П.В. Сташук

КРАТКОЕ ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

2-е издание, стереотипное

Рекомендовано УМО по специальностям педагогического образования в качестве учебного пособия для студентов вузов, обучающихся по специальности 030100 (050202) — «Информатика»

Москва
Издательство «ФЛИНТА»
2014

УДК 004.45(075.8)
ББК 32.973-018.2я73
С78

Рецензенты:

проф., д-р физ.-мат. наук *С.И. Кадченко*
проф., канд. педагог. наук *Г.А. Лисьев*
доцент, канд. техн. наук *В.В. Баранков*

Стащук П.В.

С78 Краткое введение в операционные системы [Электронный ресурс] : учеб. пособие / П.В. Стащук. — 2-е изд., стер. — М. : ФЛИНТА, 2014. —124 с.

ISBN 978-5-9765-0143-0

Применение вычислительной техники не может быть эффективным без знания современного програмного обеспечения, основу которого составляют операционные системы и их оболочки. Изучая предложенный теоретический курс, студенты должны получить представление о возможностях операционных систем, их структуре, принципах организации и функционирования, правилах конфигурирования и т.д. Работа с пособием позволит студентам приобрести знания современных операционных систем на уровне квалифицированного пользователя и поможет в закреплении практических навыков использования современного програмного обеспечения во время учебы и в профессиональной деятельности.

Для студентов, аспирантов, преподавателей вузов.

УДК 004.45(075.8)
ББК 32.973-018.2я73

ISBN 978-5-9765-0143-0

ОГЛАВЛЕНИЕ

ОСНОВНЫЕ ПОНЯТИЯ

Место операционной системы в структуре современной вычислительной системы

В широком смысле современная **вычислительная система** (ВС) представляет собой комплекс технических и программных средств, предназначенный для автоматизации решения информационных задач пользователя. Рассмотрим упрощенную структуру этой системы (способ объединения компонентов) с учетом иерархической декомпозиции.

На верхнем уровне иерархии (рис. 1) вычислительную систему определяют указанные выше компоненты.

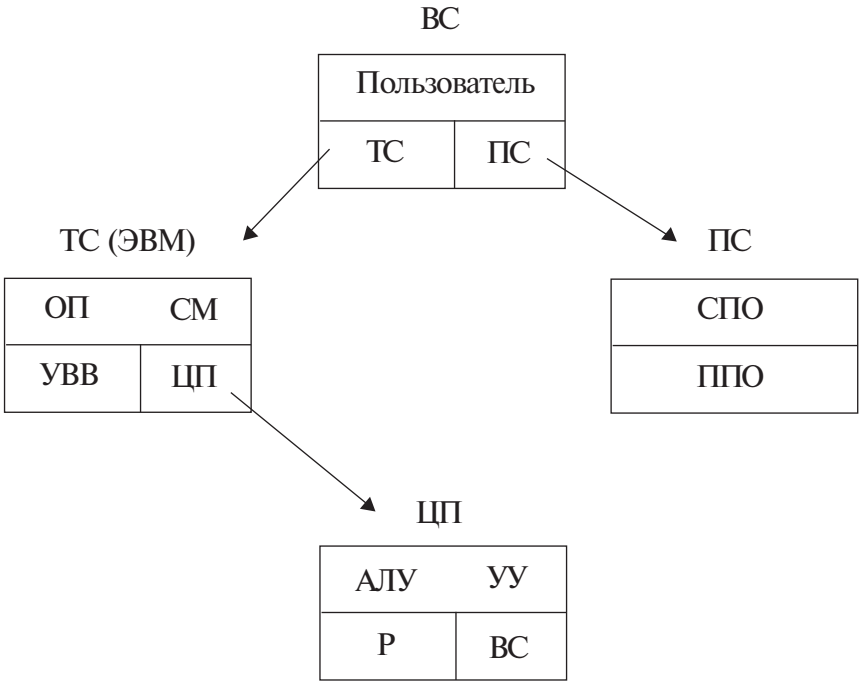

Рис. 1. Граф иерархии компонентов ВС

В качестве **пользователя** выступают люди, в чьих интересах осуществляется обработка данных ВС (выполнение сложных расчетов, управление производственными процессами и т.д.). Основным **техническим средством** (ТС) — компьютеры (ЭВМ), способные эффективно обрабатывать любые виды информации (числовую, текстовую, табличную, графическую, видео, звуковую), предварительно преобразованной в цифровую форму. **Программные средства** (ПС) представляют собой наборы цифровых кодов (инструкций), управляющих техническими средствами в зависимости от алгоритмов решения конкретных информационных задач пользователя.

На следующем уровне представлена структура двух компонентов верхнего уровня — технических и программных средств.

Любой компьютер состоит из четырех основных компонентов:

- **Центральный процессор** (ЦП) управляет другими компонентами и выполняет функции обработки информации**.**
- **Оперативная память** (ОП) временно хранит обрабатываемую информацию, программные коды и результаты обработки.
- **Устройства ввода/вывода** (внешние устройства, УВВ) перемещают информацию от/к пользователю.
- **Системная магистраль** определяет механизм взаимодействия указанных выше компонентов (ЦП, ОП, УВВ).

Конкретные компьютеры могут объединять один или более компонентов каждого типа.

Перейдем к третьему уровню иерархии ВС.

В частности, в структуре ЦП можно выделить:

- **Арифметико-логическое устройство** (АЛУ) выполняет операции по содержательной обработке информации пользователя.
- **Устройство управления** (УУ) контролирует как ЦП, так и другие компоненты компьютера.
- **Регистры** (Р) хранят оперативную информацию во время выполнения процессором текущей операции.
- **Внутренние связи** определяют механизм взаимодействия указанных выше компонентов (АЛУ, УУ, Р).

Убедившись в сложности технических средств современной ВС, рассмотрим программный ее компонент.

Программные средства компьютеров делятся на две категории:

- **Системное программное обеспечение** (СПО) — программы и пакеты программ (BIOS, операционные системы (ОС), диалоговые оболочки ОС, драйверы внешних устройств, утилиты, трансляторы языков программирования, системы компьютерных коммуникаций...), предназначенные для непосредственного управления процессами взаимодействия физических устройств компьютера.
- **Прикладное программное обеспечение** (ППО) — пакеты программ, обеспечивающие обработку данных при решении задач пользователя. К ним относятся диалоговые инструментальные (операционные) оболочки языков программирования и средств межкомпьютерной связи, узко специальные программы (информационные модели, обучающие, тестовые, бухгалтерские, банковские...), созданные для конкретных групп пользователей, и универсальные программы для широкого круга пользователей (текстовые и табличные процессоры, графические редакторы, информационно-справочные системы, СУБД и интегрированные пакеты ПП).

ППО базируется (использует при работе) на СПО.

В категории СПО главную роль играют операционные системы. Именно эти программные комплексы в наибольшей степени определяет облик всей ВС в целом. Они являются промежуточным звеном между пользователем (с его ППО) и компьютером и решают две, на первый взгляд, мало связанные задачи:

~ обеспечение пользовательского интерфейса — предоставление ему виртуального компьютера,

~ повышение эффективности использования компьютера за счет рационального управления его физическими компонентами.

Назначение, функции и определение операционной системы

Пользователь обычно рассматривает ВС в терминах прикладной задачи (т.е. ППО). Если для ее решения использовать язык программирования низкого уровня (приближенный к машинным кодам), то ему придется решать дополнительные задачи по управлению аппаратными средствами (компонентами) компьютера.

ОС ограждает пользователя от непосредственной работы с аппаратурой компьютера и предоставляет ему простой интерфейс, самостоятельно решая низкоуровневые проблемы по управлению аппаратным обеспечением (АО). С этой точки зрения назначением ОС является предоставление пользователю некоторого «расширенного» или ***виртуального компьютера*** (среда DOS, UNIX, Windows...), с которым легче работать, чем непосредственно с физическими устройствами, составляющими реальный компьютер.

Типовые процедуры и служебные функции ОС в этом контексте следующие:

- Обеспечение выполнения прикладных программ.
- Доступ к средствам разработки прикладных программ.
- Управление вводом/выводом информации (файловая система и внешние устройства).
- Защита информации от несанкционированного доступа.
- Предотвращение, обнаружение и исправление ошибок (защита от сбоев).
- Мониторинг эффективности и производительности.

С другой стороны, компьютер представляет собой набор физических компонентов, осуществляющих манипуляции с информацией. Аппаратными ресурсами современных компьютеров являются процессоры, память, коммуникационная аппаратура, внешние устройства и т.д.

ОС, как менеджер ресурсов, осуществляет распределение указанных устройств и данных между процессами (выполняющимися программами), конкурирующими за эти ресурсы. Она управляет всеми ресурсами вычислительной системы таким об-

разом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами предполагает решение следующих проблем:

- Планирование ресурсов — то есть определение, какому процессу, когда, а для делимых ресурсов — и в каком количестве необходимо выделить данный ресурс;
- Контроль состояния ресурсов — то есть поддержание оперативной информации о том, занят или свободен ресурс, а для делимых ресурсов — какое количество ресурса уже распределено, а какое свободно.
- Удовлетворение текущих запросов и разрешение конфликтов.

Для решения указанных задач управления ресурсами разные ОС используют различные алгоритмы, что определяет их различия, включая характеристики производительности, область применения и даже пользовательский интерфейс.

Дадим следующее определение ОС.

Операционная система — набор программ, обеспечивающих управление данными и выполнение программ пользователей, координирующих распределение компьютерных ресурсов и поддерживающих взаимодействие с пользователями.

Эволюция ОС

Большое влияние на развитие ОС оказало постепенное совершенствование элементной базы, вычислительной архитектуры и информационных технологий. Рассмотрим основные периоды истории ОС.

Первый период (1945–1949, компонентная база — электронные лампы)

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 в рамках конкретного проекта. Про-

граммирование осуществлялось исключительно на машинном языке. Управление вычислениями производилось со специального пульта. Результаты расчетов выводились на принтер.

ОС отсутствуют.

Второй период (1950–1965, полупроводники)

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, и спектр решаемых ими практических задач существенно расширился. Произошло разделение обслуживающего персонала на программистов и операторов, эксплуатационников и разработчиков вычислительных машин. Стоимость процессорного времени постоянно возрастала, что потребовало уменьшения его непроизводительных затрат между запусками программ. Стала актуальной автоматизация программирования и организации вычислений.

Появились первые ОС — системы пакетной обработки (резидентные мониторы), которые автоматизировали чтение из внешней памяти и запуск одной программы по окончании выполнения предыдущей, чем увеличивали коэффициент загрузки процессора. Запущенная на выполнение программа монопольно использовала все ресурсы компьютера.

В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями (JCL), с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на компьютере. Совокупность нескольких заданий, как правило в виде колоды перфокарт, получила название пакета заданий (batch).

К началу 60-х появились ОС коллективного пользования с **мультипрограммированием** — организацией вычислительного процесса с попеременным выполнением нескольких программ одновременно. Каждая программа загружается в отдельный участок оперативной памяти, а процессор быстро переключается между программами.

Наряду с мультипрограммной реализацией режима пакетной обработки появились новые типы ОС. Системы разделения времени, в которых вариант мультипрограммирования обеспечива-

ет каждому отдельному пользователю, использующему персональный терминал, интерактивный (диалоговый) режим выполнения программ, создающий иллюзию единоличного использования вычислительной машины. Системы реального времени для управления сложными технологическими процессами.

Третий период (1965–1980, интегральные микросхемы)

Для этого периода характерно создание первого семейства универсальных программно-совместимых машин общего назначения (мэйнфреймов, IBM/360) для решения любых задач из различных областей приложения. Соответствующие ОС (OS/360, MULTICS) должны были работать на различных компьютерах, с различным количеством разнообразной «периферии», в различных прикладных областях.

Важнейшим достижением ОС данного поколения явились мобильность, многорежимность и поддержка многопроцессорности.

Четвертый период (1980 — настоящее время, БИС)

Мини- и персональные компьютеры, имеющие существенно упрощенную архитектуру, стали широко использоваться неспециалистами, что потребовало разработки «дружественного» программного обеспечения.

На рынке ОС доминировали: UNIX и MS-DOS. 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 «неинтеловских» мини-компьютеров (в частности, на базе высокопроизводительных RISC-процессоров).

В середине 80-х стали бурно развиваться сети ПК, работающие под управлением сетевых или распределенных ОС. Их пользователи должны быть своевременно осведомлены о наличии других компьютеров в сети и осуществлять логический вход в эти компьютеры для использования их ресурсов (преимущественно файлов). Каждая машина (хост) в сети выполняет свою собственную локальную ОС, отличающуюся от ОС автономного компьютера наличием дополнительных средств, позволяющих компьютеру работать в сети.

Сетевая ОС содержит программную поддержку для сетевых интерфейсных устройств (драйвер сетевого адаптера), средств для удаленного «входа» в другие компьютеры сети, средств доступа к удаленным файлам и средств предоставления локальных ресурсов в общее пользование.

К концу 90-х в ОС обязательна поддержка средств: графического пользовательского интерфейса, работы с Интернетом, а также сетевой безопасности, удаленного администрирования и других функций, обеспечивающих корпоративность. Желательны: мобильность и поддержка многопроцессорности.

В перспективе для пользователя — полная прозрачность ОС, включая полностью автоматические локальную установку и конфигурирование, а также организацию сетевых ресурсов для осуществления распределенных вычислений.

Классификация ОС

ОС различаются особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, внешними устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями применения и другими свойствами.

Рассмотрим классификацию ОС по нескольким основным признакам.

Особенности алгоритмов управления ресурсами

В зависимости от особенностей использованного алгоритма управления процессором, ОС делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многонитевую обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

1. Поддержка многозадачности.

По числу одновременно выполняемых задач (программ) ОС могут быть разделены на два класса: однозадачные (например,

MS-DOS, MSX) и многозадачные (ОС ЕС, OS/2, Unix, Windows 95...).

Однозадачные ОС в основном выполняют функцию предоставления пользователю одной виртуальной машины, упрощая процесс взаимодействия пользователя с АО. Однозадачные ОС включают средства управления периферийными устройствами, средства управления файлами, средства общения с пользователем.

Многозадачные ОС предоставляют пользователю несколько (по числу приложений) виртуальных компьютеров, поэтому, кроме вышеперечисленных функций, управляют разделением совместно используемых задачами ресурсов, таких как ЦП, ОП, файлы и внешние устройства.

Важным свойством многозадачных ОС является возможность распараллеливания вычислений в рамках одной задачи, то есть многонитевость (многопоточность). Многонитевая ОС разделяет процессорное время между отдельными подзадачами (нитями) каждой задачи.

2. Поддержка многопользовательского режима.

По числу одновременно работающих пользователей ОС делятся на однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2) и многопользовательские (Unix, Windows NT).

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

Не всякая многозадачная система является многопользовательской, как и не всякая однопользовательская ОС является однозадачной.

3. Многопроцессорная обработка.

Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки — мультипроцессирование.

Функции поддержки многопроцессорной обработки данных имеются, в частности, у Windows NT фирмы Microsoft и NetWare 4.1 фирмы Novell.

Многопроцессорные ОС подразделяются, в свою очередь, на **асимметричные** (ОС целиком выполняется только на одном из

процессоров системы, распределяя прикладные задачи по остальным процессорам) и **симметричные** (ОС полностью децентрализована и использует весь пул процессоров, разделяя их одновременно между системными и прикладными задачами).

Важное влияние на облик ОС в целом оказывают также особенности подсистем управления другими ресурсами компьютера — памятью, файлами, устройствами ввода-вывода.

Особенности аппаратных платформ

На свойства ОС непосредственное влияние оказывает архитектура аппаратных средств, на которые она ориентирована. По типу АО различают ОС ПК, **мейнфреймов** (большая ЭВМ), кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. Уточним специфику аппаратных средств и их использования.

Для **персональных компьютеров** характерно наличие аппаратных ресурсов в единственном количестве (рис. 2а). Применяет ПК широкий круг пользователей с низким уровнем подготовки в области информатики. Поэтому ОС ПК управляют сравнительно малым количеством ресурсов и предоставляют наиболее дружественный пользовательский интерфейс (графический).

Большие машины (Intel Paragon, IBM SP) в настоящее время разрабатывают на базе концепции массового параллелизма (MPP, massively parallel processing), то есть с использованием наборов серийных микропроцессоров, объединенных высокоскоростной коммуникационной средой (многомерные каналы связи процессоров посредством иерархической системы переключателей) и использующих общую ОП (рис. 2б). Такие компьютеры оснащаются наборами разнообразных периферийных устройств. Их ОС значительно сложнее ОС ПК и имеют специфическую функциональность в плане поддержки симметричного комплексирования многих процессоров (SMP, symmetric muli processing), управления наборами внешних устройств и обслуживания пользователей различного уровня подготовки (множественный интерфейс пользователя).

Другим вариантом реализации массового параллелизма являются кластеры.

Кластер представляет собой совокупность нескольких серийных ПК (узлов), объединенных в локальную сеть для совместного выполнения общего приложения, представляющуюся пользователю единой системой (рис. 2в). Состав и мощность узлов могут меняться в рамках одного кластера. Для объединения узлов используют обычные (Fast и Gigabits Ethernet) и специальные коммуникационные технологии (несколько сетевых карт на узел, многомерная топология кабельных связей — SCI или Myrinet — связи через коммутатор...). Внешние устройства (терминал) подключаются только к одному из узлов кластера. В кластере каждый узел работает под управлением стандартной ОС ПК. Однако для выполнения параллельных вычислений необходима дополнительная программная поддержка со стороны ОС (драйверы коммуникационной среды, средства межпроцессного взаимодействия, администрирования и конфигурирования кластера).

ОС для сетевого компьютера имеет в своем составе средства передачи сообщений по линиям связи между компьютерами различных аппаратных платформ (рис. 2г). На основе этих со-

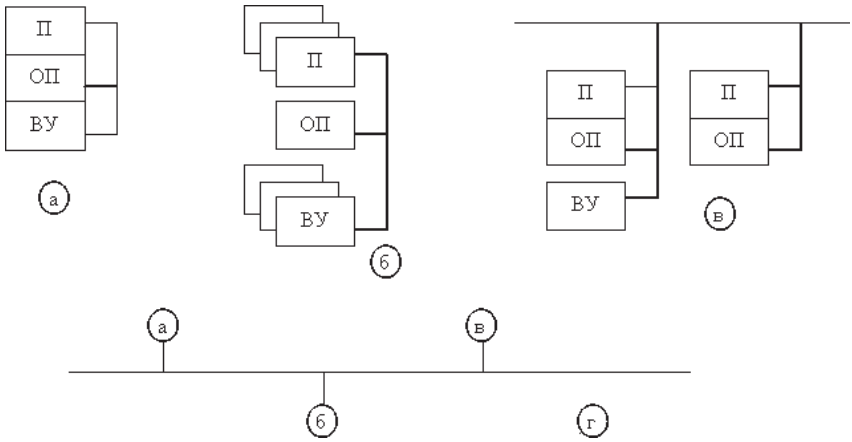

Рис. 2. Схема аппаратных ресурсов различных компьютерных систем.

общений она 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.

Специфика сетевой ОС проявляется также в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передачу 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 и др.

Наряду с ОС, ориентированными на определенный тип аппаратной платформы, существуют универсальные или мобильные ОС, сравнительно легко переносимые с компьютера одного типа на компьютер другого типа. В этих системах (UNIX) аппаратно-зависимые модули локализованы и модифицируются при переносе системы на новую платформу (что облегчается написанием кода на аппаратно-независимом языке, например, на Си).

Режимы применения

Многозадачные ОС в соответствии с использованными при их разработке критериями эффективности подразделяются на три типа: системы пакетной обработки (ранние ОС для компьютеров IBM и ЭВМ ЕС), системы разделения времени (UNIX, VMS), системы реального времени (QNX, RT/11).

Системы пакетной обработки (СПаО) предназначались для решения задач вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, то есть решение наибольшего числа задач в единицу времени (при минимальном участии

пользователя в ходе решения). Для этого в начале работы формируется пакет заданий, где каждое задание содержит требование к системным ресурсам. Из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач, у которых требования к ресурсам сбалансированы (желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом). Выбор выполняемого задания из пакета зависит от текущей внутренней ситуации в системе (подбирается «выгодное» задание).

В СПаО невозможно гарантировать выполнение конкретного задания в течение определенного периода времени. Переключение процессора с выполнения одной задачи на выполнение другой происходит (невытесняющая многозадачность) только в случае, если активная задача сама отказывается от процессора (для выполнения операции 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.

Критерием эффективности **систем разделения времени** (СРаВ) является удобство и эффективность работы пользователя. Каждому пользователю СРаВ предоставляется терминал для диалога со своей программой. При этом каждой задаче пользователя (по очереди) выделяется некоторый квант процессорного времени, и ни одна задача не занимает процессор надолго. Если квант невелик, то у всех пользователей, одновременно работающих на одной и той же ЭВМ, складывается впечатление, что каждый из них единолично использует машину.

СРаВ обладают меньшей пропускной способностью, чем СПаО, так как на выполнение принимается каждая запущенная пользователем задача, не оптимальная для системы, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу.

Системы реального времени (СРеВ) применяются для управления различными техническими объектами (станок, спутник, научная экспериментальная установка...) или технологическими процессами (сборочный конвейер, доменный процесс...). В них закладывается жесткое ограничение на время выполнения конкретной программы управления объектом (иначе может произойти авария). Таким образом, критерием эффективности для СРеВ является их реактивность — способность выдерживать заранее заданные интервалы времени реакции системы или управляющего воздействия (длительность между запуском программы и получением результата). Для таких систем мультипрограммная смесь представляет собой фиксированный набор специальным образом спроектированных программ, а выбор программы на выполнение осуществляется исходя из текущего состояния объекта или в соответствии с расписанием плановых работ.

Некоторые ОС могут совмещать в себе свойства систем указанных выше 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.

Вопросы и упражнения

1. Что представляет собой современная ВС?
2. Какова иерархическая структура современной ВС?
3. Перечислите основные компоненты современного компьютера.
4. Чем различаются СПО и ППО?
5. Какая задача выполнятся ОС непосредственно для пользователя?
6. Какие функции ОС предоставляются различным пользователям (например, программисту, системному администратору и дизайнеру)?
7. Какие функции выполняет ОС как менеджер ресурсов компьютера?
8. Определите понятие «операционная система».
9. Может ли компьютер работать без ОС?

10. Что определяло эволюцию ОС?
11. Для какого периода характерны наиболее дорогие и сложные ОС?
12. Какой фактор представляется в настоящее время ведущим для дальнейшего развития ОС?
13. Назовите некоторые варианты классификации ОС.
14. Укажите различия режимов применения компьютеров.

ФУНКЦИОНАЛЬНОСТЬ ОС. ПОЛЬЗОВАТЕЛЬСКИЙ АСПЕКТ

С точки зрения пользователя основными задачами ОС являются: запуск и выполнение приложений (задействованы: процессор и ОП); обеспечение длительного хранения данных и управление ими (внешняя память); использование устройств ввода/вывода и передачи сообщений по сети (внешние устройства).

Командная среда

Для ведения диалога ОС предоставляет пользователю некоторую командную среду. Исторически первые командные среды представляли собой интерпретаторы текстовых формальных языков (**интерфейс командной строки**). Команды вызова требуемых функций ОС, вводимые пользователем с клавиатуры, дополняются необходимыми параметрами (рис. 3). Синтаксически ошибочные команды обычно игнорируются.

```
/home/gr11# mkdir foo
/home/gr11# ls -F
foo/
/home/gr11# cd foo
/home/gr11/foo#
```

Рис. 3. Интерфейс командной строки ОС UNIX

Позднее появились «более дружественные» **полноэкранные текстовые диалоговые оболочки**, упрощающие ввод команд пользователем за счет использования специальных элементов управления (меню, псевдокнопки, окна диалога, «горячие» клавиши...) и улучшающие представление возможностей и текущего состояния системы, а также результатов выполнения команд (рис. 4). В случае возникновения ошибок при выполнении команд пользователя командная среда автоматически модифицируется в соответствии с ситуацией.

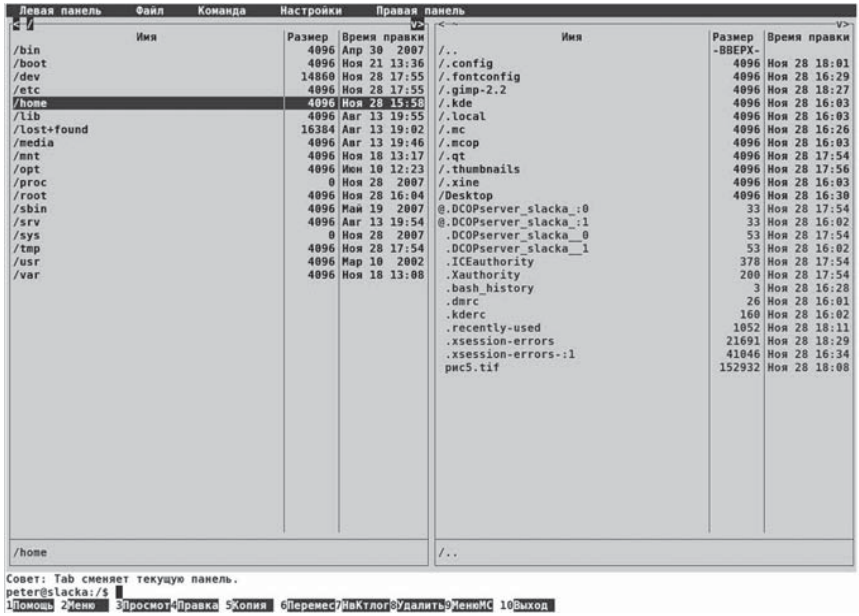

Рис. 4. Интерфейс диалоговой оболочки Midnight Commander

В настоящее время большинство ОС имеют в своем составе «максимально дружественные» командные среды, реализованные в виде **графических оболочек** (рис. 5).

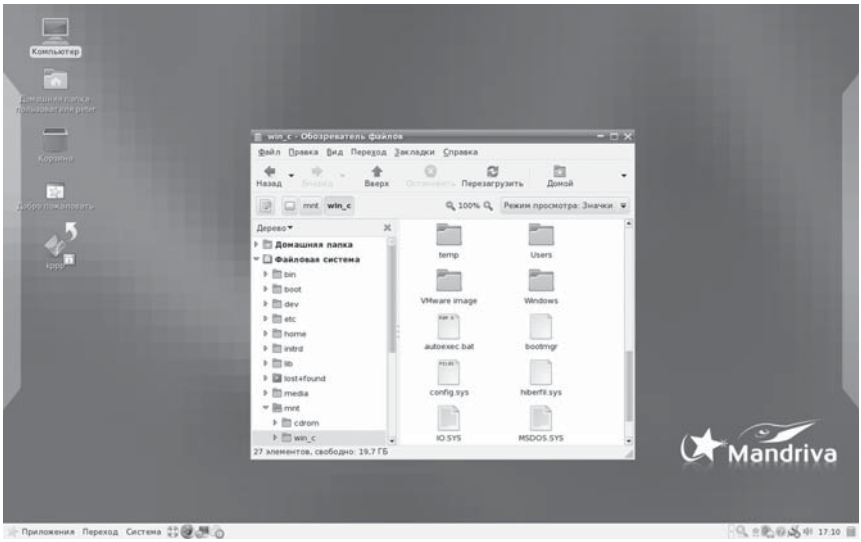

Рис. 5. Интерфейс графической оболочки Gnom

Организация и управление данными (файловая система)

Файловая система — это логическая часть ОС, предназначенная для создания удобного пользовательского интерфейса при работе с данными, хранящимися во внешней памяти, и обеспечения совместного использования указанных данных несколькими пользователями или процессами.

Таким образом, **файл** — именованная структура данных, содержащая пользовательскую или системную информацию, которая хранится во внешней памяти

В широком смысле понятие «файловая система» включает:

- совокупность всех файлов на диске,

- наборы структур данных, используемых для управления файлами (каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске...),
- комплекс системных программных средств, реализующих управление файлами (создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами).

Именование файлов

Файлы идентифицируются пользователями посредством **символьных имен** с учетом ограничений ОС на используемые символы и их количество (длину имени).

В файловой системе FAT длина имен ограничивается известной схемой 8.3 (8 символов — собственно имя, 3 символа — расширение имени), а в ОС UNIX System V имя не может содержать более 14 символов. Поскольку пользователю гораздо удобнее работать с длинными именами, имеющими достаточную информативность, современные файловые системы поддерживают длинные символьные имена файлов (NTFS Windows NT и ext2 Linux устанавливают допустимую длину имени файла в 255 символов, не считая завершающего нулевого символа).

При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем. Например, в ОС Windows 95 используется файловая система VFAT, представляющая собой существенно измененный вариант FAT. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важно было реализовать хранение длинных имен так, чтобы метод хранения и структура данных на диске принципиально не изменились.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым **составным именем**, представляющим собой последовательность символьных имен соответствующих каталогов (**путь**). В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае ОС присваивает файлу дополнительно уникальное имя, представляющее собой числовой идентификатор (номер индексного дескриптора в системе UNIX).

Типы файлов

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы, в свою очередь, подразделяются на текстовые и двоичные. **Текстовые файлы** состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. **Двоичные файлы** не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все ОС должны уметь распознавать хотя бы один тип файлов — их собственные исполняемые файлы.

Специальные — это файлы, ассоциированные с устройствами ввода/вывода, которые позволяют пользователю выполнять операции ввода/вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода/вывода, делятся на блок-ориентированные и байт-ориентированные (см. раздел Управление ВУ).

Каталог — это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны —

это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

Рис. 6. Характеристики файлов в системе а) FAT и б) UNIX System V

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

— информация о разрешенном доступе,
— пароль для доступа к файлу,
— владелец файла,
— создатель файла,
— признак «только для чтения»,
— признак «скрытый файл»,
— признак «системный файл»,
— признак «архивный файл»,
— признак «двоичный/символьный»,
— признак «временный» (удалить после завершения процесса),
— признак блокировки,
— длина записи,
— указатель на ключевое поле в записи,
— длина ключа,

— времена создания, последнего доступа и последнего изменения,
— текущий размер файла,
— максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS (рис. 6а), или ссылаться на таблицы (дескрипторов), содержащие эти характеристики, как это реализовано в ОС UNIX (рис. 6б). Каталоги могут быть одноуровневыми (рис. 7а) или образовывать иерархическую структуру за счет включения каталогов более низкого уровня в каталог более высокого уровня.

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть — если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру (рис. 7б), а в UNIX — сетевую (рис. 7в). Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

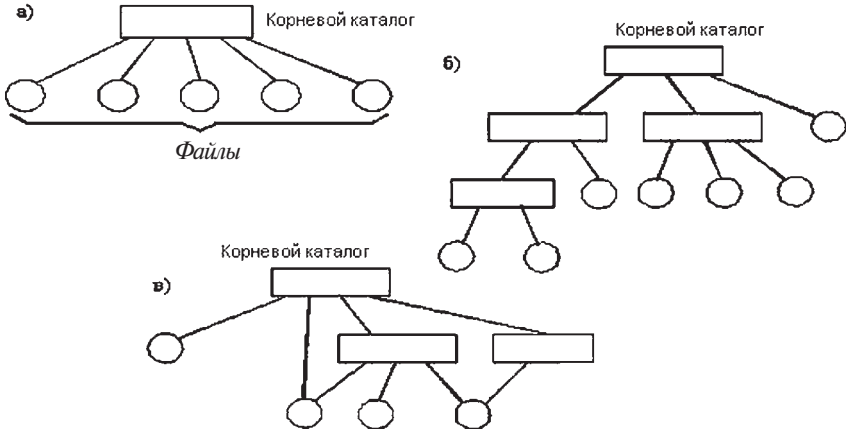

Рис. 7. Иерархия каталогов а) отсутствует, б) FAT и в) SV

Надежность, защищенность и управление пользователями

Как любая программа, ОС содержит ошибки, наличие которых может приводить к функциональным сбоям в ситуациях с экстремальными нагрузками на ресурсы. Принимая во внимание также вероятные отказы АО, всегда следует допускать реальную возможность потери работоспособности всей ВС в целом, что, в свою очередь, определяет потерю пользователем времени, затраченного на обработку информации, и результатов данной обработки.

В многопользовательской системе большую актуальность имеет также проблема предотвращения несанкционированного доступа пользователя к данным и ресурсам системы, находящимся вне его компетенции (прав использования) с целью их чтения, изменения или разрушения.

Поэтому **надежность** — устойчивость к ошибкам и сбоям и **защищенность** — способность противостоять попыткам нарушения прав использования со стороны пользователей являются важными факторами при проектировании ОС. Для их реализации используются специальные архитектурные решения, а также методы парольной защиты и криптографические, механизмы запрета и блокировки.

Основные функции подсистемы защиты ОС:

1. **Разграничение доступа**. Каждый пользователь системы может работать только с теми объектами, которые ему доступны в соответствии с текущей политикой безопасности.
2. **Идентификация и аутентификация**. Ни один пользователь не может начать работу с системой, не идентифицировав себя и не предоставив системе аутентифицирующую информацию, подтверждающую, что он действительно является тем, за кого себя выдает.
3. **Аудит**. ОС регистрирует в специальном журнале события, потенциально опасные для поддержания безопасности системы. Записи об этих событиях могут просматривать в дальнейшем только администраторы ОС.
4. **Управление политикой безопасности**. **Политика безопасности** — система мер (действий) по обеспечению инфор-

мационной защиты должна постоянно поддерживаться в адекватном состоянии, т.е. гибко реагировать на изменения условий функционирования ОС, требований к защите информации, хранимой и обрабатываемой в системе, и т.д. Управление политикой безопасности осуществляется администраторами системы с использованием соответствующих средств, встроенных в ОС.

5. **Криптографические функции**. В настоящее время защита информации немыслима без использования криптографических средств защиты. В ОС шифрование используется при хранении и передаче по каналам связи паролей пользователей и других данных, критичных для безопасности системы.

Подсистема защиты практически никогда не представляет собой единый программный модуль. Каждая из перечисленных задач подсистемы защиты решается одним или несколькими программными модулями. Некоторые функции встраиваются непосредственно в ядро ОС, другие осуществляются программами утилитами. В любом случае должен существовать четко определенный интерфейс между различными модулями подсистемы защиты, используемый при взаимодействии модулей для решения общих задач информационной безопасности (ИБ).

В одних ОС (Windows NT) подсистема защиты четко выделяется в общей архитектуре ОС, в других (UNIX) защитные функции «размазаны» по различным элементам ОС.

Часто подсистема защиты ОС допускает свое расширение дополнительными программными модулями.

Базовые понятия разграничения доступа к объектам

Объект доступа — любой элемент системы, доступ к которому субъектов доступа может быть произвольно ограничен.

Метод доступа к объекту — операция, определенная для некоторого объекта. Например, для файлов могут быть определены методы доступа: создание файла, уничтожение файла, открытие/закрытие файла, чтение файла, запись в файл, дополне-

ние файла, поиск в файле, получение и установление новых значений атрибутов, переименование, выполнение файла, чтение каталога и другие операции с файлами и каталогами.

Субъект доступа — любая сущность, способная инициировать выполнение операций над объектами (обращаться к объектам по некоторым методам доступа). Например, пользователи или процессы, выполняющиеся в системе от имени и под управлением пользователей (прямым или косвенным).

Для объекта доступа может быть определен **владелец** — субъект, которому принадлежит данный объект и который несет ответственность за конфиденциальность содержащейся в объекте информации, а также за целостность и доступность объекта. Обычно владельцем объекта автоматически назначается субъект, создавший его. В дальнейшем владелец объекта может быть изменен с использованием соответствующего метода доступа к объекту. На владельца, как правило, возлагается ответственность за корректное ограничение прав доступа к данному объекту других субъектов.

Право доступа к объекту — право на выполнение к объекту некоторого метода или группы методов доступа. Например, если пользователь имеет возможность читать файл, говорят, что он имеет право на чтение этого файла.

Понятие метода доступа и понятие права доступа не идентичны.

Например, в ОС UNIX право на запись в файл дает возможность субъекту обращаться к файлу как по методу «запись», так и по методу «добавление», при этом, поскольку право доступа «добавление» в UNIX отсутствует, невозможно разрешить субъекту операцию добавления, одновременно запретив операцию записи.

Говорят, что субъект имеет некоторую **привилегию**, если он имеет право на доступ по некоторому методу или группе методов ко всем объектам ОС, поддерживающим данный метод доступа. Например, если субъект ОС Windows NT имеет привилегию отлаживать программы, он имеет право доступа ко всем объектам типа «процесс» и «поток» по группе методов, используемых отладчиками при отладке программ.

Разграничением доступа субъектов к объектам является совокупность правил, определяющая для каждой тройки субъект-объект-метод, разрешен ли доступ данного субъекта к данному объекту по данному методу. При избирательном разграничении доступа возможность доступа определена однозначно для каждой тройки, при полномочном разграничении доступа ситуация несколько сложнее.

Субъект доступа является **суперпользователем**, если он имеет возможность игнорировать правила разграничения доступа к объектам.

Правила разграничения доступа, действующие в ОС, устанавливаются администраторами системы при определении текущей политики безопасности. За соблюдением этих правил субъектами доступа следит **монитор ссылок** — часть подсистемы защиты ОС. Правила разграничения доступа должны удовлетворять следующим требованиям.

- Соответствовать аналогичным правилам, принятым в организации, в которой установлена ОС.
- Не должны допускать разрушающие воздействия субъектов доступа, не обладающих соответствующими привилегиями, на ОС, выражающиеся в несанкционированном изменении, удалении или другом воздействии на объекты, жизненно важные для обеспечения нормального функционирования ОС.
- Любой объект доступа должен иметь владельца. Присутствие объектов, не имеющих владельца, недопустимо.
- Присутствие **недоступных объектов** — объектов, к которым не может обратиться ни один субъект доступа ни по одному методу доступа, непозволительно. Недоступные объекты фактически бесполезно растрачивают аппаратные ресурсы компьютера.
- Утечка конфиденциальной информации недопустима.

1. Избирательное разграничение доступа.

Система правил **избирательного,** или **дискреционного,** разграничения доступа (discretionary access control) формулируется следующим образом.

~ Для любого объекта ОС существует владелец.

~ Владелец объекта может произвольно ограничивать доступ других субъектов к данному объекту.

~ Для каждой тройки субъект-объект-метод возможность доступа определена однозначно.

~ Существует хотя бы один привилегированный пользователь (администратор), имеющий возможность обратиться к любому объекту по любому методу доступа. Это не означает, что этот пользователь может игнорировать разграничение доступа к объектам и поэтому является суперпользователем. Не всегда для реализации возможности доступа к объекту ОС администратору достаточно просто обратиться к объекту. Например, в Windows NT администратор для обращения к чужому (принадлежащему другому субъекту) объекту должен вначале объявить себя владельцем этого объекта, использовав привилегию администратора объявлять себя владельцем любого объекта, затем дать себе необходимые права, и только после этого администратор может обратиться к объекту. При этом использование администратором своей привилегии не остается незамеченным для прежнего владельца объекта.

Последнее требование введено для реализации механизма удаления потенциально недоступных объектов.

При создании объекта его владельцем назначается субъект, создавший данный объект. В дальнейшем субъект, обладающий необходимыми правами, может назначить объекту нового владельца. Обычно при изменении владельца объекта допускается назначать новым владельцем объекта только субъекта, изменяющего владельца объекта. Другими словами, субъект, изменяющий владельца объекта, может назначить новым владельцем объекта только себя. Такое ограничение вводится для того, чтобы владелец объекта не мог отдать «владение» объектом другому

субъекту и тем самым снять с себя ответственность за некорректные действия с объектом.

Для определения прав доступа субъектов к объектам при избирательном разграничении доступа используется **матрица доступа**. Строки этой матрицы представляют собой объекты, столбцы — субъекты (или наоборот). В каждой ячейке матрицы хранится совокупность прав доступа, предоставленных данному субъекту на данный объект.

Имена пользователей	Имена файлов			
	post.txt	base.dbf	write.exe	prime.gif
goblin	читать	читать	выполнять	читать, писать
tur	читать	читать, писать	–	–
vanya	читать	–	–	читать
nata	читать, писать	читать	выполнять	читать

Рис. 8. Матрица прав доступа к файлам

Поскольку матрица доступа очень велика (типичный объем для современной ОС составляет несколько десятков мегабайтов), матрица доступа никогда не хранится в системе в явном виде. Для сокращения объема матрицы доступа используется объединение субъектов доступа в группы. Права, предоставленные группе субъектов для доступа к данному объекту, предоставляются каждому субъекту группы.

Для каждого объекта доступа хранятся его **атрибуты защиты** (идентификатор владельца объекта и строка матрицы доступа в кодированном виде).

На практике используются два способа хранения строки матрицы доступа:

Вектор доступа (UNIX) — запись фиксированной длины, разбитая на несколько полей. Каждое поле описывает права досту-

па к данному объекту некоторого субъекта. С помощью вектора доступа можно описать права доступа к объекту только фиксированного числа субъектов, что накладывает существенные ограничения на систему разграничения доступа.

Список доступа (VAX/VMS, Windows NT) — запись переменной длины, элементами которой являются поля, содержащие:

- идентификатор субъекта;
- права, предоставленные этому субъекту на данный объект;
- различные флаги и атрибуты.

Фактически вектор доступа представляет собой список доступа фиксированной длины и является его частным случаем.

Хранение матрицы доступа в виде совокупности списков доступа позволяет реализовать более мощный и гибкий механизм разграничения доступа, однако требует гораздо больше оперативной и дисковой памяти для хранения атрибутов защиты объекта, усложняет техническую реализацию правил разграничения доступа и создает проблему, связанную с тем, что значения элементов списка доступа могут противоречить друг другу.

Предположим, один элемент списка доступа разрешает некоторому пользователю доступ к объекту, а другой элемент списка запрещает доступ к объекту группе, в которую входит этот пользователь. При использовании списков доступа правила разграничения доступа должны включать в себя правила разрешения этих противоречий.

При создании нового объекта владелец объекта должен определить права доступа различных субъектов к этому объекту. Если владелец объекта не сделал этого, то либо новому объекту назначаются атрибуты защиты по умолчанию, либо новый объект наследует атрибуты защиты от родительского объекта (каталога, контейнера и т.д.).

Избирательное разграничение доступа является наиболее распространенным механизмом ОС. Это обусловлено сравнительной простотой реализации и необременительностью правил доступа для пользователей. Вместе с тем защищенность ОС, подсистема защиты которой реализует только избирательное разграничение доступа, во многих случаях недостаточна.

2. Изолированная программная среда. Изолированная, или **замкнутая, программная среда** представляет собой расширение модели избирательного разграничения доступа модификацией третьего правила:

~ Для каждой четверки субъект-объект-метод-процесс возможность доступа определена однозначно.

И добавлением нового правила:

~ Для каждого субъекта определен список программ, которые этот субъект может запускать.

При использовании изолированной программной среды права субъекта на доступ к объекту определяются не только правами и привилегиями субъекта, но и процессом, с помощью которого субъект обращается к объекту. Можно, например, разрешить обращаться к файлам с расширением .doc только конкретным программам (Word, Word Viewer).

Изолированная программная среда существенно повышает защищенность ОС от разрушающих программных воздействий, включая программные закладки и компьютерные вирусы. Кроме того, при использовании данной модели повышается целостность данных, хранящихся в системе. В то же время изолированная программная среда создает определенные сложности в администрировании ОС. Например, при инсталляции нового программного продукта администратор должен модифицировать списки разрешенных программ для пользователей, которые должны иметь возможность работать с этим программным продуктом.

3. Полномочное разграничение доступа. Полномочное, или **мандатное,** разграничение доступа (mandatory access control) также применяется в совокупности с избирательным. Правила разграничения доступа в данной модели формулируются следующим образом

~ Для любого объекта ОС существует владелец.

~ Владелец объекта может произвольно ограничивать доступ других субъектов к данному объекту.

~ Для каждой четверки субъект-объект-метод-процесс возможность доступа определена однозначно в каждый момент

времени. При изменении состояния процесса со временем возможность предоставления доступа также может измениться, т.е. если в некоторый момент времени к некоторому объекту разрешен доступ некоторого субъекта посредством некоторого процесса, это не означает, что в другой момент времени доступ тоже будет разрешен. Вместе с тем в каждый момент времени возможность доступа определена однозначно — никаких случайных величин здесь нет. Поскольку права процесса на доступ к объекту меняются с течением времени, они должны проверяться не только при открытии объекта, но и перед выполнением над объектом таких операций, как чтение и запись.

~ Существует хотя бы один привилегированный пользователь (администратор), имеющий возможность удалить любой объект.

~ В множестве объектов выделяется множество **объектов полномочного разграничения доступа**. Каждый объект полномочного разграничения доступа имеет гриф секретности. Чем выше числовое значение грифа секретности, тем секретнее объект. Нулевое значение грифа секретности означает, что объект несекретен. Если объект не является объектом полномочного разграничения доступа или если объект несекретен, администратор может обратиться к нему по любому методу, как и в предыдущей модели разграничения доступа.

~ Каждый субъект доступа имеет **уровень допуска**. Чем выше числовое значение уровня допуска, тем больший допуск имеет субъект. Нулевое значение уровня допуска означает, что субъект не имеет допуска. Обычно ненулевое значение допуска назначается только субъектам-пользователям и не назначается субъектам, от имени которых выполняются системные процессы.

При использовании данной модели разграничения доступа существенно страдает производительность ОС, поскольку права доступа к объекту должны проверяться не только при открытии объекта, но и при каждой операции чтения/записи.

Кроме того, данная модель разграничения доступа создает пользователям определенные неудобства, связанные с тем, что если уровень конфиденциальности процесса строго выше нуля, то вся информация в памяти процесса фактически является

секретной и не может быть записана в несекретный объект. Если процесс одновременно работает с двумя объектами, только один из которых является секретным, процесс не может записывать информацию из памяти во второй объект.

Эта проблема решается посредством использования специального программного интерфейса (API) для работы с памятью. Области ОП, выделяемые процессам, могут быть описаны как объекты полномочного разграничения доступа, после чего им могут назначаться грифы секретности.

Из вышеизложенного следует, что пользователи ОС, реализующих данную модель разграничения доступа, вынуждены использовать ПО, разработанное с учетом этой модели. В противном случае пользователи будут испытывать серьезные проблемы в процессе работы с объектами ОС, имеющими ненулевой гриф секретности. Пусть, например, пользователь работает в среде Windows с установленным дополнительным пакетом защиты, реализующим данную модель разграничения доступа. Пользователь открывает с помощью Microsoft Word два документа, один из которых является секретным. Уровень конфиденциальности процесса winword.exe повышается, и пользователь теряет возможность сохранить изменения, внесенные им в данном сеансе работы в несекретный документ.

Также вызывает определенные проблемы вопрос о назначении грифов секретности создаваемым объектам. Если пользователь создает новый объект с помощью процесса, имеющего ненулевой уровень конфиденциальности, пользователь вынужден присвоить новому объекту гриф секретности не ниже уровня конфиденциальности процесса. Во многих ситуациях это неудобно.

Сравнительный анализ моделей разграничения доступа

Каждая из приведенных моделей разграничения доступа имеет свои достоинства и недостатки. Табл. 1 позволяет провести их сравнительный анализ.

Таблица 1

Свойства модели	Избирательное разграничение доступа	Изолированная программная среда	Полномочное разграничение доступа
Защита от утечки информации	—	—	+
Защищенность от разрушающих воздействий	низкая (н)	высокая (в)	н
Сложность реализации	н	средняя (с)	в
Сложность администрирования	н	с	в
Затраты ресурсов компьютера	н	н	в
Использование ПО, разработанного для других систем	+	+	—

Вопросы и упражнения

1. Назовите разновидности диалоговых сред (пользовательских интерфейсов). Уточните их преимущества и недостатки.
2. Определите понятия файл и файловая система.
3. Как связаны между собой функции управления пользователями и обеспечения защищенности ОС?
4. Почему ранние версии файловой системы FAT не позволяют хранить длинные имена файлов?
5. Как решается проблема совместимости при использовании длинных имен файлов?
6. Где хранятся имена файлов в различных файловых системах?
7. Чем различаются древовидная и сетевая иерархии ФС?
8. Какие функции выполняет подсистема защиты ОС?
9. Поясните понятия: идентификация, аутентификация, криптография, политика безопасности и аудит в контексте ОС.
10. Определите основные понятия разграничения доступа.
11. Назовите основные модели разграничения доступа. Какие правила им соответствуют?

12. Что такое матрица прав доступа?
13. Назовите и обоснуйте достоинства и недостатки основных моделей разграничения доступа?
14. Что общего в понятиях надежность и защищенность и чем они различаются в контексте ОС?

ФУНКЦИОНАЛЬНОСТЬ ОС. АППАРАТНЫЙ АСПЕКТ

Как менеджер ресурсов для параллельной реализации нескольких задач пользователя ОС выполняет следующие функции: выделение процессорного времени приложениям, своевременное перемещение программ между внешней и ОП, управление устройствами ввода/вывода. При этом целевой функцией часто является обеспечение оптимальной пропускной способности ВС отношению к приложениям пользователя.

ОС представляет собой по сути обычную компьютерную программу, которая способна извлекать команды из различных участков ОП, передавая таким образом управление различным компонентам ПО.

Чтобы повысить эффективность ВС и оптимизировать использование ее ресурсов, в современных ОС необходима аппаратная поддержка многозадачности, реализация концепций квантования и приоритетов, прерывания, долгосрочного и текущего планирования, виртуальной памяти, кэширования.

Средства аппаратной поддержки многозадачности

Аппаратная поддержка многозадачности в однопроцессорной системе включает в себя средства:

- поддержки привилегий команд;
- динамической трансляции адресов;
- переключения процессов;
- прерываний;

– системный таймер;
– защиты областей памяти.

Средства поддержки привилегий команд обычно основаны на системном регистре процессора (слово состояния машины или процессора), который содержит некоторые признаки, определяющие режимы работы процессора (текущий режим привилегий). Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды.

Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро-пользователь) или четыре (например, ядро-супервизор-выполнение-пользователь у платформы VAX или 0-1-2-3 у процессоров Intel x86/Pentium). В обязанности средств поддержки привилегированного режима входит выполнение проверки допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности.

Средства трансляции адресов выполняют операции преобразования виртуальных адресов, которые содержатся в кодах процесса, в адреса физической памяти (см. раздел Управление ОП). Таблицы, используемые при трансляции адресов, обычно имеют большой объем, поэтому хранятся в системных областях ОП, а регистры процессора содержат только указатели на эти области. Средства трансляции адресов используют данные указатели для доступа к элементам таблиц и аппаратного выполнения алгоритма преобразования адреса, что значительно ускоряет процедуру трансляции по сравнению с ее чисто программной реализацией.

Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным (см. раздел Распределение процессорного времени). Содержимое контекста обычно включает содержимое всех регистров общего назначения процессора, регистра флагов операций (то есть флагов нуля, переноса, переполнения и т.п.), а также тех системных регистров и указателей, которые связаны с отдельным процессом, а не ОС, например, указатель на таблицу

трансляции адресов процесса. Для хранения контекстов приостановленных процессов также используются системные области ОП, которые поддерживаются указателями процессора.

Переключение контекста выполняется по определенным командам процессора, например, по команде перехода на новую задачу. Такая команда вызывает автоматическую загрузку данных сохраненного контекста в регистры процессора, после чего процесс продолжается с прерванного ранее места.

Система прерываний позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу внешних устройств, быстро переходить с одной программы на другую. **Прерывание** — временное прекращение (приостановка) ЦП выполнения текущей задачи, вызванное внешним по отношению к нему событием, и переход к обработке возникшей ситуации.

Примерами таких событий могут служить: завершение операции ввода/вывода внешним устройством (например, запись блока данных контроллером диска), некорректное завершение арифметической операции (например, переполнение регистра), истечение интервала астрономического времени. При возникновении условий прерывания его источник (контроллер внешнего устройства, таймер, арифметический блок процессора и т.п.) отправляет определенный электрический сигнал. Этот сигнал прерывает выполнение процессором последовательности команд, задаваемой исполняемым кодом, и вызывает автоматический переход на заранее определенную процедуру, называемую **процедурой обработки прерываний**.

В большинстве моделей процессоров отрабатываемый аппаратурой переход на процедуру обработки прерываний сопровождается заменой слова состояния машины (или даже всего контекста процесса), что позволяет одновременно с переходом по нужному адресу выполнить переход в привилегированный режим. После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода.

Прерывания играют важнейшую роль в работе любой ОС, являясь ее движущей силой. Большая часть действий ОС инициируется прерываниями различного типа. Даже системные вызовы

от приложений выполняются на многих аппаратных платформах с помощью специальной инструкции прерывания, вызывающей переход к выполнению соответствующих процедур ядра (например, инструкция int в процессорах Intel или SVC в мэйнфреймах IBM).

Системный таймер (регистр-счетчик) необходим ОС для выдержки интервалов времени. Для этого в регистр таймера программно загружается значение требуемого интервала в условных единицах, из которого затем автоматически с определенной частотой начинает вычитаться по единице. Частота «тиков» таймера, как правило, тесно связана с частотой тактового генератора процессора. (Не следует путать таймер ни с тактовым генератором, который вырабатывает сигналы, синхронизирующие все операции в компьютере, ни с системными часами — работающей на батареях электронной схеме, — которые ведут независимый отсчет времени и календарной даты.) При достижении нулевого значения счетчика таймер инициирует прерывание, которое обрабатывается процедурой ОС. Прерывания от системного таймера используются ОС в первую очередь для слежения за тем, как отдельные процессы расходуют время процессора. Например, в системе разделения времени при обработке очередного прерывания от таймера планировщик процессов может принудительно передать управление другому процессу, если данный процесс исчерпал выделенный ему квант времени.

Средства защиты областей ОП обеспечивают на аппаратном уровне проверку возможности программного кода осуществлять с данными определенной области памяти такие операции, как чтение, запись или выполнение (при передачах управления). Если аппаратура компьютера поддерживает механизм трансляции адресов, то средства защиты областей памяти встраиваются в этот механизм. Функции аппаратуры по защите памяти обычно состоят в сравнении уровней привилегий текущего кода (команды) процессора и области памяти, к которой производится обращение.

Распределение процессорного времени (подсистема управления процессами)

Процесс (задача) — абстракция (термин), описывающая выполняющуюся программу. Для ОС процесс представляет собой единицу работы, заявку на распределение системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, создает и уничтожает процессы, обеспечивает их другими необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Состояние процессов

В многозадачной системе процесс может находиться в одном из трех основных состояний:

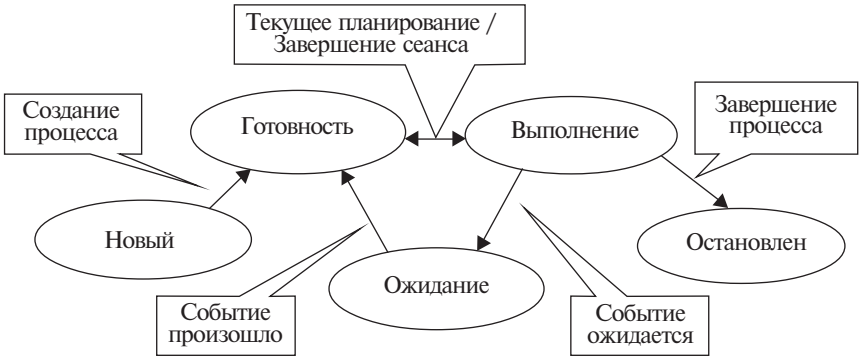

Рис. 9. Граф состояний процесса в многозадачной ВС

Выполнение (В) — активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

Ожидание (О) — пассивное состояние процесса, когда процесс заблокирован и не может выполняться по внутренним при-

чинам: пребывает в ожидании некоторого события (завершения операции ввода/вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса);

Готовность (Г) — также пассивное состояние процесса, когда процесс заблокирован в связи с внешними обстоятельствами: процесс имеет все требуемые ресурсы, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной ОС.

В состоянии **В** в однопроцессорной системе может находиться только один процесс, а в каждом из состояний **О** и **Г** — несколько процессов, образующих очереди, соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния **Г**. При активизации процесс переходит в состояние **В** и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние **О**, либо будет «вытеснен» из процессора, например, вследствие исчерпания отведенного ему кванта процессорного времени. В последнем случае процесс возвращается в состояние **Г**. В это же состояние процесс переходит из состояния **О**, после того как ожидаемое событие произойдет.

Дескриптор и контекст процесса

Программный код начинает выполняться после создания соответствующего процесса. Создание процесса означает:

— создание дескриптора;
— включение его в очередь готовых процессов;
— загрузку кодового сегмента процесса в ОП или в область свопинга (swapping, «подкачка» на/с диска).

Дескриптор процесса — структура данных, необходимых для идентификации процесса (идентификатор процесса, данные о степени привилегированности, место нахождения кодового сегмента, состояние...).

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено в соответствии с

алгоритмом квантования процессорного времени. При возобновлении выполнения процесса, необходимо восстановить состояние его операционной среды (значения регистров и программного счетчика, режим работы процессора, указатели на открытые файлы, информация о незавершенных операциях ввода/вывода, коды ошибок системных вызовов...). Эта информация называется **контекстом** процесса.

Дескриптор и контекст составляют блок управления процессом. Дескриптор необходим для планирования процессов. Контекст процесса содержит менее актуальную информацию и используется ОС при возобновлении прерванного процесса.

Алгоритмы планирования процессов

По виду решаемых ОС задач различают долгосрочное, среднесрочное, краткосрочное и планирование ввода/вывода.

При долгосрочном планировании определяется очередь выполняемых процессов (отбор программ на выполнение). Пока ВС не перегружена, все запущенные пользователем программы поступят на выполнение. Среднесрочное планирование решает вопрос о приостановке процессов и временном вытеснении их из ОП во внешнюю (свопинг). При планировании ввода/вывода определяется очередь процессов на использование доступных внешних устройств.

Наиболее интересное для нас краткосрочное (текущее) планирование процессов включает в себя решение следующих задач:

- определение момента времени для смены выполняемого процесса;
- выбор процесса на выполнение из очереди готовых процессов (списка их дескрипторов);
- переключение контекстов «старого» и «нового» процессов.

Существует различные алгоритмы планирования процессов, по-разному решающие вышеперечисленные задачи, преследующие различные цели и обеспечивающие различное качество мультипрограммирования. Чаще встречаются алгоритмы, основанные на квантовании и приоритетах.

В соответствии с алгоритмами, основанными на квантовании (циклические), процесс, который исчерпал свой **квант** (временной интервал обслуживания процессором), переводится в состояние **Г** и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени.

Кванты, выделяемые процессам, могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса. Процессы, которые не полностью использовали выделенный им квант (например, из-за ожидания выполнения операций ввода/вывода), могут получить компенсацию в виде привилегий при последующем обслуживании. По разным правилам может быть организована очередь готовых процессов: «первый пришел — первый обслужился» (FIFO) или «последний пришел — первый обслужился» (LIFO).

Другая группа алгоритмов использует понятие «приоритет» процесса и реализуется на базе прерываний. **Приоритет** — это число, характеризующее степень привилегированности процесса при использовании системных ресурсов. Чем выше приоритет, тем выше привилегии.

Приоритет может выражаться целыми или дробными, положительным или отрицательным значениями. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться пользователем либо вычисляться самой ОС по определенным правилам. Он может оставаться фиксированным на протяжении всей жизни процесса либо — динамическим, т.е. изменяться во времени в соответствии с некоторым законом.

Чаще алгоритмы планирования разрабатываются с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора процесса из очереди готовых определяется приоритетами процессов.

По принципам переключения процессов (выход из состояния выполнения) различают вытесняющие (preemptive) и невытесняющие (non-preemptive) алгоритмы.

Различие между этими вариантами многозадачности определяется уровнем централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в ОС (программист пишет приложение, не заботясь о параллельном выполнении его с другими). При этом ОС определяет момент снятия с выполнения активного процесса, запоминает его контекст, выбирает из очереди готовых следующий и запускает его на выполнение, загружая нужный контекст.

При невытесняющей многозадачности механизм планирования распределен между ОС и приложениями. Прикладная программа, получив управление процессором от ОС, сама определяет момент завершения своей очередной итерации, после чего передает управление назад ОС с помощью какого-либо системного вызова. ОС лишь формирует очереди задач и выбирает следующую задачу на выполнение. Такой механизм создает проблемы переключения между задачами и снижает надежность ОС.

Во всех современных ОС, ориентированных на высокопроизводительное выполнение комплекса приложений (UNIX, Windows NT, OS/2...), реализована вытесняющая многозадачность.

Средства взаимодействия и синхронизации (согласования) процессов

Процессы называют **параллельными**, если они существуют в системе одновременно. Параллельные процессы могут выполняться независимо друг от друга или быть **асинхронными**, т.е. периодически взаимодействовать (согласовываться для передачи данных или совместного использования ресурсов).

Обычно ОС имеет несколько разновидностей межпроцессорного обмена информацией (IPC, InterProcess Communications), например: сигналы, разделяемая память, неименованные и именованные каналы, сообщения и сокеты.

Сигналы (signals) являются программным средством уведомления (прерывания) процессов ОС или другими процессами.

По своей природе сигналы (программные, от оборудования) являются асинхронными и могут быть получены процессом в любое время независимо от него самого.

Получение процессом одних сигналов приводит к аварийному его завершению. Другие сигналы процесс может игнорировать либо определить для них свои обработчики. Последние будут выполняться при получении соответствующего сигнала, после чего выполнение процесса продолжится с того места, на котором он прервался при получении сигнала.

Из-за своей асинхронности и примитивности сигналы являются не слишком удобным средством взаимодействия процессов, поскольку не позволяют передать какие-либо данные.

Адресное пространство процесса защищается ОС от вмешательства других процессов. Однако нередко возникает необходимость наличия области **разделяемой памяти** (РП), к которой имеют доступ несколько процессов.

В отличие от других механизмов IPC, РП не требует для своей эксплуатации системных вызовов. Системные вызовы необходимы только для создания РП и подключения ее к адресному пространству процесса. Работа с самой памятью производится через указатель на нее обычными средствами языка программирования.

Суть **неименованных каналов** сводится к организации циклического буфера двум процессам, один из которых пишет в канал, а второй читает (в некоторых реализациях выборочно) из канала. Таким образом, неименованный канал является однонаправленной очередью информационных блоков и для организации двунаправленного обмена данными необходимо создание двух каналов.

Из-за того что доступ к неименованным каналам осуществляется только через дескрипторы файлов, обмен данными между каналами могут производить только родственные процессы, например, родительский и дочерний.

От неименованных каналов **именованные** отличаются тем, что к ним можно обратиться через файловую систему, в которой

для именованных каналов существует специальный тип файлов. При этом контроль доступа к каналам осуществляется в рамках подсистемы ИБ ОС.

Сокеты (программные гнезда) изначально появились в ОС UNIX BSD как программный интерфейс для взаимодействия между локальными процессами и впоследствии стали одной из реализаций интерфейса протокола TCP/IP, стандартного для всех ОС.

Локальные и сетевые сокеты отличаются по способу своего создания и назначения имен. Локальные сокеты адресуются через файловую систему, сетевые — через указание IP адреса, типа протокола и номера порта на удаленной и локальной стороне.

Сообщения (message) — адресуемые процессам блоки информации (Windows).

Сложность проблемы синхронизации (согласования) процессов состоит в нерегулярности происходящих в системе событий. Ситуации, когда два или более процессов конкурируют за доступ к одному ресурсу, называются **гонками**. В ходе гонок за требуемые ресурсы могут возникнуть три различные проблемы: необходимость взаимного исключения, тупик и голодание.

Рассмотрим программу печати файлов (принт-сервер). Эта программа печатает по очереди все файлы, имена которых последовательно в порядке поступления записывают в специальный общедоступный файл «заказов» другие программы (рис. 10).

Особая переменная NEXT, также доступная всем процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла «заказов». Процессы-клиенты читают эту переменную, записывают в соответствующую позицию файла «заказов» имя своего файла и наращивают значение NEXT на единицу. Предположим, что в некоторый момент процесс **R** решил распечатать свой файл, для этого он прочитал значение переменной NEXT, предположим — 4. Процесс запомнил это значение, но поместить имя файла не успел, так как его выполнение было прервано (например, вследствие исчерпания кванта). Очередной процесс **S**, желающий распечатать файл, прочитал то же самое значение переменной NEXT, поместил в четвертую позицию имя своего файла и нарастил значение переменной на

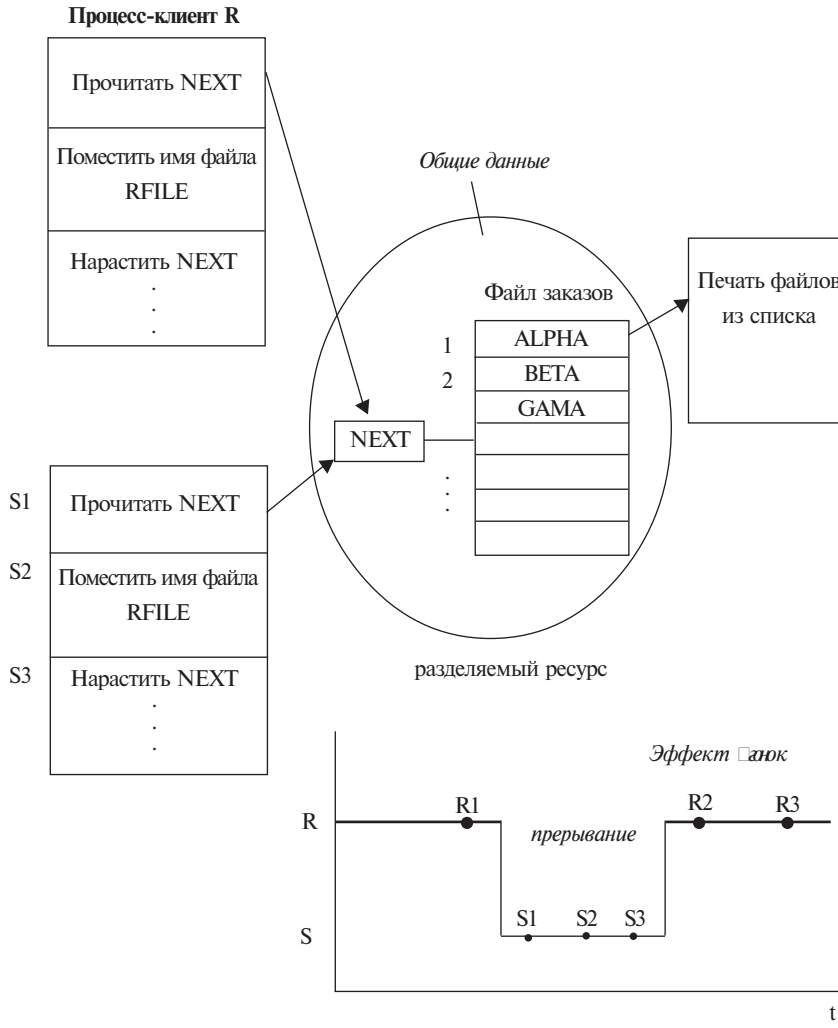

Рис. 10. Гонка за ресурсом и возможные последствия

единицу. Когда в очередной раз управление будет передано процессу **R**, то он, продолжая свое выполнение, в полном соответствии со значением текущей свободной позиции, полученным во время предыдущей итерации, запишет имя файла также в позицию 4, поверх имени файла процесса **S**. Таким образом, процесс **S** никогда не увидит свой файл распечатанным.

Часть программы в которой осуществляется доступ к разделяемым ресурсам называется **критической секцией** (разделом, critical section).

Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Такой прием называют **взаимным исключением** (мутекс, mutual exclusion).

Осуществление взаимных исключений порождает две другие проблемы.

Пусть двум процессам, выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например, принтер и диск. После того как процесс **A** занял принтер (вошел в критическую секцию), он был прерван. Управление получил процесс **B**, который сначала занял диск (вошел в критическую секцию), но при выполнении следующей команды был заблокирован, так как принтер оказался уже занятым процессом **A**. Управление снова получил процесс **A**, который в соответствии со своей программой сделал попытку занять диск и был заблокирован: диск уже распределен процессу **B**. В таком положении процессы **A** и **B** могут ожидать нужные ресурсы бесконечно долго. Такие ситуации взаимных блокировок называют **тупиками**, дедлоками (deadlocks) или клинчами (clinch).

В зависимости от соотношения скоростей процессов они могут либо совершенно независимо использовать разделяемые ресурсы (рис. 11г), либо образовывать очереди к разделяемым ресурсам (рис. 11в), либо взаимно блокировать друг друга (рис. 11б).

В рассмотренном примере тупик был образован двумя процессами, но взаимно блокировать друг друга может и большее число процессов.

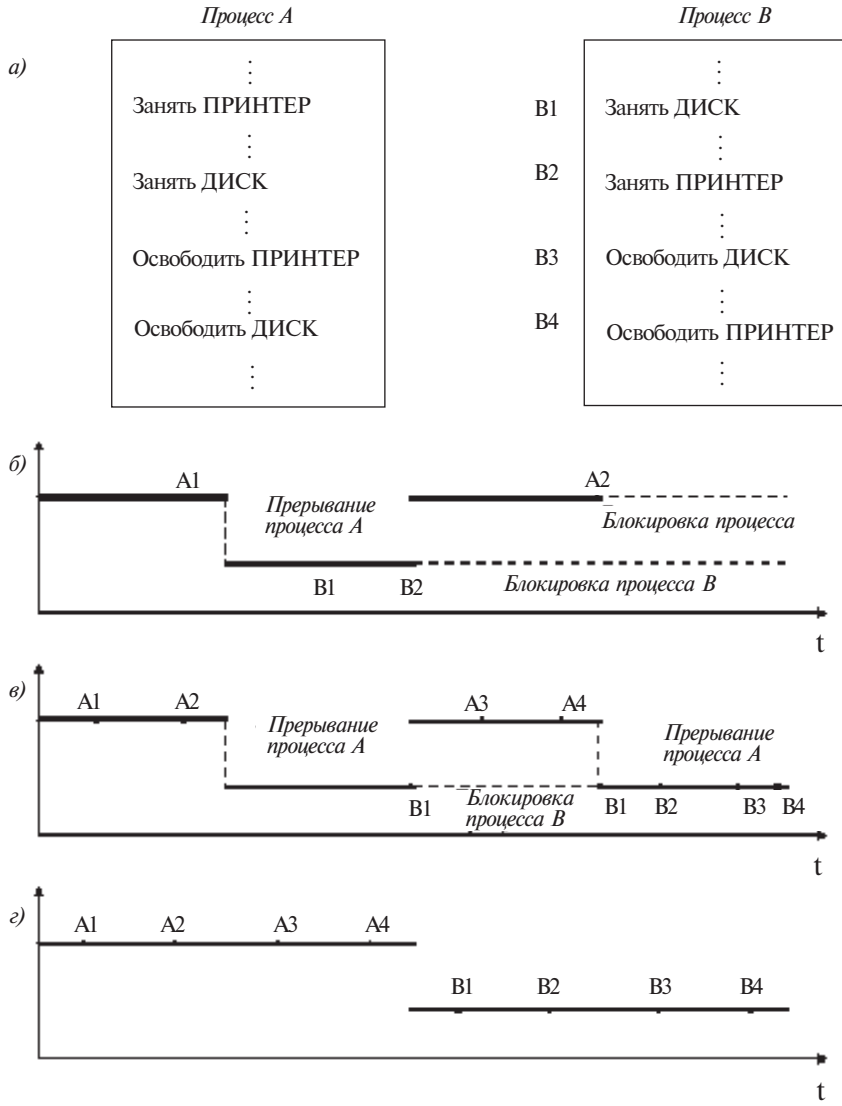
а)
Процесс А
Занять ПРИНТЕР
Занять ДИСК
Освободить ПРИНТЕР
Освободить ДИСК
Процесс В
B1
Занять ДИСК
B2
Занять ПРИНТЕР
B3
Освободить ДИСК
B4
Освободить ПРИНТЕР
б)
A1
Прерывание процесса А
A2
Блокировка процесса
B1
B2
Блокировка процесса В
t
в)
A1
A2
Прерывание процесса А
A3
A4
Прерывание процесса А
B1
Блокировка процесса В
B1
B2
B3
B4
t
г)
A1
A2
A3
A4
B1
B2
B3
B4
t

Рис. 11. Тупик

Предположим имеются три процесса: **A**, **B** и **C**, каждому из которых периодически требуется доступ к одному ресурсу. При использовании ресурса процессом **A** процессы **B** и **C** блокируются в ожидании ресурса. После освобождения ресурса процессом **A** в критическую секцию входит другой процесс, например **B**. Пока он работает с ресурсом, процессу **A** вновь может потребоваться тот же ресурс. После освобождения система по ряду причин может предоставить ресурс снова процессу A (выше приоритет), затем — процессу **B** и т.д. Такая ситуация, когда выполнение процесса (**C**) бесконечно откладывается по причине предпочтения других процессов, называется **голодание**.

Для решения указанных проблем синхронизации в ОС существуют такие механизмы, как блокирующие переменные (бинарные семафоры), считающие семафоры, очереди и мониторы.

Простейший способ обеспечить мутекс — позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, поскольку неуправляемый ОС процесс может надолго занять процессор, а при крахе процесса в критической области «зависнет» вся система (прерывания никогда не будут разрешены).

Другим способом исключения эффекта гонок является использование **блокирующих переменных**. С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят.

Перед входом в критическую секцию процесс проверяет, свободен ли ресурс (значение блокирующей переменной равно 1). Если он занят, то проверка циклически повторяется, если свободен, то значение блокирующей переменной устанавливается в 0 и процесс входит в критическую секцию. После выполнения всех действий с разделяемым ресурсом значение переменной снова устанавливается равным 1.

Если все процессы написаны с использованием указанных соглашений, то взаимное исключение гарантируется. При этом операция проверки и установки блокирующей переменной должна быть неделимой (непрерываемой).

Реализация критических секций с использованием блокирующих переменных имеет существенный недостаток: в течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время.

Для устранения циклического опроса ожидающим процессом значений блокирующей переменной используют аппарат событий. В этом случае используются системные функции WAIT(x) и POST(x), где x — идентификатор события освобождения некоторого ресурса. Если ресурс занят, то процесс не выполняет циклический опрос, а вызывает системную функцию WAIT(x), которая переводит активный процесс в состояние ожидания. Другой процесс, который в это время использует данный ресурс, после выхода из критической секции выполняет системную функцию POST(x), в результате чего ОС просматривает очередь ожидающих процессов и переводит процесс, ожидающий события x, в состояние готовности.

Обобщающее средство синхронизации процессов, позволяющее иногда учитывать и размер доступного ресурса, предложил Дейкстра в своей концепции семафоров. **Семафор** — защищенная переменная, значение которой можно опрашивать и менять только при помощи специальных операций: инициализации, уменьшения (P) и увеличения (V). Двоичные семафоры могут принимать только значения 0 и 1 (аналог блокирующей переменной). Считающие семафоры (семафоры со счетчиками) могут принимать неотрицательные целые значения, не превышающие значения инициализации. Указанные операции выполняются следующим образом:

Пусть S является семафором (ограничителем) доступа к критической секции некоторого ресурса.

Алгоритм операции P(S):

если S>0

то S:=S−1 (одним неделимым действием — выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессам во время выполнения этой операции), процесс получает ресурс;

иначе процесс ожидает освобождения ресурса в очереди на S.

Алгоритм операции V(S):

если при освобождении ресурса имеется очередь процессов, ожидающих на S

то разрешить первому из них продолжить работу;

иначе S:=S+1 также одним неделимым действием.

Процесс, требующий ресурс, пытается выполнить операцию P(S). При S=0, операция невозможна, и он переходит в состояние O, при S>0 операция выполняется, и процесс получает доступ к ресурсу (входит в критическую секцию). После освобождения ресурса процесс выполняет операцию V(S).

Очереди — последовательности сообщений, предназначенных для организации обмена между процессами блоками информации различной длины, называемыми **записями**. Процесс-отправитель снабжает записи служебной информацией, в частности, для указания получателя и помещает в очередь. Процессу-получателю при обращении к очереди возвращается содержимое первой записи с его адресом.

Монитор — это программный модуль (компонент ОС), реализующий набор специальных процедур, переменных и структур данных, необходимых для реализации динамического распределения ресурсов. Для получения ресурса процесс должен предварительно обратиться к одной из процедур монитора. Кроме предоставления требуемых ресурсов монитор обеспечивает взаимоисключение синхронных требований от разных процессов (контроль критических секций) и формирует их в очереди.

В распределенных системах, состоящих из нескольких процессоров, каждый из которых имеет собственную оперативную память, семафоры и мониторы оказываются непригодными. В таких системах синхронизация может быть реализована только с помощью обмена сообщениями.

Нити (потоки)

Обычно, ОС поддерживает параллельность (обособленность) процессов: у каждого из них имеется свое виртуальное адресное

пространство (см. раздел Управление ОП), каждому процессу назначаются свои ресурсы — файлы, окна, семафоры и т.д., что необходимо для защиты одного процесса от другого.

При мультипрограммировании повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем если бы он выполнялся в однопрограммном режиме (всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождения ресурса). Однако задача, традиционно решаемая в рамках одного процесса, может обладать внутренним параллелизмом, который в принципе позволяет ускорить ее решение. Например, в ходе выполнения задачи происходит обращение к внешнему устройству, и на время этой операции можно продолжить вычисления по другой «ветви» процесса.

Для этих целей современные ОС предлагают использовать сравнительно новый механизм многонитевой обработки (multithreading). При этом вводится новое понятие «нить» (thread, поток), а понятие «процесс» в некоторой степени меняет смысл.

Нити, относящиеся к одному процессу, не настолько изолированы друг от друга, как процессы в традиционной многозадачной системе, между ними легче организовать взаимодействие. В отличие от процессов, которые принадлежат конкурирующим приложениям, все нити одного процесса всегда принадлежат только ему, поэтому программист, разрабатывающий приложение, может предусмотреть алгоритм, не допускающий конкуренцию нитей за ресурсы. Кроме того, использование нитей может сократить необходимость в прерываниях пользовательского уровня.

Нити называют облегченными процессами или мини-процессами. Во многих отношениях они подобны процессам. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити, как и процессы, могут порождать нити-потомки, переходить из состояния в состояние. Подобно традиционным процессам (однонитевым), нити могут находиться в одном из стандартных состояний: **В**, **О**

и **Г**. Пока одна нить заблокирована, другая нить того же процесса может выполняться. Нити разделяют процессор так, как это делают процессы, в соответствии с различными вариантами планирования.

При этом различные нити в рамках одного процесса не настолько независимы, как отдельные процессы. Все такие нити имеют одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждая нить может иметь доступ к каждому виртуальному адресу, одна нить может использовать стек другой нити. Между нитями нет полной защиты. Кроме разделения адресного пространства, все нити разделяют также набор открытых файлов, таймеров, сигналов и т.п.

Многонитевая обработка повышает эффективность работы системы по сравнению с обычной многозадачной обработкой. Например, в среде Windows можно одновременно работать с табличным и текстовым процессором. Однако если пользователь запрашивает пересчет своего рабочего листа, электронная таблица блокируется до тех пор, пока эта операция не завершится, что может потребовать значительного времени. В многонитевой среде, в случае если электронная таблица была разработана с учетом возможностей многонитевой обработки, этой проблемы не возникает, и пользователь всегда имеет интерактивный доступ к электронной таблице.

В мультипроцессорных системах для нитей из одного адресного пространства имеется возможность выполняться параллельно на разных процессорах. Это один из главных путей реализации разделения ресурсов в таких системах. С другой стороны, правильно сконструированные программы, основанные на многонитевости, должны работать одинаково хорошо как на однопроцессорной машине (в режиме разделения времени между нитями), так и на настоящем мультипроцессоре.

И наконец, наиболее эффективно используется многонитевость для выполнения распределенных приложений, например, многонитевый сервер параллельно выполняет запросы сразу нескольких клиентов...

Управление оперативной памятью

Обычно сама ОС располагается в самых младших адресах ОП. 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 недостаточны 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.

Типы адресации ОП

Существуют три аспекта адресации ОП (рис. 12): символьные имена (переменных), виртуальные адреса и физические адреса.

Символьные имена (идентификаторы переменных) присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

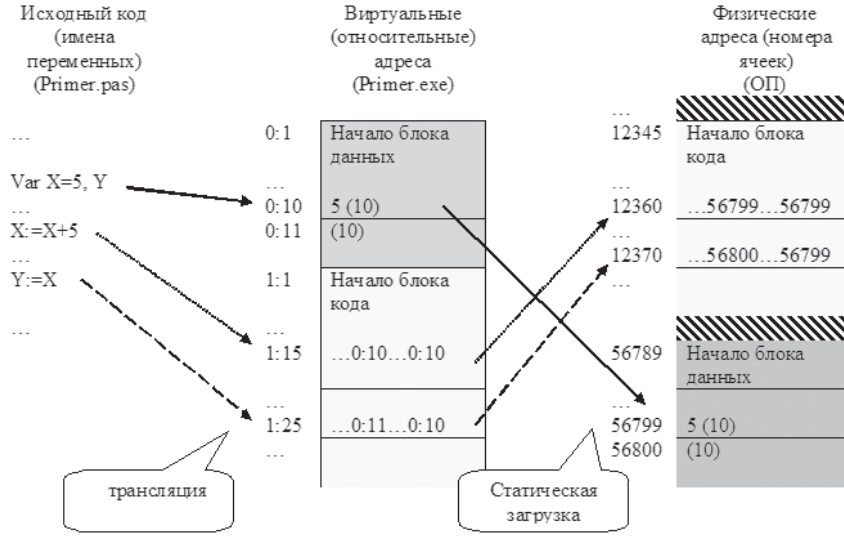

Рис. 12. Три аспекта адресации ОП

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Поскольку в ходе трансляции в общем случае неизвестно, в какое место ОП будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные, мнимые) адреса, предполагая, что программа будет размещена, начиная с некоторого нулевого адреса. Совокупность виртуальных адресов процесса называется его виртуальным адресным пространством. Каждый процесс имеет собственное виртуальное адресное пространство. Максимальный размер виртуального адресного пространства ограничивается разрядностью адреса, присущей данной архитектуре компьютера (значительно превышает объем имеющейся физической памяти).

Физические адреса соответствуют номерам ячеек ОП, где в действительности располагаются значения переменных и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами: статически (перемещающим **загрузчиком***)* или динамически в ходе выполнения программы.

Перемещающий загрузчик на основании исходных данных о выделенном процессу начальном адресе физической памяти и информации транслятора об адресно-зависимых командах выполняет загрузку программы в ОП, совмещая ее с заменой виртуальных адресов в командах физическими. Такой подход затрудняет реализацию виртуальной памяти.

При втором способе программа загружается в ОП с виртуальными адресами в командах. Во время выполнения программы при каждом обращении команды к ОП производится предварительное преобразование виртуального адреса, содержащегося в команде в физический.

Методы распределения ОП без использования внешней

Все методы управления ОП могут быть разделены на два класса:

не использующие перемещение процессов между ОП и внешней:

- фиксированными разделами,

- динамическими разделами,
- перемещаемыми разделами,

использующие перемещение процессов между ОП и внешней:

- страничное,
- сегментное,
- сегментно-страничное,
- свопинг.

Распределение памяти фиксированными разделами

Простейшим способом управления ОП является исходное разделение ее на несколько разделов фиксированной величины. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рис. 13а), либо в очередь к некоторому разделу (рис. 13б).

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 очевидной простоте реализации данный метод совершенно не гибок, при этом уровень мультипрограммирования заранее ограничен числом разделов независимо от размера кода программы. Любая программа будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объем ОП ЭВМ позволяет выполнить некоторую большую программу, разбиение памяти на разделы не позволит сделать это.

Распределение памяти динамическими разделами (переменной величины)

ОП машины исходно не делится на разделы. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и ожидает загрузки в очереди. После завершения задачи память освобождается, и на ее место может быть загружена другая задача. Таким образом, в произвольный момент ОП представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

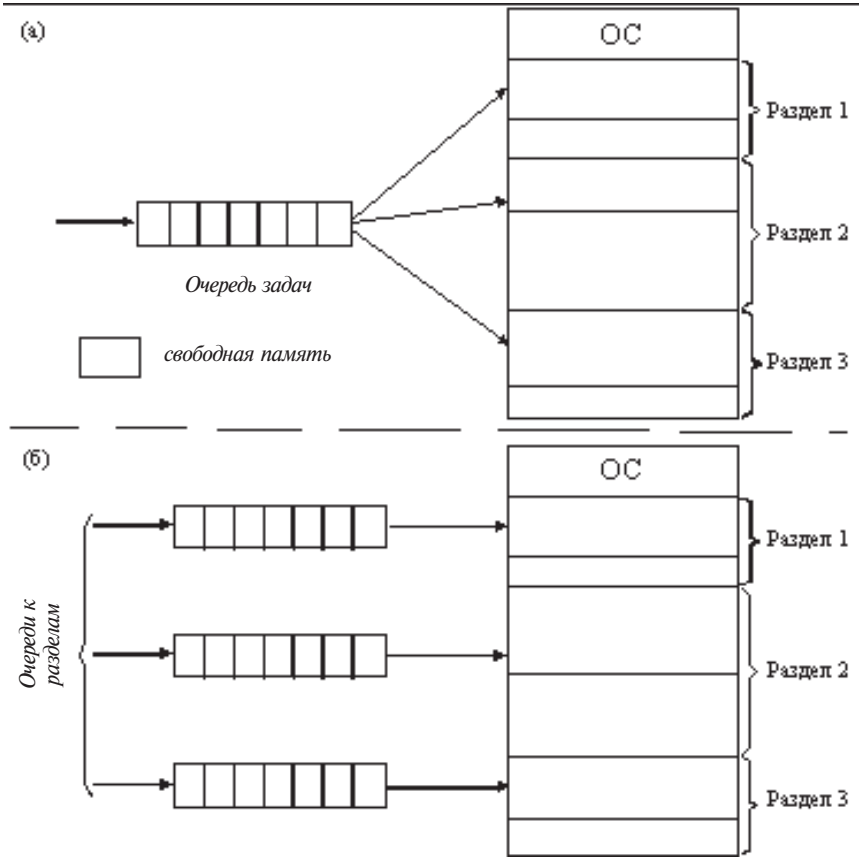

Рис. 13. Распределение ОП фиксированными разделами

На рис. 14 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так, в момент t_0 в памяти находится только ОС, а к моменту t_1 память разделена между пятью задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t_3.

Задачами ОС при реализации данного метода управления ОП являются:

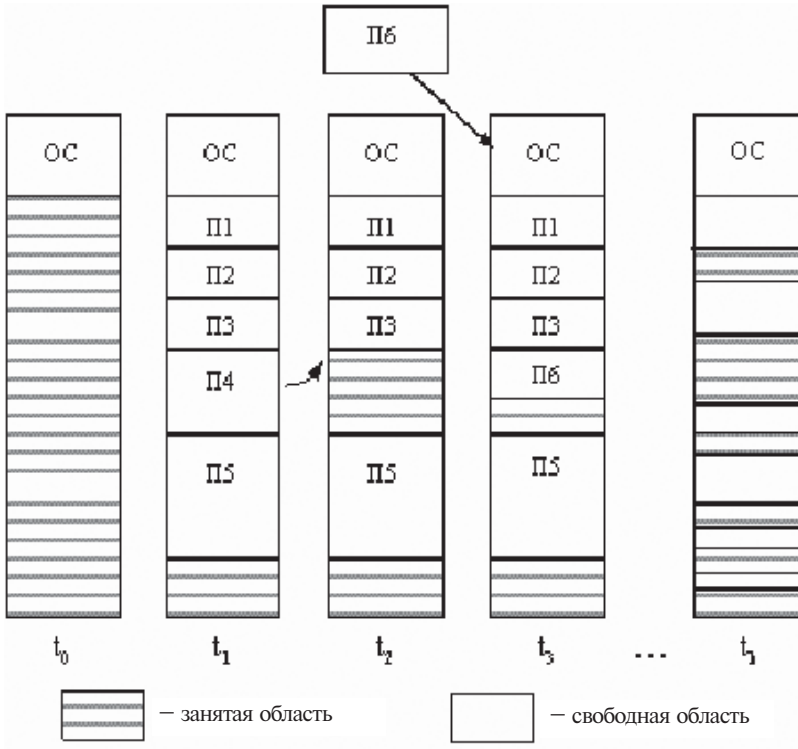

Рис. 14. Распределение ОП динамическими разделами (переменной величины)

— 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 (например, «первый попавшийся раздел достаточного размера», или «раздел, имеющий наименьший достаточный размер», или «раздел, имеющий наибольший достаточный размер»).

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но характеризуется фрагментацией памяти. **Фрагментация** — это наличие большого числа несмежных участков свободной памяти маленького размера (фрагментов), недоступных вследствие этого поступающим на выполнение программам, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Перемещаемые разделы

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 (рис. 15). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени перемещать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Указанная 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 ОП 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 процессорного времени, что уменьшает преимущества данного метода.

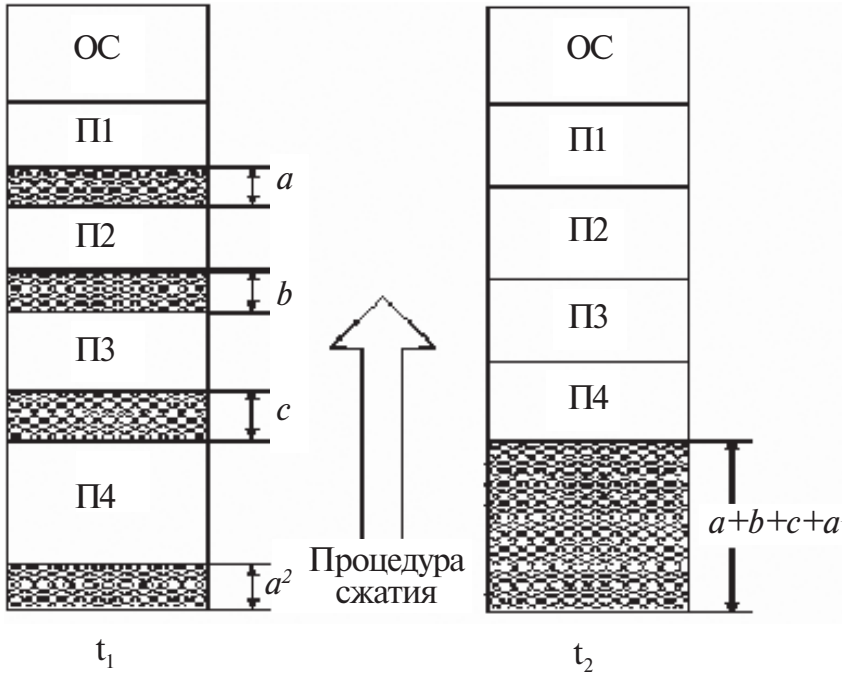

Рис. 15. Распределение ОП перемещаемыми разделами

Методы распределения ОП с использованием внешней памяти

Уже достаточно давно пользователи столкнулись с проблемой размещения в памяти программ, размер которых превышал имеющуюся в наличии свободную память. Одним из решений было разбиение программы на части, называемые оверлеями. 0-й оверлей начинал выполняться первым. Когда он заканчивал свое выполнение, он вызывал другой оверлей. Все оверлеи хранились на диске и перемещались между памятью и диском средствами операционной системы. Однако разбиение программы на части и планирование их загрузки в ОП должен был осуществлять программист.

Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием виртуальная память.

Виртуальным (условным, мнимым) называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, отсутствующими в действительности (например, пользователю может быть предоставлена виртуальная ОП, размер которой превосходит всю имеющуюся в системе физическую ОП).

Виртуальная память — это программно-аппаратная технология, позволяющая пользователям выполнять программы, размер которых превосходит имеющуюся физическую ОП. Для этого виртуальная память решает следующие задачи:

— размещает программы в запоминающих устройствах разного типа: одна часть — в ОП, а другая — во внешней;
— перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы из внешней в оперативную память (и наоборот);
— преобразует виртуальные адреса в физические.

Наиболее распространенными реализациями виртуальной памяти являются: страничное, сегментное и страично-сегментное распределение памяти, а также свопинг.

Страничное распределение

На рис. 16 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 ОП машины также делится на части такого же размера, называемые физическими страницами (или блоками), с размером равным степени двойки: 512, 1024 и т.д., что позволяет упростить механизм преобразования адресов.

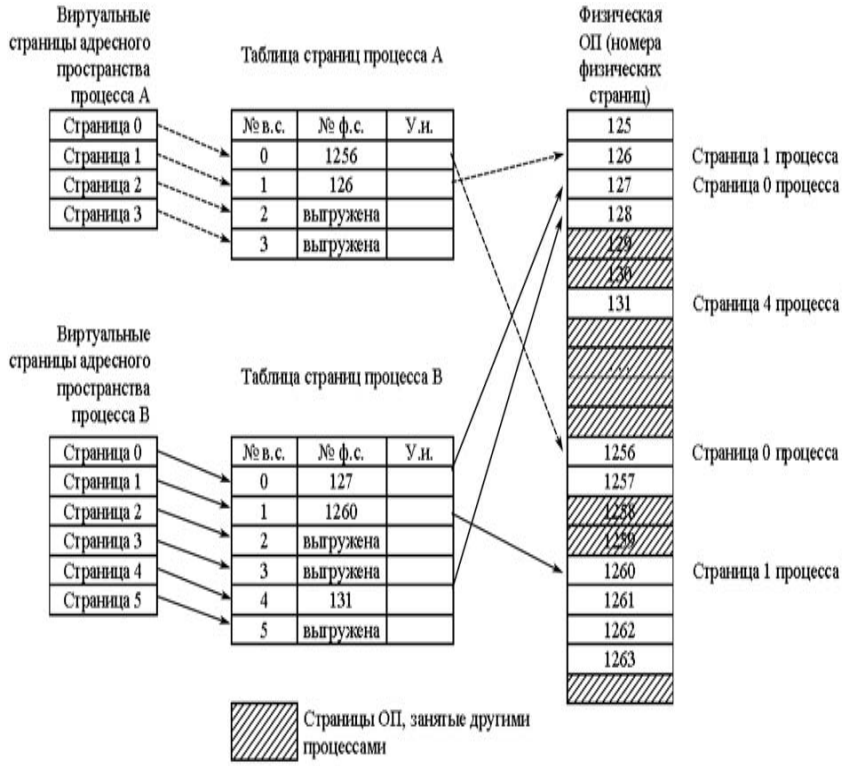

Рис. 16. Страничное распределение ОП

При загрузке процесса часть его виртуальных страниц помещается в ОП, а остальные — на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке ОС создает для каждого процесса информационную структуру — **таблицу страниц**, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в ОП, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация — признак модификации страницы, признак невы-

гружаемости (выгрузка 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 ОП, 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 О, и активизируется другой процесс из очереди Г. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в ОП. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из ОП (дольше всего не использовавшаяся страница, первая попавшаяся страница, редко использовавшаяся страница).

После того как выбрана страница, которая должна покинуть ОП, 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.

Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти (рис. 17).

Виртуальный адрес при страничном распределении может быть представлен в виде пары (p, s), где p — номер виртуальной страницы процесса (нумерация страниц начинается с 0), а s — смещение в пределах виртуальной страницы. Учитывая, что размер страницы равен 2 в степени k, смещение s может быть получено простым отделением k младших разрядов в двоичной запи-

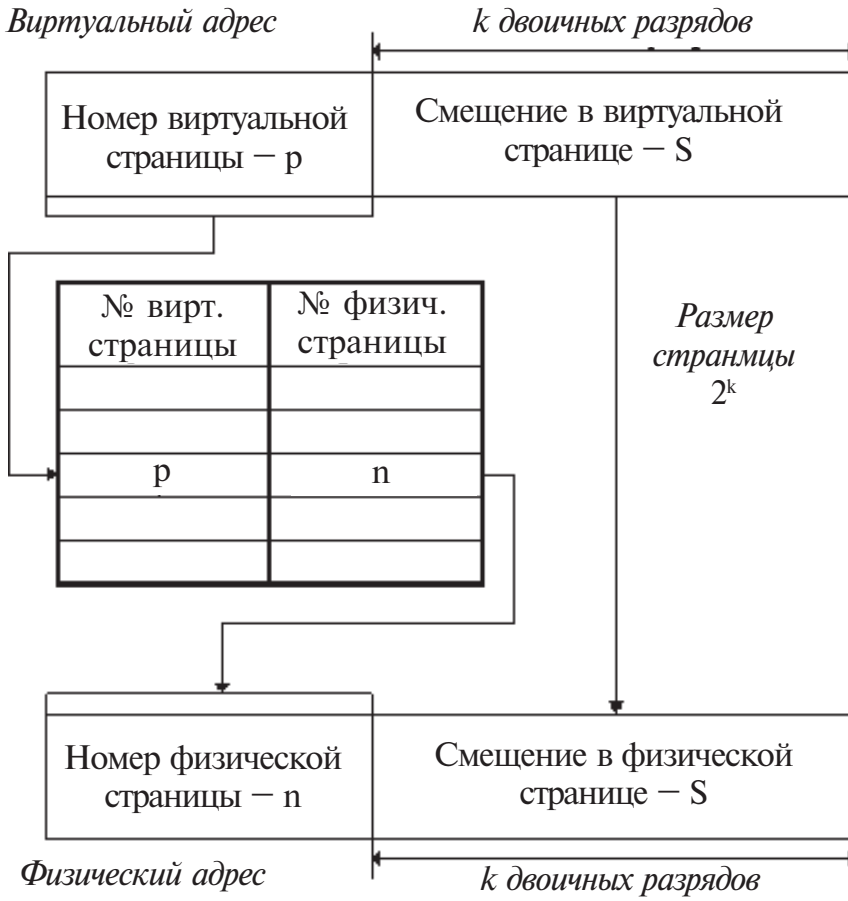

Рис. 17. Схема преобразования виртуального адреса в физический при страничной организации памяти

си виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы *p*.

При каждом обращении к ОП аппаратными средствами выполняются следующие действия:

- на основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адре-

са) 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 третьем пункте 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 ОП на ее хранение. 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 (кэширование данных).

Страничное распределение памяти может быть реализовано без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в ОП. Такой вариант страничной организации исключает страничные прерывания.

Сегментное распределение

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это затрудняет дифференцирование способов доступа к разным частям программы (сегментам), что 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 ОП может быть загружена только одна копия этой подпрограммы (**реентерабельность** кода).

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности (рис. 18). 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 ОП (для каждого из них ОС подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в ОП несмежные участки. Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и 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 ОП, 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 ОП 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 являются фрагментация ОП и медленное по сравнению со страничной организацией преобразование адреса.

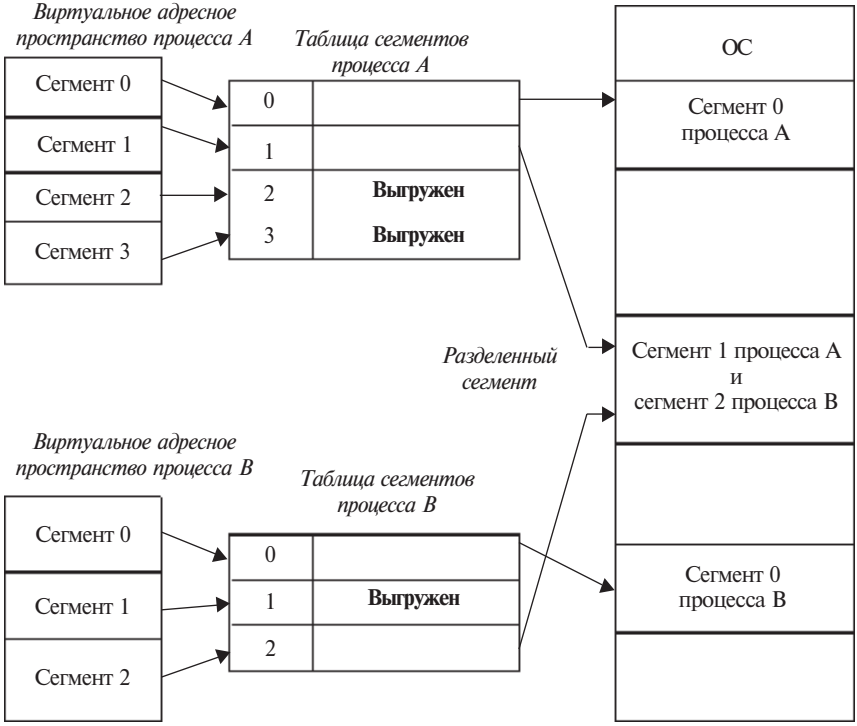

Рис. 18. Сегментное распределение ОП

Сегментно-страничное распределение

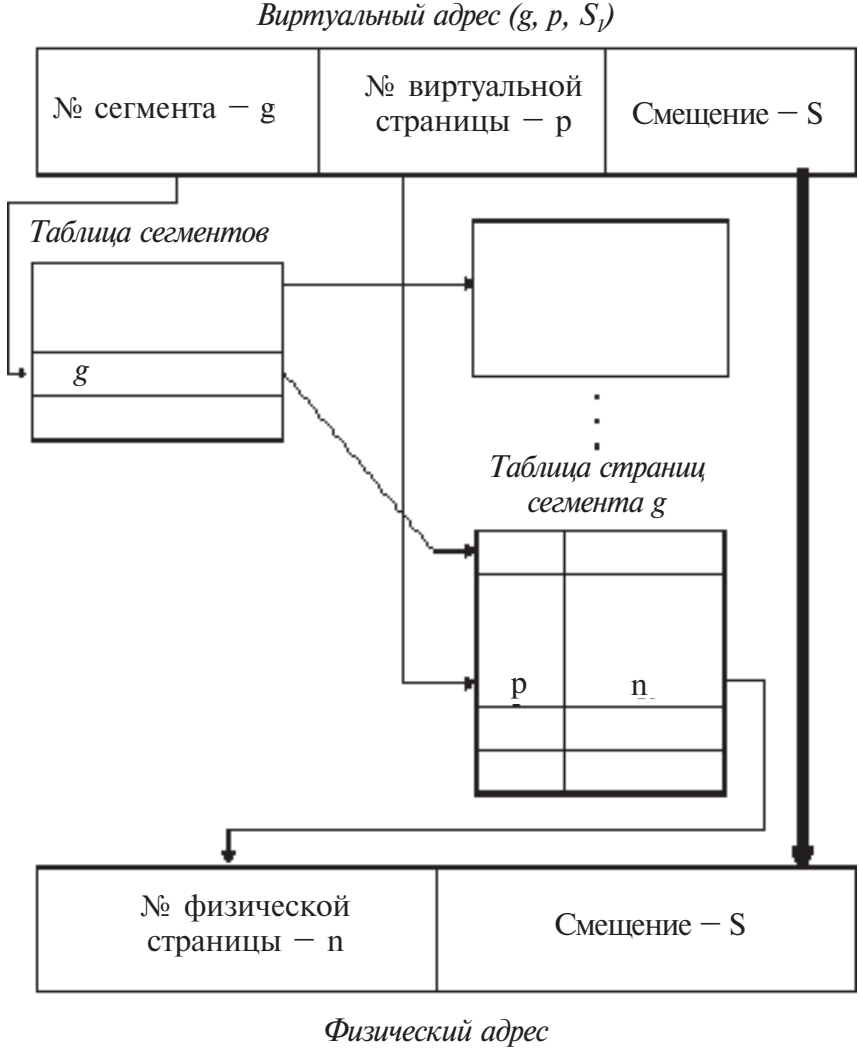

Рис. 19. Сегментно-страничное распределение ОП

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. ОП делится на физические страницы.

Загрузка процесса выполняется ОС постранично, при этом часть страниц размещается в ОП, а часть на диске. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс. На рис. 19 показана схема преобразования виртуального адреса в физический для данного метода.

Свопинг

Разновидностью виртуальной памяти является свопинг.

На рис. 20 показан график зависимости коэффициента загрузки процессора в зависимости от числа одновременно выполняемых процессов и доли времени, проводимого этими процессами в состоянии ожидания ввода/вывода.

Из рисунка видно, что для загрузки процессора на 90% достаточно всего трех счетных задач. Однако для того, чтобы обеспечить такую же загрузку интерактивными задачами, выполняющими интенсивный ввод/вывод, потребуются десятки таких задач.

В последнем случае доля процессов в состоянии **О** велика, поэтому некоторые из них с целью экономии ОП временно выгружаются на диск. Планировщик ОС не исключает их из своего рассмотрения, и при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, такой процесс перемещается в ОП. Если свободного места в ОП не хватает, то выгружается другой процесс.

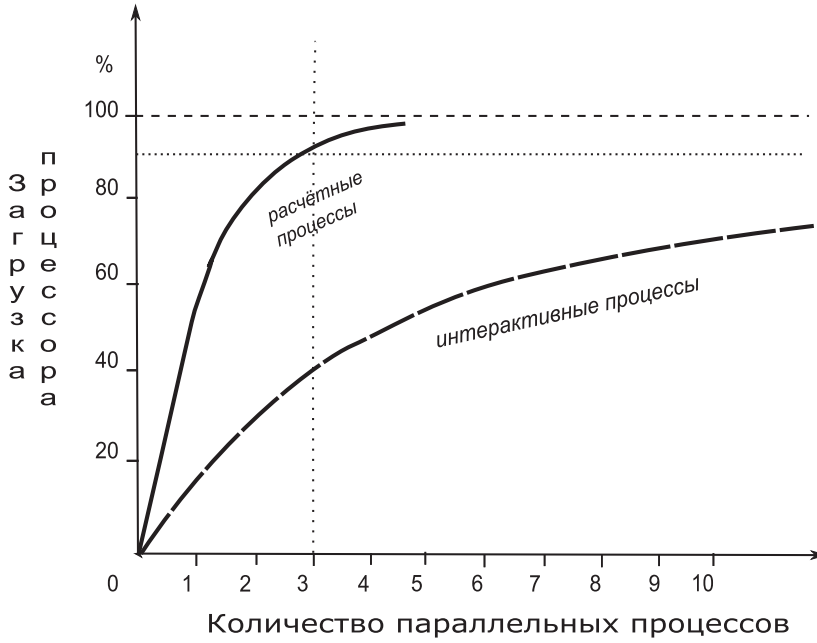

Рис. 20. Свопинг

При свопинге, в отличие от рассмотренных ранее методов реализации виртуальной памяти, процесс перемещается между памятью и диском целиком, то есть в течение некоторого времени процесс может полностью отсутствовать в ОП. Существуют различные алгоритмы выбора процессов на загрузку и выгрузку, а также различные способы выделения оперативной и дисковой памяти загружаемому процессу.

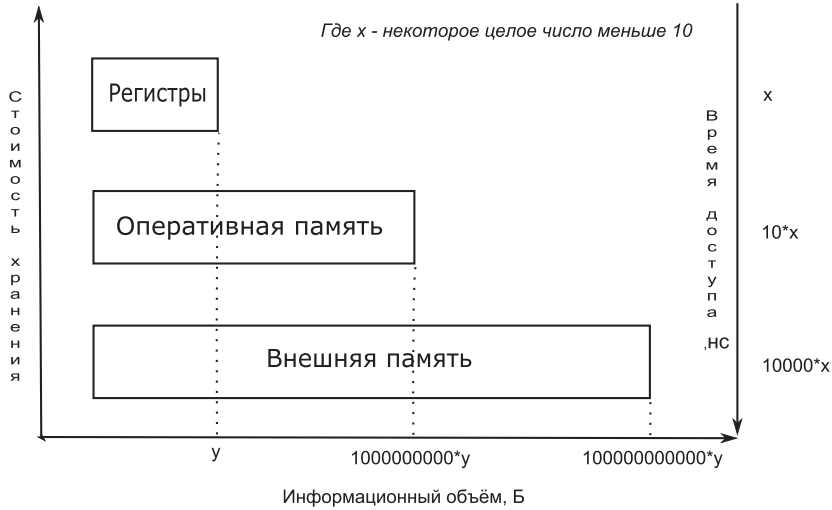

Рис. 21. Иерархия запоминающих устройств

Память ЭВМ представляет собой иерархию запоминающих устройств (ЗУ: внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит (рис. 21). Пользователю хотелось бы иметь и недорогую, и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Кэш-память — это программно-аппаратная технология сопряжения двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, которая позволяет уменьшить среднее время доступа к данным за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ.

Часто кэш-памятью называют также устройство или блок элементов «быстрого» ЗУ, которое играет роль промежуточной памяти. Оно стоит дороже «медленного» ЗУ и, как правило, имеет сравнительно небольшой объем.

Механизм кэш-памяти является прозрачным для программ, использующих его, и поддерживается автоматически системными средствами.

Рассмотрим частный случай использования кэш-памяти между регистром процессора и ОП (рис. 22).

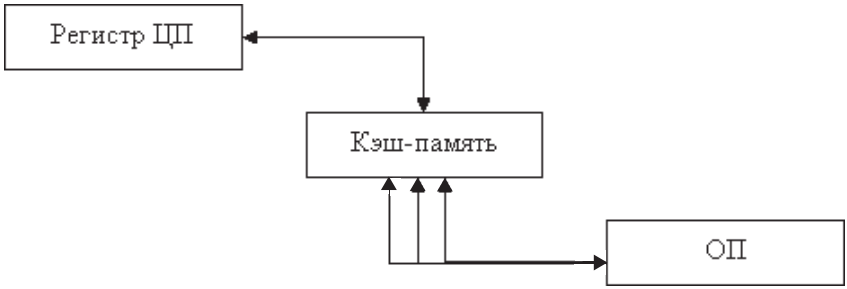

Рис. 22 Связь регистра ЦП с ОП через кэш

В кэше временно хранится копия некоторого фрагмента информации из ОП. Когда процессору требуется получить в регистр очередную команду или данные из ОП, сначала проверяется содержимое кэша. Если нужная информация там есть (кэш-попадание), она передается в регистр. В противном случае (кэш-промах) из ОП в кэш считывается блок данных фиксированной длины, в составе которого имеется требуемая информация. Отношение количества кэш-попаданий к общему количеству операций заполнения регистра определяет эффективность кэш-памяти.

Рассмотрим подробнее возможную структуру организации кэш и взаимодействие с ОП (рис. 23).

ОП представляет собой последовательность относительно адресуемых байт, которая может быть виртуально разбита на M блоков по N байт для сопоставления с содержимым кэша. Адрес байта ОП условно разбит на две части: номер блока и номер в пределах блока. Кэш-память структурно состоит из K строк длиной N байт (причем, K<<M). Номер блока ОП, байты которого хранятся в строке кэша, записывается в поле — тэг. Каждая строка включает в себя также поле управляющей информации

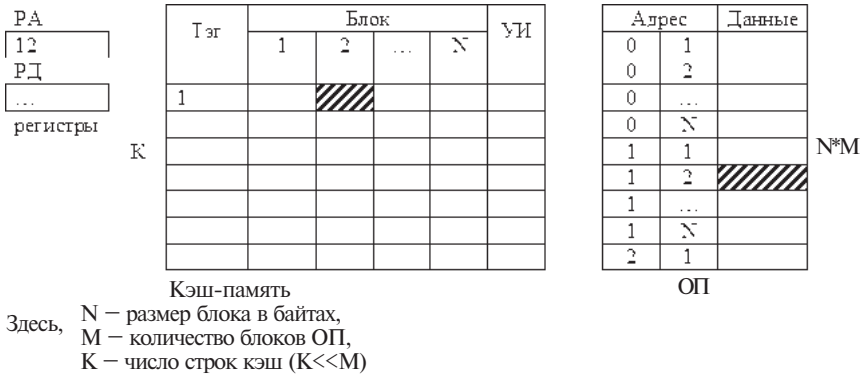

Рис. 23. Информационная структура кэш и ОП

(УИ: признак модификации и признак обращения к данным за некоторый последний период времени), необходимой для работы кэша.

В процессе выполнения программы часть ее блоков находится в строках кэша. Если требуется передать в кэш байт из ОП, в свободную строку копируется соответствующий блок. Если свободных строк нет, выполняется частичная очистка кэша.

В системах, оснащенных кэш-памятью, каждый запрос к ОП выполняется в соответствии со следующим алгоритмом:

— Просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти. Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — значению поля адреса в ОП (тэг + номер байта в строке кэша), взятому из запроса (содержимое адресного регистра).

— Если данные обнаруживаются в кэш-памяти, то они считываются из нее в регистр данных ЦП.

— Если нужных данных нет, то они вместе со соответствующим блоком копируются из ОП в строку кэш-памяти, и результат выполнения запроса передается в процессор. При копировании данных может оказаться, что в кэш-памяти нет свободного места, тогда выбираются строки, к которым в последний период было меньше всего обращений, для вытеснения из кэш-

памяти. Если вытесняемые данные были модифицированы за время нахождения в кэш-памяти, то они переписываются в ОП. Если же эти данные не были модифицированы, то их место в кэш-памяти объявляется свободным.

В реальных системах вероятность попадания в кэш составляет примерно 90%, что связано с наличием у данных в ОП объективных свойств: пространственно-временной локальности.

Принцип пространственно-временной локальности. Если произошло обращение по некоторому адресу выполняемой программы, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Рассмотренный выше механизм реализуется и для других пар ЗУ, например, для ОП и внешней памяти. В этом случае уменьшается среднее время доступа к данным, расположенным на диске, и роль кэш-памяти выполняет буфер (блок ячеек) в ОП.

Управление внешними устройствами

Следующей важнейшей функцией ОС является управление всеми устройствами ввода/вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки, а также обеспечивать интерфейс между устройствами и процессами. Указанный интерфейс со стороны процессов должен быть одинаковым для всех типов устройств (независимость от устройств).

Структура устройств ввода/вывода

Устройства ввода/вывода (внешние, ВУ) делятся на два типа: блок-ориентированные и байт-ориентированные устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, имеющих свой собственный адрес (диски). Байт-ориентированные устройства не адресуемы, не позволяют производить операцию поиска, генерируют или потребляют последовательность байтов (терминалы, строчные принтеры, сетевые адаптеры...). Отдельные внешние устройства могут не подходить под указанную классификацию (например,

часы не адресуемы, но и не порождают потока байтов, просто выдают сигнал прерывания в заданные моменты времени).

ВУ обычно состоит из управляющего компонента (контроллера) и исполнительного (адаптера). Иногда, **контроллеры** могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать так называемые совместимые контроллеры и устройства.

ОС обычно работает с контроллером. Последний представляет собой микропроцессор, выполняющий простейшие функции обработки информации: преобразует поток бит в блоки, состоящие из байт, осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров (быстрая память), которые используются для взаимодействия с ЦП. В некоторых компьютерах эти регистры являются частью физического адресного пространства ЦП. В таких компьютерах нет специальных операций ввода/вывода. В других компьютерах адреса регистров ввода/вывода, называемые **портами**, образуют собственное адресное пространство за счет введения специальных операций ввода/вывода (например, команд IN и OUT в процессорах i*86).

ОС выполняет ввод/вывод, записывая команды в регистры контроллера. Например, контроллер гибкого диска IBM PC принимает 15 команд, таких как READ, WRITE, SEEK, FORMAT и т.д. Когда команда принята контроллером, процессор занимается другой работой. При завершении команды контроллер организует прерывание для передачи управления ОС, которая должна проверить результаты операции. После чего процессор получает результаты и статус устройства ввода/вывода, читая информацию из регистров контроллера.

Организация ПО ввода/вывода

Основная идея организации ПО ввода/вывода состоит в разбиении его на несколько уровней: нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те,

в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом ПО является его независимость от устройств. Очень близкой к идее независимости от устройств является идея единообразного именования, то есть для именования устройств должны быть приняты единые правила.

Другим важным вопросом для ПО ввода/вывода является обработка ошибок. Ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода/вывода (ошибки, вызванные наличием пылинок на головках чтения или на диске). И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Следующий ключевой вопрос — использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода/вывода выполняется **асинхронно** — процессор начинает передачу и переходит на другую работу, пока не наступает прерывание.

Пользовательские программы намного легче писать, если операции ввода/вывода блокирующие. После команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. Однако ОС выполняет операции ввода/вывода асинхронно, представляя их для пользовательских программ в синхронной форме.

Последняя особенность состоит в том, что одни устройства являются **разделяемыми** для пользователей, а другие — **выделенными**. Разделяемые устройства (диски) легко обеспечивают одновременный доступ нескольких пользователей. Выделенные устройства (принтеры) делают это «с трудом» (нельзя смешивать строчки различных пользовательских документов). Наличие выделенных устройств создает для ОС дополнительные сложности.

С точки зрения решения указанных проблем целесообразно разделить ПО ввода/вывода на четыре слоя (рис. 24):

— Обработка прерываний,
— Драйверы устройств,
— Независимый от устройств слой ОС,
— Прикладное ПО.

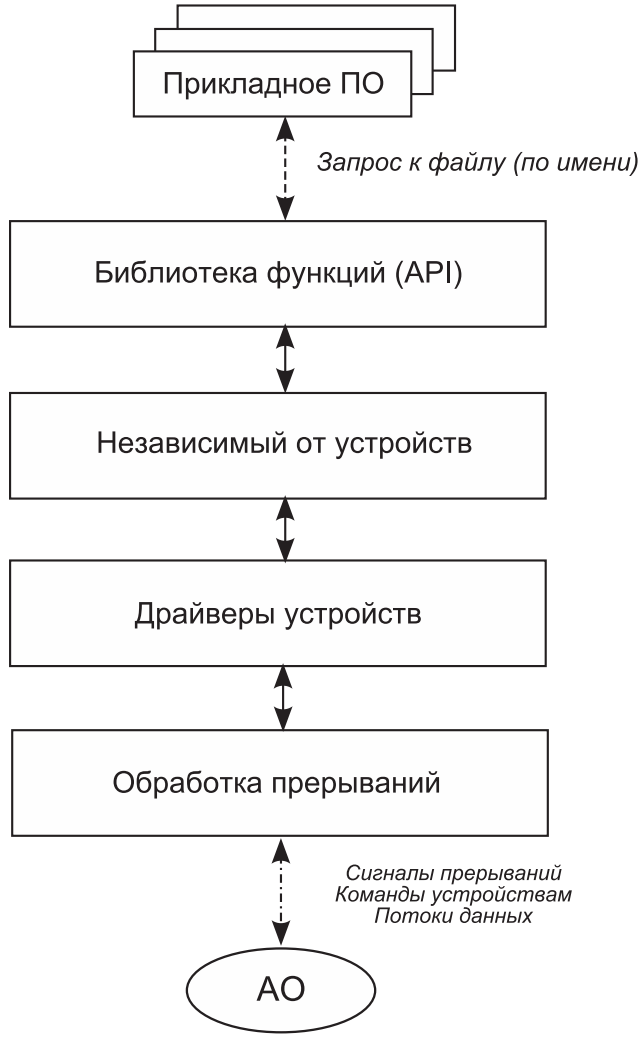

Рис. 24. Структура ПО ввода/вывода

Обработка прерываний. Прерывания должны быть скрыты «как можно глубже в недрах» ОС. Наилучший способ состоит в разрешении процессу, инициировавшему операцию ввода/вывода, блокировать себя до завершения операции и наступления прерывания (вызов функции P для семафора, или вызов WAIT для блокирующей переменной). При генерации прерывания от устройства процедура его обработки выполняет разблокирование процесса, инициировавшего операцию ввода/вывода. После чего последний продолжит свое выполнение.

Драйверы устройств. Весь зависимый от устройства код помещается в драйвер устройства. Каждый драйвер управляет устройствами одного типа или, может быть, одного класса.

В ОС только соответствующий драйвер «знает» о конкретных особенностях конкретного устройства (драйвер диска имеет дело с дорожками, секторами, цилиндрами, временем установления головки и другими факторами, обеспечивающими правильную работу диска).

Драйвер устройства принимает запрос от устройств программного слоя и решает, как его выполнить. Типичным запросом является чтение N блоков данных. Если драйвер был свободен во время поступления запроса, он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди имеющихся запросов.

Первый шаг в реализации запроса ввода/вывода состоит в преобразовании его из абстрактной формы в конкретную. Например, для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром. То есть он должен решить, какие операции контроллера нужно выполнить и в какой последовательности.

После передачи команды контроллеру драйвер должен решить, блокировать ли себя до окончания заданной операции или нет. Если операция занимает значительное время, как при печати некоторого блока данных, то драйвер блокируется до завершения операции. Иначе (например, прокрутка экрана) драйвер ожидает ее завершения без блокирования.

Независимый от устройств слой ОС. Большая часть ПО ввода/вывода является независимой от устройств. Типичными функциями для независимого от устройств слоя ПО являются:

- обеспечение общего интерфейса к драйверам устройств,
- именование устройств,
- защита устройств,
- обеспечение независимого размера блока,
- буферизация,
- распределение памяти на блок-ориентированных устройствах,
- распределение и освобождение выделенных устройств,
- уведомление об ошибках.

Верхним слоям программного обеспечения неудобно работать с блоками разной величины, поэтому данный слой обеспечивает единый размер блока, например, за счет объединения нескольких различных блоков в единый логический блок. В связи с этим верхние уровни имеют дело с абстрактными устройствами, которые используют единый размер логического блока независимо от размера физического сектора.

При создании файла или заполнении его новыми данными необходимо выделить ему новые блоки. Для этого ОС должна вести список или «карту» свободных блоков диска. На основании информации о наличии свободного места на диске может быть разработан алгоритм поиска свободного блока, не зависимый от устройства и реализуемый программным слоем, находящимся выше слоя драйверов.

Пользовательский слой программного обеспечения. Хотя большая часть ПО ввода/вывода находится внутри ОС, некоторая его часть содержится в библиотеках, связываемых с пользовательскими программами (приложениями). Системные вызовы, включающие вызовы ввода/вывода, обычно делаются библиотечными процедурами.

Пример: Если программа, написанная на языке С, содержит вызов count = write (fd, buffer, nbytes), то библиотечная процедура write будет связана с программой. Набор подобных процедур является частью системы ввода/вывода. В частности, форматирова-

ние ввода или вывода выполняется библиотечными процедурами. Функция printf языка С, которая принимает строку формата и, возможно, некоторые переменные в качестве входной информации, затем строит строку символов ASCII и делает вызов write для вывода этой строки. Стандартная библиотека ввода/вывода содержит большое число процедур, которые выполняют ввод/вывод и работают как часть пользовательской программы.

Другой категорией ПО ввода/вывода является подсистема спулинга (spooling). *Спулинг* — совмещенное с выполнением главной цели чтение входного и запись выходного потоков данных на ВУ в форме удобной для последующей обработки — базовый способ работы с выделенными устройствами в мультипрограммной системе.

Пример: Печать на принтере. Хотя технически можно позволить каждому пользовательскому процессу открыть специальный файл, связанный с принтером, такой монопольный способ опасен своей неуправляемостью со стороны ОС. Поэтому система создает специальный процесс — монитор принтера, получающий исключительные права на использование этого устройства. Также создается специальный каталог, называемый каталогом спулинга. Для того чтобы напечатать файл, пользовательский процесс помещает выводимую информацию в этот файл и помещает его в каталог спулинга. Процесс-монитор по очереди распечатывает все файлы, содержащиеся в каталоге спулинга.

Логическая организация файла

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись — это наименьший элемент данных, которым может оперировать программист при обмене с ВУ. Даже если физический обмен с устройством осуществляется большими единицами, ОС обеспечивает программисту доступ к отдельной логической записи.

На рис. 25 показаны несколько схем логической организации файла. Записи могут быть фиксированной или переменной

Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс ≡ ключ

Рис. 25. Варианты логической организации файлов

длины, располагаться в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В большинстве файловых систем файл имеет простейшую логическую структуру — последовательность однобайтовых записей.

Физическая организация и адрес файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей — блоков. **Блок** (сектор) — наименьшая единица данных, которой внешнее устройство обменивается с ОП.

Непрерывное размещение — простейший вариант физической организации (рис. 26а), при котором файлу предоставляется последовательность блоков диска, образующих сплошной участок дисковой памяти. Достоинство этого метода — простота адресации (достаточно указать только номер начального блока). Недостатки: во-первых, во время создания файла заранее не известна его длина (сколько памяти надо зарезервировать для этого файла), во-вторых, при таком порядке размещения пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально один блок) могут остаться не используемыми.

Следующий способ физической организации — размещение в виде связанного списка блоков дисковой памяти (рис. 26б). При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом — номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно, фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла (для чтения пятого по порядку блока файла необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков). Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

Другим достаточно популярным способом (используемым в файловой системе FAT ОС MS-DOS) является использование

связанного списка индексов (рис. 26в). С каждым блоком связывается некоторый элемент — индекс. Индексы располагаются в отдельной (системной) области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа и снимаются недостатки: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество

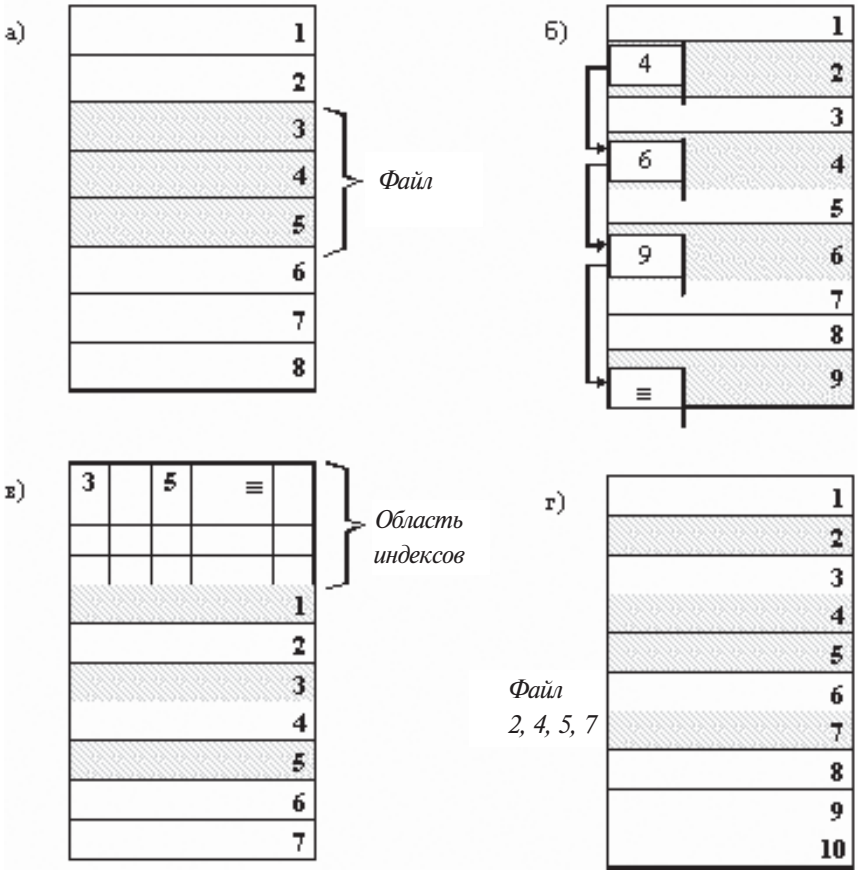

Рис. 26. Варианты физической организации файлов

блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит, имеют объем, равный степени двойки.

В заключение рассмотрим указание физического расположения файла способом простого перечисления номеров блоков, занимаемых этим файлом (рис. 26г). ОС UNIX использует вариант данного способа, позволяющий обеспечить фиксированную длину адреса, независимо от размера файла. Для хранения адреса файла выделено 13 полей индексного дескриптора. Если размер файла меньше или равен 10 блокам, то номера этих блоков непосредственно перечислены в первых десяти полях адреса. Если размер файла больше 10 блоков, то следующее 11-е поле содержит адрес блока, в котором могут быть расположены еще 128 номеров следующих блоков файла. Если файл больше, чем 10 + 128 блоков, то используется 12-е поле, в котором находится номер блока, содержащего 128 номеров блоков, которые содержат по 128 номеров блоков данного файла. И наконец, если файл больше 10 + 128 + 128*128, то используется последнее 13-е поле для тройной косвенной адресации, что позволяет задать адрес файла, имеющего размер максимум 10 + 128 + 128*128+128*128*128.

Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рис. 27), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, MS-DOS), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является уникальным и для ОС. В других файловых системах, в которых один и тот же файл может иметь несколько символь-

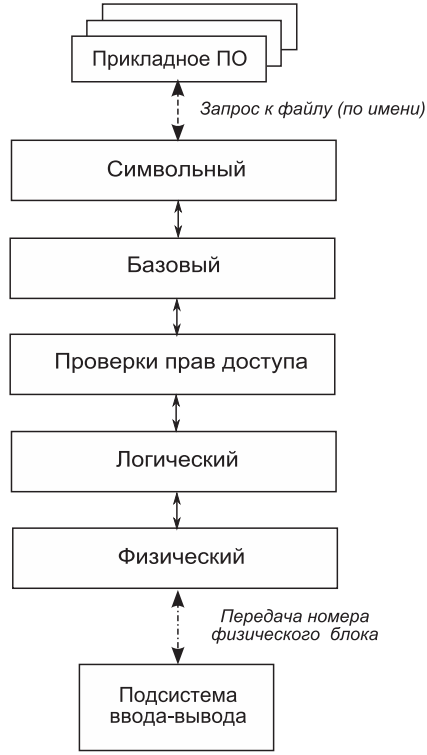

Рис. 27. Многоуровневая модель файловой системы

ных имен, на данном уровне просматривается системная информация для определения уникального имени файла. В файловой системе UNIX System 5, например, уникальным именем является номер индексного дескриптора файла (i-node).

На следующем, базовом уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в ОП для уменьшения среднего времени доступа к файлу. В некоторых файловых системах (например, HPFS, NTFS) при открытии файла вместе с его характеристиками в ОП перемещаются несколько первых блоков файла, содержащих данные.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, в противном случае выдается сообщение о нарушении прав доступа.

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая запись. Независимо от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла.

Пример: Если файл организован как последовательность логических записей фиксированной длины l, n-я логическая запись имеет смещение от начала файла l(n−1) байт. Для определения координат логической записи в файле с индексно-последовательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.

На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня — смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока. Рис. 28 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков (задача физического уровня решается независимо от того, как был логически организован файл).

Пусть V — размер физического блока, N — номер первого блока файла, S — смещение логической записи в файле. На физическом уровне: n — номер блока, содержащего требуемую логическую запись и s — смещение логической записи в пределах физического блока определяются по следующим формулам:

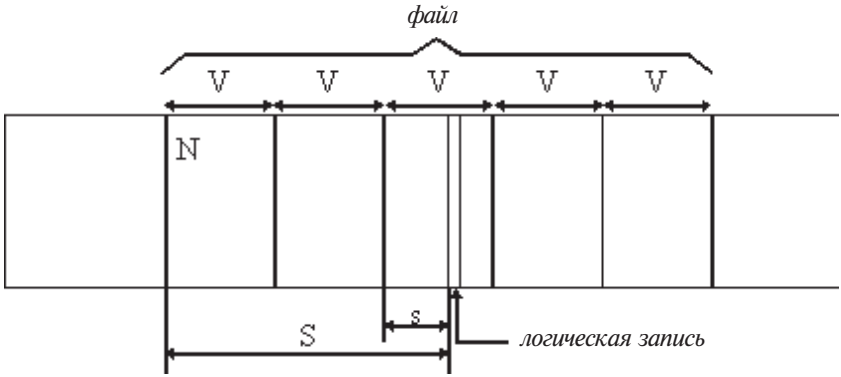

Рис. 28. Схема определения физического адреса в последовательном файле по логическому

n = N + [S/V], где [S/V] — целая часть числа S/V;
s = R [S/V] — дробная часть числа S/V.

После определения номера физического блока, файловая система обращается к системе ввода/вывода для выполнения операции обмена с внешним устройством. В ответ на запрос в буфер файловой системы будет записан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

Современная архитектура файловой системы

Разработчики новых ОС стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В новом понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Она имеет многоуровневую структуру (рис. 29), на верхнем уровне которой располагается так называемый переключатель файловых систем (в Windows 95, например, такой переключатель называется устанавливаемым диспетчером файловой сис-

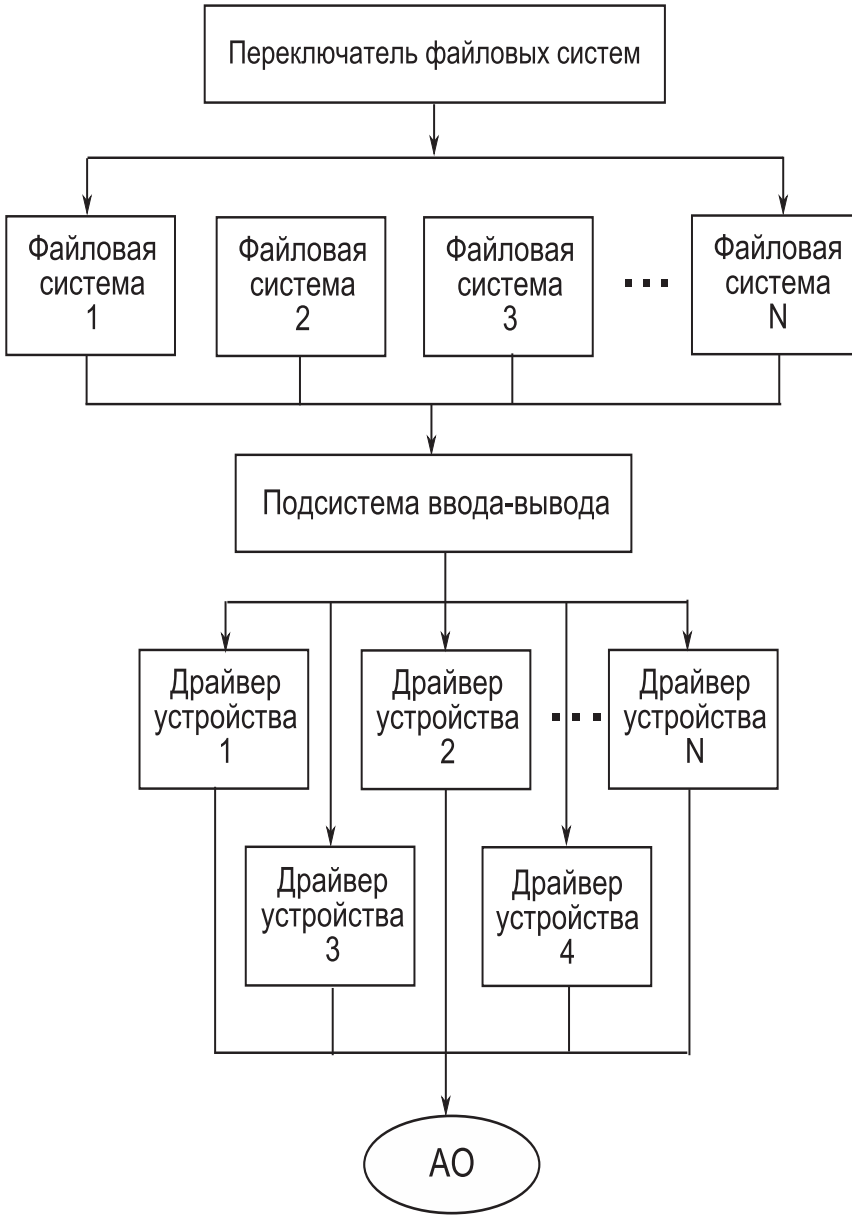

Рис. 29. Современная архитектура файловой системы

темы — installable filesystem manager, IFS). Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой обращается это приложение. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем — уровнем файловых систем.

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную организацию файловой системы. Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую. Драйвер файловой системы может быть написан в виде реентерабельного кода, что позволяет сразу нескольким приложениям выполнять операции с файлами этой системы. Каждый драйвер файловой системы в процессе собственной инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях к файловой системе.

Для выполнения своих функций драйверы файловых систем обращаются к подсистеме аппаратного ввода/вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода/вывода — это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы. Обычно эти модули представляют собой драйверы портов, которые непосредственно занимаются работой с аппаратными средствами. Кроме этого подсистема ввода/вывода обеспечивает некоторый сервис драйверам файловой системы, что позволяет им осуществлять запросы к конкретным устройствам. Подсистема ввода/вывода должна постоянно присутствовать в ОП и организовывать совместную работу иерархии драйверов устройств. В эту иерархию могут входить драйверы устройств определенного типа (драйверы жестких дисков или накопителей на лентах), драйверы, поддерживаемые поставщиками (такие драйверы перехватывают запросы к блочным устройствам и могут частично изменить поведение существующего драйвера этого устройства, например, зашифровать данные), а также драйверы портов, которые управляют конкретными адаптерами.

Примечание: Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость — драйвер может получить управление на любом этапе выполнения запроса — от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством цепочек вызова.

Вопросы и упражнения

1. Охарактеризуйте средства аппаратной поддержки многозадачности.
2. В чем различие прерывания и процедуры обработки прерывания?
3. Определите разницу между понятиями процесса и программы.
4. Опишите основные состояния процесса.
5. Почему системе необходимы дескрипторы и контексты процессов?
6. Какие алгоритмы планирования процессов предпочтительнее, основанные на квантовании или приоритетах?
7. Перечислите и охарактеризуйте средства межпроцессорного обмена информацией.
8. Приведите примеры ситуаций: гонка за ресурсом, тупик и голодание. Какие средства использует ОС для их ликвидации?
9. Что общего и чем различаются понятия «процесс» и «нить»?
10. Охарактеризуйте различные типы адресации ОП.
11. Чем ограничивается максимальный размер виртуального адресного пространства?
12. Может ли виртуальное адресное пространство процесса превысить физическую ОП?
13. Опишите алгоритмы распределения ОП без использования внешней памяти.

14. Опишите алгоритмы виртуальной памяти.
15. Что представляет собой структура и механизм функционирования кэш-памяти.
16. Охарактеризуйте организацию ПО ввода/вывода.
17. Приведите примеры реализаций различных схем логической и физической организации файлов.
18. Определите понятие «файловый блок».
19. Как согласуется общая модель файловой системы с современной архитектурой файловой системы?

СОВРЕМЕННЫЕ КОНЦЕПЦИИ И ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ОС

Требования, предъявляемые к современным ОС

Очевидно, что главным требованием является способность выполнения основных функций: эффективного управления ресурсами и обеспечения удобного интерфейса для пользователя и прикладных программ. Современная ОС, как правило, должна реализовывать мультипрограммную обработку, поддерживать виртуальную память, свопинг, многооконный интерфейс, а также выполнять другие необходимые функции. Кроме этих функциональных требований к ОС предъявляются не менее важные рыночные требования: **расширяемость** (возможность быстрого дополнения и изменения без нарушения целостности системы), **мобильность** (переносимость, поддержка процессоров и аппаратных платформ разного типа), **надежность**/отказоустойчивость (защита от внутренних и внешних ошибок, сбоев и отказов), **совместимость** (наличие средств для выполнения прикладных программ, написанных для других ОС), **безопасность** (наличие средств защиты ресурсов одних пользователей от других), **производительность** (высокое значение критерия эффективности).

Структурное построение (архитектура) ОС

Для удовлетворения требований, предъявляемых к современной ОС, большое значение имеет ее структурное построение. С точки зрения их архитектуры ОС прошли длительный путь развития от монолитных систем к хорошо структурированным модульным системам, способным к развитию, расширению и легкому переносу на новые платформы.

Обычно ОС состоит из двух компонентов: ядра и вспомогательных модулей.

Ядро ОС — компонент, выполняющий основные функции ОС по управлению ресурсами (обычно в привилегированном режиме процессора, режиме супервизора). **Вспомогательные компоненты ОС** (демоны — резидентные программы, драйверы периферийных устройств и утилиты), осуществляющие дополнительные сервисные функции, могут выполняться в режиме приложений (непривилегированном).

Ядро решает внутрисистемные задачи организации вычислительного процесса (переключение контекстов, загрузка/выгрузка станиц, обработка прерываний) и создает **прикладную программную среду** для поддержки запросов приложений (**системных вызовов** — для выполнения тех или иных действий, например для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т.д). Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования — API.

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями ОС, поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в ОП, то есть являются **резидентными**.

Ядро является движущей силой всех вычислительных процессов в компьютерной системе, и крах ядра равносилен краху всей системы. Поэтому разработчики ОС уделяют особое внимание надежности кодов ядра.

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

Остальные модули ОС выполняют полезные, но менее обязательные служебные функции (архивирование данных на магнитной ленте, дефрагментация диска...). Вспомогательные модули ОС оформляются либо в виде отдельных приложений, либо в виде библиотек процедур.

Часто служебная программа может существовать определенное время как пользовательское приложение, а потом стать частью ОС, или наоборот. Ярким примером такого изменения статуса программы является Web-браузер компании Microsoft, который сначала поставлялся как отдельное приложение, затем стал частью ОС Windows NT 4.0 и Windows 95/98.

Вспомогательные модули ОС обычно подразделяются на следующие группы:

- **утилиты** — программы, решающие отдельные задачи управления и сопровождения компьютерной системы (сжатие дисков, архивирование данных на магнитную ленту);
- **системные обрабатывающие программы —** драйверы периферийных устройств, не входящих в базовую конфигурацию компьютера компиляторы (принтеры, сканеры...), компоновщики, отладчики;
- **программы предоставления пользователю дополнительных услуг** — специальный вариант пользовательского интерфейса, калькулятор и даже игры;
- **библиотеки процедур** различного назначения, упрощающие разработку приложений, например библиотека математических функций, функций ввода/вывода и т.д.

Как и обычные приложения, для выполнения своих задач утилиты, обрабатывающие программы и библиотеки ОС, обращаются к функциям ядра посредством системных вызовов.

Разделение ОС на ядро и модули-приложения обеспечивает легкую расширяемость ОС. Чтобы добавить новую высокоуровневую функцию, достаточно разработать новое приложение, и при этом не требуется модифицировать ответственные функции, образующие ядро системы.

Как рассмотрено ранее (см. раздел Управление ОП), каждое приложение работает в своем **адресном пространстве**. Это свойство позволяет локализовать некорректно работающее приложение в собственной области памяти, так что его ошибки не оказывают влияния на остальные приложения и ОС. Распределение и контроль адресных пространств ОС осуществляет специальными командами процессора, недоступными для приложений, — **привилегированными**.

Между количеством уровней привилегий, реализуемых аппаратно, и количеством уровней привилегий, поддерживаемых ОС, нет прямого соответствия. Так, на базе четырех уровней, обеспечиваемых процессорами компании Intel, операционная система OS/2 строит трехуровневую систему привилегий, а ОС Windows NT, UNIX и некоторые другие ограничиваются двухуровневой системой.

На основе двух режимов привилегий процессора ОС может построить сложную систему индивидуальной защиты ресурсов, примером которой является типичная система защиты файлов и каталогов. Такая система позволяет задать для любого пользователя определенные права доступа к каждому из файлов и каталогов (см. раздел Надежность, защищенность и управление пользователями).

Повышение устойчивости ОС, обеспечиваемое переходом ядра в привилегированный режим, достигается за счет некоторого замедления при выполнении системных вызовов. Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению — переключение из привилегированного режима в пользовательский (рис. 30). Во всех типах процессоров переход на процедуру со сменой режима выполняется медленнее, чем вызов процедуры без смены режима.

Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической. Ее используют многие популярные ОС, в том числе многочисленные версии UNIX, VAX VMS, IBM OS/390, OS/2 и с определенными модификациями — Windows NT.

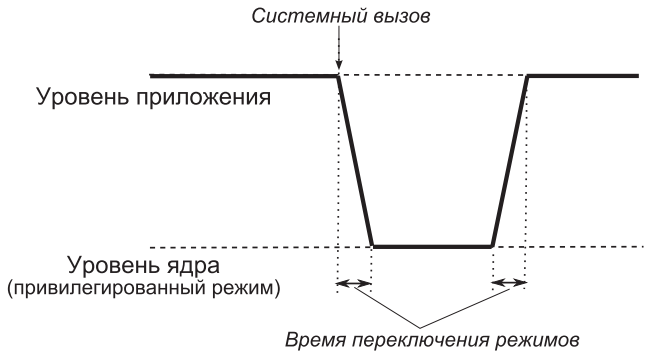

Рис. 30. Потери времени на переключение режимов

В некоторых случаях разработчики ОС отступают от этого классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, известная сетевая ОС NetWare компании Novell использует привилегированный режим процессоров Intel x86/ Pentium как для работы ядра, так и для работы своих специфических приложений — загружаемых модулей NLM. При таком построении ОС обращения приложений к ядру выполняются быстрее, так как нет переключения режимов, однако при этом отсутствует надежная аппаратная защита памяти, занимаемой модулями ОС, от некорректно работающего приложения. Разработчики NetWare пошли на такое потенциальное снижение надежности своей ОС, поскольку ограниченный набор ее специализированных приложений позволяет компенсировать этот архитектурный недостаток за счет тщательной отладки каждого приложения.

Обычно различают следующие архитектуры ОС (рис. 31).

Системы с монолитным ядром

Традиционная монолитная система, по определению, не имеет четкой структуры и пишется как набор процедур единой программы (рис. 31а). При использовании этой техники каждая процедура системы имеет определенный интерфейс (в терминах параметров и результатов) и может вызывать другие в случае необходимости.

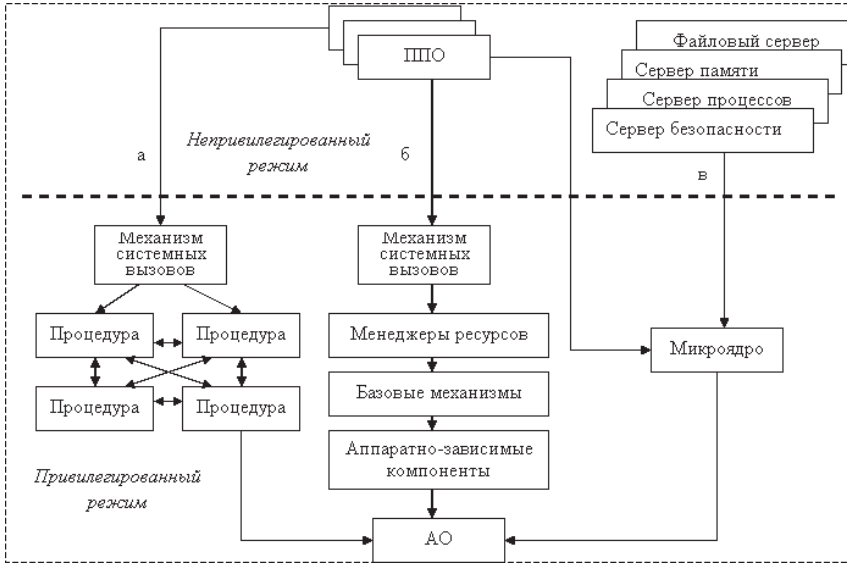

Рис. 31. Основные разновидности архитектур ОС

Для построения монолитной системы необходимо скомпилировать все отдельные процедуры, а затем связать их вместе в единый объектный файл с помощью компоновщика (примерами могут служить ранние версии UNIX или Novell NetWare). При этом каждая процедура «видит» любую другую процедуру в отличие от структуры, содержащей модули, где большая часть информации локальна для модуля.

Многоуровневые системы (слоистое ядро)

Даже монолитные системы имеют определенную функциональную структуру. При обращении к системным вызовам необходимые параметры помещаются в строго определенные места (регистры, стек...), затем выполняется специальная команда прерывания, которая переключает компьютер из режима пользователя в режим ядра (или супервизора) и передает управление ОС. ОС проверяет параметры вызова, определяет соответствующую процедуру обработки вызова.

Обобщением такого модульного подхода является организация ядра ОС как иерархии уровней, которые образуются группами функций ОС (файловая система, управление процессами и устройствами и т.п.). Каждый уровень может взаимодействовать только со своим непосредственным соседом (выше/нижележащим уровнем). Прикладные программы или модули самой ОС передают запросы вверх и вниз по этим уровням. При этом каждый уровень иерархии выполняет специфические функции, освобождая уровни, находящиеся выше по иерархии, от их выполнения, не занимаясь, с другой стороны, деталями функций, присущих нижним уровням.

Ядро может состоять из следующих слоев (рис. 31б).

– **Средства аппаратной поддержки ОС** — слой, прямо участвующий в организации вычислительных процессов: средства поддержки привилегированного режима, система прерываний, средства переключения контекстов процессов, средства защиты областей памяти и т. п.

– **Аппаратно-зависимые компоненты ОС —** программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры, что позволяет разрабатывать вышележащие слои на основе аппаратно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС. Примером экранирующего слоя может служить слой HAL ОС Windows NT.

– **Базовые механизмы ядра** — слой, который выполняет наиболее примитивные операции ядра, такие как программное переключение контекстов процессов, диспетчеризацию прерываний, перемещение страниц из памяти на диск и обратно и т. п. Модули данного слоя не принимают решений о распределении ресурсов — они только отрабатывают «принятые наверху решения».

– **Менеджеры ресурсов —** слой, состоящий из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Обычно на данном слое работают менеджеры (называемые так-

же диспетчерами) процессов, ввода/вывода, файловой системы и ОП. Для исполнения принятых решений менеджер обращается к нижележащему слою базовых механизмов. Внутри слоя менеджеров существуют тесные взаимные связи, отражающие тот факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам — процессору, области памяти, возможно, к определенному файлу или устройству ввода/вывода. Например, при создании процесса менеджер процессов обращается к менеджеру ОП, который должен выделить процессу определенную область памяти для его кодов и данных.

— **Интерфейс системных вызовов** — слой, являющийся самым верхним слоем ядра и взаимодействующий непосредственно с приложениями и системными утилитами. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Например, в ОС UNIX с помощью системного вызова fd = open(«/doc/a.txt», 0_RDONLY) приложение открывает файл a.txt, хранящийся в каталоге /doc, а с помощью системного вызова read(fd, buffer, count) читает из этого файла в область своего адресного пространства, имеющую имя buffer, некоторое количество байт. Для осуществления таких комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов, причем для выполнения одного системного вызова может понадобиться несколько таких обращений;

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними может быть иным. В системах, предназначенных для аппаратных платформ одного типа, например ОС NetWare, слой аппаратно-зависимых модулей обычно не выявляется, сливаясь со слоем базовых механизмов и частично со слоем менеджеров ресурсов. Не всегда оформляются в отдельный слой базовые механизмы — в этом случае менеджеры ресурсов не только планируют использование ресурсов, но и самостоятельно реализуют свои планы.

Возможна и противоположная картина, когда ядро состоит из большего количества слоев. Например, менеджеры ресурсов,

составляя определенный слой ядра, в свою очередь, могут обладать многослойной структурой. Прежде всего это относится к менеджеру ввода/вывода, нижний слой которого составляют драйверы устройств, например, драйвер жесткого диска или драйвер сетевого адаптера, а верхние слои — драйверы файловых систем или протоколов сетевых служб, имеющие дело с логической организацией информации (см. раздел Управление ВУ).

Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению работы ядра за счет дополнительных накладных расходов на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и логичность системы. Обычно ОС, прошедшие долгий путь эволюционного развития, например, многие версии UNIX, имеют неупорядоченное ядро с небольшим числом четко выделенных слоев, а у сравнительно «молодых» ОС, таких как Windows NT, ядро разделено на большее число слоев и их взаимодействие формализовано в гораздо большей степени.

Модель клиент-сервер и микроядра

Хотя многоуровневый подход на практике обычно работал неплохо, сегодня он все больше воспринимается монолитным. В системах, имеющих многоуровневую структуру, сложно удалить один слой и заменить его другим в силу множественности и размытости интерфейсов между слоями. На смену ему пришла модель клиент-сервер и тесно связанная с ней концепция микроядра (рис. 31в).

Модель клиент-сервер — это еще один подход к структурированию ОС. В широком смысле модель клиент-сервер предполагает наличие программного компонента — потребителя какого-либо сервиса — **клиента** и программного компонента — поставщика этого сервиса — **сервера**. Взаимодействие между клиентом и сервером стандартизуется, так что сервер может обслуживать клиентов, реализованных различными способами и, может быть, разными производителями. При этом главным требованием является то, чтобы они запрашивали услуги сервера понят-

ным ему способом. Инициатором обмена обычно является клиент, который посылает запрос на обслуживание серверу, находящемуся в состоянии ожидания запроса. Один и тот же программный компонент может быть клиентом по отношению к одному виду услуг, и сервером для другого вида услуг. Модель клиент-сервер является скорее удобным концептуальным средством четкого представления функций того или иного программного элемента в той или иной ситуации. Она успешно применяется также в разработке прикладного программного обеспечения.

Применительно к структурированию ОС идея состоит в разбиении ее на несколько процессов — серверов, каждый из которых выполняет отдельный набор сервисных функций: например, управление памятью, создание или планирование процессов. Каждый сервер выполняется в пользовательском режиме. Клиент, которым может быть либо другой компонент ОС, либо прикладная программа, запрашивает сервис, посылая сообщение на сервер. Ядро ОС, являющееся по сути микроядром, работая в привилегированном режиме, доставляет сообщение нужному серверу, сервер выполняет операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения (рис. 32).

Подход с использованием микроядра заменил вертикальное распределение функций ОС на горизонтальное. Компоненты, лежащие выше микроядра, хотя и используют сообщения, пересылаемые через микроядро, взаимодействуют друг с другом непосредственно. Микроядро играет роль регулировщика, который проверяет сообщения, пересылает их между серверами и клиентами и предоставляет доступ к аппаратуре.

Примером реализации рассмотренной структурной модели является Windows NT, в составе которой имеется подсистема (NT executive), работающая в режиме ядра и выполняющая функции обеспечения безопасности, ввода/вывода и другие.

Микроядро реализует жизненно важные функции, лежащие в основе ОС. Это базис для менее существенных системных служб и приложений. В общем случае подсистемы, бывшие традиционно неотъемлемыми частями ОС — файловые системы,

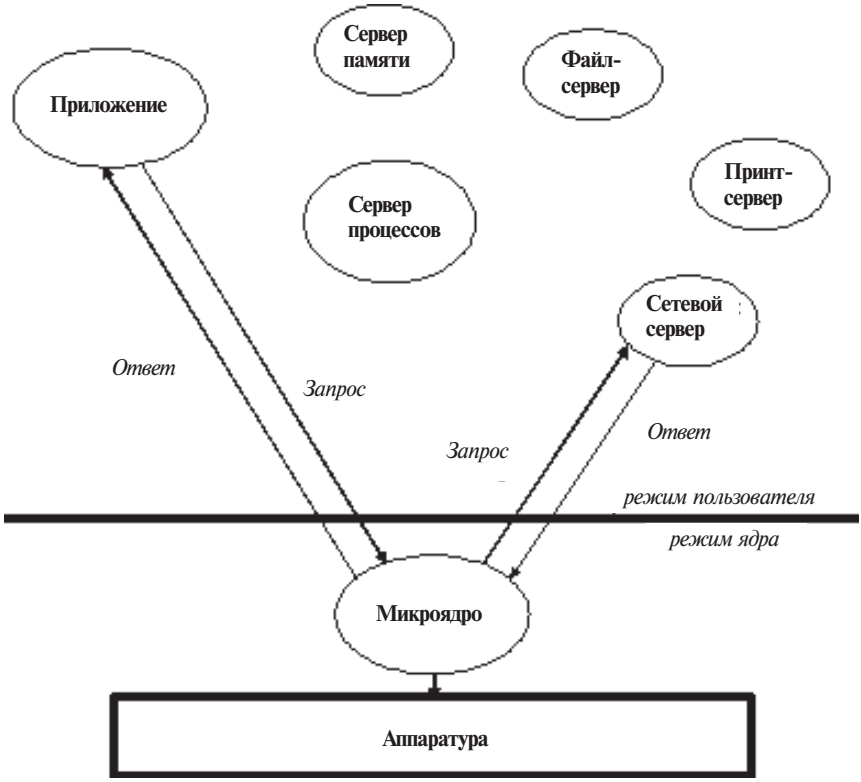

Рис. 32. Взаимодействие клиента с сервером «через» микроядро

управление окнами и обеспечение безопасности, — становятся периферийными модулями, взаимодействующими с ядром и друг с другом.

Главный принцип разделения работы между микроядром и окружающими его модулями — включать в микроядро только те функции, которым абсолютно необходимо исполняться в режиме супервизора. Под этим обычно подразумеваются аппаратно-зависимые программы (включая поддержку нескольких процессоров), некоторые функции управления процессами, обработка

прерываний, поддержка пересылки сообщений, некоторые функции управления устройствами ввода/вывода, связанные с загрузкой команд в регистры устройств.

В настоящее время именно ОС, построенные с использованием модели клиент-сервер и концепции микроядра, в наибольшей степени удовлетворяют современным требованиям.

Высокая степень переносимости обусловлена тем, что весь аппаратно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.

Технология микроядер является основой построения множественных прикладных сред, которые обеспечивают совместимость программ, написанных для разных ОС.

Ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС.

Использование модели клиент-сервер повышает надежность. Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти и таким образом защищен от других процессов. Более того, поскольку серверы выполняются в пространстве пользователя, они не имеют непосредственного доступа к аппаратуре и не могут модифицировать память, в которой хранится управляющая программа. И если отдельный сервер может потерпеть крах, то он может быть перезапущен без останова или повреждения остальной части ОС.

Эта модель хорошо подходит для распределенных вычислений, так как отдельные серверы могут работать на разных процессорах мультипроцессорного компьютера или даже на разных компьютерах.

Аппаратно-зависимые компоненты ОС

Одна ОС не может без каких-либо изменений устанавливаться на компьютерах, отличающихся типом процессора или/и способом организации всей аппаратуры (архитектура). В модулях ядра ОС всегда отражены особенности аппаратной платформы (количество типов прерываний и формат таблицы ссылок на процедуры обработки прерываний, состав регистров об-

щего назначения и системных регистров, состояние которых нужно сохранять в контексте процесса, особенности подключения внешних устройств...).

Однако ядро можно спроектировать частично аппаратно-зависимым. При этом аппаратно-зависимые модули локализованы и образуют программный слой, естественно примыкающий к слою аппаратуры. Такая локализация аппаратно-зависимых модулей существенно упрощает перенос ОС на другую аппаратную платформу.

Объем аппаратно-зависимых компонентов ОС зависит от того, насколько велики отличия в аппаратных платформах, для которых разрабатывается ОС. Например, несовпадение системы команд процессоров преодолевается компиляцией исходных текстов для конкретного типа процессора (соответствующим компилятором). В других случаях различия в аппаратуре компьютеров лежат гораздо глубже. Например, однопроцессорный и двухпроцессорный компьютеры требуют применения в ОС совершенно разных алгоритмов распределения процессорного времени. Отсутствие аппаратной поддержки виртуальной памяти приводит к принципиальному различию в реализации подсистемы управления ОП. ОС, построенная на 32-битовых адресах, для переноса на машину с 16-битовыми адресами должна быть практически переписана заново.

Для уменьшения количества аппаратно-зависимых модулей производители ОС обычно ограничивают универсальность аппаратно-независимых модулей. Это означает, что их независимость носит условный характер и распространяется только на несколько типов процессоров и созданных на их основе аппаратных платформ (Windows NT рассчитана на четыре типа процессоров и поставляется в различных вариантах кодов ядра для однопроцессорных и многопроцессорных компьютеров).

Особое место среди модулей ядра занимают низкоуровневые драйверы внешних устройств. Для компьютеров на основе процессоров Intel x86/Pentium разработка экранирующего аппаратно-зависимого слоя ОС несколько упрощается за счет встроенной в постоянную память компьютера базовой системы ввода/вывода — BIOS. BIOS содержит драйверы для всех устройств,

входящих в базовую конфигурацию компьютера (жестких и гибких дисков, клавиатуры, дисплея и т. д.). Эти драйверы выполняют примитивные операции с устройствами (чтение группы секторов данных с определенной дорожки диска), но за счет этих операций экранируются различия аппаратных платформ персональных компьютеров на процессорах Intel разных производителей.

Разработчики ОС могут пользоваться слоем драйверов BIOS как частью аппаратно-зависимого слоя ОС, а могут и заменить все или часть драйверов BIOS компонентами ОС.

Объем аппаратно-зависимых частей кода, которые непосредственно взаимодействуют с аппаратными средствами, должен быть по возможности минимизирован.

Аппаратно-зависимый код надежно изолируют (в нескольких модулях). В идеале слой аппаратно-зависимых компонентов ядра полностью экранирует остальную часть ОС от конкретных деталей аппаратной платформы (кэши, контроллеры прерываний ввода/вывода и т.п.). В результате и происходит подмена реальной аппаратуры некой унифицированной виртуальной машиной, одинаковой для всех вариантов аппаратной платформы.

Вопросы и упражнения

1. Какие требования предъявляются к современным ОС?
2. Определите понятия и функциональность ядра ОС и вспомогательных компонентов.
3. Перечислите виды вспомогательных модулей ОС.
4. Какие преимущества и недостатки дает поддержка процессором привилегированного режима?
5. Перечислите основные архитектуры ОС. Охарактеризуйте их различия.
6. Приведите примеры ОС различных архитектур.
7. Почему монолитная ОС быстрее многоуровневой, а многоуровневая быстрее микроядерной?
8. Опишите типичные функции слоев в многоуровневой ОС.

9. Определите различия понятий сервер и сервис в контексте ОС.
10. Какое свойство ОС обеспечивается локализацией ее аппаратно-зависимого кода?
11. Как называется модуль аппаратно-зависимого кода ОС для компьютеров платформы Intel?

ОСОБЕННОСТИ РАСПРЕДЕЛЕННЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Виды многопроцессорных ОС. Понятие распределенной ОС

Как отмечалось ранее (см. раздел Классификация ОС), с точки зрения классификации по типу аппаратной платформы различают несколько видов многопроцессорных ОС, а именно, **больших ЭВМ** (мэйнфреймов), **кластеров** и **сетевых**.

ОС мультипроцессорных компьютеров обеспечивают единую очередь ожидания и выполнение процессов, одну файловую систему (физически единый компьютер).

Сетевые ОС обеспечивают высокую степень автономности (общесистемные требования характерны в основном для коммуникационной среды) для отдельных компьютеров, объединенных коммуникационной средой. Пользователи могут вести диалог с пользователями других компьютеров, осуществлять удаленную регистрацию и доступ, работать с удаленными файлами (причем иерархия директорий может быть разной для разных пользователей) и т.д.

В последнее время в связи с ростом актуальности концепции распределенной компьютерной системы можно выделить еще один вид многопроцессорных ОС — распределенные ОС (в частности, **кластерные ОС)**.

Распределенная система — совокупность независимых компьютеров, которая представляется пользователю единым компьютером.

Предпосылкой для создания распределенных систем является наличие большого количества объединенных персональных компьютеров разных пользователей, выполняющих совместную работу (естественная распределенность ресурсов — банк, корпоративная или торговая сеть...). Для которых не должно быть ощущения неудобства от территориального распределения людей, данных и компьютеров. Кроме того, в таких системах можно достичь высокой производительности, недостижимой для централизованного компьютера в сочетании с высокой надежностью (выход из строя нескольких узлов незначительно снизит производительность системы в целом) и гибкостью использования (перераспределение нагрузки, постепенная модернизация).

Примеры: кластер, роботизированный завод (роботы связаны с разными компьютерами, но действуют как внешние устройства единой системы производства, банк со множеством филиалов, система резервирования авиабилетов).

К недостаткам распределенных систем можно отнести: проблемы разработки и настройки ПО (приложения, языки, ОС), проблемы коммуникационной среды (плохая связь, потери информации, перегрузка, развитие и замена) и возможные нарушения конфиденциальности информации.

Распределенные ОС, являясь по сути сетевыми, обеспечивают единые (глобальные для всей системы): межпроцессный коммуникационный механизм, схему контроля доступа ко всем объектам системы, файловую систему (логически единый компьютер). В табл. 2 сведены характеристики различных видов ОС.

Таблица 2

	Мультипроцессорная ОС	**Сетевая ОС**	**Распределенная ОС**
Компьютерная система выглядит как виртуальная однопроцессорная ЭВМ	+	–	–

Окончание табл. 2

	Мультипроцессорная ОС	**Сетевая ОС**	**Распределенная ОС**
Одна и та же ОС выполняется на всех процессорах	+	–	+
Количество копий ОС в ОП	1	N	N
Способ взаимодействия процессов	Разделяемая память	Разделяемые файлы	Сообщения
Необходимость согласованного сетевого протокола	–	+	+
Единая очередь выполняющихся процессов	+	–	–
Определенная семантика разделения файлов	+	–	+

Принципы построения распределенных ОС

Прозрачность (незаметность, простота использования для пользователя и приложения):

- расположения и миграции — пользователь не должен знать, где расположены ресурсы и как они перемещаются (без изменения имен),
- размножения — пользователь не должен знать, сколько копий ресурсов существует,
- конкуренции — множество пользователей разделяет ресурсы автоматически,
- параллелизма — работа программ может выполняться параллельно без участия пользователей.

Гибкость (легкость настройки и модификации)**.** Использование монолитного ядра ОС или микроядра.

Надежность и защита

- доступность, устойчивость к ошибкам (fault tolerance),
- конфиденциальность и секретность.

Производительность

Параллелизм (parallelism). Устойчивость к ошибкам требует дополнительных накладных расходов.

Масштабируемость

Устранение централизованных:

- компонентов (один почтовый-сервер);
- таблиц (один телефонный справочник);
- алгоритмов управления (маршрутизатор на основе полной информации).

Использование децентрализованных алгоритмов со следующими чертами:

- ни один компьютер не имеет полной информации о состоянии системы в целом;
- компьютеры принимают решения на основе только локальной информации;
- выход из строя одного компьютера не должен приводить к отказу.

Структура сетевой ОС

В широком смысле СОС (основа любой вычислительной сети) представляет собой совокупность ОС отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам — протоколам. В узком смысле СОС — это ОС отдельного компьютера, обеспечивающая ему возможность работать в сети.

В СОС отдельной машины можно выделить несколько частей (рис. 33):

- Средства управления локальными ресурсами компьютера (функции распределения ОП между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие).
- Средства предоставления собственных ресурсов и услуг в общее пользование — **серверная часть** ОС (сервер, который обеспечивает, например, блокировку файлов и записей при их совместном использовании; ведение справоч-

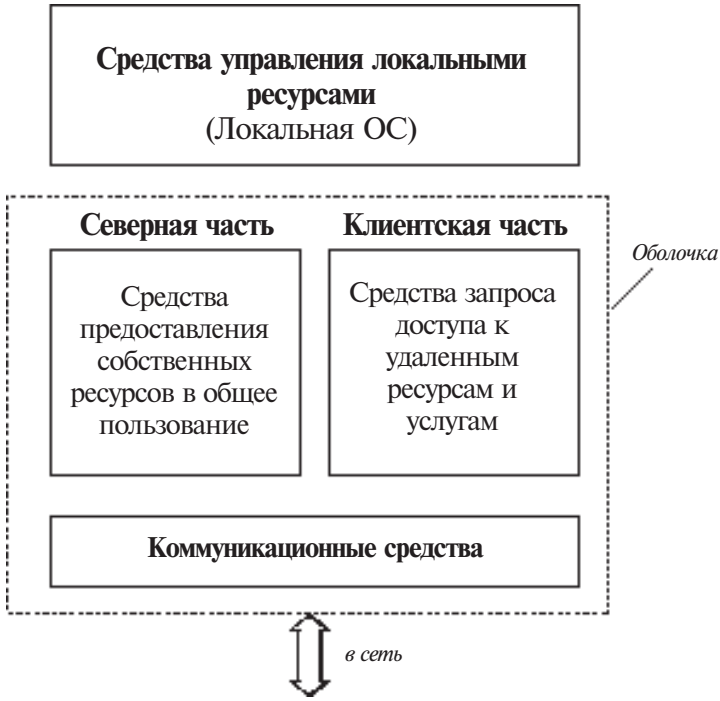

Рис. 33. Упрощенная структура СОС

ников имен сетевых ресурсов; обработку запросов удаленного доступа к собственной файловой системе и базе данных; управление очередями запросов удаленных пользователей к своим периферийным устройствам).

- Средства запроса доступа и использования удаленных ресурсов и услуг — **клиентская часть** ОС (**редиректор**). Эта часть выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передается в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет прием ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо.

- **Коммуникационные средства** ОС (транспортировки сообщений), с помощью которых происходит обмен сообщениями в сети (адресация и буферизация сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п).

В зависимости от функций, возлагаемых на конкретный компьютер в сети, в его операционной системе может отсутствовать клиентская либо серверная части.

На рис. 34 показано взаимодействие двух ПК в сети, причем, компьютер 1 является «чистым клиентом», а компьютер 2 — «чистым сервером».

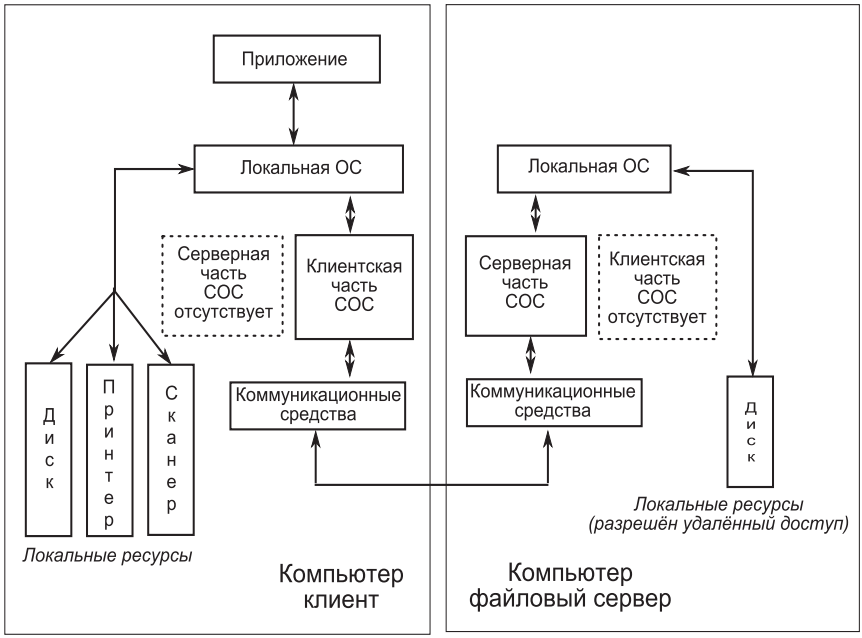

Рис. 34. Взаимодействие «чистого клиента» с «чистым сервером»

На практике сложилось несколько подходов к построению (архитектуре) СОС (рис. 35).

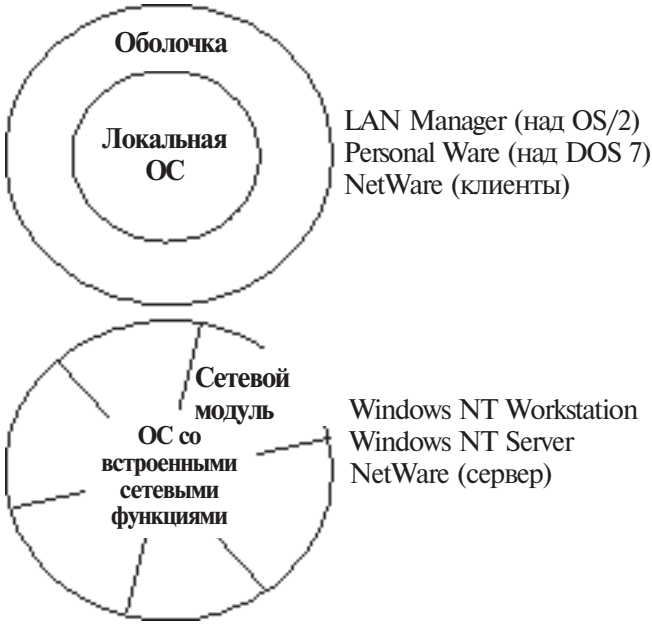

Рис. 35. Варианты архитектур СОС

Первые СОС представляли собой совокупность существующей локальной ОС и надстроенной над ней сетевой оболочки. При этом в локальную ОС встраивался минимум сетевых функций, необходимых для работы сетевой оболочки, которая выполняла основные сетевые функции. Примером такого подхода является использование на каждой машине сети операционной системы MS DOS (у которой начиная с ее третьей версии появились такие встроенные функции, как блокировка файлов и записей, необходимые для совместного доступа к файлам). Принцип построения сетевых ОС в виде сетевой оболочки над локальной ОС используется в ОС LANtastic или Personal Ware.

Однако более эффективным представляется путь разработки СОС, у которых сетевые функции реализуются непосредственно основными модулями системы, что обеспечивает их логическую стройность, простоту эксплуатации и модификации, а также

высокую производительность. Примером такой ОС является система Windows NT фирмы Microsoft, которая за счет встроенности сетевых средств обеспечивает более высокие показатели производительности и защищенности информации по сравнению с сетевой ОС LAN Manager той же фирмы (совместная разработка с IBM), являющейся надстройкой над локальной ОС OS/2.

Одноранговые сетевые ОС и ОС сетей с выделенными серверами

В зависимости от того, как распределены функции между отдельными компьютерами сети, СОС, а следовательно, и сети делятся на два класса: одноранговые и двухранговые — с выделенными серверами.

Как отмечалось выше, любой компьютер, работающий в сети, может выполнять функции либо клиента, либо сервера, либо совмещать обе эти функции.

Если выполнение каких-либо серверных функций является основным назначением компьютера (например, предоставление файлов в общее пользование всем остальным пользователям сети или организация совместного использования факса, или предоставление всем пользователям сети возможности запуска на данном компьютере своих приложений), то такой компьютер называется выделенным сервером. В зависимости от того, какой ресурс сервера является разделяемым, он называется файл-сервером, факс-сервером, принт-сервером, сервером приложений и т.д.

На выделенных серверах устанавливают ОС, оптимизированные для выполнения серверных функций. Выделенный сервер нецелесообразно использовать для выполнения прикладных пользовательских задач, поскольку уменьшается производительность его работы как сервера.

К примеру, в ОС Novell NetWare на серверной части возможность выполнения обычных прикладных программ вообще не предусмотрена, то есть сервер не содержит клиентской части, а на рабочих станциях отсутствуют серверные компоненты.

Однако в других сетевых ОС функционирование на выделенном сервере клиентской части допустимо. Например, под управлением Windows NT Server могут запускаться приложения локального пользователя, которые могут потребовать выполнения клиентских функций ОС при появлении запросов к ресурсам других компьютеров сети. При этом рабочие станции, на которых установлена ОС Windows NT Workstation, могут выполнять функции невыделенного сервера.

Сеть с выделенным сервером функционально несимметрична — в ней реализованы два типа компьютеров: одни, в большей степени ориентированные на выполнение серверных функций (работают под управлением специализированных серверных компонентов ОС), а другие — в основном выполняющие клиентские функции (работают под управлением клиентских компонентов ОС). Функциональная несимметричность, как правило, вызывает и несимметричность аппаратуры — для выделенных серверов используются более мощные компьютеры с большими объемами оперативной и внешней памяти. Таким образом, функциональная несимметричность в сетях с выделенным сервером сопровождается несимметричностью операционных систем (специализация ОС) и аппаратной несимметричностью (специализация компьютеров).

В одноранговых сетях все компьютеры равны в правах доступа к ресурсам друг друга. Каждый пользователь может по своему желанию объявить какой-либо ресурс своего компьютера разделяемым, после чего другие пользователи могут его эксплуатировать. В таких сетях на всех компьютерах устанавливается одна и та же ОС, которая предоставляет всем компьютерам в сети потенциально равные возможности. Одноранговые сети могут быть построены, например, на базе ОС LANtastic, Personal Ware, Windows for Workgroup, Windows NT Workstation.

В одноранговых сетях также может возникнуть функциональная несимметричность: одни пользователи не желают разделять свои ресурсы с другими, и в таком случае их компьютеры выполняют роль клиента. За другими компьютерами администратор закрепил только функции по организации совместного использования ресурсов, а значит, они являются серверами;

в третьем случае, когда локальный пользователь не возражает против использования его ресурсов и сам не исключает возможности обращения к другим компьютерам, ОС, устанавливаемая на его компьютере, должна включать и серверную, и клиентскую части.

В отличие от сетей с выделенными серверами, в одноранговых сетях отсутствует специализация ОС в зависимости от преобладающей функциональной направленности — клиентская или серверная. Все вариации реализуются средствами конфигурирования одного и того же варианта ОС.

Одноранговые сети проще в организации и эксплуатации и применяются для объединения небольших групп пользователей, оперирующих сравнительно небольшим объемом слабо защищенной информации.

Особенности ОС сетей различных масштабов

СОС имеют разные свойства в зависимости от того, предназначены они для сетей масштаба рабочей группы (отдела), для сетей масштаба кампуса или для сетей масштаба предприятия.

- **Сети отделов** — используются небольшой группой сотрудников, решающих общие задачи. Главной целью сети отдела является разделение локальных ресурсов, таких как приложения, данные, лазерные принтеры и модемы.
- **Сети кампусов** — соединяют несколько сетей отделов внутри отдельного здания или внутри одной территории предприятия. Эти сети являются все еще локальными сетями, хотя и могут покрывать территорию в несколько квадратных километров. Сервисы такой сети включают взаимодействие между сетями отделов, доступ к базам данных предприятия, доступ к факс-серверам, высокоскоростным модемам и высокоскоростным принтерам.
- **Сети предприятия** (корпоративные сети) — объединяют все компьютеры удаленных территорий отдельного предприятия. Они могут покрывать город, регион или даже континент. В таких сетях пользователям предоставляется доступ к информации и приложениям, находящимся в других

рабочих группах, других отделах, подразделениях и штаб-квартирах корпорации.

Обычно сети отделов имеют один или два файловых сервера и не более чем 30 пользователей. Задачи СОС управления на уровне отдела относительно просты. В задачи администратора входит добавление новых пользователей, устранение простых отказов, инсталляция новых узлов и установка новых версий ПО. Сети отделов, уже давно применяющиеся и достаточно отлаженные. Такая сеть обычно использует одну или максимум две СОС — чаще с выделенным сервером NetWare 3.x или Windows NT или же одноранговую, например, Windows for Workgroups.

Следующим шагом в эволюции сетей является объединение локальных сетей нескольких отделов в единую сеть здания или группы зданий. Такие сети называют сетями кампусов. Сети кампусов могут простираться на несколько километров, но при этом глобальные соединения не требуются.

СОС, работающие в сети кампуса, должны обеспечивать для сотрудников одних отделов доступ к некоторым файлам и ресурсам сетей других отделов. Услуги, предоставляемые ОС сетей кампусов, не ограничиваются простым разделением файлов и принтеров, а ориентированы на доступ к серверам других типов, например, к факс-серверам, серверам высокоскоростных модемов, корпоративным распределенным базам данных.

Именно на уровне сети кампуса начинаются проблемы интеграции ресурсов. В общем случае отделы уже выбрали для себя типы компьютеров, сетевого оборудования и СОС. Например, инженерный отдел может использовать ОС UNIX и сетевое оборудование Ethernet, отдел продаж может использовать ОС DOS/Novell и оборудование Token Ring. Очень часто сеть кампуса соединяет разнородные по архитектуре компьютеры, в то время как сети отделов используют однотипные.

Корпоративная сеть соединяет сети всех подразделений предприятия, в общем случае находящихся на значительных расстояниях. Корпоративные сети используют глобальные связи (WAN links) для соединения локальных сетей или отдельных удаленных компьютеров.

Пользователям корпоративных сетей требуются все те приложения и услуги, которые имеются в сетях отделов и кампусов, плюс некоторые дополнительные, например, доступ к приложениям мейнфреймов и мини-компьютеров и к глобальным сетям. Когда ОС разрабатывается для локальной сети или рабочей группы, то ее главной обязанностью является разделение файлов и других сетевых ресурсов (обычно принтеров) между локально подключенными пользователями. Такой подход неприменим для уровня предприятия. Наряду с базовыми сервисами, связанными с разделением файлов и принтеров, СОС, которая разрабатывается для корпораций, должна поддерживать более широкий набор сервисов, в который обычно входят почтовая служба, средства коллективной работы, поддержка удаленных пользователей, факс-сервис, обработка голосовых сообщений, организация видеоконференций и др.

Кроме того, многие существующие методы и подходы к решению традиционных задач сетей меньших масштабов для корпоративной сети непригодны. На первый план вышли такие задачи и проблемы, которые в сетях отделов и кампусов либо имели второстепенное значение, либо вообще не проявлялись. Например, простейшая для небольшой сети задача ведения учетной информации о пользователях выросла в сложную проблему для сети масштаба предприятия. А использование глобальных связей требует от корпоративных ОС поддержки протоколов, хорошо работающих на низкоскоростных линиях, и отказа от некоторых традиционно используемых протоколов (например, тех, которые активно используют широковещательные сообщения). Особое значение приобрели задачи преодоления гетерогенности (неоднородности) — в сети появились многочисленные шлюзы, обеспечивающие согласованную работу различных ОС и сетевых системных приложений. А также сервис ИБ.

К признакам корпоративных ОС могут быть отнесены следующие особенности.

Поддержка приложений. В корпоративных сетях выполняются сложные приложения, требующие для выполнения большой вычислительной мощности. Такие приложения разделяются на несколько частей, например, на одном компьютере выполняет-

ся часть приложения, связанная с выполнением запросов к базе данных, на другом — запросов к файловому сервису, а на клиентских машинах — часть, реализующая логику обработки данных приложения и организующая интерфейс с пользователем. Вычислительная часть общих для корпорации программных систем может быть слишком объемной и неподъемной для рабочих станций клиентов, поэтому приложения будут выполняться более эффективно, если их наиболее сложные в вычислительном отношении части перенести на специально предназначенный для этого мощный компьютер — сервер приложений.

Сервер приложений должен базироваться на мощной аппаратной платформе (мультипроцессорные системы, часто на базе RISC-процессоров, специализированные кластерные архитектуры). ОС сервера приложений должна обеспечивать высокую производительность вычислений, а значит, поддерживать многонитевую обработку, вытесняющую многозадачность, мультипроцессирование, виртуальную память и наиболее популярные прикладные среды (UNIX, Windows, MS-DOS, OS/2). В этом отношении сетевую ОС NetWare трудно отнести к корпоративным продуктам, так как в ней отсутствуют почти все требования, предъявляемые к серверу приложений.

Справочная служба. Корпоративная ОС должна обладать способностью хранить информацию обо всех пользователях и ресурсах таким образом, чтобы обеспечивалось управление ею из одной центральной точки. Подобно большой организации, корпоративная сеть нуждается в централизованном хранении как можно более полной справочной информации о самой себе (начиная с данных о пользователях, серверах, рабочих станциях и кончая данными о кабельной системе). Естественно организовать эту информацию в виде базы данных. Данные из этой базы могут быть востребованы многими сетевыми системными приложениями, в первую очередь системами управления и администрирования. Кроме того, такая база полезна при организации электронной почты, систем коллективной работы, службы безопасности, службы инвентаризации программного и аппаратного обеспечения сети практически любого крупного бизнес-приложения.

База данных, хранящая справочную информацию, предоставляет все то же многообразие возможностей и порождает все то же множество проблем, что и любая другая крупная база данных. Она позволяет осуществлять различные операции поиска, сортировки, модификации и т.п., что очень сильно облегчает жизнь как администраторам, так и пользователям. Но за эти удобства приходится расплачиваться решением проблем распределенности, репликации и синхронизации.

В идеале сетевая справочная информация должна быть реализована в виде единой базы данных, а не представлять собой набор баз данных, специализирующихся на хранении информации того или иного вида, как это часто бывает в реальных ОС.

Например, в Windows NT имеется по крайней мере пять различных типов справочных баз данных. Главный справочник домена (NT Domain Directory Service) хранит информацию о пользователях, которая используется при организации их логического входа в сеть. Данные о тех же пользователях могут содержаться и в другом справочнике, используемом электронной почтой Microsoft Mail. Еще три базы данных поддерживают разрешение низкоуровневых адресов: WINS — устанавливает соответствие Netbios-имен IP-адресам, справочник DNS — сервер имен домена — оказывается полезным при подключении NT-сети к Internet и, наконец, справочник протокола DHCP используется для автоматического назначения IP-адресов компьютерам сети.

Ближе к идеалу находятся справочные службы, поставляемые фирмой Banyan (продукт Streettalk III) и фирмой Novell (NetWare Directory Services), предлагающие единый справочник для всех сетевых приложений. Наличие единой справочной службы для сетевой операционной системы — один из важнейших признаков ее корпоративности.

Информационная безопасность. Особую важность для ОС корпоративной сети приобретают вопросы безопасности данных. С одной стороны, в крупномасштабной сети объективно существует больше возможностей для несанкционированного доступа — из-за децентрализации данных и большой распределенности «законных» точек доступа, из-за большого числа

пользователей, благонадежность которых трудно установить, а также из-за большого числа возможных точек несанкционированного подключения к сети. С другой стороны, корпоративные бизнес-приложения работают с данными, которые имеют жизненно важное значение для успешной работы корпорации в целом. И для защиты таких данных в корпоративных сетях наряду с различными аппаратными средствами используется весь спектр средств защиты, предоставляемый ОС: избирательные или мандатные права доступа, сложные процедуры аутентификации пользователей, программная шифрация и т.д.

Вопросы и упражнения

1. Сформулируйте определения сетевой и распределенной ОС.
2. Перечислите и охарактеризуйте принципы построения распределенных ОС.
3. Какие функциональные возможности отличают СОС от локальной?
4. Определите понятия (в контексте ОС): редиректор, сервер, клиент, сервис.
5. Какие архитектуры характерны для СОС?
6. Определите различия одноранговых и двухранговых сетей. В каких случаях эффективность их использования наибольшая?
7. Может ли сервер быть невыделенным?
8. Приведите примеры ОС одноранговых и ОС двухранговых сетей.
9. Начиная с какого масштаба сетей допускается их гетерогенность?
10. Поддержка каких функций является признаком корпоративной сети?
11. Какие задачи решает справочная служба ОС корпоративной сети?

ЛИТЕРАТУРА

1. *Дейтел Г.* Введение в операционные системы: В 2-х т. Пер. с англ. М.: Мир, 1987.
2. *Донован Дж.* Системное программирование: Пер с англ. / под ред. Л.Д. Райткова. М.: Мир, 1975.
3. *Гранже М., Менсье Ф.* OS/2: принципы построения и установка: Пер. с франц. М.: Мир, 1991.
4. *Фролов А.В. и др.* Операционная система OS/2 Warp. М.: Диалог-МИФИ, 1995.
5. *Минаси М. и др.* OS/2 Warp изнутри: Пер. с англ. Т. 1–2. СПб.: Питер, 1996.
6. *Ливингстон Б.* Секреты Windows 3.1. К.: Диалектика, 1994.
7. *Шульман Э.* Неофициальная Windows 95. К.: Диалектика, 1995.
8. *Кинг А.* Windows 95 изнутри. СПб.: Питер, 1995.
9. *Борланд Р.* Знакомство с MS Windows 98: Пер. с англ. М.: Русская редакция, 1997.
10. *Фролов А.В. и др.* Программирование для Windows NT: 1-2 ч. М.: Диалог-МИФИ, 1996/97.
11. *Браун П.* Введение в ОС Unix: Пер. с англ. М.: Мир, 1987.
12. *Келли-Бутл С.* Введение в Unix: Пер. с англ. М.: Лори, 1995.
13. *Керниган Б.* Unix — универсальная среда программирования: Пер. с англ. М.: Финансы и статистика, 1992.
14. *Пасечник А.* RedHat Linux 6.2: учебный курс. СПб.: Питер, 2000.
15. *Олифер В.Г., Олифер Н.А.* Сетевые операционные системы. СПб.: Питер, 2001.
16. *Иртегов Д.В.* Введение в операционные системы. СПб.: БХВ-Петербург, 2002.
17. *Столлингс В.* Операционные системы. 4-е изд.: Пер. с англ. М.: Вильямс, 2002.
18. *Таненбаум Э.* Современные операционные системы. СПб.: Питер, 2002.

19. *Лацис А.О.* Как построить и использовать суперкомпьютеры. М.: Бестселлер. 2003.

20. *Андреев А.Г. и др.* MS Windows 2000: Server и Professional. Русские версии. СПб.: БХВ-Петербург, 2001.

21. *Немеет Э. и др.* UNIX: руководство системного администратора. Для профессионалов: Пер. с англ. СПб.: Питер, К.: БХВ, 2002.

22. *Одинцов И.О.* Профессиональное программирование. Системный подход. СПб.: БХВ-Петербург, 2002.

23. *Вудхалл А., Таненбаум Э.* Операционные системы: разработка и реализация. СПб.: Питер, 2005.

Учебное издание

Стащук Петр Владимирович

КРАТКОЕ ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие

Подписано в печать 08.09.2014

Электронное издание для распространения через Интернет.

ООО «ФЛИНТА», 117342, г. Москва,
ул. Бутлерова, д. 17-Б, комн. 324.
Тел./факс: (495) 334-82-65; тел. (495) 336-03-11. E-mail:
flinta@mail.ru; WebSite: www.flinta.ru.