два класса:

- однозадачные (например, MS-DOS, MSX)
- многозадачные (ОС ЕС, OS/2, UNIX, Windows 95).

Однозадачные ОС в основном выполняют функцию предоставления пользователю виртуальной машины, делая более простым и удобным процесс взаимодействия пользователя с компьютером. Однозадачные ОС включают средства управления периферийными устройствами, средства управления файлами, средства общения с пользователем.

Многозадачные ОС, кроме вышеперечисленных функций, управляют разделением совместно используемых ресурсов таких, как процессор, оперативная память, файлы и внешние устройства.

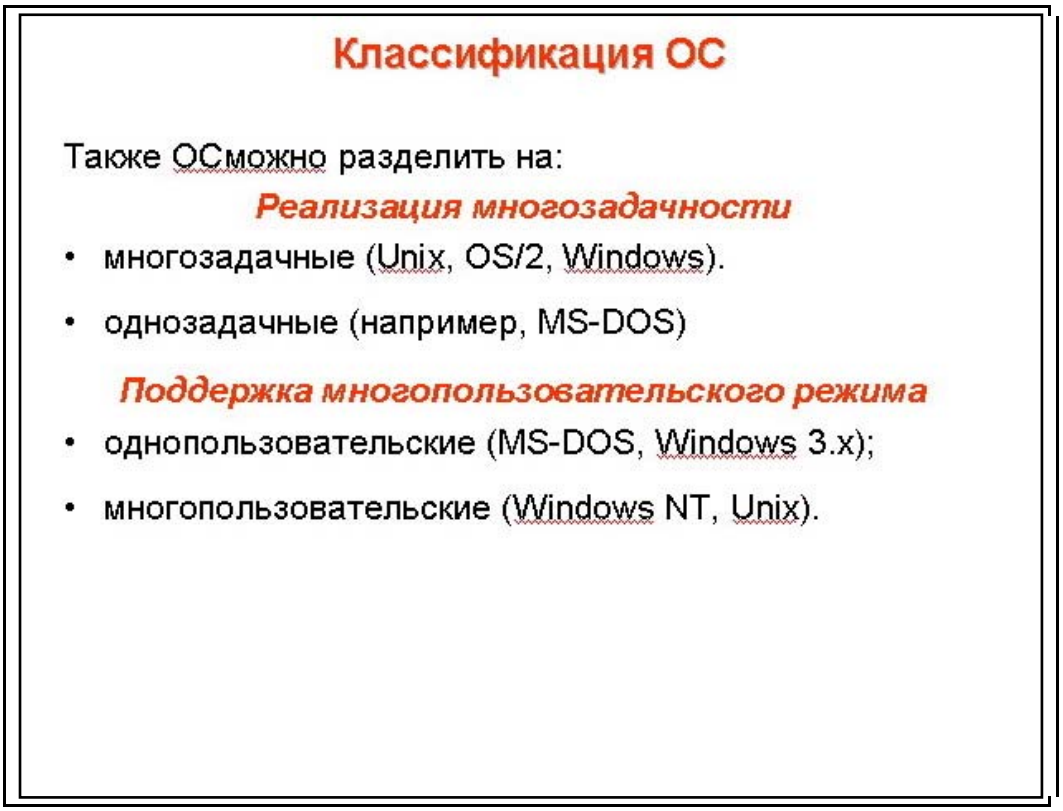

Рис. 10

Поддержка многопользовательского режима

По числу одновременно работающих пользователей ОС делятся на:

- однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2);
- многопользовательские (UNIX, Windows NT).

Главным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует

заметить, что не всякая многозадачная система является многопользовательской, и не всякая однопользовательская ОС является однозадачной.

Вытесняющая и невытесняющая многозадачность

Важнейшим разделяемым ресурсом является процессорное время. Способ распределения процессорного времени между несколькими одновременно существующими в системе процессами (или нитями) во многом определяет специфику ОС. Среди множества существующих вариантов реализации многозадачности можно выделить две группы алгоритмов:

- невытесняющая многозадачность (NetWare, Windows 3.x);
- вытесняющая многозадачность (Windows NT, OS/2, UNIX).

Основным различием между вытесняющим и невытесняющим вариантами многозадачности является степень централизации механизма планирования процессов. В первом случае механизм планирования процессов целиком сосредоточен в операционной системе, а во втором – распределен между системой и прикладными программами. При невытесняющей многозадачности активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению процесс. При вытесняющей многозадачности решение о переключении процессора с одного процесса на другой принимается операционной системой, а не самим активным процессом.

Поддержка многонитевости

Важным свойством операционных систем является возможность распараллеливания вычислений в рамках одной задачи. Многонитевая ОС разделяет процессорное время не между задачами, а между их отдельными ветвями (нитями).

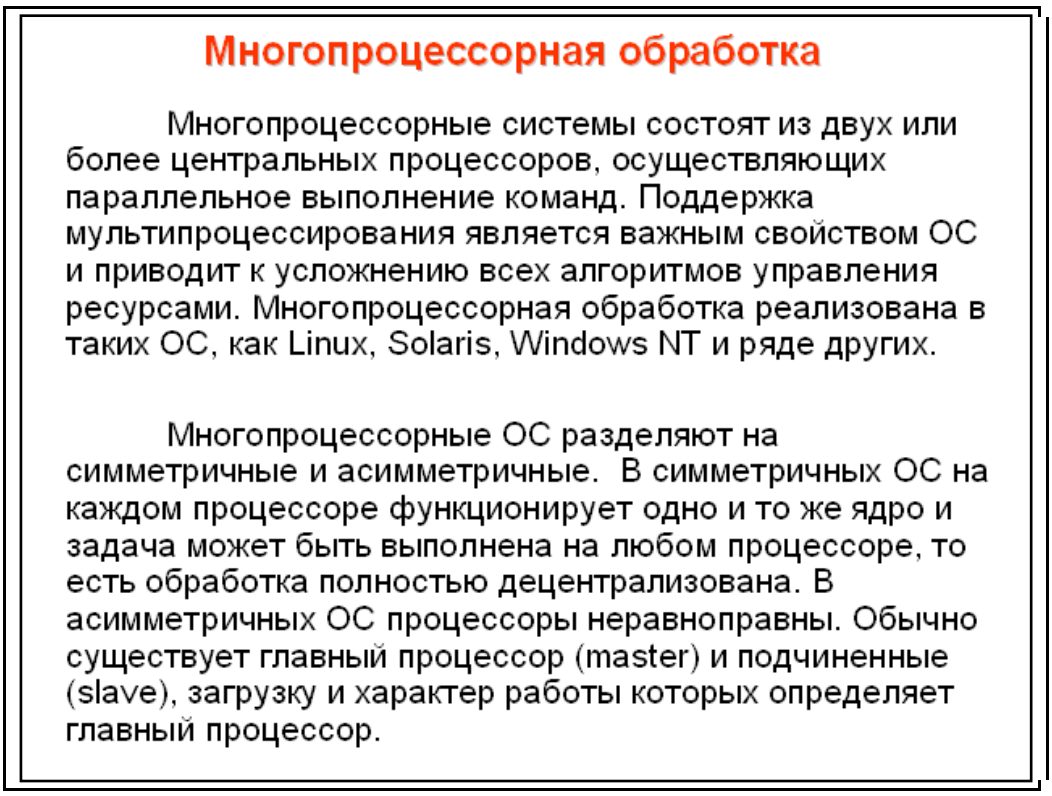

Рис. 11

Многопроцессорная обработка

Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки – *мультипроцессирование*. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами.

В наши дни становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris 2.x фирмы Sun, Open Server 3.x компании Santa Crus Operations, OS/2 фирмы IBM, Windows NT фирмы Microsoft и NetWare 4.1 фирмы Novell.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: асимметричные ОС и симметричные ОС. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

Выше были рассмотрены характеристики ОС, связанные с управлением только одним типом ресурсов – процессором. Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами – подсистем управления памятью, файлами, устройствами ввода-вывода.

Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов. При реализации сетевых функций возникает комплекс задач, связанных с распределенным характером хранения и обработки данных в сети: ведение справочной информации о всех доступных в сети ресурсах и серверах, адресация взаимодействующих процессов, обеспечение прозрачности доступа, тиражирование данных, согласование копий, поддержка безопасности данных.

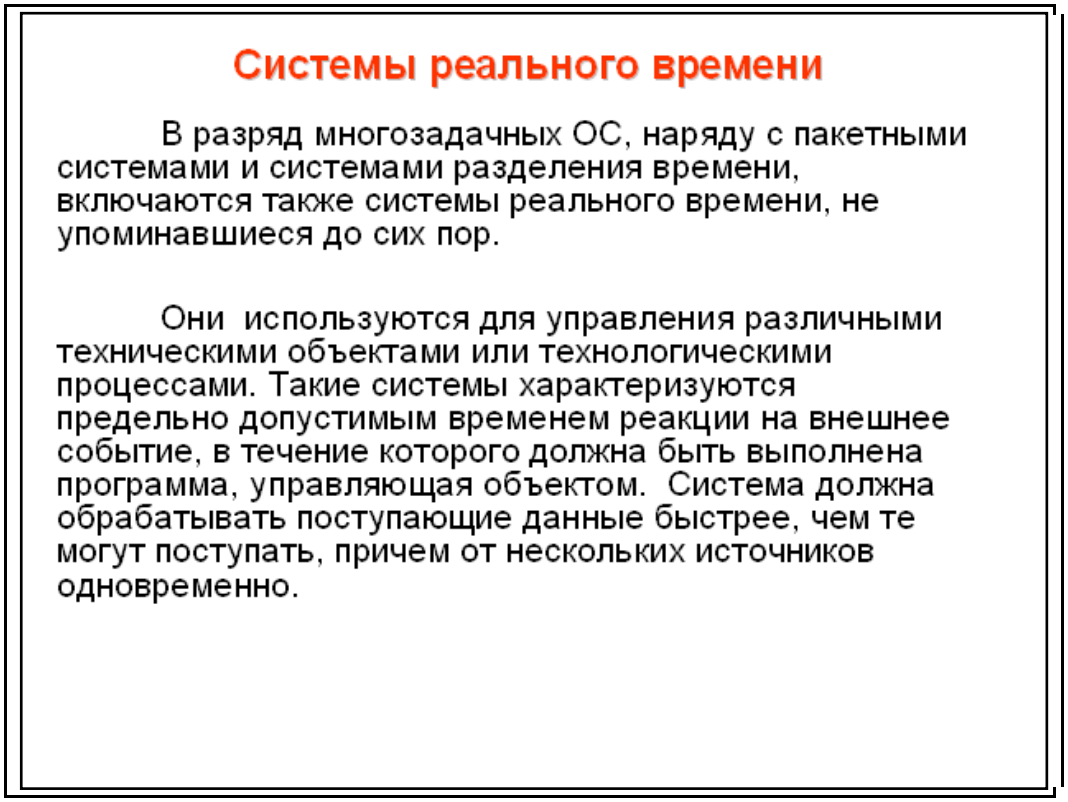

Рис. 12

Системы реального времени применяются для управления различными техническими объектами, такими, например, как станок, спутник, научная экспериментальная установка или технологическими процессами такими, как гальваническая линия, доменный процесс и т. п. Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа, управляющая объектом, в противном случае может произойти авария: спутник выйдет из зоны видимости, экспериментальные данные, поступающие с датчиков, будут потеряны, толщина гальванического покрытия не будет соответствовать норме. Таким образом, критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы – реактивностью. Для этих систем мультипрограммная смесь

представляет собой фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется, исходя из текущего состояния объекта или в соответствии с расписанием плановых работ.

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть задач может выполняться в режиме пакетной обработки, а часть – в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Характеристики операционной системы

Операционные системы	Характер взаимодействия пользователя с заданием	Число одновременно обслуживаемых пользователей	Обеспечиваемый режим работы ЭВМ
Пакетной обработки	Взаимодействие невозможно или ограничено	Один или несколько	Однопрограммный или мультипрограммный
Разделения времени	Диалоговый	Несколько	Мультипрограммный
Реального времени	Оперативный		Многозадачный
Диалоговая	Диалоговый	Один	Однопрограммный

Рис. 13

Особенности методов построения

При описании операционной системы часто указываются особенности ее структурной организации и основные концепции, положенные в ее основу.

К таким базовым концепциям относятся:

· Способы построения ядра системы – монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС – серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно,

так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой – ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы.

· Построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства, хорошо зарекомендовавшие себя на уровне приложений, внутри операционной системы, а именно:
аккумуляцию удачных решений в форме стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма наследования, хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне, структуризованность системы, состоящей из набора хорошо определенных объектов.

· Наличие нескольких прикладных сред дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, UNIX (POSIX), OS/2 или хотя бы некоторого подмножества из этого популярного набора. Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализует прикладную среду той или иной операционной системы.

Распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерными признаками распределенной организации ОС являются: наличие единой справочной службы разделяемых ресурсов, единой службы времени, использование механизма вызова удаленных процедур (RPC) для прозрачного распределения программных процедур по машинам, многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети, а также наличие других распределенных служб.

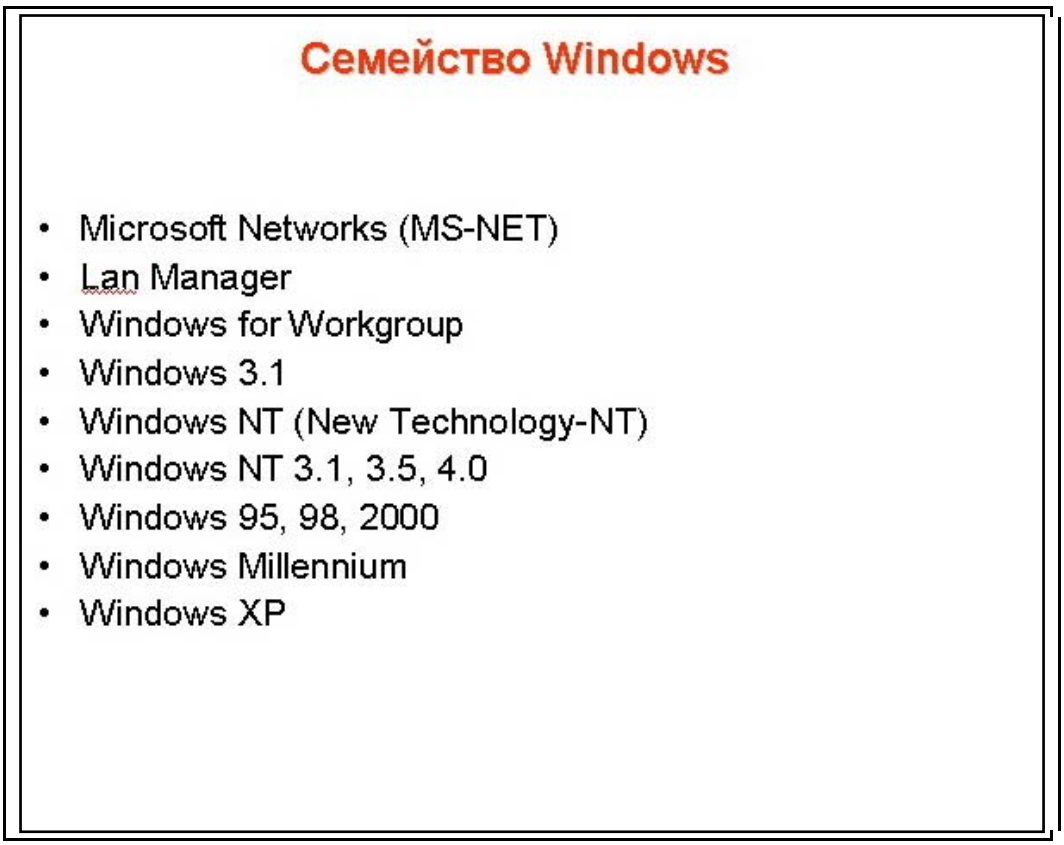

Рис. 14

Все ОС, созданные корпорацией Microsoft, можно условно разделить на три большие семейства:

1. MS-DOS и графические надстройки на него в виде Windows 3x
2. Consumer Windows (Windows'95, 98/Me)
3. Windows NT MS-DOS.

В 1981 году корпорация IBM создала ПК IBM PC. ПК был оснащен 16-разрядной однопользовательской ОС, названной MS-DOS 1.0. ОС, состояла из резидентной программы размером 8 Кбайт.

Два года спустя была создана более мощная ОС MS-DOS 2.0, которая уже содержала 24 Кбайта резидентного кода.

После этого компания Intel выпускает 286 процессор и на базе его ПК IBM PC AT. Данный компьютер поставлялся с ОС MS-DOS 3.0, которая состояла из 36 Кбайт резидентного кода, но ОС MS-DOS все равно оставалась ориентирована на командную строку.

Windows 9x Windows 1.0 вышла в 1985 году. В 1987 году вышла новая версия 2.0. Обе они не добились коммерческого успеха только ОС 3.0 выпущенная в 1990 году принесла значительные доходы Microsoft.

В Windows 3x все программы работали в одном и том же адресном пространстве. Из-за чего ошибка в одной из них приводила к зависанию всей системы.

В августе 1995 года выходит Windows'95; новая версия Windows, так же как и поставляемая с ней MS-DOS v7.0, обладала всеми особенностями

монолитной ОС, в том числе включая виртуальную память и систему управления процессом.

Windows 95 работала на основе файловой системы MS-DOS. В июне 1997 года была выпущена версия Windows 98. Основным отличием данной версии Windows от Windows 95 была разница в интерфейсе пользователя, более полная интеграция с INTERNET поддержка USB-устройств.

В Windows 95 были 2 основные проблемы: во-первых, не смотря на то, что система была многозадачной, могла сложиться ситуация: если процесс был занят управлением какой-либо структуры данных в ядре, то по истечении его кванта времени управление передавалось другому процессу, при этом новый процесс мог получить структуру данных в противоречивом состоянии.

Чтобы предотвратить данную ситуацию, каждому процессу, попадающему в ядро, навешивался гигантский мьютекс, покрывающий все систему, но из-за этого уничтожалось много преимуществ многозадачности. Во-вторых, нижний 1 Мбайт адресного пространства совместно использовался всеми процессами. В результате ошибка одной программы могла повредить структуры нескольких других программ.

В 2000 году выходит ОС Windows Me. Это была та же Windows 98, включающая в себя некоторые дополнительные функции: функцию отката системы, улучшенную поддержку сети и многопользовательских игр, так же использование изображений, музыки и фильмов.

Windows NT

К концу 80-х годов корпорация Microsoft осознала, что монтирование 32-разрядной ОС поверх 16-ти разрядной MS-DOS не является лучшим решением. Проект оказался успешным, и в 1993 году была выпущена версия Windows NT v 3.1. Наиболее важным в данной ОС была безопасность, высокая надежность, отсутствующая в других версиях Windows. Первое значительное усовершенствование Windows NT произошло в 1996 году с выпуском Windows NT 4.0. Кроме того, что она отличалась высокой безопасностью и надежностью, она обладала удобным пользовательским интерфейсом от Windows 95 что и обусловило ее коммерческий успех. ОС серии NT были практически полностью написаны на языке Си с небольшими включениями на Assembler для низкоуровневых функций. Всего в версии NT 4.0 было 16 000 000 строк. Windows 2000 следом за Windows 4.0 было решено выпустить версию Windows NT 5.0, которую переименовали в Windows 2000. Основной ее особенностью является то, что это полностью 32-разрядная ОС, многозадачная, с индивидуально защищенными процессами, каждый из которых имеет 32-разрядное адресное пространство. Еще более полное усовершенствование заключается в том, что появилась поддержка информационных портов, каталоговая служба Active Directory, поддержка смарт карт, инструменты мониторинга системы, совершенную структуру администрирования.

К сожалению, Windows 2000 стала поддерживать только 2 платформы: Intel IA - 64 Pentium. Всего ОС Windows 2000 выходила в 4-х разновидностях: Professional, Server, Advanced server и Datacenter Windows XP.

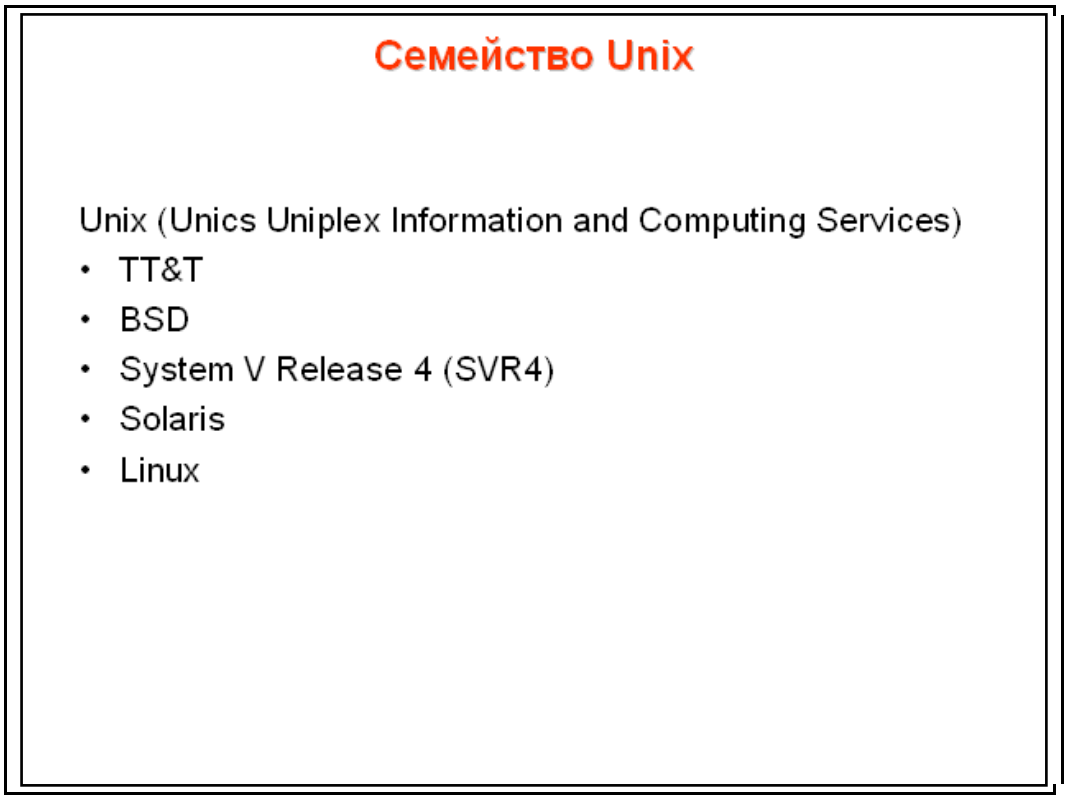

Рис. 15

Стандартная система UNIX

К концу 80-х широкое распространение получили две различные и в чем-то несовместимые версии системы UNIX: 4.3BSD и System V Release 3. Кроме того, практически каждый производитель добавлял свои нестандартные усовершенствования. Этот раскол в мире UNIX, вместе с тем фактом, что стандарта на формат двоичных программ не было, сильно замедлил коммерческое признание операционной системы UNIX. Производители программного обеспечения не могли написать пакет программ для системы UNIX так, чтобы он гарантированно мог быть запущен на любой системе UNIX (как, например, это делалось в системе MS-DOS). Различные попытки стандартизации системы UNIX провалились с самого начала. Например, корпорация AT&T выпустила стандарт **SVID** (System V Interface Definition — описание интерфейса UNIX System V), в котором определялись все системные вызовы, форматы файлов и т. д. Этот документ был попыткой построить в одну шеренгу всех производителей UNIX System V, но он не оказал никакого влияния на вражеский лагерь (BSD), который просто проигнорировал его.

Первая попытка примирить два варианта системы UNIX была предпринята при содействии Совета по стандартам при Институте инженеров по электротехнике и электронике IEEE Standard Boards, глубокоуважаемой и,

что еще важнее, нейтральной организации. В этой работе приняли участие сотни людей из промышленности, академических и правительственных организаций. Коллективное название данного проекта — **POSIX.** Первые три буквы этого сокращения означали Portable Operating System — переносимая операционная система. Буквы *IX* в конце слова были добавлены, чтобы имя проекта выглядело юниксообразно.

После большого количества высказанных аргументов и контраргументов, опровержений и опровергнутых опровержений, комитет POSIX выработал стандарт, известный как 1003.1. Этот стандарт определяет набор библиотечных процедур, которые должна предоставлять каждая соответствующая данному стандарту система UNIX. Большая часть этих процедур обращается к системному вызову, но некоторые из них могут быть реализованы вне ядра. Типичными процедурами являются *open, read* и *fork.* Идея стандарта POSIX заключается в том, что производитель программного обеспечения при написании программы использует только процедуры, описанные в стандарте 1003.1, таким образом, гарантируя, что эта программа будет работать на любой версии системы UNIX, поддерживающей данный стандарт.

Хотя большинство комитетов по стандартам, как правило, создают нечто ужасное, сплошь состоящее из компромиссов, стандарт 1003.1 заметно отличается от общего правила в лучшую сторону, особенно если учитывать большое число заинтересованных сторон, принимавших участие в его разработке. Вместо того, чтобы принять за точку отсчета объединение множеств всех свойств System V и BSD (норма для большинства комитетов по стандартам), комитет IEEE взял за основу пересечение множеств. То есть в первом приближении дело обстоит так: если какое-либо свойство присутствовало как в System V, так и в BSD, то оно включалось в стандарт. В противном случае это свойство в стандарт не включалось. В результате применения такого алгоритма стандарт 1003.1 сильно напоминает прямого общего предка систем System V и BSD, а именно Version 7. От Version 7 стандарт сильнее всего отличается в двух областях: обработке сигналов (что по большей части взято из BSD) и управлению терминалом, что представляет собой нововведение. Документ 1003.1 написан так, чтобы как разработчики операционной системы, так и создатели программного обеспечения были способны его понять, что также ново в мире стандартов, хотя в настоящее время уже полным ходом ведется работа по исправлению этого нестандартного для стандартов свойства.

Ядро состояло из 1600 строк на C и 800 ассемблерных строк. По техническим причинам, связанным с архитектурой процессора Intel 8088, драйверы устройств ввода-вывода (еще 2900 строк на C) также были размещены в ядре. Файловая система (5100 строк на C) и менеджер памяти (2200 строк на C) работали как два раздельных пользовательских процесса.

Преимущество микроядер перед монолитными системами заключается в том, что устройство микроядра легко понять, да и поддержка системы, основанной на микроядре, проще, благодаря модульной структуре такой

системы. Кроме того, перемещение программного обеспечения из ядра в пространство пользователя существенно повышает надежность системы, так как сбой процесса, работающего в режиме пользователя, способен нанести меньший ущерб, чем сбой компонента в режиме ядра. Основной недостаток состоит в несколько меньшей производительности, связанной с дополнительными переключениями из режима пользователя в режим ядра. Однако производительность – не единственное достоинство системы. На всех современных системах UNIX оконная система X Windows работает в режиме пользователя, в результате чего производительность несколько снижается, зато достигается большая модульность (в отличие от системы Windows, у которой весь графический интерфейс пользователя расположен в ядре). Среди других хорошо известных примеров схемы микроядра того времени можно назвать Mach [4] и Chorus [282]. Обсуждение производительности микроядерной системы UNIX приводится в [42].

Уже через несколько месяцев после своего появления система MINIX стала чем-то вроде объекта культа — у нее есть своя конференция, *comp.os.minix,* и более 40 000 пользователей. Многие пользователи сами стали писать команды и пользовательские программы, так что система MINIX стала продуктом коллективного творчества большого количества пользователей, обменивающихся своими разработками по Интернету. Она стала прототипом других коллективных работ, появившихся позднее. В 1997 году была выпущена версия 2.0 системы MINIX. Базовая система теперь включала в себя сетевое программное обеспечение, и ее размер вырос до 62 200 строк. О системе MINIX написана книга, содержащая 500 страниц исходного текста в приложении к книге, а также на поставляемом с книгой CD-ROM [326]. Исходный текст операционной системы MINIX можно бесплатно получить на web-сайте www.cs.vu.nl/~ast/minix.html.

Linux

В ранние годы развития системы MINIX и обсуждений этой системы в Интернете многие люди просили (а часто требовали) все больше новых и более сложных функций, и на эти просьбы автор часто отвечал отказом (сохраняя небольшие размеры системы, чтобы студенты могли полностью понять ее за один семестр). Эти отказы раздражали многих пользователей. В те времена бесплатной системы FreeBSD еще не было. Наконец через несколько лет финский студент Линус Торвальдс решил сам написать еще один клон системы UNIX, который он назвал Linux. Это должна была быть полноценная операционная система, со многими функциями, отсутствующими (по намерению авторов) в системе MINIX. Первая версия операционной системы Linux 0.01 была выпущена в 1991 году. Она была разработана и собрана в системе MINIX и заимствовала некоторые идеи системы MINIX, начиная со структуры дерева исходных текстов и кончая структурой файловой системы. Однако, в отличие от микроядерной системы MINIX, Linux была монолитной системой, то есть вся операционная система помещалась в ядре. Размер

исходного текста составил 9300 строк на C и 950 строк на ассемблере, что приблизительно совпадало с версией MINIX. Функционально первая версия Linux также практически не отличалась от MINIX.

Операционная система Linux быстро росла в размерах и впоследствии развилась в полноценный клон UNIX с виртуальной памятью, более сложной файловой системой и многими другими добавленными функциями. Хотя изначально система Linux работала только на процессоре Intel 386 (и даже содержала ассемблерный код 386-й машины посреди процедур на языке C), она была довольно быстро перенесена на другие платформы и теперь работает на широком спектре машин, как и UNIX. Однако одно из основных отличий системы Linux от других клонов системы UNIX заключается в использовании многих специальных особенностей компилятора *gcc,* поэтому, чтобы откомпилировать ее стандартным ANSI C компилятором, потребуется приложить немало усилий.

Следующим основным выпуском системы Linux была версия 1.0, появившаяся в 1994 году. Она состояла из 165 000 строк кода и включала новую файловую систему, отображение файлов на адресное пространство памяти и совместимое с BSD сетевое программное обеспечение с сокетами и TCP/IP. Она также включала многие новые драйверы устройств. Следующие два года выходили версии с незначительными исправлениями.

К этому времени операционная система Linux стала достаточно совместимой с UNIX, поэтому в нее было перенесено большое количество программного обеспечения UNIX, что значительно увеличило полезность рассматриваемой системы. Кроме того, операционная система Linux привлекла большое количество людей, которые начали работу над ее совершенствованием и расширением под общим руководством Торвальдса.

Следующий главный выпуск, версия 2.0, вышел в свет в 1996 году. Эта версия системы Linux состояла из 470 000 строк на C и 8000 строк ассемблерного текста. Она включала в себя поддержку 64-разрядной архитектуры, симметричной многозадачности, новых сетевых протоколов и прочих многочисленных функций. Значительную часть от общей массы исходного текста составляла внушительная коллекция различных драйверов устройств. Следом за версией 2.0 довольно часто выходили дополнительные выпуски.

В систему Linux была перенесена внушительная часть стандартного программного обеспечения UNIX, включая более 1000 утилит, оконную систему X Windows и большую часть сетевого программного обеспечения. Кроме того, специально для Linux было написано два различных графических интерфейса пользователя: GNOME и KDE. В общем, система Linux выросла в полноценный клон UNIX со всеми «погремушками», какие только могут понадобиться фанату UNIX.

Необычной особенностью Linux является ее бизнес-модель: это свободно распространяющееся программное обеспечение. Ее можно скачать с различных Интернет-сайтов, например www.kernel.org. Система Linux поставляется вместе

с лицензией, разработанной Ричардом Столманом, основателем Фонда бесплатно распространяемых программ Free Software Foundation. Несмотря на то, что система Linux распространяется бесплатно, эта лицензия, называющаяся GPL (GNU Public License — общедоступная лицензия), по длине превышает лицензию корпорации Microsoft для операционной системы Windows 2000. Она содержит сведения о том, что вы можете и чего не можете делать с исходным текстом. Пользователи могут свободно использовать, копировать, модифицировать и распространять дальше исходные тексты и двоичные файлы. Основной запрет касается отдельной продажи или распространения двоичного кода без исходных текстов. Исходные тексты должны либо поставляться вместе с двоичными файлами, либо предоставляться по требованию.

Хотя Торвальдс до сих пор довольно внимательно контролирует ядро системы, большое количество программ пользовательского уровня было написано другими многочисленными программистами, многие из которых изначально перешли на Linux из сетевых сообществ MINIX, BSD и GNU (Free Software Foundation). Однако по мере развития системы Linux все меньшая часть сообщества Linux желает ковыряться в исходном тексте (свидетельством тому служат сотни книг, описывающих, как установить систему Linux и как ею пользоваться, и только несколько книг, в которых обсуждается сама система, или как она работает). Кроме того, многие пользователи Linux теперь предпочитают бесплатному скачиванию системы по Интернету покупку одного из CD-ROM, распространяемых многочисленными коммерческими компаниями. На web-сайте www.linux.org перечислено более 50 компаний, продающих различные пакеты Linux. По мере того, как все больше и больше компаний, занимающихся программным обеспечением, начинают продавать свои версии Linux, а все большее число производителей компьютеров поставляют систему Linux со своими машинами, граница между коммерческим и бесплатным программным обеспечением слегка размывается.

Память

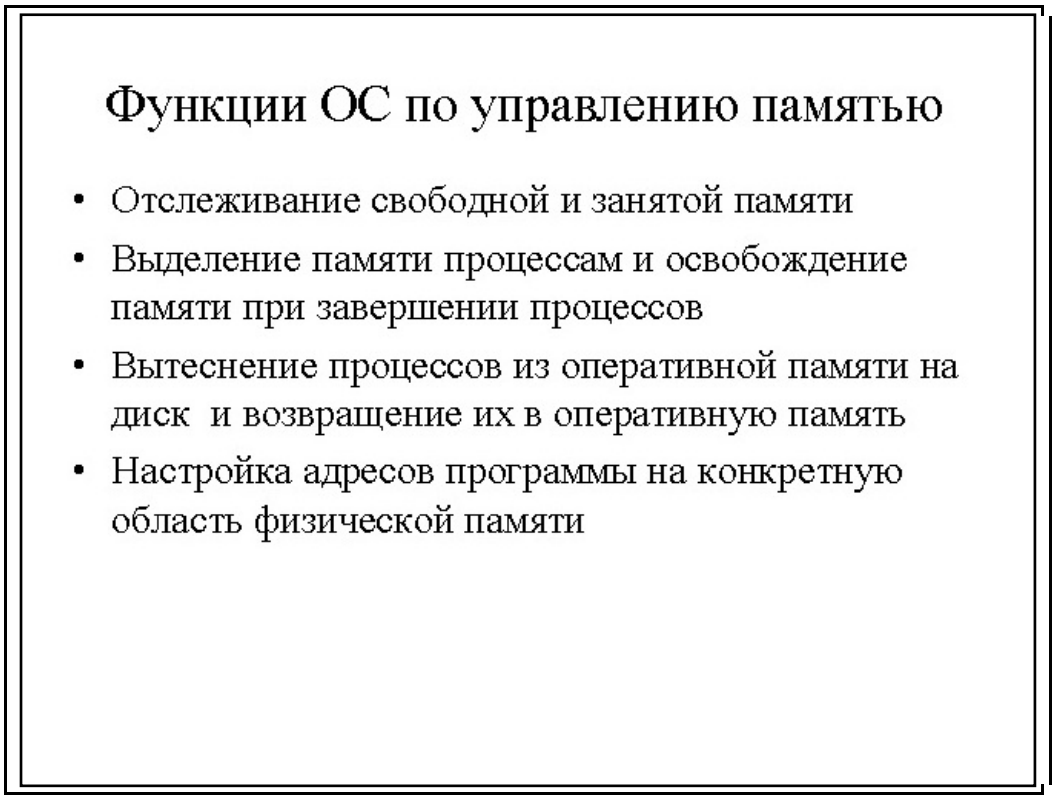

Рис. 16

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти, выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов программы на конкретную область физической памяти.


Схема распределения памяти
Без использования внешней памяти
С использованием внешней памяти
1. С фиксированными разделами
2. С динамическими разделами
3. С перемещаемыми разделами
1. Страничный
2. Сегментарный
3. Странично-сегментарный

Рис. 17

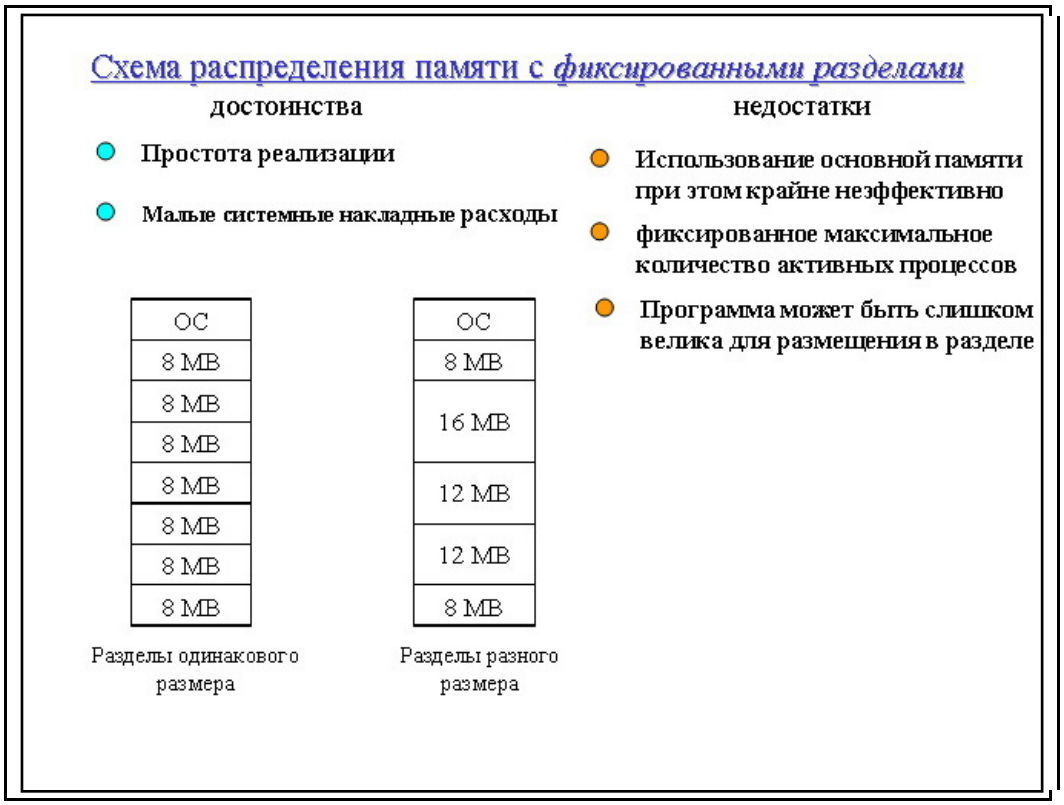
Схема распределения памяти с фиксированными разделами
достоинства
Простота реализации
Малые системные накладные расходы
недостатки
Использование основной памяти при этом крайне неэффективно
фиксированное максимальное количество активных процессов
Программа может быть слишком велика для размещения в разделе
ОС
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB
Разделы одинакового размера
ОС
8 MB
16 MB
12 MB
12 MB
8 MB
Разделы разного размера

Рис. 18

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации.

Подсистема управления памятью в этом случае выполняет следующие задачи: сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел, осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе – простоте реализации – данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов независимо от того, какой размер имеют программы. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объем оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

Рис. 19

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача.

Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

Задачами операционной системы при реализации данного метода управления памятью является: ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти, при поступлении новой задачи – анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи, загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей, после завершения задачи – корректировка таблиц свободных и занятых областей.

Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам, таким, например, как «первый попавшийся раздел достаточного размера», или «раздел, имеющий наименьший достаточный размер», или «раздел, имеющий наибольший достаточный размер». Все эти правила имеют свои преимущества и недостатки.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – *фрагментация памяти*. Фрагментация – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

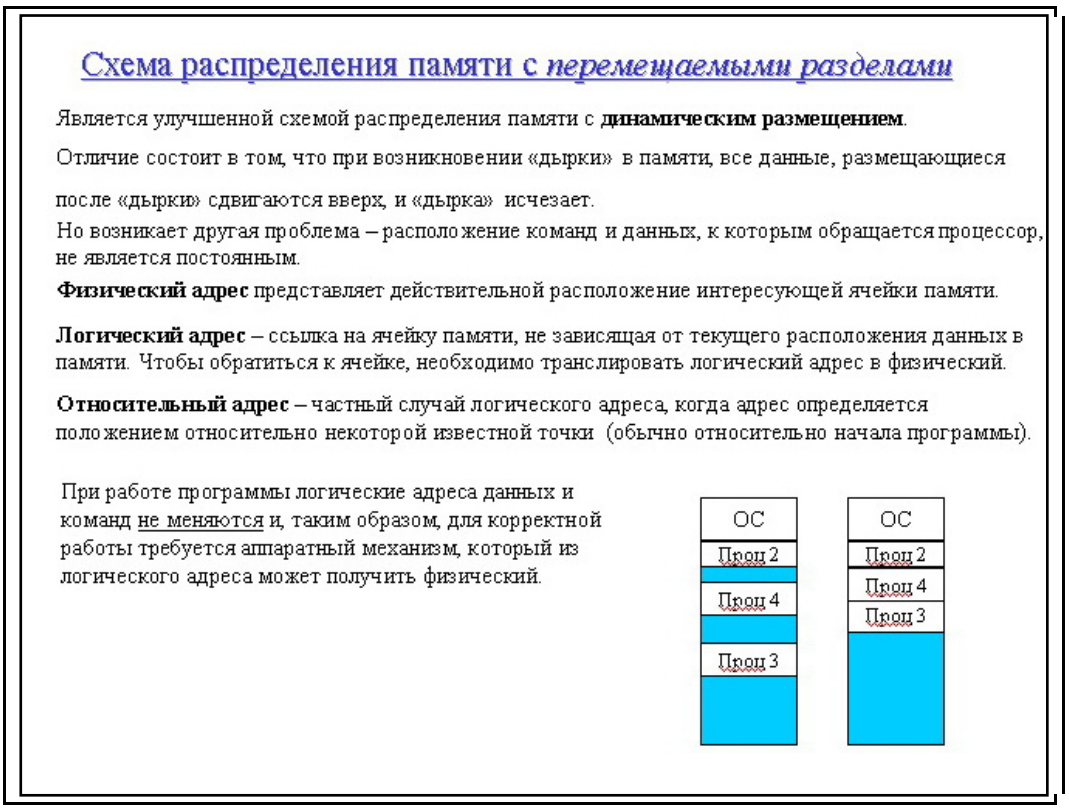

Рис. 20

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших, либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область. В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется «сжатием». Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором – реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

Рис. 21

В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

- дольше всего не использовавшаяся страница;
- первая попавшаяся страница;
- страница, к которой в последнее время было меньше всего обращений.

В некоторых системах используется понятие рабочего множества страниц. Рабочее множество определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, которые должны постоянно находиться в оперативной памяти и поэтому не подлежат выгрузке.

После того, как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

При каждом обращении к оперативной памяти аппаратными средствами выполняются следующие действия:

- на основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адреса) и длины записи в таблице

страниц (системная константа) определяется адрес нужной записи в таблице, из этой записи извлекается номер физической страницы;
- к номеру физической страницы присоединяется смещение (младшие разряды виртуального адреса).

Использование в пункте (3) того факта, что размер страницы равен степени 2, позволяет применить операцию *конкатенации (присоединения)* вместо более длительной операции сложения, что уменьшает время получения физического адреса, а значит повышает производительность компьютера.

На производительность системы со страничной организацией памяти влияют временные затраты, связанные с обработкой страничных прерываний и преобразованием виртуального адреса в физический. При часто возникающих страничных прерываниях система может тратить большую часть времени впустую, на свопинг страниц. Чтобы уменьшить частоту страничных прерываний, следовало бы увеличивать размер страницы. Кроме того, увеличение размера страницы уменьшает размер таблицы страниц, а значит уменьшает затраты памяти. С другой стороны, если страница велика, значит велика и фиктивная область в последней виртуальной странице каждой программы. В среднем на каждой программе теряется половина объема страницы, что в сумме при большой странице может составить существенную величину. Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в «быстрых» запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

Страничное распределение памяти может быть реализовано в упрощенном варианте, без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в оперативной памяти. Такой вариант страничной организации хотя и не предоставляет пользователю виртуальной памяти, но почти исключает фрагментацию за счет того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

Рис. 22

На рисунке показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками).

Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т. д., это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные – на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру – таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

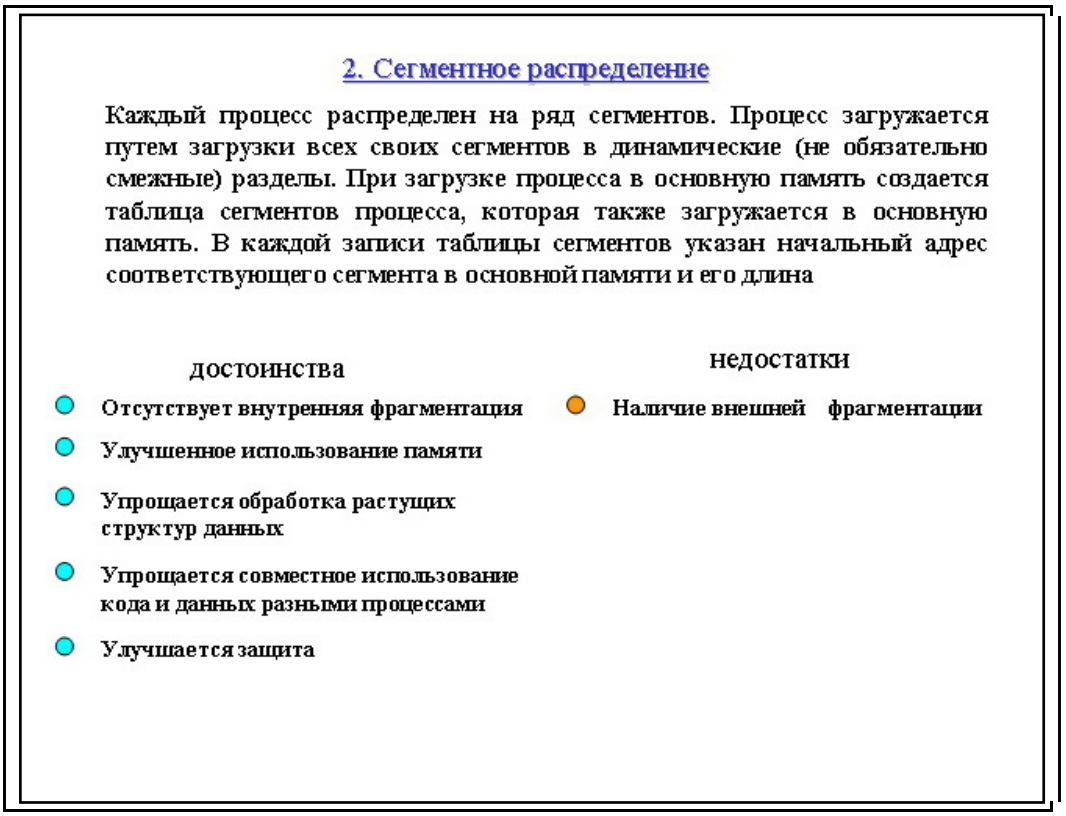

Рис. 23

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение. Кроме того, разбиение программы на «осмысленные» части делает принципиально возможным разделение одного

сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

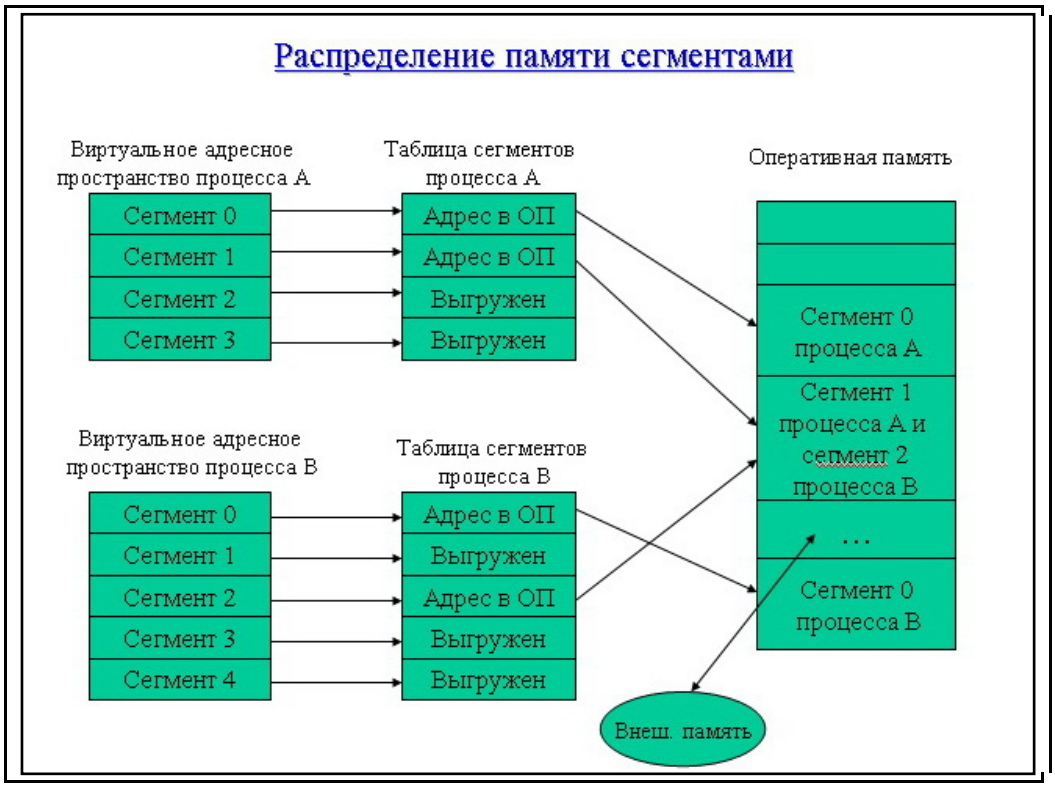

Рис. 24

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности (рисунок.) Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Иногда сегментация программы выполняется по умолчанию компилятором.

При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок

оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.

Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g – номер сегмента, а s – смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру g и смещения s.

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

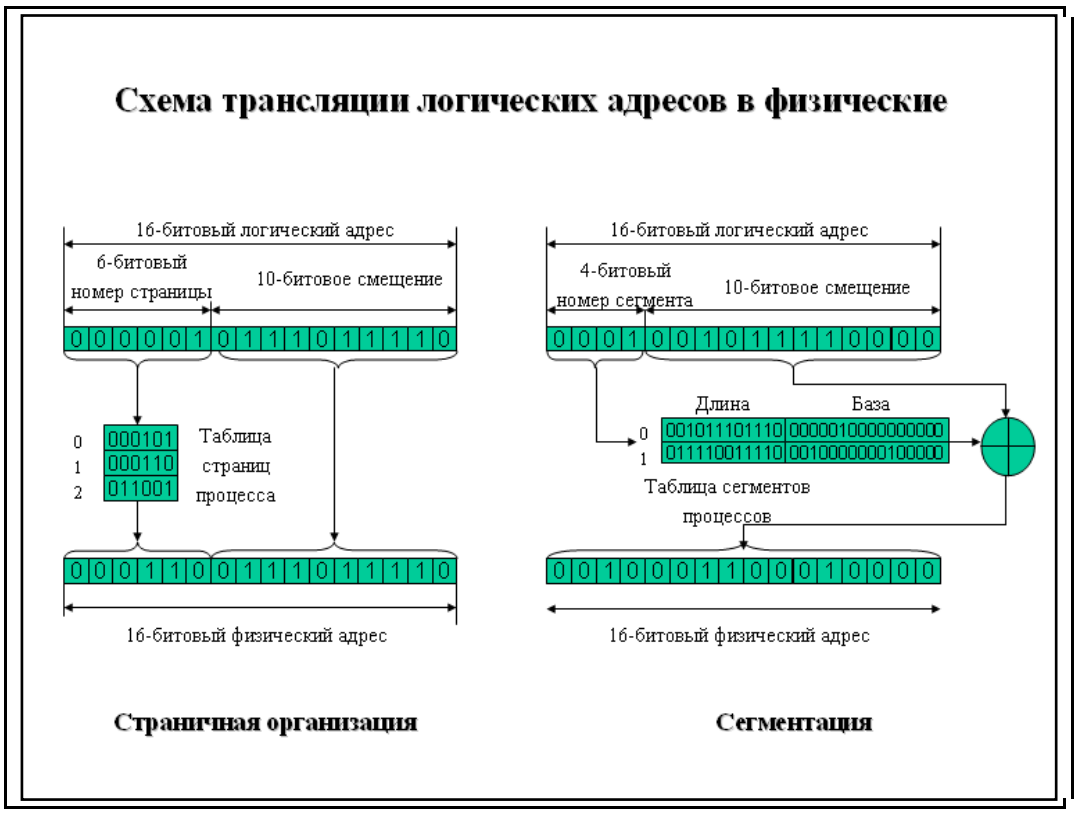

Рис. 25

3. Сегментно-страничное распределение

Представляет собой комбинацию сегментной и страничной организации. В такой комбинированной системе адресное пространство пользователя разбивается на ряд сегментов по усмотрению программиста. Каждый сегмент разбивается на ряд страниц фиксированного размера, соответствующего размеру кадра основной памяти. Если размер сегмента меньше размера страницы, он занимает страницу целиком. Логический адрес в этом случае состоит из номера сегмента и смещения в нем. С позиции операционной системы смещение в сегменте следует рассматривать как номер страницы определенного сегмента и смещение в ней.

достоинства

- **Отсутствует внешняя и внутренняя фрагментация**
- **Облегчается создание эффективных алгоритмов управления памятью**
- **Модульность**
- **Возможность обработки растущих структур данных**
- **Поддержка совместного использования**
- **Защита памяти**

Рис. 26

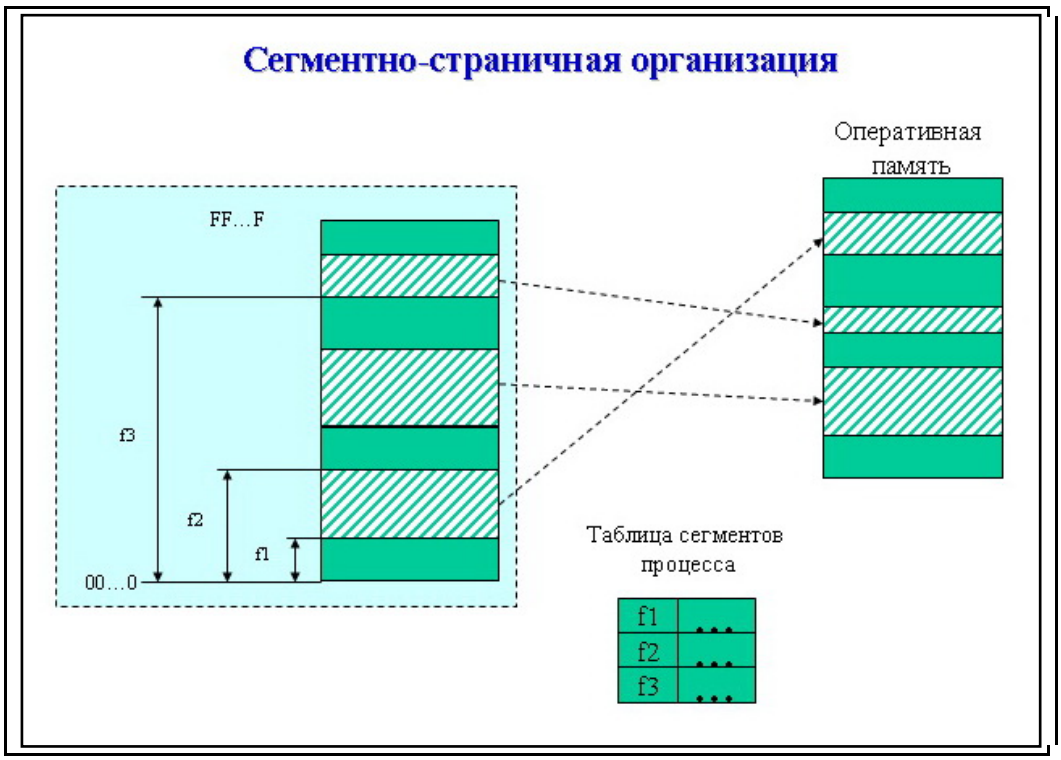

Рис. 27

В большинстве современных реализации сегментно-страничной организации памяти в отличие от набора виртуальны диапазонов адресов при сегментной организации памяти все виртуальные сегменты образуют одно непрерывное пространство.

Для каждого процесса операционная система создает отдельную таблицу сегментов, в которой содержатся описатели (дескрипрторы) всех сегментов процесса. Описание сегмента включает назначенные ему права доступа и другие характеристики. В поле базового адреса указывается начальный линейный виртуальный адрес сегмента в пространстве виртуальных адресов (f1, f2, f3).

Наличие базового виртуального адреса сегмента в дескрипторе позволят однозначно преобразовать адрес, заданный в виде пары (номер сегмента, смещение в сегменте), в линейный виртуальный адрес байта, который затем преобразуется в физический адрес страничным механизмом

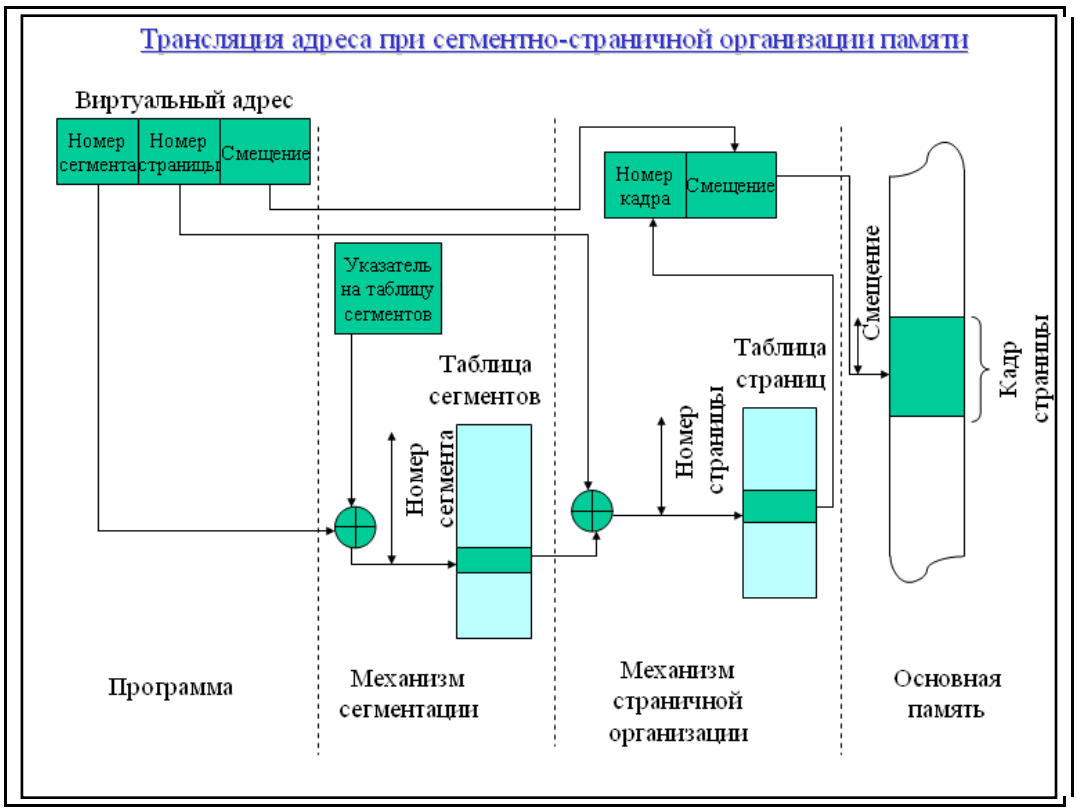

Рис. 28

Как видно из названия, данный метод представляет собой комбинацию страничного и сегментного распределения памяти и, вследствие этого, сочетает в себе достоинства обоих подходов. Виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы. Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создается своя

таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс. На рисунке показана схема преобразования виртуального адреса в физический для данного метода.

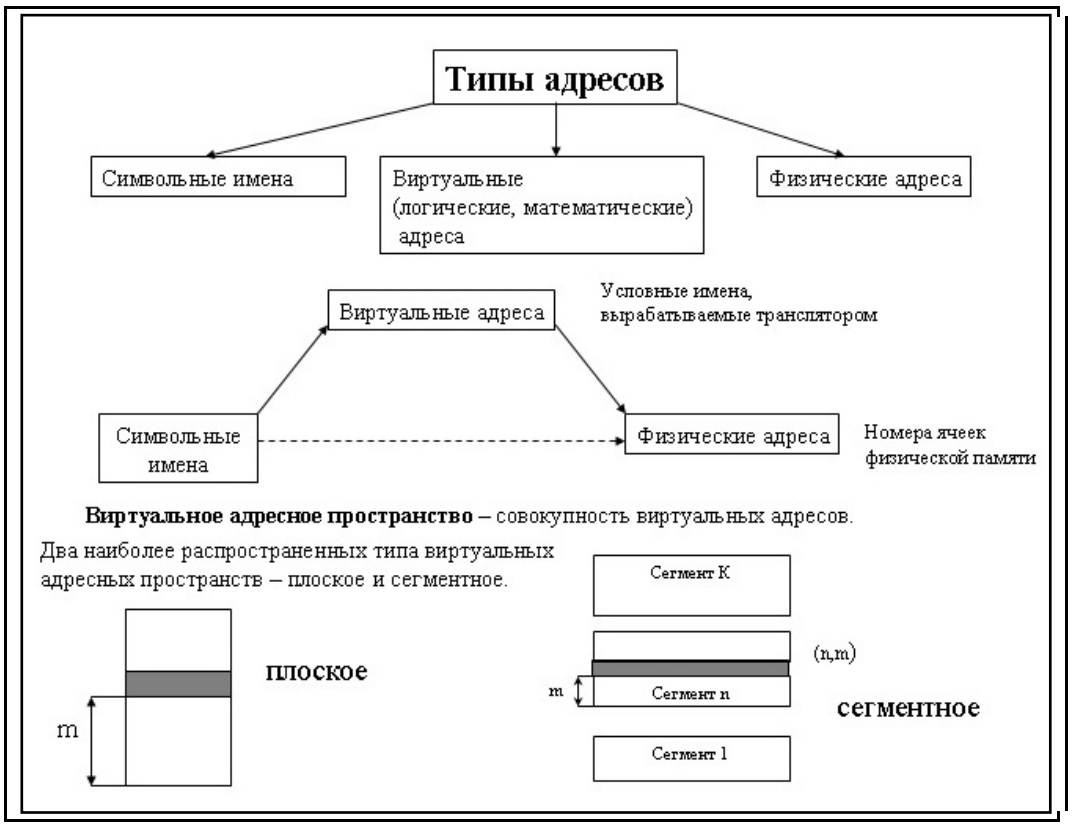

Рис. 179

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса.

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что программа будет размещена, начиная с нулевого адреса. Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством*. Каждый процесс имеет собственное виртуальное адресное пространство. Максимальный размер виртуального адресного пространства ограничивается разрядностью адреса, присущей данной

архитектуре компьютера, и, как правило, не совпадает с объемом физической памяти, имеющимся в компьютере.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа – *перемещающий загрузчик*. Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Второй способ является более гибким, он допускает перемещение программы во время ее выполнения, в то время как перемещающий загрузчик жестко привязывает программу к первоначально выделенному ей участку памяти. Вместе с тем использование перемещающего загрузчика уменьшает накладные расходы, так как преобразование каждого виртуального адреса происходит только один раз во время загрузки, а во втором случае – каждый раз при обращении по данному адресу.

В некоторых случаях (обычно в специализированных системах), когда заранее точно известно, в какой области оперативной памяти будет выполняться программа, транслятор выдает исполняемый код сразу в физических адресах.

Рис. 30

Когда происходит страничное прерывание, операционная система должна выбрать страницу для удаления из памяти, чтобы освободить место для страницы, которую нужно перенести в память. Если удаляемая страница была изменена за время своего присутствия в памяти, ее необходимо переписать на диск, чтобы обновить копию, хранящуюся там. Однако если страница не была модифицирована (например, она содержит текст программы), копия на диске уже является самой новой и ее не надо переписывать. Тогда страница, которую нужно прочитать, просто считывается поверх выгружаемой страницы.

Хотя в принципе можно при каждом страничном прерывании выбирать случайную страницу для удаления из памяти, производительность системы заметно повышается, когда предпочтение отдается редко используемой странице. Если выгружается страница, обращения к которой происходят часто, велика вероятность , то вскоре опять потребуется ее возврат в память, что даст в результате дополнительные издержки.

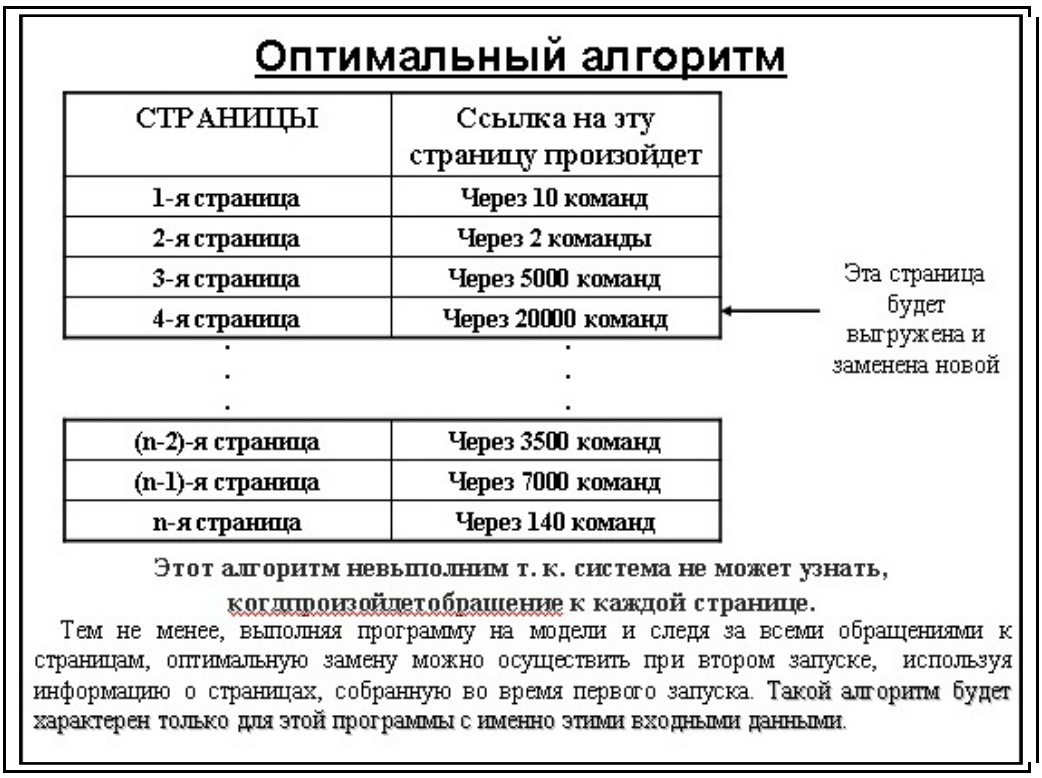

Рис. 31

Оптимальный алгоритм (OPT). Одним из последствий открытия ***аномалии Билэди*** стал поиск оптимального алгоритма, который при заданной строке обращений имел бы минимальную частоту ***page faults*** среди всех других алгоритмов. Такой алгоритм был найден. Он прост: замещай страницу, которая не будет использоваться в течение самого длительного периода времени. Каждая страница должна быть помечена числом инструкций, которые будут выполнены, прежде чем на эту страницу будет сделана первая ссылка. Выталкиваться должна страница, для которой это число наибольшее. Этот алгоритм легко описать, но реализовать невозможно. ОС не знает, к какой странице будет следующее обращение. (Ранее такие проблемы возникали при планировании процессов – алгоритм SJF). Зато мы можем сделать вывод, что для того, чтобы алгоритм замещения был максимально близок к идеальному алгоритму, система должна как можно точнее предсказывать обращения процессов к памяти. Данный алгоритм применяется для оценки качества реализуемых алгоритмов.

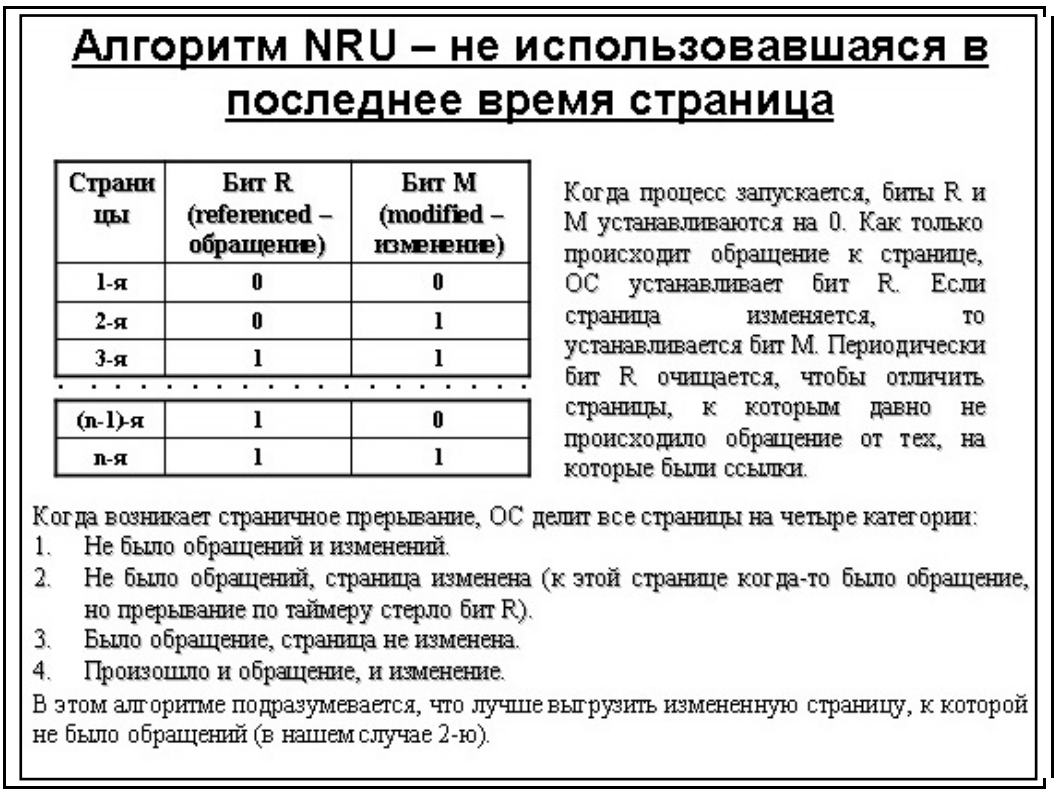

Рис. 32

Используются биты обращения (R-Referenced) и изменения (M-Modified) в таблице страниц.

При обращении бит R выставляется в 1, через некоторое время ОС не переведет его в 0.

M переводится в 0 только после записи на диск.

Благодаря этим битам можно получить 4 класса страниц:

1. Не было обращений и изменений (R=0, M=0).
2. Не было обращений, было изменение (R=0, M=1).
3. Было обращение, не было изменений (R=1, M=0).
4. Было обращения и изменения (R=1, M=1).

Рис. 33

SCHED_FIFO: планировщик FIFO (First In-First Out)

Алгоритм *SCHED_FIFO* можно использовать только со значениями статическогоприоритета, большими нуля.Это означает, что если процесс с алгоритмом *SCHED_FIFO* готов к работе, то он сразу запустится, а все обычные процессы с алгоритмом *SCHED_OTHER* будут приостановлены. *SCHED_FIFO* – это простой алгоритм без квантования времени. Процессы, работающие согласно алгоритму *SCHED_FIFO,* подчиняются следующим правилам: процесс с алгоритмом *SCHED_FIFO*, приостановленный другим процессом с большим приоритетом, останется в начале очереди процессов с равным приоритетом, и его исполнение будет продолжено сразу после того, как закончатся процессы с большими приоритетами. Когда процесс с алгоритмом *SCHED_FIFO* готов к работе, он помещается в конец очереди процессов с тем же приоритетом. Вызов функции **sched_setscheduler** или **sched_setparam** , который посылается процессом под номером *pid* с алгоритмом *SCHED_FIFO,* приведет к тому, что процесс будет перемещен в конец очереди процессов с тем же приоритетом. Процесс, вызывающий **sched_yield**, также будет помещен в конец списка. Других способов перемещения процесса с алгоритмом *SCHED_FIFO* в очереди процессов с одинаковыми статическим приоритетом не существует. Процесс с алгоритмом *SCHED_FIFO* работает до тех пор, пока не будет заблокирован запросом на

ввод/вывод, приостановлен процессом с большим статическим приоритетом или не вызовет **sched_yield**.

Рис. 33

Подобен FIFO, но если R=1, то страница переводится в конец очереди, если R=0, то страница выгружается.

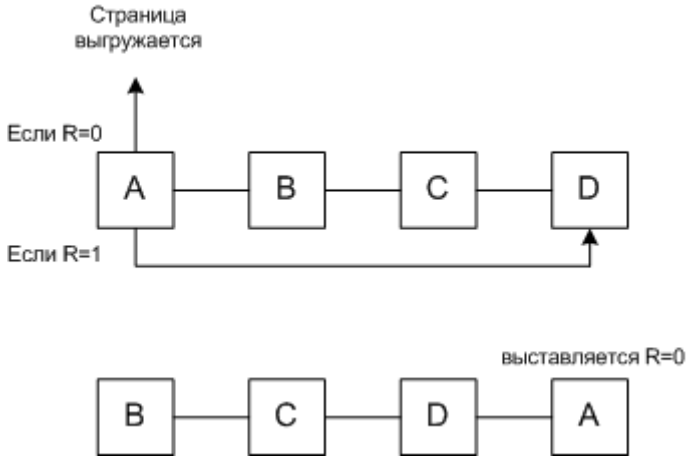

Рис. 34 Алгоритм «вторая попытка»

В таком алгоритме часто используемая страница никогда не покинет память. Но в этом алгоритме приходится часто перемещать страницы по списку.

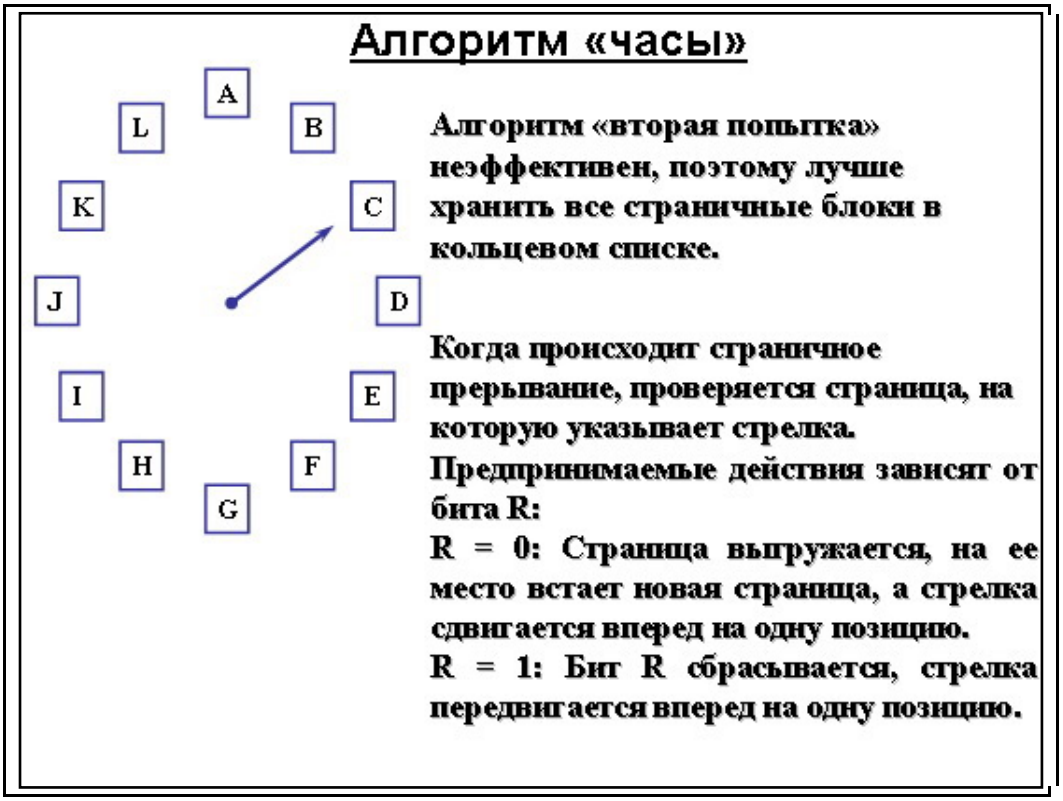

Рис. 35

Рис. 36

Когда требуется загрузить блок в заполненный до предела кэш, какой-либо другой блок должен быть удален из кэша (и записан на диск, если он был модифицирован в кэше). Эта ситуация очень похожа на страничную организацию памяти, и к ней применимы все обычные алгоритмы замены такие как, mFIFO (First In First Out — первым прибыл — первым обслужен), «вторая попытка» и LRU (Least Recently Used — с наиболее давним использованием). Одно приятное отличие кэширования от страничной организации памяти состоит в том, что обращения к кэшу производятся относительно нечасто, что позволяет хранить все блоки в точном LRU-порядке со связными списками.

К сожалению, здесь есть одна загвоздка. Теперь, когда мы можем реализовать точное выполнение алгоритма LRU, оказывается, что алгоритм LRU является нежелательным. Вызвано это тем, что буквальное применение алгоритма LRU снижает надежность файловой системы и угрожает ее непротиворечивости (обсуждавшейся в предыдущем разделе). Если в кэш считывается и модифицируется критический блок, например блок i-узла, но не записывается тут же на диск, то компьютерный сбой может привести к тому, что файловая система окажется в противоречивом состоянии. Если блок i-узла поместить в конец цепочки LRU, то может пройти довольно много времени, прежде чем этот блок попадет в ее начало и будет записан на диск.

Рис. 37

Поскольку большинство современных процессоров не предоставляют соответствующей аппаратной поддержки для реализации алгоритма *LRU*, хотелось бы иметь алгоритм, достаточно близкий к *LRU*, но не требующий специальной поддержки. Программная реализация алгоритма, близкого к *LRU*, - алгоритм NFU (Not Frequently Used). Для него требуются программные счетчики, по одному на каждую страницу, которые сначала равны нулю. При каждом прерывании по времени (а не после каждой инструкции) операционная система сканирует все страницы в памяти и у каждой страницы с установленным флагом обращения увеличивает на единицу значение счетчика, а флаг обращения сбрасывает. Таким образом, кандидатом на освобождение оказывается страница с наименьшим значением счетчика, как страница, к которой реже всего обращались. Главный недостаток алгоритма NFU состоит в том, что он ничего не забывает. Например, страница, к которой очень часто обращались в течение некоторого времени, а потом обращаться перестали, все равно не будет удалена из памяти, потому что ее счетчик содержит большую величину. Например, в многопроходных компиляторах страницы, которые активно использовались во время первого прохода, могут надолго сохранить большие значения счетчика, мешая загрузке полезных в дальнейшем страниц. К счастью, возможна небольшая модификация алгоритма, которая позволяет ему «забывать». Достаточно, чтобы при каждом прерывании по времени содержимое счетчика сдвигалось вправо на 1 бит, а уже затем производилось бы его увеличение для страниц с установленным флагом обращения. Другим,

уже более устойчивым недостатком алгоритма является длительность процесса сканирования таблиц страниц.

Рис. 38

Замещение страниц по запросу – когда страницы загружаются по требованию, а не заранее, т. е. процесс прерывается и ждет загрузки страницы.

Буксование – когда каждую следующую страницу приходится процессу загружать в память.

Чтобы не происходило частых прерываний, желательно чтобы часто запрашиваемые страницы загружались заранее, а остальные подгружались по необходимости.

Рабочий набор – множество страниц (к), которое процесс использовал до момента времени (t). Т. е. можно записать функцию w(k,t).

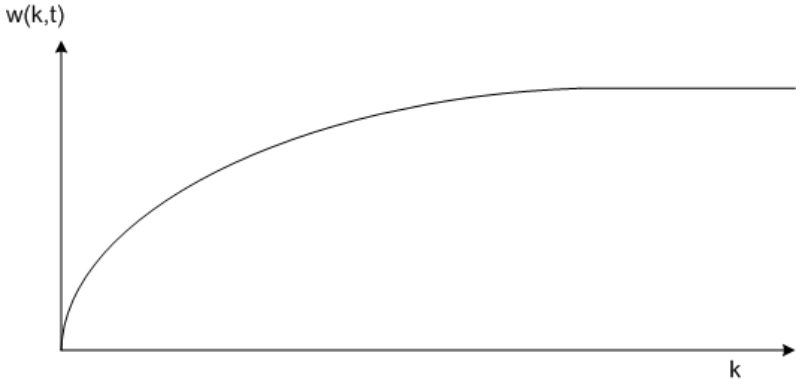

Рис. 39 Зависимость рабочего набора w(k,t)
от количества запрошенных страниц

Рабочий набор выходит в насыщение, значение w(k,t) в режиме насыщения может служить для рабочего набора, который необходимо загружать до запуска процесса.

Алгоритм заключается в том, чтобы определить рабочий набор, найти и выгрузить страницу, которая не входит в рабочий набор.

Этот алгоритм можно реализовать, записывая при каждом обращении к памяти, номер страницы в специальный сдвигающийся регистр, затем удалялись бы дублирующие страницы. Но это дорого.

В принципе можно использовать множество страниц, к которым обращался процесс за последние **t** секунд.

Текущее виртуальное время (Tv) – время работы процессора, которое реально использовал процесс.

Время последнего использования (Told) – текущее время при R=1, т.е. все страницы проверяются на R=1, и если да, то текущее время записывается в это поле.

Теперь можно вычислить возраст страницы (не обновления) **Tv-Told,** и сравнить с **t**, если больше, то страница не входит в рабочий набор, и страницу можно выгружать.

Получается три варианта:

- если R=1, то текущее время запоминается в поле время последнего использования;
- если R=0 и возраст > t, то страница удаляется;

- если R=0 и возраст =< t, то эта страница входит в рабочий набор.

Резюме

Алгоритм	Комментарий
Оптимальный	Не осуществим
NRU	Очень грубый
FIFO	Может выгрузить нужные страницы
Вторая попытка	Усовершенствованный FIFO
Часы	Реалистичный
LRU	Отличный алгоритм, но его сложно осуществить целиком
NFU	Довольно грубое приближение LRU
Старение	Эффективный, хорошее улучшение LRU
Рабочий набор	Немного дорог для реализации
WSClock	Хороший рациональный алгоритм

Рис. 40

Кэш

Рис. 41

Память вычислительной машины представляет собой иерархию запоминающих устройств (ЗУ), отличающихся средним временем доступа к данным, объемом и стоимостью хранения одного бита. Фундаментом этой пирамиды запоминающих устройств служит внешняя память, как правило, представляемая жестким диском. Она имеет большой объем (десятки и сотни гигабайт), но скорость доступа к данным является невысокой. Время доступа к диску измеряется миллисекундами.

На следующем уровне располагается более быстродействующая (время доступа равно примерно 10-20 наносекундам) и менее объемная (от десятков мегабайт до нескольких гигабайт) оперативная память, реализуемая на относительно медленной динамической памяти DRAM.

Для хранения данных, к которым необходимо обеспечить быстрый доступ, используются компактные быстродействующие запоминающие устройства на основе статической памяти SRAM, объем которых составляет от нескольких десятков до нескольких сотен килобайт, а время доступа к данным обычно не превышает 8.

Все перечисленные характеристики ЗУ быстро изменяются по мере совершенствования вычислительной аппаратуры. В данном случае важны не абсолютные значения времени доступа или объема памяти, а их соотношение для разных типов запоминающих устройств.

И, наконец, верхушку в этой пирамиде составляют внутренние регистры процессора, которые также могут быть использованы для промежуточного хранения данных. Общий объем регистров составляет несколько десятков байт, а время доступа определяется быстродействием процессора и равно в настоящее время примерно 2-3.

Таким образом, можно констатировать печальную закономерность — чем больше объем устройства, тем менее быстродействующим оно является. Более того, стоимость хранения данных в расчете на один бит также увеличивается с ростом быстродействия устройств. Однако пользователю хотелось бы иметь и недорогую, и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Рис. 42

Рассмотрим одну из возможных схем кэширования. Содержимое кэш-памяти представляет собой совокупность *записей* обо всех загруженных в нее элементах данных из основной памяти. Каждая запись об элементе данных включает в себя:

- Q значение элемента данных;
- Q адрес, который этот элемент данных имеет в основной памяти;
- Q дополнительную информацию, которая используется для реализации алгоритма замещения данных в кэше и обычно включает признак модификации и признак действительности данных.

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти с целью определения, не находятся ли там нужные данные. Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — по взятому из запроса значению поля адреса в оперативной памяти. Далее возможен один из двух вариантов развития событий:

Q, если данные обнаруживаются в кэш-памяти, то есть произошло *кэш-попадание (cache-hit),* они считываются из нее и результат передается источнику запроса;

Ц, если нужные данные отсутствуют в кэш-памяти, то есть произошел *кэш-промах (cache-miss),* они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память. Интуитивно понятно, что эффективность кэширования зависит от вероятности попадания в кэш. Покажем это путем нахождения зависимости среднего времени доступа к основной памяти от вероятности кэш-попаданий. Пусть имеется основное запоминающее устройство со средним временем доступа к данным tl и кэш-память, имеющая время доступа t2, очевидно, что t2<tl. Пусть t – среднее время доступа к данным в системе с кэш-памятью, ар – вероятность кэш-попадания. По формуле полной вероятности имеем:

t * 1Д1 - р) + t2p = (t2 -t)p +* t,

Среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности попадания в кэш и изменяется от среднего времени доступа в основное запоминающее устройство tl при p=0 до среднего времени доступа непосредственно в кэш-память t2 при p=1. Отсюда видно, что использование кэш-памяти имеет смысл только при высокой вероятности кэш-попадания.

Вероятность обнаружения данных в кэше зависит от разных факторов, таких, например, как объем кэша, объем кэшируемой памяти, алгоритм замещения данных в кэше, особенности выполняемой программы, время ее работы, уровень мультипрограммирования и других особенностей вычислительного процесса. Тем не менее в большинстве реализаций кэш-памяти процент кэш-попаданий оказывается весьма высоким — свыше 90%. Такое высокое значение вероятности нахождения данных в кэш-памяти объясняется наличием у данных объективных свойств: пространственной и временной локальности.

Временная локальность	Пространственная локальность
Если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время.	Если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Сквозная запись	Обратная запись
При каждом запросе к основной памяти, в том числе и при записи, просматривается кэш. Если данные по запрашиваемому адресу отсутствуют, то запись выполняется только в основную память. Иначе запись выполняется одновременно в кэш и основную память.	Аналогично при возникновении запроса к памяти выполняется просмотр кэша, и если запрашиваемых данных там нет, то запись выполняется только в основную память. Иначе запись производится ***только в кэш-память***, при этом в описателе данных делается специальная отметка (признак модификации), указывающая на то, что при вытеснении этих данных из кэша необходимо переписать их в основную память.

Рис. 43

Q *Временная локальность.* Если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время.

Q *Пространственная локальность.* Если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Именно основываясь на свойстве временной локальности, данные, только что считанные из основной памяти, размещают в запоминающем устройстве быстрого доступа, предполагая, что скоро они опять понадобятся. В начале работы системы, когда кэш-память еще пуста, почти каждый запрос к основной памяти выполняется «по полной программе»: просмотр кэша, констатация промаха, чтение данных из основной памяти, передача результата источнику запроса и копирование данных в кэш. Затем, по мере заполнения кэша, в полном соответствии со свойством временной локальности возрастает вероятность обращения к данным, которые уже были использованы на предыдущем этапе работы системы, то есть к данным, которые содержатся в кэше и могут быть считаны значительно быстрее, чем из основной памяти.

Свойство пространственной локальности также используется для увеличения вероятности кэш-попадания: как правило, в кэш-память считывается не один информационный элемент, к которому произошло обращение, а целый блок данных, расположенных в основной памяти в непосредственной близости с данным элементом. Поскольку при выполнении программы очень высока вероятность, что команды выбираются из памяти последовательно одна за другой из соседних ячеек, то имеет смысл загружать в

кэш-память целый фрагмент программы. Аналогично, если программа ведет обработку некоторого массива данных, то ее работу можно ускорить, загрузив в кэш-часть или даже весь массив данных. При этом учитывается высокая вероятность того, что значительное число обращений к памяти будет выполняться к адресам массива данных.

Проблема согласования данных

В процессе работы содержимое кэш-памяти постоянно обновляется, а значит, время от времени данные из нее должны вытесняться. Вытеснение означает либо простое объявление свободной соответствующей области кэш-памяти (сброс бита действительности), если вытесняемые данные за время нахождения в кэше не были изменены, либо в дополнение к этому копирование данных в основную память, если они были модифицированы. Алгоритм замены данных в кэш-памяти существенно влияет на ее эффективность. В идеале такой алгоритм должен, во-первых, быть максимально быстрым, чтобы не замедлять работу кэш-памяти, а, во-вторых, обеспечивать максимально возможную вероятность кэш-попаданий. Поскольку из-за непредсказуемости вычислительного процесса ни один алгоритм замещения данных в кэш-памяти не может гарантировать оптимальный результат, разработчики ограничиваются рациональными решениями, которые по крайней мере, не сильно замедляют работу кэша — запоминающего устройства, изначально призванного быть быстрым.

Наличие в компьютере двух копий данных – в основной памяти и в кэше – порождает проблему согласования данных. Если происходит запись в основную память по некоторому адресу, а содержимое этой ячейки находится в кэше, то в результате соответствующая запись в кэше становится недостоверной. Рассмотрим два подхода к решению этой проблемы:

- Q *Сквозная запись (write through).* При каждом запросе к основной памяти, в том числе и при записи, просматривается кэш. Если данные по запрашиваемому адресу отсутствуют, то запись выполняется только в основную память. Если же данные, к которым выполняется обращение, находятся в кэше, то запись выполняется одновременно в кэш и основную память.
- Q *Обратная запись (write back).* Аналогично при возникновении запроса к памяти выполняется просмотр кэша, и если запрашиваемых данных там нет, то запись выполняется только в основную память. В противном же случае запись производится *только в кэш-память,* при этом в описателе данных делается специальная отметка (признак модификации), которая указывает на то, что при вытеснении этих данных из кэша необходимо переписать их в основную память, чтобы актуализировать устаревшее содержимое основной памяти.

В некоторых алгоритмах замещения предусматривается первоочередная выгрузка модифицированных, или, как еще говорят, «грязных» данных.

Модифицированные данные могут выгружаться не только при освобождении места в кэш-памяти для новых данных, но и в «фоновом режиме», когда система не очень загружена.

Рис. 44

Алгоритм поиска и алгоритм замещения данных в кэше непосредственно зависят от того, каким образом основная память отображается на кэш-память. Принцип прозрачности требует, чтобы правило отображения основной памяти на кэш-память не зависело от работы программ и пользователей. При кэшировании данных из оперативной памяти широко используются две основные схемы отображения: случайное отображение и детерминированное отображение.

При *случайном* отображении элемент оперативной памяти в общем случае может быть размещен в произвольном месте кэш-памяти. Для того, чтобы в дальнейшем можно было найти нужные данные в кэше, они помещаются туда вместе со своим адресом, то есть тем адресом, который данные, которые имеются в оперативной памяти. При каждом запросе к оперативной памяти выполняется поиск в кэше, причем критерием поиска выступает адрес оперативной памяти из запроса. Очевидная схема простого перебора для поиска нужных данных в случае кэша оказывается непригодной из-за недопустимо больших временных затрат.

Рис. 45

Для кэшей со случайным отображением используется так называемый *ассоциативный поиск,* при котором сравнение выполняется не последовательно с каждой записью кэша, а параллельно со всеми его записями (рис). Признак, по которому выполняется сравнение, называется *тегом (tag).* В данном случае тегом является адрес данных в оперативной памяти. Электронная реализация такой схемы приводит к удорожанию памяти, причем стоимость существенно возрастает с увеличением объема запоминающего устройства. Поэтому ассоциативная кэш-память используется в тех случаях, когда для обеспечения высокого процента попадания достаточно небольшого объема памяти.

В кэшах, построенных на основе случайного отображения, вытеснение старых данных происходит только в том случае, когда вся кэш-память заполнена, и нет свободного места. Выбор данных на выгрузку осуществляется среди всех записей кэша. Обычно этот выбор основывается на тех же приемах, что и в алгоритмах замещения страниц, например выгрузка, данных, к которым дольше всего не было обращений, или данных, к которым было меньше всего обращений.

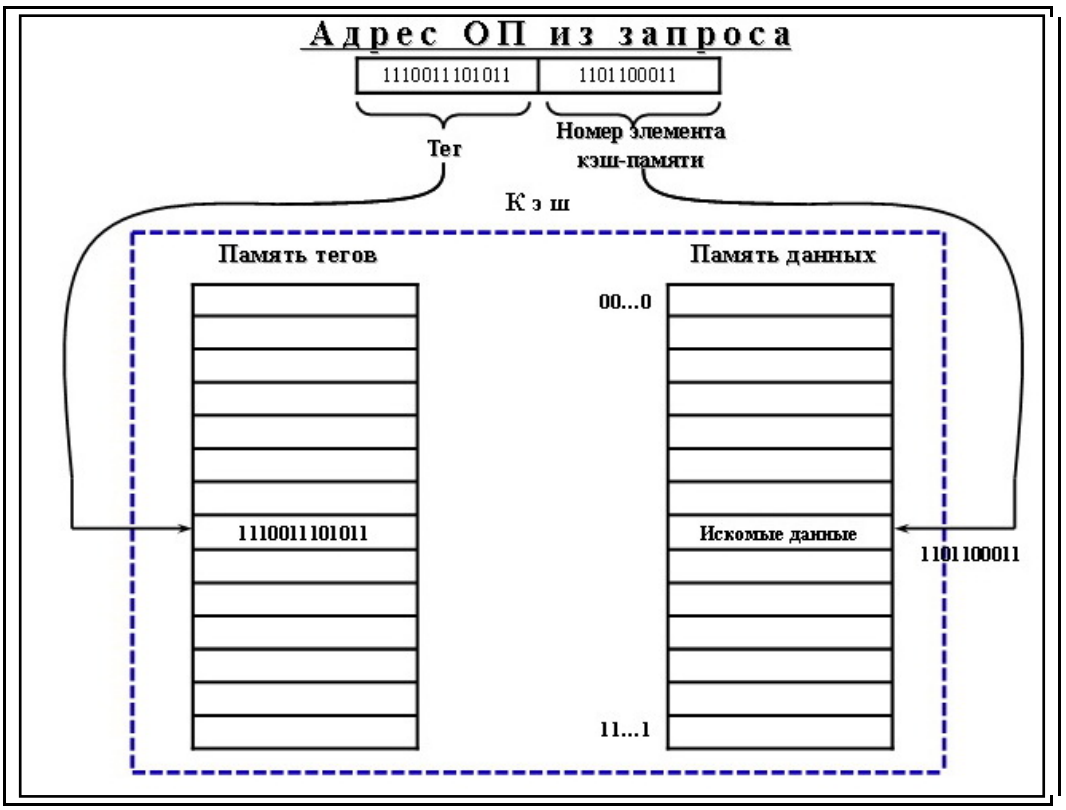

Рис. 46

Второй, *детерминированный* способ отображения предполагает, что любой элемент основной памяти всегда отображается в одно и то же место кэш-памяти. В этом случае кэш-память разделена на строки, каждая из которых предназначена для хранения одной записи об одном элементе данных и имеет свой номер. Между номерами строк кэш-памяти и адресами оперативной памяти устанавливается соответствие «один ко многим»: одному номеру строки соответствует несколько адресов оперативной памяти.

В качестве отображающей функции может использоваться простое выделение нескольких разрядов из адреса оперативной памяти, которые интерпретируются как номер строки кэш-памяти (такое отображение называется *прямым)*. Например, пусть в кэш-памяти может храниться 1024 записи, то есть кэш имеет 1024 строки, пронумерованные от 0 до 1023. Тогда любой адрес оперативной памяти может быть отображен на адрес кэш-памяти простым отделением 10 двоичных разрядов.

В действительности запись в кэше обычно содержит несколько элементов данных. При поиске данных в кэше используется быстрый прямой доступ к записи по номеру строки, полученному путем обработки адреса оперативной памяти из запроса. Однако поскольку в найденной строке могут находиться данные из любой ячейки оперативной памяти, младшие разряды адреса которой совпадают с номером строки, необходимо выполнить дополнительную проверку. Для этих целей каждая строка кэш-памяти дополняется тегом, содержащим старшую часть адреса данных в оперативной памяти. При

совпадении тега с соответствующей частью адреса из запроса констатируется кэш-попадание.

Если же произошел кэш-промах, то данные считываются из оперативной памяти и копируются в кэш. Если строка кэш-памяти, в которую должен быть скопирован элемент данных из оперативной памяти, содержит другие данные, то последние вытесняются из кэша. Заметим, что процесс замещения данных в кэш-памяти на основе прямого отображения существенно отличается от процесса замещения данных в кэш-памяти со случайным отображением. Во-первых, вытеснение данных происходит не только в случае отсутствия свободного места в кэше, во-вторых, никакого выбора данных на замещение не существует.

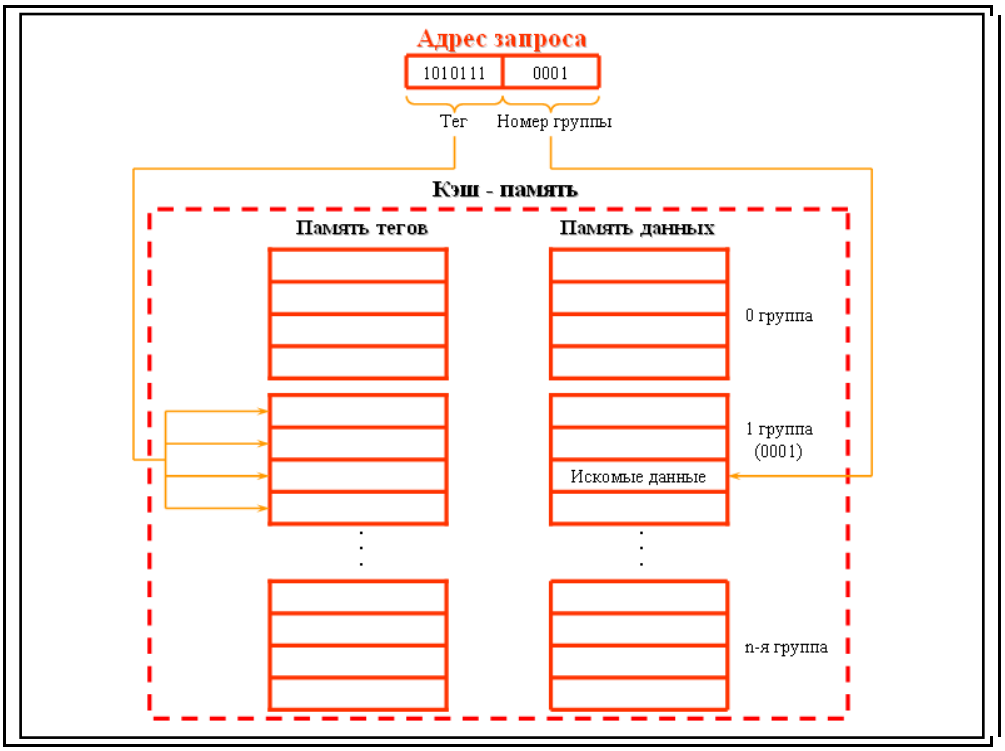

Рис. 47

Во многих современных процессорах кэш-память строится на основе сочетания этих двух подходов, что позволяет найти компромисс между сравнительно низкой стоимостью кэша с прямым отображением и интеллектуальностью алгоритмов замещения в кэше со случайным отображением. При смешанном подходе произвольный адрес оперативной памяти отображается не на один адрес кэш-памяти (как это характерно для прямого отображения) и не на любой адрес кэш-памяти (как это делается при случайном отображении), а на некоторую группу адресов. Все группы пронумерованы. Поиск в кэше осуществляется вначале по номеру группы, полученному из адреса оперативной памяти из запроса, а затем в пределах группы путем ассоциативного просмотра всех записей в группе на предмет совпадения старших частей адресов оперативной памяти.

При промахе данные копируются по любому свободному адресу из однозначно заданной группы. Если свободных адресов в группе нет, то выполняется вытеснение данных. Поскольку кандидатов на выгрузку несколько – все записи из данной группы – алгоритм замещения может учесть интенсивность обращений к данным и тем самым повысить вероятность попаданий в будущем. Таким образом, в данном способе комбинируется прямое отображение на группу и случайное отображение в пределах группы.

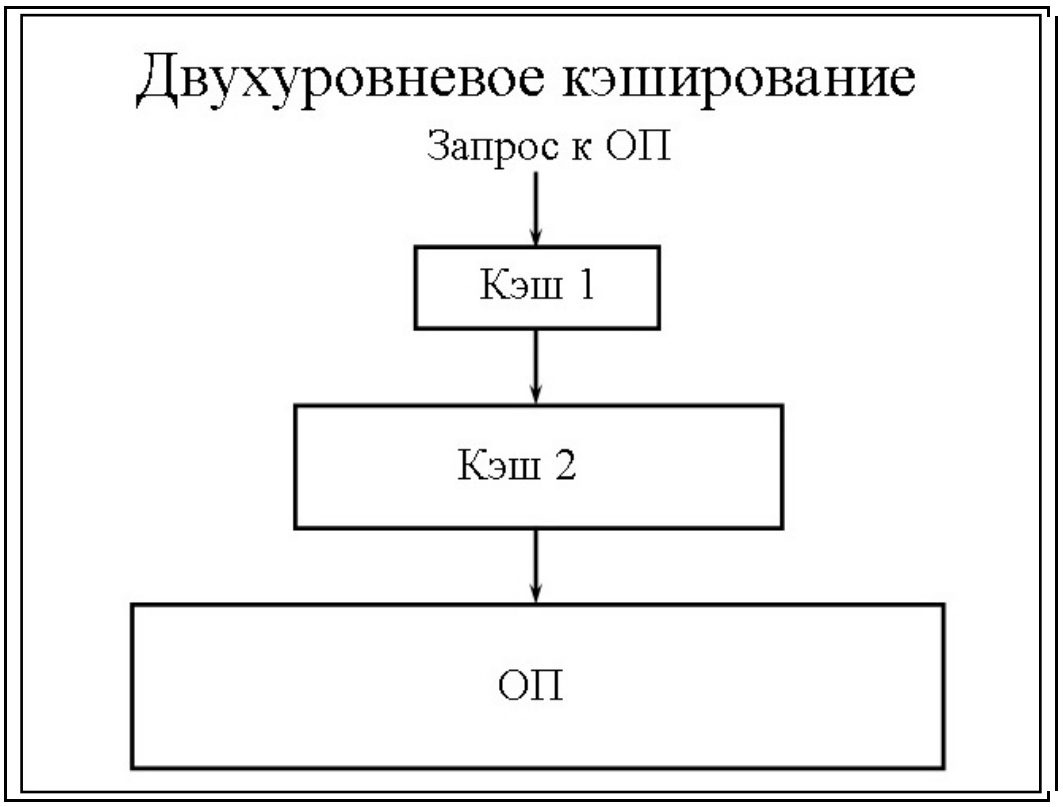

Рис. 48

При выполнении запросов к оперативной памяти во многих вычислительных системах используется двухуровневое кэширование. Кэш первого уровня имеет меньший объем и более высокое быстродействие, чем кэш второго уровня. Кэш второго уровня играет роль основной памяти по отношению к кэшу первого уровня.

Показана схема выполнения запроса на чтение в системе с двухуровневым кэшем. Сначала делается попытка обнаружить данные в кэше первого уровня. Если произошел промах, поиск продолжается в кэше второго уровня. Если же нужные данные отсутствуют и здесь, тогда происходит считывание данных из основной памяти. Понятно, что время доступа к данным оказывается минимальным, когда кэш-попадание происходит уже на первом уровне, несколько большим — при обнаружении данных на втором уровне и обычным временем доступа к оперативной памяти, если нужных данных нет ни в том, ни в другом кэше. При считывании данных из оперативной памяти происходит их

копирование в кэш второго уровня, а если данные считываются из кэша второго уровня, то они копируются в кэш первого уровня.

При работе такой иерархической организованной памяти необходимо обеспечить непротиворечивость данных на всех уровнях. Кэши разных уровней могут согласовывать данные разными способами. Пусть, например, кэш первого уровня использует сквозную запись, а кэш второго уровня — обратную запись.

Если данные обнаружены в кэше первого уровня, то вступает в силу алгоритм сквозной записи: выполняется запись в кэш первого уровня и передается запрос на запись в кэш второго уровня, играющий в данном случае роль основной памяти. Запись в кэш второго уровня в соответствии с алгоритмом обратной записи, принятом на данном уровне, сопровождается установкой признака модификации, при этом никакой записи в оперативную память не производится.

Если данные найдены в кэше второго уровня, то, так же как и в предыдущем случае, выполняется запись в этот кэш и устанавливается признак модификации.

Рассмотренные в данном разделе проблемы кэширования охватывают только такой класс систем организации памяти, в котором на каждом уровне имеется одно кэширующее устройство. Существует и другой класс систем памяти, главной отличительной особенностью которого является наличие нескольких кэшей одного уровня. Этот вариант характерен для распределенных систем обработки информации — мультипроцессорных компьютеров и компьютерных сетей.

Содержание

Учебное издание

ОПЕРАЦИОННЫЕ СИСТЕМЫ:
УПРАВЛЕНИЕ ПАМЯТЬЮ
Методические указания

Составитель: МАКАРОВ Павел Сергеевич

Редактор М. В. Штаева

Подписано в печать 29.12.2008. Формат 60×84/16.
Усл. печ. л. 3,95. Тираж 100 экз.

Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, д. 32.

Типография УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32.