Архитектура операционных систем

Операционные системы,

Современный компьютер – сложнейшая аппаратно-программная система.

Написание программ для компьютера, их отладка и последующее выполнение представляет собой сложную трудоемкую задачу.

Основная причина этого – огромная разница между тем, что удобно для людей, и тем, что удобно для компьютеров.

Компьютер понимает только свой, машинный язык (назовем его Я0), а для человека наиболее удобен разговорный или хотя бы язык описания алгоритмов – алгоритмический язык.

Проблему можно решить двумя способами.

Оба способа связаны с разработкой команд, которые были бы более удобны для человека, чем встроенные машинные команды компьютера.

Эти новые команды в совокупности формируют некоторый язык, который назовем Я1.

Упомянутые два способа решения проблемы различаются тем, каким образом компьютер будет выполнять программы, написанные на языке Я1.

Первый способ – замена каждой команды языка Я1 на эквивалентный набор команд в языке Я0. В этом случае компьютер выполняет новую программу, написанную на языке Я0, вместо программы, написанной на языке Я1.

Эта технология называется трансляцией.

Второй способ – написание программы на языке Я0, которая берет программы, написанные на языке Я1, в качестве входных данных, рассматривает каждую команду по очереди и сразу выполняет эквивалентный набор команд языка Я0.

Эта технология не требует составления новой программы на Я0.

Она называется интерпретацией, а программа, которая осуществляет интерпретацию, **называется интерпретатором**.

В подобной ситуации проще представить себе существование гипотетического компьютера или виртуальной машины, для которой машинным языком является язык Я1, чем думать о трансляции и интерпретации.

Назовем такую виртуальную машину М1, а виртуальную машину с языком Я0 – М0. Для виртуальных машин можно будет писать программы, как будто они (машины) действительно существуют.

Очевидно, можно пойти дальше – создать еще набор команд, который в большей степени ориентирован на человека и в меньшей степени на компьютер, чем Я1.

Этот набор формирует язык Я2 и, соответственно, виртуальную машину М2. Так можно продолжать до тех пор, пока не дойдем до подходящего нам языка уровня n.

Большинство современных компьютеров состоит из двух и более уровней.

Уровень 0 – аппаратное обеспечение машины. Электронные схемы этого уровня выполняют программы, написанные на языке уровня 1.

Следующий уровень – микроархитектурный уровень.

На этом уровне можно видеть совокупности 8 или 32 (иногда и больше) регистров, которые формируют локальную память и АЛУ (арифметико-логическое устройство).

Следующий (второй) уровень составляет уровень архитектуры системы команд. Команды используют регистры и другие возможности аппаратуры. Команды формируют уровень ISA (Instruction Set Architecture), называемый машинным языком.

Следующий (третий) уровень обычно – гибридный.

Четвертый уровень представляет собой символическую форму одного из языков низкого уровня (обычно ассемблер).

Уровни с пятого и выше предназначены для прикладных программистов, решающих конкретные задачи на языках высокого уровня (C, C++, C#, VBA и др.).

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

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

Операционная система, среда и операционная оболочка

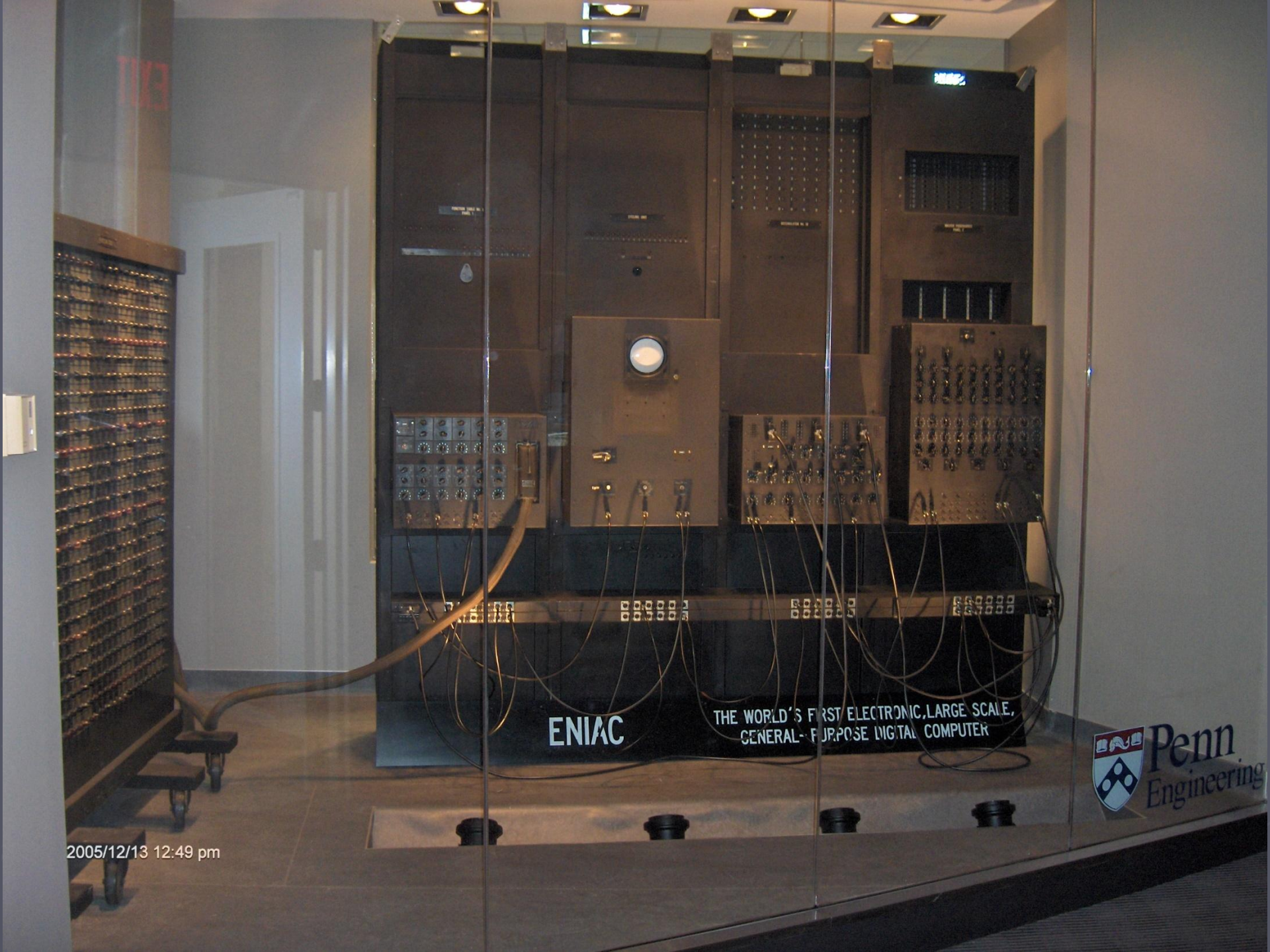
Операционные системы (ОС) в современном их понимании (их назначении и сущности) появились значительно позже первых компьютеров (правда, по всей видимости, и исчезнут в этой сущности в компьютерах будущего).

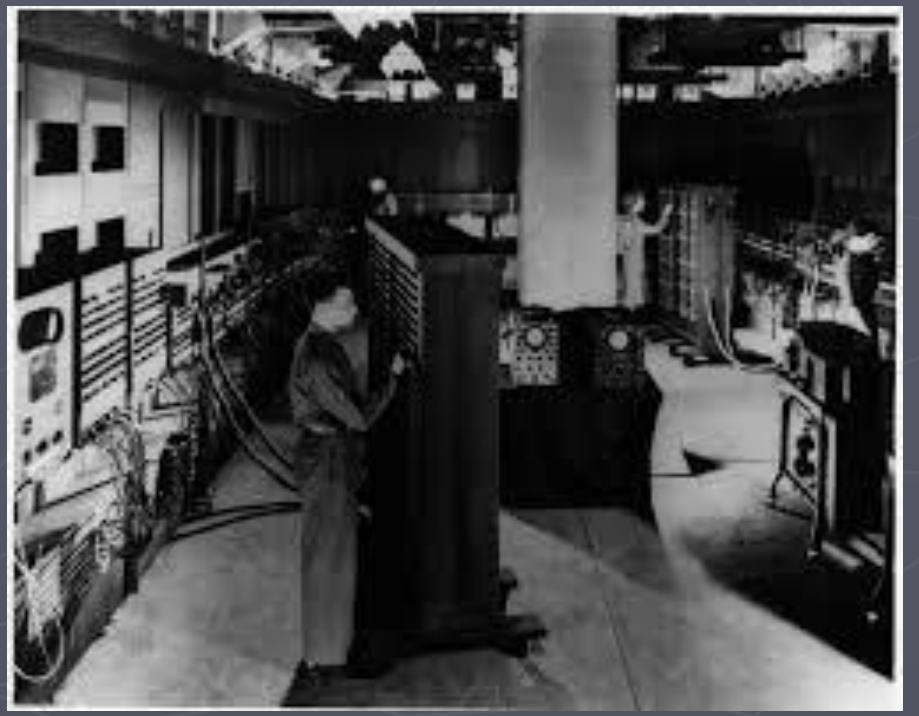
Почему и когда появились ОС?

Считается1 что первая цифровая вычислительная машина ENIAC (Electronic Numerical Integrator and Computer) была создана в 1946 году по проекту "Проект PX" Министерства обороны США.

На реализацию проекта затрачено 500 тыс. долларов. Компьютер содержал 18000 электронных ламп, массу всякой электроники, включал в себя 12 десятиразрядных сумматоров, а для ускорения некоторых арифметических операций имел умножитель и "делитель-извлекатель" квадратного корня.

Программирование сводилось к связыванию различных блоков проводами. Конечно, никакого программного обеспечения и тем более операционных систем тогда еще не существовало.


ENIAC
THE WORLD'S FIRST ELECTRONIC, LARGE SCALE,
GENERAL-PURPOSE DIGITAL COMPUTER
Penn
Engineering
2005/12/13 12:49 pm



Интенсивное создание различных моделей ЭВМ относится к началу 50-х годов прошлого века.

В эти годы одни и те же группы людей участвовали и в проектировании, и в создании, и в программировании, и в эксплуатации ЭВМ.

Программирование осуществлялось исключительно на машинном языке (а затем на ассемблере), не было никакого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

Операционные системы еще не появились, а все задачи организации вычислительного процесса решались вручную каждым программистом с примитивного пульта управления ЭВМ.

С появлением полупроводниковых элементов вычислительные возможности компьютеров существенно выросли.

Выполнение программ усложнилось и включало в себя следующие основные действия:

загрузка нужного транслятора (и др.);

запуск транслятора и получение программы в машинных кодах;

связывание программы с библиотечными подпрограммами;

загрузка программы в оперативную память;

запуск программы;

вывод результатов работы программы на печатающее или другое периферийное устройство.

Для организации эффективной загрузки всех средств компьютера в штаты вычислительных центров ввели должности специально обученных операторов.

Считается, что первую операционную систему создала в 1952 году для своих компьютеров IBM-701 исследовательская лаборатория фирмы General Motors.

В 1955 году эта фирма и North American Aviation совместно разработали ОС для компьютера IBM-704.

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

пакетная обработка одного потока задач;

наличие стандартных программ ввода-вывода;

возможности автоматического перехода от программы к программе;

средства восстановления после ошибок;

языки управления заданиями, предоставляющие пользователям возможность описывать свои задания и ресурсы, требуемые для их выполнения.

Иерархическая структура программно-аппаратных средств компьютера

Операционная система предназначена для того, чтобы скрыть все эти сложности. Конечный пользователь обычно не интересуется деталями устройства аппаратного обеспечения компьютера. Компьютер ему видится как набор приложений. Приложение может быть написано программистом на каком-либо языке программирования.

Операционная система выступает в роли посредника, облегчая программисту, пользователям и программным приложениям доступ к различным службам и возможностям компьютера.

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

Образно можно сказать, что аппаратура компьютера предоставляет "сырую" вычислительную мощность, а задача операционной системы заключается в том, чтобы сделать использование этой вычислительной мощности доступным и по возможности удобным для пользователя.

Таким образом, операционная среда – это программная среда, образуемая операционной системой, определяющая интерфейс прикладного программирования (API) как множество системных функций и сервисов (системных вызовов), которые предоставляются прикладным программам.

Еще одно важное понятие, связанное с операционной системой, относится к реализации пользовательских интерфейсов.

Как правило, любая операционная система обеспечивает удобную работу пользователя за счет средств пользовательского интерфейса.

В общем случае под оболочкой операционной системы понимается часть операционной среды, определяющая интерфейс пользователя, его реализацию (текстовый, графический и т.п.), командные и сервисные возможности пользователя по управлению прикладными программами и компьютером.

Перейдем к рассмотрению эволюции операционных систем.

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

Существуют также различные подходы к определению поколений ОС. Известно разделение ОС на поколения в соответствии с поколениями вычислительных машин и систем.

Другая точка зрения не связывает поколение ОС с соответствующими поколениями ЭВМ. Так, например, известно определение поколений ОС по уровням входного языка ЭВМ, режимам использования центральных процессоров, формам эксплуатации систем и т.п.

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

Первым этапом развития системного программного обеспечения можно считать использование библиотечных программ. Концепция библиотек подпрограмм является наиболее ранней и восходит к 1949 году. Эти средства применялись в ЭВМ первого поколения, когда операционных систем как таковых еще не существовало.

Возникли специальные программы методов доступа. Таким образом было создано базовое системное программное обеспечение.

С улучшением характеристик ЭВМ и ростом их производительности стало ясно, что существующего базового программного обеспечения (ПО) недостаточно. Появились операционные системы ранней пакетной обработки – мониторы.

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

Появление новых аппаратных разработок (1959-1963 гг.) – систем прерываний, таймеров, каналов – стимулировало дальнейшее развитие ОС. Возникли исполнительные системы, которые представляли собой набор программ для распределения ресурсов ЭВМ, связей с оператором, управления вычислительным процессом и управления вводом-выводом. Такие исполнительные системы позволили реализовать довольно эффективную по тому времени форму эксплуатации вычислительной системы – однопрограммную пакетную обработку.

Однако однопрограммная пакетная обработка с ростом производительности ЭВМ не могла обеспечить экономически приемлемый уровень эксплуатации машин. Решением стало мультипрограммирование – способ организации вычислительного процесса, при котором в памяти компьютера находится несколько программ, попеременно выполняющихся одним процессором.

Теория построения операционных систем в этот период обогатилась рядом плодотворных идей. Появились различные формы мультипрограммных режимов работы, в том числе разделение времени – режим, обеспечивающий работу многотерминальной системы. Режим разделения времени позволил пользователю интерактивно взаимодействовать со своими программами.

Одной из первых ОС, использующих эти новейшие решения, была операционная система MCP (главная управляющая программа), созданная фирмой Burroughs для своих компьютеров B5000 в 1963 году.

В этой ОС были реализованы многие концепции и идеи, ставшие впоследствии стандартными для многих операционных систем:

мультипрограммирование;

мультипроцессорная обработка;

виртуальная память;

возможность отладки программ на исходном языке;

написание операционной системы на языке высокого уровня.

CTSS (Compatible Time Sharing System) – совместимая система разделения времени, разработанная в Массачусетском технологическом институте (1963 год) для компьютера IBM-7094.

Эта система была использована для разработки в этом же институте совместно с Bell Labs и General Electric системы разделения времени следующего поколения MULTICS (Multiplexed Information And Computing Service).

Одним из важнейших событий в истории операционных систем считается появление в 1964 году семейства компьютеров под названием System/360 фирмы IBM, а позже – System/370.

Это было первой в мире реализацией концепции семейства программно и информационно совместимых компьютеров.

ОС добавились модули, реализующие протоколы связи.

Операционная система становится "неотъемлемой частью ЭВМ".

В процессорах появился привилегированный режимы работы, мощная система прерываний, защита памяти, специальные регистры для быстрого переключения программ, средства поддержки виртуальной памяти и др.

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

Сюда относятся:

диагностические программы для выявления ошибок в конфигурации ОС;

средства восстановления последней работоспособной конфигурации;

средства восстановления поврежденных и пропавших системных файлов и др.

Следует отметить еще одно назначение ОС.

Возможность развития. Современные ОС организуются таким образом, что допускают эффективную разработку, тестирование и внедрение новых системных функций, не прерывая процесса нормального функционирования вычислительной системы.

Большинство операционных систем постоянно развиваются (нагляден пример Windows). Происходит это в силу следующих причин.

Обновление и возникновение новых видов аппаратного обеспечения.

Новые сервисы. Для удовлетворения пользователей или нужд системных администраторов ОС должны постоянно предоставлять новые возможности.

Исправления. В каждой ОС есть ошибки. Время от времени они обнаруживаются и исправляются. Важную роль играет хорошая и полная документированность системы.

Перейдем к рассмотрению состава компонентов и функций ОС.

Современные операционные системы содержат сотни и тысячи модулей (например, Windows 2000 содержит 29 млн строк исходного кода на языке С).

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

Управление процессами. Подсистема управления процессами непосредственно влияет на функционирование вычислительной системы. Для каждой выполняемой программы ОС организует один или более процессов.

Каждый такой процесс представляется в ОС информационной структурой (таблицей, дескриптором, контекстом процессора), содержащей данные о потребностях процесса в ресурсах, а также о фактически выделенных ему ресурсах (область оперативной памяти, количество процессорного времени, файлы, устройства ввода-вывода и др.).

В современных мультипрограммных ОС может существовать одновременно несколько процессов, порожденных по инициативе пользователей и их приложений, а также инициированных ОС для выполнения своих функций (системные процессы).

Управление памятью. Подсистема управления памятью производит распределение физической памяти между всеми существующими в системе процессами, загрузку и удаление программных кодов и данных процессов в отведенные им области памяти, настройку адресно-зависимых частей кодов процесса на физические адреса выделенной области, а также защиту областей памяти каждого процесса.

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

Важная функция управления памятью – защита памяти.

Нарушения защиты памяти связаны с обращениями процессов к участкам памяти, выделенной другим процессам прикладных программ или программ самой ОС.

Управление файлами.

Функции управления файлами сосредоточены в файловой системе ОС.

Операционная система виртуализирует отдельный набор данных, хранящихся на внешнем накопителе, в виде файла – простой неструктурированной последовательности байтов, имеющих символьное имя.

Управление внешними устройствами.

Функции управления внешними устройствами возлагаются на подсистему управления внешними устройствами, называемую также подсистемой ввода-вывода.

Она является интерфейсом между ядром компьютера и всеми подключенными к нему устройствами.

Программа, управляющая конкретной моделью внешнего устройства и учитывающая все его особенности, называется **драйвером**.

Наличие большого количества подходящих драйверов во многом определяет успех ОС на рынке. ОС должна поддерживать четко определенный интерфейс между драйверами и остальными частями ОС.

Защита данных и администрирование.

Безопасность данных вычислительной системы обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок программного обеспечения, а также средствами защиты от несанкционированного доступа.

Важным средством защиты являются функции аудита ОС, заключающегося в фиксации всех событий, от которых зависит безопасность системы.

Интерфейс прикладного программирования.

Прикладные программисты используют в своих приложениях обращения к операционной системе, когда для выполнения тех или иных действий им требуется особый статус, которым обладает только ОС. Возможности операционной системы доступны программисту в виде набора функций, который называется **интерфейсом прикладного программирования (Application Programming Interface, API).**

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

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

Пользовательский интерфейс. ОС обеспечивает удобный интерфейс не только для прикладных программ, но и для пользователя (программиста, администратора).

Современные ОС поддерживают развитые функции пользовательского интерфейса для интерактивной работы за терминалами двух типов: алфавитно-цифрового и графического.

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

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

Ввод команд может быть упрощен, если операционная система поддерживает графический пользовательский интерфейс.

Архитектура операционной системы

Под архитектурой операционной системы понимают структурную и функциональную организацию ОС на основе некоторой совокупности программных модулей.

В состав ОС входят исполняемые и объектные модули стандартных для данной ОС форматов, программные модули специального формата (например, загрузчик ОС, драйверы ввода-вывода), конфигурационные файлы, файлы документации, модули справочной системы и т.д.

Первая версия ОС OS/360 была создана коллективом из 5000 человек за 5 лет и содержала более 1 млн строк кода. Разработанная несколько позже операционная система Mastics содержала к 1975 году уже 20 млн строк.

Стало ясно, что разработка таких систем должна вестись на основе модульного программирования.

Большинство современных ОС представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой унифицированной архитектуры ОС не существует, но известны универсальные подходы к структурированию ОС.

Принципиально важными универсальными подходами к разработке архитектуры ОС являются:

модульная организация;

функциональная избыточность;

функциональная избирательность;

параметрическая универсальность;

концепция многоуровневой иерархической вычислительной системы;

разделение модулей на две группы по функциям;

разделение модулей ОС на две группы по размещению в памяти вычислительной системы;

реализация двух режимов работы вычислительной системы: привилегированного режима (Kernel mode), или режима супервизора (supervisor mode), и пользовательского режима (user mode), или режима задачи (task mode);

ограничение функций ядра.

Кренкель Т. Э., Коган А. Г., Тараторин А. М. Персональные ЭВМ в инженерной практике. - М.: Радио и связь, 1989

Цитата:

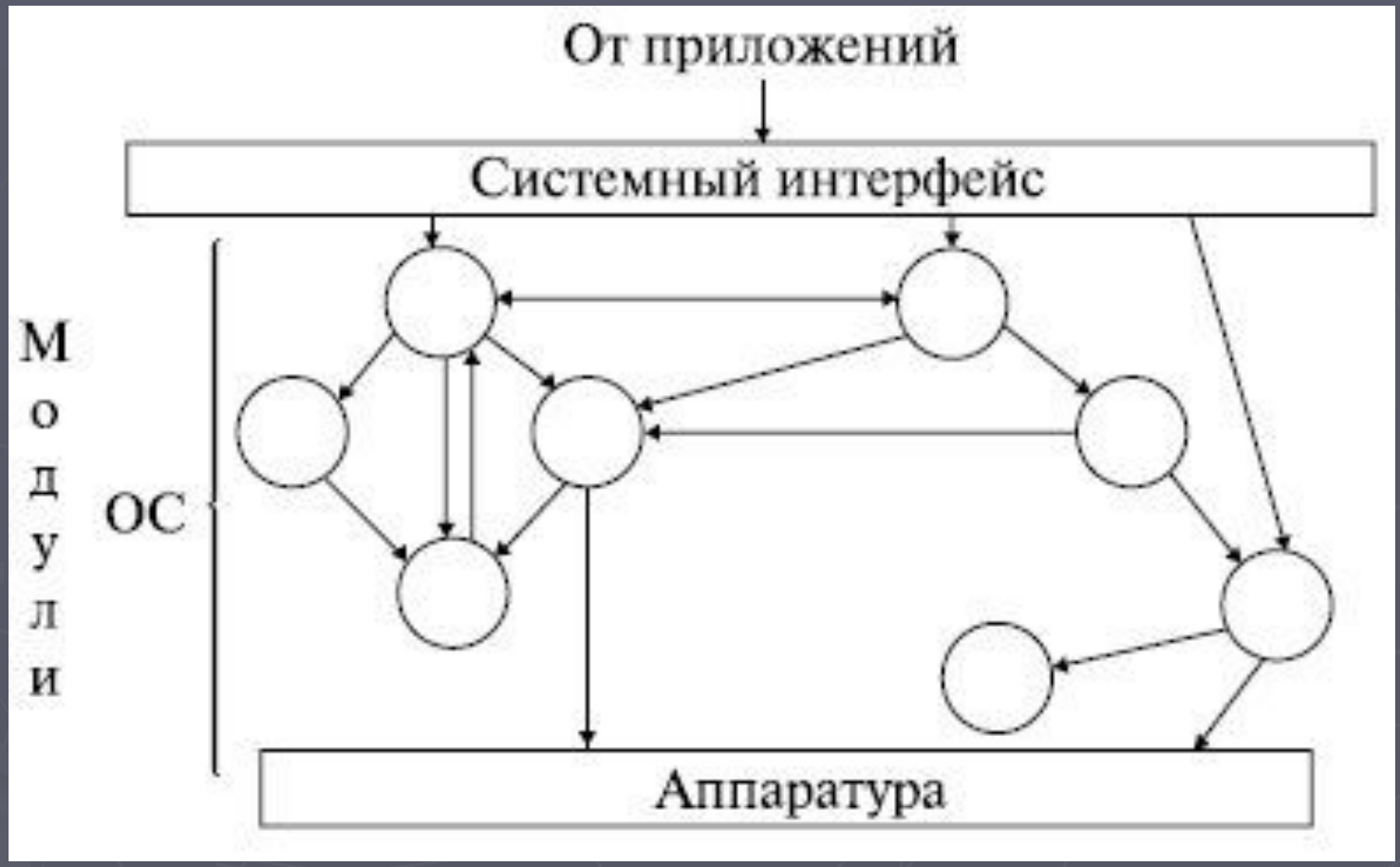
...Одним из примеров громоздкой и, по мнению авторов, бесполезной надстройки является интегрированная система WINDOWS фирмы Microsoft.

Эта система занимает почти 1 Мбайт дисковой памяти и рассчитана на преимущественное использование совместно с устройством типа <мышь>...

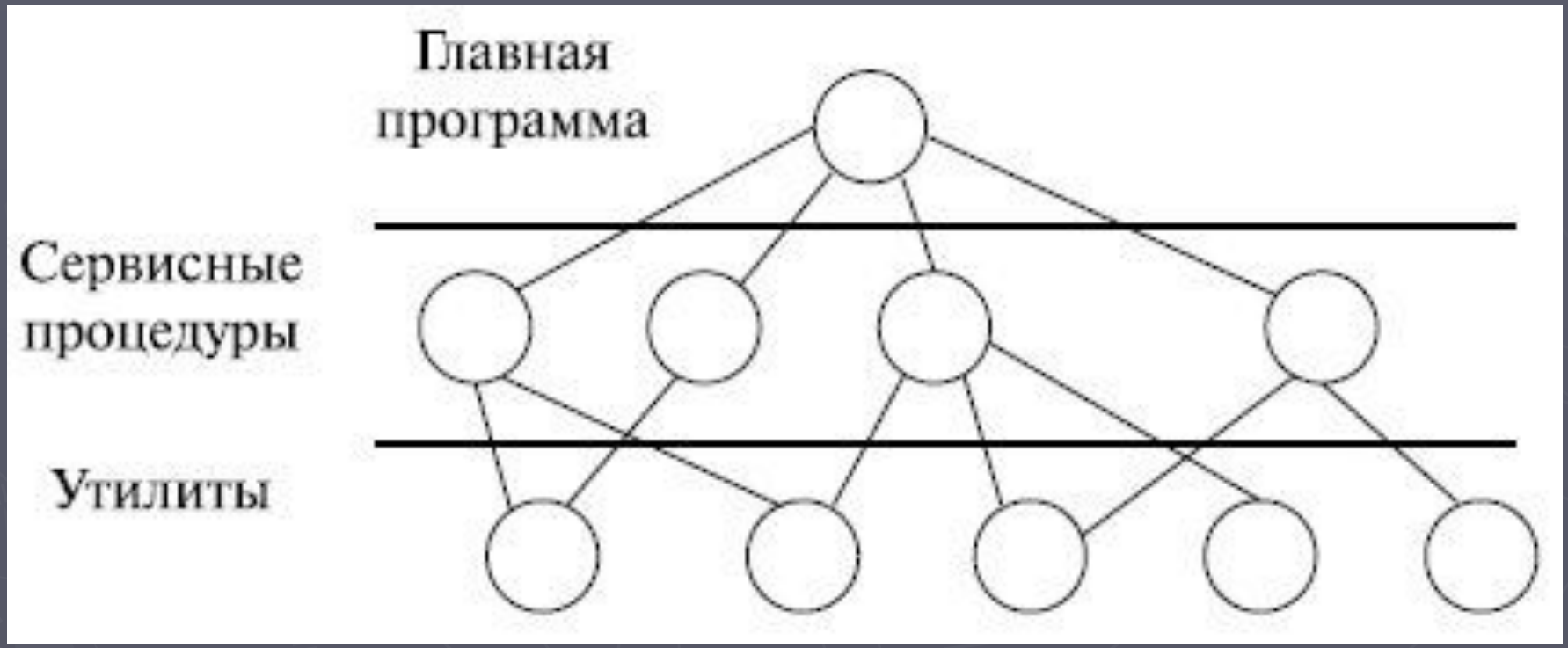
...Таким образом, читатель уже понял, что среди надстроек над ДОС бывают довольно бесполезные системы, которые только выглядят красиво, а на самом деле отнимают время пользователя, память на дисках и оперативную память ЭВМ.

Обманчивая красота таких систем, однако, сильно воздействует на неискушенных пользователей, которые не имели практики работы на машине. Инерция мышления бывает столь сильна, что авторам приходилось наблюдать, как люди, начавшие работать с подобной настройкой, впоследствии с трудом заставляют себя изучать команды ДОС...

Хочется предостеречь от этой ошибки читателей...



Монолитная архитектура



Структурированная архитектура

Такая организация ОС предполагает следующую структуру:

главная программа, которая вызывает требуемые сервисные процедуры;

набор сервисных процедур, реализующих системные вызовы;

набор утилит, обслуживающих сервисные процедуры.

Вспомогательные модули обычно подразделяются на группы:

утилиты – программы, выполняющие отдельные задачи управления и сопровождения вычислительной системы;

системные обрабатывающие программы – текстовые и графические редакторы (Paint, Imaging в Windows 2000), компиляторы и др.;

программы предоставления пользователю дополнительных услуг (специальный вариант пользовательского интерфейса, калькулятор, игры, средства мультимедиа Windows 2000);

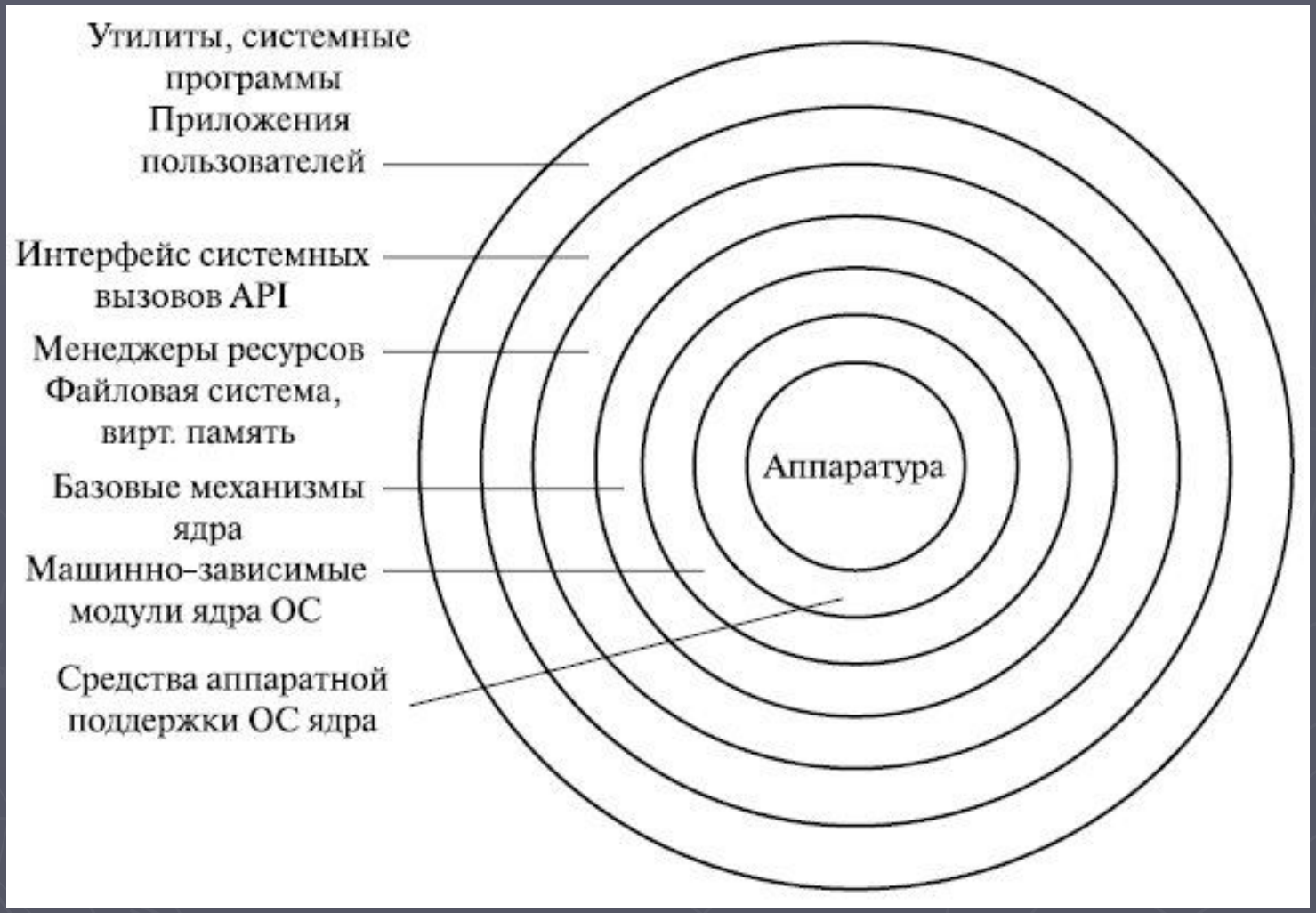
библиотеки процедур различного назначения, упрощения разработки приложений, например, библиотека функций ввода-вывода, библиотека математических функций и т.п.

В концепции многоуровневой (многослойной) иерархической машины структура ОС также представляется рядом слоев.

При такой организации каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс.

На основе этих функций следующий верхний по иерархии слой строит свои функции – более сложные и более мощные и т.д.

Кроме того, модули каждого слоя можно изменять без необходимости изменений в других слоях (но не меняя межслойных интерфейсов!).



Многослойная структура ОС

Обработка системного вызова

Переход к микроядерной архитектуре

Клиент-серверная архитектура

Обработка системного вызова в микроядерной архитектуре

В то же время признаны следующие достоинства микроядерной архитектуры:

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

Многое зависит от размеров и функциональных возможностей микроядра.

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

типичное микроядро первого поколения – 300 Кбайт кода и 140 интерфейсов системных вызовов;

микроядро ОС L4 (второе поколение) – 12 Кбайт кода и 7 интерфейсов системных вызовов.

В современных операционных системах различают следующие виды ядер.

Наноядро (НЯ). Крайне упрощённое и минимальное ядро, выполняет лишь одну задачу – обработку аппаратных прерываний, генерируемых устройствами компьютера.

Микроядро (МЯ) предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием.

Экзоядро (ЭЯ) предоставляет лишь набор сервисов для взаимодействия между приложениями, а также необходимый минимум функций, связанных с защитой

Монолитное ядро (МнЯ) предоставляет широкий набор абстракций оборудования.

Модульное ядро (Мод. Я) – современная, усовершенствованная модификация архитектуры МЯ.

Гибридное ядро (ГЯ) – модифицированные микроядра, позволяющие для ускорения работы запускать "несущественные" части в пространстве ядра. Имеют "гибридные" достоинства и недостатки. Примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра.

Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре Windows NT. Хотя Windows NT часто называют микроядерной операционной системой, это не совсем так.

Микроядро NT слишком велико (более 1 Мбайт), чтобы носить приставку "микро". Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах.

В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром.

Классификация операционных систем

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

По назначению ОС делятся на универсальные и специализированные.

Специализированные ОС, как правило, работают с фиксированным набором программ (функциональных задач). Применение таких систем обусловлено невозможностью использования универсальной ОС по соображениям эффективности, надежности, защищенности и т.п., а также вследствие специфики решаемых задач.

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

По способу загрузки можно выделить загружаемые ОС (большинство) и системы, постоянно находящиеся в памяти вычислительной системы.

По особенностям алгоритмов управления ресурсами. Главным ресурсом системы является процессор, поэтому дадим классификацию по алгоритмам управления процессором, хотя можно, конечно, классифицировать ОС по алгоритмам управления памятью, устройствами ввода-вывода и.т.д.

Поддержка многозадачности (многопрограммности).

Однопрограммные ОС предоставляют пользователю виртуальную машину, делая более простым и удобным процесс взаимодействия пользователя с компьютером.

Они также имеют средства управления файлами, периферийными устройствами и средства общения с пользователем.

Многозадачные ОС, кроме того, управляют разделением совместно используемых ресурсов (процессор, память, файлы и т.д.), это позволяет значительно повысить эффективность вычислительной системы.

Главное отличие многопользовательских систем от однопользовательских – наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей.

Виды многопрограммной работы. Специфику ОС во многом определяет способ распределения времени между несколькими одновременно существующими в системе процессами (или потоками).

По этому признаку можно выделить 2 группы алгоритмов: не вытесняющая многопрограммность (Windows3.x, NetWare) и вытесняющая многопрограммность (Windows 2000/2003/XP, OS/2, Unix).

В первом случае активный процесс выполняется до тех пор, пока он сам не отдает управление операционной системе.

Во втором случае решение о переключении процессов принимает операционная система. Возможен и такой режим многопрограммности, когда ОС разделяет процессорное время между отдельными ветвями (потоками, волокнами) одного процесса.

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

По области использования и форме эксплуатации:

системы пакетной обработки (OS/360, ОС ЕС);
системы разделения времени (UNIX, VMS);
системы реального времени (QNX, RT/11).

По аппаратной платформе (типу вычислительной техники), для которой они предназначаются, операционные системы делят на следующие группы.

Операционные системы для смарт-карт. Некоторые из них могут управлять только одной операцией, например, электронным платежом. Некоторые смарт-карты являются JAVA-ориентированным и содержат интерпретатор виртуальной машины JAVA. Апплеты JAVA загружаются на карту и выполняются JVM-интерпретатором.

Встроенные операционные системы. Управляют карманными компьютерами (Palm OS, Android, Windows CE – Consumer Electronics – бытовая техника), мобильными телефонами, телевизорами, микроволновыми печами и т.п.

Операционные системы для персональных компьютеров, например, Windows, Linux, Mac OSX и др.

Операционные системы мини-ЭВМ, например, RT-11 для PDP-11 – ОС реального времени, RSX-11 M для PDP-11 – ОС разделения времени, UNIX для PDP-7.

Операционные системы мэйнфреймов (больших машин), например, OS/390, происходящая от OS/360 (IBM). Обычно ОС мэйнфреймов предполагает одновременно три вида обслуживания: пакетную обработку, обработку транзакций (например, работа с БД, бронирование авиабилетов, процесс работы в банках) и разделение времени.

Серверные операционные системы, например, UNIX, Windows Server, Linux.

Кластерные операционные системы. Кластер – слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений и представляющихся пользователю единой системной.

Эффективность и требования, предъявляемые к ОС

К операционным системам современных компьютеров предъявляется ряд требований.

Эффективность.

Надежность и отказоустойчивость ОС.

Безопасность (защищенность).

Предсказуемость

Переносимость.

Совместимость.

Удобство.

Масштабируемость.

Совместимость и множественные прикладные среды

Многие архитектурные особенности ОС непосредственно касаются только системных программистов, концепция множественных прикладных (операционных) средств непосредственно связана с нуждами конечных пользователей – возможностью операционной системы выполнять приложения, написанные для других операционных систем.

Такое свойство операционной системы называется совместимостью.

Совместимость приложений может быть на двоичном уровне и на уровне исходных текстов.

Вид возможной совместимости зависит от многих факторов.

Самый главный из них – архитектура процессора.

Если процессор применяет тот же набор команд (возможно, с добавлениями, как в случае IBM PC: стандартный набор + мультимедиа + графика + потоковые) и тот же диапазон адресов, то двоичная совместимость может быть достигнута достаточно просто.

Для этого необходимо соблюдение следующих условий:

- API, который использует приложение, должен поддерживаться данной ОС;

- внутренняя структура исполняемого файла приложения должна соответствовать структуре исполняемых файлов данной ОС.

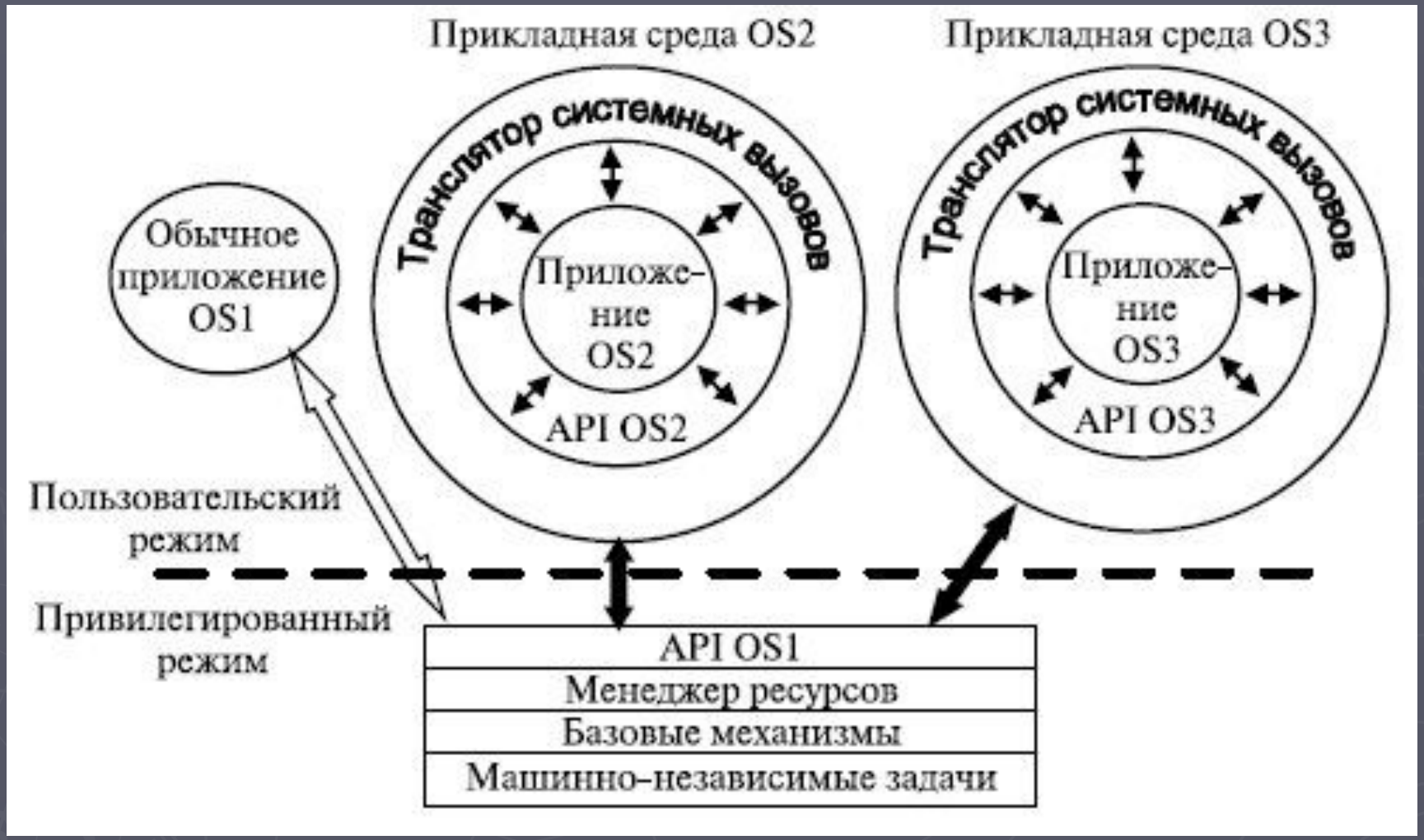
Чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь обеспечивать совместимость API.

Концепции, положенные в основу разных ОС, могут входить в противоречия друг с другом.

Каждая ОС имеет свои собственные механизмы защиты ресурсов, свои алгоритмы обработки ошибок и исключительных ситуаций, особую структуру процессора и схему управления памятью, свою семантику доступа к файлам и графический пользовательский интерфейс.

Существуют различные варианты построения множественных прикладных сред, отличающиеся как особенностями архитектурных решений, так и функциональными возможностями, обеспечивающими разную степень переносимости приложений.

Совместимость и множественные прикладные среды



Организация множественных прикладных сред

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

очень просто можно добавлять и исключать прикладные среды, что является следствием хорошей расширяемости микро ядерных ОС;

при отказе одной из прикладных сред остальные сохраняют работоспособность;

низкая производительность микроядерных ОС сказывается на скорости работы прикладных средств, а значит, и на скорости работы приложений.

Создание в рамках одной ОС нескольких прикладных средств для выполнения приложений различных ОС представляет собой путь, который позволяет иметь единственную версию программы и переносить ее между различными операционными системами.

Множественные прикладные среды обеспечивают совместимость на двоичном уровне данной ОС с приложениями, написанными для других ОС.

Виртуальные машины как современный подход к реализации множественных прикладных сред

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

Каждая из этих виртуальных машин (ВМ) была настолько похожа на базовую физическую машину, что существующее программное обеспечение могло выполняться на ней в неизменном виде.

В то время вычислительные задачи общего характера решались на дорогих мэйнфреймах (типа IBM/360), и пользователи высоко оценили способность МВМ распределять дефицитные ресурсы среди нескольких приложений.

Как ни странно, развитие современных ОС и снижение стоимости оборудования привели к появлению проблем, которые исследователи надеялись решить с помощью МВМ.

В настоящее время МВМ стал не столько средством организации многозадачности, каким он был когда-то задуман, сколько решением проблем обеспечения безопасности, мобильности и надежности.

Виртуализация – позволяет отделить ПО от нижележащей аппаратной инфраструктуры.

Фактически она разрывает связь между определенным набором программ и конкретным компьютером.

Монитор виртуальных машин отделяет программное обеспечение от оборудования и формирует промежуточный уровень между ПО, выполняемым виртуальными машинами, и аппаратными средствами.

Этот уровень позволяет МВМ полностью контролировать использование аппаратных ресурсов *гостевыми операционными системами (GuestOS)*, которые выполняются на ВМ.

Инкапсуля́ция (лат. in capsula) — размещение в оболочке, изоляция, закрытие чего-либо инородного с целью исключения влияния на окружающее.

Например, поместить радиоактивные отходы в капсулу, закрыть кожухом механизм, убрать мешающее в ящик или шкаф.

Инкапсуляция также означает, что администратор может в любой момент приостановить или возобновить работу ВМ, а также сохранить текущее состояние виртуальной машины либо вернуть ее к предыдущему состоянию.

МВМ играет роль посредника во всех взаимодействиях между ВМ и базовым оборудованием, поддерживая выполнение множества виртуальных машин на единой аппаратной платформе и обеспечивая их надежную изоляцию.

Полная изоляция также важна для надежности и обеспечения безопасности. Приложения, которые раньше выполнялись на одной машине, теперь можно распределить по разным ВМ.

МВМ – это инструмент реструктуризации системы для повышения ее устойчивости и безопасности, не требующий дополнительных площадей и усилий по администрированию, которые необходимы при выполнении приложений на отдельных физических машинах.

МВМ должен связать аппаратный интерфейс с ВМ, сохранив полный контроль над базовой машиной и процедурами взаимодействия с ее аппаратными средствами.

Вместо того чтобы заниматься сложной переработкой кода гостевой операционной системы, можно внести некоторые изменения в основную операционную систему, изменив некоторые наиболее "мешающие" части ядра.

Подобный подход называется паравиртуализацией.

При паравиртуализации разработчик МВМ переопределяет интерфейс виртуальной машины, заменяя непригодное для виртуализации подмножество исходной системы команд более удобными и эффективными эквивалентами.

Самый большой недостаток паравиртуализации – несовместимость.

Чтобы добиться высокой производительности и совместимости при виртуализации архитектуры x86, применяется метод виртуализации, который объединяет традиционное прямое выполнение с быстрой трансляцией двоичного кода "на лету".

В большинстве современных ОС режимы работы процессора при выполнении обычных прикладных программ легко поддаются виртуализации.

Преобразованный код очень похож на результаты паравиртуализации.

Обычные команды выполняются в неизменном виде, а команды, нуждающиеся в специальной обработке (такие как POPF и команды чтения регистров сегмента кода), транслятор заменяет последовательностями команд, которые подобны требующимся для выполнения на паравиртуализованной виртуальной машине.

Но есть важное различие: вместо того, чтобы изменять исходный код операционной системы или приложений, транслятор двоичного кода изменяет код при его выполнении в первый раз.

Эффекты виртуализации

Администраторы могут с единой консоли быстро вводить в действие ВМ и управлять тысячами виртуальных машин, выполняющихся на сотнях физических серверов.

Администраторы могут создавать по имеющимся шаблонам новые экземпляры виртуальных серверов и отображать их на физические ресурсы в соответствии с политиками администрирования.

Компьютеры просто как часть пула универсальных аппаратных ресурсов (примером тому может служить виртуальный центр VMware VirtualCenter).

Отображение виртуальных машин на аппаратные ресурсы очень динамично.

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

Локализация неисправностей.

Гибкая обработка отказов.

Разные уровни безопасности.

Компьютерные системы существуют и продолжают развиваться благодаря тому, что разработаны по законам иерархии и имеют хорошо определенные интерфейсы, отделяющие друг от друга уровни абстракции.

Подсистемы и компоненты, разработанные по спецификациям разных интерфейсов, не способны взаимодействовать друг с другом.

Виртуализация позволяет обойти эту несовместимость.

Виртуализация системы или компонента (например, процессора, памяти или устройства ввода/вывода) на конкретном уровне абстракции отображает его интерфейс и видимые ресурсы на интерфейс и ресурсы реальной системы.

В отличие от абстракции, виртуализация не всегда нацелена на упрощение или сокрытие деталей.

Операция записи на виртуальный диск преобразуется в операцию записи в файл (и следовательно, в операцию записи на реальный диск).

История семейства операционных систем UNIX/Linux

Семейство операционных систем UNIX уникально по нескольким причинам:

оно является долгожителем и, претерпев многочисленные изменения, "завоевало" разнообразную аппаратуру;

при переходе UNIX на другие аппаратные платформы возникали интересные задачи, решение которых принесло много нового в компьютерные технологии;

на одной из версий UNIX были реализованы протоколы обмена данными в компьютерных сетях с разной аппаратной платформой, что позволяет считать UNIX предвестницей сегодняшнего Интернета, а также основой для широкого развития локальных сетей;

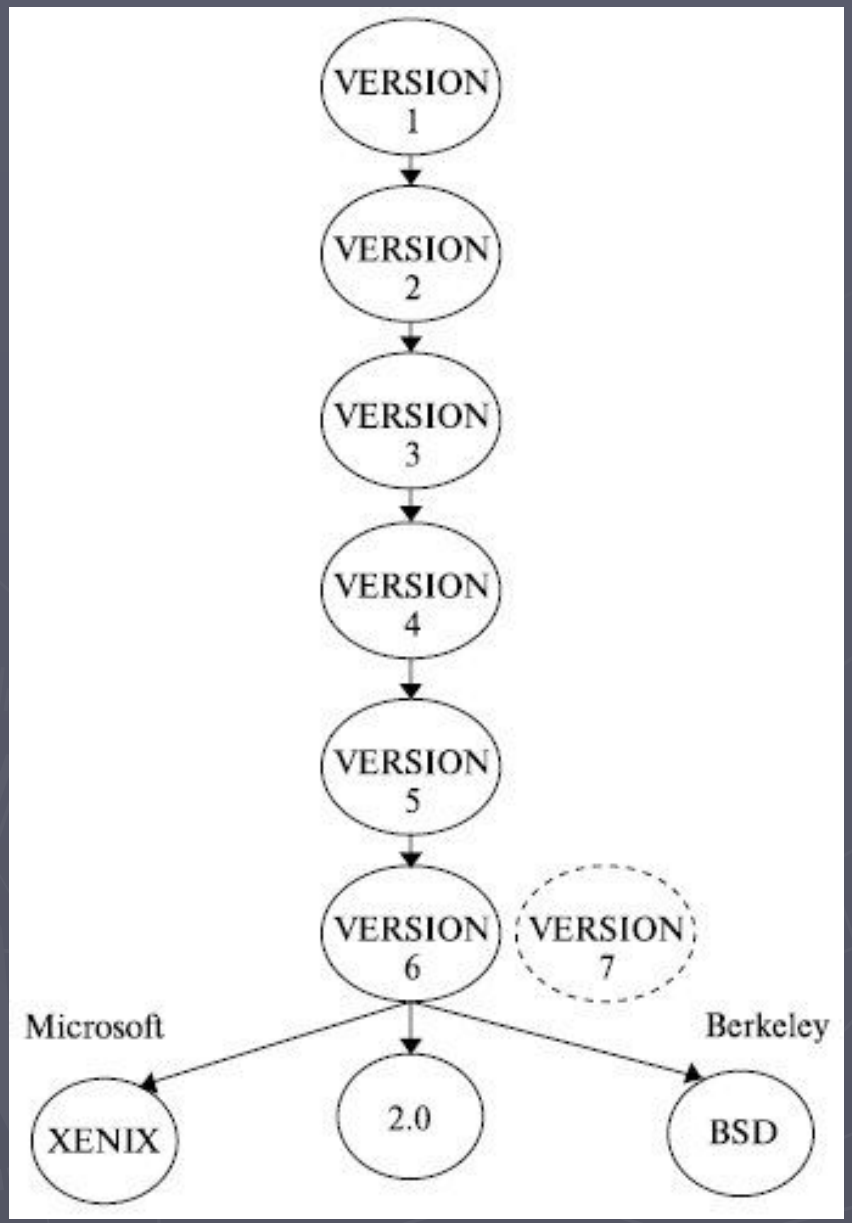
авторы ее первых версий создали язык программирования высокого уровня С, который можно назвать (с учетом его последующего совершенствования) самым распространенным среди разработчиков;

появившиеся в семействе UNIX свободно распространяемые операционные системы внесли много нового в представление о том, как разрабатывать и распространять программы для компьютеров.

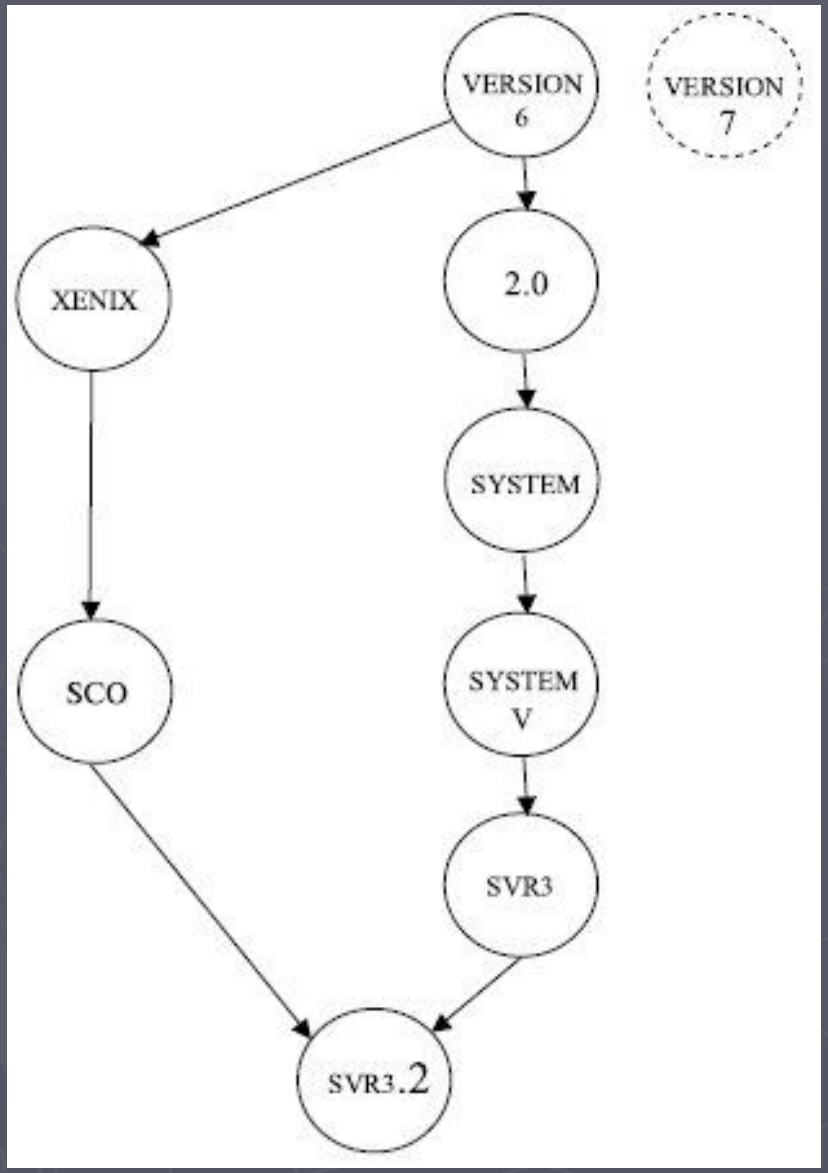
История семейства операционных систем UNIX/Linux

Характеристика редакций UNIX AT&T

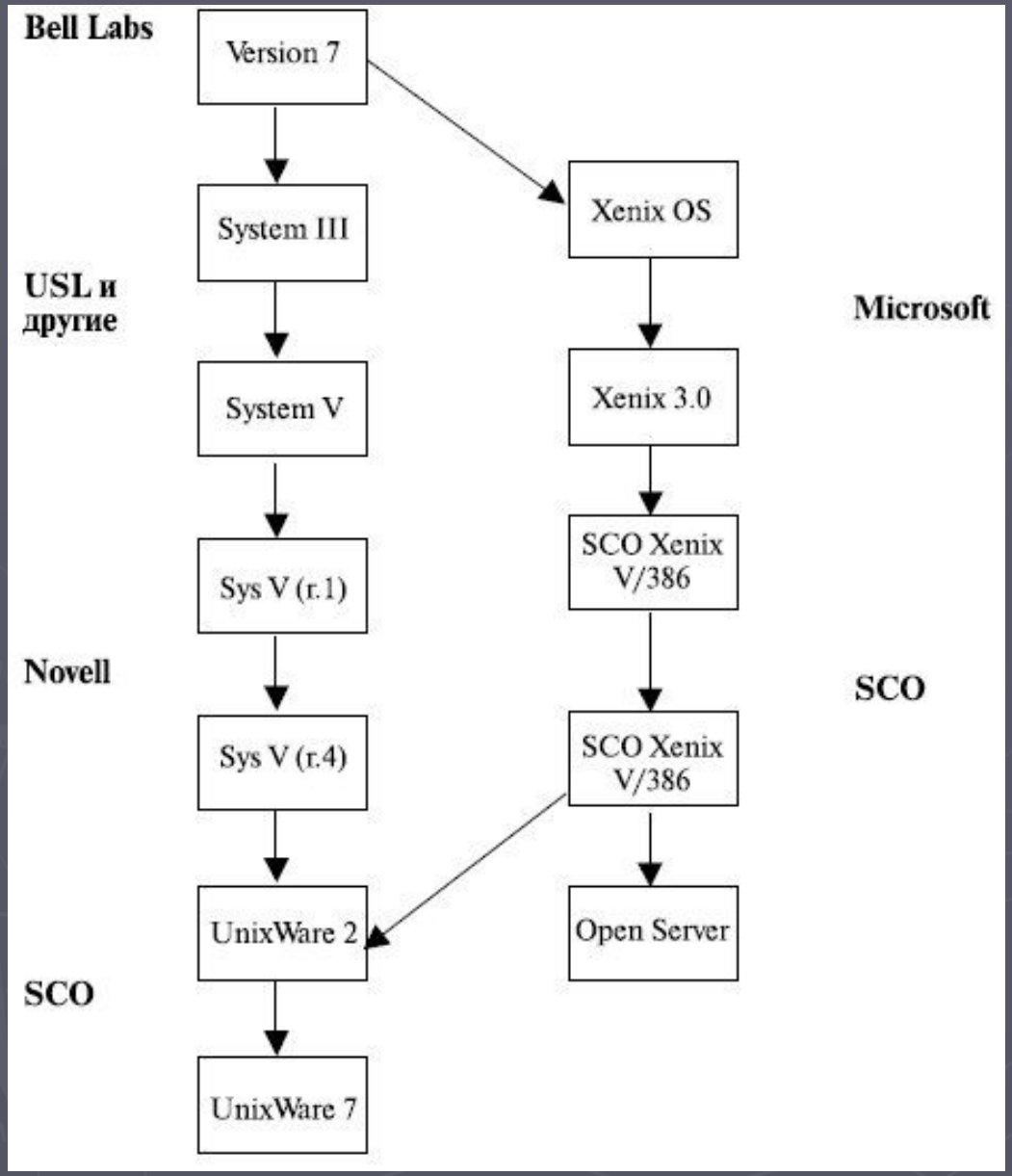
№ редакции	Год выпуска	Краткая характеристика
1	1971	Первая версия UNIX, написанная на ассемблере для PDP-11. Включала компилятор B и много известных команд и утилит, в том числе cat(1), chdir(1), chmod(1), cp(1), ed(1), find(1), mail(1), mkdir(1), mkfs(1M), mount(1M), mv(1), rm(1), rmdir(1), w(1), who(1). В основном использовалась как инструментальное средство обработки текстов для патентного отдела
3	1973	В системе появилась команда cc(1), запускавшая компилятор C. Число установленных систем достигло 16
4	1973	Первая система, в которой ядро написано на языке высокого уровня C
6	1975	Первая версия системы, доступная за пределами Bell Labs. Система полностью переписана на языке C. С этого времени начинается появление новых версий, разработанных за пределами Bell Labs, и рост популярности UNIX. В частности, эта версия системы была установлена Томпсоном в Калифорнийском университете в Беркли, и на ее основе вскоре была выпущена первая версия BSD (Berkeley Software Distribution) UNIX
7	1979	Эта версия включала командный интерпретатор Bourne Shell и компилятор C от Кернигана и Ритчи. Ядро было переписано для упрощения переносимости системы на другие платформы. Лицензия на эту версию была куплена фирмой Microsoft, которая разработала на ее базе операционную систему Xenix



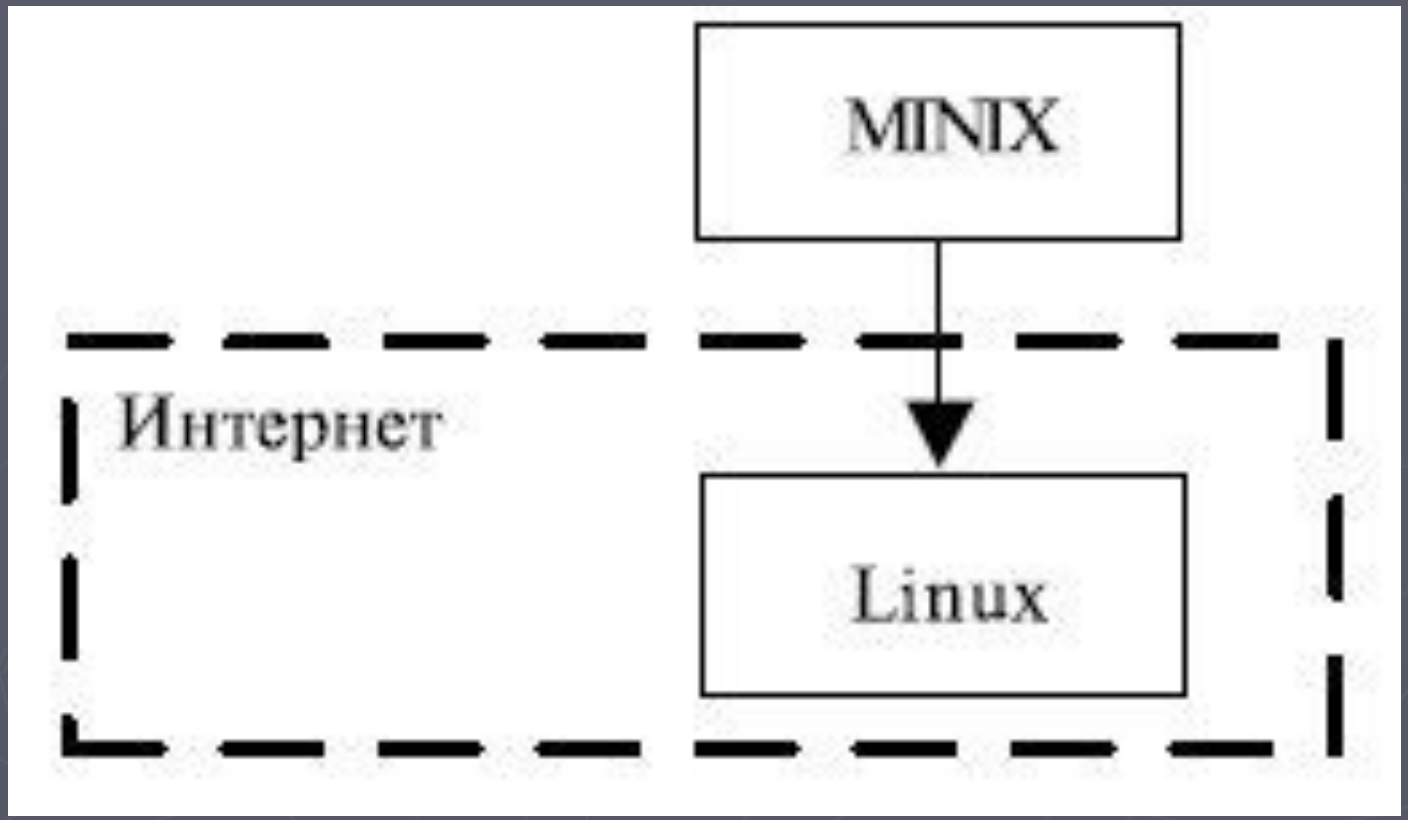
UNIX распалась на три ключевых компонента



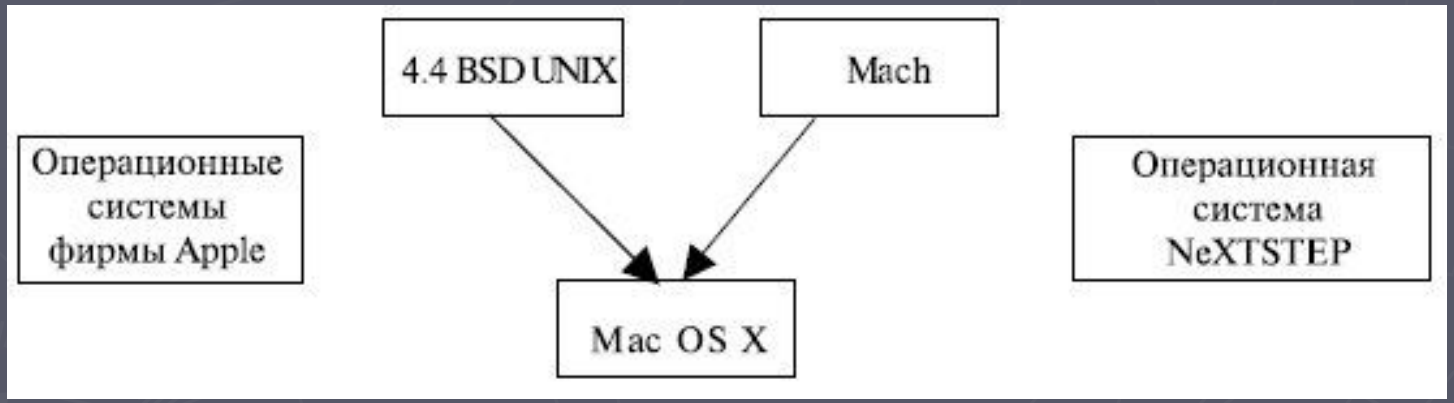
SCO Unix - SVR - SVR3.2

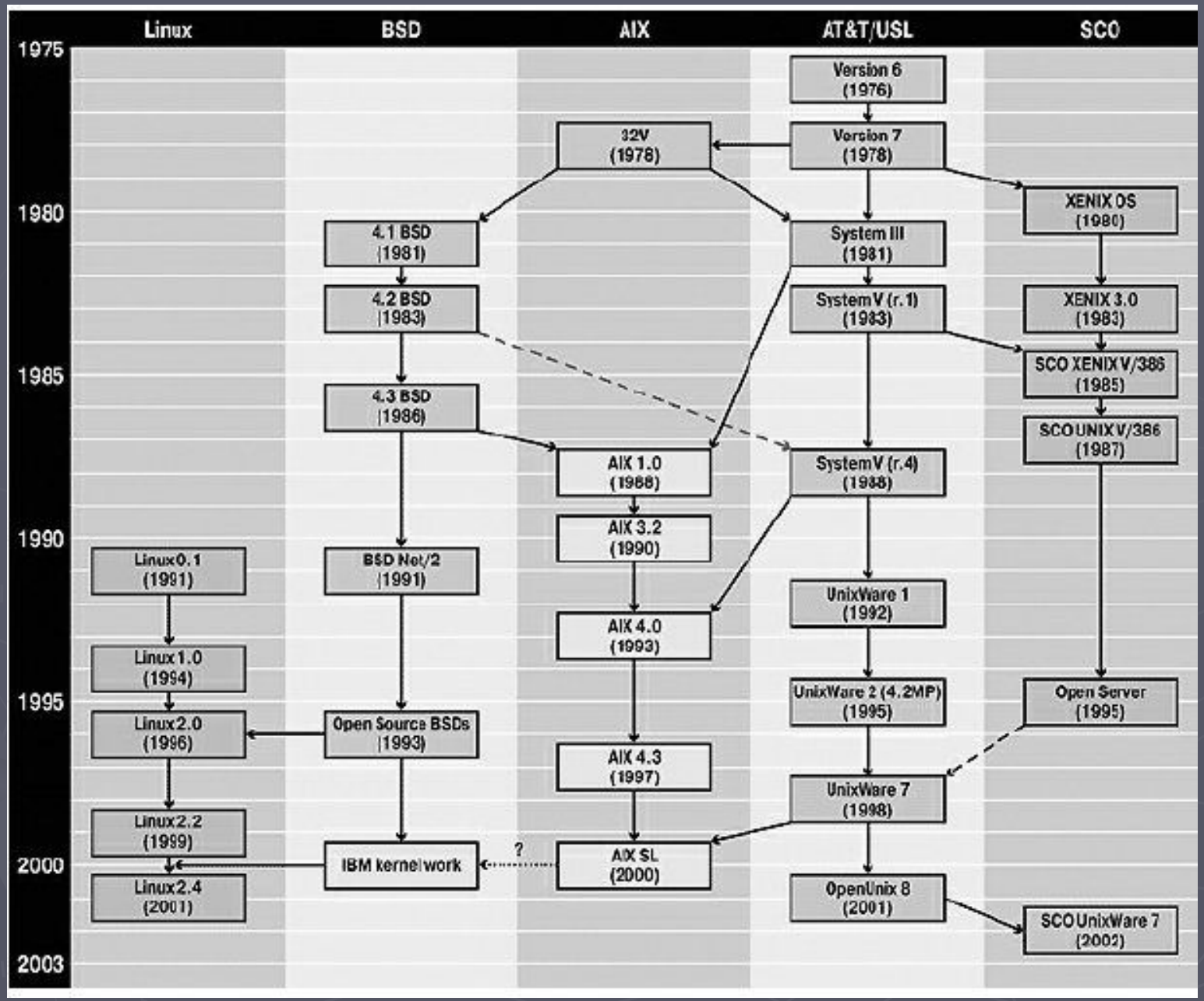


Правопреемники исходных текстов UNIX



Linux - Minix


4.4 BSD UNIX
Mach
Операционные системы фирмы Apple
Операционная система NeXTSTEP
Mac OS X


Linux
BSD
AIX
AT&T/USL
SCO
1975
1980
1985
1990
1995
2000
2003
Version 6 (1976)
Version 7 (1978)
32V (1978)
XENIX OS (1980)
4.1 BSD (1981)
System III (1981)
4.2 BSD (1983)
System V (r.1) (1983)
XENIX 3.0 (1983)
SCO XENIX V/386 (1985)
4.3 BSD (1986)
SCO UNIX V/386 (1987)
AIX 1.0 (1988)
System V (r.4) (1988)
AIX 3.2 (1990)
Linux0.1 (1991)
BSD Net/2 (1991)
UnixWare 1 (1992)
AIX 4.0 (1993)
Linux1.0 (1994)
UnixWare 2 (4.2MP) (1995)
Open Server (1995)
Linux2.0 (1996)
Open Source BSDs (1993)
AIX 4.3 (1997)
UnixWare 7 (1998)
Linux2.2 (1999)
IBM kernel work
?
AIX SL (2000)
OpenUnix 8 (2001)
Linux2.4 (2001)
SCO UnixWare 7 (2002)

UX	AIX	BSD	Dynix FreeBSD
GNU	GNU/Linux	HP-UX	IRIX
Mac OS X	Minix	NetBSD	NeXTSTEP
OpenBSD	PC-BSD	Plan 9 Plan B	QNX
SCO OpenServer	Solaris System V	Tru64	Xenix

На странице Википедии приводятся такие варианты UNIX-подобных операционных систем

http://www.levenez.com/unix/

Операционные системы фирмы Microsoft

Microsoft (Microsoft Corporation, читается "майкрософт", NASDAQ: MSFT) – крупнейшая (прибыль за 2014 год – 17,7 млрд долл. при обороте в 60,4 млрд долл.) транснациональная компания по производству программного обеспечения для различного рода вычислительной техники – персональных компьютеров, игровых приставок, КПК, мобильных телефонов и прочего, разработчик наиболее широко распространенной на данный момент в мире программной платформы – семейства операционных систем Windows.

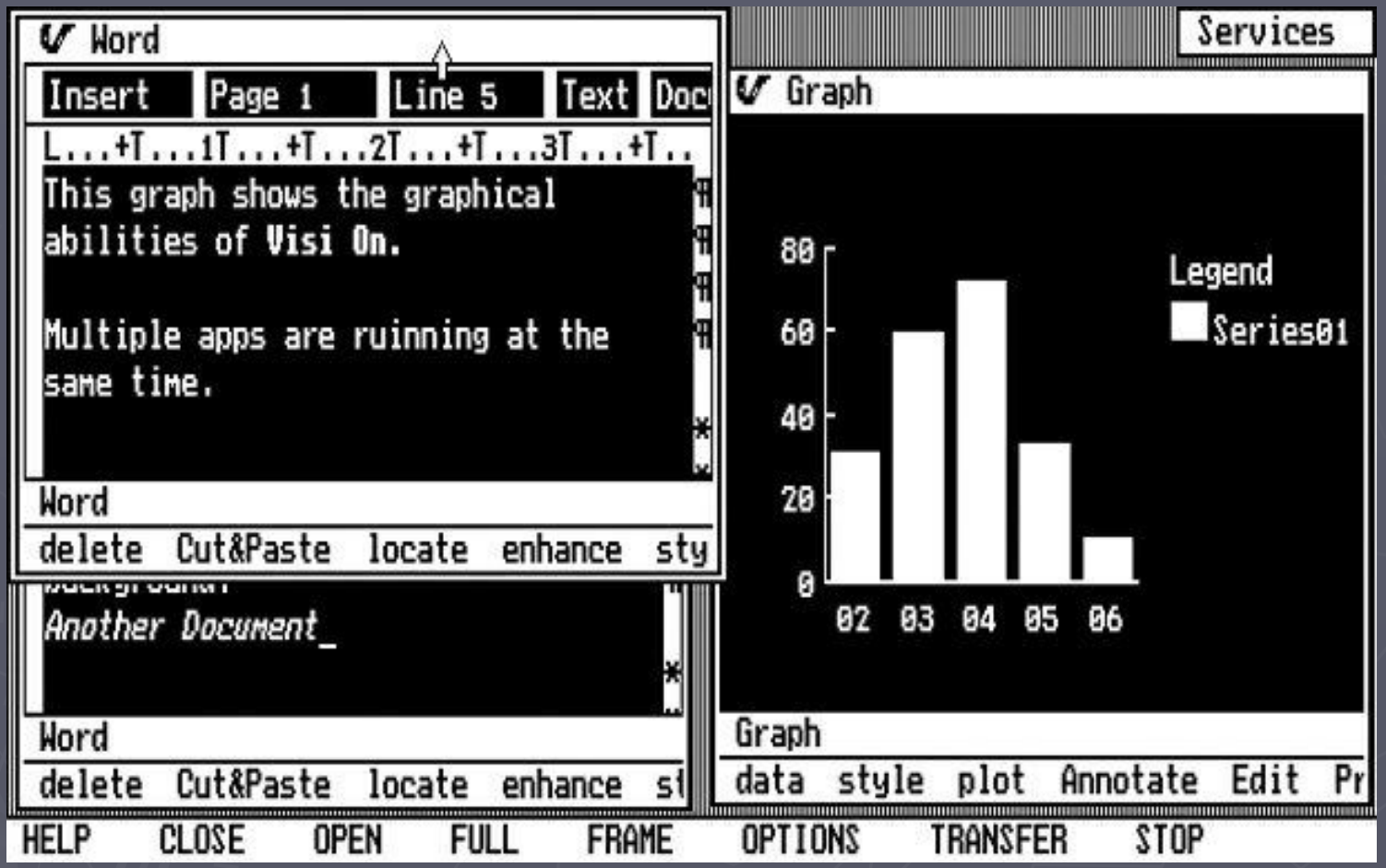
Фирма Microsoft была основана двумя студентами: Биллом Гейтсом и Полом Алленом в 1975 году.

История операционных систем для персональных компьютеров IBM PC начинается в 1981 году, когда на этом оборудовании была установлена MS DOS 1.0.

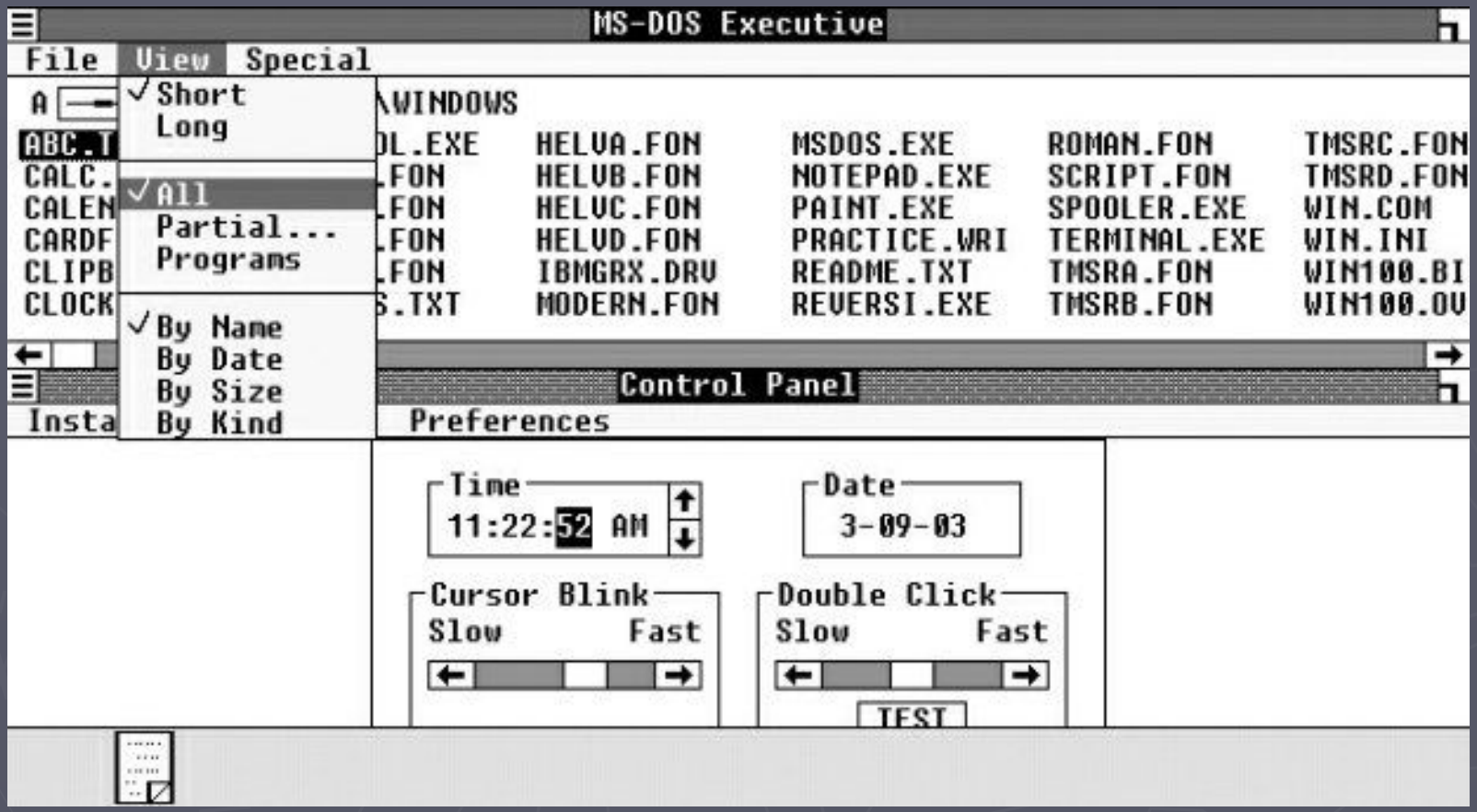
1. MS DOS. Серия операционных систем, поддерживающих только командную строку как интерфейс пользователя. Выпущены версии от 1.0 (1981 год) до 6.22 (1994 год). Многие компании (в числе которых IBM, DEC и даже МФТИ) создавали свои версии этой системы.

2 . Windows 1, 2, 3 и 3.11. Надстройки над операционными системами MS DOS, обеспечивающими режим графического интерфейса пользователя.

Следует заметить, что имелся предшественник Windows – графическая оболочка компании Visi Corp под названием Visi On.



Пример интерфейса графической оболочки Visi On



Пример интерфейса графической оболочки Windows1

3. Windows 9X. Эта серия операционных систем представлена такими версиями: Windows 95, Windows 98 и Windows Me.

4. Windows NT. Сокращение в NT ее названии образовано от New Technology. Первая ее версия, созданная к 1993 году, должна была вытеснить MS DOS, чего не произошло. Следующие версии должны были потеснить на рынке Windows 95, что случилось только в начале 2000 годов. Создавались варианты этой системы как для работы пользователя на локальном компьютере, так и для управления локальной сетью. Версии этого направления до определенного времени назывались NT, а с 2000 года получала разные имена: NT 5.0 – Windows 2000, NT 5.2 – Windows 2003, NT 6.0 – Windows Vista и Windows 2008, NT 6.1 – Windows 7.

5. Windows CE. Эти операционные системы начали разрабатываться в 1996 году.

Можно отметить, что фирма Microsoft является фактически монополистом на производство программного обеспечения для персональных компьютеров.

Интерфейс операционной системы Windows95

Отличия семейства UNIX/Linux от операционных систем Windows и MS DOS

Исходные тексты компонентов системы доступны для просмотра и модификации. Чаще всего они располагаются в подкаталоге с именем source, который подчинен каталогу /usr.

Модифицировать систему можно перекомпилировав ядро – основу системы, которая непрерывно развивается и настраивается на конфигурацию вычислительной установки.

Существует несколько уровней настройки параметров работы системы:
- работа с утилитами, в том числе в режиме графического интерфейса;
- корректировка файлов конфигурации;
- внесение изменений в исходные тексты и их дальнейшая перекомпиляция.

Первоначально загружается командный режим, а графический интерфейс требует дополнительного вызова. Последний имеет несколько методов реализации.

В инсталляторы системы Linux включается полный набор программного обеспечения, необходимый для работы как в качестве офисного или домашнего компьютера, так и сервера.

Интересной особенностью работы системы является возможность одновременной регистрации нескольких пользователей на виртуальных терминалах.

В системе существует множество оболочек (аналог командного интерпретатора comand.com в MS DOS).

Файловая система Linux на жестком диске может расположиться на нескольких разделах диска, а для области подкачки всегда выделяется отдельный дисковый раздел с типом файловой системы, отличной от основной. Также в отдельных разделах диска можно разместить следующую информацию (приводится список, доступный Linux):

(/boot);
(/var);
(/home);
(/usr).

Доступ к данным, получаемым с разнообразного оборудования, осуществляется не в одной из вершин верхнего уровня файловой системы, а в одной из вершин, подчиненных единственному корню иерархической файловой системы (ее имя /).

Ядро и вспомогательные модули операционной системы

- При функциональной декомпозиции ОС модули разделяются на две группы:
  - ядро – модули, выполняющие основные функции ОС;
  - модули, выполняющие вспомогательные функции ОС.

Модули ядра ОС

- Модули ядра ОС выполняют следующие базовые функции ОС:
  - управление процессами
  - управление памятью
  - управление устройствами ввода-вывода
- Ядро обеспечивает решение задачи **организации вычислительного процесса**: переключение контекстов, загрузка/выгрузка страниц, обработка прерываний и т.п.
- Другая задача – поддержка приложений, создание для них **прикладной программной среды**. Приложения обращаются к ядру с запросами (**системными вызовами**) для выполнения базовых операций (открытие и чтение файла, вывод информации на дисплей и т.п.)
- Функции выполняемые ядром ОС требуют высокой скорости выполнения и для этого размещаются постоянно в оперативной памяти (**резидентные модули**).

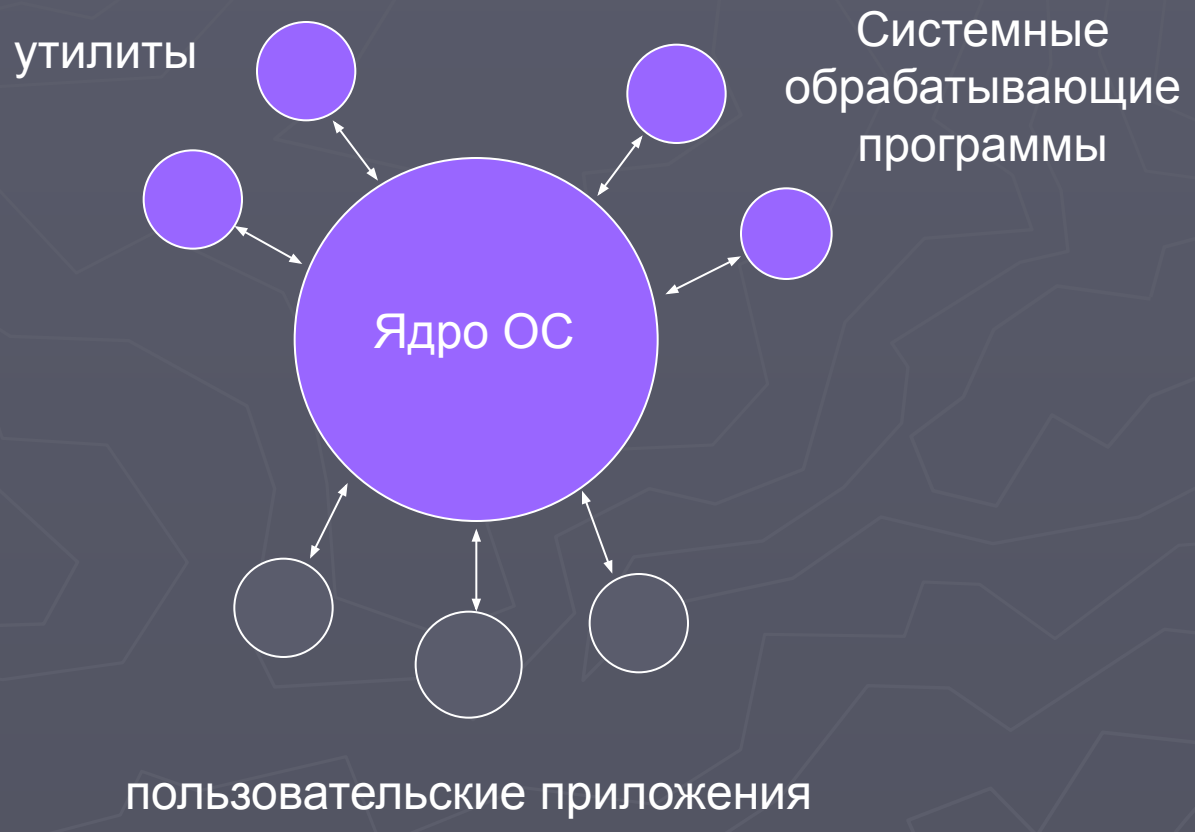
Вспомогательные модули операционной системы

- Вспомогательные модули выполняют полезные, но менее обязательные функции. Например:
  - архивирование информации;
  - дефрагментация данных на диске;
  - поиск необходимого файла и т.п.
- Вспомогательные модули часто оформляются как обычные приложения и провести границу между ними и обычными приложениями сложно.
- Деление на основные и вспомогательные модули ОС условно. Некоторые программы переходят из разряда вспомогательных модулей в основные и наоборот.

Вспомогательные модули операционной системы

- Вспомогательные модули ОС условно разделяются на следующие группы:
  - *Утилиты* – приложения, решающие отельные задачи управления и сопровождения ОС
  - *Системные обрабатывающие программы* – текстовые и графические редакторы, компиляторы, компоновщики и т.п.
  - *Программы предоставления пользователю дополнительных услуг* – специальный вариант пользовательского интерфейса, калькулятор, игры и т.п.
  - *Библиотеки процедур* – модули различного назначения, упрощающие разработку приложений.
- Вспомогательные модули обращаются к функциям ядра ОС посредством системных вызовов.

Ядро и вспомогательные модули операционной системы



Привилегированный режим процессора

- Для надежного управления работой приложений ядро ОС должно обладать некоторыми привилегиями по отношению к остальным приложениям.
- Обеспечивается привилегированный режим специальными средствами аппаратной поддержкой. Процессор компьютера поддерживает как минимум два режима работы – **пользовательский** (user mode) и **привилегированный** (kernel mode).
- Приложения в пользовательском режиме не могут выполнять некоторые критичные команды (переключение процессора с задачи на задачу, доступ к механизму выделения и защиты областей памяти и т. п.).

Привилегированный режим работы

- Между числом привилегий, поддерживаемых аппаратурой и операционной системой нет однозначного соответствия:
  - процессор Intel поддерживает 4 режима работы процессора – операционные системы Windows используют два из них.
- Для реализации привилегированного режима достаточно поддержки двух режимов работы
- Повышение устойчивости ОС, обеспечивающееся использованием работы в привилегированном режиме, достигается за счет некоторого замедления, вызванного необходимостью переключения работы ядра.
- Архитектура ОС, основанная на разделении привилегированного режима для ядра и пользовательского режима для приложений – стала классической.

Многослойная структура ОС

- Вычислительная система под управлением ОС можно рассматривать как состоящую из нескольких слоев:
  - Нижний слой – аппаратура;
  - Средний – ядро ОС;
  - Верхний – утилиты, приложения и т.п.

Детализация структуры ядра

- Ядро, являясь структурным элементом ОС, может быть логически разложен на ряд слоев:
  - Средства аппаратной поддержки ОС
  - Машинно-зависимые компоненты ОС (включает модули, отражающие специфику аппаратной платформы компьютера)
  - Базовые механизмы ядра (включает наиболее примитивные операции ядра – переключение контекстов процессов, диспетчеризация прерываний), модули выполняют решения принятые на более высоких уровнях
  - Менеджеры ресурсов (реализует задачи стратегического управления), включает менеджеры – диспетчеры процессов, ввода-вывода и т.п.
  - Интерфейсы системных вызовов (включает модули взаимодействия с приложениями и системными утилитами, функции API.

Аппаратная зависимость ОС

- Операционная система в процессе работы взаимодействует с аппаратными средствами компьютера:
  - Средства поддержки привилегированного режима
  - Средства трансляции адресов
  - Средства переключения процессов
  - Защита областей памяти
  - Система прерываний
  - Системный таймер
- Это делает ОС привязанной к определенной аппаратной платформе

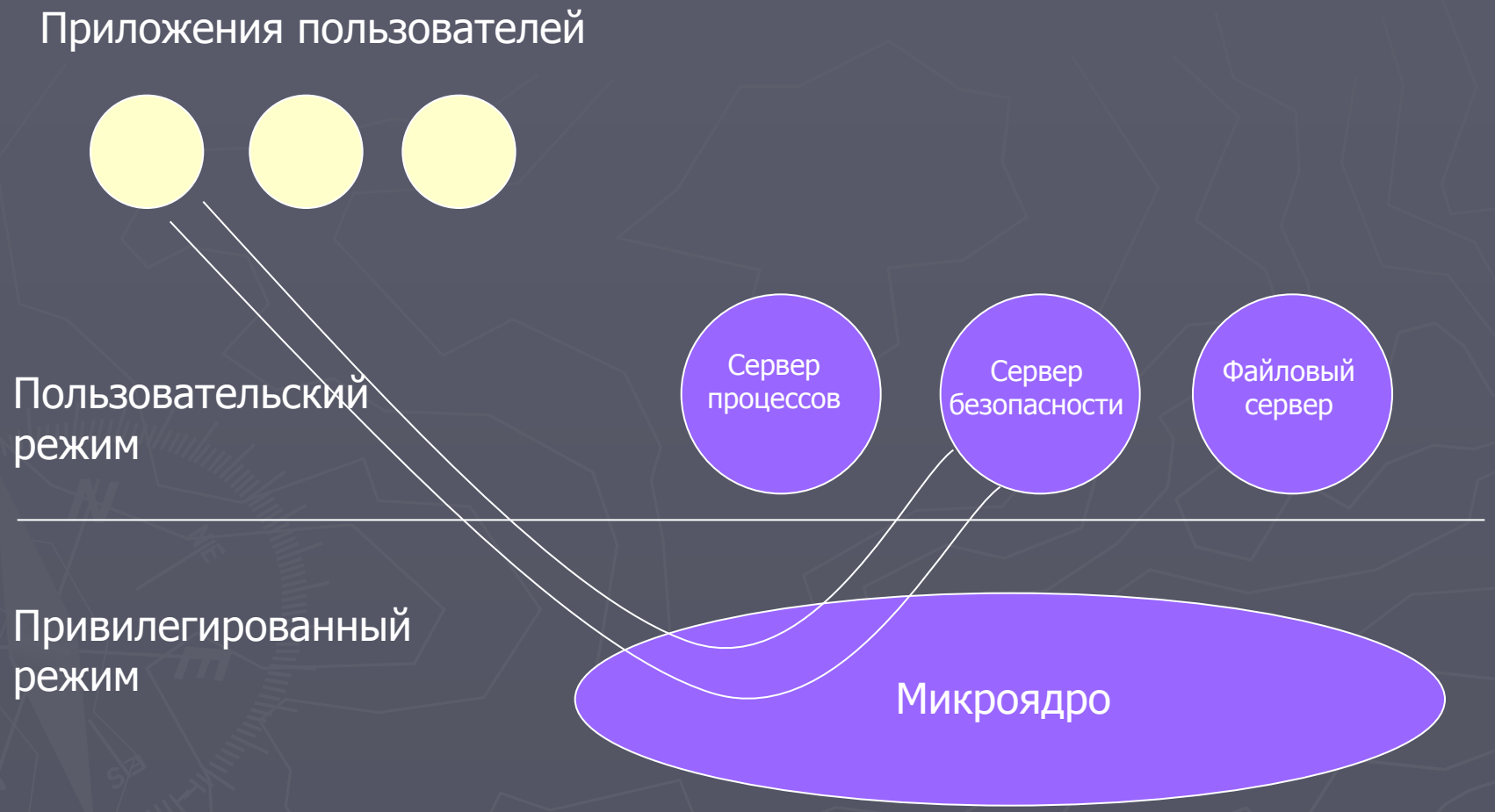
Переносимость операционной системы

- Под переносимостью операционной системы понимается способность использования ОС на различных аппаратных платформах с минимальными изменениями в ее структуре. Для уменьшения числа машинно-зависимых модулей разработчики ОС ограничивают универсальность машинно-независимых модулей. Например, Windows разработана для нескольких типов процессоров и для многопроцессорных систем используются собственные модули.
- Для обеспечения переносимости следуют следующим правилам:
  - Большая часть кода написана на языке, трансляторы которого существуют для всех планируемых платформ;
  - Объем машино-зависимых частей кода должен быть минимизирован;
  - Аппаратно-зависимый код должен быть изолирован в нескольких модулях
- В идеале машино-зависимые модули ядра полностью экранируют остальную часть ОС от конкретных деталей аппаратной платформы (кэши, контроллеры прерываний и т.п.).

Микроядерная архитектура

- Концепция микроядерной архитектуры заключается в выделении в качестве работающего в привилегированном режиме части ОС, ответственном за небольшой набор системных функций (управление процессами, обработка прерываний, управление виртуальной памятью, пересылка сообщений). Данная часть ОС называется **микроядром**.
- Все остальные высокоуровневые функции ядра разрабатываются в виде приложений, работающих в пользовательском режиме – **серверы ОС**.
- Взаимодействие между обычными приложениями и серверами ОС осуществляется через механизм обращений. **Клиентское приложение** отправляет запрос к серверу ОС через микроядро ОС. Такой механизм обеспечивает защиту работы приложений.

Микроядерная архитектура



Достоинства микроядерной архитектуры

- Операционные системы, основанные на микроядерной архитектуре обладают рядом преимуществ, предъявляемых к современным ОС:
  - Переносимость (обусловлена малым числом модулей в аппаратно-зависимом микроядре)
  - Расширяемость (добавление новых функций связано с включением новых серверов ОС)
  - Надежность (обусловлена изолированностью процессов)
  - Поддержка распределенных вычислений (используется механизм взаимодействия приложений аналогичный взаимодействию в распределенных системах)
- Недостаток
  - Производительность (обладают меньшей производительностью)

Совместимость операционных систем

- Совместимость – возможность операционной системы выполнять приложения, написанные для других ОС.
- Выделяют
  - **Двоичная совместимость** – на уровне кодов (программные модули могут быть просто перенесены и запущены)
  - **Совместимость исходных текстов** – приложения могут быть перекомпилированы в новый исполняемый модуль для ОС.
- Совместимость на уровне кодов может быть достигнута с помощью **эмуляции** двоичного кода.

Прикладные программные среды

- Прикладная программная среда – совокупность средств ОС, предназначенная для организации выполнения приложений, использующих определенную систему машинных команд, определенный тип API.
- Каждая ОС создает хотя бы одну программную среду.
- Для обеспечения совместимости различных программных сред используются решения:
  - эмуляция двоичного кода,
  - трансляция API.

Основные понятия, связанные с интерфейсом операционных систем

В области информационных технологий имеется несколько фундаментальных понятий. Одно из них – "интерфейс".

Интерфейс в широком смысле – определенная стандартами граница между взаимодействующими независимыми объектами. Интерфейс задает параметры, процедуры и характеристики взаимодействия объектов.

Система связей и взаимодействия устройств компьютера.

Средства взаимодействия пользователей с операционной системой компьютера, или пользовательской программой.

средства отображения информации, отображаемая информация, форматы и коды;

командные режимы, язык пользователь-интерфейс;

устройства и технологии ввода данных;

диалоги, взаимодействие и транзакции между пользователем и компьютером;

обратная связь с пользователем;

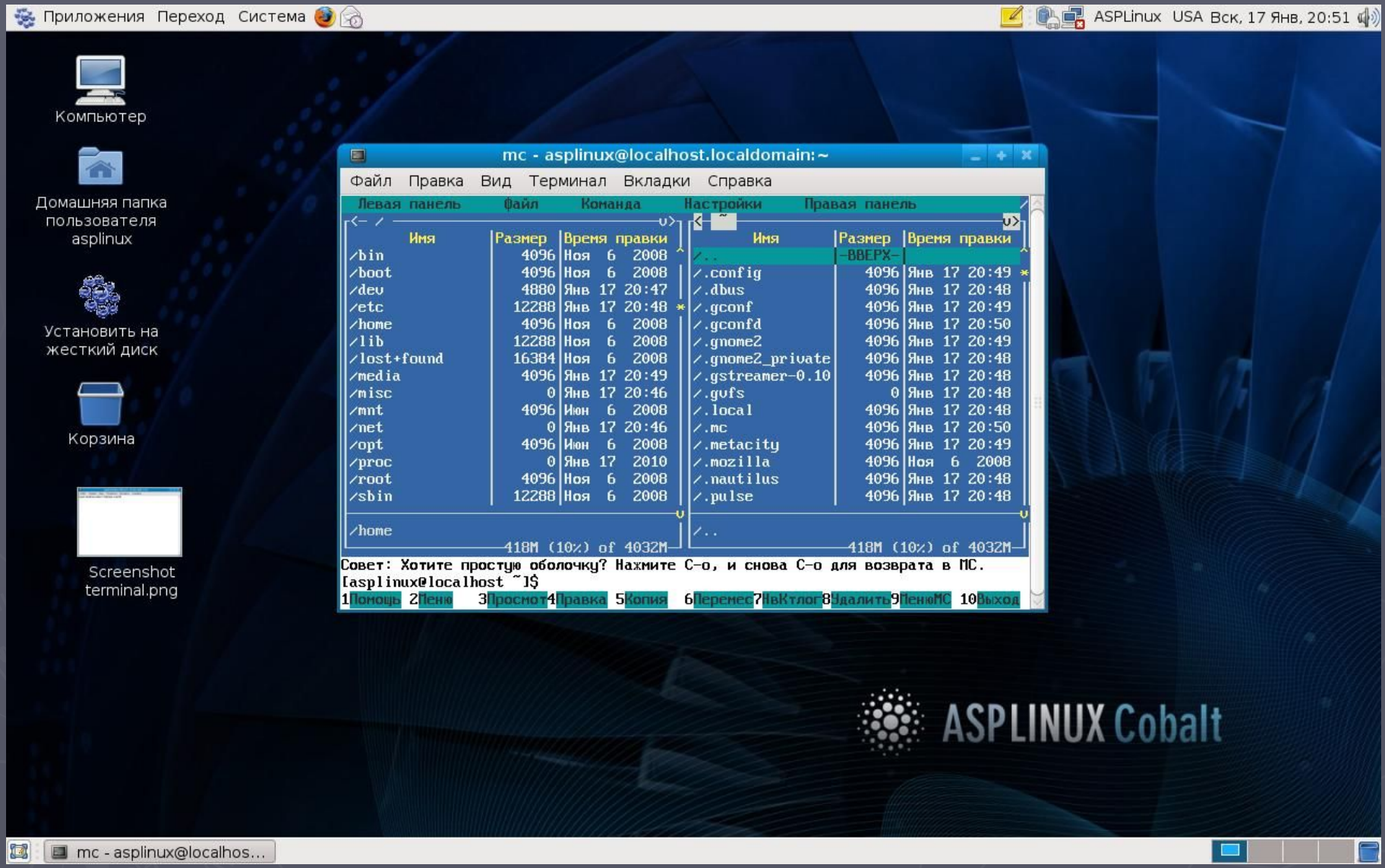
поддержка принятия решений в конкретной предметной области;

порядок использования программы и документация на нее".

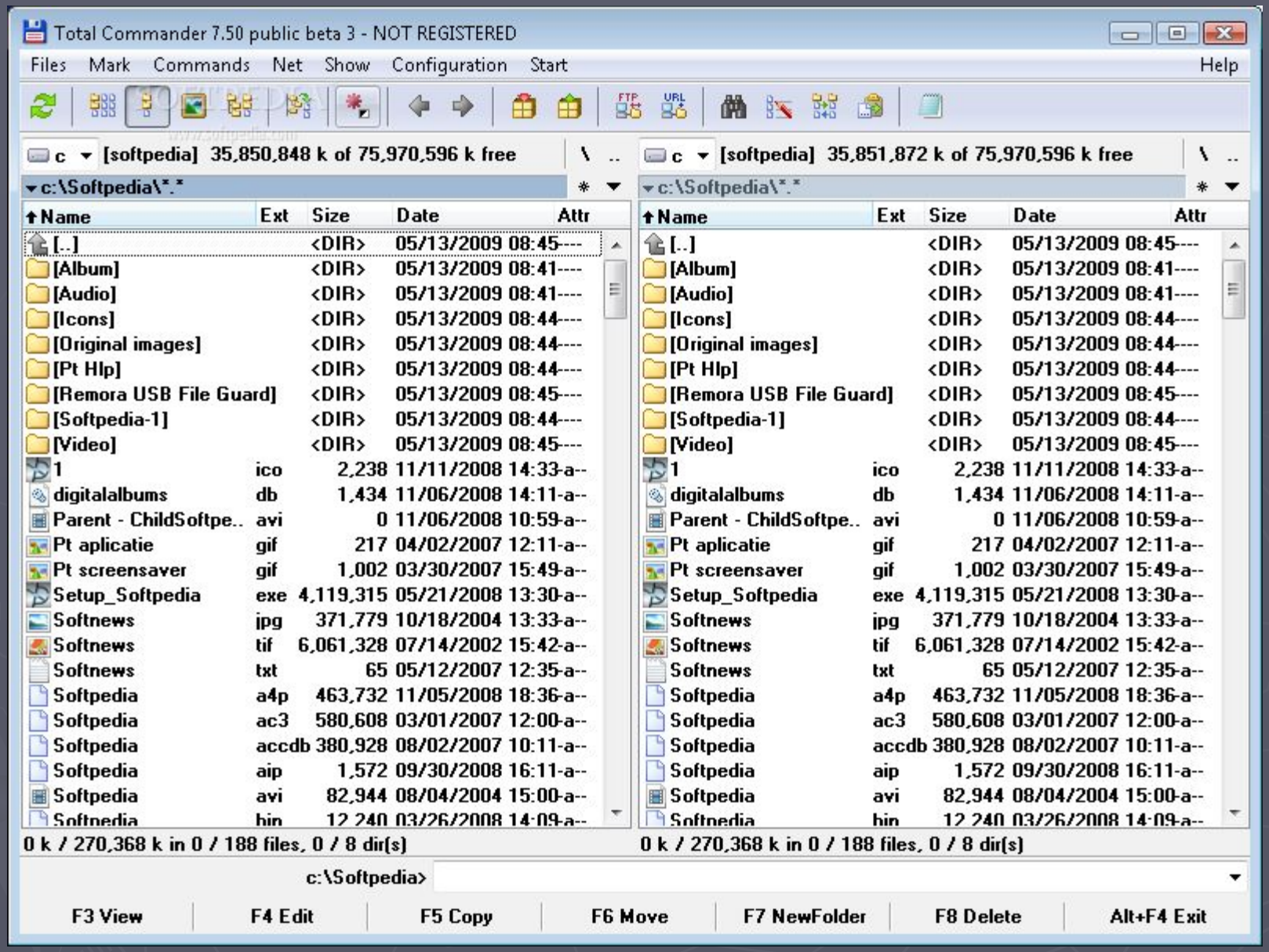
```
[asplinux@asplinuxlive ~]#
```

```
[asplinux@asplinuxlive ~]# data
Пят Янв 22 14:10:00 MSD 2010
```

Легендарный файловый менеджер Norton Commander



Программа Midnight Commander, выполненная в Терминале



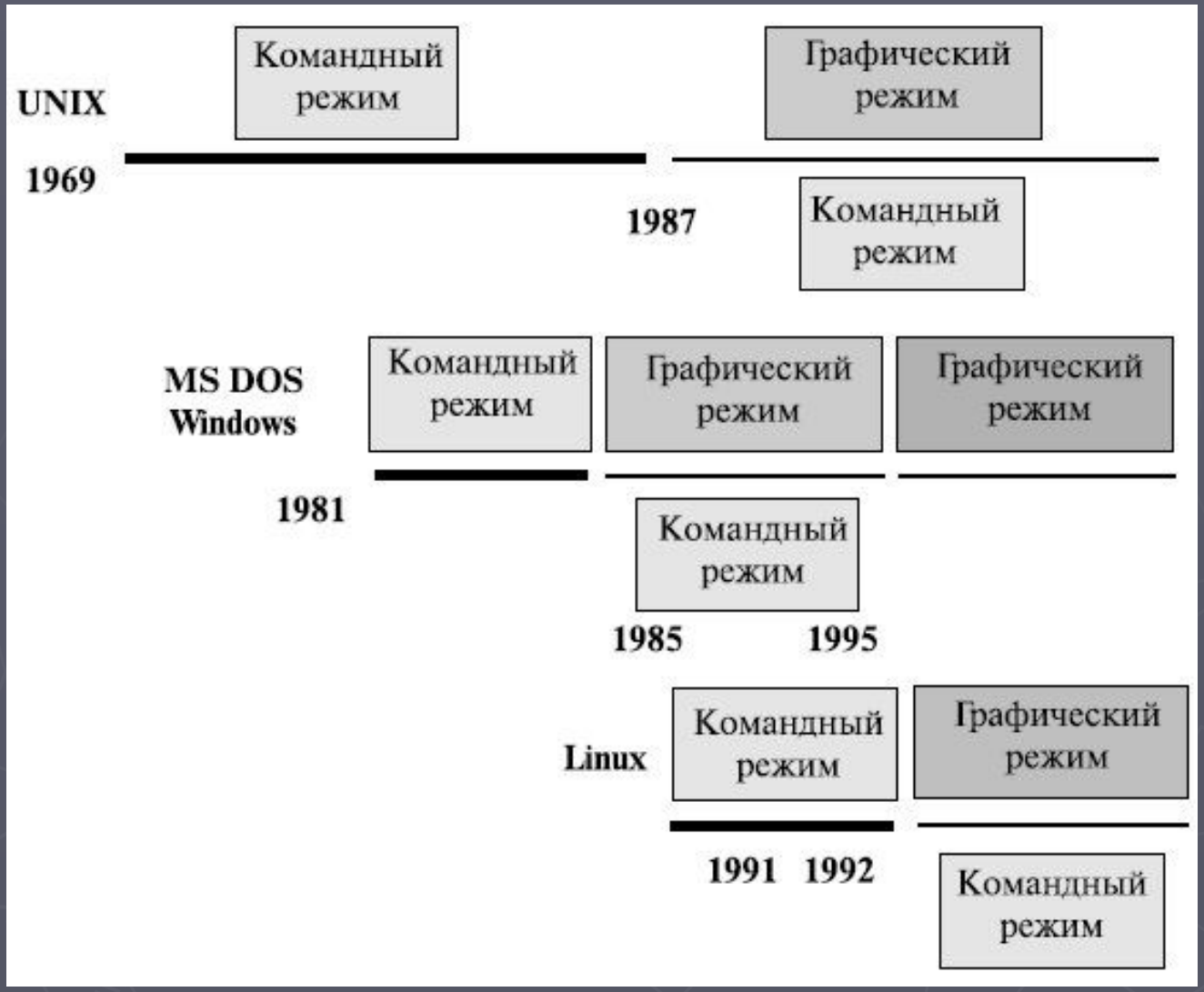
Файловый менеджер Total Commander

WIMP-интерфейс (Window – окно, Image – образ, Menu – меню, Pointer – указатель). Этот вид интерфейса реализован на двух уровнях технологий: простой графический интерфейс и "чистый" WIMP-интерфейс.

SILK-интерфейс (Speech – речь, Image – образ, Language – язык, Knowledge – знание). Этот вид интерфейса наиболее требователен к аппаратным ресурсам компьютера, и поэтому его применяют в основном для военных целей.

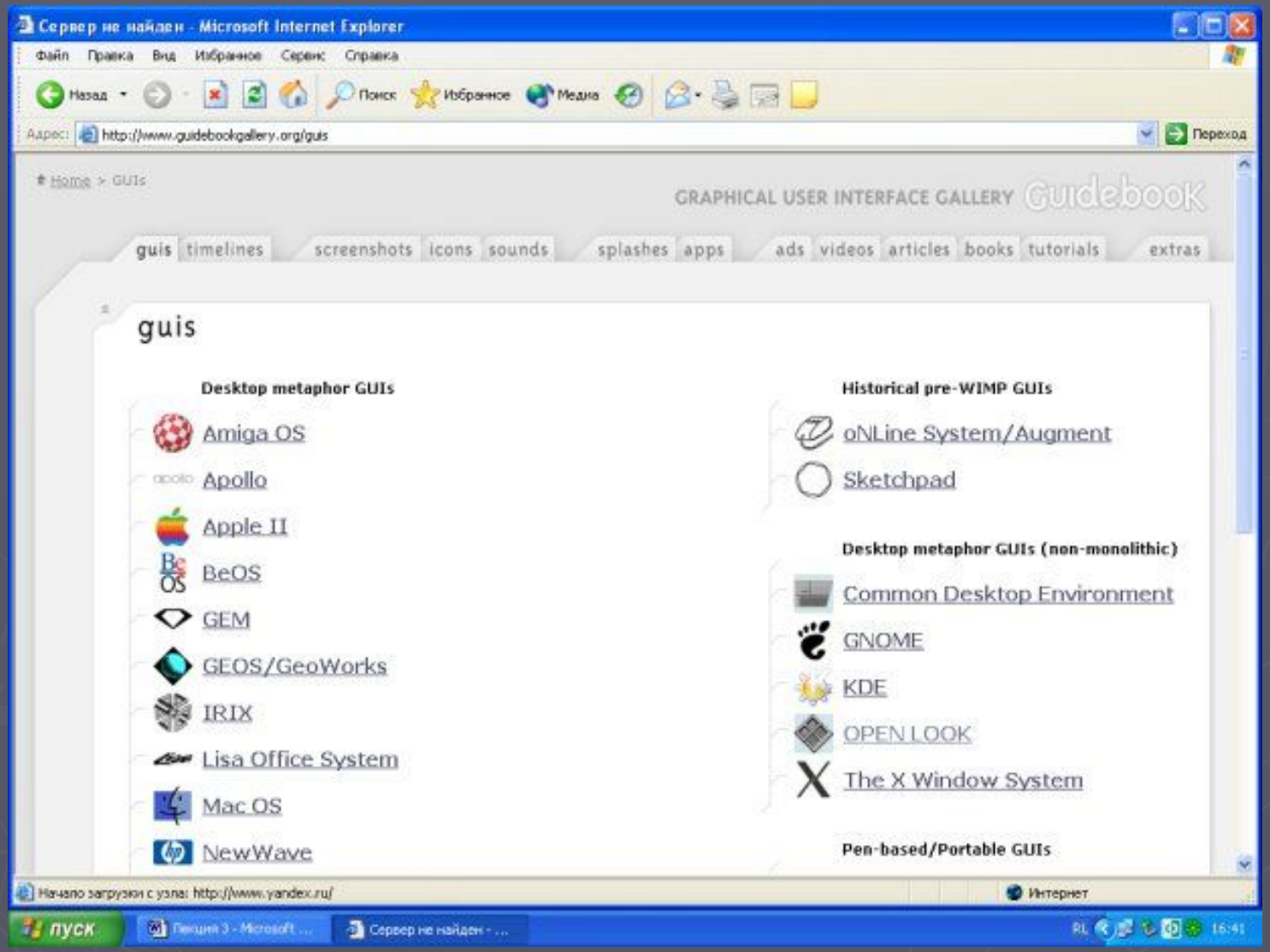
Для обозначения графического режима используют аббревиатуру GUI (Graphics User Interface), что дословно переводят как "графический интерфейс пользователя", но часто при переводе заменяют на "многооконный графический интерфейс".

Первый графический интерфейс от фирмы Xerox

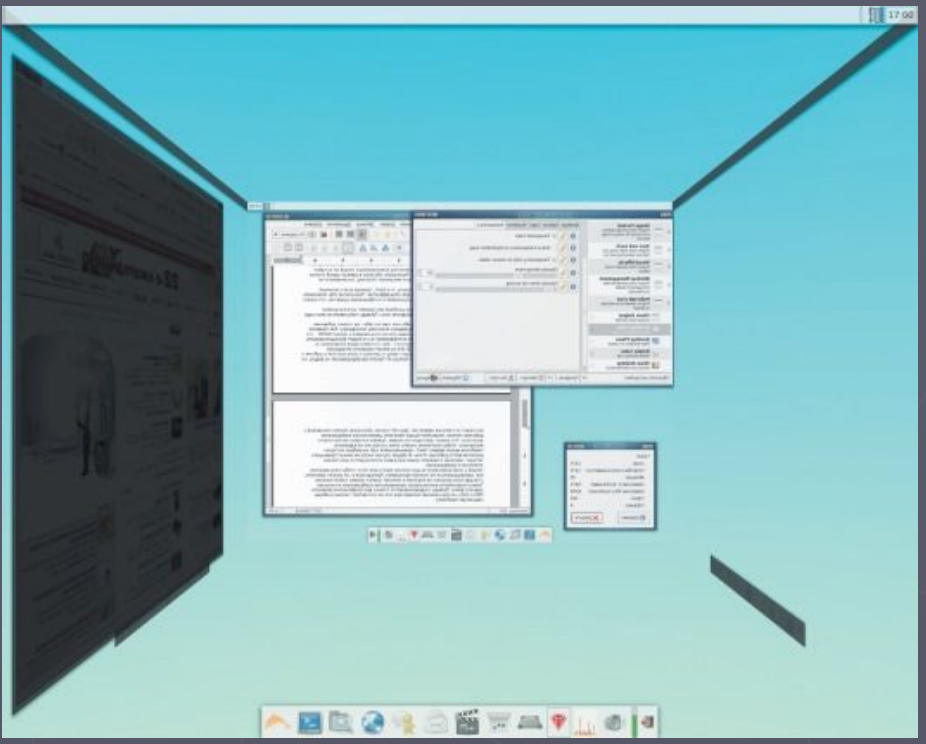


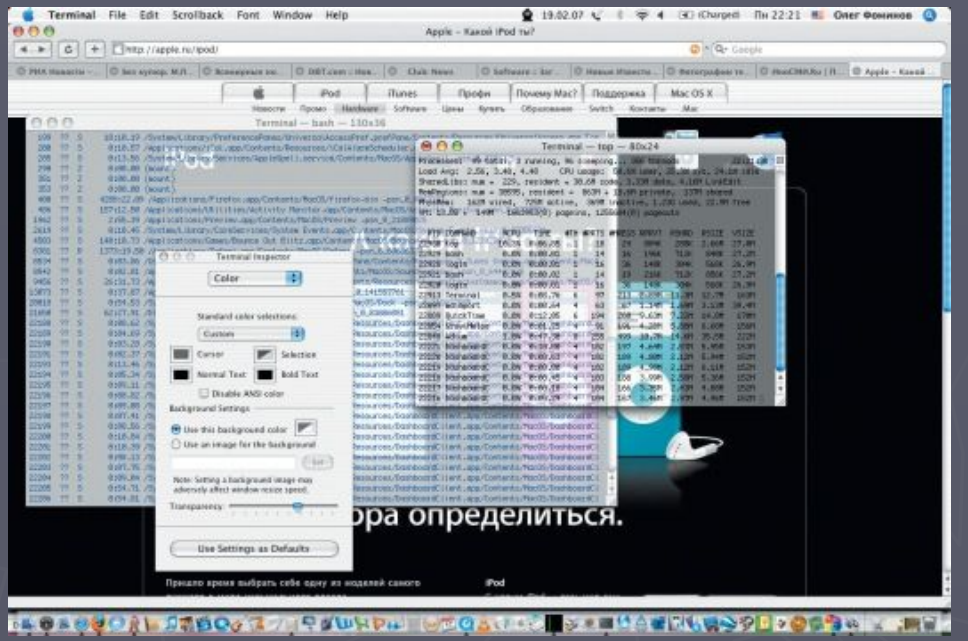
Командный и графический интерфейс семейства UNIX/Linux и Windows

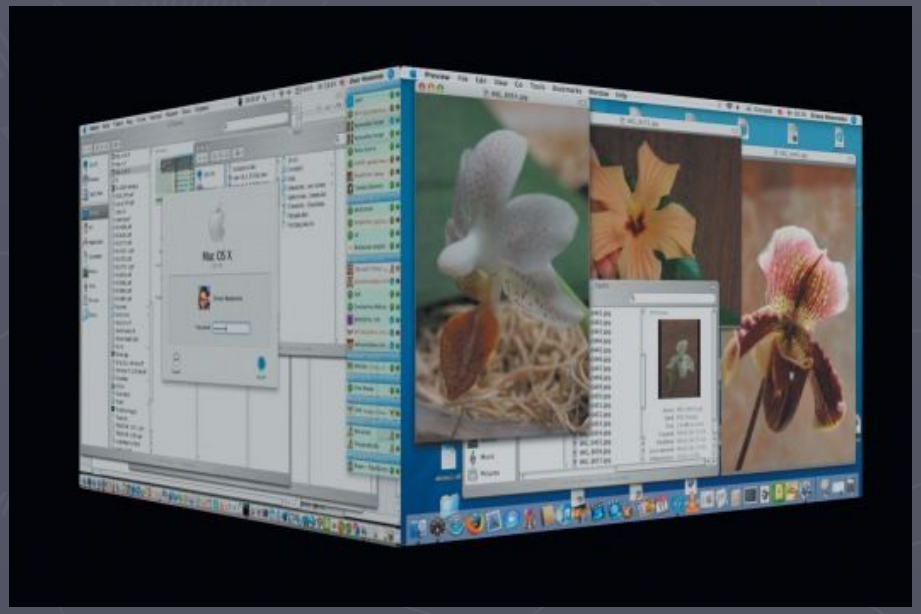
Графический интерфейс OPENSTEP Jan 1997 платформы

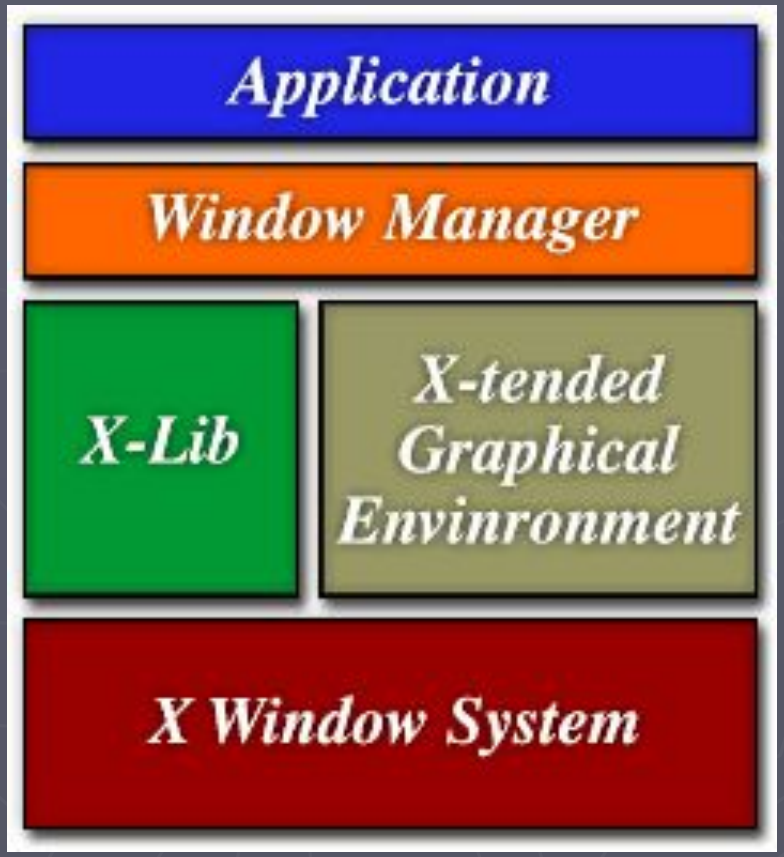


Галерея графических интерфейсов на разной аппаратуре

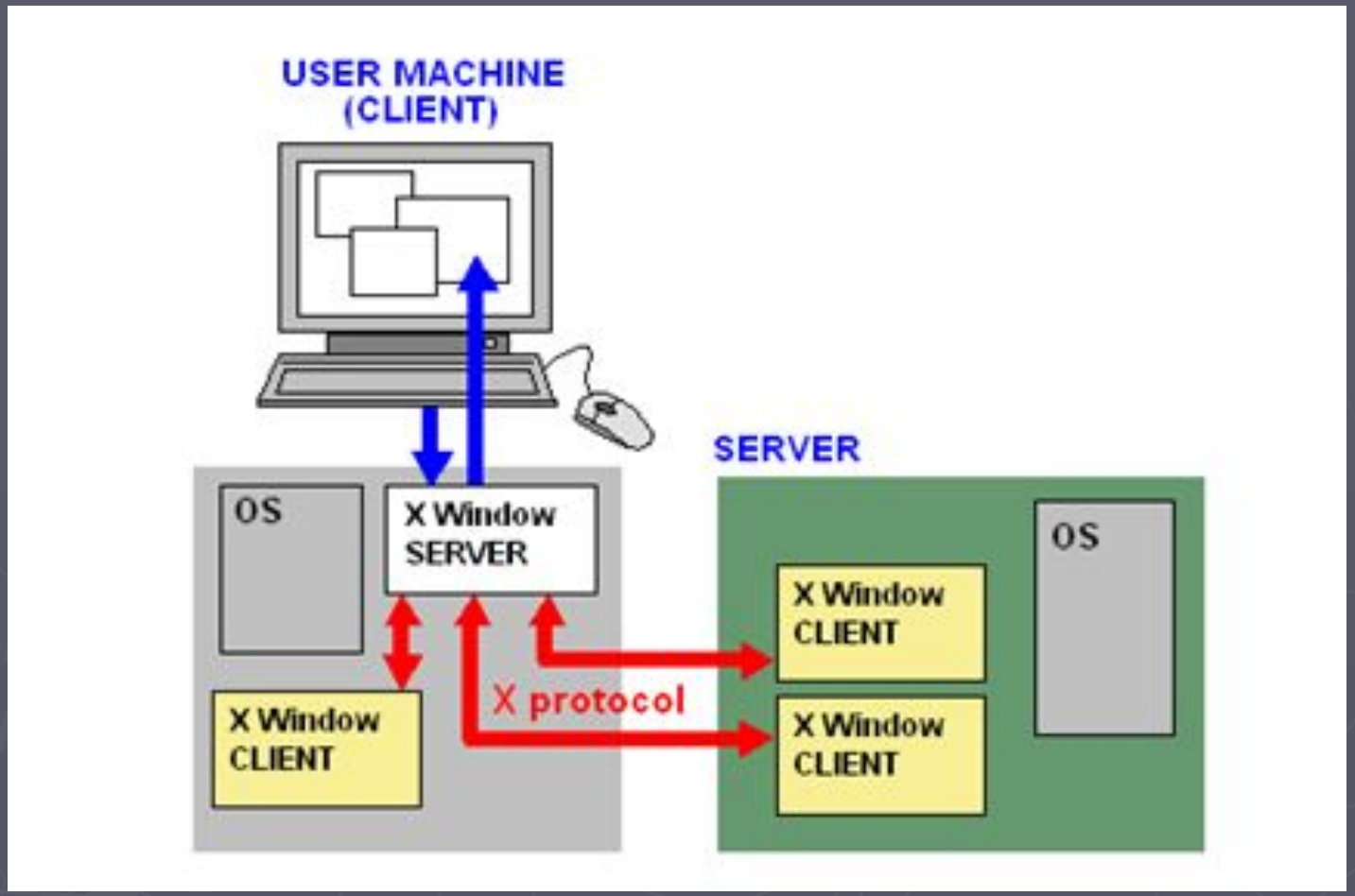



Terminal File Edit Scrollback Font Window Help
Apple - Какой iPod ты?
iPod
iTunes
Профи
Почему Mac?
Поддержка
Mac OS X
Terminal — bash — 110x36
Terminal — top — 80x24
Terminal Inspector
Color
Standard color selections:
Custom
Cursor
Selection
Normal Text
Bold Text
Disable ANSI color
Background Settings
Use this background color
Use an image for the background
Transparency:
Use Settings as Defaults
Пришло время выбрать себе одну из моделей самого
iPod





Архитектура X Window

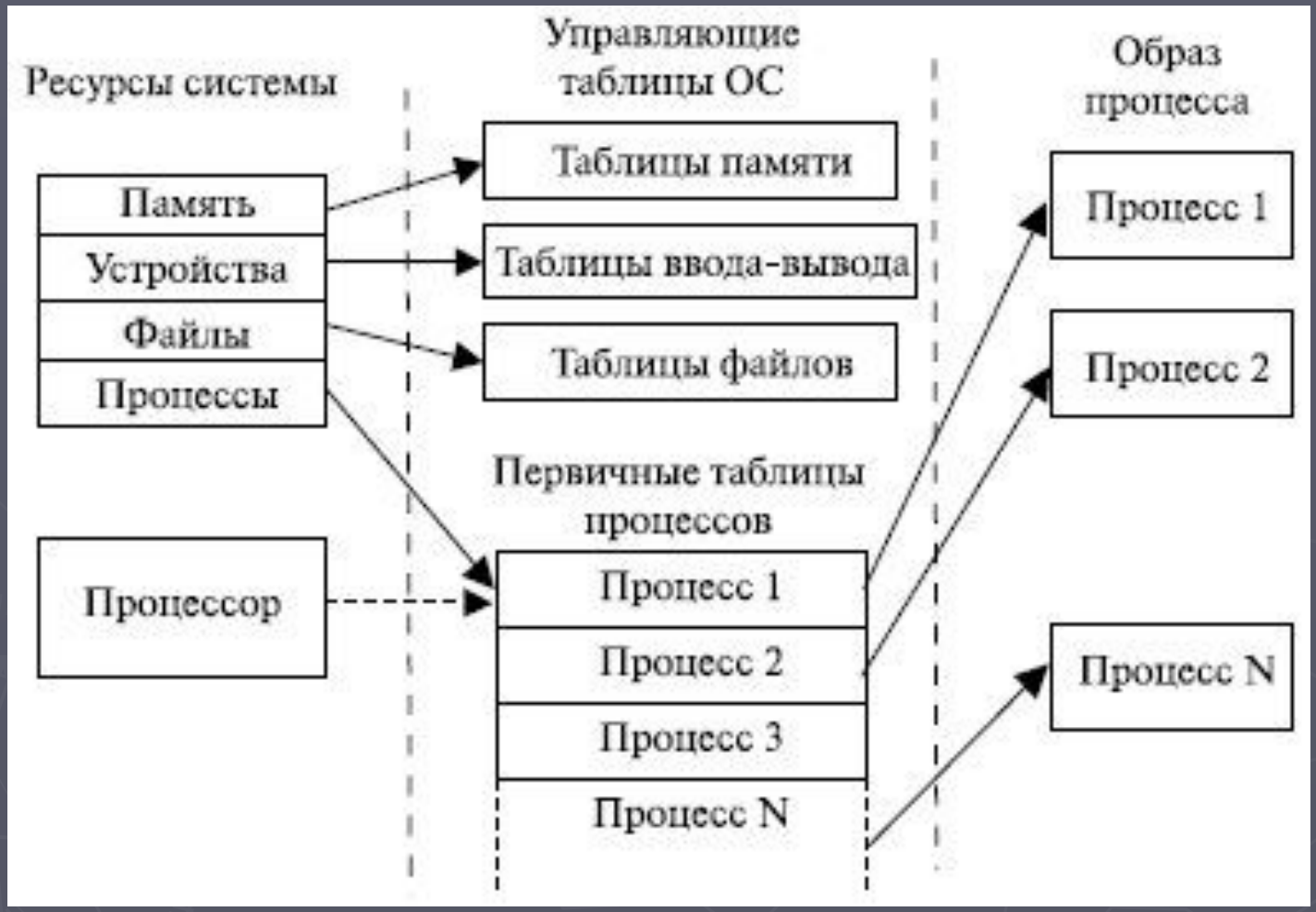


X Windows сервер выполняется на клиенте

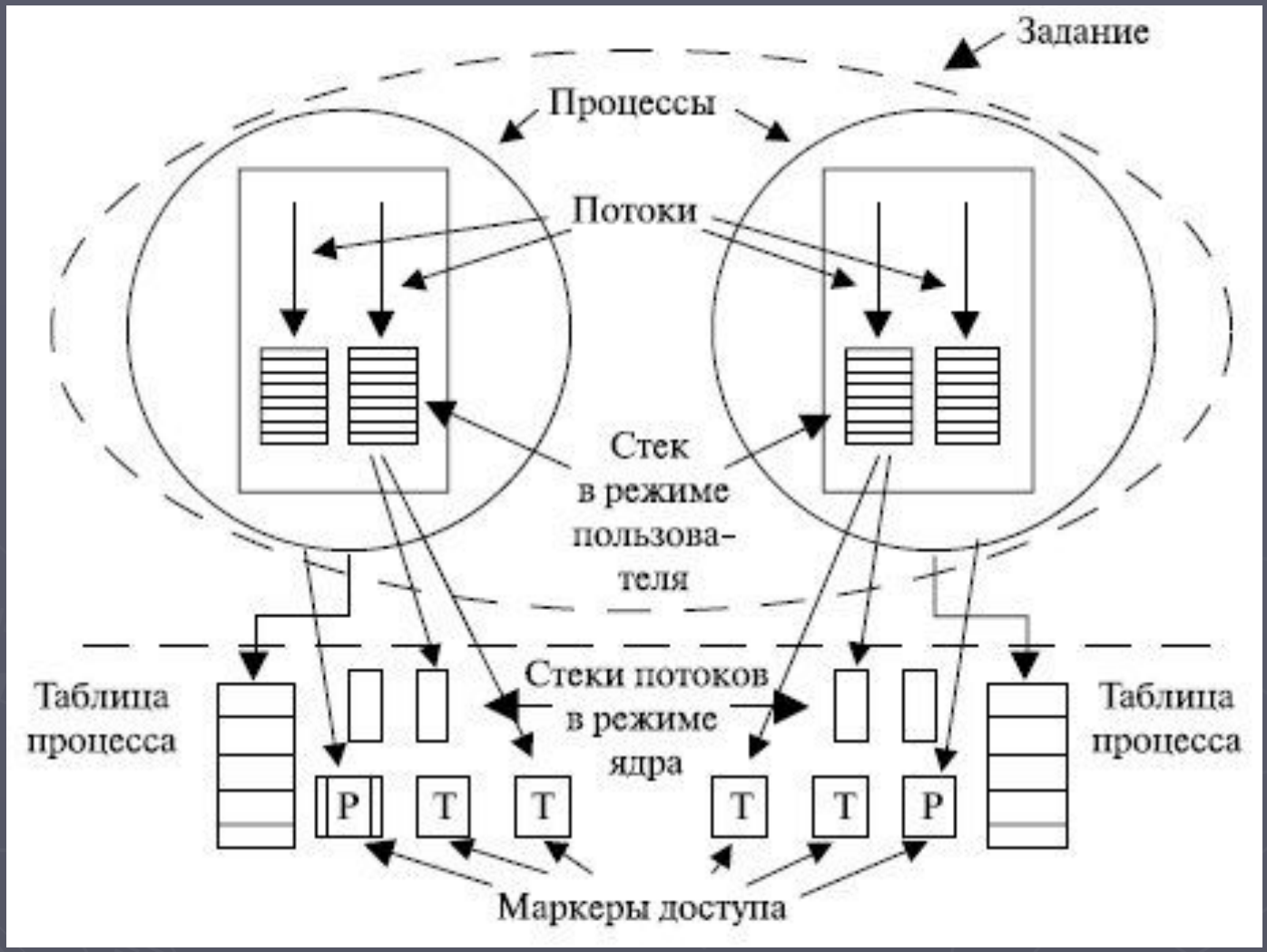

GNOME™

Концепция процессов и потоков.

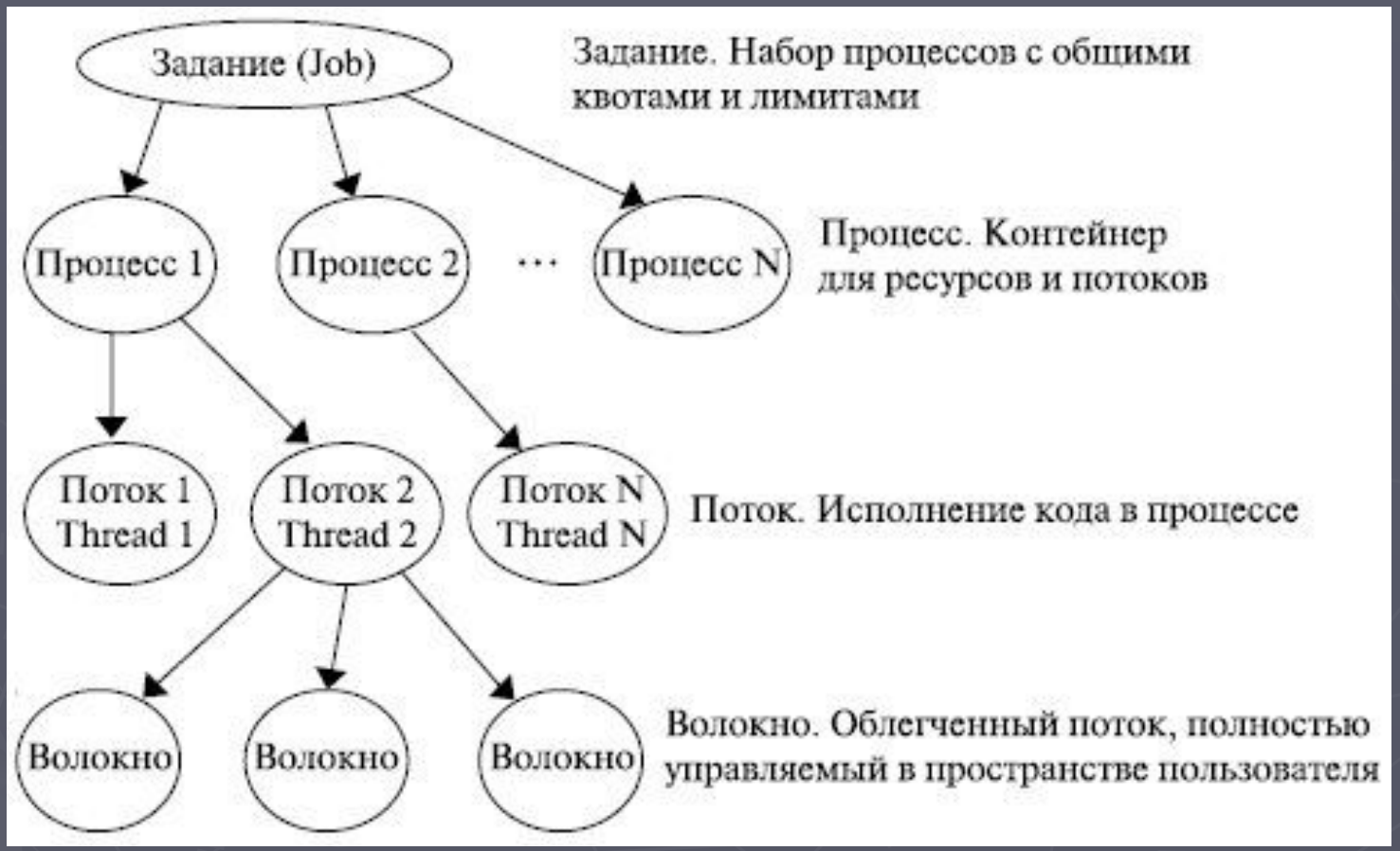
Задание, процессы, потоки (нити), волокна


Ресурсы системы
Управляющие таблицы ОС
Образ процесса
Память
Устройства
Файлы
Процессы
Процессор
Таблицы памяти
Таблицы ввода-вывода
Таблицы файлов
Первичные таблицы процессов
Процесс 1
Процесс 2
Процесс 3
Процесс N
Процесс 1
Процесс 2
Процесс N

Таблицы ОС



Задания, процессы, потоки



Иерархия рабочих единиц ОС

Мультипрограммирование призвано повысить эффективность использования вычислительной системы.

Наиболее характерными показателями эффективности вычислительных систем являются:

пропускная способность – количество задач, выполняемых системой в единицу времени;

удобство работы пользователей, заключающихся, в частности, в том, что они могут одновременно работать в интерактивном режиме с несколькими приложениями на одной машине;

реактивность системы – способность выдерживать заранее заданные (возможно, очень короткие) интервалы времени между запуском программы и получением конечного результата.

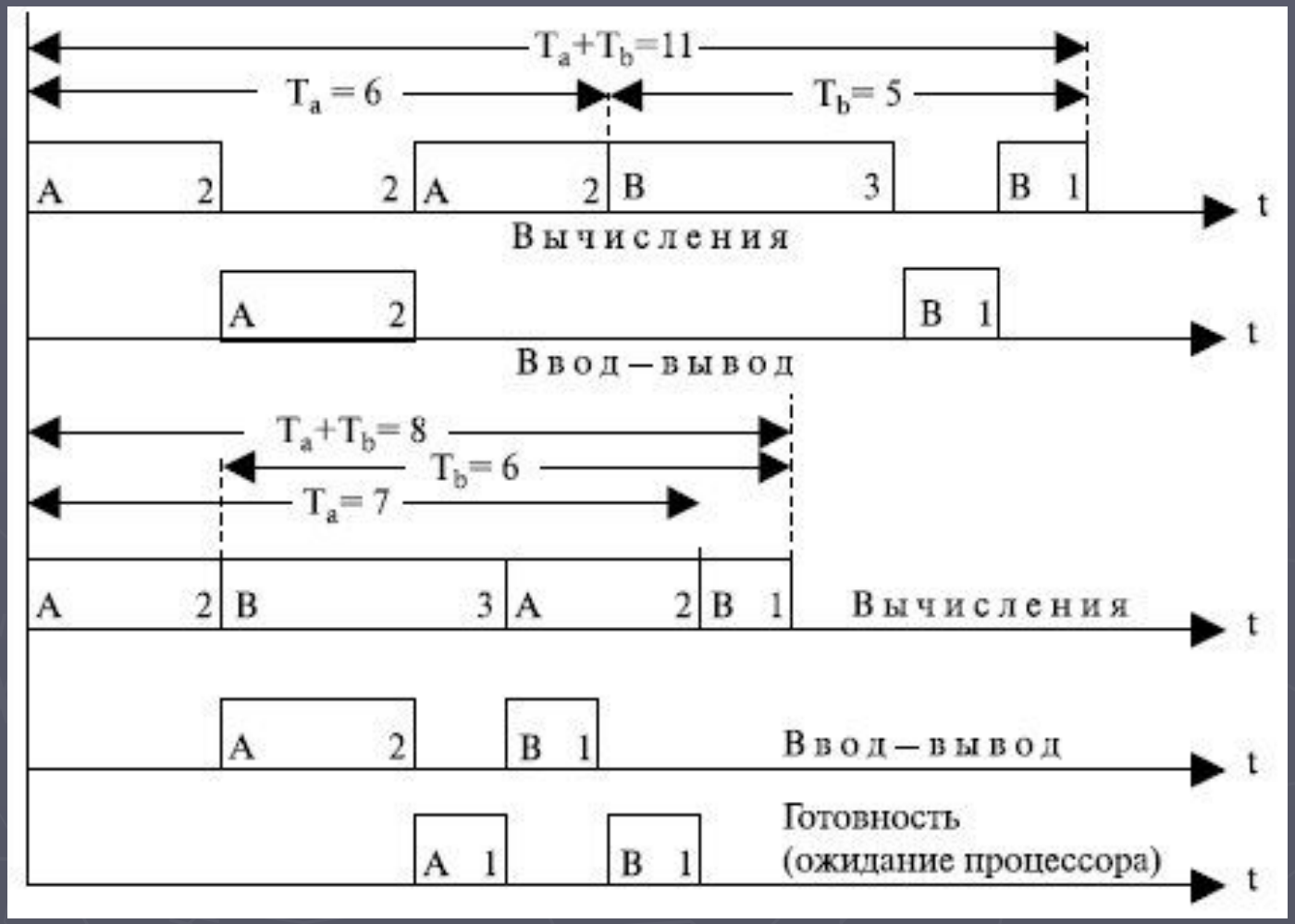


Иллюстрация эффекта мультипрограммирования

Система разделения времени

Управление процессами и потоками

Одной из основных подсистем любой современной мультипрограммной ОС, непосредственно влияющей на функционирование компьютера, является подсистема управления процессами и потоками.

Основные функции этой подсистемы:

создание процессов и потоков;

обеспечение процессов и потоков необходимыми ресурсами;

изоляция процессов;

планирование выполнения процессов и потоков;

диспетчеризация потоков;

организация межпроцессного взаимодействия;

синхронизация процессов и потоков;

завершение и уничтожение процессов и потоков.

К созданию процесса приводят пять основных событий:

инициализация ОС (загрузка);

выполнение запроса работающего процесса на создание процесса;

запрос пользователя на создание процесса, например, при входе в систему в интерактивном режиме;

инициирование пакетного задания;

создание операционной системой процесса, необходимого для работы каких-либо служб.

Диспетчеризация заключается в реализации найденного в результате планирования решения, т.е. в переключении процессора с одного потока на другой.

Диспетчеризация проходит в три этапа:

сохранение контекста текущего потока;

загрузка контекста потока, выбранного в результате планирования;

запуск нового потока на выполнение.

Создание процессов и потоков. Модели процессов и потоков

Создать процесс – это, прежде всего, создать описатель процесса: несколько информационных структур, содержащих все сведения о процессе, необходимые операционной системе для управления им.

В число таких сведений могут входить: идентификатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности процесса.

Примерами таких описателей процесса являются:

блок управления задачей (TCB – Task Control Block) в OS/360;

управляющий блок процесса (PCB – Process Control Block) в OS/2;

дескриптор процесса в UNIX;

объект-процесс (object-process) в Windows NT/2000/2003.

Информация	Описание
Данные пользователя	Изменяемая часть пользовательского адресного пространства (данные программы, пользовательский стек, модифицируемый код)
Пользовательская программа	Программа, которую необходимо выполнить
Системный стек	Один или несколько системных стеков для хранения параметров и адресов вызова процедур и системных служб
Управляющий блок процесса	Данные, необходимые операционной системе для управления процессом

В дескрипторе процесса содержится такая информация о процессе, которая необходима ядру в течение всего жизненного цикла процесса независимо от того, находится он в активном или пассивном состоянии и находится ли образ в оперативной памяти или на диске.

Эту информацию можно разделить на три категории:

информация по идентификации процесса;

информация по состоянию процесса;

информация, используемая при управлении процессом.

Информация по состоянию и управлению процессом включает следующие основные данные:

состояние процесса, определяющее готовность процесса к выполнению;

данные о приоритете;

информация о событиях;

указатели, позволяющие определить расположение образа процесса в оперативной памяти и на диске;

указатели на другие ресурсы;

флаги, сигналы и сообщения;

данные о привилегиях, определяющих права доступа;

указатели на ресурсы, которыми управляет процесс;

сведения по истории использования ресурсов и процессора;

информация, связанная с планированием.

В контексте процесса содержится следующая основная информация:

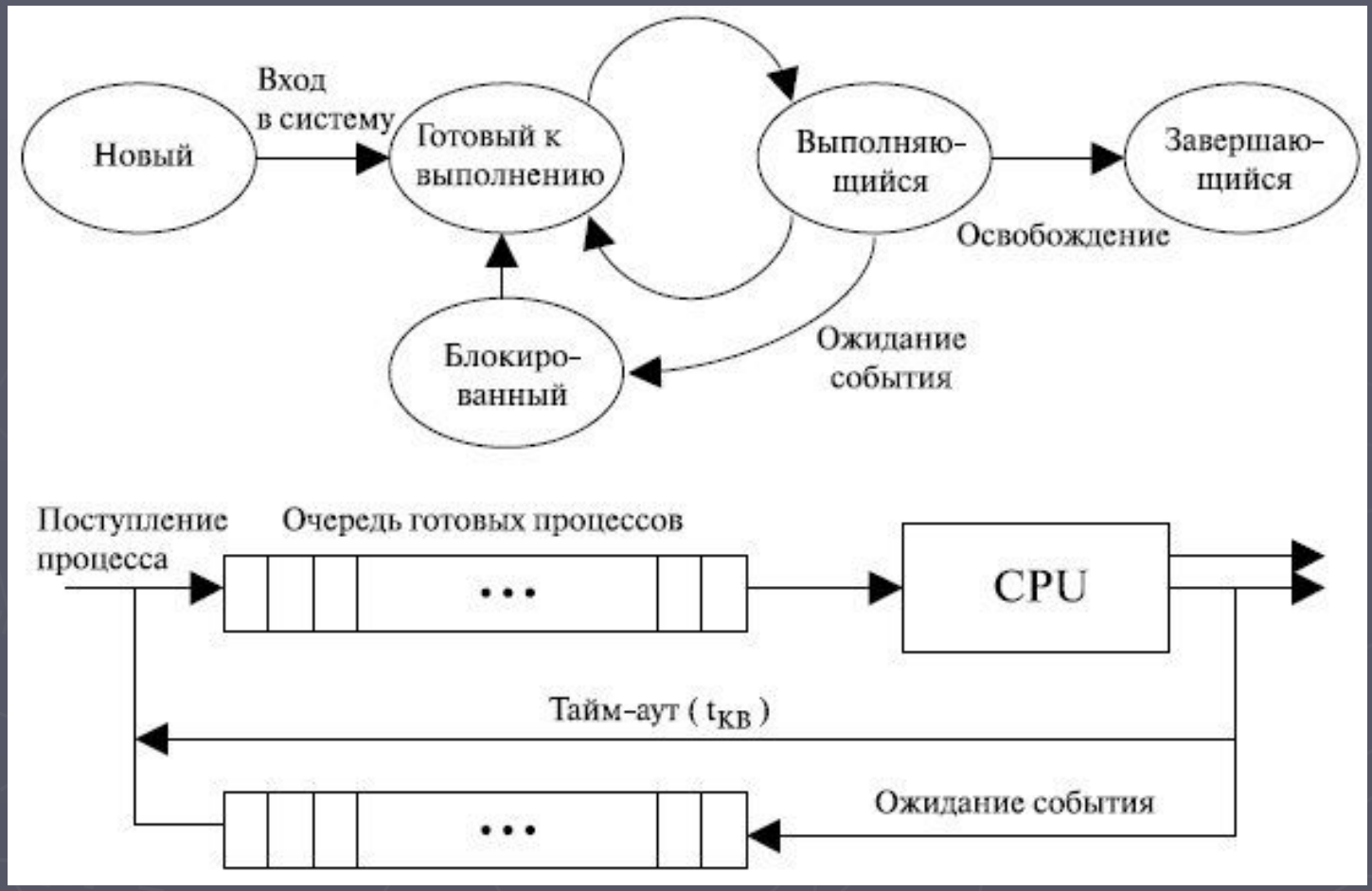
содержимое регистров процессора, доступных пользователю;

содержимое счетчика команд;

состояние управляющих регистров и регистров состояния;

коды условий, отражающие результат выполнения последней арифметической или логической операции;

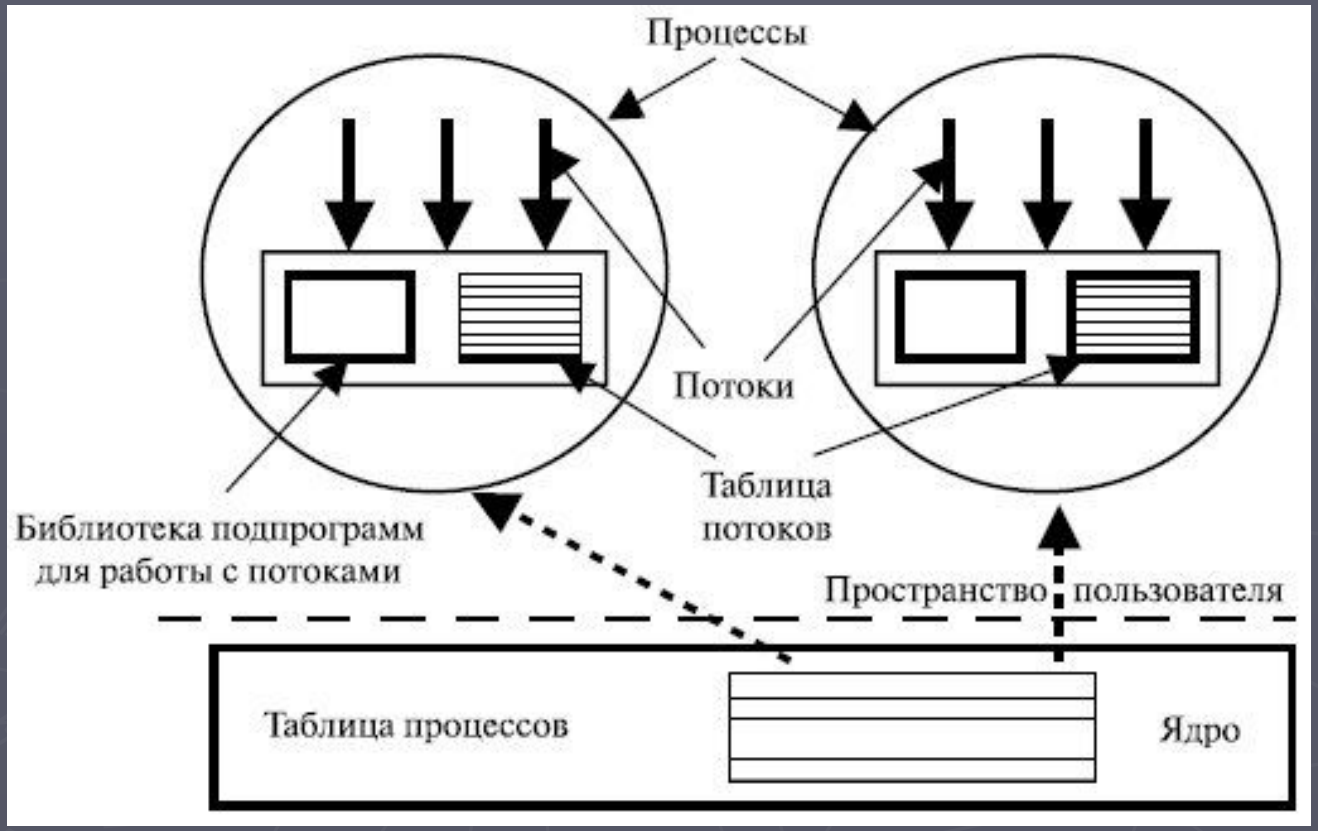
указатели вершин стеков, хранящие параметры и адреса вызова процедур и системных служб.



Состояния процесса

Есть два способа реализации пакета потоков:

- в пространстве пользователя или на уровне пользователя (User-level threads – ULT);
- в ядре или на уровне ядра (kernel-level threads – KLT).



Потоки в пространстве пользователя

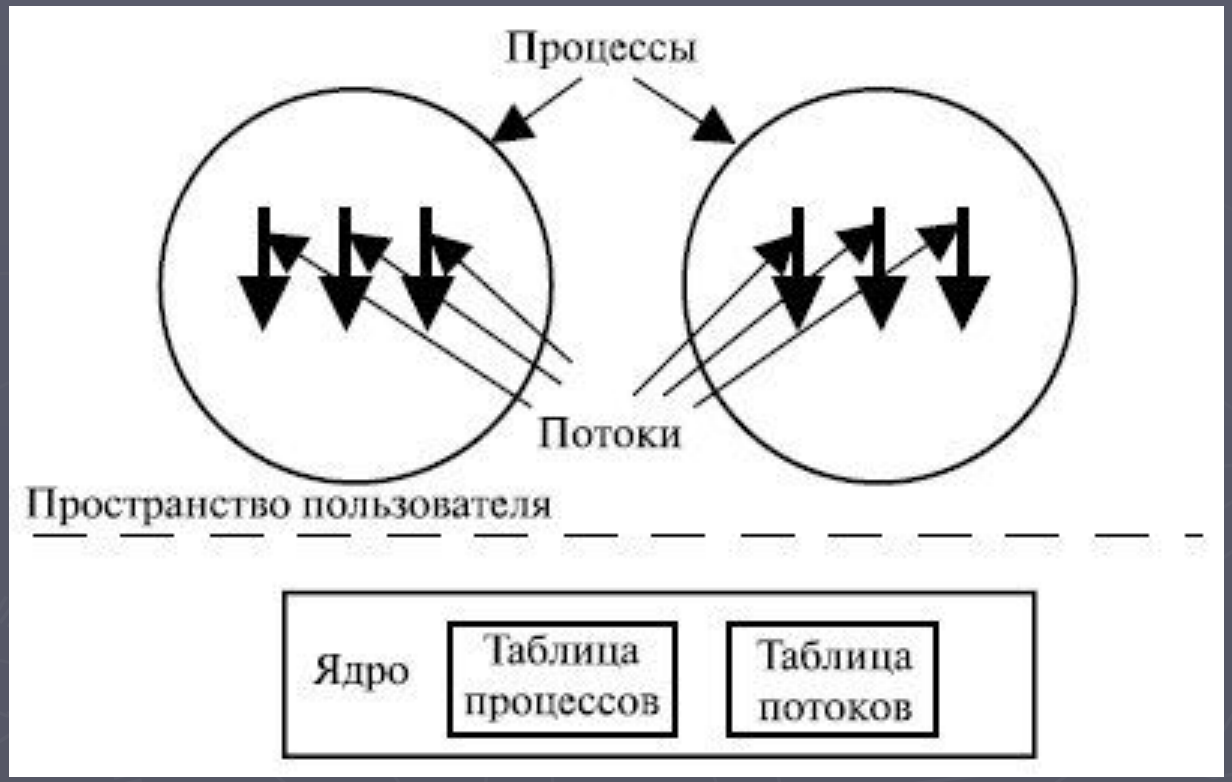
Использование потоков на уровне пользователя имеет следующие преимущества:

1. высокая производительность, поскольку для управления потоками процессу не нужно переключаться в режим ядра и обратно;

2. имеется возможность использования различных алгоритмов планирования потоков в различных приложениях (процессах) с учетом их специфики;

3. использование потоков на пользовательском уровне применимо для любой операционной системы

- в типичной ОС многие системные вызовы являются блокирующими;
- в стратегии с наличием потоков только на пользовательском уровне приложение не может воспользоваться преимуществом многопроцессорной системы;
- при запуске одного потока ни один другой поток не будет запущен, пока первый добровольно не отдаст процессор.



Потоки в пространстве ядра

Возможно планирование работы нескольких потоков одного и того же процесса на нескольких процессорах:

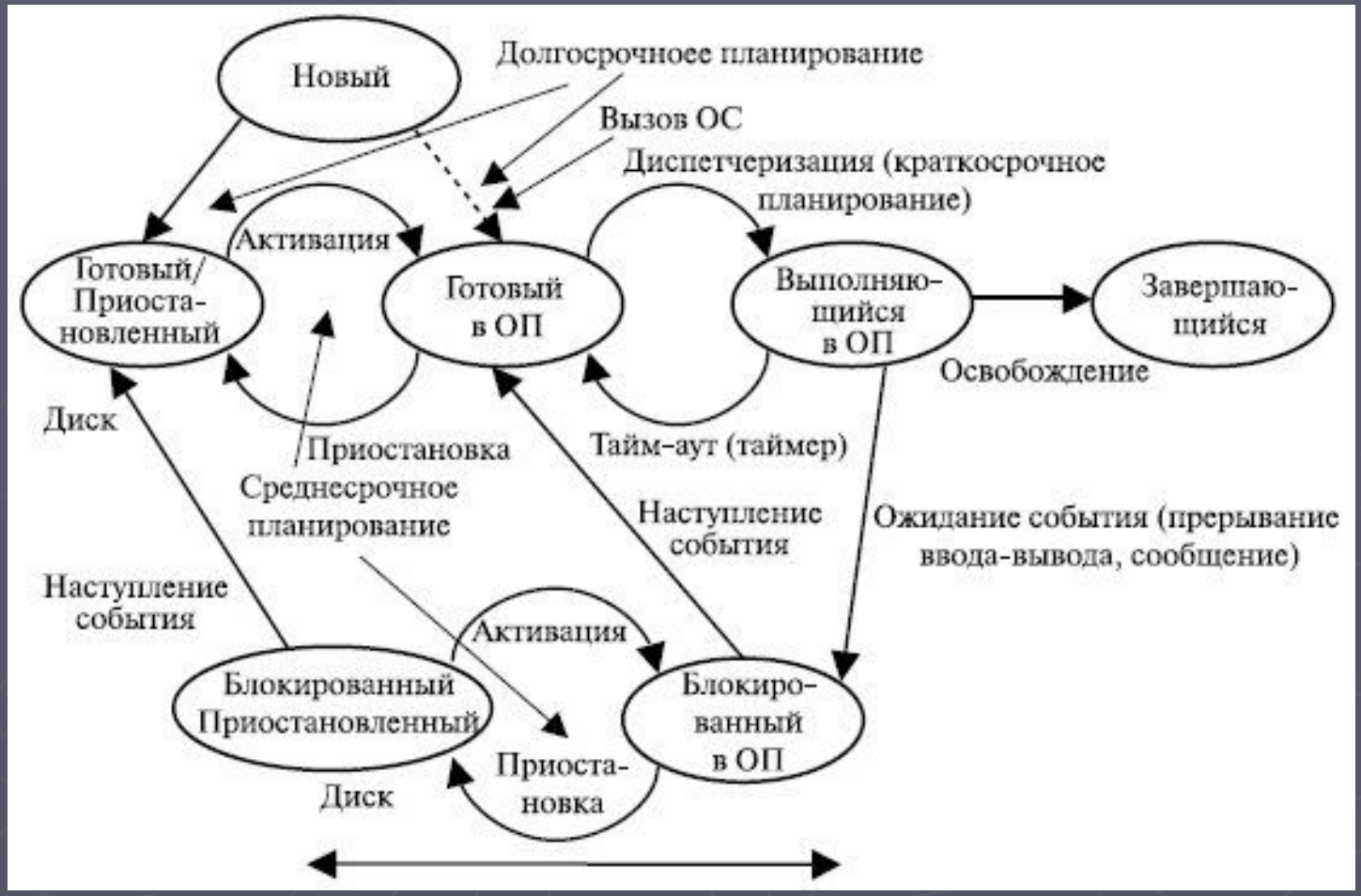
1. реализуется мультипрограммирование в режимах нескольких процессов (вообще – всех);

2. при блокировке одного из потоков процесса ядро может выбрать для выполнения другой поток этого же процесса;

3. процедуры ядра могут быть многопоточными.

Планирование заданий, процессов и потоков

Вид планирования	Выполняемые функции
Долгосрочное	Решение о добавлении задания (процесса) в пул выполняемых в системе
Среднесрочное	Решение о добавлении процесса к числу процессов, полностью или частично размещенных в основной памяти
Краткосрочное	Решение о том, какой из доступных процессов (потоков) будет выполняться процессором
Планирование ввода-вывода	Решение о том, какой из запросов процессов (потоков) на операцию ввода-вывода будет выполняться свободным устройством ввода-вывода

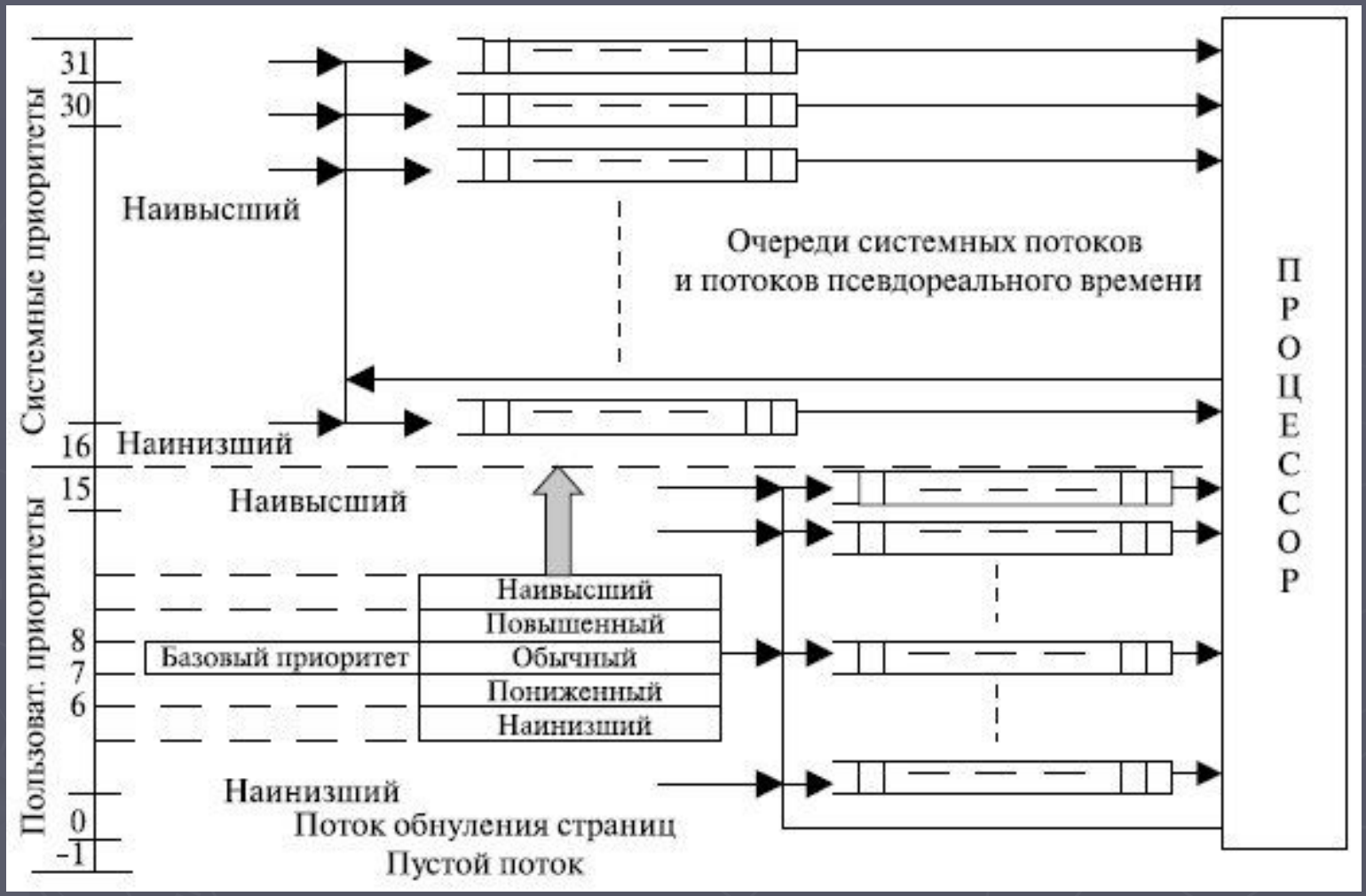


Место планирования в графе процессов

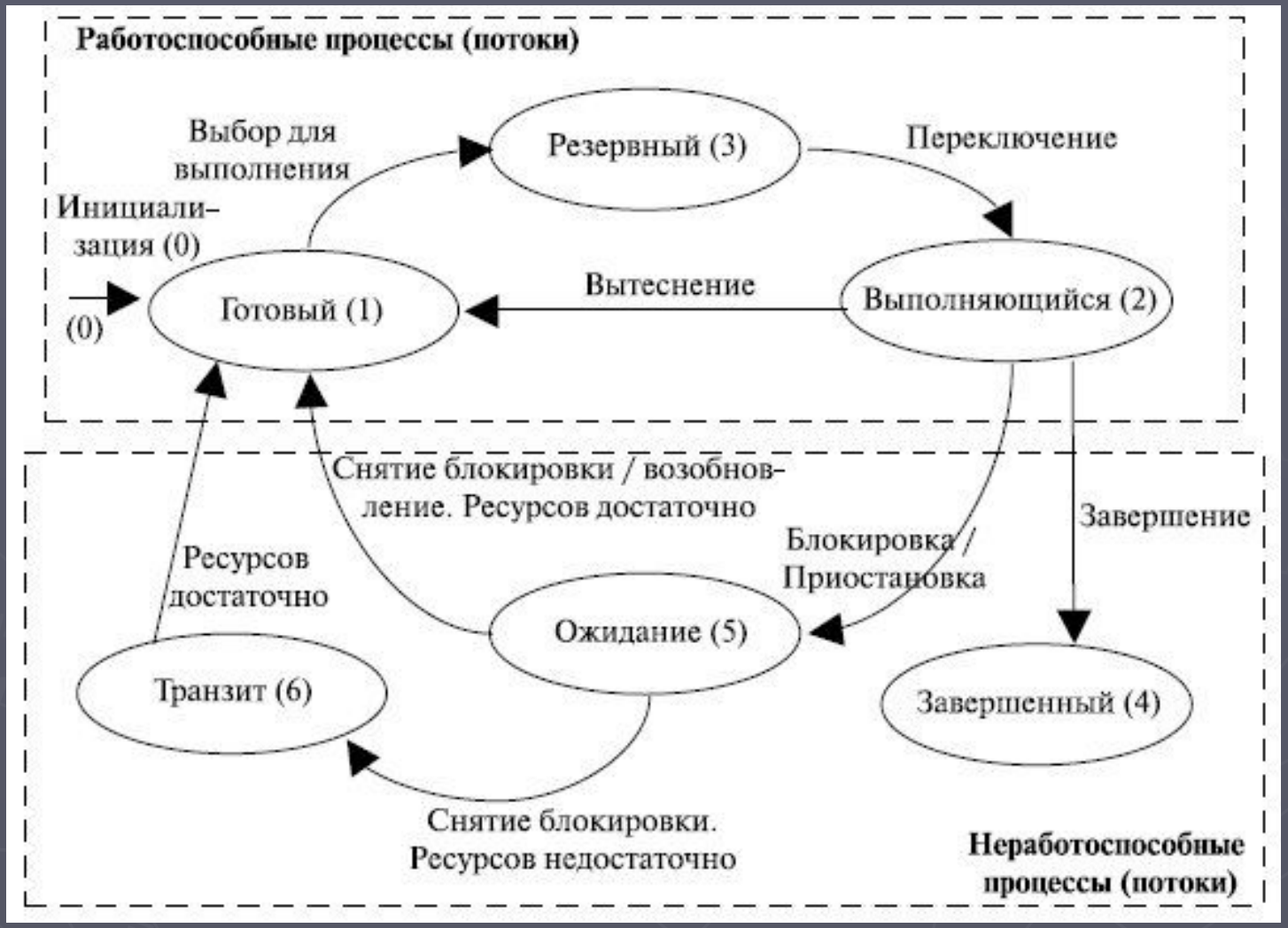
Краткосрочный планировщик (диспетчер) реализует найденное решение, т.е. переключает процессор с одного процесса (потока) на другой.

Примерами этих событий могут быть:

- прерывание таймера;
- прерывание ввода-вывода;
- вызовы операционной системы;
- сигналы.



Планирование в Windows



Состояния потоков в Windows

Способы взаимодействия процессов (потоков) можно классифицировать по степени осведомленности одного процесса о существовании другого.

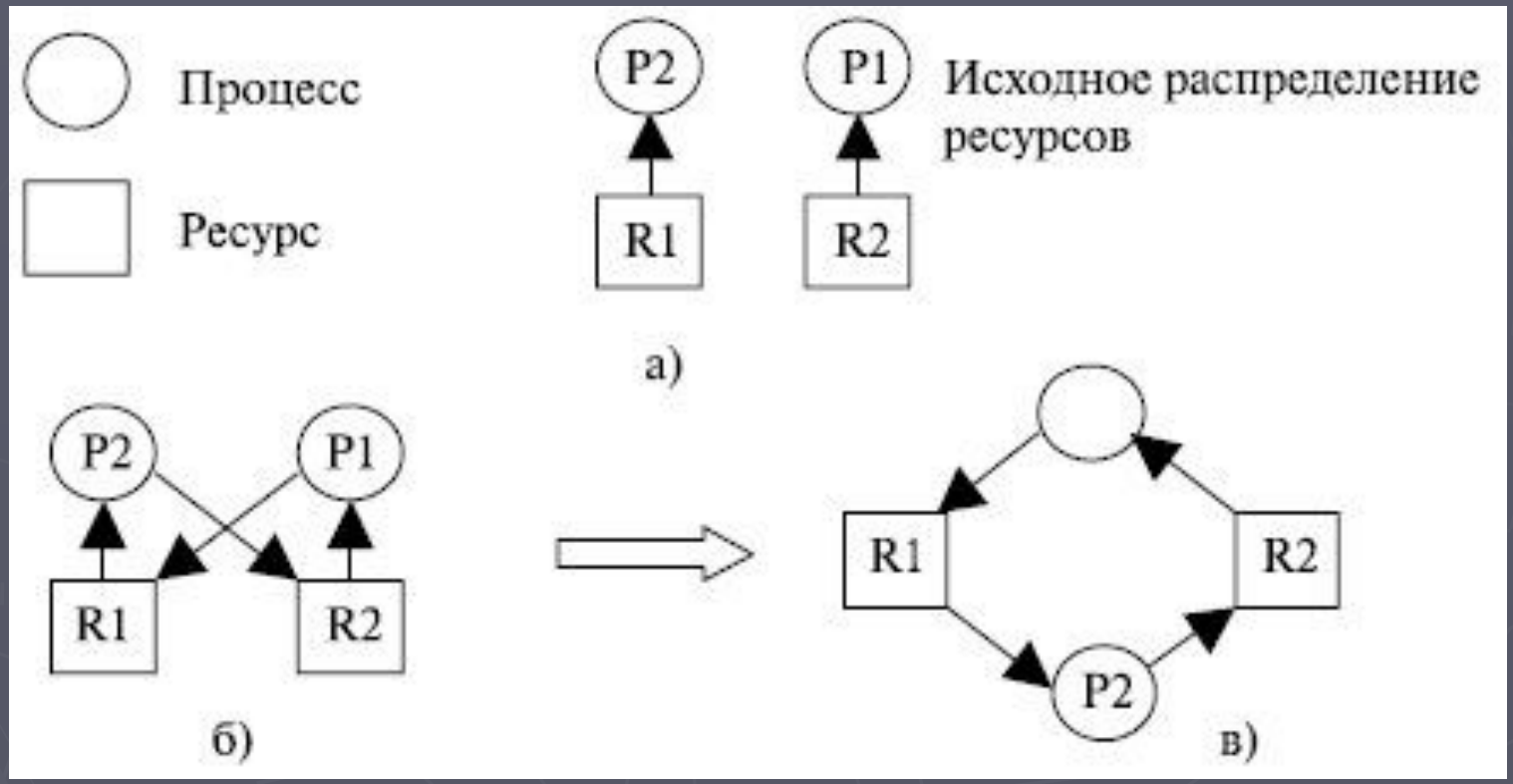
Процессы не осведомлены о наличии друг друга. ОС должна регулировать такие обращения.

Процессы косвенно осведомлены о наличии друг друга. Такие процессы демонстрируют сотрудничество при разделении общего объекта.

Процессы непосредственно осведомлены о наличии друг друга. Эти процессы также демонстрируют сотрудничество при работе.

Степень осведомленности	Взаимосвязь	Влияние одного процесса на другой	Потенциальные проблемы
Процессы не осведомлены друг о друге	Конкуренция	•Результат работы одного процесса не зависит от действий других. •Возможно влияние одного процесса на время работы другого	•Взаимоисключения •Взаимоблокировки •Голодание
Процессы косвенно осведомлены о наличии друг друга	Сотрудничество с использованием разделения	•Результат работы одного процесса может зависеть от информации, полученной от других. •Возможно влияние одного процесса на время работы другого	•Взаимоисключения •Взаимоблокировки •Голодание •Синхронизация
Процессы непосредственно осведомлены о наличии друг друга	Сотрудничество с использованием связи	•Результат работы одного процесса зависит от информации, полученной от других процессов. •Возможно влияние одного процесса на время работы другого	•Взаимоблокировки (расходуемые ресурсы) •Голодание

Критическая секция



Тупиковая ситуация

Пусть имеются два процесса, представленные последовательностью неделимых (атомарных) операций:

P: a; b; c; и Q: d; e; f;

где a, b, c, d, e, f – атомарные операции.

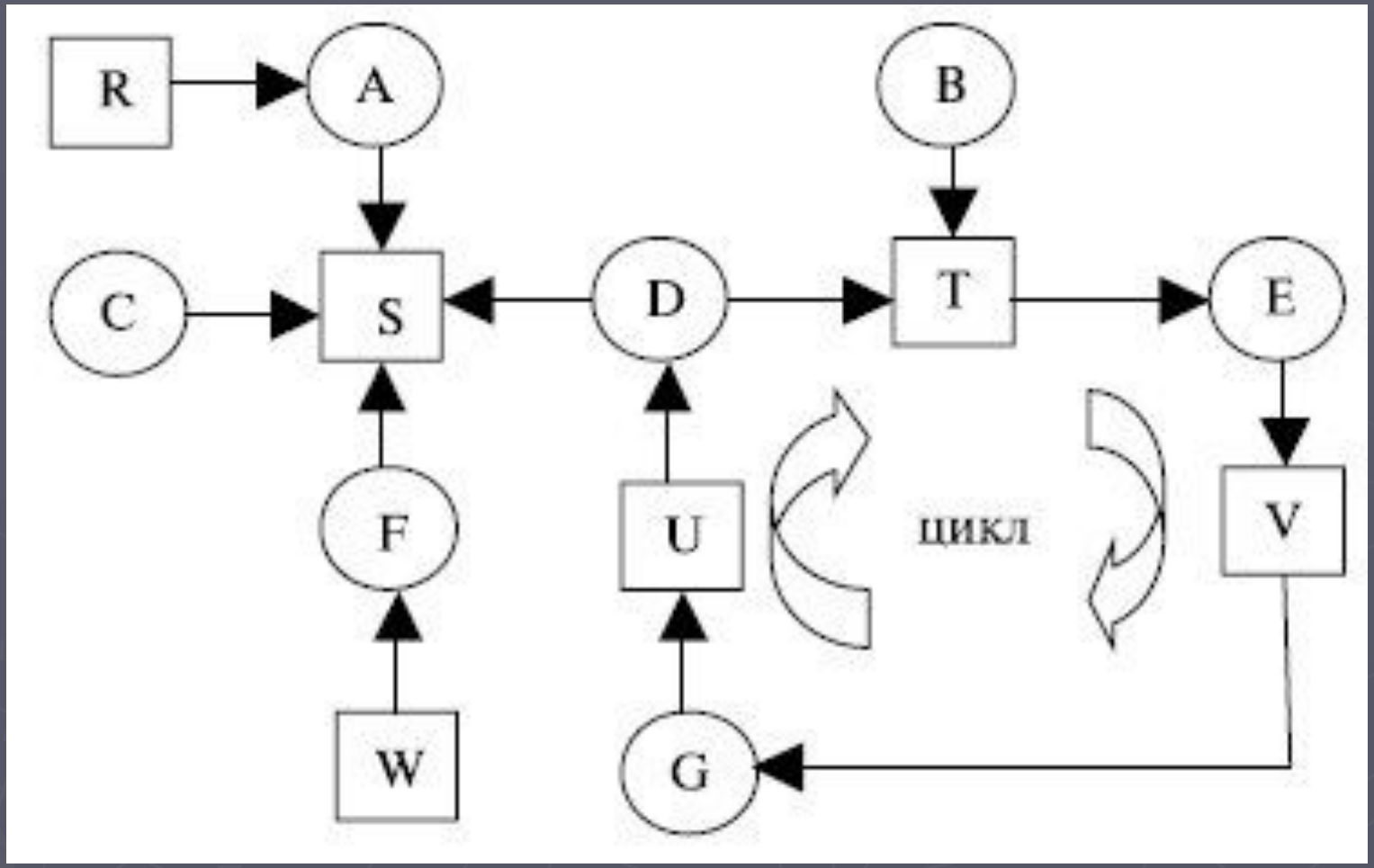
При последовательном выполнении активностей мы получаем следующую последовательность атомарных действий:

PQ: a b c d e f

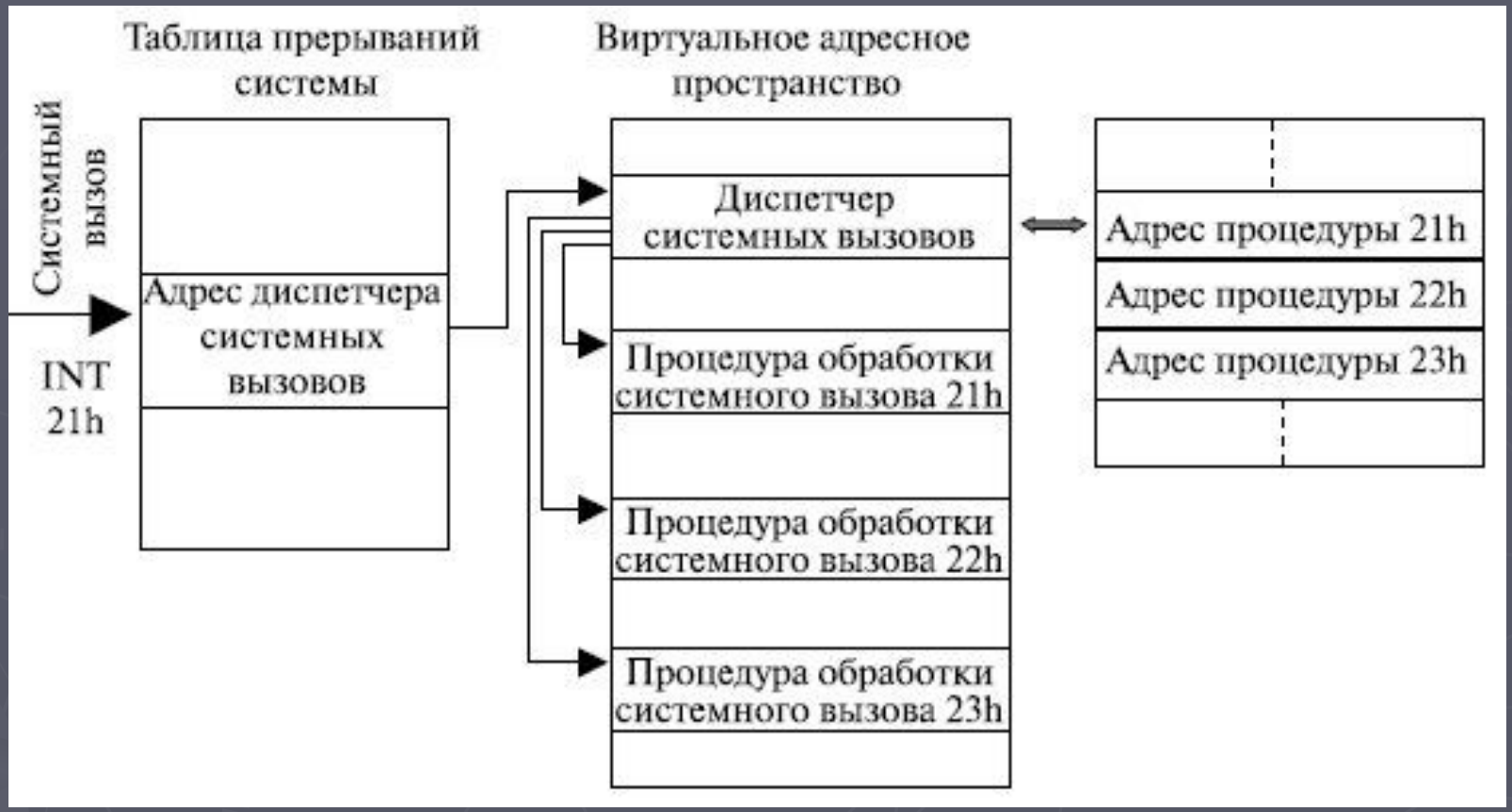
Что произойдет при исполнении этих процессов псевдопараллельно, в режиме разделения времени?

Процессы могут расслоиться на неделимые операции с различным их чередованием, то есть может произойти то, что на английском языке принято называть словом interleaving. Возможные варианты чередования:

a b c d e f
a b d c e f
a b d e c f
a b d e f c
a d b c e f
……
d e f a b c



Граф ресурсов и процессов



Диспетчер системных вызовов

Организация памяти

Память – важнейший ресурс вычислительной системы, требующий эффективного управления.

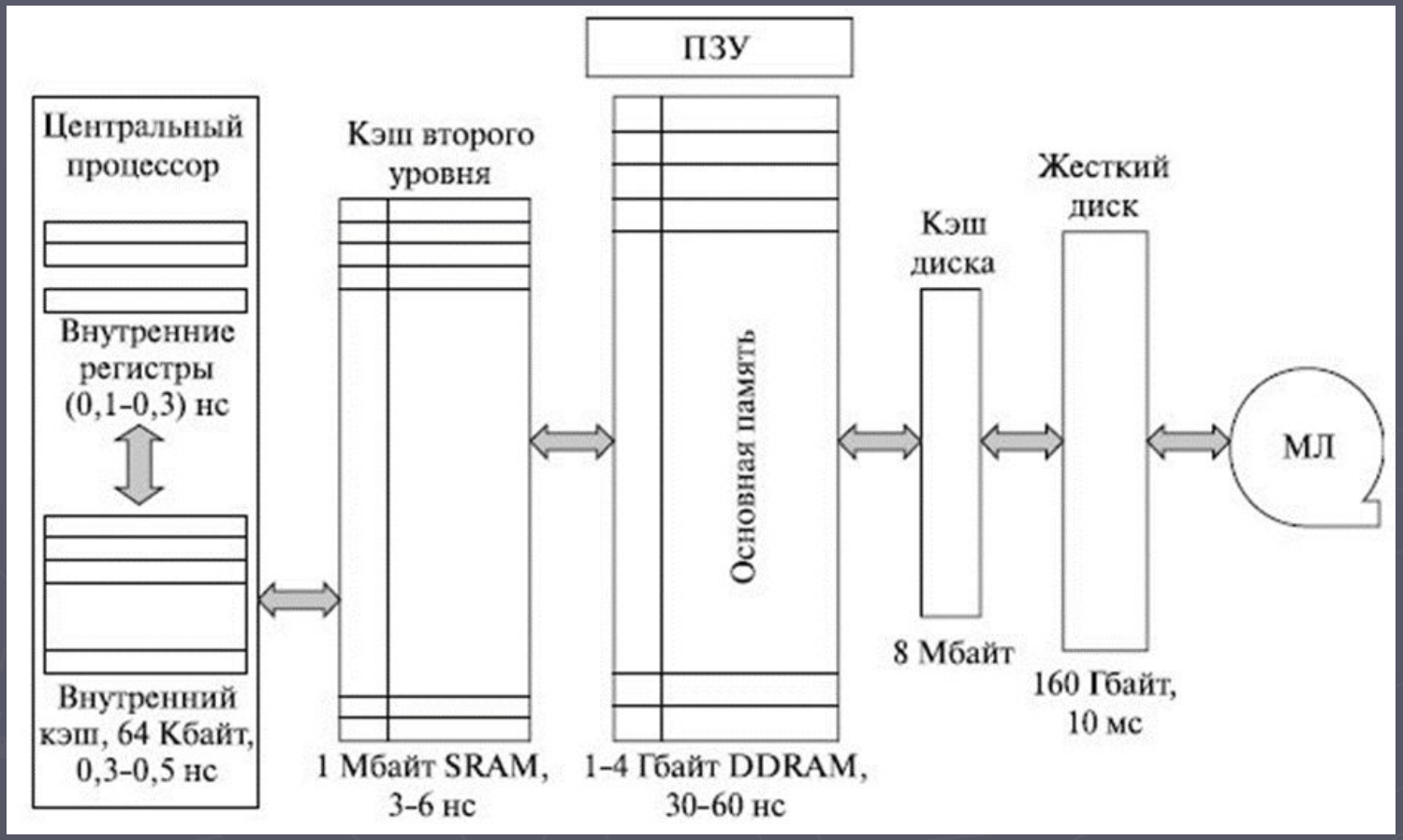
1. Модули могут быть созданы и скомпилированы независимо друг от друга, при этом все ссылки из одного модуля в другой разрешаются системой во время работы программы.

2. Разные модули могут получать разные степени защиты (только чтение, только исполнение) за счет весьма умеренных накладных расходов.

3. Возможно применение механизма, обеспечивающего совместное использование модулей разными процессами (для случая сотрудничества процессов в работе над одной задачей).

- чем меньше время доступа, тем дороже бит;
- чем выше емкость, тем ниже стоимость бита;
- чем выше емкость, тем больше время доступа.

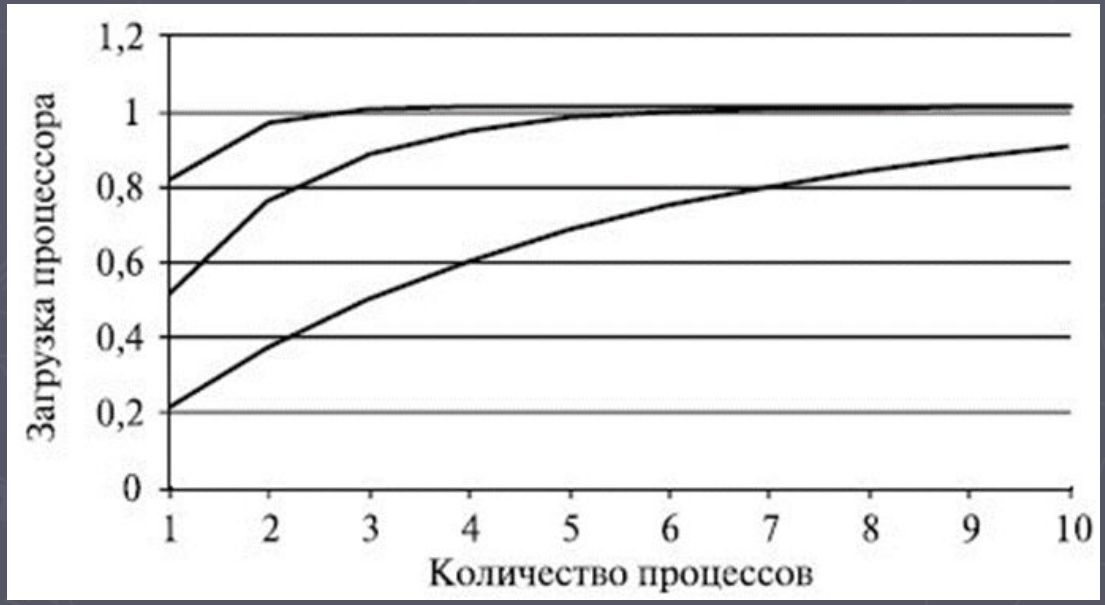
- снижается стоимость бита;
- возрастает емкость;
- возрастает время доступа;
- снижается частота обращений процессора к памяти.



Иерархия памяти

Tср = 0,95*1нс + 0,05* (1нс+10нс)=1,55нс

Z = 1 - pn, где n – число процессов.



Загрузка процессора при различном числе процессов

Виртуализация памяти возможна на основе двух возможных подходов:

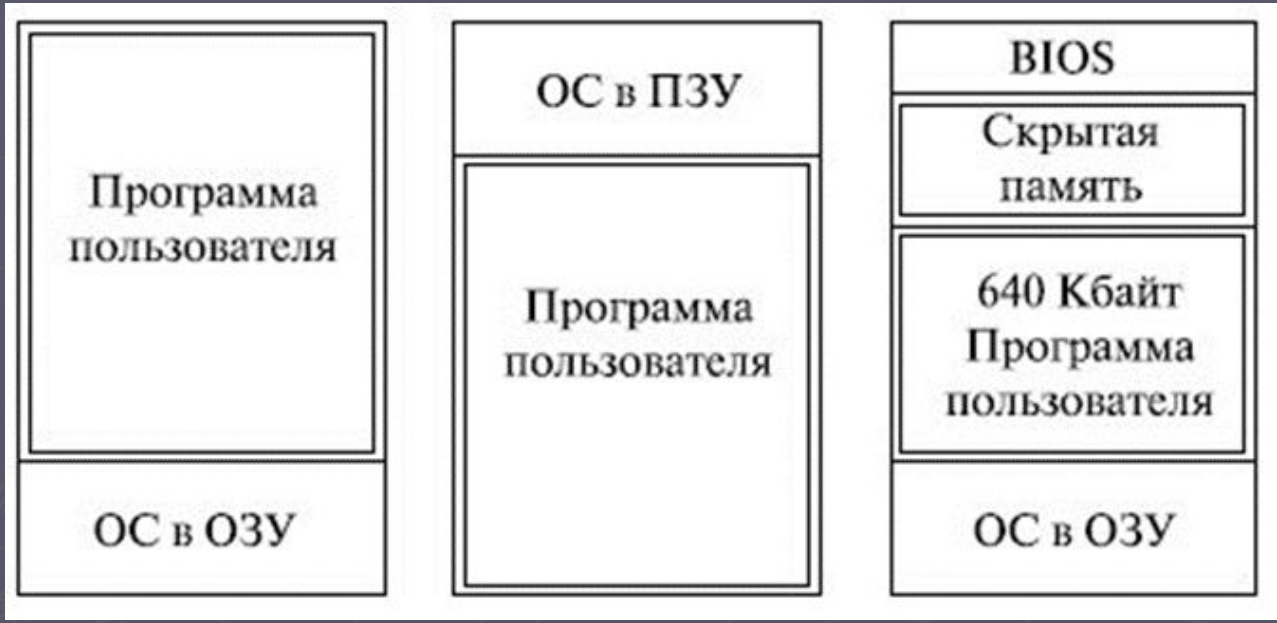
- свопинг (swapping) – образы процессов выгружаются на диск и возвращаются в оперативную память целиком;

- виртуальная память (virtual memory) – между оперативной памятью и диском перемещаются части образов (сегменты, страницы, блоки и т. п.) процессов.

Недостатки свопинга:

- избыточность;

- невозможность загрузить процесс.



Варианты распределения памяти

Функциями ОС по управлению памятью в мультипрограммных системах являются:

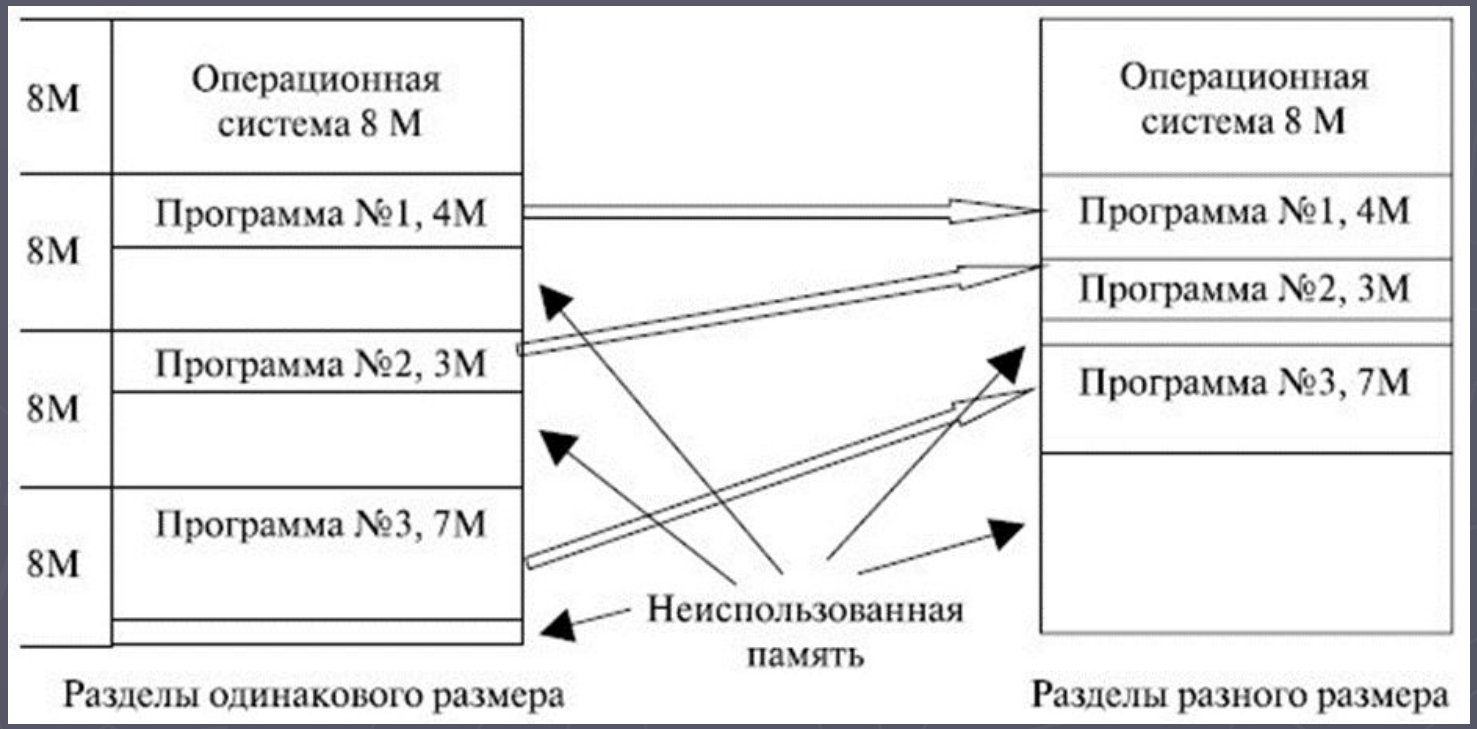
- отслеживание (учет) свободной и занятой памяти;
- первоначальное и динамическое выделение памяти процессам приложений и самой операционной системе и освобождение памяти по завершении процессов;
- настройка адресов программы на конкретную область физической памяти;
- полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП недостаточны для размещения всех процессов, и возвращение их в ОП;
- защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;
- дефрагментация памяти.

Типы адресов

Совокупность виртуальных адресов процесса называется виртуальным адресным пространством.

Диапазон адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для Pentium адресное пространство составляет объем, равный 232 байт, с диапазоном адресов от 0000.000016 до FFFF.FFFF16).

Классификация методов распределения памяти

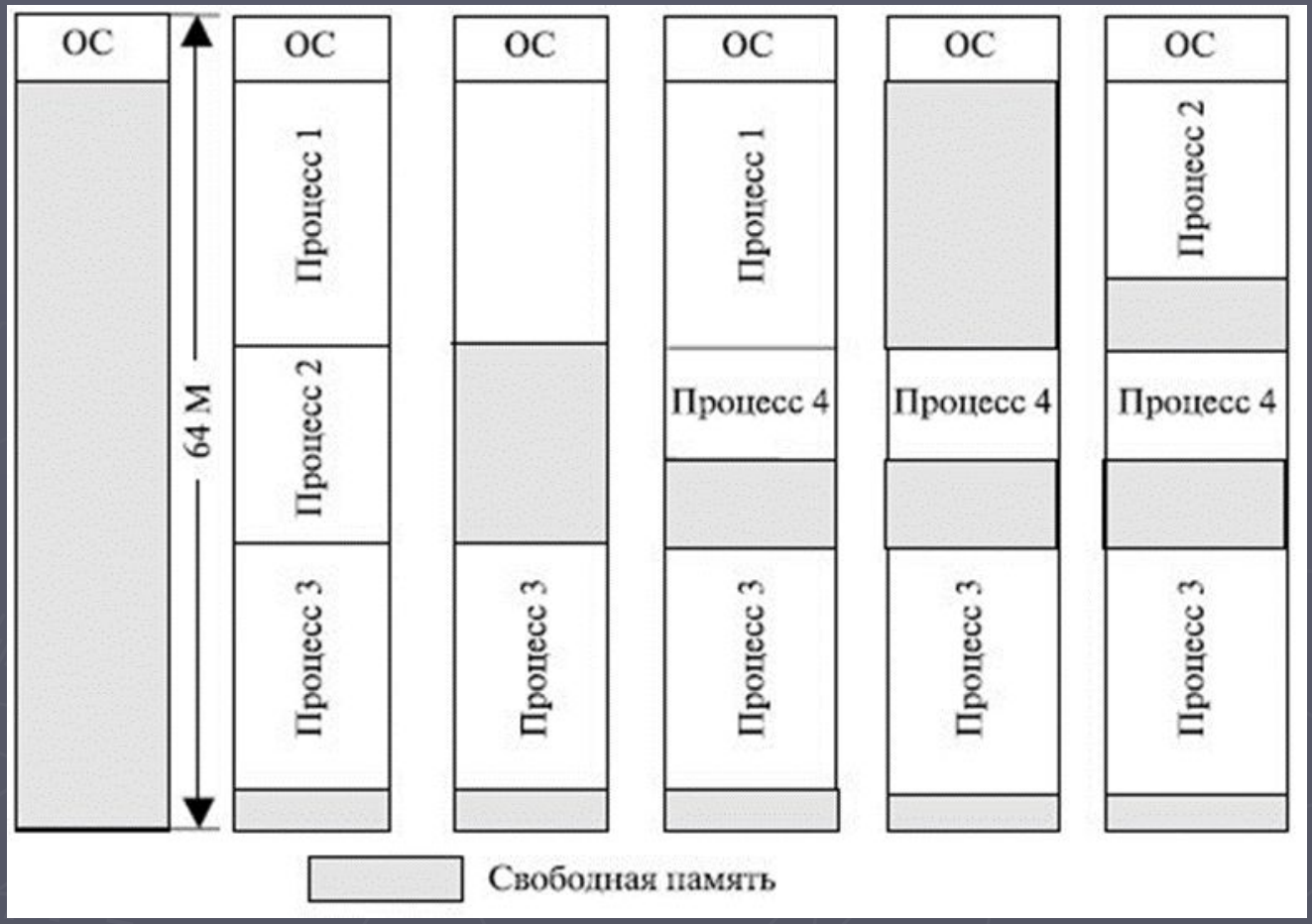


Варианты фиксированного распределения памяти

При использовании разделов с одинаковым размером имеются две проблемы.

Программа может быть слишком велика для размещения в разделе, в данном случае управление памятью во многом возлагается на программиста.

Использование ОП крайне неэффективно. Любая программа, независимо от ее размера, занимает раздел целиком. Этот феномен появления неиспользованной памяти называется внутренней фрагментацией (internal fragmentation).



Вариант использования памяти

Перечислим функции операционной системы по управлению памятью в этом случае.

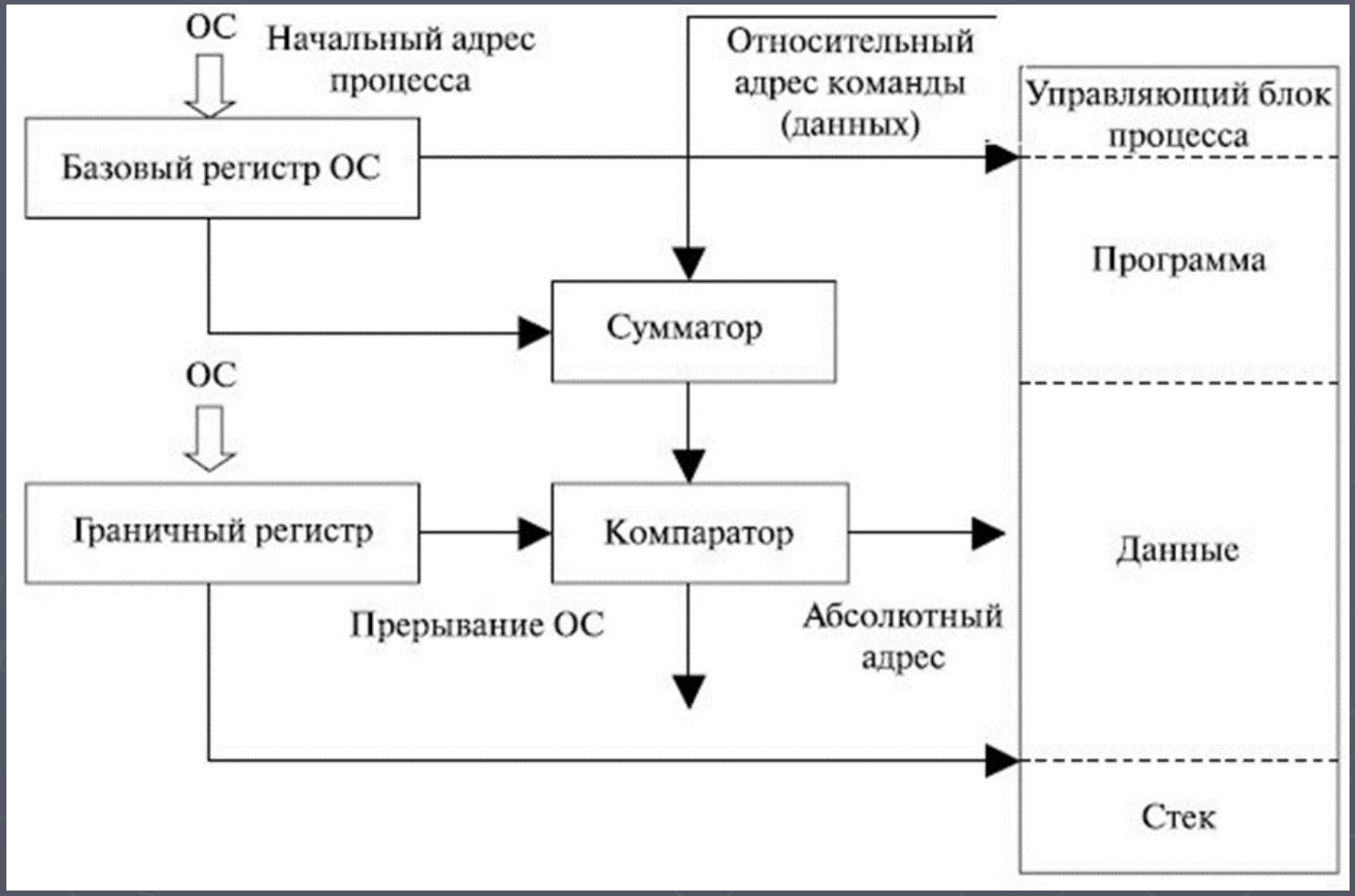
1. Перемещение всех занятых участков в сторону старших или младших адресов при каждом завершении процесса или для вновь создаваемого процесса в случае отсутствия раздела достаточного размера.

2. Коррекция таблиц свободных и занятых областей.

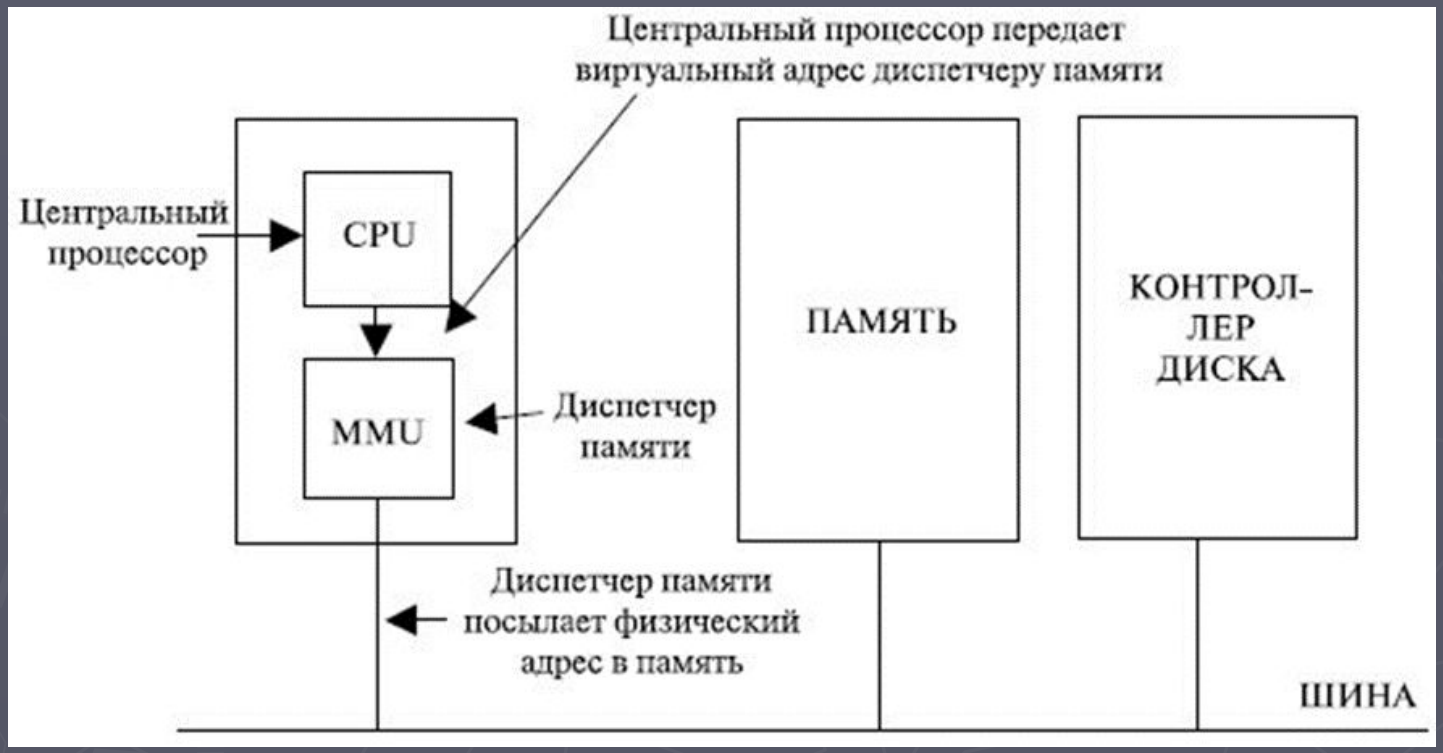
3. Изменение адресов команд и данных, к которым обращаются процессы при их перемещении в памяти, за счет использования относительной адресации.

4. Аппаратная поддержка процесса динамического преобразования относительных адресов в абсолютные адреса основной памяти.

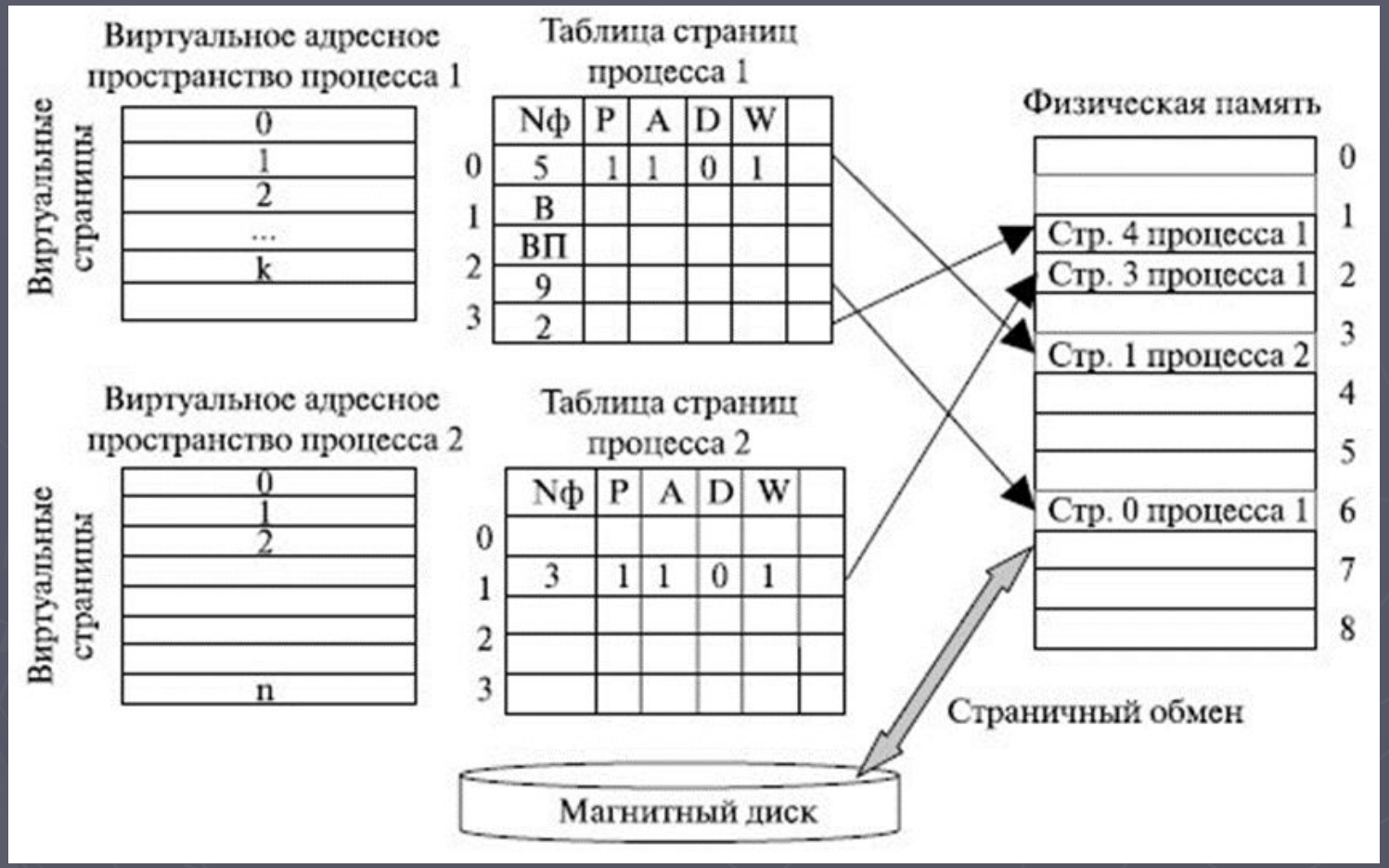
5. Защита памяти, выделяемой процессу, от взаимного влияния других процессов.



Преобразование адресов



Диспетчер памяти



Таблицы страниц виртуальной памяти

Запись таблицы (дескриптор страницы) включает следующую информацию:

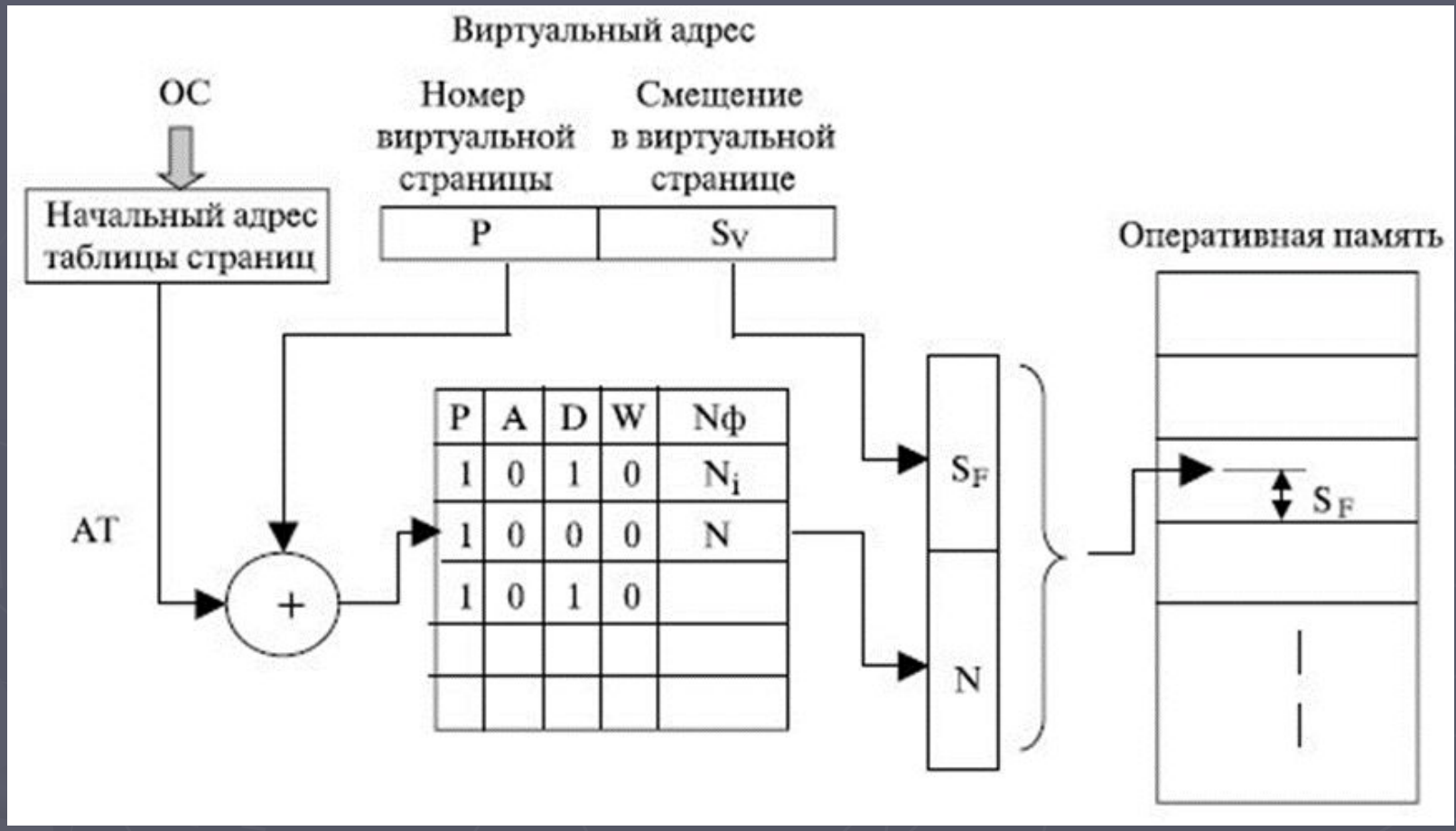
1. номер физической страницы (N ф.с.), в которую загружена данная виртуальная страница;

2. признак присутствия P, устанавливаемый в единицу, если данная страница находится в оперативной памяти;

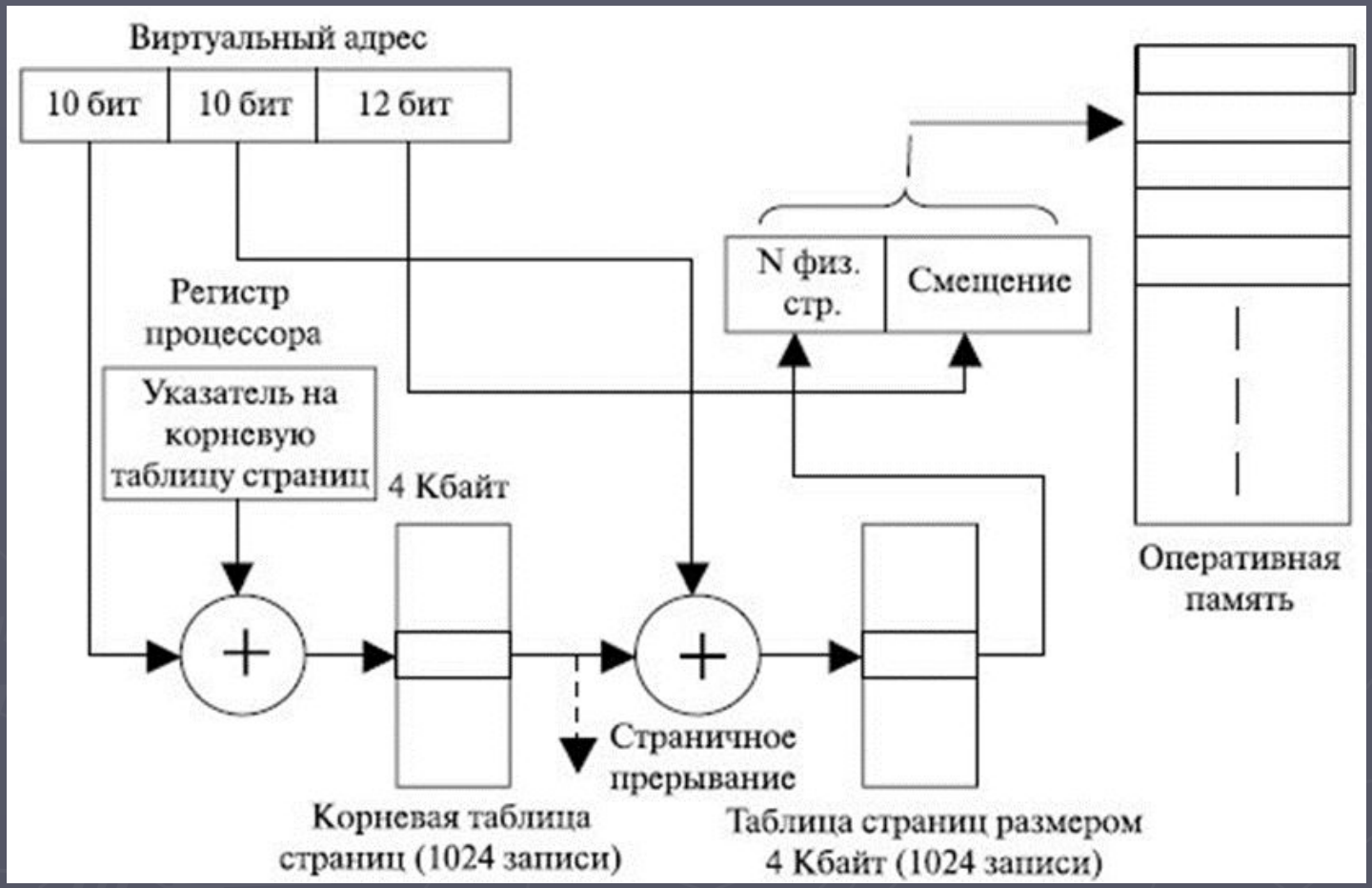
3. признак модификации страницы D, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

4. признак обращения A к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице;

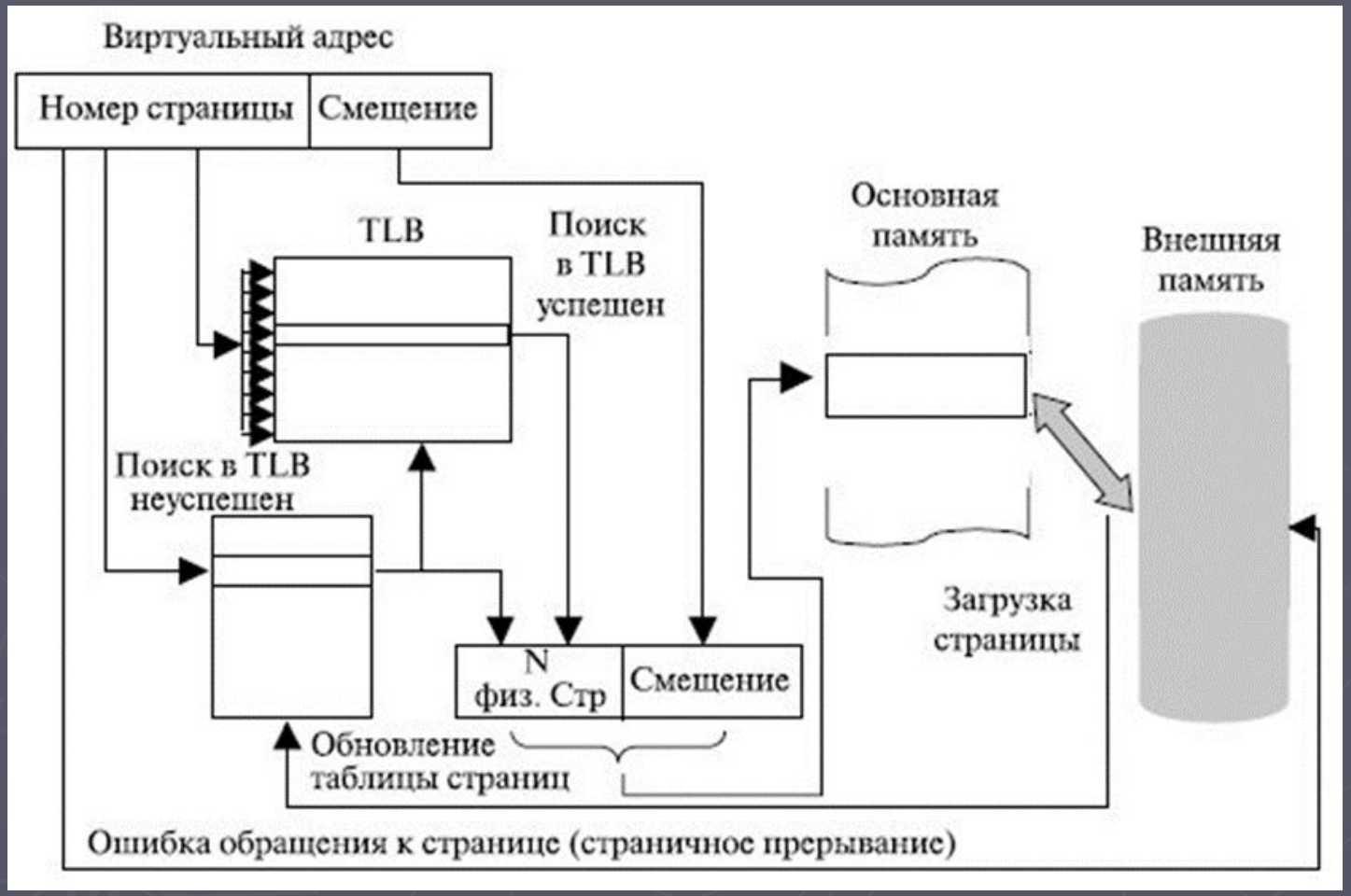
5. другие управляющие биты, служащие, например, для целей защиты или совместного использования памяти на уровне страниц.



Преобразование виртуального адреса

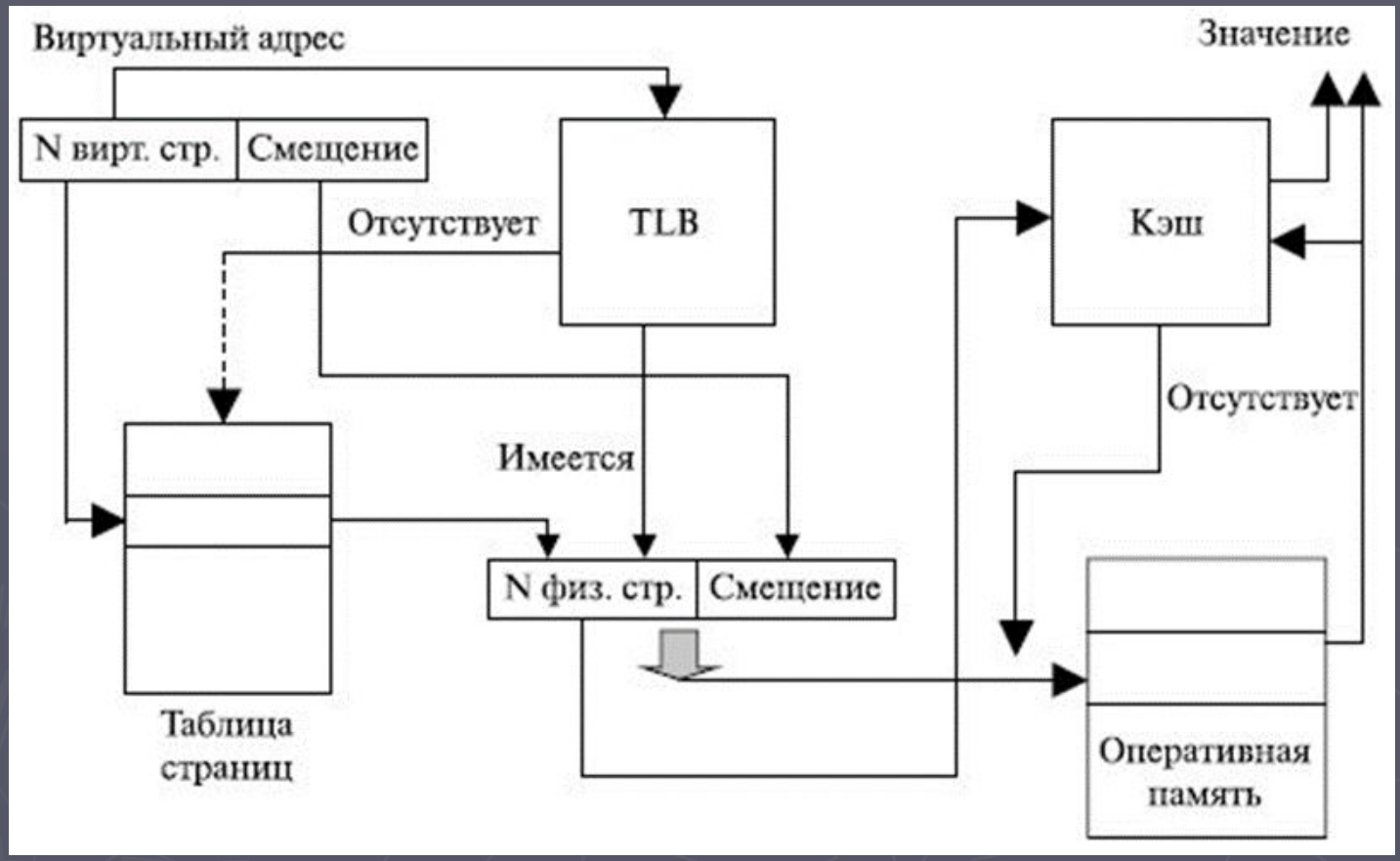


Двухзвенная схема таблиц страниц

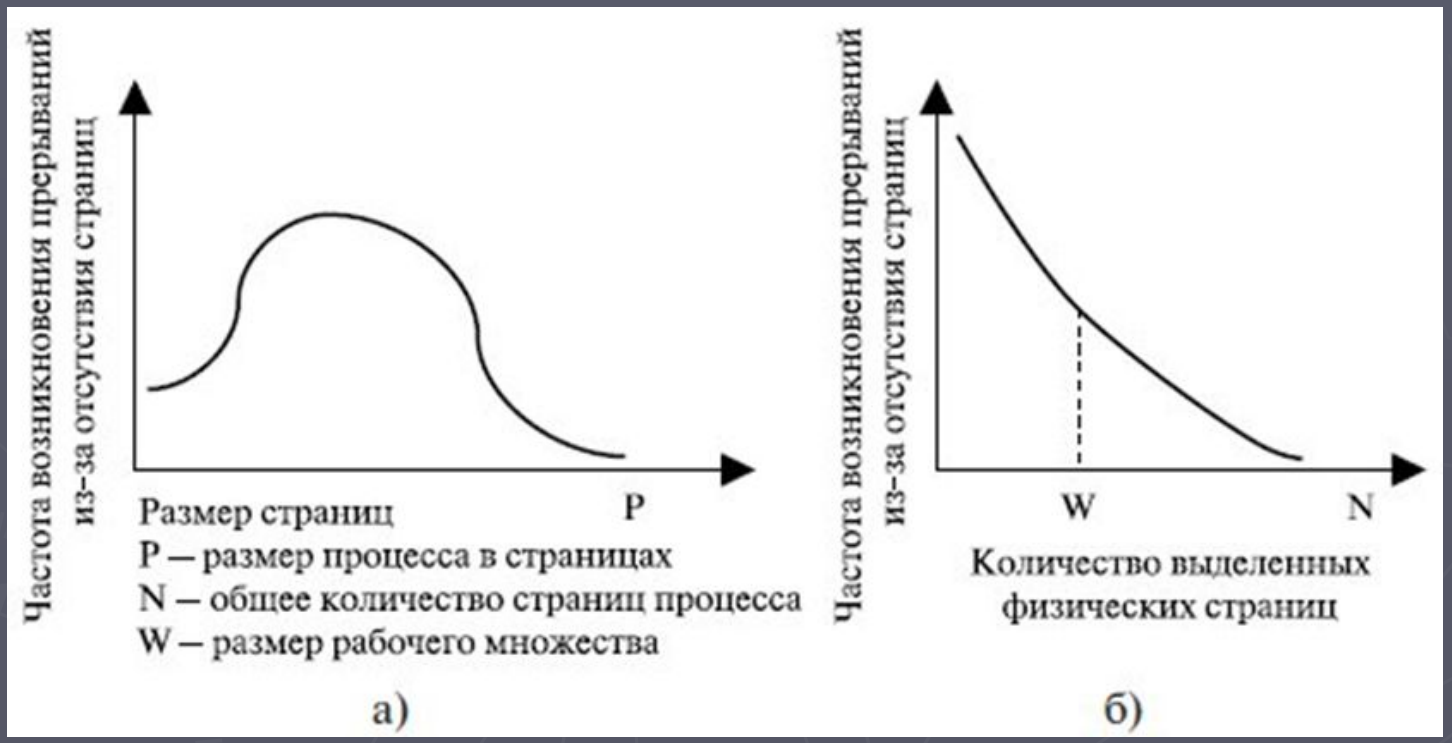


Буфер быстрой переадресации

Ассоциативная память



Использование кэша оперативной памяти

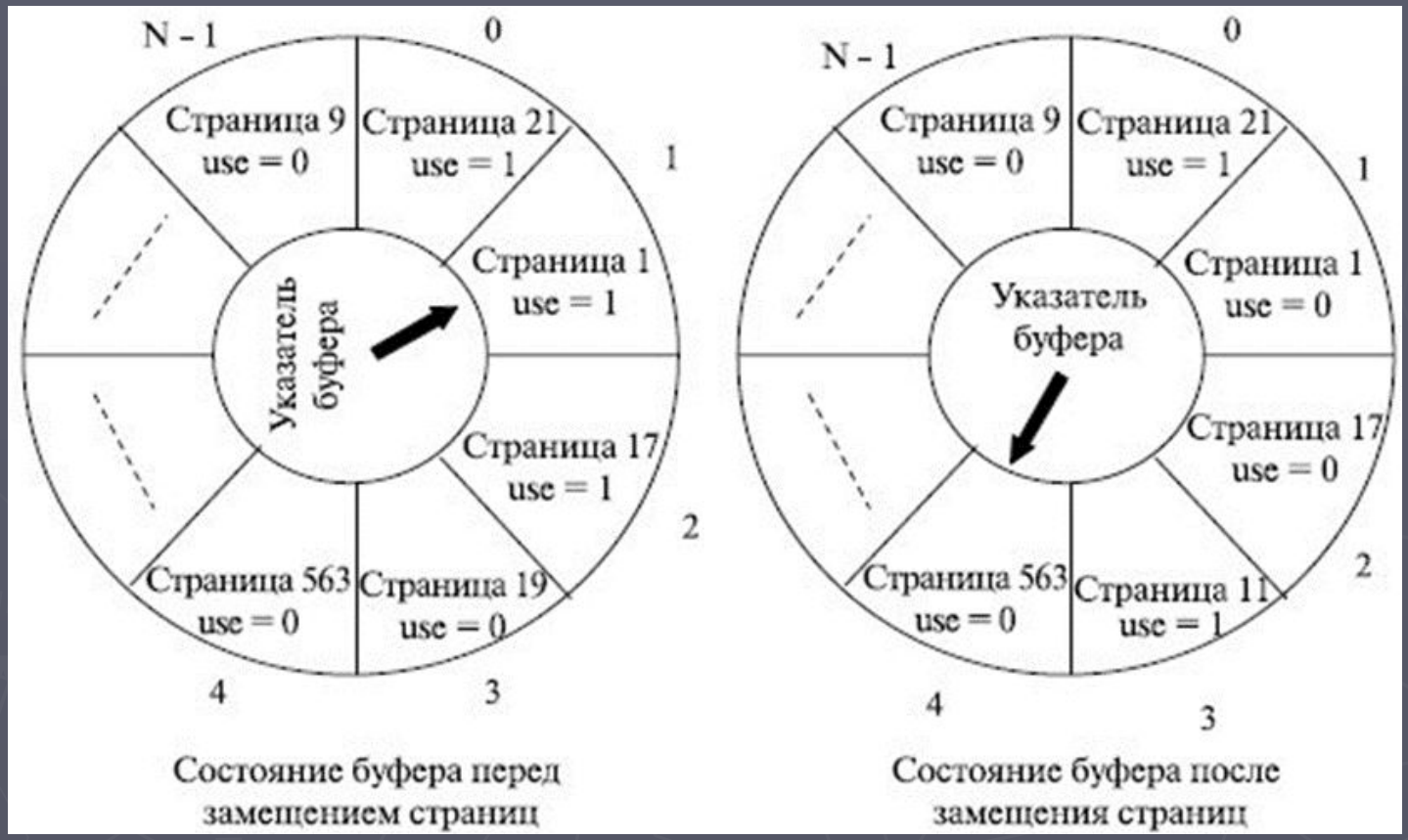


Изменение частоты страничных прерываний

Для управления страничным обменом нужно решить следующие задачи:

- когда передавать страницу в основную память;
- где размещать страницу в физической памяти;
- какую страницу основной памяти выбрать для замедления, если в основной памяти нет свободной физической страницы;
- сколько страниц процесса следует загрузить в основную память;
- когда измененная страница, должна быть записана во вторичную память;
- сколько процессов размещать в основной памяти.

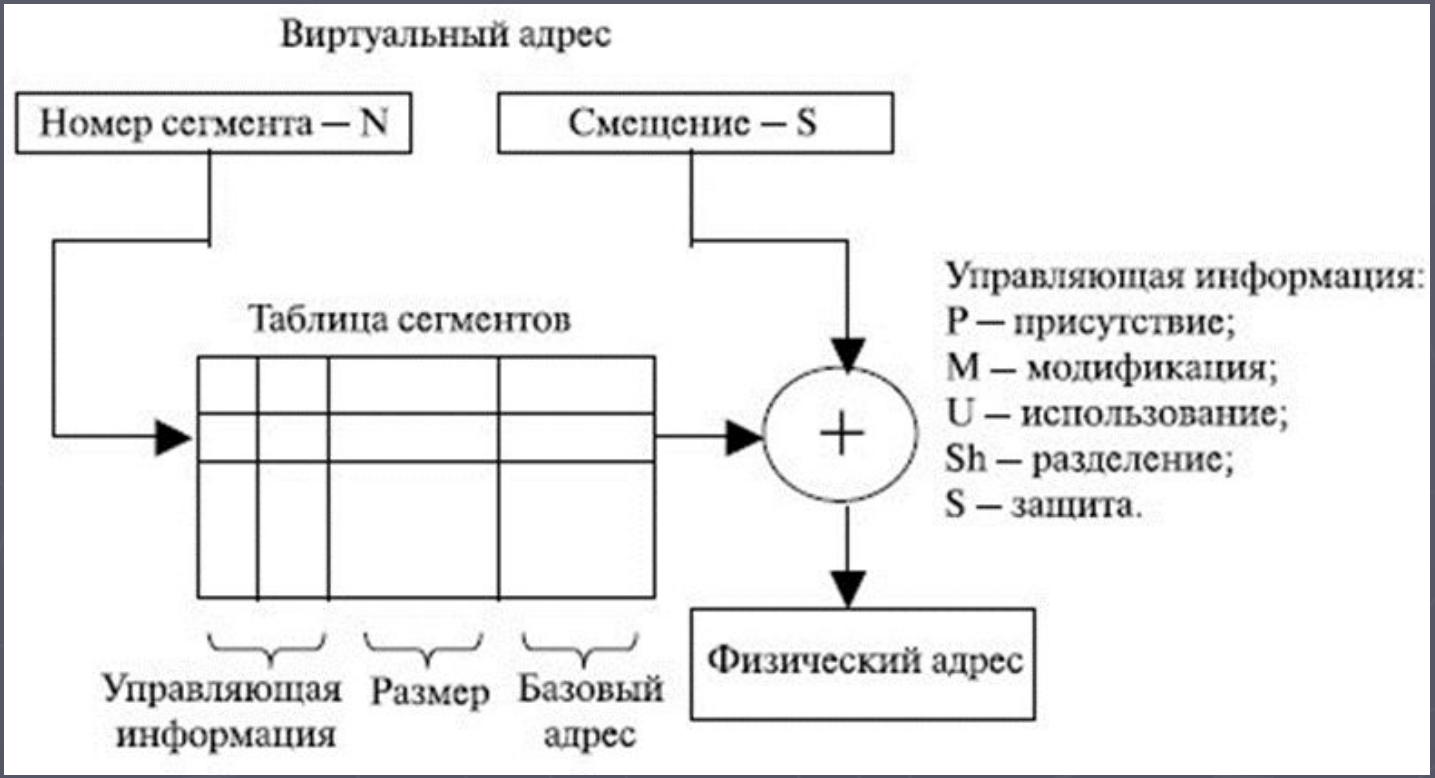
Наименование	Возможные алгоритмы (решения)
Стратегия выборки (когда?)	По требованию, предварительная выборка
Стратегия размещения (где?)	Первый подходящий раздел (для сегментной виртуальной памяти). Любая страница физической памяти (для сегментно-страничной и страничной организации виртуальной памяти)
Стратегия замещения (какие?)	Оптимальный выбор, дольше всех неиспользовавшиеся. Первым вошел – первым вышел (FIFO), часовой, буферизация страниц
Управление резидентным множеством (сколько?)	Фиксированный размер, переменный размер, локальная и глобальная области видимости
Стратегия очистки (когда?)	По требованию, предварительная очистка
Управление загрузкой (сколько?)	Рабочее множество, критерии L=S и 50%



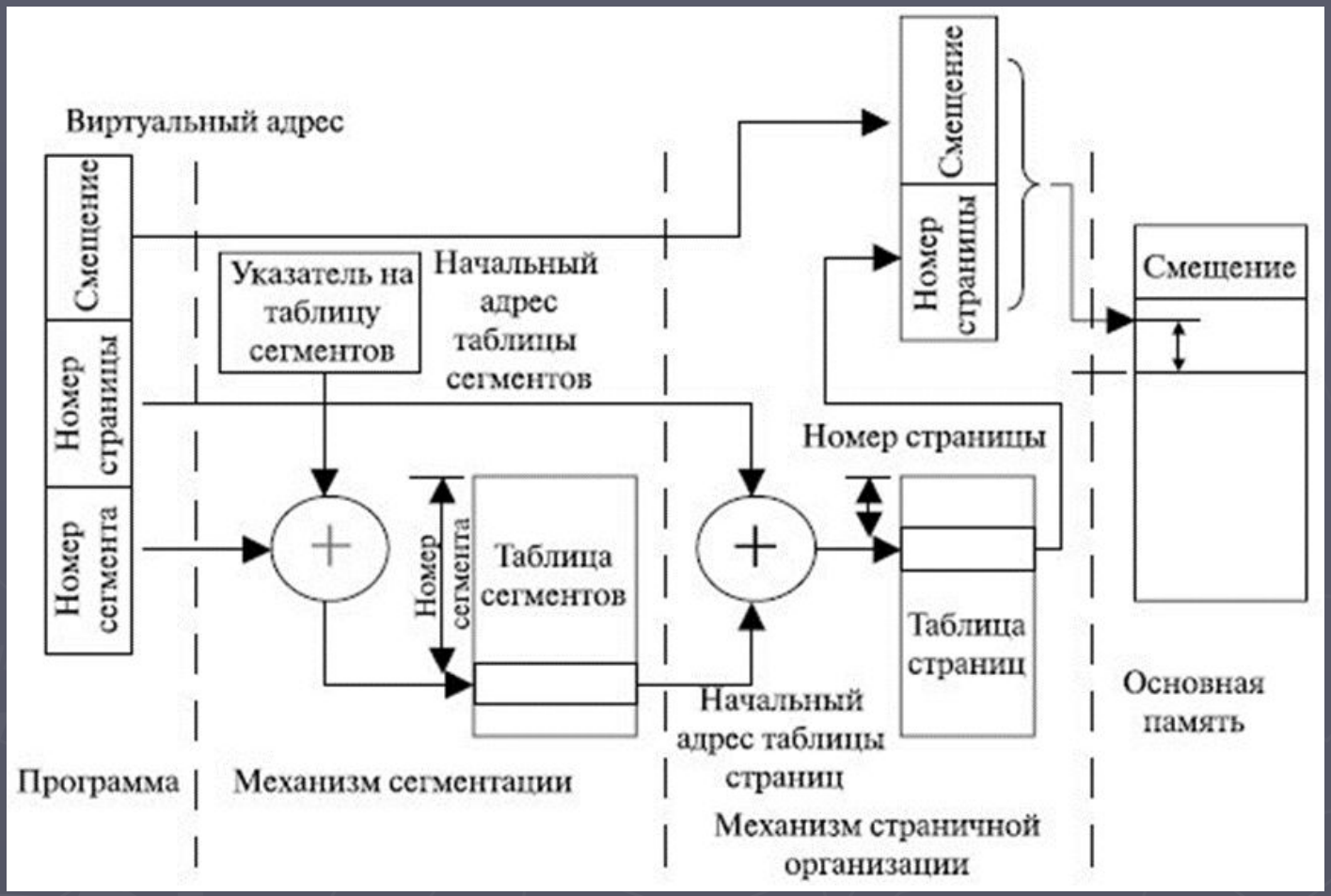
Часовая стратегия замещения

Сложности размещения в одном виртуальном адресном пространстве

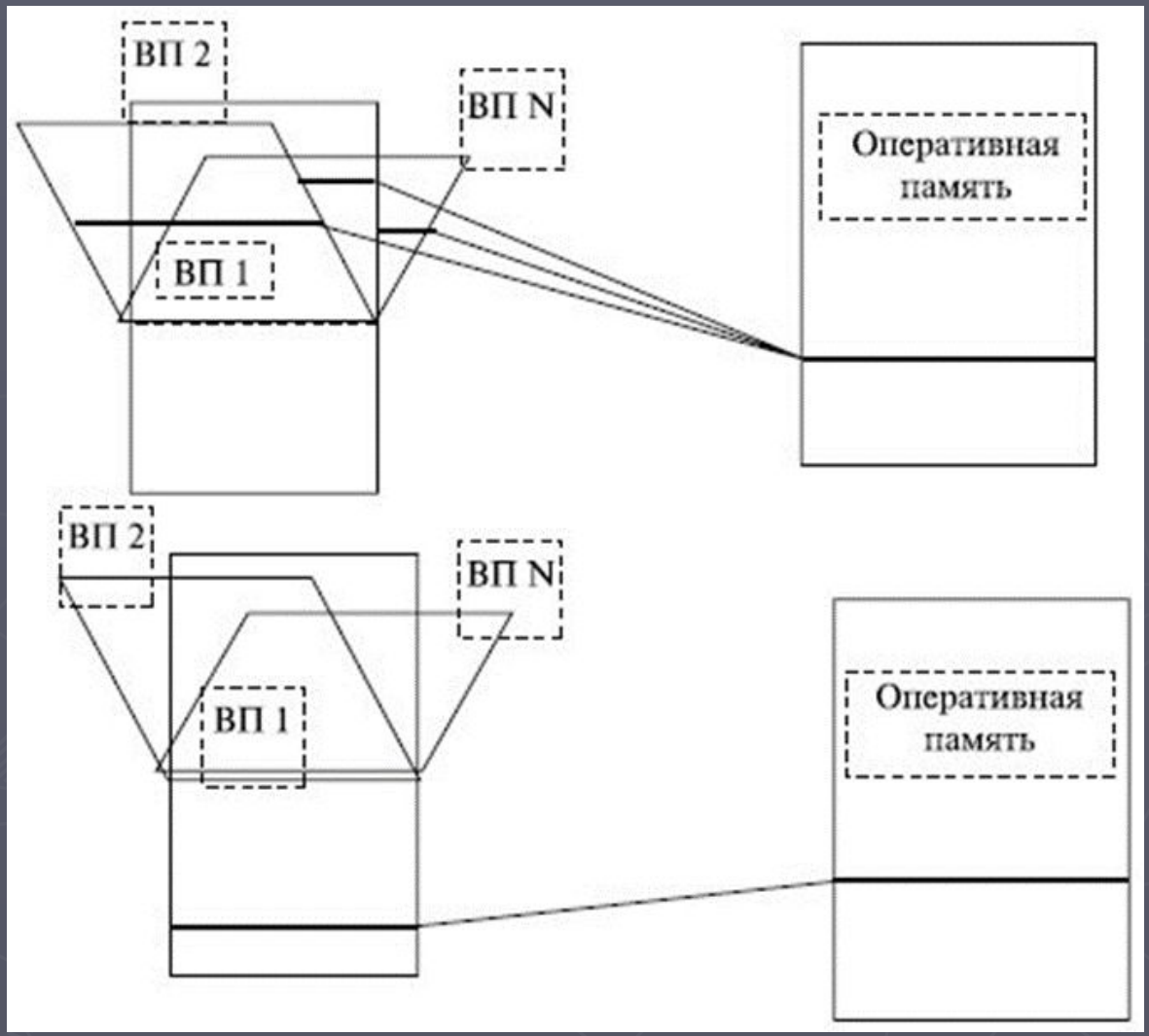
Вопрос	Страничная	Сегментация
Нужно ли программисту знать о том, что используется эта техника?	Нет	Да
Сколько в системе линейных адресных пространств?	Одно	Много
Может ли суммарное адресное пространство превышать размеры физической памяти?	Да	Да
Возможно ли разделение процедур и данных, а также раздельная защита для них?	Нет	Да
Легко ли размещаются таблицы с непостоянными размерами?	Нет	Да
Облегчен ли совместный доступ пользователей к процедурам?	Нет	Да
Зачем была придумана эта техника?	Чтобы получить большое линейное адресное пространство без затрат на физическую память	Для разбиения программ и данных на независимые адресные пространства, облегчения защиты и совместного доступа



Преобразование виртуального адреса



Сегментно-страничная организация памяти



Разделяемые сегменты

Сегментно-страничная организация памяти в Windows

Кольца защиты в Windows

Система защиты манипулирует несколькими переменными, характеризующими уровень привилегий:

DPL (Descriptor Privilege Level) – задается полем DPL в дескрипторе сегмента;

RPL (Requested Privilege Level) – запрашиваемый уровень привилегий,

задается полем RPL селектора сегмента;

CPL (Current Privilege Level) – текущий уровень привилегий выполняемого кода, задается полем RPL селектора кодового сегмента;

EPL (Effective Privilege Level) – эффективный уровень привилегий запроса.


N
NE
E
SE
S
SW
W
NW