Сетевые операционные системы

Н. А. Олифер, В. Г. Олифер, Центр Информационных Технологий

1. Введение
    - Определение операционной системы
    - Эволюция ОС
        - Первый период (1945 -1955)
        - Второй период (1955 - 1965)
        - Третий период (1965 - 1980)
        - Четвертый период (1980 - настоящее время)
    - Классификация ОС
        - Особенности алгоритмов управления ресурсами
        - Особенности аппаратных платформ
        - Особенности областей использования
        - Особенности методов построения
    - Сетевые операционные системы
        - Структура сетевой операционной системы
        - Одноранговые сетевые ОС и ОС с выделенными серверами
        - ОС для рабочих групп и ОС для сетей масштаба предприятия
2. Управление локальными ресурсами
    - Управление процессами
        - Состояние процессов
        - Контекст и дескриптор процесса
        - Алгоритмы планирования процессов
        - Вытесняющие и невытесняющие алгоритмы планирования
        - Средства синхронизации и взаимодействия процессов
        - Нити
    - Управление памятью
        - Типы адресов
        - Методы распределения памяти без использования дискового пространства
            - Распределение памяти фиксированными разделами
            - Распределение памяти разделами переменной величины
            - Перемещаемые разделы
        - Методы распределения памяти с использованием дискового пространства
            - Понятие виртуальной памяти
            - Страничное распределение
            - Сегментное распределение
            - Странично-сегментное распределение
            - Свопинг

        - Иерархия запоминающих устройств. Принцип кэширования данных
    - Средства аппаратной поддержки управления памятью и многозадачной среды в микропроцессорах Intel 80386, 80486 и Pentium
        - Средства поддержки сегментации памяти
        - Сегментно-страничный механизм
        - Средства вызова подпрограмм и задач
    - Управление вводом-выводом
        - Физическая организация устройств ввода-вывода
        - Организация программного обеспечения ввода-вывода
        - Обработка прерываний
        - Драйверы устройств
        - Независимый от устройств слой операционной системы
        - Пользовательский слой программного обеспечения
    - Файловая система
        - Имена файлов
        - Типы файлов
        - Логическая организация файла
        - Физическая организация и адрес файла
        - Права доступа к файлу
        - Кэширование диска
        - Общая модель файловой системы
        - Отображаемые в память файлы
        - Современные архитектуры файловых систем
3. Управление распределенными ресурсами
    - Базовые примитивы передачи сообщений в распределенных системах
        - Способы адресации
        - Блокирующие и неблокирующие примитивы
        - Буферизуемые и небуферизуемые примитивы
        - Надежные и ненадежные примитивы
    - Вызов удаленных процедур (RPC)
        - Концепция удаленного вызова процедур
        - Базовые операции RPC
        - Этапы выполнения RPC
        - Динамическое связывание
        - Семантика RPC в случае отказов
    - Синхронизация в распределенных системах
        - Алгоритм синхронизации логических часов
        - Алгоритмы взаимного исключения
        - Неделимые транзакции
    - Процессы и нити в распределенных системах
        - Понятие "нить"

        - Различные способы организации вычислительного процесса с использованием нитей
        - Вопросы реализации нитей
        - Нити и RPC
    - Распределенные файловые системы
        - Интерфейс файлового сервиса
        - Интерфейс сервиса каталогов
        - Семантика разделения файлов
        - Вопросы разработки структуры файловой системы
        - Кэширование
        - Репликация
    - Проблемы взаимодействия операционных систем в гетерогенных сетях
        - Понятия "internetworking" и "interoperability"
        - Гетерогенность
        - Основные подходы к реализации взаимодействия сетей
        - Шлюзы
        - Мультиплексирование стеков протоколов
        - Использование магистрального протокола
        - Вопросы реализации
        - Сравнение вариантов организации взаимодействия сетей
    - Службы именования ресурсов и проблемы прозрачности доступа
        - Доменный подход
            - Основной и резервные контроллеры домена
            - Четыре модели организации связи доменов
4. Современные концепции и технологии проектирования операционных систем
    - Требования, предъявляемые к ОС 90-х годов
        - Расширяемость
        - Переносимость
        - Совместимость
        - Безопасность
    - Тенденции в структурном построении ОС
        - Монолитные системы
        - Многоуровневые системы
        - Модель клиент-сервер и микроядра
        - Объектно-ориентированный подход
        - Множественные прикладные среды
        - Сетевой пакет DCE фирмы OSF
5. Семейство операционных систем UNIX
    - История и общая характеристика семейства операционных систем UNIX
    - Концепции UNIX System V Release 4
        - Управление процессами

        - Образ, дескриптор, контекст процесса
        - Порождение процессов
        - Планирование процессов
    - Файловые системы UNIX System V Release 4
        - Традиционная файловая система s5
        - Виртуальная файловая система VFS
        - Сетевая файловая система NFS
    - Управление памятью. Свопинг
    - Система ввода-вывода
        - Подсистема буферизации
        - Драйверы
- Коммерческие реализации UNIX
    - UnixWare
        - Дополнительные свойства UnixWare по сравнению с UNIX System V Release 4
    - Solaris
    - SCO UNIX System V/386

6. Микроядро Mach
    - Введение в Mach
        - История Mach
        - Цели Mach
        - Основные концепции Mach
        - Сервер Mach BSD UNIX
    - Управление процессами в Mach
        - Процессы
        - Примитивы управления процессами
        - Нити
        - Планирование
    - Управление памятью в Mach
        - Виртуальная память
        - Разделение памяти
        - Внешние менеджеры памяти
        - Распределенная разделяемая память в Mach
    - Коммуникации в ядре Mach
        - Порты
        - Отправка и получение сообщений
        - Сервер сетевых сообщений
    - Эмуляция BSD UNIX в Mach
7. Сетевые продукты фирмы Novell
    - История и версии сетевой ОС NetWare
    - Версии 4.0, 4.01 и 4.02
    - Версия NetWare 4.1
    - Концепции построения NetWare
        - Структура NetWare и обзор особенностей

        - Способы повышения производительности
        - Способы обеспечения открытости и расширяемости
        - Способы обеспечения надежности
        - Защита информации
      - Управление процессами
      - Файловая система
    - Основные направления развития NetWare
      - Поддержка мультипроцессирования
      - Обеспечение процессорной независимости
    - Операционные системы рабочих станций фирмы Novell
    - Сетевые системные утилиты
      - NetWare Connect 1.0 фирмы Novell
      - WinView for Networks v2.2 фирмы Citrix Systems
      - Шлюзы IP-сетей
      - Системы обработки сообщений MHS и GroupWise
8. Семейство сетевых ОС компании Microsoft
    - Сетевые продукты Microsoft
    - История Windows NT
    - Версии Windows NT
      - Windows NT 3.1
      - Windows NT 3.5
      - Windows NT 4.0
      - Области использования Windows NT
    - Концепции Windows NT
      - Структура: NT executive и защищенные подсистемы
      - Множественные прикладные среды
      - Объектно-ориентированный подход
      - Процессы и нити
      - Алгоритм планирования процессов и нитей
      - Сетевые средства
      - Совместимость Windows NT с NetWare
    - Средства BackOffice
      - Сервер баз данных SQL Server
      - Шлюз SNA Server
      - Почтовые системы Microsoft Mail и система коллективной работы Microsoft Exchange
      - Система управления компьютерами System Management Server
    - Windows 95 и будущее семейства Windows
9. Операционная система OS/2
    - История развития OS/2 и ее место на рынке
      - Битва Microsoft - IBM на рынке настольных ОС
      - OS/2 - постепенные улучшения
    - OS/2 Warp

        - Общая характеристика
        - Внутренняя организация OS/2 Warp
        - Файловая система HPFS
    - LAN Server 4.0
        - Общая характеристика
        - Сетевые возможности
        - Управление сервером LAN Server 4.0
        - Совместимость с NetWare
10. Обзор сетевых операционных систем

Введение

Определение операционной системы

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

ОС как расширенная машина

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

С этой точки зрения функцией ОС является предоставление пользователю некоторой расширенной или виртуальной машины, которую легче

программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

ОС как система управления ресурсами

Идея о том, что ОС прежде всего система, обеспечивающая удобный интерфейс пользователям, соответствует рассмотрению сверху вниз. Другой взгляд, снизу вверх, дает представление об ОС как о некотором механизме, управляющем всеми частями сложной системы. Современные вычислительные системы состоят из процессоров, памяти, таймеров, дисков, накопителей на магнитных лентах, сетевых коммуникационной аппаратуры, принтеров и других устройств. В соответствии со вторым подходом функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:

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

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

Эволюция ОС

Первый период (1945 -1955)

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

Некоторый прогресс в создании цифровых вычислительных машин произошел после второй мировой войны. В середине 40-х были созданы первые ламповые вычислительные устройства. В то время одна и та же группа людей участвовала и в проектировании, и в эксплуатации, и в программировании вычислительной

машины. Это была скорее научно-исследовательская работа в области вычислительной техники, а не использование компьютеров в качестве инструмента решения каких-либо практических задач из других прикладных областей. Программирование осуществлялось исключительно на машинном языке. Об операционных системах не было и речи, все задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Не было никакого другого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

Второй период (1955 - 1965)

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

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

Третий период (1965 - 1980)

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

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

Программная совместимость требовала и совместимости операционных систем. Такие операционные системы должны были бы работать и на больших, и на малых вычислительных системах, с большим и с малым количеством разнообразной периферии, в коммерческой области и в области научных исследований. Операционные системы, построенные с намерением удовлетворить всем этим противоречивым требованиям, оказались чрезвычайно сложными "монстрами". Они состояли из многих миллионов ассемблерных строк, написанных тысячами программистов, и содержали тысячи ошибок, вызывающих нескончаемый поток исправлений. В каждой новой версии операционной системы исправлялись одни ошибки и вносились другие.

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

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

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

Четвертый период (1980 - настоящее время)

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

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

На рынке операционных систем доминировали две системы: MS-DOS и UNIX. Однопрограммная однопользовательская ОС MS-DOS широко использовалась для компьютеров, построенных на базе микропроцессоров Intel 8088, а затем 80286, 80386 и 80486. Мультипрограммная многопользовательская ОС UNIX доминировала в среде "не-интеловских" компьютеров, особенно построенных на базе высокопроизводительных RISC-процессоров.

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

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

Классификация ОС

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

Ниже приведена классификация ОС по нескольким наиболее основным признакам.

Особенности алгоритмов управления ресурсами

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

однопользовательские, на системы, поддерживающие многонитевую обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

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

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

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

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

Поддержка многопользовательского режима. По числу одновременно работающих пользователей ОС делятся на:

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

Главным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует заметить, что не всякая многозадачная система является многопользовательской, и не всякая однопользовательская ОС является однозадачной.

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

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

Основным различием между вытесняющим и невытесняющим вариантами многозадачности является степень централизации механизма планирования процессов. В первом случае механизм планирования процессов целиком сосредоточен в операционной системе, а во втором - распределен между системой и прикладными программами. При невытесняющей многозадачности активный процесс выполняется до тех пор, пока он сам, по собственной

инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению процесс. При вытесняющей многозадачности решение о переключении процессора с одного процесса на другой принимается операционной системой, а не самим активным процессом.

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

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

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

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

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

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

Особенности аппаратных платформ

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры

различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. В любом случае специфика аппаратных средств, как правило, отражается на специфике операционных систем.

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

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

Многопроцессорные системы требуют от операционной системы особой организации, с помощью которой сама операционная система, а также поддерживаемые ею приложения могли бы выполняться параллельно отдельными процессорами системы. Параллельная работа отдельных частей ОС создает дополнительные проблемы для разработчиков ОС, так как в этом случае гораздо сложнее обеспечить согласованный доступ отдельных процессов к общим системным таблицам, исключить эффект гонок и прочие нежелательные последствия асинхронного выполнения работ.

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

Наряду с ОС, ориентированными на совершенно определенный тип аппаратной платформы, существуют операционные системы, специально разработанные таким образом, чтобы они могли быть легко перенесены с компьютера одного типа на компьютер другого типа, так называемые *мобильные* ОС. Наиболее

ярким примером такой ОС является популярная система UNIX. В этих системах аппаратно-зависимые места тщательно локализованы, так что при переносе системы на новую платформу переписываются только они. Средством, облегчающем перенос остальной части ОС, является написание ее на машинно-независимом языке, например, на С, который и был разработан для программирования операционных систем.

Особенности областей использования

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС ЕС),
- системы разделения времени (UNIX, VMS),
- системы реального времени (QNX, RT/11).

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

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки - изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени

предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая "выгодна" системе, и, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу. Критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

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

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

Особенности методов построения

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

К таким базовым концепциям относятся:

- Способы построения ядра системы - монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС - серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой - ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы.
- Построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства, хорошо зарекомендовавшие себя на уровне приложений, внутри операционной системы, а именно: аккумуляцию удачных решений в форме стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма наследования, хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне, структуризованность системы, состоящей из набора хорошо определенных объектов.
- Наличие нескольких прикладных сред дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, UNIX (POSIX), OS/2 или хотя бы некоторого подмножества из этого популярного набора. Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализуют прикладную среду той или иной операционной системы.
- Распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерными признаками распределенной организации ОС являются: наличие единой справочной службы разделяемых ресурсов, единой службы времени, использование механизма вызова удаленных процедур (RPC) для прозрачного

распределения программных процедур по машинам, многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети, а также наличие других распределенных служб.

Сетевые операционные системы

Структура сетевой операционной системы

Сетевая операционная система составляет основу любой вычислительной сети. Каждый компьютер в сети в значительной степени автономен, поэтому под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам - протоколам. В узком смысле сетевая ОС - это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

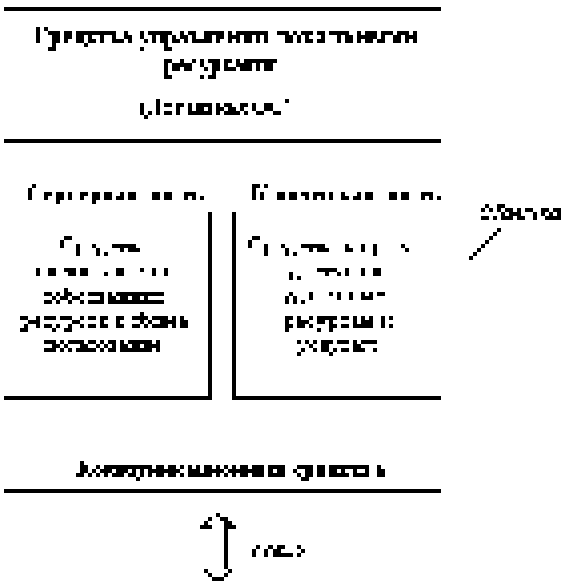


Рис. 1.1. Структура сетевой ОС

В сетевой операционной системе отдельной машины можно выделить несколько частей (рисунок 1.1):

- Средства управления локальными ресурсами компьютера: функции распределения оперативной памяти между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие функции управления ресурсами локальных ОС.
- Средства предоставления собственных ресурсов и услуг в общее пользование - серверная часть ОС (сервер). Эти средства обеспечивают, например, блокировку файлов и записей, что необходимо для их совместного использования; ведение справочников 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 к удаленным ресурсам и услугам и их использования - 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, с помощью которых происходит обмен сообщениями в сети. Эта часть обеспечивает адресацию и буферизацию сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п., то есть является средством транспортировки сообщений.

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

На рисунке 1.2 показано взаимодействие сетевых компонентов. Здесь компьютер 1 выполняет роль "чистого" клиента, а компьютер 2 - роль "чистого" сервера, соответственно на первой машине отсутствует серверная часть, а на второй - клиентская. На рисунке отдельно показан компонент клиентской части - редиректор. Именно редиректор перехватывает все запросы, поступающие от приложений, и анализирует их. Если выдан запрос к ресурсу данного компьютера, то он переадресовывается соответствующей подсистеме локальной ОС, если же это запрос к удаленному ресурсу, то он переправляется в сеть. При этом клиентская часть преобразует запрос из локальной формы в сетевой формат и передает его транспортной подсистеме, которая отвечает за доставку сообщений указанному серверу. Серверная часть операционной системы компьютера 2 принимает запрос, преобразует его и передает для выполнения своей локальной ОС. После того, как результат получен, сервер обращается к транспортной подсистеме и направляет ответ клиенту, выдавшему запрос. Клиентская часть преобразует результат в соответствующий формат и адресует его тому приложению, которое выдало запрос.

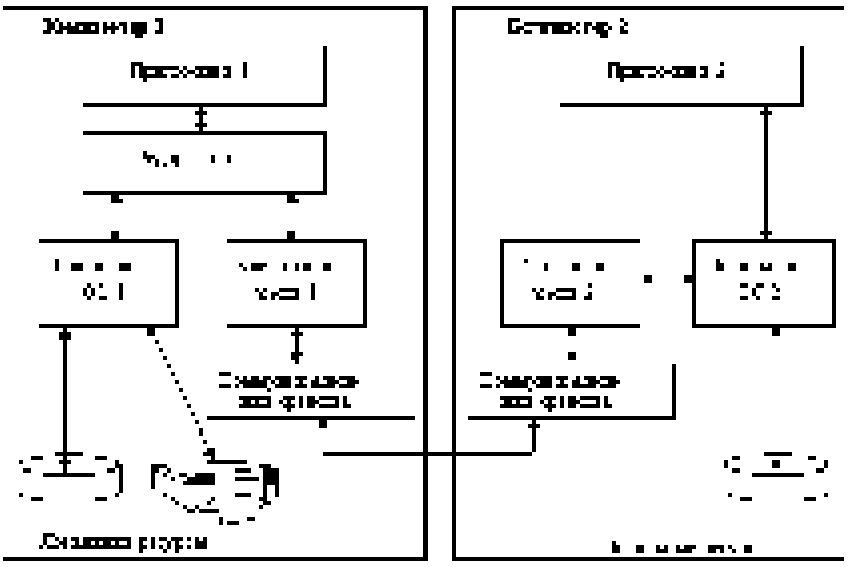


Рис. 1.2. взаимодействие компонентов операционной системы при взаимодействии компьютеров

На практике сложилось несколько подходов к построению сетевых операционных систем (рисунок 1.3).

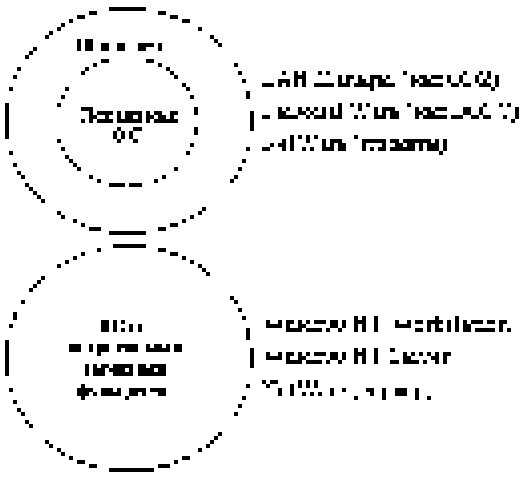


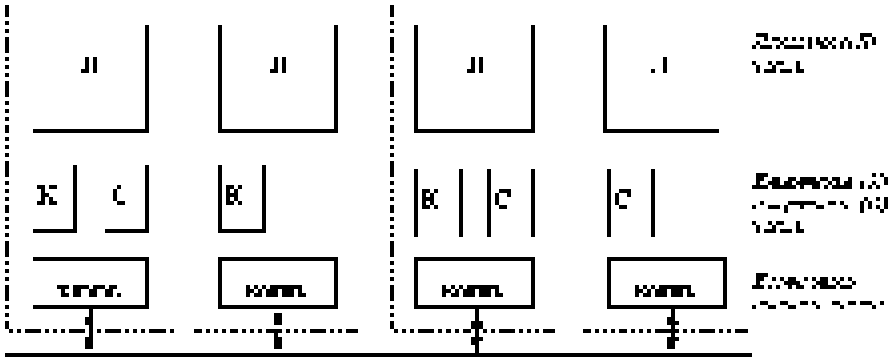
Рис. 1.3. Варианты построения сетевых ОС

Первые сетевые ОС 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 и в современных ОС, таких, например, как LANtastic или Personal Ware.

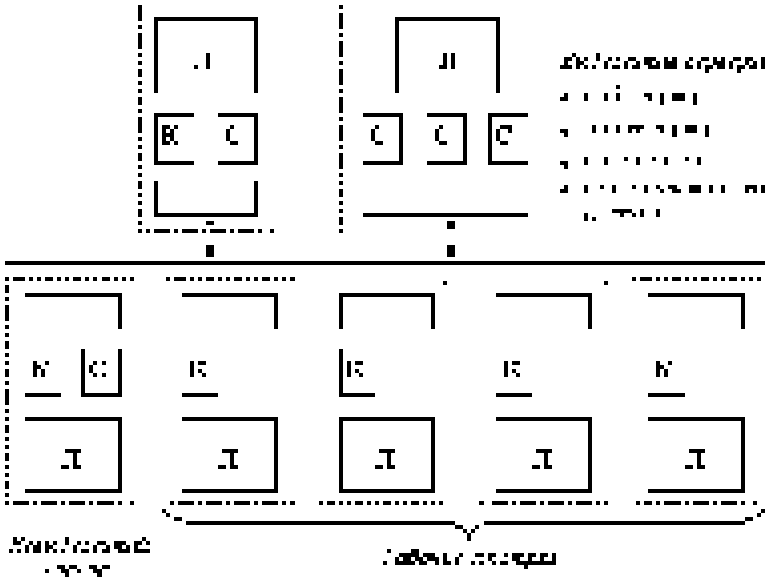
Однако более эффективным представляется путь разработки операционных систем, изначально предназначенных для работы в сети. Сетевые функции у ОС такого типа глубоко *встроены* в основные модули системы, что обеспечивает их логическую стройность, простоту эксплуатации и модификации, а также высокую производительность. 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 операционной системой OS/2.

Одноранговые сетевые ОС и ОС с выделенными серверами

В зависимости от того, как распределены функции между компьютерами сети, сетевые операционные системы, а следовательно, и сети делятся на два класса: одноранговые и двухранговые (рисунок 1.4). Последние чаще называют сетями с выделенными серверами.



(а)



(б)

Рис. 1.4. (а) - Одноранговая сеть, (б) - Двухранговая сеть

Если компьютер предоставляет свои ресурсы другим пользователям сети, то он играет роль сервера. При этом компьютер, обращающийся к ресурсам другой машины, является клиентом. Как уже было сказано, 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.

Очевидно, что на выделенных серверах желательно устанавливать ОС, специально оптимизированные для выполнения тех или иных серверных функций. Поэтому в сетях с выделенными серверами чаще всего используются сетевые операционные системы, в состав которых входит нескольких вариантов ОС, отличающихся возможностями серверных частей. Например, сетевая ОС Novell NetWare имеет серверный вариант, оптимизированный для работы в качестве файл-сервера, а также варианты оболочек для рабочих станций с различными локальными ОС, причем эти оболочки выполняют исключительно функции клиента. Другим примером ОС, ориентированной на построение сети с выделенным сервером, является операционная система Windows NT. В отличие от NetWare, оба варианта данной сетевой ОС - Windows NT Server (для выделенного сервера) и Windows NT Workstation (для рабочей станции) - могут поддерживать функции и клиента и сервера. Но серверный вариант Windows NT имеет больше возможностей для предоставления ресурсов своего компьютера другим пользователям сети, так как может выполнять более широкий набор функций, поддерживает большее количество одновременных соединений с клиентами, реализует централизованное управление сетью, имеет более развитые средства защиты.

Выделенный сервер не принято использовать в качестве компьютера для выполнения текущих задач, не связанных с его основным назначением, так как это может уменьшить производительность его работы как сервера. В связи с такими соображениями 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 вполне возможно. Например, под управлением Windows NT Server могут запускаться обычные программы 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.

Важно понять, что несмотря на то, что в сети с выделенным сервером все компьютеры в общем случае могут выполнять одновременно роли и сервера, и клиента, эта сеть функционально не симметрична: аппаратно и программно в ней реализованы два типа компьютеров - одни, в большей степени ориентированные на выполнение серверных функций и работающие под управлением специализированных серверных ОС, а другие - в основном выполняющие клиентские функции и работающие под управлением соответствующего этому назначению варианта 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 - клиента или сервера. Все вариации реализуются средствами конфигурирования одного и того же варианта ОС.

Одноранговые сети проще в организации и эксплуатации, однако они применяются в основном для объединения небольших групп пользователей, не предъявляющих больших требований к объемам хранимой информации, ее защищенности от несанкционированного доступа и к скорости доступа. При повышенных требованиях к этим характеристикам более подходящими являются двухранговые сети, где сервер лучше решает задачу обслуживания пользователей своими ресурсами, так как его аппаратура и сетевая операционная система специально спроектированы для этой цели.

ОС для рабочих групп и ОС для сетей масштаба предприятия

Сетевые операционные системы 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 отделов обычно не разделяются на подсети.
- *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 всех 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.

Главной задачей операционной системы, используемой в сети масштаба отдела, является организация разделения ресурсов, таких как приложения, данные, лазерные принтеры и, возможно, низкоскоростные модемы. Обычно сети отделов имеют один или два файловых сервера и не более чем 30 пользователей. Задачи управления на уровне отдела относительно просты. В задачи администратора входит добавление новых пользователей, устранение простых отказов, инсталляция новых узлов и установка новых версий программного обеспечения. Операционные системы сетей отделов хорошо отработаны и разнообразны, также, как и сами сети отделов, уже давно применяющиеся и достаточно отлаженные. Такая сеть обычно использует одну или максимум две сетевые ОС. Чаще всего это сеть с выделенным сервером NetWare 3.x или Windows NT, или же одноранговая сеть, например сеть Windows for Workgroups.

Пользователи и администраторы сетей отделов вскоре осознают, что они могут улучшить эффективность своей работы путем получения доступа к информации других отделов своего предприятия. Если сотрудник, занимающийся продажами, может получить доступ к характеристикам конкретного продукта и включить их в презентацию, то эта информация будет более свежей и будет оказывать большее влияние на покупателей. Если отдел маркетинга может получить доступ к характеристикам продукта, который еще только разрабатывается инженерным отделом, то он может быстро подготовить маркетинговые материалы сразу же после окончания разработки.

Итак, 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.

Операционная система, работающая в сети кампуса, должна 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 часто предоставляют доступ и к серверам других типов, например, к факс-серверам и к серверам высокоскоростных модемов. Важным сервисом, предоставляемым операционными системами данного класса, является доступ к корпоративным базам данных, независимо от того, располагаются ли они на серверах баз данных или на миникомпьютерах.

Именно на уровне сети кампуса начинаются проблемы интеграции. В общем случае, отделы уже выбрали для себя типы компьютеров, сетевого оборудования и сетевых операционных систем. Например, инженерный отдел может использовать операционную систему UNIX и сетевое оборудование Ethernet, отдел продаж может использовать операционные среды DOS/Novell и оборудование Token Ring. Очень часто сеть кампуса соединяет разнородные компьютерные системы, в то время как сети отделов используют однотипные компьютеры.

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 компьютеров.

Пользователям корпоративных сетей требуются все те приложения и услуги, которые имеются в сетях отделов и кампусов, плюс некоторые дополнительные приложения и услуги, например, доступ к приложениям мейнфреймов и миникомпьютеров и к глобальным связям. 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 не применим для уровня предприятия. Наряду с базовыми сервисами, связанными с разделением файлов и принтеров, сетевая ОС, 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 оказались непригодными. На первый план вышли такие задачи и проблемы, которые в сетях рабочих групп, отделов и даже 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 - в сети появились многочисленные шлюзы, обеспечивающие согласованную работу различных ОС и сетевых системных приложений.

К признакам корпоративных ОС могут быть отнесены также 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 выполняется 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. В то же время хорошая поддержка универсальных приложений в Windows NT собственно и позволяет ей претендовать на место в мире корпоративных продуктов.

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 этого, такая база полезна при организации электронной почты, систем коллективной работы, службы безопасности, службы инвентаризации программного и аппаратного обеспечения сети, да и для 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 операционных системах. 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.

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 операционной системой: избирательные или мандатные права доступа, сложные процедуры аутентификации пользователей, программная шифрация.

Управление локальными ресурсами

Важнейшей функцией операционной системы является организация рационального использования всех аппаратных и программных ресурсов системы. К основным ресурсам могут быть отнесены: процессоры, память, внешние устройства, данные и программы. Располагающая одними и теми же ресурсами, но управляемая различными ОС, вычислительная система может работать с разной степенью эффективности. Поэтому знание внутренних механизмов операционной системы позволяет косвенно судить о ее эксплуатационных возможностях и характеристиках.

Управление процессами

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. *Процесс* (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Состояние процессов

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

ВЫПОЛНЕНИЕ - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

ОЖИДАНИЕ - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;

ГОТОВНОСТЬ - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, 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 операционной системе. Типичный граф состояний процесса показан на рисунке 2.1.

В состоянии ВЫПОЛНЕНИЕ в однопроцессорной системе может находиться только один процесс, а в каждом из состояний ОЖИДАНИЕ и ГОТОВНОСТЬ - несколько процессов, эти процессы образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния ГОТОВНОСТЬ, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние ВЫПОЛНЕНИЕ и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние ОЖИДАНИЯ какого-нибудь события, либо будет насильно "вытеснен" из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае процесс возвращается в состояние ГОТОВНОСТЬ. В это же состояние процесс переходит из состояния ОЖИДАНИЕ, после того, как ожидаемое событие произойдет.

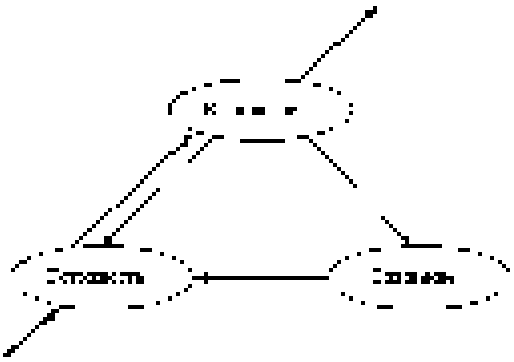


Рис. 2.1. Граф состояний процесса в многозадачной среде

Контекст и дескриптор процесса

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется *контекстом процесса*.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют*дескриптором процесса*.

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс - это значит:

1. создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;
2. включить дескриптор нового процесса в очередь готовых процессов;
3. загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

Алгоритмы планирования процессов

Планирование процессов включает в себя решение следующих задач:

1. определение момента времени для смены выполняемого процесса;
2. выбор процесса на выполнение из очереди готовых процессов;
3. переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени аппаратно (см. раздел 2.3. *"Средства аппаратной поддержки управления памятью и многозадачной среды в микропроцессорах Intel 80386, 80486 и Pentium"*).

Существует множество различных алгоритмов планирования процессов, по разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на *квантовании,* и алгоритмы, основанные на *приоритетах*.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

- процесс завершился и покинул систему,
- произошла ошибка,

- процесс перешел в состояние ОЖИДАНИЕ,
- исчерпан квант процессорного времени, отведенный данному процессу.

Процесс, который исчерпал свой квант, переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени. Граф состояний процесса, изображенный на рисунке 2.1, соответствует алгоритму планирования, основанному на квантовании.

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 ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании. По разному может быть организована очередь готовых процессов: циклически, по правилу "первый пришел - первый обслужился" (FIFO) или по правилу "последний пришел - первый обслужился" (LIFO).

Другая группа алгоритмов использует понятие "приоритет" процесса. *Приоритет* - это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.

Приоритет может выражаться целыми или дробными, положительным или отрицательным значением.Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы, либо вычисляться самой ОС по определенным правилам, он может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими.

Существует две разновидности приоритетных алгоритмов: алгоритмы, использующие относительные приоритеты, и алгоритмы, использующие абсолютные приоритеты.

В обоих случаях выбор процесса на выполнение из очереди готовых осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По разному решается проблема определения момента смены активного процесса. В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ОЖИДАНИЕ (или же произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса

прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности. На рисунке 2.2 показаны графы состояний процесса для алгоритмов с относительными (а) и абсолютными (б) приоритетами.

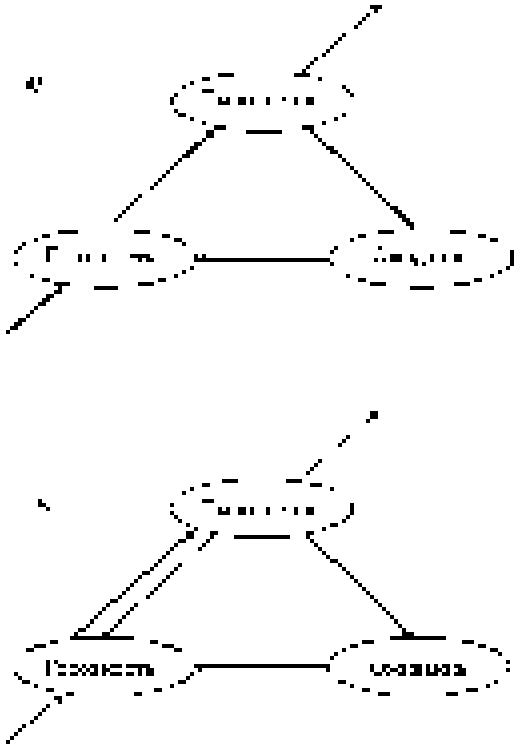


Рис. 2.2. Графы состояний процессов в системах
(а) с относительными приоритетами; (б)с абсолютными приоритетами

Во многих операционных системах алгоритмы планирования построены 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.

Вытесняющие и невытесняющие алгоритмы планирования

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - невытесняющая многозадачность - это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - вытесняющая многозадачность - это такой способ, при котором решение о переключении процессора с выполнения одного процесса на

выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Понятия preemptive и non-preemptive иногда отождествляются с понятиями приоритетных и бесприоритетных дисциплин, что совершенно неверно, а также с понятиями абсолютных и относительных приоритетов, что неверно отчасти. Вытесняющая и невытесняющая многозадачность - это более широкие понятия, чем типы приоритетности. Приоритеты задач могут как использоваться, так и не использоваться и при вытесняющих, и при невытесняющих способах планирования. Так в случае использования приоритетов дисциплина относительных приоритетов может быть отнесена к классу систем с невытесняющей многозадачностью, а дисциплина абсолютных приоритетов - к классу систем с вытесняющей многозадачностью. А бесприоритетная дисциплина планирования, основанная на выделении равных квантов времени для всех задач, относится к вытесняющим алгоритмам.

Основным различием между preemptive и non-preemptive вариантами многозадачности является степень централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в операционной системе, и программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст.

При невытесняющей многозадачности механизм планирования распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление ОС с помощью какого-либо системного вызова, а ОС формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом приоритетов) следующую задачу на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, например, на форматирование диска, пользователь не может переключиться с этой задачи на другую задачу, например, на текстовый редактор, в то время как форматирование продолжалось бы в фоновом режиме. Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу.

Поэтому разработчики приложений для non-preemptive операционной среды, возлагая на себя функции планировщика, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Например, программа

форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить "дружественное" отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая им управление. Крайним проявлением "недружественности" приложения является его зависание, которое приводит к общему краху системы. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный планирующий механизм снимет зависшую задачу с выполнения.

Однако распределение функций планировщика между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в "неудобные" для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их монопольно и уверена, что на протяжении этого периода никто другой не изменит эти данные. Существенным преимуществом non-preemptive систем является более высокая скорость переключения с задачи на задачу.

Примером эффективного использования невытесняющей многозадачности является файл-сервер NetWare, в котором, в значительной степени благодаря этому, достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.x.

Однако почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT, OS/2, VAX/VMS), реализована вытесняющая многозадачность. В последнее время дошла очередь и до ОС класса настольных систем, например, OS/2 Warp и Windows 95. Возможно в связи с этим вытесняющую многозадачность часто называют истинной многозадачностью.

Средства синхронизации и взаимодействия процессов

Проблема синхронизации

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и

активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

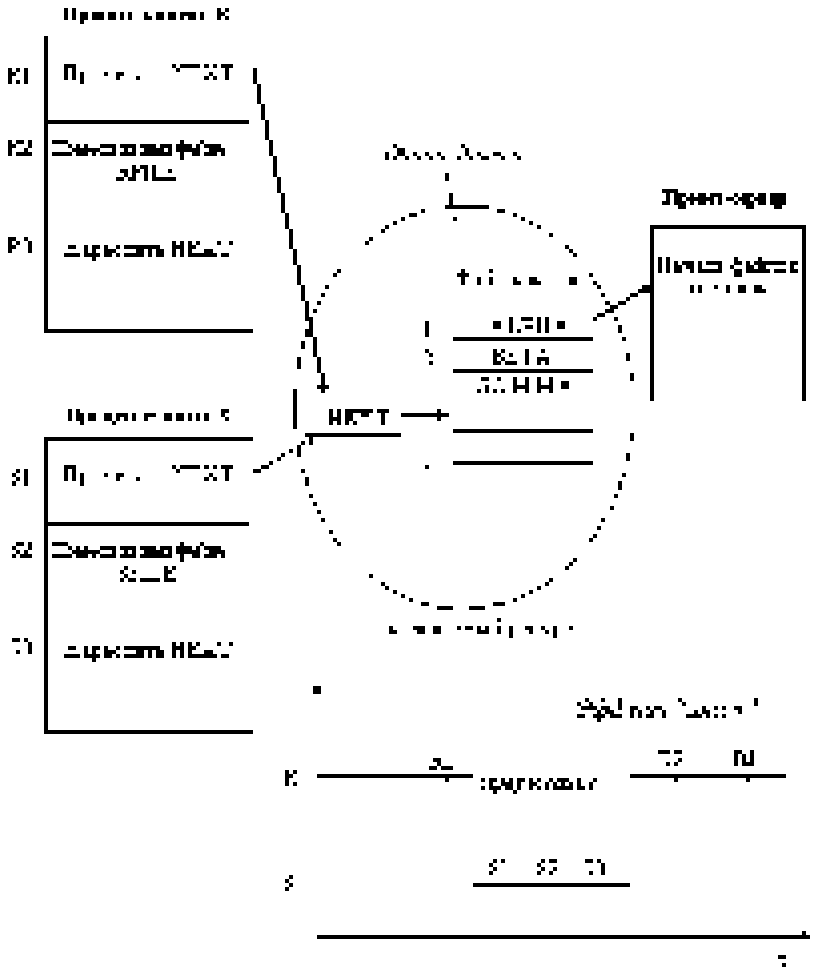


Рис. 2.3. Пример необходимости синхронизации

Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы. Рассмотрим, например (рисунок 2.3), 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. 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, значение которой для определенности предположим равным 4. Процесс запомнил это значение, но поместить имя файла не успел, так как его выполнение было прервано (например, в следствие 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 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. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций: в предыдущем примере можно представить и другое развитие событий: были потеряны файлы нескольких процессов или, напротив, не был потерян ни один файл. В данном случае все определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов является сложной задачей. Ситуации подобные той, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются *гонками*.

Критическая секция

Важным понятием синхронизации процессов является понятие "критическая секция" программы.*Критическая секция* - это часть программы, в которой осуществляется доступ к разделяемым данным. 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. Этот прием называют взаимным исключением.

Простейший способ обеспечить взаимное исключение - позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, так как опасно доверять управление системой пользовательскому процессу; он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что прерывания никогда не будут разрешены.

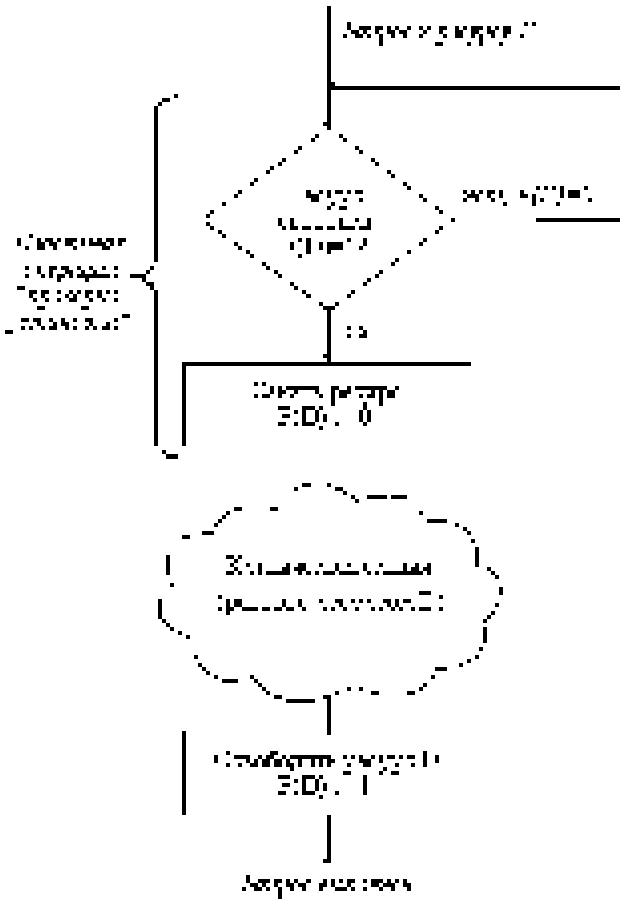


Рис. 2.4. Реализация критических секций с использованием блокирующих переменных

Другим способом 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. На рисунке 2.4 показан фрагмент алгоритма процесса, использующего для реализации взаимного исключения доступа к разделяемому ресурсу D блокирующую переменную F(D). Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D. Если он занят, то проверка циклически повторяется, если свободен, то значение переменной F(D) устанавливается в 0, и процесс входит в критическую секцию. После того, как процесс выполнит все действия с разделяемым ресурсом D, значение переменной F(D) снова устанавливается равным 1.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. Следует заметить, что операция проверки и установки блокирующей переменной должна быть неделимой. Поясним это. Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную в 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. Когда управление было возвращено первому процессу, он, считая

ресурс свободным, установил признак занятости и начал выполнять свою критическую секцию. Таким образом был нарушен принцип взаимного исключения, что потенциально может привести к нежелаемым последствиям. Во избежание таких ситуаций в системе команд машины желательно иметь единую команду "проверка-установка", или же реализовывать системными средствами соответствующие программные примитивы, которые бы запрещали прерывания на протяжении всей операции проверки и установки.

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 таких ситуаций может быть использован так называемый аппарат событий. С помощью этого средства могут решаться не только проблемы взаимного исключения, но и более общие задачи синхронизации процессов. В разных операционных системах аппарат событий реализуется по своему, но в любом случае используются системные функции аналогичного назначения, которые условно назовем WAIT(x) и POST(x), где x - идентификатор некоторого события. На рисунке 2.5 показан фрагмент алгоритма процесса, использующего эти функции. Если ресурс занят, то процесс не выполняет циклический опрос, а вызывает системную функцию WAIT(D), здесь D обозначает событие, заключающееся в освобождении ресурса D. Функция WAIT(D) переводит активный процесс в состояние ОЖИДАНИЕ и делает отметку в его дескрипторе о том, что процесс ожидает события D. Процесс, который в это время использует ресурс D, после выхода из критической секции выполняет системную функцию POST(D), в результате чего операционная система просматривает очередь ожидающих процессов и переводит процесс, ожидающий события D, в состояние ГОТОВНОСТЬ.

Обобщающее средство синхронизации процессов предложил Дейкстра, который ввел два новых примитива. В абстрактной форме эти примитивы, обозначаемые P и V, оперируют над целыми неотрицательными переменными, называемыми *семафорами*. Пусть S такой семафор. Операции определяются следующим образом:

V(S) : переменная S увеличивается на 1 одним неделимым действием; выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессам во время выполнения этой операции.

P(S) : уменьшение S на 1, если это возможно. Если S=0, то невозможно уменьшить S и остаться в области целых неотрицательных значений, в этом случае процесс, вызывающий P-операцию, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также является неделимой операцией.

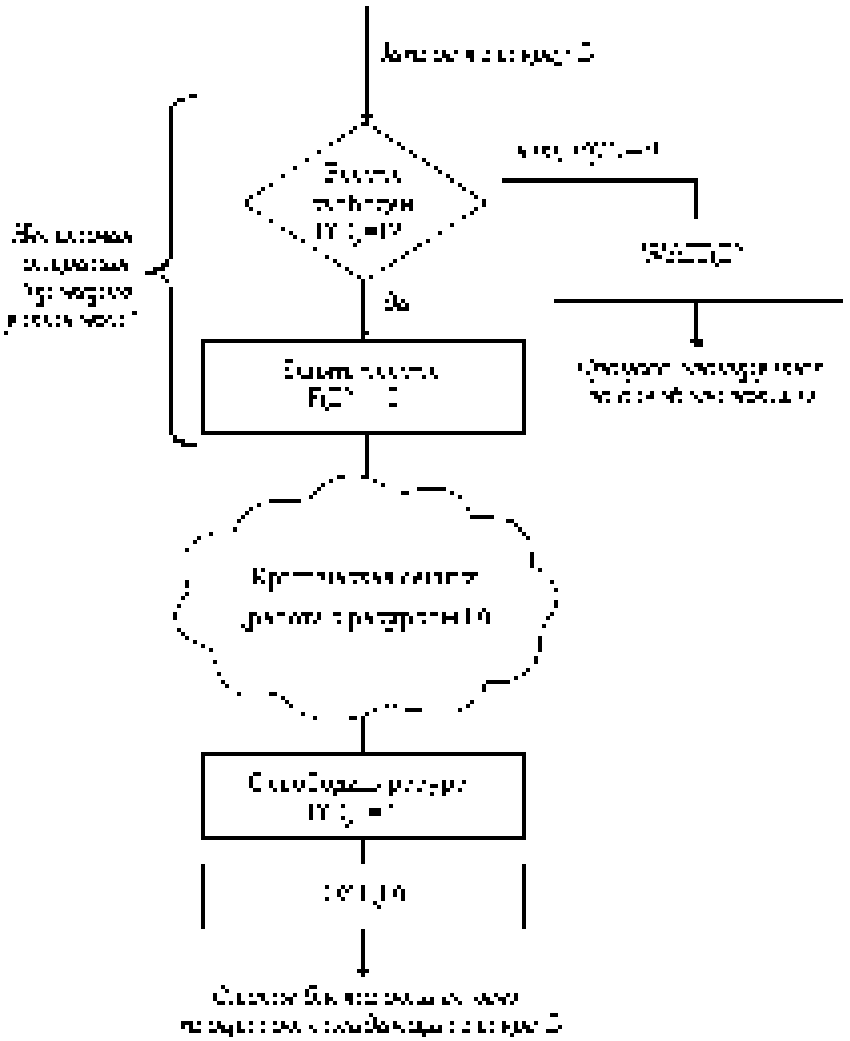


Рис. 2.5. Реализация критической секции с использованием системных функций WAIT(D) и POST(D)

В частном случае, когда семафор S может принимать только значения 0 и 1, он превращается в блокирующую переменную. Операция P заключает в себе потенциальную возможность перехода процесса, который ее выполняет, в состояние ожидания, в то время как V-операция может при некоторых обстоятельствах активизировать другой процесс, приостановленный операцией P (сравните эти операции с системными функциями WAIT и POST).

Рассмотрим использование семафоров на классическом примере взаимодействия двух процессов, выполняющихся в режиме мультипрограммирования, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула. Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. Процесс "писатель" должен приостанавливаться, когда все буфера оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, процесс "читатель" приостанавливается, когда все буферы пусты, и активизируется при появлении хотя бы одной записи.

Введем два семафора: e - число пустых буферов и f - число заполненных буферов. Предположим, что запись в буфер и считывание из буфера являются

критическими секциями (как в примере с принт-сервером в начале данного раздела). Введем также двоичный семафор b, используемый для обеспечения взаимного исключения. Тогда процессы могут быть описаны следующим образом:

```
// Глобальные переменные



#define N 256



int e = N, f = 0, b = 1;

void Writer ()



{



while(1)



{



PrepareNextRecord(); /* подготовка новой записи   */



P(e);            /* Уменьшить число свободных буферов, если они есть */



               /* в противном случае - ждать, пока они освободятся */



P(b);            /* Вход в критическую секцию  */



AddToBuffer();  /* Добавить новую запись в буфер */



V(b);            /* Выход из критической секции   */



V(f);            /* Увеличить число занятых буферов */
```

```
}

}
void Reader ()

{

while(1)

{

P(f);              /* Уменьшить число занятых буферов, если они есть */

                   /* в противном случае ждать, пока они появятся    */

P(b);              /* Вход в критическую секцию                      */

GetFromBuffer();   /* Взять запись из буфера                         */

V(b);              /* Выход из критической секции                    */

V(e);              /* Увеличить число свободных буферов              */

ProcessRecord();   /* Обработать запись                              */

}
```

```
}
```

Тупики

Приведенный выше пример поможет нам проиллюстрировать еще одну проблему синхронизации -*взаимные блокировки*, называемые также *дедлоками (deadlocks), клинчами (clinch)* или *тупиками*. Если переставить местами операции P(e) и P(b) в программе "писателе", то при некотором стечении обстоятельств эти два процесса могут взаимно заблокировать друг друга. Действительно, пусть "писатель" первым войдет в критическую секцию и обнаружит отсутствие свободных буферов; он начнет ждать, когда "читатель" возьмет очередную запись из буфера, но "читатель" не сможет этого сделать, так как для этого необходимо войти в критическую секцию, вход в которую заблокирован процессом "писателем".

Рассмотрим еще один пример тупика. Пусть двум процессам, выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например, принтер и диск. На рисунке 2.6,а показаны фрагменты соответствующих программ. И пусть после того, как процесс А занял принтер (установил блокирующую переменную), он был прерван. Управление получил процесс В, который сначала занял диск, но при выполнении следующей команды был заблокирован, так как принтер оказался уже занятым процессом А. Управление снова получил процесс А, который в соответствии со своей программой сделал попытку занять диск и был заблокирован: диск уже распределен процессу В. В таком положении процессы А и В могут находиться сколь угодно долго.

В зависимости от соотношения скоростей процессов, они могут либо совершенно независимо использовать разделяемые ресурсы (*г*), либо образовывать очереди к разделяемым ресурсам (*в*), либо взаимно блокировать друг друга (*б*). Тупиковые ситуации надо отличать от простых очередей, хотя и те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: процесс приостанавливается и ждет освобождения ресурса. Однако очередь - это нормальное явление, неотъемлемый признак высокого коэффициента использования ресурсов при случайном поступлении запросов. Она возникает тогда, когда ресурс недоступен в данный момент, но через некоторое время он освобождается, и процесс продолжает свое выполнение. Тупик же, что видно из его названия, является в некотором роде неразрешимой ситуацией.

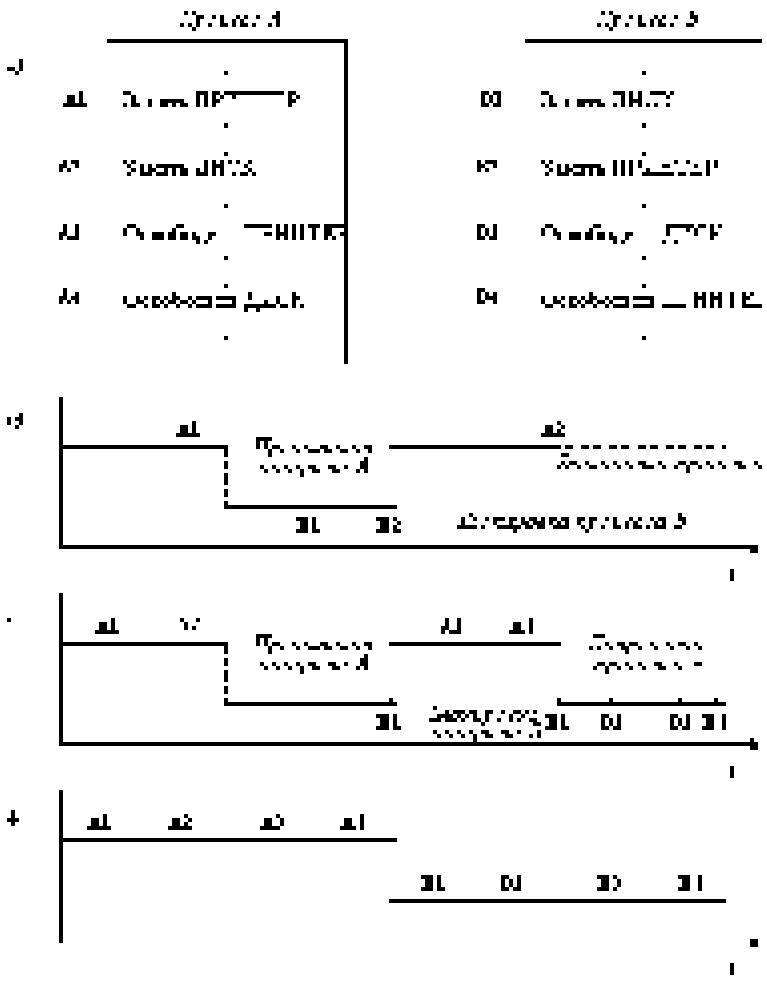


Рис. 2.6. (а) фрагменты программ A и B, разделяющих принтер и диск; (б) взаимная блокировка (клинч);(в) очередь к разделяемому диску; (г) независимое использование ресурсов

В рассмотренных примерах тупик был образован двумя процессами, но взаимно блокировать друг друга могут и большее число процессов.

Проблема тупиков включает в себя следующие задачи:

- предотвращение тупиков,
- распознавание тупиков,
- восстановление системы после тупиков.

Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов. Так, если бы в предыдущем примере процесс A и процесс B запрашивали ресурсы в одинаковой последовательности, то тупик был бы в принципе невозможен. Второй подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам, например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.

В некоторых случаях, когда тупиковая ситуация образована многими процессами, использующими много ресурсов, распознавание тупика является нетривиальной задачей. Существуют формальные, программно-реализованные методы распознавания тупиков, основанные на ведении таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ этих таблиц позволяет обнаружить взаимные блокировки.

Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные процессы. Можно снять только часть из них, при этом освобождаются ресурсы, ожидаемые остальными процессами, можно вернуть некоторые процессы в область свопинга, можно совершить "откат" некоторых процессов до так называемой контрольной точки, в которой запоминается вся информация, необходимая для восстановления выполнения программы с данного места. Контрольные точки расставляются в программе в местах, после которых возможно возникновение тупика.

Из всего вышесказанного ясно, что использовать семафоры нужно очень осторожно, так как одна незначительная ошибка может привести к останову системы. Для того, чтобы облегчить написание корректных программ, было предложено высокоуровневое средство синхронизации, называемое монитором. *Монитор* - это набор процедур, переменных и структур данных. Процессы могут вызывать процедуры монитора, но не имеют доступа к внутренним данным монитора. Мониторы имеют важное свойство, которое делает их полезными для достижения взаимного исключения: только один процесс может быть активным по отношению к монитору. Компилятор обрабатывает вызовы процедур монитора особым образом. Обычно, когда процесс вызывает процедуру монитора, то первые несколько инструкций этой процедуры проверяют, не активен ли какой-либо другой процесс по отношению к этому монитору. Если да, то вызывающий процесс приостанавливается, пока другой процесс не освободит монитор. Таким образом, исключение входа нескольких процессов в монитор реализуется не программистом, а компилятором, что делает ошибки менее вероятными.

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. Подробнее об этом смотрите в разделе "Синхронизация в распределенных системах".

Нити

Многозадачность является важнейшим свойством ОС. Для поддержки этого свойства ОС определяет и оформляет для себя те внутренние единицы работы, между которыми и будет разделяться процессор и другие ресурсы компьютера. Эти внутренние единицы работы в разных ОС носят разные названия - задача, задание, процесс, нить. В некоторых случаях сущности, обозначаемые этими понятиями, принципиально отличаются друг от друга.

Говоря о процессах, мы отмечали, что операционная система поддерживает их обособленность: у каждого процесса имеется свое виртуальное адресное пространство, каждому процессу назначаются свои ресурсы - файлы, окна, семафоры и т.д. Такая обособленность нужна для того, чтобы защитить один процесс от другого, поскольку они, совместно используя все ресурсы машины, конкурируют с друг другом. В общем случае процессы принадлежат разным пользователям, разделяющим один компьютер, и ОС берет на себя роль арбитра в спорах процессов за ресурсы.

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, 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 не блокировать полностью выполнение процесса, а 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), а понятие "процесс" в значительной степени меняет смысл.

Мультипрограммирование теперь реализуется на уровне нитей, и задача, оформленная в виде нескольких нитей в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной системе) выполнения ее отдельных частей. Например, если электронная таблица была разработана с учетом возможностей многонитевой обработки, то пользователь может запросить пересчет своего рабочего листа и одновременно продолжать заполнять таблицу. Особенно эффективно можно использовать многонитевость для выполнения распределенных приложений, например, многонитевый сервер может параллельно выполнять запросы сразу нескольких клиентов.

Нити, относящиеся к одному процессу, не настолько изолированы друг от друга, как процессы в традиционной многозадачной системе, между ними легко организовать тесное взаимодействие. Действительно, в отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все нити одного процесса всегда принадлежат одному приложению, поэтому программист, пишущий это приложение, может заранее продумать работу множества нитей процесса таким образом, чтобы они могли взаимодействовать, а не бороться за ресурсы.

В традиционных ОС понятие "нить" тождественно понятию "процесс". В действительности часто бывает желательно иметь несколько нитей, разделяющих

единое адресное пространство, но выполняющихся квазипараллельно, благодаря чему нити становятся подобными процессам (за исключением разделяемого адресного пространства).

Нити иногда называют облегченными процессами или мини-процессами. Действительно, нити во многих отношениях подобны процессам. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити, как и процессы, могут, например, порождать нити-потомки, могут переходить из состояния в состояние. Подобно традиционным процессам (то есть процессам, состоящим из одной нити), нити могут находится в одном из следующих состояний: ВЫПОЛНЕНИЕ, ОЖИДАНИЕ и ГОТОВНОСТЬ. 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.

Однако 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, потому что, во-первых, это невозможно, а во-вторых, не нужно. Все нити одного процесса всегда решают общую задачу одного пользователя, и аппарат нитей используется здесь для более быстрого решения задачи путем ее распараллеливания. При этом программисту очень важно получить в свое распоряжения удобные средства организации взаимодействия частей одной задачи. Кроме разделения адресного пространства, все нити разделяют также набор открытых файлов, таймеров, сигналов и т.п.

Итак, нити имеют собственные:

- программный счетчик,
- стек,
- регистры,
- нити-потомки,
- состояние.

Нити разделяют:

- адресное пространство,
- глобальные переменные,
- открытые файлы,
- таймеры,
- семафоры,
- статистическую информацию.

Многонитевая обработка повышает эффективность работы системы по сравнению с многозадачной обработкой. Например, в многозадачной среде Windows можно одновременно работать с электронной таблицей и текстовым редактором. 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, предоставляемых программисту, этой проблемы не возникает, и пользователь всегда имеет доступ к электронной таблице.

Широкое применение находит многонитевая обработка в распределенных системах. Смотрите об этом в разделе "Процессы и нити в распределенных системах".

Некоторые прикладные задачи легче программировать, используя параллелизм, например задачи типа "писатель-читатель", в которых одна нить выполняет запись в буфер, а другая считывает записи из него. Поскольку они разделяют общий буфер, не стоит их делать отдельными процессами. Другой пример использования нитей - это управление сигналами, такими как прерывание с клавиатуры (del или break). Вместо обработки сигнала прерывания, одна нить назначается для постоянного ожидания поступления сигналов. Таким образом, использование нитей может сократить необходимость в прерываниях пользовательского уровня. В этих примерах не столь важно параллельное выполнение, сколь важна ясность программы.

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

Управление памятью

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

Типы адресов

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса (рисунок 2.7).

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

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

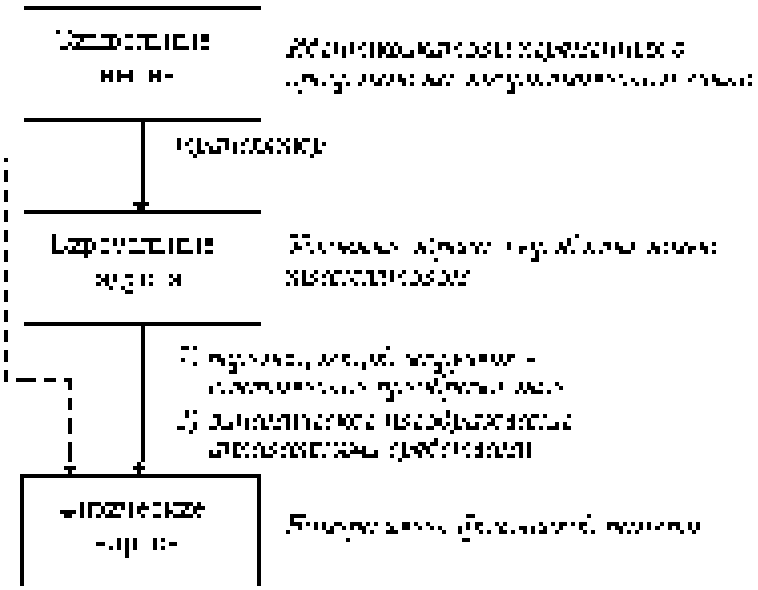


Рис. 2.7. Типы адресов

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

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

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

Методы распределения памяти без использования дискового пространства

Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого (рисунок 2.8). Начнем с последнего, более простого класса методов.

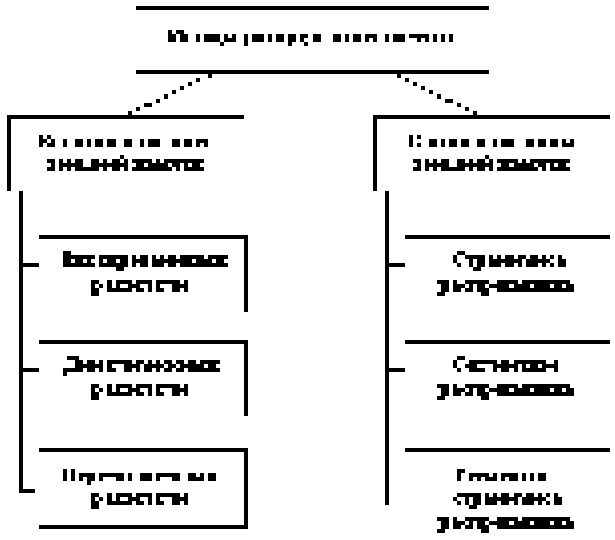


Рис. 2.8. Классификация методов распределения памяти

Распределение памяти фиксированными разделами

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено

вручную оператором во время старта системы или во время ее генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рисунок 2.9,а), либо в очередь к некоторому разделу (рисунок 2.9,б).

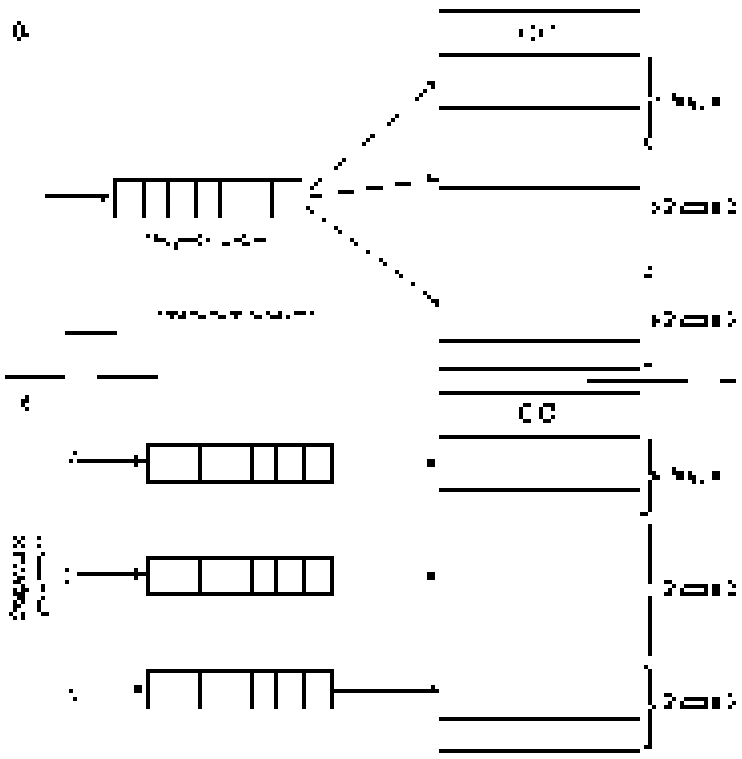


Рис. 2.9. Распределение памяти фиксированными разделами:
а - с общей очередью; б - с отдельными очередями

Подсистема управления памятью в этом случае выполняет следующие задачи:

- сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел,
- осуществляет загрузку программы и настройку адресов.

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

Распределение памяти разделами переменной величины

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

освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рисунке 2.10 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так в момент t0 в памяти находится только ОС, а к моменту t1 память разделена между 5 задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t3.

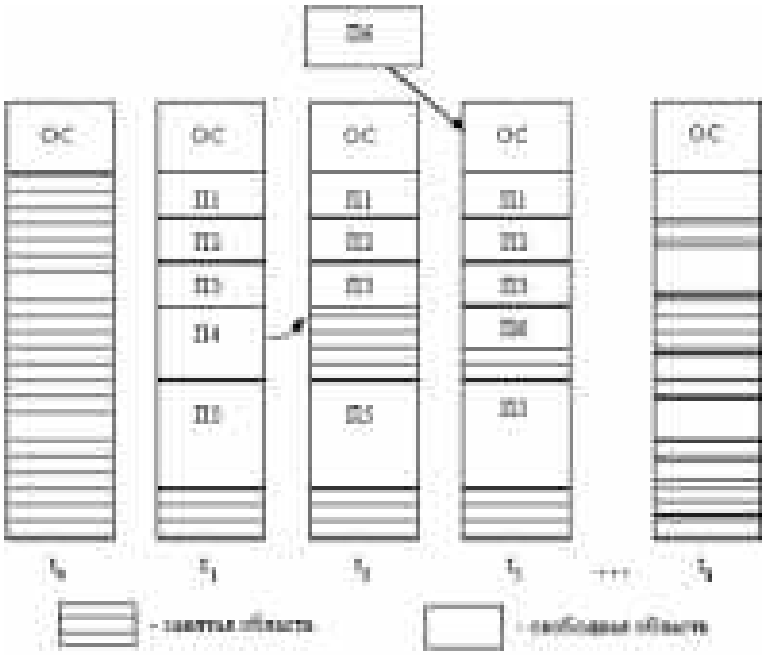


Рис. 2.10. Распределение памяти динамическими разделами

Задачами операционной системы при реализации данного метода управления памятью является:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
- при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,
- загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей,
- после завершения задачи корректировка таблиц свободных и занятых областей.

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

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам, таким, например, как "первый попавшийся раздел достаточного

размера", или "раздел, имеющий наименьший достаточный размер", или "раздел, имеющий наибольший достаточный размер". Все эти правила имеют свои преимущества и недостатки.

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

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 (рисунок 2.11). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием". Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором - реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

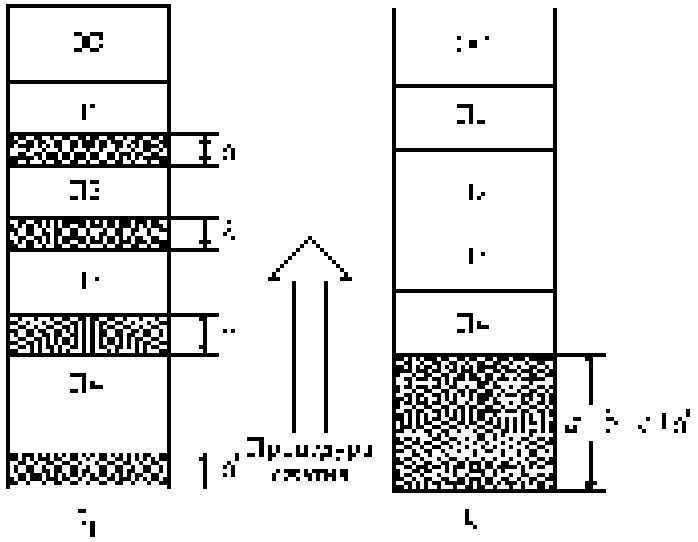


Рис. 2.11. Распределение памяти перемещаемыми разделами

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

Методы распределения памяти с использованием дискового пространства

Понятие виртуальной памяти

Уже достаточно давно пользователи столкнулись с проблемой размещения в памяти программ, размер которых превышал имеющуюся в наличии свободную память. Решением было разбиение программы на части, называемые *оверлеями*. 0-ой 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 оперативную память должен был осуществлять программист.

Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием *виртуальная память*. Виртуальным называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. Так, например, пользователю может быть предоставлена виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную оперативную память. Пользователь пишет программы так, как будто в его распоряжении имеется однородная оперативная память большого объема, но в действительности все данные, используемые программой, хранятся на одном или нескольких разнородных запоминающих устройствах, обычно на дисках, и при необходимости частями отображаются в реальную память.

Таким образом, виртуальная память - это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память; для этого виртуальная память решает следующие задачи:

- размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
- перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
- преобразует виртуальные адреса в физические.

Все эти действия выполняются *автоматически*, без участия программиста, то есть механизм виртуальной памяти является прозрачным по отношению к пользователю.

Наиболее распространенными реализациями виртуальной памяти является страничное, сегментное и странично-сегментное распределение памяти, а также свопинг.

Страничное распределение

На рисунке 2.12 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

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

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

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

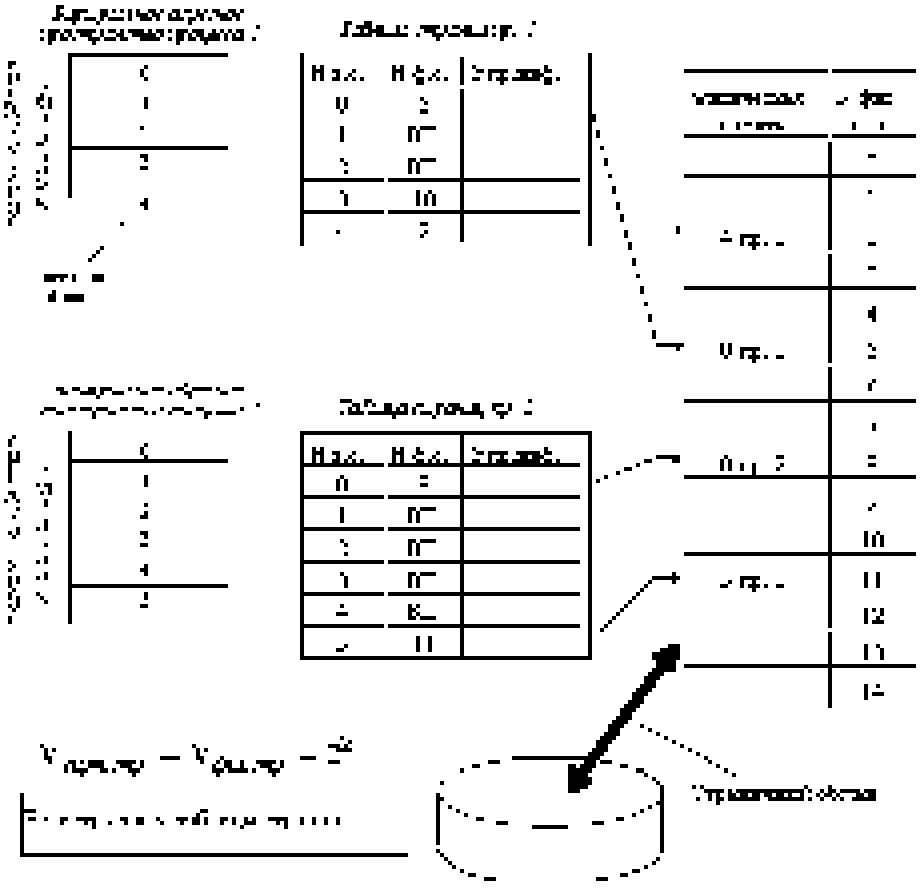


Рис. 2.12. Страничное распределение памяти

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

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

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

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

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

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

Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти (рисунок 2.13).

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

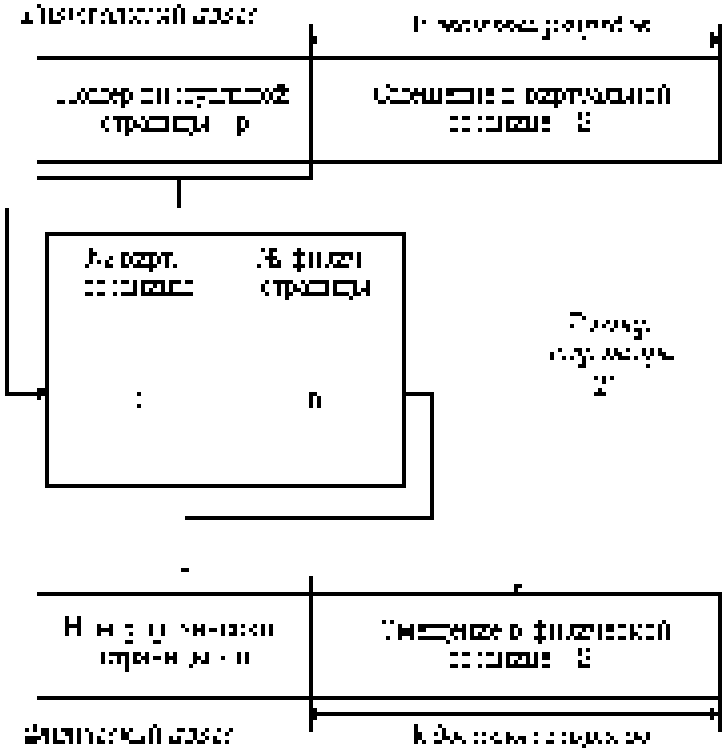


Рис. 2.13. Механизм преобразования виртуального адреса в физический при страничной организации памяти

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

1. на основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адреса) и длины записи в таблице страниц (системная константа) определяется адрес нужной записи в таблице,
2. из этой записи извлекается номер физической страницы,
3. к номеру физической страницы присоединяется смещение (младшие разряды виртуального адреса).

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

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

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

Сегментное распределение

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить

только чтение. Кроме того, разбиение программы на "осмысленные" части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности (рисунок 2.14). Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

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

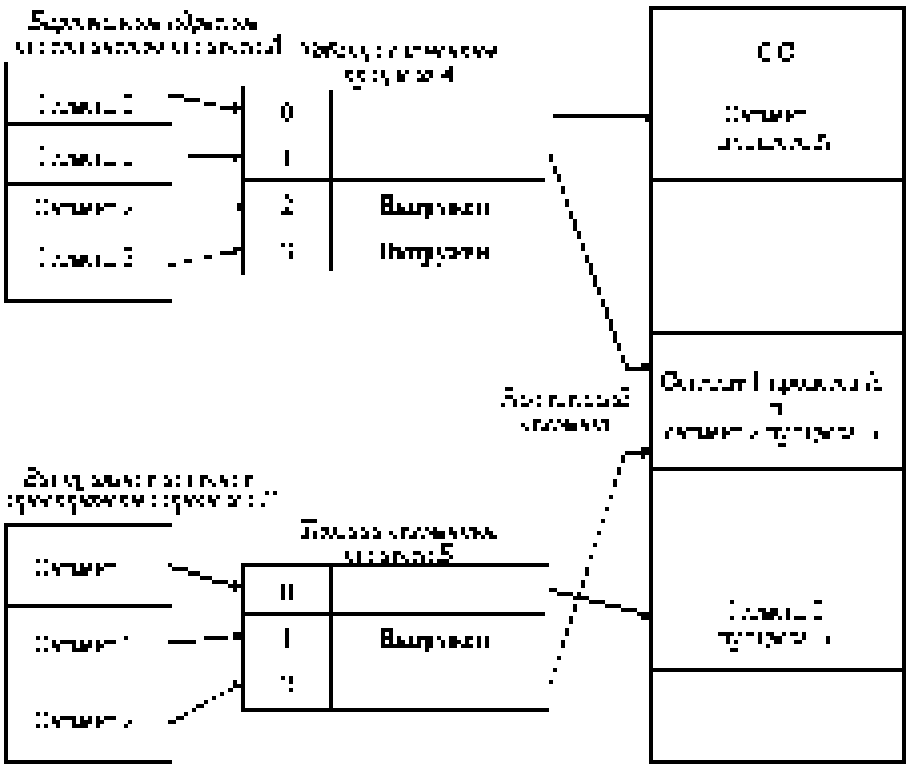


Рис. 2.14. Распределение памяти сегментами

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

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

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

Странично-сегментное распределение

Как видно из названия, данный метод представляет собой комбинацию страничного и сегментного распределения памяти и, вследствие этого, сочетает в себе достоинства обоих подходов. Виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы. Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс. На рисунке 2.15 показана схема преобразования виртуального адреса в физический для данного метода.

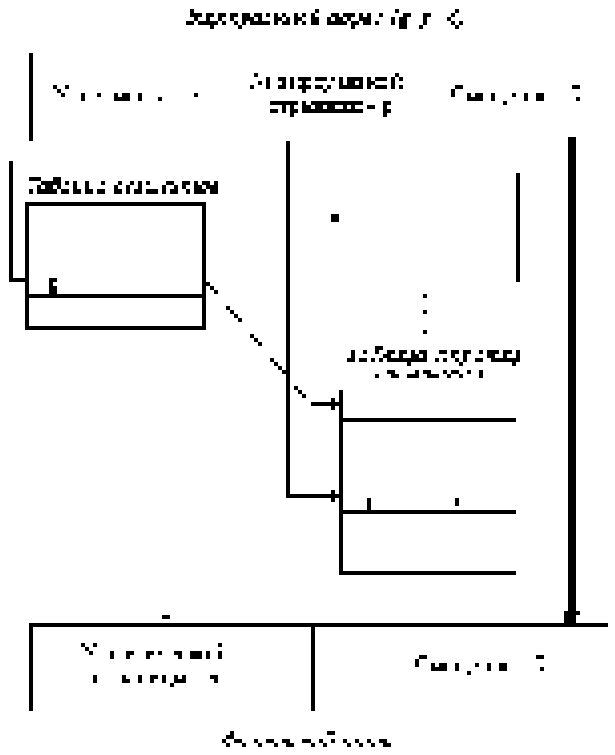


Рис. 2.15. Схема преобразования виртуального адреса в физический для сегментно-страничной организации памяти

Свопинг

Разновидностью виртуальной памяти является свопинг.

На рисунке 2.16 показан график зависимости коэффициента загрузки процессора в зависимости от числа одновременно выполняемых процессов и доли времени, проводимого этими процессами в состоянии ожидания ввода-вывода.

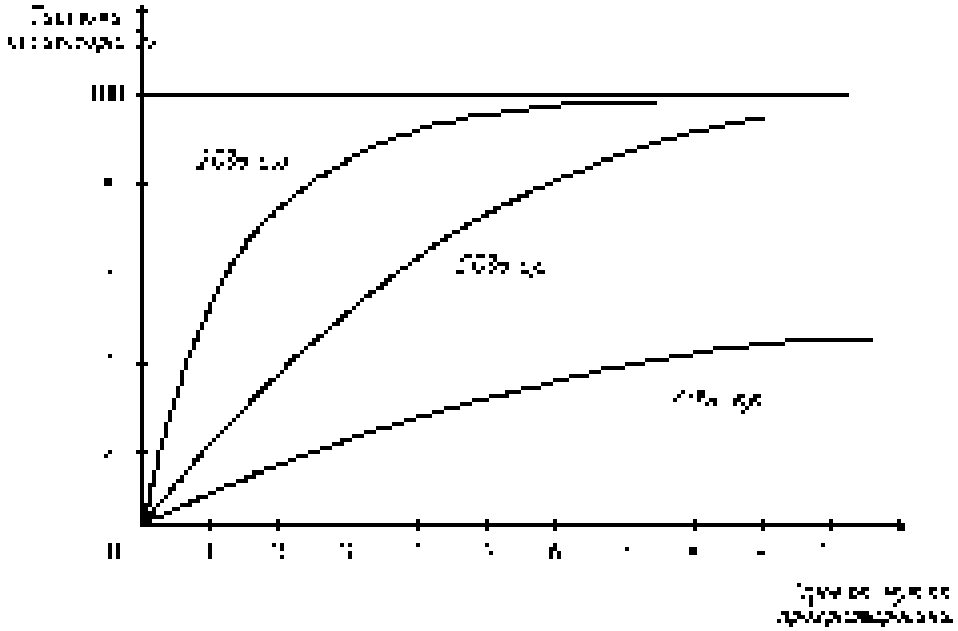


Рис. 2.16. Зависимость загрузки процессора от числа задач и интенсивности ввода-вывода

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. Необходимым условием для выполнения задачи является загрузка ее в оперативную память, объем которой ограничен. В этих условиях был предложен метод организации вычислительного процесса, называемый свопингом. В соответствии с этим методом некоторые процессы (обычно находящиеся в состоянии ожидания) временно выгружаются на диск. Планировщик операционной системы не исключает их из своего рассмотрения, и при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, этот процесс перемещается в оперативную память. Если свободного места в оперативной памяти не хватает, то выгружается другой процесс.

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 оперативной памяти. Существуют различные алгоритмы выбора процессов на загрузку и выгрузку, а также различные способы выделения оперативной и дисковой памяти загружаемому процессу.

Иерархия запоминающих устройств. Принцип кэширования данных

Память вычислительной машины представляет собой иерархию запоминающих устройств (внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит (рисунок 2.17). Пользователю хотелось бы иметь и недорогую и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

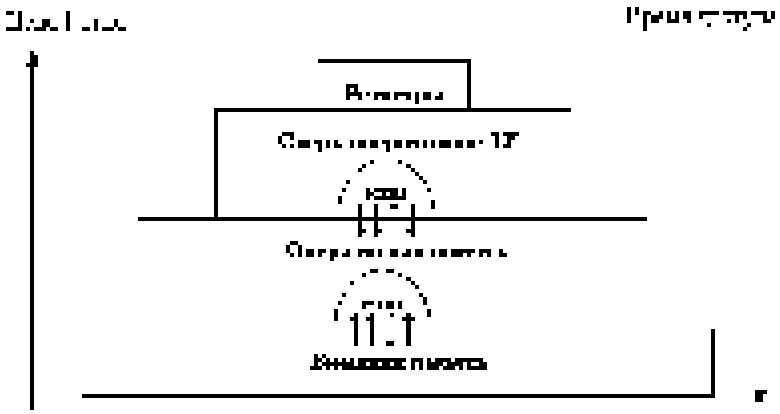


Рис. 2.17. Иерархия ЗУ

Кэш-память - это способ организации совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в "быстрое" ЗУ наиболее часто используемой информации из "медленного" ЗУ.

Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств - "быстрое" ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. Важно, что механизм кэш-памяти является прозрачным для пользователя, который не должен сообщать никакой информации об интенсивности использования данных и не должен никак участвовать в перемещении данных из ЗУ одного типа в ЗУ другого типа, все это делается автоматически системными средствами.

Рассмотрим частный случай использования кэш-памяти для уменьшения среднего времени доступа к данным, хранящимся в оперативной памяти. Для этого между процессором и оперативной памятью помещается быстрое ЗУ, называемое просто кэш-памятью (рисунок 2.18). В качестве такового может быть использована, например, ассоциативная память. Содержимое кэш-памяти представляет собой совокупность записей обо всех загруженных в нее элементах данных. Каждая запись об элементе данных включает в себя адрес, который этот элемент данных имеет в оперативной памяти, и управляющую информацию: признак модификации и признак обращения к данным за некоторый последний период времени.

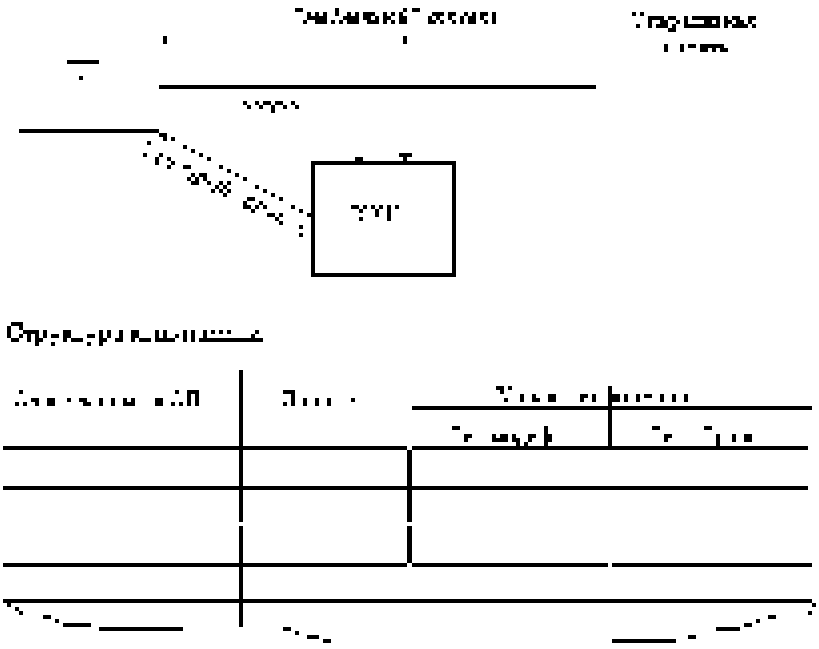


Рис. 2.18. Кэш-память

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

1. 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 "адрес в оперативной памяти", взятому из запроса.
2. Если данные обнаруживаются в кэш-памяти, то они считываются из нее, и результат передается в процессор.
3. Если нужных данных нет, то они вместе со своим адресом копируются из оперативной памяти в кэш-память, и результат выполнения запроса передается в процессор. При копировании данных может оказаться, что в кэш-памяти нет свободного места, тогда выбираются данные, 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 оперативную память. Если же эти данные не были модифицированы, то их место в кэш-памяти объявляется свободным.

На практике в кэш-память считывается не один элемент данных, к которому произошло обращение, а целый блок данных, это увеличивает вероятность так называемого "попадания в кэш", то есть нахождения нужных данных в кэш-памяти.

Покажем, как среднее время доступа к данным зависит от вероятности попадания в кэш. Пусть имеется основное запоминающие устройство со средним временем доступа к данным t1 и кэш-память, имеющая время доступа t2, очевидно, что t2<t1. Обозначим через t среднее время доступа к данным в системе с кэш-памятью, а через p -вероятность попадания в кэш. По формуле полной вероятности имеем:

```
t = t1((1 - p) + t2(p
```

Из нее видно, что среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности попадания в кэш и изменяется от среднего времени доступа в основное ЗУ (при p=0) до среднего времени доступа непосредственно в кэш-память (при p=1).

В реальных системах вероятность попадания в кэш составляет примерно 0,9. Высокое значение вероятности нахождения данных в кэш-памяти связано с наличием у данных объективных свойств: пространственной и временной локальности.

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

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

Все предыдущие рассуждения справедливы и для других пар запоминающих устройств, например, для оперативной памяти и внешней памяти. В этом случае уменьшается среднее время доступа к данным, расположенным на диске, и роль кэш-памяти выполняет буфер в оперативной памяти.

Средства аппаратной поддержки управления памятью и многозадачной среды в микропроцессорах Intel 80386, 80486 и Pentium

Процессоры Intel 80386, 80486 и Pentium с точки зрения рассматриваемых в данном разделе вопросов имеют аналогичные средства, поэтому для краткости в тексте используется термин "процессор i386", хотя вся информация этого раздела в равной степени относится к трем моделям процессоров фирмы Intel.

Процессор i386 имеет два режима работы - реальный (real mode) и защищенный (protected mode). В реальном режиме процессор i386 работает как быстрый процессор 8086 с несколько расширенным набором команд. В защищенном режиме процессор i386 может использовать все механизмы 32-х разрядной организации памяти, в том числе механизмы поддержки виртуальной памяти и механизмы переключения задач. Кроме этого, в защищенном режиме для каждой задачи процессор i386 может эмулировать 86 и 286 процессоры, которые в этом случае называются виртуальными процессорами. Таким образом, при многозадачной работе в защищенном режиме процессор i386 работает как несколько виртуальных процессоров, имеющих общую память. В отличие от реального режима, режим виртуального процессора i86, который называется в этом случае режимом V86, поддерживает страничную организацию памяти и средства многозадачности. Поэтому задачи, выполняющиеся в режиме V86, используют те же средства межзадачной защиты и защиты ОС от пользовательских задач, что и задачи, работающие в защищенном режиме i386. Однако максимальный размер виртуального адресного пространства составляет 1 Мб, как и у процессора i86.

Переключение процессора i386 из реального режима в защищенный и обратно осуществляется просто путем выполнения команды MOV, которая изменяет бит режима в одном из управляющих регистров процессора. Переход процессора в режим V86 происходит похожим образом путем изменения значения определенного бита в другом регистре процессора.

Средства поддержки сегментации памяти

Физическое адресное пространство процессора i386 составляет 4 Гбайта, что определяется 32-разрядной шиной адреса. Физическая память является линейной

с адресами от 00000000 до FFFFFFFF в шестнадцатеричном представлении. Виртуальный адрес, используемый в программе, представляет собой пару - номер сегмента и смещение внутри сегмента. Смещение хранится в соответствующем поле команды, а номер сегмента - в одном из шести сегментных регистров процессора (CS, SS, DS, ES, FS или GS), каждый из которых является 16-битным. Средства сегментации образуют верхний уровень средств управления виртуальной памятью процессора i386, а средства страничной организации - нижний уровень. Средства страничной организации могут быть как включены, так и выключены (за счет установки определенного бита в управляющем регистре процессора), и в зависимости от этого изменяется смысл преобразования виртуального адреса, которое выполняют средства сегментации. Сначала рассмотрим случай работы средств сегментации при отключенном механизме управления страницами.

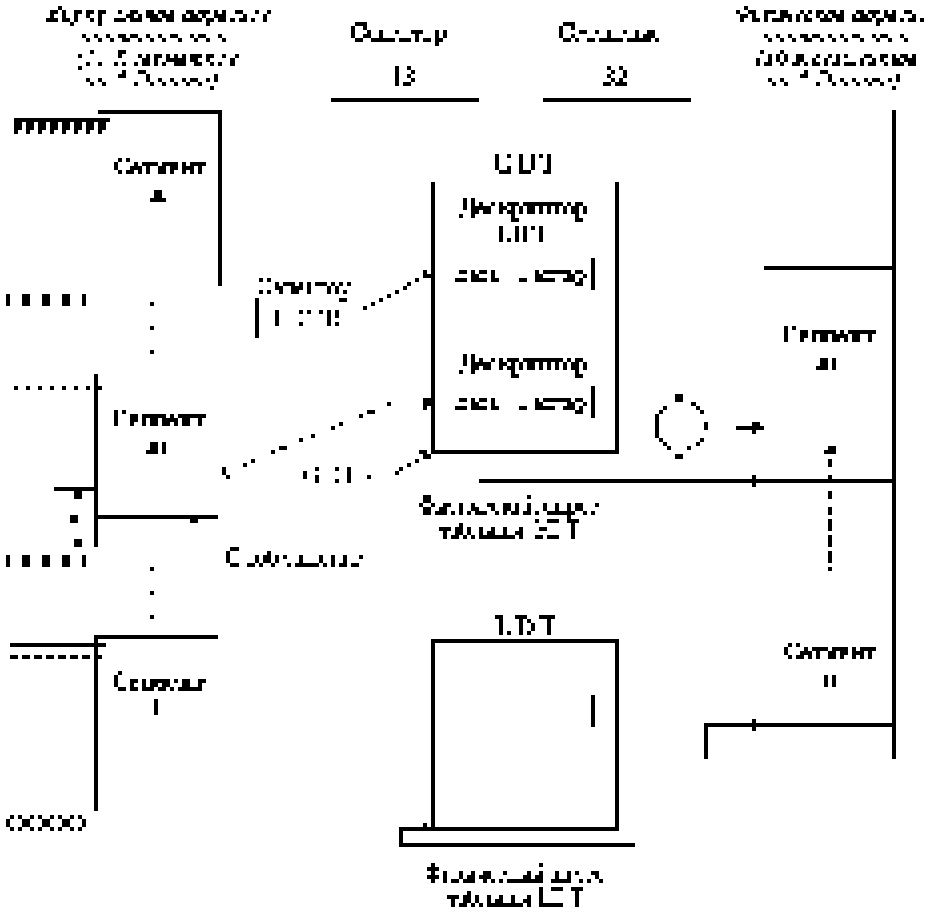


Рис. 2.19. Поддержка сегментов

На рисунке 2.19 показано виртуальное адресное пространство процессора i386 при отключенном механизме управления страницами. 32-битное смещение определяет размер виртуального сегмента в 2^{32}=4 Гбайта, а количество сегментов определяется размером поля, отведенного в сегментном регистре под номер сегмента. На рисунке 2.20,а показана структура данных в сегментном регистре. Эта структура называется селектором, так как предназначена для выбора дескриптора определенного сегмента из таблиц дескрипторов сегментов.

Дескриптор сегмента описывает все характеристики сегмента, необходимые для проверки правильности доступа к нему и нахождения его в физическом адресном пространстве. Процессор i386 поддерживает две таблицы дескрипторов сегментов - глобальную (Global Descriptor Table, GDT) и локальную (Local Descriptor Table, LDT). Глобальная таблица предназначена для описания сегментов операционной системы и сегментов межзадачного взаимодействия, то есть сегментов, которые в принципе могут использоваться всеми процессами, а локальная таблица - для сегментов отдельных задач. Таблица GDT одна, а таблиц LDT должно быть столько, сколько в системе выполняется задач. При этом активной в каждый момент времени может быть только одна из таблиц LDT.

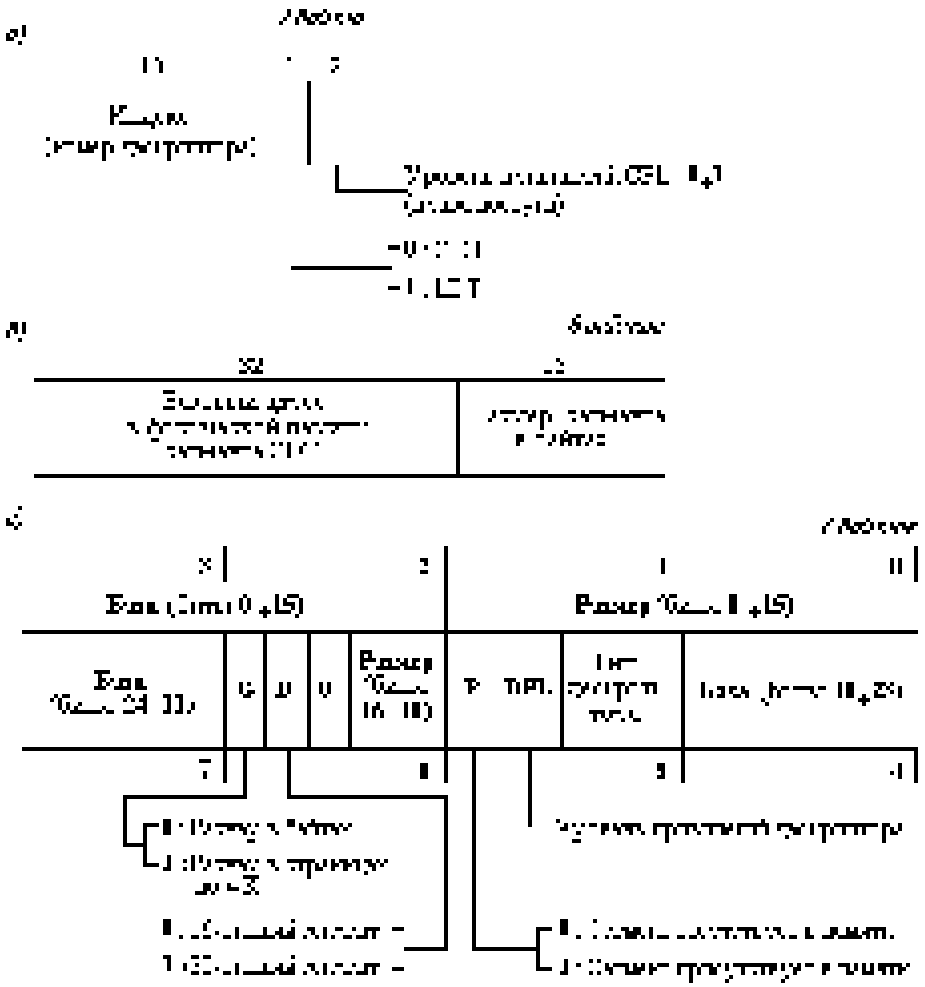


Рис. 2.20. Форматы селектора и дескрипторов данных и кода:
а - формат селектора; б - формат регистра GDTR;
в - формат дескриптора сегмента данных или кода

Из рисунка 2.20 видно, что селектор состоит из трех полей - 13-битного поля индекса (номера сегмента) в таблицах GDT и LDT, 1-битного поля - указателя типа используемой таблицы дескрипторов и двухбитного поля текущих прав доступа задачи - CPL. Разрядность поля индекса определяет максимальное число глобальных и локальных сегментов задачи - по 8K (2^{13}) сегментов каждого типа, всего 16 K. С учетом максимального размера сегмента - 4 Гбайта - каждая задача

при чисто сегментной организации виртуальной памяти работает в виртуальном адресном пространстве в 64 Тбайта.

Теперь проследим, как виртуальное пространство отображается на физическое пространство размером в 4 Гбайта при чисто сегментном механизме отображения. Итак, когда задаче необходимо получить доступ к ячейке физической памяти, то для выбора дескриптора виртуального сегмента используется значение селектора из соответствующего (в зависимости от команды и стадии ее выполнения - выборка кода команды или данных) сегментного регистра процессора. Значение поля типа таблицы указывает на то, какую таблицу нужно использовать - GDT или LDT. Рассмотрим сначала случай использования таблицы GDT. Для хранения таблиц GDT и LDT используется оперативная память (использование быстрой ассоциативной памяти процессора для хранения элементов этих таблиц рассмотрим позже). Для того, чтобы процессор смог найти в физической памяти таблицу GDT, ее полный 32-битный физический адрес (адрес начала таблицы), а также размер (поле в 16 бит) хранятся в специальном регистре процессора GDTR (рисунок 2.20, б). Каждый дескриптор в таблицах GDT и LDT имеет размер 8 байт, поэтому максимальный размер этих таблиц - 64 К (8(8 К дескрипторов). Для извлечения нужного дескриптора из таблицы процессор складывает базовый адрес таблицы GDT из регистра GDTR со сдвинутым на 3 разряда влево (умножение на 8, в соответствии с числом байтов в элементе таблицы GDT) значением поля индекса из сегментного регистра и получает физический линейный адрес нужного дескриптора в физической памяти. Таблица GDT постоянно присутствует в физической памяти, поэтому процессор извлекает по этому адресу нужный дескриптор сегмента и помещает его во внутренний (программно недоступный) регистр процессора. (Таких регистров шесть и каждый из них соответствует определенному сегментному регистру, что значительно ускоряет работу процессора).

Дескриптор виртуального сегмента (рисунок 2.20,в) состоит из нескольких полей, основными из которых являются поле базы - базового 32-разрядного физического адреса начала сегмента, поле размера сегмента и поле прав доступа к сегменту - DPL (Descriptor Privilege Level). Сначала процессор определяет правильность адреса, сравнивая смещение и размер сегмента (в случае выхода за границы сегмента происходит прерывание типа исключение - exception). Потом процессор проверяет права доступа задачи к данному сегменту, сравнивая значения полей CPL селектора и DPL дескриптора сегмента. В процессоре i386 мандатный способ определения прав доступа (называемый также механизмом колец защиты), при котором имеется несколько уровней прав доступа, и объекты какого-либо уровня имеют доступ ко всем объектам равного уровня или более низких уровней, но не имеет доступа к объектам более высоких уровней. В процессоре i386 существует четыре уровня прав доступа - от 0-го, который является самым высоким, до 3-го - самого низкого. Очевидно, что операционная система может использовать механизм уровней защиты по своему усмотрению. Однако предполагается, что нулевой уровень будет использован для ядра операционной системы, а третий уровень - для прикладных программ,

промежуточные уровни - для утилит и подсистем операционной системы, менее привилегированных, чем ядро.

Таким образом, доступ к виртуальному сегменту считается законным, если уровень прав селектора CPL выше или равен уровню прав сегмента DPL (CPL (DPL). При нарушении прав доступа происходит прерывание, как и в случае несоблюдения границ сегмента. Далее проверяется наличие сегмента в физической памяти по значению бита P дескриптора, и если сегмент отсутствует в физической памяти, то происходит прерывание. При наличии сегмента в памяти вычисляется физический линейный адрес путем сложения базы сегмента и смещения и производится доступ к элементу физической памяти по этому адресу.

В случае, когда селектор указывает на таблицу LDT, виртуальный адрес преобразуется в физический аналогичным образом, но для доступа к самой таблице LDT добавляется еще один этап, так как в процессоре регистр LDTR указывает на размещение таблицы LDT не прямо, а косвенно. Сам регистр LDTR имеет размер 16 бит и содержит селектор дескриптора таблицы GDT, который описывает расположение этой таблицы в физической памяти. Поэтому при доступе к элементу физической памяти через таблицу LDT происходит двукратное преобразование виртуального адреса в физический, причем оба раза по описанной выше схеме. Сначала по значению селектора LDTR определяется физический адрес дескриптора из таблицы GDT, описывающего начало расположения таблицы LDT в физической памяти, а затем с помощью селектора задачи вычисляется смещение в таблице LDT и определяется физический адрес нужного дескриптора. Далее процесс аналогичен преобразованию виртуального адреса с помощью таблицы GDT.

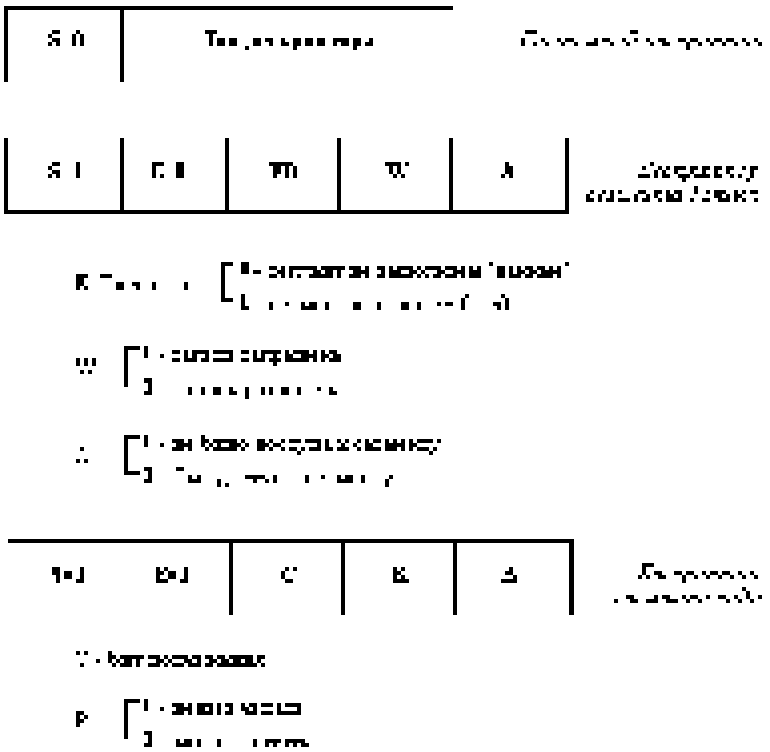


Рис. 2.21. Типы дескрипторов

Дескриптор сегмента содержит еще несколько полей. Однобитное поле G определяет единицу измерения размера сегмента, при G = 0 размер определяется в байтах, и тогда сегмент не может быть больше 64 К, а при G = 1 размер определяется в 4К-байтных страницах, при этом максимальный размер сегмента достигает указанных 4 Гбайт. Поле D определяет тип адресации сегмента: при D = 0 сегмент является 16-битным (для режима эмуляции 16-битных процессоров i86 и i286), а при D = 1 сегмент является 32-битным. Кроме этого в дескрипторе имеется поле типа сегмента, которое в свою очередь делится на несколько полей (рисунок 2.21). Поле S определяет, является ли сегмент системным (S = 1) или пользовательским (S = 0). В свою очередь пользовательские сегменты делятся на сегменты данных (E=0) и сегменты кода (E=1). Для сегмента данных определяются однобитные поля:

ED - направления распространения сегмента (ED = 0 для обычного сегмента данных, распространяющегося в сторону увеличения адресов, ED = 1 для стекового сегмента данных, распространяющегося в сторону уменьшения адресов),

W - поле разрешения записи в сегмент (при W=1 запись разрешена, при W=0 - запрещена),

A - поле доступа к сегменту (1 означает, что после очистки этого поля к сегменту было обращение по чтению или записи, это поле может использоваться операционной системой в ее стратегии замены страниц в оперативной памяти).

Для сегмента кода используются однобитные признаки:

A - имеет смысл, аналогичный полю A сегмента данных,

R - разрешает или запрещает чтение из кодового сегмента,

C - бит подчинения, разрешает или запрещает вызов данного кодового сегмента из другого кодового сегмента с более низкими правами доступа.

В процессоре i386 существует большое количество системных сегментов, к которым в частности относятся системные сегменты типа LDT, шлюзы вызова подпрограмм и задач и сегменты состояния задачи TSS.

Таким образом, для использования чисто сегментного механизма процессора i386 операционной системе необходимо сформировать таблицы GDT и LDT, загрузить их в память (для начала достаточно загрузить только таблицу GDT), загрузить указатели на эти таблицы в регистры GDTR и LDTR и выключить страничную поддержку. Если же операционная система не хочет использовать сегментную организацию виртуальной памяти, то ей достаточно создать таблицу дескрипторов из одного входа (дескриптора) и загрузить базовые значения сегмента в дескриптор. Виртуальное адресное пространство задачи будет состоять из одного сегмента длиной максимум в 4 Гбайта.

Сегментно-страничный механизм

При включенной системе управления страницами работает как описанный выше сегментный механизм, так и механизм управления страницами, однако при этом смысл работы сегментного механизма меняется. В этом случае виртуальное адресное пространство задачи имеет размер в 4 Гбайта, в котором размещаются все сегменты (рисунок 2.22). По прежнему селектор задачи определяет номер виртуального сегмента, а смещение в команде задачи - смещение внутри этого сегмента. Так как теперь все сегменты разделяют одно адресное пространство, то возможно их наложение, но процессор не контролирует такие ситуации, оставляя эту проблему операционной системе. Первый этап преобразования виртуального адреса, связанный с преобразованием смещения и селектора с использованием таблиц GDT и LDT, содержащих дескрипторы сегментов, в точности совпадает с этапом преобразования этих данных при отключенном механизме управления страницами. Все структуры данных этих таблиц такие же. Однако, если раньше дескриптор сегмента содержал его базовый адрес в физическом адресном пространстве, и при сложении его со смещением из команды программы получался линейный искомый адрес в физической памяти, то теперь дескриптор содержит базовый адрес сегмента в виртуальном адресном пространстве. Поэтому в результате его сложения со смещением получается линейный виртуальный адрес, который на втором этапе (страничном) преобразуется в номер физической страницы. Для реализации механизма управления страницами как физическое, так и виртуальное адресное пространства разбиты на страницы размером 4 К. Всего в этих адресных пространствах насчитывается 1 М страниц. Несмотря на наличие нескольких виртуальных сегментов, все виртуальное адресное пространство задачи имеет общее разбиение на страницы, так что нумерация виртуальных страниц сквозная.

Линейный виртуальный адрес содержит в своих старших 20 разрядах номер виртуальной страницы, а в младших 12 разрядах смещение внутри страницы. Для отображения виртуальной страницы в физическую достаточно построить таблицу страниц, каждый элемент которой - дескриптор виртуальной страницы - содержал бы номер соответствующей ей физической страницы и ее атрибуты. В процессоре i386 так и сделано, и структура дескриптора страницы показана на рисунке 2.23. 20-ти разрядов номера страницы достаточно для определения физического адреса начала страницы, так как при ее фиксированном размере 4 К младшие 12 разрядов этого адреса всегда равны нулю. Дескриптор страницы также содержит следующие поля, близкие по смыслу соответствующим полям дескриптора сегмента:

P - бит присутствия страницы в физической памяти,

W - бит разрешения записи в страницу,

U - бит пользователь/супервизор

A - признак того, был ли доступ к странице,

D - признак модификации содержимого страницы,

PWT и **PCD** - управляют механизмом кэширования страниц (введены, начиная с процессора i486),

AVL - резерв для нужд операционной системы (available for use).

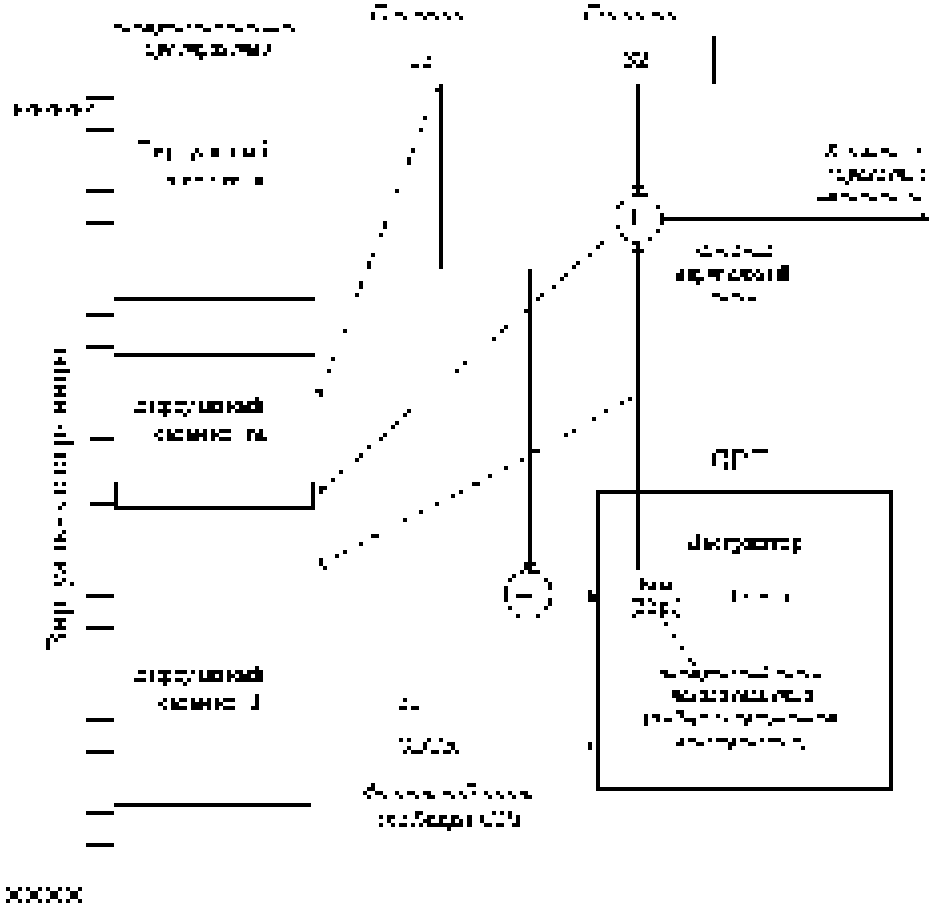


Рис. 2.22. Сегментно-страничный механизм

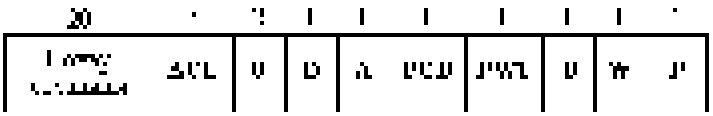


Рис. 2.23. Формат дескриптора страницы

При небольшом размере страницы процессора i386 относительно размеров адресных пространств, таблица страниц должна занимать в памяти весьма значительное место - 4 байта (1M = 4 Мбайта. Это слишком много для нынешних моделей персональных компьютеров, поэтому в процессоре i386 используется деление всей таблицы страниц на разделы по 1024 дескриптора. Размер раздела выбран так, чтобы один раздел занимал одну физическую страницу (1024 (4 байта = 4 Кбайта). Всего получается 1024 раздела (1024 (1024 = 1M). Для того, чтобы не хранить все разделы таблицы страниц одновременно в физической памяти, используется каталог разделов таблицы страниц, который использует

такие же по структуре дескрипторы страниц, что и в таблице страниц. Поэтому для хранения информации о дескрипторах 1024 разделов необходима память 4 К, т.е. одна физическая страница. Совокупность дескрипторов, описывающих состояние и характеристики виртуальных страниц разделов таблицы страниц, называется каталогом разделов или таблиц. Виртуальная страница, хранящая содержимое каталога, всегда находится в физической памяти, и номер ее физической страницы указан в специальном управляющем регистре CR3 процессора (точнее, в одном из полей этого регистра).

Преобразование линейного виртуального адреса в физический происходит следующим образом (рисунок 2.24). Поле номера виртуальной страницы (старшие 20 разрядов) делится на две равные части по 10 разрядов - поле номера раздела и поле номера страницы в разделе. С помощью номера физической страницы, хранящей каталог и смещения в этой странице, задаваемого полем номера раздела, процессор находит дескриптор виртуальной страницы раздела. В соответствии с атрибутами этого дескриптора определяются права доступа к этой странице, а также наличие ее в физической памяти. В случае ее отсутствия происходит страничное прерывание, и операционная система должна в этом случае переместить ее в память. После того, как нужная страница находится в памяти, для определения адреса элемента данных используется смещение, определяемое полем номера страницы линейного виртуального адреса.

Таким образом, при доступе к странице в процессоре используется двухуровневая схема адресации страниц, что замедляет преобразование, но позволяет использовать страничный механизм и для хранения самой таблицы страниц, и существенно уменьшить объем физической памяти для ее хранения. Для ускорения страничных преобразований в блоке управления страницами используется ассоциативная память, в которой кэшируется 32 комбинации "номер виртуальной страницы - номер физической страницы". Эта специальная кэш-память (дополнительная по отношению к 8 Кбайтному кэшу данных процессоров i486 и Pentium) значительно ускоряет преобразование адресов, так как в случае попадания в кэш длительный процесс, описанный выше, исключается.

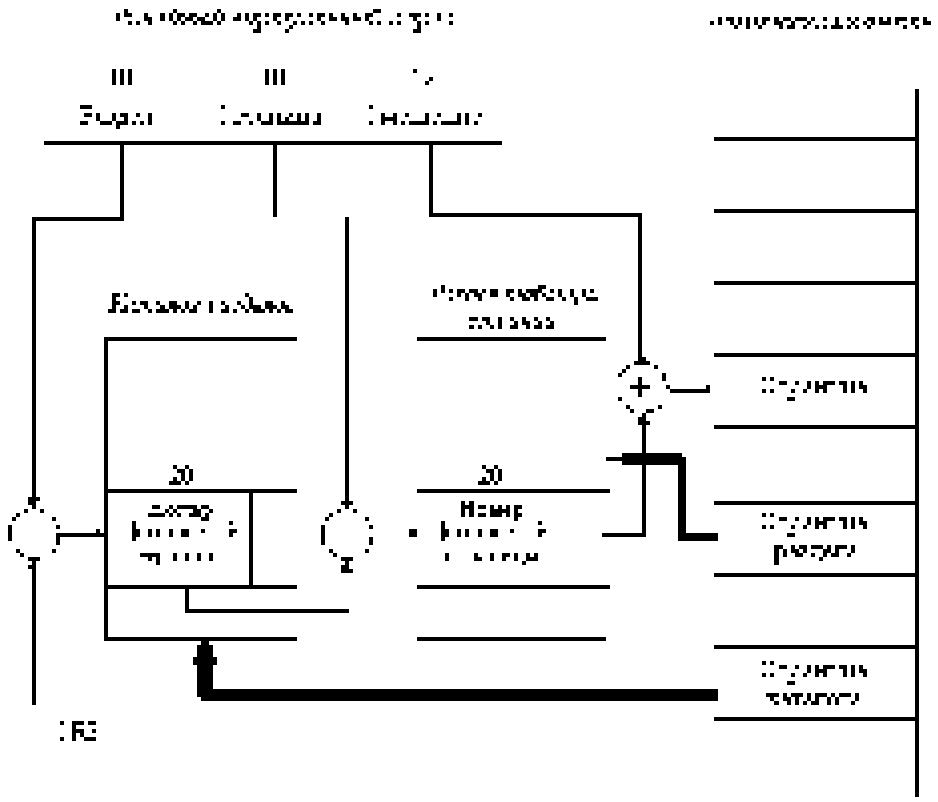


Рис. 2.24. Преобразование линейного виртуального адреса в физический адрес

Организация виртуальной памяти в процессоре i386 позволяет защитить адресные пространства различных процессов за счет двух механизмов:

1. Изоляция адресных пространств процессов в физической памяти путем назначения им различных физических страниц или сегментов (если страничный механизм отключен).
2. Защита сегментов от несанкционированного доступа с помощью привилегий четырех уровней.

Средства вызова подпрограмм и задач

Операционная система, как однозадачная, так и многозадачная, должна предоставлять задачам средства вызова подпрограмм операционной системы, библиотечных подпрограмм, а также иметь средства для запуска задач, а при многозадачной работе средства быстрого переключения с задачи на задачу. Вызов подпрограммы отличается от запуска задачи тем, что в первом случае адресное пространство остается тем же (таблица LDT остается прежней), а при вызове задачи адресное пространство полностью меняется.

Вызов подпрограмм без смены кодового сегмента в защищенном режиме процессора i386 производится аналогично вызову в реальном режиме с помощью команд JMP и CALL.

Для вызова подпрограммы, код которой находится в другом сегменте (который может принадлежать библиотеке, другой задаче или операционной системе), процессор i386 предоставляет 2 варианта вызова, причем оба используют защиту с помощью прав доступа.

Первый способ состоит в непосредственном указании в поле команды JMP или CALL селектора сегмента, содержащего код вызываемой подпрограммы, а также смещение в этом сегменте адреса начала подпрограммы.

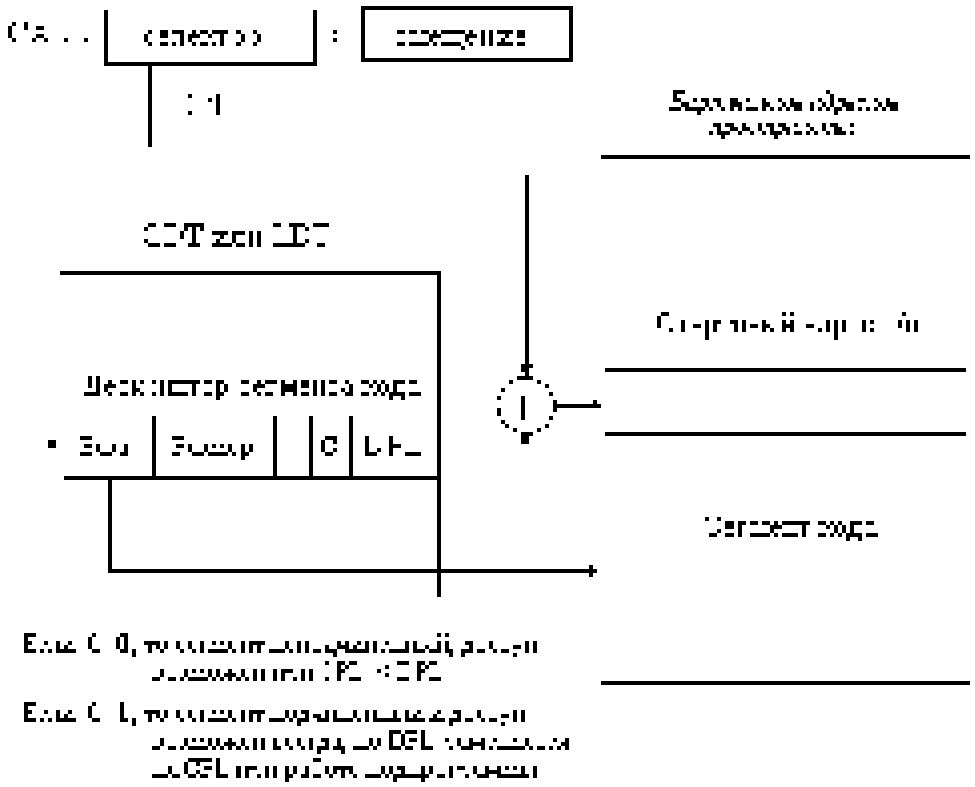


Рис. 2.25. Непосредственный вызов подпрограммы

Схема такого вызова приведена на рисунке 2.25. Здесь и далее показан только этап получения линейного адреса в виртуальном пространстве, а следующий этап (подразумевается, что механизм поддержки страниц включен) преобразования этого адреса в номер физической страницы опущен, так как он не содержит ничего специфического по сравнению с доступом к сегменту данных, рассмотренному выше. Разрешение вызова происходит в зависимости от значения поля C в дескрипторе сегмента вызываемого кода. При C=0 вызываемый сегмент не считается подчиненным, и вызов разрешается, только если уровень прав вызывающего кода не меньше уровня прав вызываемого сегмента. При C=1 вызываемый сегмент считается подчиненным и допускает вызов из кода с любым уровнем прав доступа, но при выполнении подпрограмма наделяется уровнем прав вызвавшего кода.

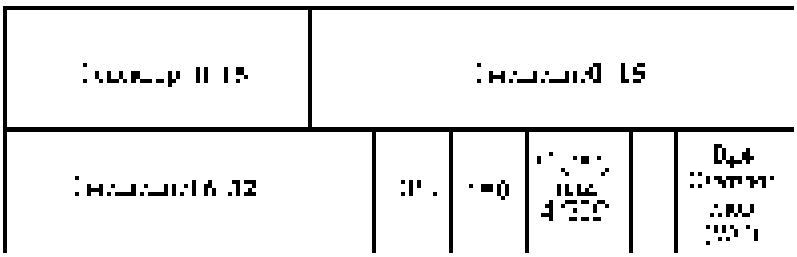


Рис. 2.26. Формат дескриптора шлюза вызова подпрограммы

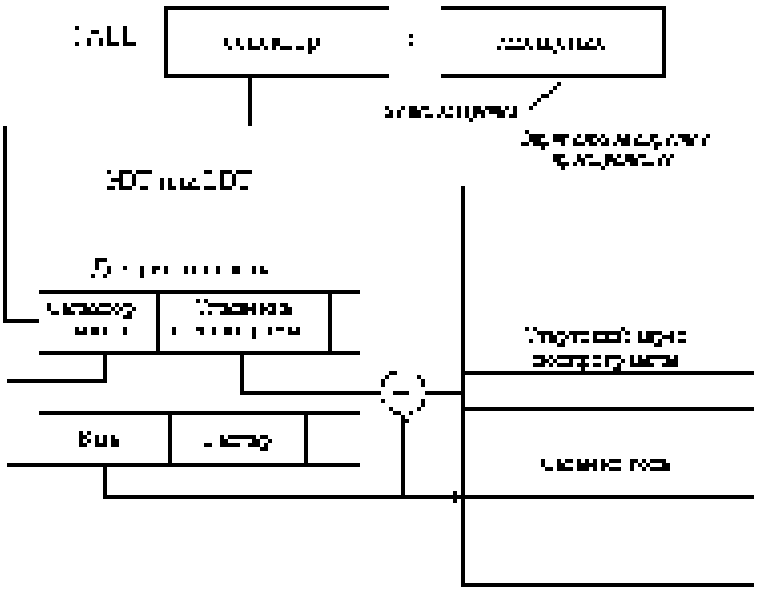


Рис. 2.27. Вызов подпрограммы через шлюз вызова

Очевидно, что первый способ непригоден для вызова функций операционной системы, имеющей обычно нулевой уровень прав, из пользовательской программы, работающей, как правило, на третьем уровне. Поэтому процессор i386 предоставляет другой способ вызова подпрограмм, основанный на том, что заранее определяется набор точек входа в привилегированные кодовые сегменты, и эти точки входа описываются с помощью специальных дескрипторов - дескрипторов шлюзов вызова подпрограмм. Этот дескриптор принадлежит к системным дескрипторам, и его структура отличается от структуры дескрипторов сегментов кода и данных (рисунок 2.26). Схема его использования приведена на рисунке 2.27. Селектор из поля команды CALL используется для указания на дескриптор шлюза вызова подпрограммы в таблицах GDT или LDT. Для использования этого дескриптора вызывающий код должен иметь не меньший уровень прав, чем дескриптор, но если дескриптор шлюза доступен, то он может указывать на дескриптор сегмента вызываемого кода, имеющий более высокий уровень, чем имеет шлюз, и вызов при этом произойдет. При определении адреса входа в вызываемом сегменте смещение из поля команды CALL не используется, а используется смещение из дескриптора шлюза, что не дает возможности задаче самой определять точку входа в защищенный кодовый сегмент.

При вызове кодов, обладающих различными уровнями привилегий, возникает проблема передачи параметров между различными стеками, так как для

надежной защиты задачи различного уровня привилегий имеют различные сегменты стеков. Селекторы этих сегментов хранятся в контексте задачи - сегменте TSS (Task State Segment). Если вызывается подпрограмма, имеющая другой уровень привилегий, то из текущего стека в стек уровня доступа вызываемого сегмента копируется столько 32-разрядных слов, сколько указано в поле счетчика слов дескриптора шлюза.

Структура сегмента TSS задачи приведена на рисунке 2.28. Контекст задачи должен содержать все данные для того, чтобы можно было восстановить выполнение прерванной в произвольный момент времени задачи, то есть значения регистров процессора, указатели на открытые файлы и некоторые другие, зависящие от операционной системы, переменные. Скорость переключения контекста в значительной степени влияет на скоростные качества многозадачной операционной системы. Процессор i386 производит аппаратное переключение контекста задачи, хранящегося в сегменте TSS. Как видно из рисунка, сегмент TSS имеет фиксированные поля, отведенные для значений регистров процессора, как универсальных, так и некоторых управляющих (например LDTR и CR3). Для описания возможностей доступа задачи к портам ввода-вывода процессор использует в защищенном режиме поле IOPL (Input/Output Privilege Level) в своем регистре EFLAGS и карту битовых полей доступа к портам. Для получения возможности выполнять команды ввода-вывода выполняемый код должен иметь уровень прав не ниже значения поля IOPL. Если же это условие соблюдается, то возможность доступа к порту с конкретным адресом определяется значением соответствующего бита в карте ввода-вывода сегмента TSS (карта состоит из 64 Кбит для описания доступа к 65536 портам).

Битовая карта ввода/вывода (БКВВ)				((8 Кбайт
Дополнительная информация ОС				
Относительный адрес БККВ			0	...0T64
0	...	0	Селектор LDT	60
0	...	0		GS 5C
0	...	0		FS 58
0	...	0		DS 54

0	...	0		SS 50
0	...	0		CS 4C
0	...	0		ES 48
EDI				44
ESI				40
EBP				3C
ESP				38
EBX				34
EDX				30
ECX				2C
EAX				28
EFLAGS				24
EIP				20
CR3				1C
0	...	0	SS уровня 2	18
ESP2				14
0	...	0	SS уровня 1	10
ESP1				C
0	...	0	SS уровня 0	8
ESP0				4
0	...	0	Селектор TSS возврата	0

Рис. 2.28. Структура сегмента TSS

Кроме этого, сегмент TSS может включать дополнительную информацию, необходимую для работы задачи и зависящую от конкретной операционной системы (например, указатели открытых файлов или указатели на именованные конвейеры сетевого обмена). Включенная в этот сегмент информация автоматически заменяется процессором при выполнении команды CALL, селектор которой указывает на дескриптор сегмента TSS в таблице GDT (дескрипторы этого типа могут быть расположены только в этой таблице). Формат дескриптора сегмента TSS аналогичен формату дескриптора сегмента данных.

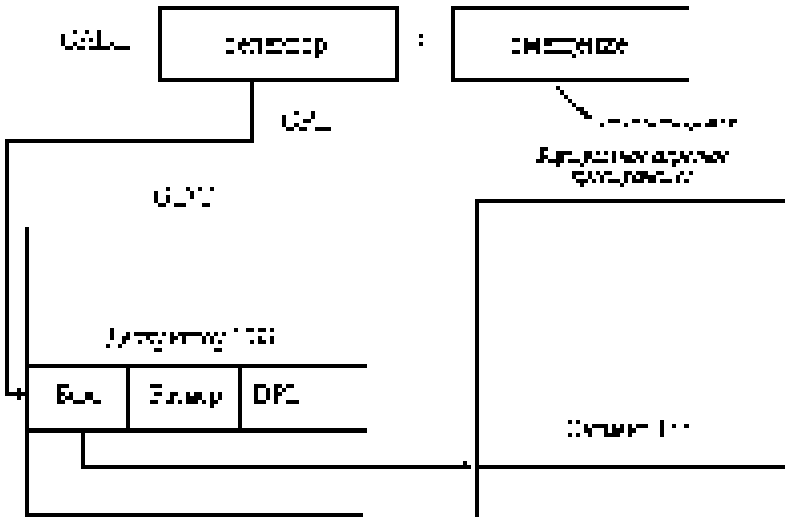


Рис. 2.29. Непосредственный вызов задачи

Как и в случае вызова подпрограмм, имеется две возможности вызова задачи - непосредственный вызов через указание селектора сегмента TSS нужной задачи в поле команды CALL и косвенный вызов через шлюз вызова задачи. Как и при вызове подпрограмм, непосредственный вызов возможен только в случае, если вызывающий код обладает уровнем привилегий, не меньшим, чем вызываемая задача. При вызове через шлюз (который может располагаться и в таблице LDT) достаточно иметь права доступа к шлюзу. Непосредственный вызов задачи показан на рисунке 2.29. При переключении задач процессор выполняет следующие действия:

1) Выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS.

2) В TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента.

3) В TR загружается селектор сегмента TSS задачи, на которую переключается процессор.

4) Из нового TSS в регистр LDTR переносится значение селектора таблицы LDT в таблице GDT задачи.

5) Восстанавливаются значения регистров процессора (из соответствующих полей нового сегмента TSS).

6) В поле селектора возврата заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к прерванной задаче в будущем.

Вызов задачи через шлюз происходит аналогично, добавляется только этап поиска дескриптора сегмента TSS по значению селектора дескриптора шлюза вызова.

Использование всех возможностей, предоставляемых процессорами Intel 80386, 80486 и Pentium, позволяет организовать операционной системе высоконадежную многозадачную среду.

Управление вводом-выводом

Одной из главных функций ОС является управление всеми устройствами ввода-вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; она также должна обеспечивать интерфейс между устройствами и остальной частью системы. В целях развития интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

Физическая организация устройств ввода-вывода

Устройства ввода-вывода делятся на два типа: *блок-ориентированные* устройства и *байт-ориентированные* устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство - диск. Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры. Однако некоторые внешние устройства не относятся ни к одному классу, например, часы, которые, с одной стороны, не адресуемы, а с другой стороны, не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать совместимые как контроллеры, так и устройства.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода (например, команд IN и OUT в процессорах i86).

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, процессор оставляет контроллер и занимается другой работой. При завершении команды контроллер организует прерывание для того, чтобы передать управление процессором операционной системе, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

Организация программного обеспечения ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска.

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 программного обеспечения ввода-вывода является обработка ошибок. Вообще говоря, 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, например, 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.

Еще один ключевой вопрос - это 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. ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие - выделенными. Диски - это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры - это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода-вывода на четыре слоя (рисунок 2.30):

- Обработка прерываний,
- Драйверы устройств,
- Независимый от устройств слой операционной системы,
- Пользовательский слой программного обеспечения.

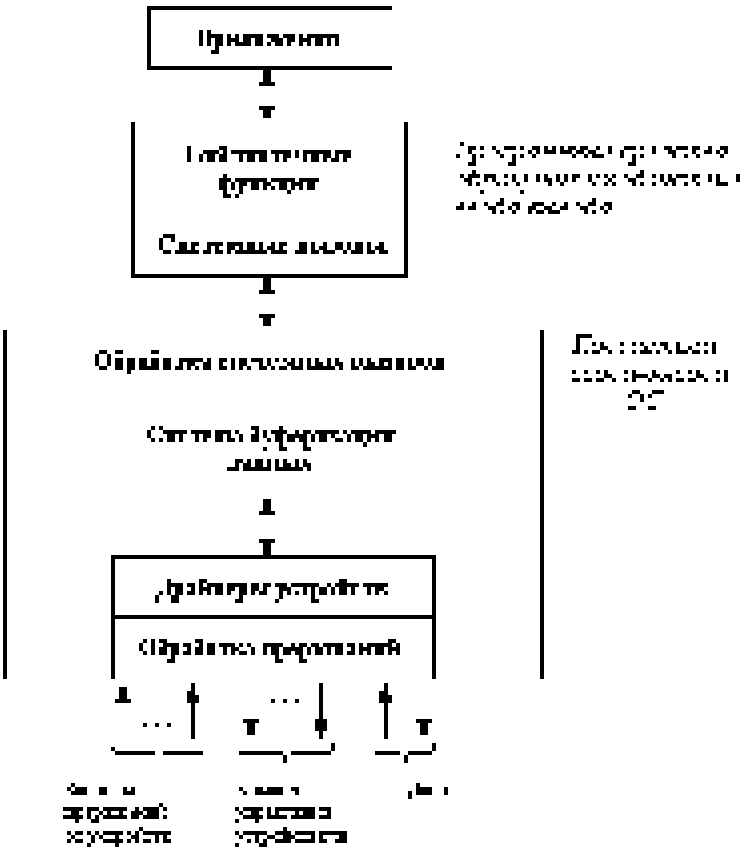


Рис. 2.30. Многоуровневая организация подсистемы ввода-вывода

Обработка прерываний

Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело. Наилучший способ состоит в разрешении процессу, инициировавшему операцию ввода-вывода, блокировать себя до завершения операции и наступления прерывания. Процесс может блокировать себя, используя, например, вызов DOWN для семафора, или вызов WAIT для переменной условия, или вызов RECEIVE для ожидания сообщения. При наступлении прерывания процедура обработки прерывания выполняет разблокирование процесса, инициировавшего операцию ввода-вывода, используя вызовы UP, SIGNAL или посылая процессу сообщение. В любом случае эффект от прерывания будет состоять в том, что ранее заблокированный процесс теперь 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 операционной системе только драйвер устройства знает о конкретных особенностях какого-либо устройства. Например, только 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, то он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди уже имеющихся запросов, и он будет выполнен, когда наступит его очередь.

Первый шаг в реализации запроса ввода-вывода, например, для диска, состоит в преобразовании его из абстрактной формы в конкретную. Для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром. Короче говоря, 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 тех пор, пока операция не завершится, и обработчик прерывания не разблокирует его. Если команда ввода-вывода выполняется быстро (например, прокрутка экрана), то драйвер ожидает ее завершения без блокирования.

Независимый от устройств слой операционной системы

Большая часть программного обеспечения ввода-вывода является независимой от устройств. Точная граница между драйверами и независимыми от устройств программами определяется системой, так как некоторые функции, которые могли бы быть реализованы независимым способом, в действительности выполнены в виде драйверов для повышения эффективности или по другим причинам.

Типичными функциями для независимого от устройств слоя 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.

Остановимся на некоторых функциях данного перечня. Верхним слоям программного обеспечения не удобно 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 битовую карту свободных блоков диска. На основании информации о наличии свободного места на диске может быть разработан алгоритм поиска свободного блока, независимый от устройства и реализуемый программным слоем, находящимся выше слоя драйверов.

Пользовательский слой программного обеспечения

Хотя большая часть программного обеспечения ввода-вывода находится внутри ОС, некоторая его часть содержится в библиотеках, связываемых с пользовательскими программами. Системные вызовы, включающие вызовы ввода-вывода, обычно делаются библиотечными процедурами. 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, форматирование

ввода или вывода выполняется библиотечными процедурами. Примером может служить функция `printf` языка C, 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 программного обеспечения ввода-вывода является подсистема спулинга (spooling). Спулинг - это способ работы с выделенными устройствами в мультипрограммной системе. Рассмотрим типичное устройство, требующее спулинга - строчный принтер. Хотя технически легко позволить каждому пользовательскому процессу открыть специальный файл, связанный с принтером, такой способ опасен из-за того, что пользовательский процесс может монополизировать принтер на произвольное время. Вместо этого создается специальный процесс - монитор, который получает 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.

Файловая система

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

Имена файлов

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими. Так в популярной 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. Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, Windows NT в своей новой файловой системе NTFS устанавливает, что имя файла может содержать до 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. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

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. Среди многих других усовершенствований одним из главных достоинств VFAT является поддержка длинных имен. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важной задачей была задача хранения длинных имен при условии, что принципиально метод хранения и структура данных на диске не должны были измениться.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющем собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уникальное имя, так, чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы. Примером такого уникального имени файла является 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 операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - 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.

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).

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, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в ОС UNIX (рисунок 2.31). Каталоги

могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рисунок 2.32).

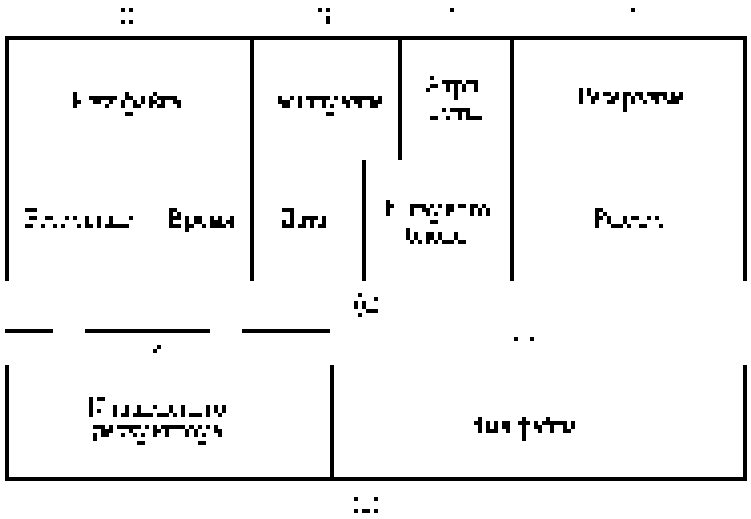


Рис. 2.31. Структура каталогов: а - структура записи каталога MS-DOS (32 байта);
б - структура записи каталога ОС UNIX

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, а в UNIX'е - сетевую. 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.

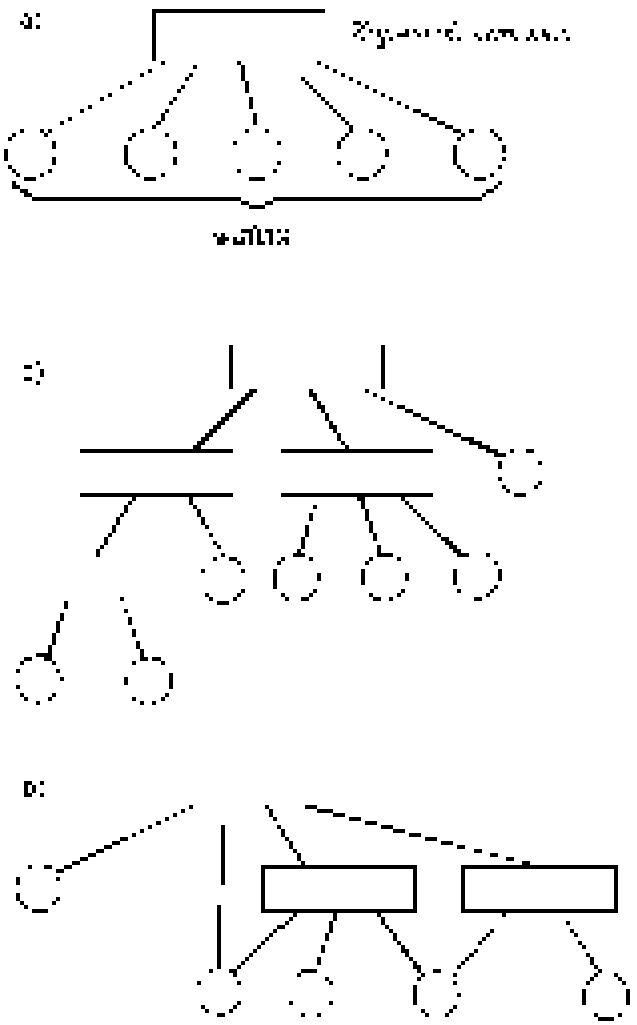


Рис. 2.32. Логическая организация файловой системы
а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)

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 внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рисунке 2.33 показаны несколько схем логической организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены 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 файловых системах ОС UNIX и MS-DOS файл имеет простейшую логическую структуру - последовательность однобайтовых записей.

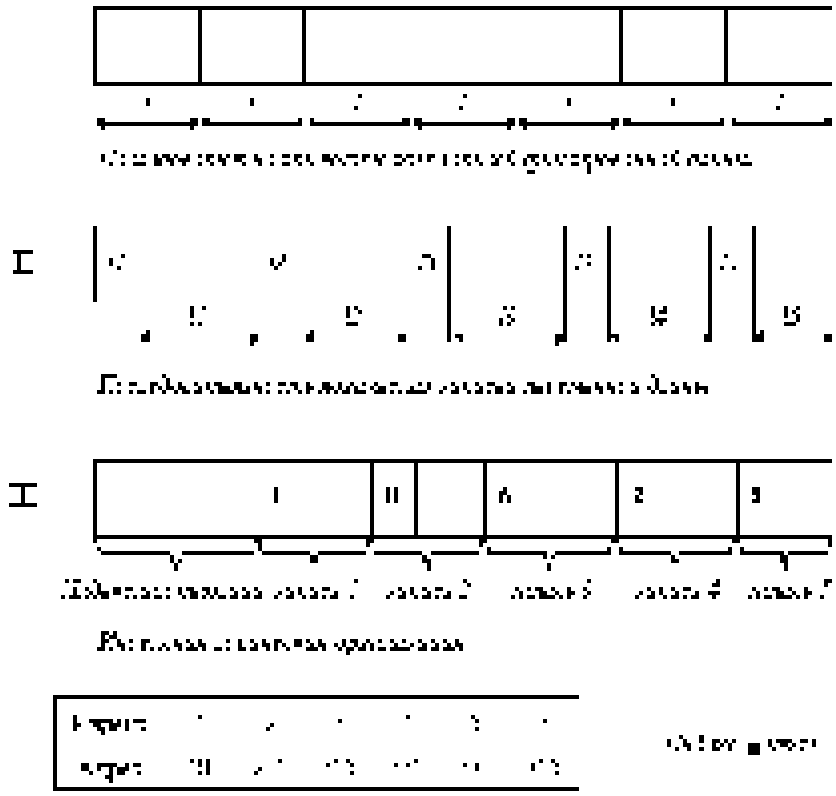


Рис. 2.33. Способы логической организации файлов

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 - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение - простейший вариант физической организации (рисунок 2.34,а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти (рисунок 2.34,б). 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 того, чтобы прочитать пятый по порядку блок 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.

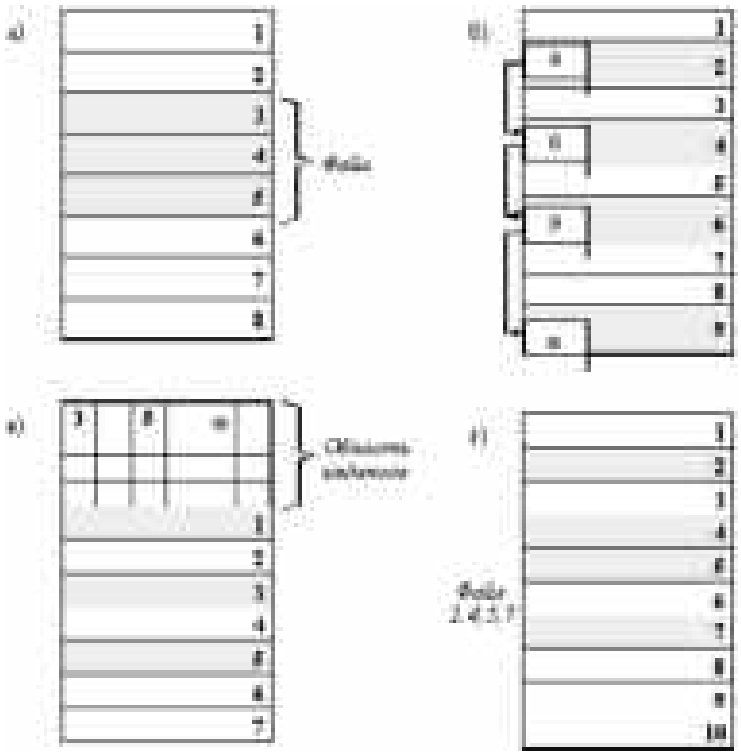


Рис. 2.34. Физическая организация файла
а - непрерывное размещение; б - связанный список блоков;
в - связанный список индексов; г - перечень номеров блоков

Популярным способом, используемым, например, в файловой системе FAT операционной системы MS-DOS, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент - индекс. Индексы располагаются в отдельной 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, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество 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 задание физического расположения файла путем простого перечисления номеров блоков, занимаемых этим файлом. ОС UNIX использует вариант данного способа, позволяющий обеспечить фиксированную

длину адреса, независимо от размера файла. Для хранения адреса файла выделено 13 полей. 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.

Права доступа к файлу

Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

- создание файла,
- уничтожение файла,
- открытие файла,
- закрытие файла,
- чтение файла,
- запись в файл,
- дополнение файла,
- поиск в файле,
- получение атрибутов файла,
- установление новых значений атрибутов,
- переименование,
- выполнение файла,
- чтение каталога,

и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рисунок 2.35). В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

Рис. 2.35. Матрица прав доступа

Различают два основных подхода к определению прав доступа:

- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

Кэширование диска

В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (диски, ленты), перехватываются промежуточным программным слоем-подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рисунок 2.36), 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.

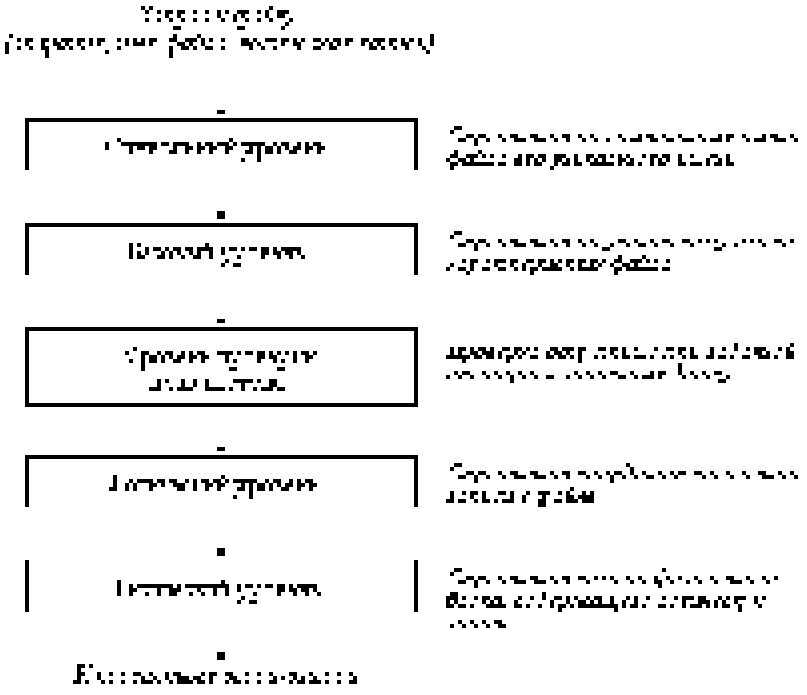


Рис. 2.36. Общая модель файловой системы

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 одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла. В файловой системе UNIX, 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 оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых файловых системах (например, HPFS) при открытии файла вместе с его характеристиками в оперативную память 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, если нет, то 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. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины l, то n-ая логическая запись имеет смещение 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.

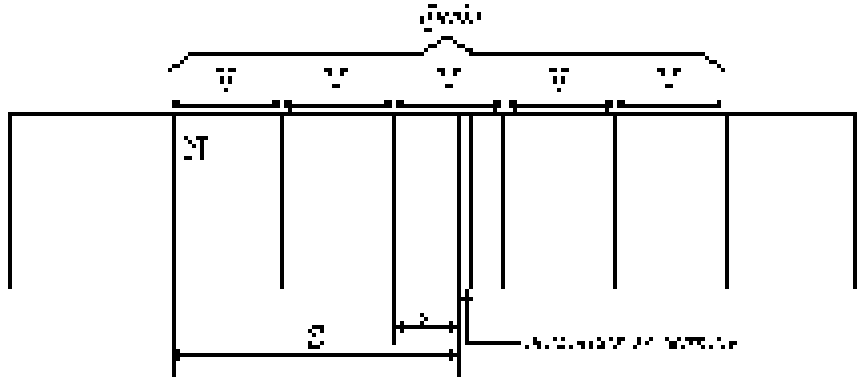


Рис. 2.37. Функции физического уровня файловой системы

Исходные данные:
V - размер блока
N - номер первого блока файла
S - смещение логической записи в файле

Требуется определить на физическом уровне:

n - номер блока, содержащего требуемую логическую запись

s - смещение логической записи в пределах блока

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

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. Рисунок 2.37 иллюстрирует работу физического уровня для

простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физического уровня решается независимо от того, как был логически организован файл.

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

Отображаемые в память файлы

По сравнению с доступом к памяти, традиционный доступ к файлам выглядит запутанным и неудобным. По этой причине некоторые ОС, начиная с MULTICS, обеспечивают отображение файлов в адресное пространство выполняемого процесса. Это выражается в появлении двух новых системных вызовов: MAP (отобразить) и UNMAP (отменить отображение). Первый вызов передает операционной системе в качестве параметров имя файла и виртуальный адрес, и операционная система отображает указанный файл в виртуальное адресное пространство по указанному адресу.

Предположим, например, что файл *f* имеет длину 64 К и отображается на область виртуального адресного пространства с начальным адресом 512 К. После этого любая машинная команда, которая читает содержимое байта по адресу 512 К, получает 0-ой байт этого файла и т.д. Очевидно, что запись по адресу 512 К + 1100 изменяет 1100 байт файла. При завершении процесса на диске остается модифицированная версия файла, как если бы он был изменен комбинацией вызовов SEEK и WRITE.

В действительности при отображении файла внутренние системные таблицы изменяются так, чтобы данный файл служил хранилищем страниц виртуальной памяти на диске. Таким образом, чтение по адресу 512 К вызывает страничный отказ, в результате чего страница 0 переносится в физическую память. Аналогично, запись по адресу 512 К + 1100 вызывает страничный отказ, в результате которого страница, содержащая этот адрес, перемещается в память, после чего осуществляется запись в память по требуемому адресу. Если эта страница вытесняется из памяти алгоритмом замены страниц, то она записывается обратно в файл в соответствующее его место. При завершении процесса все отображенные и модифицированные страницы переписываются из памяти в файл.

Отображение файлов лучше всего работает в системе, которая поддерживает сегментацию. В такой системе каждый файл может быть отображен в свой собственный сегмент, так что k-ый байт в файле является k-ым байтом сегмента. На рисунке 2.38,а изображен процесс, который имеет два сегмента-кода и данных. Предположим, что этот процесс копирует файлы. Для этого он сначала

отображает файл-источник, например, abc. Затем он создает пустой сегмент и отображает на него файл назначения, например, файл ddd.

С этого момента процесс может копировать сегмент-источник в сегмент-приемник с помощью обычного программного цикла, использующего команды пересылки в памяти типа *mov*. Никакие вызовы READ или WRITE не нужны. После выполнения копирования процесс может выполнить вызов UNMAP для удаления файла из адресного пространства, а затем завершиться. Выходной файл ddd будет существовать на диске, как если бы он был создан обычным способом.

Хотя отображение файлов исключает потребность в выполнении ввода-вывода и тем самым облегчает программирование, этот способ порождает и некоторые новые проблемы. Во-первых, для системы сложно узнать точную длину выходного файла, в данном примере ddd. Проще указать наибольший номер записанной страницы, но нет способа узнать, сколько байт в этой странице было записано. Предположим, что программа использует только страницу номер 0, и после выполнения все байты все еще установлены в значение 0 (их начальное значение). Быть может, файл состоит из 10 нулей. А может быть, он состоит из 100 нулей. Как это определить? Операционная система не может это сообщить. Все, что она может сделать, так это создать файл, длина которого равна размеру страницы.

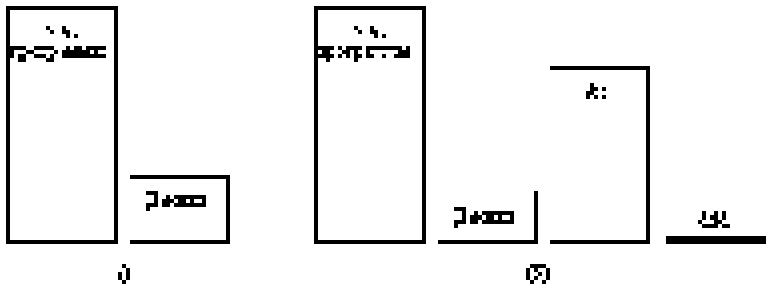


Рис. 2.38. (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) Процесс после отображения существующего файла abc в один сегмент и создания нового сегмента для файла ddd

Вторая проблема проявляется (потенциально), если один процесс отображает файл, а другой процесс открывает его для обычного файлового доступа. Если первый процесс изменяет страницу, то это изменение не будет отражено в файле на диске до тех пор, пока страница не будет вытеснена на диск. Поддержание согласованности данных файла для этих двух процессов требует от системы больших забот.

Третья проблема состоит в том, что файл может быть больше, чем сегмент, и даже больше, чем все виртуальное адресное пространство. Единственный способ ее решения состоит в реализации вызова MAP таким образом, чтобы он мог отображать не весь файл, а его часть. Хотя такая работа, очевидно, менее удобна, чем отображение целого файла.

Современные архитектуры файловых систем

Разработчики новых операционных систем 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.

Новая файловая система имеет многоуровневую структуру (рисунок 2.39), на верхнем уровне которой располагается так называемый переключатель файловых систем (в Windows 95, например, такой переключатель называется устанавливаемым диспетчером файловой системы - 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.

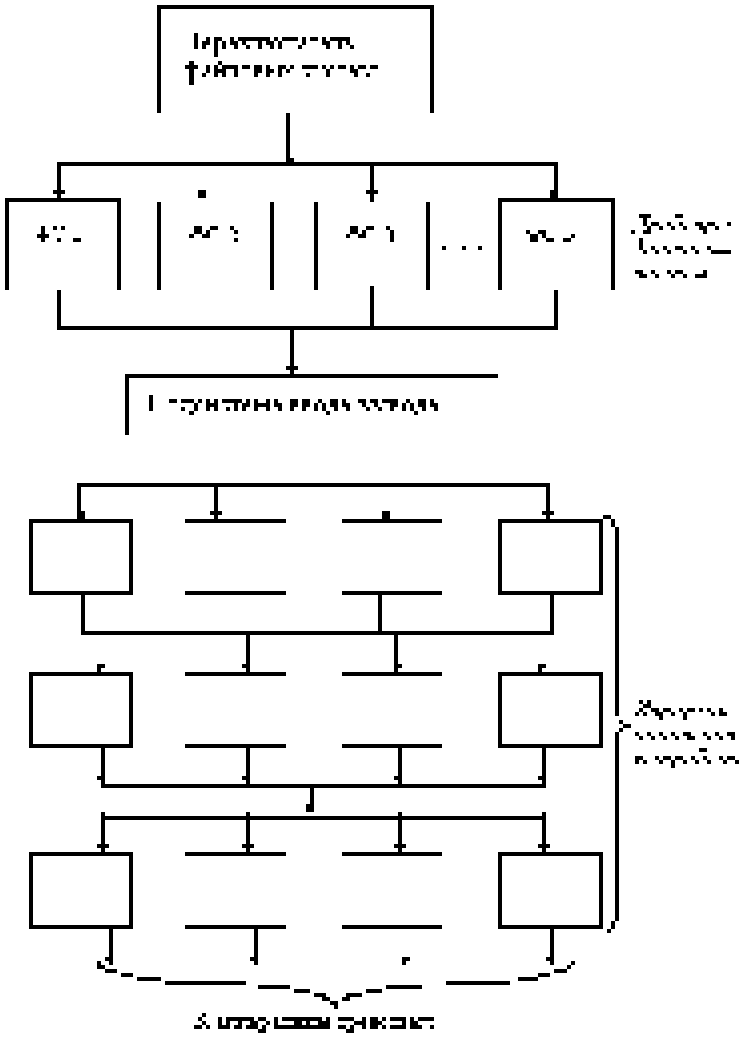


Рис. 2.39. Архитектура современной файловой системы

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. 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 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 памяти 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), драйверы портов, которые управляют конкретными адаптерами.

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.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в цепочку вызова устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса, подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше - в измененном или в неизмененном виде - на следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

Управление распределенными ресурсами

Базовые примитивы передачи сообщений в распределенных системах

Единственным по-настоящему важным отличием распределенных систем от централизованных является межпроцессная взаимосвязь. В централизованных системах связь между процессами, как правило, предполагает наличие разделяемой памяти. Типичный пример - проблема "поставщик-потребитель", в этом случае один процесс пишет в разделяемый буфер, а другой - читает из него. Даже наиболее простая форма синхронизации - семафор - требует, чтобы хотя бы одно слово (переменная самого семафора) было разделяемым. В распределенных системах нет какой бы то ни было разделяемой памяти, таким образом вся природа межпроцессных коммуникаций должна быть продумана заново.

Основой этого взаимодействия может служить только передача по сети сообщений. В самом простом случае системные средства обеспечения связи могут быть сведены к двум основным системным вызовам (примитивам), один - для посылки сообщения, другой - для получения сообщения. В дальнейшем на их базе могут быть построены более мощные средства сетевых коммуникаций, такие как распределенная файловая система или вызов удаленных процедур, которые, в свою очередь, также могут служить основой для построения других сетевых сервисов.

Несмотря на концептуальную простоту этих системных вызовов - ПОСЛАТЬ и ПОЛУЧИТЬ - существуют различные варианты их реализации, от правильного выбора которых зависит эффективность работы сети. В частности, эффективность коммуникаций в сети зависит от способа задания адреса, от того, является ли системный вызов блокирующим или неблокирующим, какие выбраны способы буферизации сообщений и насколько надежным является протокол обмена сообщениями.

Способы адресации

Для того, чтобы послать сообщение, необходимо указать адрес получателя. В очень простой сети адрес может задаваться в виде константы, но в более сложных сетях нужен и более изощренный способ адресации.

Одним из вариантов адресации на верхнем уровне является использование физических адресов сетевых адаптеров. Если в получающем компьютере выполняется только один процесс, то ядро будет знать, что делать с поступившим сообщением - передать его этому процессу. Однако, если на машине выполняется несколько процессов, то ядру не известно, какому из них предназначено сообщение, поэтому использование сетевого адреса адаптера в качестве адреса получателя приводит к очень серьезному ограничению - на каждой машине должен выполняться только один процесс.

Альтернативная адресная система использует имена назначения, состоящие из двух частей, определяющие номер машины и номер процесса. Однако адресация

типа "машина-процесс" далека от идеала, в частности, она не гибка и не прозрачна, так как пользователь должен явно задавать адрес машины-получателя. В этом случае, если в один прекрасный день машина, на которой работает сервер, отказывает, то программа, в которой жестко используется адрес сервера, не сможет работать с другим сервером, установленном на другой машине.

Другим вариантом могло бы быть назначение каждому процессу уникального адреса, который никак не связан с адресом машины. Одним из способов достижения этой цели является использование централизованного механизма распределения адресов процессов, который работает просто, как счетчик. При получении запроса на выделение адреса он просто возвращает текущее значение счетчика, а затем наращивает его на единицу. Недостатком этой схемы является то, что централизованные компоненты, подобные этому, не обеспечивают в достаточной степени расширяемость систем. Еще один метод назначения процессам уникальных идентификаторов заключается в разрешении каждому процессу выбора своего собственного идентификатора из очень большого адресного пространства, такого как пространство 64-х битных целых чисел. Вероятность выбора одного и того же числа двумя процессами является ничтожной, а система хорошо расширяется. Однако здесь имеется одна проблема: как процесс-отправитель может узнать номер машины процесса-получателя. В сети, которая поддерживает широковещательный режим (то есть в ней предусмотрен такой адрес, который принимают все сетевые адаптеры), отправитель может широковещательно передать специальный пакет, который содержит идентификатор процесса назначения. Все ядра получат эти сообщения, проверят адрес процесса и, если он совпадает с идентификатором одного из процессов этой машины, пошлют ответное сообщение "Я здесь", содержащее сетевой адрес машины.

Хотя эта схема и прозрачна, но широковещательные сообщения перегружают систему. Такой перегрузки можно избежать, выделив в сети специальную машину для отображения высокоуровневых символьных имен. При применении такой системы процессы адресуются с помощью символьных строк, и в программы вставляются эти строки, а не номера машин или процессов. Каждый раз перед первой попыткой связаться, процесс должен послать запрос специальному отображающему процессу, обычно называемому сервером имен, запрашивая номер машины, на которой работает процесс-получатель.

Совершенно иной подход - это использование специальной аппаратуры. Пусть процессы выбирают свои адреса случайно, а конструкция сетевых адаптеров позволяет хранить эти адреса. Теперь адреса процессов не обнаруживаются путем широковещательной передачи, а непосредственно указываются в кадрах, заменяя там адреса сетевых адаптеров.

Блокирующие и неблокирующие примитивы

Примитивы бывают блокирующими и неблокирующими, иногда они называются соответственно синхронными и асинхронными. При использовании

блокирующего примитива, процесс, выдавший запрос на его выполнение, приостанавливается до полного завершения примитива. Например, вызов примитива ПОЛУЧИТЬ приостанавливает вызывающий процесс до получения сообщения.

При использовании неблокирующего примитива управление возвращается вызывающему процессу немедленно, еще до того, как требуемая работа будет выполнена. Преимуществом этой схемы является параллельное выполнение вызывающего процесса и процесса передачи сообщения. Обычно в ОС имеется один из двух видов примитивов и очень редко - оба. Однако выигрыш в производительности при использовании неблокирующих примитивов компенсируется серьезным недостатком: отправитель не может модифицировать буфер сообщения, пока сообщение не отправлено, а узнать, отправлено ли сообщение, отправитель не может. Отсюда сложности в построении программ, которые передают последовательность сообщений с помощью неблокирующих примитивов.

Имеется два возможных выхода. Первое решение - это заставить ядро копировать сообщение в свой внутренний буфер, а затем разрешить процессу продолжить выполнение. С точки зрения процесса эта схема ничем не отличается от схемы блокирующего вызова: как только процесс снова получает управление, он может повторно использовать буфер.

Второе решение заключается в прерывании процесса-отправителя после отправки сообщения, чтобы проинформировать его, что буфер снова доступен. Здесь не требуется копирование, что экономит время, но прерывание пользовательского уровня делает программирование запутанным, сложным, может привести к возникновению гонок.

Вопросом, тесно связанным с блокирующими и неблокирующими вызовами, является вопрос тайм-аутов. В системе с блокирующим вызовом ПОСЛАТЬ при отсутствии ответа вызывающий процесс может заблокироваться навсегда. Для предотвращения такой ситуации в некоторых системах вызывающий процесс может задать временной интервал, в течение которого он ждет ответ. Если за это время сообщение не поступает, вызов ПОСЛАТЬ завершается с кодом ошибки.

Буферизуемые и небуферизуемые примитивы

Примитивы, которые были описаны, являются небуферизуемыми примитивами. Это означает, что вызов ПОЛУЧИТЬ сообщает ядру машины, на которой он выполняется, адрес буфера, в который следует поместить пребывающее для него сообщение.

Эта схема работает прекрасно при условии, что получатель выполняет вызов ПОЛУЧИТЬ раньше, чем отправитель выполняет вызов ПОСЛАТЬ. Вызов ПОЛУЧИТЬ сообщает ядру машины, на которой выполняется, по какому адресу должно поступить ожидаемое сообщение, и в какую область памяти необходимо

его поместить. Проблема возникает тогда, когда вызов ПОСЛАТЬ сделан раньше вызова ПОЛУЧИТЬ. Каким образом сможет узнать ядро на машине получателя, какому процессу адресовано вновь поступившее сообщение, если их несколько? И как оно узнает, куда его скопировать?

Один из вариантов - просто отказаться от сообщения, позволить отправителю взять тайм-аут и надеяться, что получатель все-таки выполнит вызов ПОЛУЧИТЬ перед повторной передачей сообщения. Этот подход не сложен в реализации, но, к сожалению, отправитель (или скорее ядро его машины) может сделать несколько таких безуспешных попыток. Еще хуже то, что после достаточно большого числа безуспешных попыток ядро отправителя может сделать неправильный вывод об аварии на машине получателя или о неправильности использованного адреса.

Второй подход к этой проблеме заключается в том, чтобы хранить хотя бы некоторое время, поступающие сообщения в ядре получателя на тот случай, что вскоре будет выполнен соответствующий вызов ПОЛУЧИТЬ. Каждый раз, когда поступает такое "неожидаемое" сообщение, включается таймер. Если заданный временной интервал истекает раньше, чем происходит соответствующий вызов ПОЛУЧИТЬ, то сообщение теряется.

Хотя этот метод и уменьшает вероятность потери сообщений, он порождает проблему хранения и управления преждевременно поступившими сообщениями. Необходимы буферы, которые следует где-то размещать, освобождать, в общем, которыми нужно управлять. Концептуально простым способом управления буферами является определение новой структуры данных, называемой почтовым ящиком.

Процесс, который заинтересован в получении сообщений, обращается к ядру с запросом о создании для него почтового ящика и сообщает адрес, по которому ему могут поступать сетевые пакеты, после чего все сообщения с данным адресом будут помещены в его почтовый ящик. Такой способ часто называют буферизуемым примитивом.

Надежные и ненадежные примитивы

Ранее подразумевалось, что когда отправитель посылает сообщение, адресат его обязательно получает. Но реально сообщения могут теряться. Предположим, что используются блокирующие примитивы. Когда отправитель посылает сообщение, то он приостанавливает свою работу до тех пор, пока сообщение не будет послано. Однако нет никаких гарантий, что после того, как он возобновит свою работу, сообщение будет доставлено адресату.

Для решения этой проблемы существует три подхода. Первый заключается в том, что система не берет на себя никаких обязательств по поводу доставки сообщений. Реализация надежного взаимодействия становится целиком заботой пользователя.

Второй подход заключается в том, что ядро принимающей машины посылает квитанцию-подтверждение ядру отправляющей машины на каждое сообщение. Посылающее ядро разблокирует пользовательский процесс только после получения этого подтверждения. Подтверждение передается от ядра к ядру. Ни отправитель, ни получатель его не видят.

Третий подход заключается в использовании ответа в качестве подтверждения в тех системах, в которых запрос всегда сопровождается ответом. Отправитель остается заблокированным до получения ответа. Если ответа нет слишком долго, то посылающее ядро может переслать запрос специальной службе предотвращения потери сообщений.

Вызов удаленных процедур (RPC)

Концепция удаленного вызова процедур

Идея вызова удаленных процедур *(Remote Procedure Call - RPC)* состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Характерными чертами вызова локальных процедур являются:

- Асимметричность, то есть одна из взаимодействующих сторон является инициатором;
- Синхронность, то есть выполнение вызывающей процедуры при останавливается с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Начнем с того, что поскольку вызывающая и вызываемая процедуры выполняются на разных машинах, то они имеют разные адресные пространства, и это создает проблемы при передаче параметров и результатов, особенно если машины не идентичны. Так как RPC не может рассчитывать на разделяемую память, то это означает, что параметры RPC не должны содержать указателей на ячейки нестековой памяти и что значения параметров должны копироваться с одного компьютера на другой. Следующим отличием RPC от локального вызова является то, что он обязательно использует нижележащую систему связи, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах. Удаленность вносит дополнительные проблемы. Выполнение вызывающей программы и вызываемой локальной

процедуры в одной машине реализуется в рамках единого процесса. Но в реализации RPC участвуют как минимум два процесса - по одному в каждой машине. В случае, если один из них аварийно завершится, могут возникнуть следующие ситуации: при аварии вызывающей процедуры удаленно вызванные процедуры станут "осиротевшими", а при аварийном завершении удаленных процедур станут "обездоленными родителями" вызывающие процедуры, которые будут безрезультатно ожидать ответа от удаленных процедур.

Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Эти и некоторые другие проблемы решает широко распространенная технология RPC, лежащая в основе многих распределенных операционных систем.

Базовые операции RPC

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычной машине, работающей автономно. Пусть это, например, будет системный вызов

```
count=read (fd,buf,nbytes);
```

где `fd` - целое число,
`buf` - массив символов,
`nbytes` - целое число.

Чтобы осуществить вызов, вызывающая процедура заталкивает параметры в стек в обратном порядке (рисунок 3.1). После того, как вызов read выполнен, он помещает возвращаемое значение в регистр, перемещает адрес возврата и возвращает управление вызывающей процедуре, которая выбирает параметры из стека, возвращая его в исходное состояние. Заметим, что в языке С параметры могут вызываться или по ссылке (by name), или по значению (by value). По отношению к вызываемой процедуре параметры-значения являются инициализируемыми локальными переменными. Вызываемая процедура может изменить их, и это не повлияет на значение оригиналов этих переменных в вызывающей процедуре.

Если в вызываемую процедуру передается указатель на переменную, то изменение значения этой переменной вызываемой процедурой влечет изменение значения этой переменной и для вызывающей процедуры. Этот факт весьма существенен для RPC.

Существует также другой механизм передачи параметров, который не используется в языке С. Он называется call-by-copy/restore и состоит в необходимости копирования вызывающей программой переменных в стек в виде

значений, а затем копирования назад после выполнения вызова поверх оригинальных значений вызывающей процедуры.

Решение о том, какой механизм передачи параметров использовать, принимается разработчиками языка. Иногда это зависит от типа передаваемых данных. В языке С, например, целые и другие скалярные данные всегда передаются по значению, а массивы - по ссылке.

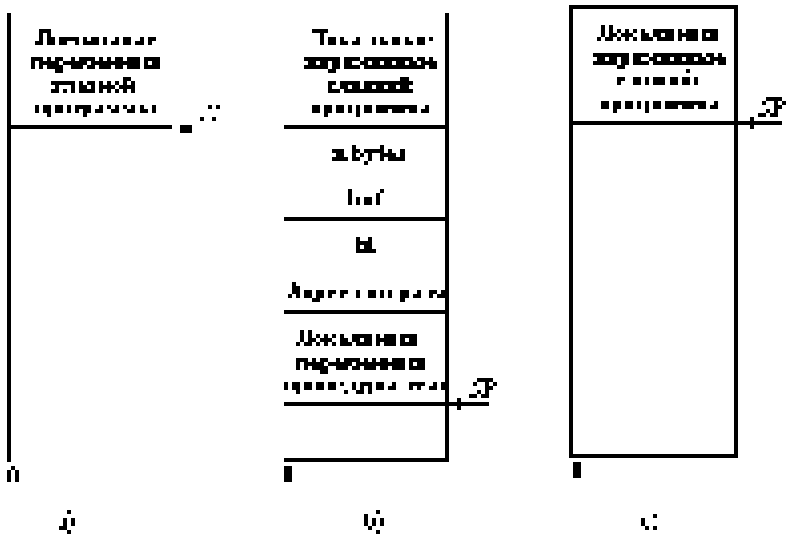


Рис. 3.1. а) Стек до выполнения вызова read; б) Стек во время выполнения процедуры; в) Стек после возврата в вызывающую программу

Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности также, как и вызов локальной процедуры. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

RPC достигает прозрачности следующим путем. Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая клиентским стабом (stub - заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности (как на рисунке 3.1), так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

Этапы выполнения RPC

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рисунком 3.2. После того, как клиентский стаб был вызван программой-клиентом, его первой задачей является заполнение буфера отправляемым сообщением. В некоторых системах клиентский стаб имеет

единственный буфер фиксированной длины, заполняемый каждый раз с самого начала при поступлении каждого нового запроса. В других системах буфер сообщения представляет собой пул буферов для отдельных полей сообщения, причем некоторые из этих буферов уже заполнены. Этот метод особенно подходит для тех случаев, когда пакет имеет формат, состоящий из большого числа полей, но значения многих из этих полей не меняются от вызова к вызову.

Затем параметры должны быть преобразованы в соответствующий формат и вставлены в буфер сообщения. К этому моменту сообщение готово к передаче, поэтому выполняется прерывание по вызову ядра.

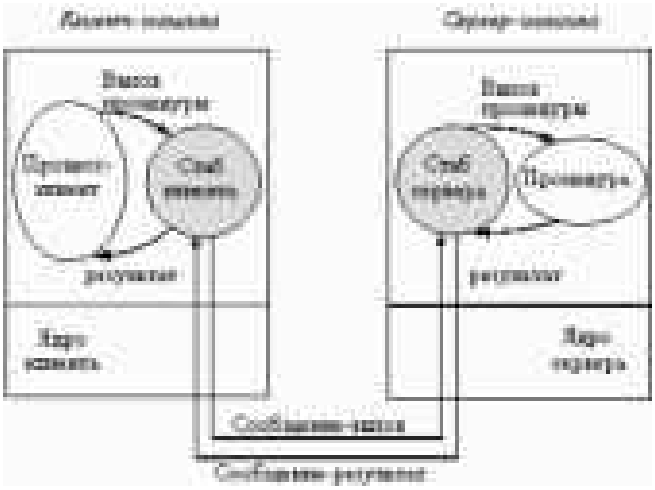


Рис. 3.2. Remote Procedure Call

Когда ядро получает управление, оно переключает контексты, сохраняет регистры процессора и карту памяти (дескрипторы страниц), устанавливает новую карту памяти, которая будет использоваться для работы в режиме ядра. Поскольку контексты ядра и пользователя различаются, ядро должно точно скопировать сообщение в свое собственное адресное пространство, так, чтобы иметь к нему доступ, запомнить адрес назначения (а, возможно, и другие поля заголовка), а также оно должно передать его сетевому интерфейсу. На этом завершается работа на клиентской стороне. Включается таймер передачи, и ядро может либо выполнять циклический опрос наличия ответа, либо передать управление планировщику, который выберет какой-либо другой процесс на выполнение. В первом случае ускоряется выполнение запроса, но отсутствует мультипрограммирование.

На стороне сервера поступающие биты помещаются принимающей аппаратурой либо во встроенный буфер, либо в оперативную память. Когда вся информация будет получена, генерируется прерывание. Обработчик прерывания проверяет правильность данных пакета и определяет, какому стабу следует их передать. Если ни один из стабов не ожидает этот пакет, обработчик должен либо поместить его в буфер, либо вообще отказаться от него. Если имеется ожидающий стаб, то сообщение копируется ему. Наконец, выполняется переключение контекстов, в результате чего восстанавливаются регистры и карта

памяти, принимая те значения, которые они имели в момент, когда стаб сделал вызов receive.

Теперь начинает работу серверный стаб. Он распаковывает параметры и помещает их соответствующим образом в стек. Когда все готово, выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

Рисунок 3.3 показывает последовательность команд, которую необходимо выполнить для каждого RPC-вызова, а рисунок 3.4 - какая доля общего времени выполнения RPC тратится на выполнение каждого их описанных 14 этапов. Исследования были проведены на мультипроцессорной рабочей станции DEC Firefly, и, хотя наличие пяти процессоров обязательно повлияло на результаты измерений, приведенная на рисунке гистограмма дает общее представление о процессе выполнения RPC.

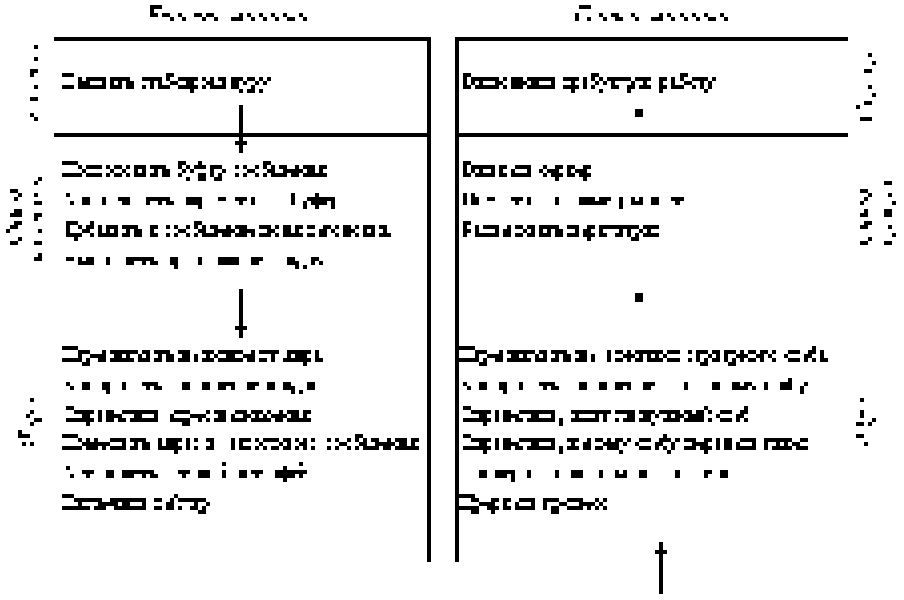


Рис. 3.3. Этапы выполнения процедуры RPC

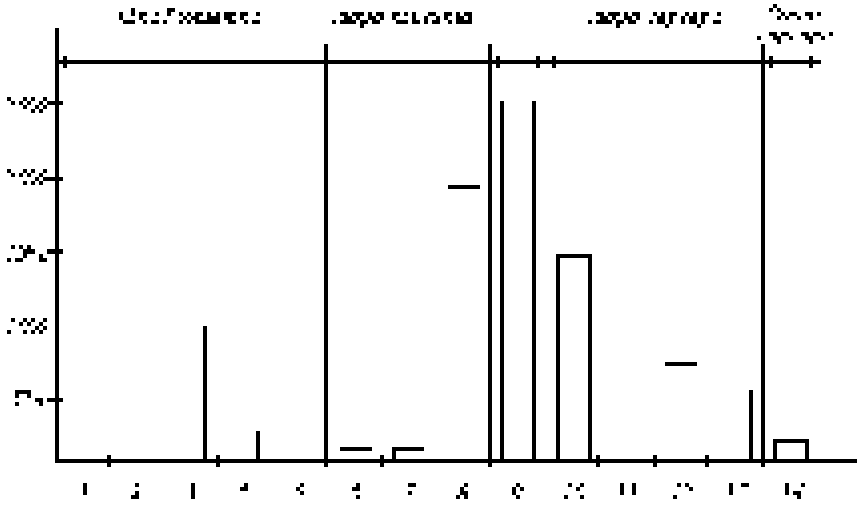


Рис. 3.4. Распределение времени между 14 этапами выполнения RPC

1. Вызов стаба

2. Подготовить буфер

3. Упаковать параметры

4. Заполнить поле заголовка

5. Вычислить контрольную сумму в сообщении

6. Прерывание к ядру

7. Очередь пакета на выполнение

8. Передача сообщения контроллеру по шине QBUS

9. Время передачи по сети Ethernet

10. Получить пакет от контроллера

11. Процедура обработки прерывания

12. Вычисление контрольной суммы

13. Переключение контекста в пространство пользователя

14. Выполнение серверного стаба

Динамическое связывание

Рассмотрим вопрос о том, как клиент задает месторасположение сервера. Одним из методов решения этой проблемы является непосредственное использование сетевого адреса сервера в клиентской программе. Недостаток такого подхода - его чрезвычайная негибкость: при перемещении сервера, или при увеличении числа серверов, или при изменении интерфейса во всех этих и многих других случаях необходимо перекомпилировать все программы, которые использовали жесткое задание адреса сервера. Для того, чтобы избежать всех этих проблем, в некоторых распределенных системах используется так называемое динамическое связывание.

Начальным моментом для динамического связывания является формальное определение (спецификация) сервера. Спецификация содержит имя файл-сервера, номер версии и список процедур-услуг, предоставляемых данным сервером для клиентов (рисунок 3.5). Для каждой процедуры дается описание ее параметров с указанием того, является ли данный параметр входным или выходным относительно сервера. Некоторые параметры могут быть

одновременно входными и выходными - например, некоторый массив, который посылается клиентом на сервер, модифицируется там, а затем возвращается обратно клиенту (операция copy/ restore).

Рис. 3.5. Спецификация сервера RPC

Формальная спецификация сервера используется в качестве исходных данных для программы-генератора стабов, которая создает как клиентские, так и серверные стабы. Затем они помещаются в соответствующие библиотеки. Когда пользовательская (клиентская) программа вызывает любую процедуру, определенную в спецификации сервера, соответствующая стаб-процедура связывается с двоичным кодом программы. Аналогично, когда компилируется сервер, с ним связываются серверные стабы.

При запуске сервера самым первым его действием является передача своего серверного интерфейса специальной программе, называемой binder'ом. Этот процесс, известный как процесс регистрации сервера, включает передачу сервером своего имени, номера версии, уникального идентификатора и описателя местонахождения сервера. Описатель системно независим и может представлять собой IP, Ethernet, X.500 или еще какой-либо адрес. Кроме того, он может содержать и другую информацию, например, относящуюся к аутентификации.

Когда клиент вызывает одну из удаленных процедур первый раз, например, read, клиентский стаб видит, что он еще не подсоединен к серверу, и посылает сообщение binder-программе с просьбой об импорте интерфейса нужной версии нужного сервера. Если такой сервер существует, то binder передает описатель и уникальный идентификатор клиентскому стабу.

Клиентский стаб при посылке сообщения с запросом использует в качестве адреса описатель. В сообщении содержатся параметры и уникальный идентификатор, который ядро сервера использует для того, чтобы направить поступившее сообщение в нужный сервер в случае, если их несколько на этой машине.

Этот метод, заключающийся в импорте/экспорте интерфейсов, обладает высокой гибкостью. Например, может быть несколько серверов, поддерживающих один и тот же интерфейс, и клиенты распределяются по серверам случайным образом. В рамках этого метода становится возможным периодический опрос серверов, анализ их работоспособности и, в случае отказа, автоматическое отключение, что повышает общую отказоустойчивость системы. Этот метод может также поддерживать аутентификацию клиента. Например, сервер может определить, что он может быть использован только клиентами из определенного списка.

Однако у динамического связывания имеются недостатки, например, дополнительные накладные расходы (временные затраты) на экспорт и импорт интерфейсов. Величина этих затрат может быть значительна, так как многие клиентские процессы существуют короткое время, а при каждом старте процесса

процедура импорта интерфейса должна быть снова выполнена. Кроме того, в больших распределенных системах может стать узким местом программа binder, а создание нескольких программ аналогичного назначения также увеличивает накладные расходы на создание и синхронизацию процессов.

Семантика RPC в случае отказов

В идеале RPC должен функционировать правильно и в случае отказов. Рассмотрим следующие классы отказов:

1. Клиент не может определить местонахождения сервера, например, в случае отказа нужного сервера, или из-за того, что программа клиента была скомпилирована давно и использовала старую версию интерфейса сервера. В этом случае в ответ на запрос клиента поступает сообщение, содержащее код ошибки.
2. Потерян запрос от клиента к серверу. Самое простое решение - через определенное время повторить запрос.
3. Потеряно ответное сообщение от сервера клиенту. Этот вариант сложнее предыдущего, так как некоторые процедуры не являются идемпотентными. Идемпотентной называется процедура, запрос на выполнение которой можно повторить несколько раз, и результат при этом не изменится. Примером такой процедуры может служить чтение файла. Но вот процедура снятия некоторой суммы с банковского счета не является идемпотентной, и в случае потери ответа повторный запрос может существенно изменить состояние счета клиента. Одним из возможных решений является приведение всех процедур к идемпотентному виду. Однако на практике это не всегда удается, поэтому может быть использован другой метод - последовательная нумерация всех запросов клиентским ядром. Ядро сервера запоминает номер самого последнего запроса от каждого из клиентов, и при получении каждого запроса выполняет анализ - является ли этот запрос первичным или повторным.
4. Сервер потерпел аварию после получения запроса. Здесь также важно свойство идемпотентности, но к сожалению не может быть применен подход с нумерацией запросов. В данном случае имеет значение, когда произошел отказ - до или после выполнения операции. Но клиентское ядро не может распознать эти ситуации, для него известно только то, что время ответа истекло. Существует три подхода к этой проблеме:

- Ждать до тех пор, пока сервер не перезагрузится и пытаться выполнить операцию снова. Этот подход гарантирует, что RPC был выполнен до конца по крайней мере один раз, а возможно и более.
- Сразу сообщить приложению об ошибке. Этот подход гарантирует, что RPC был выполнен не более одного раза.

- Третий подход не гарантирует ничего. Когда сервер отказывает, клиенту не оказывается никакой поддержки. RPC может быть или не выполнен вообще, или выполнен много раз. Во всяком случае этот способ очень легко реализовать.

Ни один из этих подходов не является очень привлекательным. А идеальный вариант, который бы гарантировал ровно одно выполнение RPC, в общем случае не может быть реализован по принципиальным соображениям. Пусть, например, удаленной операцией является печать некоторого текста, которая включает загрузку буфера принтера и установку одного бита в некотором управляющем регистре принтера, в результате которой принтер стартует. Авария сервера может произойти как за микросекунду до, так и за микросекунду после установки управляющего бита. Момент сбоя целиком определяет процедуру восстановления, но клиент о моменте сбоя узнать не может. Короче говоря, возможность аварии сервера радикально меняет природу RPC и ясно отражает разницу между централизованной и распределенной системой. В первом случае крах сервера ведет к краху клиента, и восстановление невозможно. Во втором случае действия по восстановлению системы выполнить и возможно, и необходимо.

1. Клиент потерпел аварию после отсылки запроса. В этом случае выполняются вычисления результатов, которых никто не ожидает. Такие вычисления называют "сиротами". Наличие сирот может вызвать различные проблемы: непроизводительные затраты процессорного времени, блокирование ресурсов, подмена ответа на текущий запрос ответом на запрос, который был выдан клиентской машиной еще до перезапуска системы.

Как поступать с сиротами? Рассмотрим 4 возможных решения.

- *Уничтожение.* До того, как клиентский стаб посылает RPC-сообщение, он делает отметку в журнале, оповещая о том, что он будет сейчас делать. Журнал хранится на диске или в другой памяти, устойчивой к сбоям. После аварии система перезагружается, журнал анализируется и сироты ликвидируются. К недостаткам такого подхода относятся, во-первых, повышенные затраты, связанные с записью о каждом RPC на диск, а, во-вторых, возможная неэффективность из-за появления сирот второго поколения, порожденных RPC-вызовами, выданными сиротами первого поколения.
- *Перевоплощение*. В этом случае все проблемы решаются без использования записи на диск. Метод состоит в делении времени на последовательно пронумерованные периоды. Когда клиент

перезагружается, он передает широковещательное сообщение всем машинам о начале нового периода. После приема этого сообщения все удаленные вычисления ликвидируются. Конечно, если сеть сегментированная, то некоторые сироты могут и уцелеть.

- *Мягкое перевоплощение* аналогично предыдущему случаю, за исключением того, что отыскиваются и уничтожаются не все удаленные вычисления, а только вычисления перезагружающегося клиента.
- *Истечение срока*. Каждому запросу отводится стандартный отрезок времени T, в течение которого он должен быть выполнен. Если запрос не выполняется за отведенное время, то выделяется дополнительный квант. Хотя это и требует дополнительной работы, но если после аварии клиента сервер ждет в течение интервала T до перезагрузки клиента, то все сироты обязательно уничтожаются.

На практике ни один из этих подходов не желателен, более того, уничтожение сирот может усугубить ситуацию. Например, пусть сирота заблокировал один или более файлов базы данных. Если сирота будет вдруг уничтожен, то эти блокировки останутся, кроме того уничтоженные сироты могут остаться стоять в различных системных очередях, в будущем они могут вызвать выполнение новых процессов и т.п.

Синхронизация в распределенных системах

К вопросам связи процессов, реализуемой путем передачи сообщений или вызовов RPC, тесно примыкают и вопросы синхронизации процессов. Синхронизация необходима процессам для организации совместного использования ресурсов, таких как файлы или устройства, а также для обмена данными.

В однопроцессорных системах решение задач взаимного исключения, критических областей и других проблем синхронизации осуществлялось с использованием общих методов, таких как семафоры и мониторы. Однако эти методы не совсем подходят для распределенных систем, так как все они базируются на использовании разделяемой оперативной памяти. Например, два процесса, которые взаимодействуют, используя семафор, должны иметь доступ к нему. Если оба процесса выполняются на одной и той же машине, они могут иметь совместный доступ к семафору, хранящемуся, например, в ядре, делая системные вызовы. Однако, если процессы выполняются на разных машинах, то этот метод не применим, для распределенных систем нужны новые подходы.

Алгоритм синхронизации логических часов

В централизованной однопроцессорной системе, как правило, важно только относительное время и не важна точность часов. В распределенной системе, где каждый процессор имеет собственные часы со своей точностью хода, ситуация резко меняется: программы, использующие время (например, программы,

подобные команде make в UNIX, которые используют время создания файлов, или программы, для которых важно время прибытия сообщений и т.п.) становятся зависимыми от того, часами какого компьютера они пользуются. В распределенных системах синхронизация физических часов (показывающих реальное время) является сложной проблемой, но с другой стороны очень часто в этом нет никакой необходимости: то есть процессам не нужно, чтобы во всех машинах было правильное время, для них важно, чтобы оно было везде одинаковое, более того, для некоторых процессов важен только правильный порядок событий. В этом случае мы имеем дело с логическими часами.

Введем для двух произвольных событий отношение "случилось до". Выражение a ® b читается "a случилось до b" и означает, что все процессы в системе считают, что сначала произошло событие a, а потом - событие b. Отношение "случилось до" обладает свойством транзитивности: если выражения a ® b и b ® c истинны, то справедливо и выражение a ® c. Для двух событий одного и того же процесса всегда можно установить отношение "случилось до", аналогично может быть установлено это отношение и для событий передачи сообщения одним процессом и приемом его другим, так как прием не может произойти раньше отправки. Однако, если два произвольных события случились в разных процессах на разных машинах, и эти процессы не имеют между собой никакой связи (даже косвенной через третьи процессы), то нельзя сказать с полной определенностью, какое из событий произошло раньше, а какое позже.

Ставится задача создания такого механизма ведения времени, который бы для каждого события a мог указать значение времени T(a), с которым бы были согласны все процессы в системе. При этом должно выполняться условие: если a ® b , то T(a) < T(b). Кроме того, время может только увеличиваться и, следовательно, любые корректировки времени могут выполняться только путем добавления положительных значений, и никогда - путем вычитания.

Рассмотрим алгоритм решения этой задачи, который предложил Lamport. Для отметок времени в нем используются события. На рисунке 3.6 показаны три процесса, выполняющихся на разных машинах, каждая из которых имеет свои часы, идущие со своей скоростью. Как видно из рисунка, когда часы процесса 0 показали время 6, в процессе 1 часы показывали 8, а в процессе 2 - 10. Предполагается, что все эти часы идут с постоянной для себя скоростью.

В момент времени 6 процесс 0 посылает сообщение A процессу 1. Это сообщение приходит к процессу 1 в момент времени 16 по его часам. В логическом смысле это вполне возможно, так как 6<16. Аналогично, сообщение B, посланное процессом 1 процессу 2 пришло к последнему в момент времени 40, то есть его передача заняла 16 единиц времени, что также является правдоподобным.

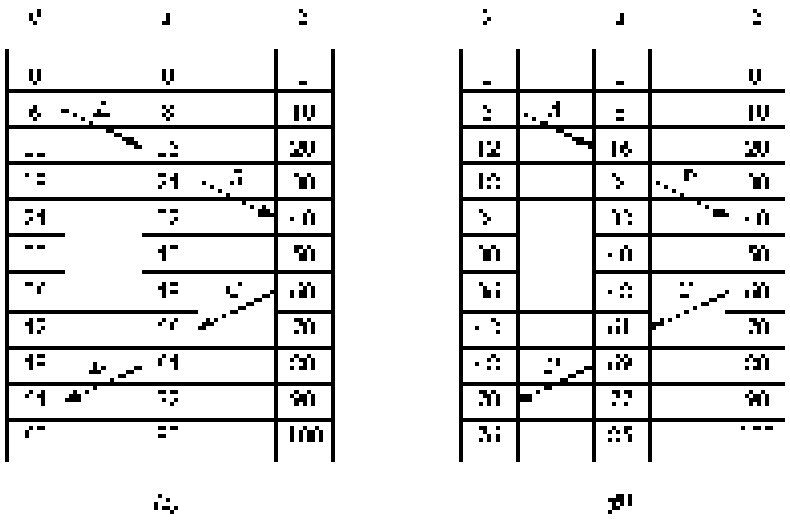


Рис. 3.6. Синхронизация логических часов
а - три процесса, каждый со своими собственными часами;
б - алгоритм синхронизации логических часов

Ну а далее начинаются весьма странные вещи. Сообщение C от процесса 2 к процессу 1 было отправлено в момент времени 64, а поступило в место назначения в момент времени 54. Очевидно, что это невозможно. Такие ситуации необходимо предотвращать. Решение Lamport'a вытекает непосредственно из отношений "случилось до". Так как C было отправлено в момент 60, то оно должно дойти в момент 61 или позже. Следовательно, каждое сообщение должно нести с собой время своего отправления по часам машины-отправителя. Если в машине, получившей сообщение, часы показывают время, которое меньше времени отправления, то эти часы переводятся вперед, так, чтобы они показали время, большее времени отправления сообщения. На рисунке 3.6,б видно, что C поступило в момент 61, а сообщение D - в 70.

Этот алгоритм удовлетворяет сформулированным выше требованиям.

Алгоритмы взаимного исключения

Системы, состоящие из нескольких процессов, часто легче программировать, используя так называемые критические секции. Когда процессу нужно читать или модифицировать некоторые разделяемые структуры данных, он прежде всего входит в критическую секцию для того, чтобы обеспечить себе исключительное право использования этих данных, при этом он уверен, что никакой процесс не будет иметь доступа к этому ресурсу одновременно с ним. Это называется взаимным исключением. В однопроцессорных системах критические секции защищаются семафорами, мониторами и другими аналогичными конструкциями. Рассмотрим, какие алгоритмы могут быть использованы в распределенных системах.

Централизованный алгоритм

Наиболее очевидный и простой путь реализации взаимного исключения в распределенных системах - это применение тех же методов, которые используются в однопроцессорных системах. Один из процессов выбирается в качестве координатора (например, процесс, выполняющийся на машине, имеющей наибольшее значение сетевого адреса). Когда какой-либо процесс хочет войти в критическую секцию, он посылает сообщение с запросом к координатору, оповещая его о том, в какую критическую секцию он хочет войти, и ждет от координатора разрешение. Если в этот момент ни один из процессов не находится в критической секции, то координатор посылает ответ с разрешением. Если же некоторый процесс уже выполняет критическую секцию, связанную с данным ресурсом, то никакой ответ не посылается; запрашивавший процесс ставится в очередь, и после освобождения критической секции ему отправляется ответ-разрешение. Этот алгоритм гарантирует взаимное исключение, но вследствие своей централизованной природы обладает низкой отказоустойчивостью.

Распределенный алгоритм

Когда процесс хочет войти в критическую секцию, он формирует сообщение, содержащее имя нужной ему критической секции, номер процесса и текущее значение времени. Затем он посылает это сообщение всем другим процессам. Предполагается, что передача сообщения надежна, то есть получение каждого сообщения сопровождается подтверждением. Когда процесс получает сообщение такого рода, его действия зависят от того, в каком состоянии по отношению к указанной в сообщении критической секции он находится. Имеют место три ситуации:

1. Если получатель не находится и не собирается входить в критическую секцию в данный момент, то он отсылает назад процессу-отправителю сообщение с разрешением.
2. Если получатель уже находится в критической секции, то он не отправляет никакого ответа, а ставит запрос в очередь.
3. Если получатель хочет войти в критическую секцию, но еще не сделал этого, то он сравнивает временную отметку поступившего сообщения со значением времени, которое содержится в его собственном сообщении, разосланном всем другим процессам. Если время в поступившем к нему сообщении меньше, то есть его собственный запрос возник позже, то он посылает сообщение-разрешение, в обратном случае он не посылает ничего и ставит поступившее сообщение-запрос в очередь.

Процесс может войти в критическую секцию только в том случае, если он получил ответные сообщения-разрешения от всех остальных процессов. Когда процесс покидает критическую секцию, он посылает разрешение всем процессам из своей очереди и исключает их из очереди.

Алгоритм Token Ring

Совершенно другой подход к достижению взаимного исключения в распределенных системах иллюстрируется рисунком 3.7. Все процессы системы образуют логическое кольцо, т.е. каждый процесс знает номер своей позиции в кольце, а также номер ближайшего к нему следующего процесса. Когда кольцо инициализируется, процессу 0 передается так называемый токен. Токен циркулирует по кольцу. Он переходит от процесса n к процессу n+1 путем передачи сообщения по типу "точка-точка". Когда процесс получает токен от своего соседа, он анализирует, не требуется ли ему самому войти в критическую секцию. Если да, то процесс входит в критическую секцию. После того, как процесс выйдет из критической секции, он передает токен дальше по кольцу. Если же процесс, принявший токен от своего соседа, не заинтересован во вхождении в критическую секцию, то он сразу отправляет токен в кольцо. Следовательно, если ни один из процессов не желает входить в критическую секцию, то в этом случае токен просто циркулирует по кольцу с высокой скоростью.

Сравним эти три алгоритма взаимного исключения. Централизованный алгоритм является наиболее простым и наиболее эффективным. При его использовании требуется только три сообщения для того, чтобы процесс вошел и покинул критическую секцию: запрос и сообщение-разрешение для входа и сообщение об освобождении ресурса при выходе. При использовании распределенного алгоритма для одного использования критической секции требуется послать (n-1) сообщений-запросов (где n - число процессов) - по одному на каждый процесс и получить (n-1) сообщений-разрешений, то есть всего необходимо 2(n-1) сообщений. В алгоритме Token Ring число сообщений переменно: от 1 в случае, если каждый процесс входил в критическую секцию, до бесконечно большого числа, при циркуляции токена по кольцу, в котором ни один процесс не входил в критическую секцию.

К сожалению все эти три алгоритма плохо защищены от отказов. В первом случае к краху приводит отказ координатора, во втором - отказ любого процесса (парадоксально, но распределенный алгоритм оказывается менее отказоустойчивым, чем централизованный), а в третьем - потеря токена или отказ процесса.

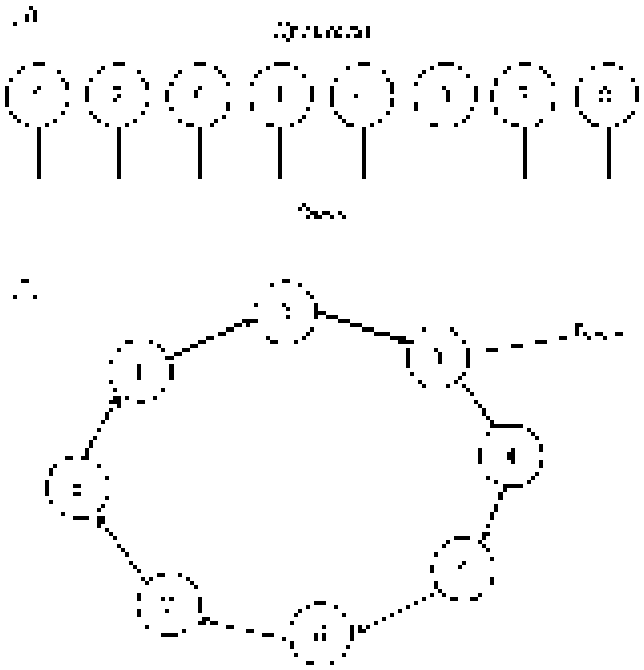


Рис. 3.7. Средства взаимного исключения в распределенных системах
а - неупорядоченная группа процессов в сети;
б - логическое кольцо, образованное программным обеспечением

Неделимые транзакции

Все средства синхронизации, которые были рассмотрены ранее, относятся к нижнему уровню, например, семафоры. Они требуют от программиста детального знания алгоритмов взаимного исключения, управления критическими секциями, умения предотвращать клинчи (взаимные блокировки), а также владения средствами восстановления после краха. Однако существуют средства синхронизации более высокого уровня, которые освобождают программиста от необходимости вникать во все эти подробности и позволяют ему сконцентрировать свое внимание на логике алгоритмов и организации параллельных вычислений. Таким средством является неделимая транзакция.

Модель неделимой транзакции пришла из бизнеса. Представьте себе переговорный процесс двух фирм о продаже-покупке некоторого товара. В процессе переговоров условия договора могут многократно меняться, уточняться. Пока договор еще не подписан обеими сторонами, каждая из них может от него отказаться. Но после подписания контракта сделка (transaction) должна быть выполнена.

Компьютерная транзакция полностью аналогична. Один процесс объявляет, что он хочет начать транзакцию с одним или более процессами. Они могут некоторое время создавать и уничтожать разные объекты, выполнять какие-либо операции. Затем инициатор объявляет, что он хочет завершить транзакцию. Если все с ним соглашаются, то результат фиксируется. Если один или более процессов отказываются (или они потерпели крах еще до выработки согласия), тогда

измененные объекты возвращается точно к тому состоянию, в котором они находились до начала выполнения транзакции. Такое свойство "все-или-ничего" облегчает работу программиста.

Для программирования с использованием транзакций требуется некоторый набор примитивов, которые должны быть предоставлены программисту либо операционной системой, либо языком программирования. Примеры примитивов такого рода:

BEGIN_TRANSACTION	команды, которые следуют за этим примитивом, формируют транзакцию.
END_TRANSACTION	завершает транзакцию и пытается зафиксировать ее.
ABORT_TRANSACTION	прерывает транзакцию, восстанавливает предыдущие значения.
READ	читает данные из файла (или другого объекта)
WRITE	пишет данные в файл (или другой объект).

Первые два примитива используются для определения границ транзакции. Операции между ними представляют собой тело транзакции. Либо все они должны быть выполнены, либо ни одна из них. Это может быть системный вызов, библиотечная процедура или группа операторов языка программирования, заключенная в скобки.

Транзакции обладают следующими свойствами: упорядочиваемостью, неделимостью, постоянством.

Упорядочиваемость гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы все транзакции выполнялись последовательно в некотором (в зависимости от системы) порядке.

Неделимость означает, что когда транзакция находится в процессе выполнения, то никакой другой процесс не видит ее промежуточные результаты.

Постоянство означает, что после фиксации транзакции никакой сбой не может отменить результатов ее выполнения.

Если программное обеспечение гарантирует вышеперечисленные свойства, то это означает, что в системе поддерживается механизм транзакций.

Рассмотрим некоторые подходы к реализации механизма транзакций.

В соответствии с первым подходом, когда процесс начинает транзакцию, то он работает в индивидуальном рабочем пространстве, содержащем все файлы и другие объекты, к которым он имеет доступ. Пока транзакция не зафиксируется

или не прервется, все изменения данных происходят в этом рабочем пространстве, а не в "реальном", под которым мы понимаем обычную файловую систему. Главная проблема этого подхода состоит в больших накладных расходах по копированию большого объема данных в индивидуальное рабочее пространство, хотя и имеются несколько приемов уменьшения этих расходов.

Второй общий подход к реализации механизма транзакций называется списком намерений. Этот метод заключается в том, что модифицируются сами файлы, а не их копии, но перед изменением любого блока производится запись в специальный файл - журнал регистрации, где отмечается, какая транзакция делает изменения, какой файл и блок изменяется и каковы старое и новое значения изменяемого блока. Только после успешной записи в журнал регистрации делаются изменения в исходном файле. Если транзакция фиксируется, то и об этом делается запись в журнал регистрации, но старые значения измененных данных сохраняются. Если транзакция прерывается, то информация журнала регистрации используется для приведения файла в исходное состояние, и это действие называется откатом.

В распределенных системах фиксация транзакций может потребовать взаимодействия нескольких процессов на разных машинах, каждая из которых хранит некоторые переменные, файлы, базы данных. Для достижения свойства неделимости транзакций в распределенных системах используется специальный протокол, называемый протоколом двухфазной фиксации транзакций. Хотя он и не является единственным протоколом такого рода, но он наиболее широко используется.

Суть этого протокола состоит в следующем. Один из процессов выполняет функции координатора (рисунок 3.8). Координатор начинает транзакцию, делая запись об этом в своем журнале регистрации, затем он посылает всем подчиненным процессам, также выполняющим эту транзакцию, сообщение "подготовиться к фиксации". Когда подчиненные процессы получают это сообщение, то они проверяют, готовы ли они к фиксации, делают запись в своем журнале и посылают координатору сообщение-ответ "готов к фиксации". После этого подчиненные процессы остаются в состоянии готовности и ждут от координатора команду фиксации. Если хотя бы один из подчиненных процессов не откликнулся, то координатор откатывает подчиненные транзакции, включая и те, которые подготовились к фиксации.

Выполнение второй фазы заключается в том, что координатор посылает команду "фиксировать" (commit) всем подчиненным процессам. Выполняя эту команду, последние фиксируют изменения и завершают подчиненные транзакции. В результате гарантируется одновременное синхронное завершение (удачное или неудачное) распределенной транзакции.

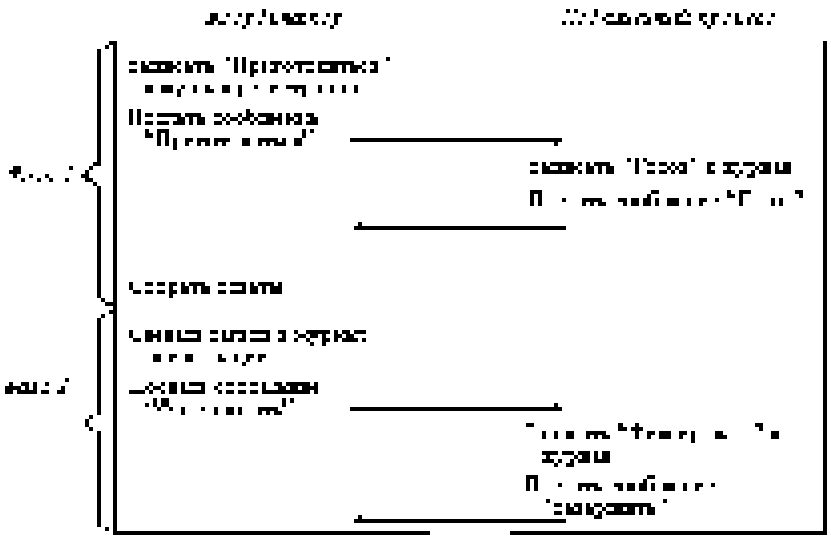


Рис. 3.8. Двухфазный протокол фиксации транзакции

Процессы и нити в распределенных системах

Понятие "нить"

В традиционных ОС понятие нити тождественно понятию процесса. В действительности желательно иметь несколько нитей управления, разделяющих единое адресное пространство, но выполняющихся квазипараллельно.

Предположим, например, что файл-сервер блокируется, ожидания выполнения операции с диском. Если сервер имеет несколько нитей управления, вторая нить может выполняться, пока первая нить находится в состоянии ожидания. Это повышает пропускную способность и производительность. Эта цель не достигается путем создания двух независимых серверных процессов, потому что они должны разделять общий буфер кэша, который требуется им, чтобы быть в одном адресном пространстве.

На рисунке 3.9,а показана машина с тремя процессами. Каждый процесс имеет собственный программный счетчик, собственный стек, собственный набор регистров и собственное адресное пространство. Каждый процесс не должен ничего делать с остальными, за исключением того, что они могут взаимодействовать посредством системных примитивов связи, таких как семафоры, мониторы, сообщения. На рисунке 3.9,б показана другая машина с одним процессом. Этот процесс состоит из нескольких нитей управления, обычно называемых просто нитями или иногда облегченными процессами. Во многих отношениях нити подобны мини-процессам. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити разделяют процессор так, как это делают процессы (разделение времени). Только на многопроцессорной системе они действительно выполняются параллельно. Нити могут, например, порождать нити-потомки, могут переходить в состояние ожидания до завершения системного вызова, как обычные процессы, пока одна нить заблокирована, другая нить того же процесса может выполняться.

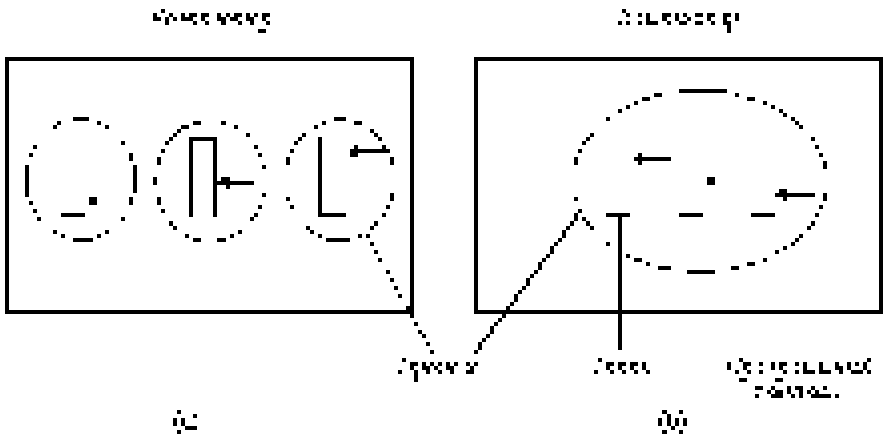


Рис. 3.9. а) Три процесса с одной нитью каждый
б) Один процесс с тремя нитями

Нити делают возможным сохранение идеи последовательных процессов, которые выполняют блокирующие системные вызовы (например, RPC для обращения к диску), и в то же время позволяют достичь параллелизма вычислений. Блокирующие системные вызовы делают проще программирование, а параллелизм повышает производительность.

Различные способы организации вычислительного процесса с использованием нитей

Один из возможных способов организации вычислительного процесса показан на рисунке 3.10,а. Здесь нить-диспетчер читает приходящие запросы на работу из почтового ящика системы. После проверки запроса диспетчер выбирает простаивающую (то есть блокированную) рабочую нить, передает ей запрос и активизирует ее, устанавливая, например, семафор, который она ожидает.

Когда рабочая нить активизируется, она проверяет, может ли быть выполнен запрос с данными разделяемого блока кэша, к которому имеют отношение все нити. Если нет, она посылает сообщение к диску, чтобы получить нужный блок (предположим, это READ), и переходит в состояние блокировки, ожидая завершения дисковой операции. В этот момент происходит обращение к планировщику, в результате работы которого активизируется другая нить, возможно, нить-диспетчер или некоторая рабочая нить, готовая к выполнению.

Структура с диспетчером не единственный путь организации многонитевой обработки. В модели "команда" все нити эквивалентны, каждая получает и обрабатывает свои собственные запросы. Иногда работы приходят, а нужная нить занята, особенно, если каждая нить специализируется на выполнении особого вида работ. В этом случае может создаваться очередь незавершенных работ. При такой организации нити должны вначале просматривать очередь работ, а затем почтовый ящик.

Нити могут быть также организованы в виде конвейера. В этом случае первая нить порождает некоторые данные и передает их для обработки следующей нити и т.д. Хотя эта организация и не подходит для файл-сервера, для других задач, например, задач типа "производитель-потребитель", это хорошее решение.

Нити часто полезны и для клиентов. Например, если клиент хочет растиражировать файл на много серверов, он может создать по одной нити для копирования на каждом сервере. Другое использование нитей клиентами - это управление сигналами, такими как прерывание с клавиатуры (del или break). Вместо обработки сигнала прерывания одна нить назначается для постоянного ожидания поступления сигналов. Таким образом, использование нитей может сократить необходимое количество прерываний пользовательского уровня.

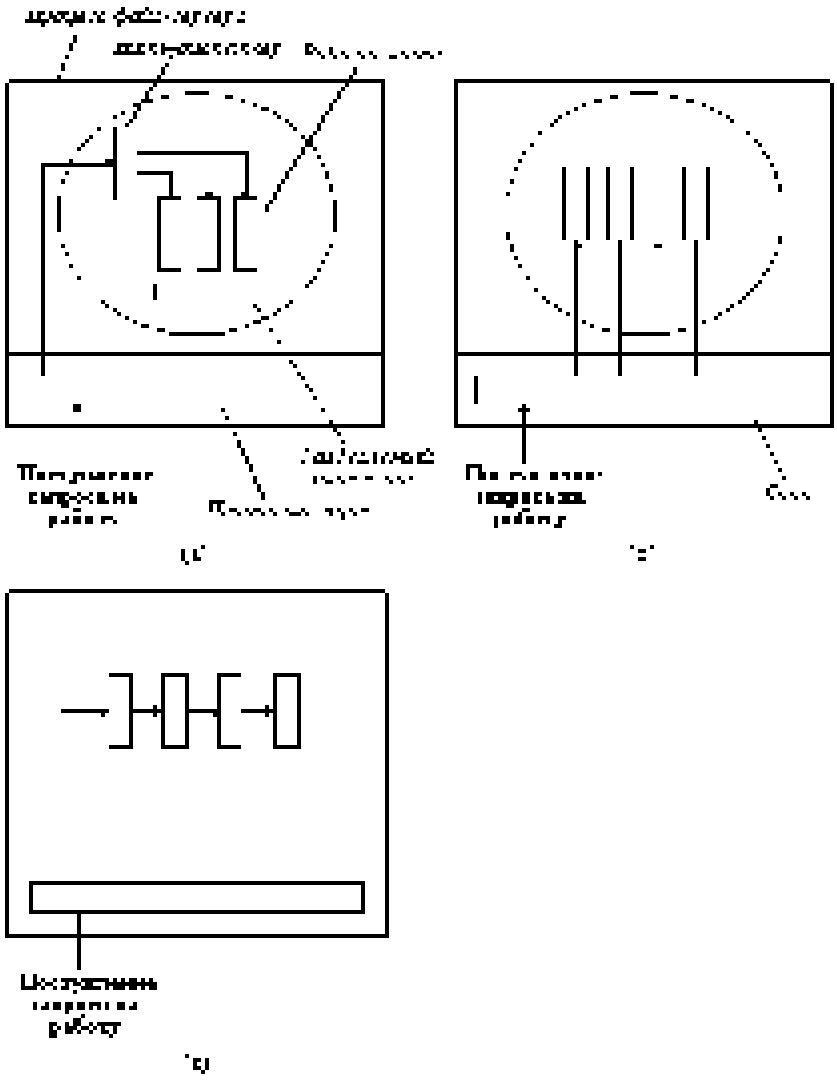


Рис. 3.10. Три способа организации нитей в процессе:
а - модель диспетчер/рабочие нити; б - модель "команда"; в - модель конвейера

Другой аргумент в пользу нитей не имеет отношения ни к удаленным вызовам, ни к коммуникациям. Некоторые прикладные задачи легче программировать, используя параллелизм, например задачи типа "производитель-потребитель". Не столь важно параллельное выполнение, сколь важна ясность программы. А

поскольку они разделяют общий буфер, не стоит их делать отдельными процессами.

Наконец, в многопроцессорных системах нити из одного адресного пространства могут выполняться параллельно на разных процессорах. С другой стороны, правильно сконструированные программы, которые используют нити, должны работать одинаково хорошо на однопроцессорной машине в режиме разделения времени между нитями и на настоящем мультипроцессоре.

Вопросы реализации нитей

Существует два подхода к управлению нитями: статический и динамический. При статическом подходе вопрос, сколько будет нитей, решается уже на стадии написания программы или на стадии компиляции. Каждой нити назначается фиксированный стек. Этот подход простой, но негибкий. Более общим является динамический подход, который позволяет создавать и удалять нити оперативно по ходу выполнения. Системный вызов для создания нити обычно содержится в нити главной программы в виде указателя на процедуру с указанием размера стека, а также других параметров, например, диспетчерского приоритета. Вызов обычно возвращает идентификатор нити, который можно использовать в последующих вызовах, связанных с этой нитью. В этой модели процесс начинается с одной нити, но может создавать их еще, когда необходимо.

Завершаться нити могут одним из двух способов: по своей инициативе, когда завершается работа, и извне. Во многих случаях, например, при конвейерной модели, нити создаются сразу же после старта процесса и никогда не уничтожаются.

Поскольку нити разделяют общую память, они могут (и, как правило, делают это) использовать ее для сохранения данных, которые совместно используются множеством нитей, таких, например, как буфер в системе "производитель-потребитель". Доступ к разделяемым данным обычно программируется с использованием критических секций, предотвращающих попытки сразу нескольких нитей обратиться к одним и тем же данным в одно и то же время. Критическая секция наиболее легко реализуется с использованием семафоров, мониторов и аналогичных конструкций.

Нити могут быть реализованы как в пользовательском пространстве, так и в пространстве ядра. В первом случае нити работают на базе прикладной системы, управляющей всеми операциями с нитями. Первым преимуществом такого способа является то, что можно реализовать нити в операционной системе, которая их не поддерживает. ОС прикладная среда, управляющая нитями, кажется одним процессом. Все вызовы (ПРИОСТАНОВИТЬ, ПРОВЕРИТЬ СЕМАФОР и т. д.) обрабатываются как вызовы функций этой прикладной среды. Она сохраняет регистры и переключает указатели счетчика команд и стека. В этом случае переключение происходит быстрее, чем с помощью ядра. Такая реализация имеет еще одно преимущество - для каждого процесса можно

организовать свою схему планирования. Однако этот подход связан с некоторыми проблемами, одна из которых состоит в следующем. При выполнении блокирующих системных вызовов приостанавливается весь набор нитей, принадлежащих этому процессу. Чтобы избежать этого, можно сделать все системные вызовы неблокирующими, но это требует изменений в ОС, что нежелательно, так как одной из целей реализации нитей в пользовательском пространстве является их работа в существующих операционных системах.

Такой проблемы не существует при реализации нитей в пространстве ядра. Преимущество заключается также и в том, что ядро может при диспетчеризации выбирать нить из другого процесса. Однако хотя механизм управления нитями аналогичен первому случаю, временные затраты на переключение нитей выше, так как тратится время на переключение из режима пользователя в режим ядра.

Нити и RPC

Обычно в распределенных системах используются как RPC, так и нити. Так как нити были введены как дешевая альтернатива стандартным процессам, то естественно, что исследователи обратили особое внимание в этом контексте на RPC: нельзя ли их также сделать облегченными. Было замечено, что в распределенных системах значительное количество RPC обрабатывается на той же машине, на которой они были вызваны (локально), например, вызовы к менеджеру окон. Поэтому была предложена новая схема, которая делает возможным для нити одного процесса вызвать нить другого процесса на этой же машине более эффективно, чем обычным способом.

Идея заключается в следующем. Когда стартует серверная нить S, то она экспортирует свой интерфейс, сообщая о нем ядру. Интерфейс определяет, какие процедуры могут быть вызваны, каковы их параметры и т.п. Когда стартует клиентская нить C, то она импортирует интерфейс из ядра в том случае, если собирается вызвать S, и ей дается специальный идентификатор для выполнения определенного вызова. Ядро теперь знает, что C собирается позже вызвать S и создает специальные структуры данных для подготовки к вызову.

Одна из этих структур данных является стеком аргументов, который разделяется нитями C и S и отображается в оба адресных пространства для чтения и записи. Для вызова сервера нить C помещает аргументы в разделяемый стек, используя обычную процедуру передачи параметров, а затем прерывает ядро, помещая данный ей идентификатор в регистр. По этому идентификатору ядро видит, что вызов является локальным. (Если бы он был удаленным, то ядро обработало бы его обычным способом для удаленных вызовов.) Затем ядро выполняет переключение из адресного пространства клиента в адресное пространство нити-сервера и запускает в рамках клиентской нити требуемую процедуру сервера. При таком способе вызова аргументы уже загружены в нужное место, так что копирование или перегруппировка аргументов не требуется. Главный результат - локальный вызов RPC - будет выполнен этим способом гораздо быстрее.

Другой прием широко используется для ускорения удаленных RPC. Идея основана на следующем наблюдении: когда нить-сервер блокируется, ожидая нового запроса, ее контекст почти всегда не содержит важной информации. Следовательно, когда нить завершает обработку запроса, то ее просто удаляют. При поступлении на сервер нового сообщения ядро создает новую нить для обслуживания этого запроса. Кроме того ядро помещает сообщение в адресное пространство сервера и устанавливает новый стек нити для доступа к сообщению. Эту схему иногда называют неявным вызовом.

Этот метод имеет несколько преимуществ по сравнению с обычным RPC. Во-первых, нити не должны блокироваться, ожидая новую работу, следовательно контекст не нужно сохранять, во-вторых, создание новой нити проще, чем активизация существующей приостановленной, так как не нужно восстанавливать контекст.

Распределенные файловые системы

Ключевым компонентом любой распределенной системы является файловая система. Как и в централизованных системах, в распределенной системе функцией файловой системы является хранение программ и данных и предоставление доступа к ним по мере необходимости. Файловая система поддерживается одной или более машинами, называемыми файл-серверами. Файл-серверы перехватывают запросы на чтение или запись файлов, поступающие от других машин (не серверов). Эти другие машины называются клиентами. Каждый посланный запрос проверяется и выполняется, а ответ отсылается обратно. Файл-серверы обычно содержат иерархические файловые системы, каждая из которых имеет корневой каталог и каталоги более низких уровней. Рабочая станция может подсоединять и монтировать эти файловые системы к своим локальным файловым системам. При этом монтируемые файловые системы остаются на серверах.

Важно понимать различие между файловым сервисом и файловым сервером. Файловый сервис - это описание функций, которые файловая система предлагает своим пользователям. Это описание включает имеющиеся примитивы, их параметры и функции, которые они выполняют. С точки зрения пользователей файловый сервис определяет то, с чем пользователи могут работать, но ничего не говорит о том, как все это реализовано. В сущности, файловый сервис определяет интерфейс файловой системы с клиентами.

Файловый сервер - это процесс, который выполняется на отдельной машине и помогает реализовывать файловый сервис. В системе может быть один файловый сервер или несколько, но в хорошо организованной распределенной системе пользователи не знают, как реализована файловая система. В частности, они не знают количество файловых серверов, их месторасположение и функции. Они только знают, что если процедура определена в файловом сервисе, то требуемая работа каким-то образом выполняется, и им возвращаются требуемые результаты. Более того, пользователи даже не должны знать, что файловый

сервис является распределенным. В идеале он должен выглядеть также, как и в централизованной файловой системе.

Так как обычно файловый сервер - это просто пользовательский процесс (или иногда процесс ядра), выполняющийся на некоторой машине, в системе может быть несколько файловых серверов, каждый из которых предлагает различный файловый сервис. Например, в распределенной системе может быть два сервера, которые обеспечивают файловые сервисы систем UNIX и MS-DOS соответственно, и любой пользовательский процесс пользуется подходящим сервисом.

Файловый сервис в распределенных файловых системах (впрочем как и в централизованных) имеет две функционально различные части: собственно файловый сервис и сервис каталогов. Первый имеет дело с операциями над отдельными файлами, такими, как чтение, запись или добавление, а второй - с созданием каталогов и управлением ими, добавлением и удалением файлов из каталогов и т.п.

Интерфейс файлового сервиса

Для любого файлового сервиса, независимо от того, централизован он или распределен, самым главным является вопрос, что такое файл? Во многих системах, таких как UNIX и MS DOS, файл - это неинтерпретируемая последовательность байтов. Значение и структура информации в файле является заботой прикладных программ, операционную систему это не интересует.

В ОС мейнфреймов поддерживаются разные типы логической организации файлов, каждый с различными свойствами. Файл может быть организован как последовательность записей, и у операционной системы имеются вызовы, которые позволяют работать на уровне этих записей. Большинство современных распределенных файловых систем поддерживают определение файла как последовательности байтов, а не последовательности записей. Файл характеризуется атрибутами: именем, размером, датой создания, идентификатором владельца, адресом и другими.

Важным аспектом файловой модели является возможность модификации файла после его создания. Обычно файлы могут модифицироваться, но в некоторых распределенных системах единственными операциями с файлами являются СОЗДАТЬ и ПРОЧИТАТЬ. Такие файлы называются неизменяемыми. Для неизменяемых файлов намного легче осуществить кэширование файла и его репликацию (тиражирование), так как исключается все проблемы, связанные с обновлением всех копий файла при его изменении.

Файловый сервис может быть разделен на два типа в зависимости от того, поддерживает ли он модель загрузки-выгрузки или модель удаленного доступа. В модели загрузки-выгрузки пользователю предлагаются средства чтения или записи файла целиком. Эта модель предполагает следующую схему обработки

файла: чтение файла с сервера на машину клиента, обработка файла на машине клиента и запись обновленного файла на сервер. Преимуществом этой модели является ее концептуальная простота. Кроме того, передача файла целиком очень эффективна. Главным недостатком этой модели являются высокие требования к дискам клиентов. Кроме того, неэффективно перемещать весь файл, если нужна его маленькая часть.

Другой тип файлового сервиса соответствует модели удаленного доступа, которая предполагает поддержку большого количества операций над файлами: открытие и закрытие файлов, чтение и запись частей файла, позиционирование в файле, проверка и изменение атрибутов файла и так далее. В то время как в модели загрузки-выгрузки файловый сервер обеспечивал только хранение и перемещение файлов, в данном случае вся файловая система выполняется на серверах, а не на клиентских машинах. Преимуществом такого подхода являются низкие требования к дисковому пространству на клиентских машинах, а также исключение необходимости передачи целого файла, когда нужна только его часть.

Интерфейс сервиса каталогов

Природа сервиса каталогов не зависит от типа используемой модели файлового сервиса. В распределенных системах используются те же принципы организации каталогов, что и в централизованных, в том числе многоуровневая организация каталогов.

Принципиальной проблемой, связанной со способами именования файлов, является обеспечение прозрачности. В данном контексте прозрачность понимается в двух слабо различимых смыслах. Первый - прозрачность расположения - означает, что имена не дают возможности определить месторасположение файла. Например, имя */server1/dir1/ dir2/x* говорит, что файл x расположен на сервере 1, но не указывает, где расположен этот сервер. Сервер может перемещаться по сети, а полное имя файла при этом не меняется. Следовательно, такая система обладает прозрачностью расположения.

Предположим, что файл x очень большой, а на сервере 1 мало места, предположим далее, что на сервере 2 места много. Система может захотеть переместить автоматически файл x на сервер 2. К сожалению, когда первый компонент всех имен - это имя сервера, система не может переместить файл на другой сервер автоматически, даже если каталоги *dir1* и *dir2* находятся на обоих серверах. Программы, имеющие встроенные строки имен файлов, не будут правильно работать в этом случае. Система, в которой файлы могут перемещаться без изменения имен, обладает свойством независимости от расположения. Распределенная система, которая включает имена серверов или машин непосредственно в имена файлов, не является независимой от расположения. Система, базирующаяся на удаленном монтировании, также не обладает этим свойством, так как в ней невозможно переместить файл из одной группы файлов в другую и продолжать после этого пользоваться старыми

именами. Независимости от расположения трудно достичь, но это желаемое свойство распределенной системы.

Большинство распределенных систем используют какую-либо форму двухуровневого именования: на одном уровне файлы имеют символические имена, такие как prog.c, предназначенные для использования людьми, а на другом - внутренние, двоичные имена, для использования самой системой. Каталоги обеспечивают отображение между двумя этими уровнями имен. Отличием распределенных систем от централизованных является возможность соответствия одному символьному имени нескольких двоичных имен. Обычно это используется для представления оригинального файла и его архивных копий. Имея несколько двоичных имен, можно при недоступности одной из копий файла получить доступ к другой. Этот метод обеспечивает отказоустойчивость за счет избыточности.

Семантика разделения файлов

Когда два или более пользователей разделяют один файл, необходимо точно определить семантику чтения и записи, чтобы избежать проблем. В централизованных системах, разрешающих разделение файлов, таких как UNIX, обычно определяется, что, когда операция ЧТЕНИЕ следует за операцией ЗАПИСЬ, то читается только что обновленный файл. Аналогично, когда операция чтения следует за двумя операциями записи, то читается файл, измененный последней операцией записи. Тем самым система придерживается абсолютного временного упорядочивания всех операций, и всегда возвращает самое последнее значение. Будем называть эту модель семантикой UNIX'a. В централизованной системе (и даже на мультипроцессоре с разделяемой памятью) ее легко и понять, и реализовать.

Семантика UNIX может быть обеспечена и в распределенных системах, но только, если в ней имеется лишь один файловый сервер, и клиенты не кэшируют файлы. Для этого все операции чтения и записи направляются на файловый сервер, который обрабатывает их строго последовательно. На практике, однако, производительность распределенной системы, в которой все запросы к файлам идут на один сервер, часто становится неудовлетворительной. Эта проблема иногда решается путем разрешения клиентам обрабатывать локальные копии часто используемых файлов в своих личных кэшах. Если клиент сделает локальную копию файла в своем локальном кэше и начнет ее модифицировать, а вскоре после этого другой клиент прочитает этот файл с сервера, то он получит неверную копию файла. Одним из способов устранения этого недостатка является немедленный возврат всех изменений в кэшированном файле на сервер. Такой подход хотя и концептуально прост, но не эффективен.

Другим решением является введение так называемой сессионной семантики, в соответствии с которой изменения в открытом файле сначала виды только процессу, который модифицирует файл, и только после закрытия файла эти изменения могут видеть другие процессы. При использовании сессионной

семантики возникает проблема одновременного использования одного и того же файла двумя или более клиентами. Одним из решений этой проблемы является принятие правила, в соответствии с которым окончательным является тот вариант, который был закрыт последним. Менее эффективным, но гораздо более простым в реализации, является вариант, при котором окончательным результирующим файлом на сервере может оказаться любой из этих файлов.

Следующий подход к разделению файлов заключается в том, чтобы сделать все файлы неизменяемыми. Тогда файл нельзя открыть для записи, а можно выполнять только операции СОЗДАТЬ и ЧИТАТЬ. Тогда для изменения файла остается только возможность создать полностью новый файл и поместить его в каталог под именем старого файла. Следовательно, хотя файл и нельзя модифицировать, его можно заменить (автоматически) новым файлом. Другими словами, хотя файлы и нельзя обновлять, но каталоги обновлять можно. Таким образом, проблема, связанная с одновременным использованием файла, просто исчезнет.

Четвертый способ работы с разделяемыми файлами в распределенных системах - это использование механизма неделимых транзакций, достаточно подробно описанного в разделе 3.3.3.

Итак, было рассмотрено четыре различных подхода к работе с разделяемыми файлами в распределенных системах.

1. **Семантика UNIX**. Каждая операция над файлом немедленно становится видимой для всех процессов.
2. **Сессионная семантика**. Изменения не видны до тех пор, пока файл не закрывается.
3. **Неизменяемые файлы**. Модификации невозможны, разделение файлов и репликация упрощаются.
4. **Транзакции**. Все изменения делаются по принципу "все или ничего".

Вопросы разработки структуры файловой системы

Рассмотрим прежде всего вопрос о распределении серверной и клиентской частей между машинами. В некоторых системах (например, NFS) нет разницы между клиентом и сервером, на всех машинах работает одно и то же базовое программное обеспечение, так что любая машина, которая хочет предложить файловый сервис, свободно может это сделать. Для этого ей достаточно экспортировать имена выбранных каталогов, чтобы другие машины могли иметь к ним доступ.

В других системах файловый сервер - это только пользовательская программа, так что система может быть сконфигурирована как клиент, как сервер или как клиент и сервер одновременно. Третьим, крайним случаем, является система, в которой клиенты и серверы - это принципиально различные машины, как в

терминах аппаратуры, так и в терминах программного обеспечения. Серверы могут даже работать под управлением другой операционной системы.

Вторым важным вопросом реализации файловой системы является структуризация сервиса файлов и каталогов. Один подход заключается в комбинировании этих двух сервисов на одном сервере. При другом подходе эти сервисы разделяются. В последнем случае при открытии файла требуется обращение к серверу каталогов, который отображает символьное имя в двоичное, а затем обращение к файловому серверу с двоичным именем для действительного чтения или записи файла.

Аргументом в пользу разделения сервисов является тот факт, что они на самом деле слабо связаны, поэтому их раздельная реализация более гибкая. Например, можно реализовать сервер каталогов MS-DOS и сервер каталогов UNIX, которые будут использовать один и тот же файловый сервер для физического хранения файлов. Разделение этих функций также упрощает программное обеспечение. Недостатком является то, что использование двух серверов увеличивает интенсивность сетевого обмена.

Постоянный поиск имен, особенно при использовании нескольких серверов каталогов, может приводить к большим накладным расходам. В некоторых системах делается попытка улучшить производительность за счет кэширования имен. При открытии файла кэш проверяется на наличие в нем нужного имени. Если оно там есть, то этап поиска, выполняемый сервером каталогов, пропускается, и двоичный адрес извлекается из кэша.

Последний рассматриваемый здесь структурный вопрос связан с хранением на серверах информации о состоянии клиентов. Существует две конкурирующие точки зрения.

Первая состоит в том, что сервер не должен хранить такую информацию (сервер stateless). Другими словами, когда клиент посылает запрос на сервер, сервер его выполняет, отсылает ответ, а затем удаляет из своих внутренних таблиц всю информацию о запросе. Между запросами на сервере не хранится никакой текущей информации о состоянии клиента. Другая точка зрения состоит в том, что сервер должен хранить такую информацию (сервер statefull).

Рассмотрим эту проблему на примере файлового сервера, имеющего команды ОТКРЫТЬ, ПРОЧИТАТЬ, ЗАПИСАТЬ и ЗАКРЫТЬ файл. Открывая файлы, statefull-сервер должен запоминать, какие файлы открыл каждый пользователь. Обычно при открытии файла пользователю дается дескриптор файла или другое число, которое используется при последующих вызовах для его идентификации. При поступлении вызова, сервер использует дескриптор файла для определения, какой файл нужен. Таблица, отображающая дескрипторы файлов на сами файлы, является информацией о состоянии клиентов.

Для сервера stateless каждый запрос должен содержать исчерпывающую информацию (полное имя файла, смещение в файле и т.п.), необходимую серверу для выполнения требуемой операции. Очевидно, что эта информация увеличивает длину сообщения.

Однако при отказе statefull-сервера теряются все его таблицы, и после перезагрузки неизвестно, какие файлы открыл каждый пользователь. Последовательные попытки провести операции чтения или записи с открытыми файлами будут безуспешными. Stateless-серверы в этом плане являются более отказоустойчивыми, и это аргумент в их пользу.

Преимущества обоих подходов можно обобщить следующим образом:

Stateless-серверы:

- отказоустойчивы;
- не нужны вызовы OPEN/CLOSE;
- меньше памяти сервера расходуется на таблицы;
- нет ограничений на число открытых файлов;
- отказ клиента не создает проблем для сервера.

Statefull-серверы:

- более короткие сообщения при запросах;
- лучше производительность;
- возможно опережающее чтение;
- легче достичь идемпотентности;
- возможна блокировка файлов.

Кэширование

В системах, состоящих из клиентов и серверов, потенциально имеется четыре различных места для хранения файлов и их частей: диск сервера, память сервера, диск клиента (если имеется) и память клиента. Наиболее подходящим местом для хранения всех файлов является диск сервера. Он обычно имеет большую емкость, и файлы становятся доступными всем клиентам. Кроме того, поскольку в этом случае существует только одна копия каждого файла, то не возникает проблемы согласования состояний копий.

Проблемой при использовании диска сервера является производительность. Перед тем, как клиент сможет прочитать файл, файл должен быть переписан с диска сервера в его оперативную память, а затем передан по сети в память клиента. Обе передачи занимают время.

Значительное увеличение производительности может быть достигнуто за счет кэширования файлов в памяти сервера. Требуются алгоритмы для определения, какие файлы или их части следует хранить в кэш-памяти.

При выборе алгоритма должны решаться две задачи. Во-первых, какими единицами оперирует кэш. Этими единицами могут быть или дисковые блоки, или целые файлы. Если это целые файлы, то они могут храниться на диске непрерывными областями (по крайней мере в виде больших участков), при этом уменьшается число обменов между памятью и диском а, следовательно, обеспечивается высокая производительность. Кэширование блоков диска позволяет более эффективно использовать память кэша и дисковое пространство.

Во-вторых, необходимо определить правило замены данных при заполнении кэш-памяти. Здесь можно использовать любой стандартный алгоритм кэширования, например, алгоритм LRU (least recently used), соответствии с которым вытесняется блок, к которому дольше всего не было обращения.

Кэш-память на сервере легко реализуется и совершенно прозрачна для клиента. Так как сервер может синхронизировать работу памяти и диска, с точки зрения клиентов существует только одна копия каждого файла, так что проблема согласования не возникает.

Хотя кэширование на сервере исключает обмен с диском при каждом доступе, все еще остается обмен по сети. Существует только один путь избавиться от обмена по сети - это кэширование на стороне клиента, которое, однако, порождает много сложностей.

Так как в большинстве систем используется кэширование в памяти клиента, а не на его диске, то мы рассмотрим только этот случай. При проектировании такого варианта имеется три возможности размещения кэша (рисунок 3.11). Самый простой состоит в кэшировании файлов непосредственно внутри адресного пространства каждого пользовательского процесса. Обычно кэш управляется с помощью библиотеки системных вызов. По мере того, как файлы открываются, закрываются, читаются и пишутся, библиотека просто сохраняет наиболее часто используемые файлы. Когда процесс завершается, все модифицированные файлы записываются назад на сервер. Хотя эта схема реализуется с чрезвычайно низкими издержками, она эффективна только тогда, когда отдельные процессы часто повторно открывают и закрывают файлы. Таким является процесс менеджера базы данных, но обычные программы чаще всего читают каждый файл однократно, так что кэширование с помощью библиотеки в этом случае не дает выигрыша.

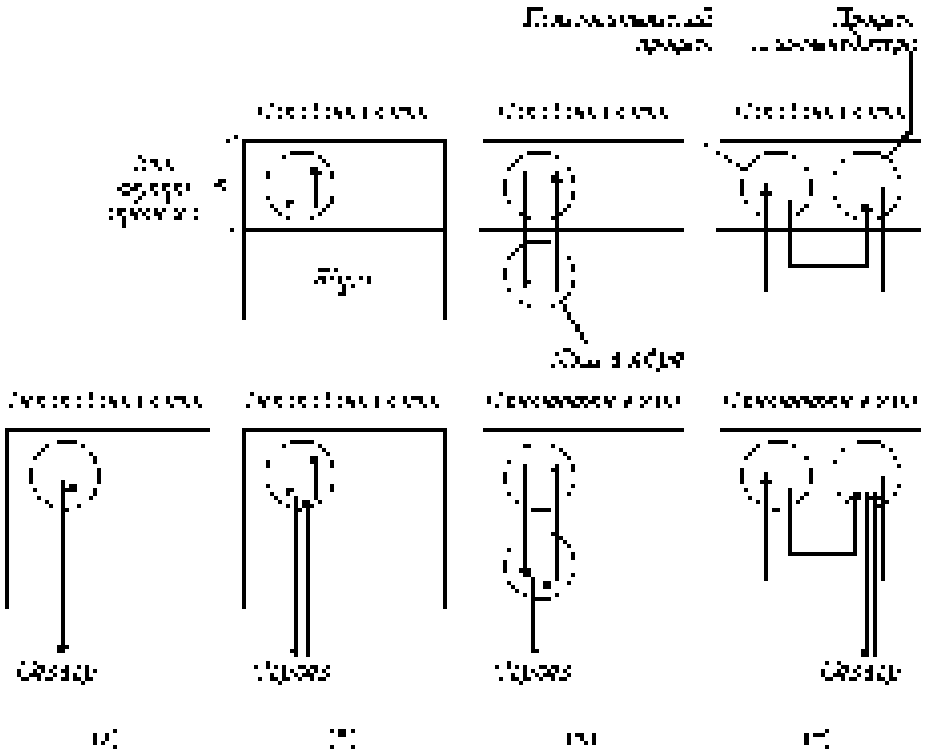


Рис. 3.11. Различные способы выполнения кэша в клиентской памяти
а - без кэширования; б - кэширование внутри каждого процесса; в - кэширование в ядре;
г - кэш-менеджер как пользовательский процесс

Другим местом кэширования является ядро. Недостатком этого варианта является то, что во всех случаях требуется выполнять системные вызовы, даже в случае успешного обращения к кэш-памяти (файл оказался в кэше). Но преимуществом является то, что файлы остаются в кэше и после завершения процессов. Например, предположим, что двухпроходный компилятор выполняется, как два процесса. Первый проход записывает промежуточный файл, который читается вторым проходом. На рисунке 3.11,в показано, что после завершения процесса первого прохода промежуточный файл, вероятно, будет находиться в кэше, так что вызов сервера не потребуется.

Третьим вариантом организации кэша является создание отдельного процесса пользовательского уровня - кэш-менеджера. Преимущество этого подхода заключается в том, что ядро освобождается от кода файловой системы и тем самым реализуются все достоинства микроядер.

С другой стороны, когда ядро управляет кэшем, оно может динамически решить, сколько памяти выделить для программ, а сколько для кэша. Когда же кэш-менеджер пользовательского уровня работает на машине с виртуальной памятью, то понятно, что ядро может решить выгрузить некоторые, или даже все страницы кэша на диск, так что для так называемого "попадания в кэш" требуется подкачка одной или более страниц. Нечего и говорить, что это полностью дискредитирует идею кэширования. Однако, если в системе имеется возможность фиксировать

некоторые страницы в памяти, то такая парадоксальная ситуация может быть исключена.

Как и везде, нельзя получить что-либо, не заплатив чем-то за это. Кэширование на стороне клиента вносит в систему проблему несогласованности данных.

Одним из путей решения проблемы согласования является использование алгоритма сквозной записи. Когда кэшируемый элемент (файл или блок) модифицируется, новое значение записывается в кэш и одновременно посылается на сервер. Теперь другой процесс, читающий этот файл, получает самую последнюю версию.

Один из недостатков алгоритма сквозной записи состоит в том, что он уменьшает интенсивность сетевого обмена только при чтении, при записи интенсивность сетевого обмена та же самая, что и без кэширования. Многие разработчики систем находят это неприемлемым и предлагают следующий алгоритм, использующий отложенную запись: вместо того, чтобы выполнять запись на сервер, клиент просто помечает, что файл изменен. Примерно каждые 30 секунд все изменения в файлах собираются вместе и отсылаются на сервер за один прием. Одна большая запись обычно более эффективна, чем много маленьких.

Следующим шагом в этом направлении является принятие сессионной семантики, в соответствии с которой запись файла на сервер производится только после его закрытия. Этот алгоритм называется "запись-по-закрытию". Как мы видели раньше, этот путь приводит к тому, что если две копии одного файла кэшируются на разных машинах и последовательно записываются на сервер, то второй записывается поверх первого. Однако это не так уж плохо, как кажется на первый взгляд. В однопроцессорной системе два процесса могут открыть и читать файл, модифицировать его в своих адресных пространствах, а затем записать его назад. Следовательно, алгоритм "запись-по-закрытию", основанный на сессионной семантике, не намного хуже варианта, уже используемого в однопроцессорной системе.

Совершенно отличный подход к проблеме согласования - это использование алгоритма централизованного управления (этот подход соответствует семантике UNIX). Когда файл открыт, машина, открывшая его, посылает сообщение файловому серверу, чтобы оповестить его об этом факте. Файл-сервер сохраняет информацию о том, кто открыл какой файл, и о том, открыт ли он для чтения, для записи, или для того и другого. Если файл открыт для чтения, то нет никаких препятствий для разрешения другим процессам открыть его для чтения, но открытие его для записи должно быть запрещено. Аналогично, если некоторый процесс открыл файл для записи, то все другие виды доступа должны быть предотвращены. При закрытии файла также необходимо оповестить файл-сервер для того, чтобы он обновил свои таблицы, содержащие данные об открытых файлах. Модифицированный файл также может быть выгружен на сервер в такой момент.

Четыре алгоритма управления кэшированием обобщаются следующим образом:

1. *Сквозная запись.* Этот метод эффективен частично, так как уменьшает интенсивность только операций чтения, а интенсивность операций записи остается неизменной.

2. *Отложенная запись.* Производительность лучше, но результат чтения кэшированного файла не всегда однозначен.

3. *"Запись-по-закрытию".* Удовлетворяет сессионной семантике.

4. *Централизованное управление.* Ненадежен вследствие своей централизованной природы.

Подводя итоги обсуждения проблемы кэширования, нужно отметить, что кэширование на сервере несложно реализуется и почти всегда дает эффект, независимо от того, реализовано кэширование у клиента или нет. Кэширование на сервере не влияет на семантику файловой системы, видимую клиентом. Кэширование у клиента напротив дает увеличение производительности, но увеличивает и сложность семантики.

Репликация

Распределенные системы часто обеспечивают репликацию (тиражирование) файлов в качестве одной из услуг, предоставляемых клиентам. Репликация - это асинхронный перенос изменений данных исходной файловой системы в файловые системы, принадлежащие различным узлам распределенной файловой системы. Другими словами, система оперирует несколькими копиями файлов, причем каждая копия находится на отдельном файловом сервере. Имеется несколько причин для предоставления этого сервиса, главными из которых являются:

1. Увеличение надежности за счет наличия независимых копий каждого файла на разных файл-серверах.

2. Распределение нагрузки между несколькими серверами.

Как обычно, ключевым вопросом, связанным с репликацией является прозрачность. До какой степени пользователи должны быть в курсе того, что некоторые файлы реплицируются? Должны ли они играть какую-либо роль в процессе репликации или репликация должна выполняться полностью автоматически? В одних системах пользователи полностью вовлечены в этот процесс, в других система все делает без их ведома. В последнем случае говорят, что система репликационно прозрачна.

На рисунке 3.12 показаны три возможных способа репликации. При использовании первого способа (а) программист сам управляет всем процессом

репликации. Когда процесс создает файл, он делает это на одном определенном сервере. Затем, если пожелает, он может сделать дополнительные копии на других серверах. Если сервер каталогов разрешает сделать несколько копий файла, то сетевые адреса всех копий могут быть ассоциированы с именем файла, как показано на рисунке снизу, и когда имя найдено, это означает, что найдены все копии. Чтобы сделать концепцию репликации более понятной, рассмотрим, как может быть реализована репликация в системах, основанных на удаленном монтировании, типа UNIX. Предположим, что рабочий каталог программиста имеет имя */machine1/usr/ast*. После создания файла, например, */machine1/usr/ast/xyz*, программист, процесс или библиотека могут использовать команду копирования для того, чтобы сделать копии */machine2/usr/ast/xyz* и *machine3/usr/ast/xyz*. Возможно программа использует в качестве аргумента строку */usr/ast/xyz* и последовательно попытается открывать копии, пока не достигнет успеха. Эта схема хотя и работает, но имеет много недостатков, и по этим причинам ее не стоит использовать в распределенных системах.

На рисунке 3.12,б показан альтернативный подход - ленивая репликация. Здесь создается только одна копия каждого файла на некотором сервере. Позже сервер сам автоматически выполнит репликации на другие серверы без участия программиста. Эта система должна быть достаточно быстрой для того, чтобы обновлять все эти копии, если потребуется.

Последним рассмотрим метод, использующий групповые связи (рисунок 3.12,в). В этом методе все системные вызовы ЗАПИСАТЬ передаются одновременно на все серверы, таким образом копии создаются одновременно с созданием оригинала. Имеется два принципиальных различия в использовании групповых связей и ленивой репликации. Во-первых, при ленивой репликации адресуется один сервер, а не группа. Во-вторых, ленивая репликация происходит в фоновом режиме, когда сервер имеет промежуток свободного времени, а при групповой репликации все копии создаются в одно и то же время.

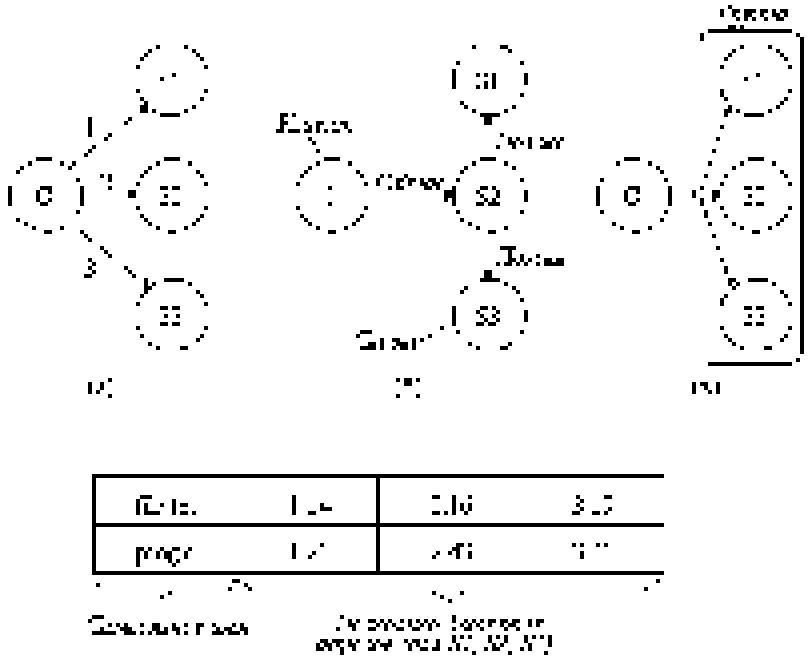


Рис. 3.12. а) Точная репликация файла; б) Ленивая репликация файла; в) Репликация файла, использующая группу

Рассмотрим, как могут быть изменены существующие реплицированные файлы. Существует два хорошо известных алгоритма решения этой проблемы.

Первый алгоритм, называемый "репликация первой копии", требует, чтобы один сервер был выделен как первичный. Остальные серверы являются вторичными. Когда реплицированный файл модифицируется, изменение посылается на первичный сервер, который выполняет изменения локально, а затем посылает изменения на вторичные серверы.

Чтобы предотвратить ситуацию, когда из-за отказа первичный сервер не успевает оповестить об изменениях все вторичные серверы, изменения должны быть сохранены в постоянном запоминающем устройстве еще до изменения первичной копии. В этом случае после перезагрузки сервера есть возможность сделать проверку, не проводились ли какие-нибудь обновления в момент краха. Недостаток этого алгоритма типичен для централизованных систем - пониженная надежность. Чтобы избежать его, используется метод, предложенный Гиффордом и известный как "голосование". Пусть имеется n копий, тогда изменения должны быть внесены в любые W копий. При этом серверы, на которых хранятся копии, должны отслеживать порядковые номера их версий. В случае, когда какой-либо сервер выполняет операцию чтения, он обращается с запросом к любым R серверам. Если R+W > n, то, хотя бы один сервер содержит последнюю версию, которую можно определить по максимальному номеру.

Интересной модификацией этого алгоритма является алгоритм "голосования с приведениями". В большинстве приложений операции чтения встречаются гораздо чаще, чем операции записи, поэтому R обычно делают небольшим, а W - близким к N. При этом выход из строя нескольких серверов приводит к

отсутствию кворума для записи. Голосование с приведениями решает эту проблему путем создания фиктивного сервера без дисков для каждого отказавшего или отключенного сервера. Фиктивный сервер не участвует в кворуме чтения (прежде всего, у него нет файлов), но он может присоединиться к кворуму записи, причем он просто записывает в никуда передаваемый ему файл. Запись только тогда успешна, когда хотя бы один сервер настоящий.

Когда отказавший сервер перезапускается, то он должен получить кворум чтения для обнаружения последней версии, которую он копирует к себе перед тем, как начать обычные операции. В остальном этот алгоритм подобен основному.

Проблемы взаимодействия операционных систем в гетерогенных сетях

Понятия "internetworking" и "interoperability"

До недавнего времени проблемы межсетевого взаимодействия не очень волновали отечественных пользователей и системных администраторов. Они уютно себя чувствовали в замкнутом мире IBM PC совместимых компьютеров, сетей Novell и сетевых адаптеров Ethernet, хотя в "большом" мире многие фирмы, в том числе и Novell, успешно продавали различные средства межсетевой связи. Однако пора монокультурного развития отечественных сетей заканчивается, организации приобретают различную технику, например, бизнес-серверы Hewlett-Packard, графические станции Sun или Silicon Graphics, мини-компьютеры AS-400 фирмы IBM и другую не менее достойную аппаратуру с разнообразными операционными системами, поэтому проблемы, характерные для западных корпоративных сетей, постепенно становятся актуальными и для нас.

Прежде, чем приступить к рассмотрению межсетевого взаимодействия, уточним, что понимается под термином "сеть". Этот термин может употребляться в широком смысле (сеть - это совокупность связанных между собой компьютеров) и в узком смысле (сеть - это совокупность компьютеров, соединенных между собой в соответствии с одной из стандартных типовых топологий - шина, звезда, кольцо, и использующих для передачи пакетов один из протоколов канального уровня, определенный для этой топологии).

Каждая сеть имеет свой номер, который используется на сетевом уровне при выполнении маршрутизации. Когда две или более сетей организуют совместную транспортную службу, то такой режим взаимодействия обычно называют *межсетевым взаимодействием (internetworking)*. Для обозначения составной сети в англоязычной литературе часто также используются термины *интерсеть*(*internetwork* или *internet).* Интерсеть обеспечивает только передачу пакетов, не занимаясь их содержанием.

Так как понятие "номер сети" определяется на сетевом уровне, то оно является относительным, то есть, если в одном компьютере установлено программное

обеспечение, поддерживающее несколько протоколов сетевого уровня (например, TCP и SPX), то данный компьютер может принадлежать нескольким сетям.

При рассмотрении вопросов межсетевого взаимодействия часто используется еще один англоязычный термин - *interoperability*. В то время как термин internetworking обозначает взаимодействие сетей на нижних уровнях, непосредственно связанных с транспортировкой пакетов, в понятие interoperability входит обеспечение согласования верхних уровней стека коммуникационных протоколов, реализуемых серверами и редиректорами операционных систем и некоторыми сетевыми приложениями.

Гетерогенность

Только небольшое количество сетей обладает *однородностью (гомогенностью)* программного и аппаратного обеспечения. Однородными чаще являются сети, которые состоят из небольшого количества компонентов от одного производителя.

Немногие организации имеют сети, составленные из оборудования, например, только IBM или DEC. Дни сетей от одного производителя миновали. Нормой сегодняшнего дня являются сети *неоднородные (гетерогенные),* которые состоят из различных рабочих станций, операционных систем и приложений, а для реализации взаимодействия между компьютерами используют различные протоколы. Разнообразие всех компонентов, из которых строится сеть, порождает еще большее разнообразие структур сетей, получающихся из этих компонентов. А если продолжить далее, и рассмотреть более сложные образования, получающиеся в результате объединения отдельных сетей в единую большую сеть, то становится понятным то множество проблем, связанных с проектированием, администрированием и управлением такой гетерогенной интерсетью. В идеале это объединение неоднородных сетей должно быть прозрачным для пользователя.

Так как сети создавались большей частью случайным образом, то приобретенные компьютеры и ОС отвечают, как правило, индивидуальным потребностям группы пользователей. Сети отдела строились для решения конкретных задач групп сотрудников. Например, инженерный отдел мог выбрать рабочие станции SPARC фирмы Sun Microsystems, соединенные сетью Ethernet, потому что им нужны были приложения, работающие только в среде UNIX. Разделение файлов при этом реализовывалось с помощью TCP/IP и NFS. В отделе продаж той же самой организации уже могли быть куплены компьютеры PS/2, установлена сеть Token Ring и операционная система NetWare для решения их собственных задач: ведения базы данных о клиентах, подготовки писем, разработки коммерческих предложений. Затем в рекламном отделе были выбраны компьютеры Macintosh, поскольку они наилучшим образом подходят для создания презентационных материалов. Macintosh'и соединены посредством LocalTalk, а файлы и принтеры разделяются с использованием AppleTalk. Отдел, отвечающий за автоматизацию

предприятия, должен*интегрировать* все эти плохо совместимые системы в единый прозрачный организм.

Добавление в вычислительную сеть новых, чужеродных элементов может происходить и при всякой значительной реорганизации предприятия, например, при смене владельца, что для нашей страны сейчас тоже становится весьма обычным делом. В этом случае вновь приобретенное предприятие и его вычислительное оборудование также должно быть интегрировано в общую структуру предприятия нового владельца.

Основные подходы к реализации взаимодействия сетей

Основные проблемы при организации взаимодействия различных сетей связаны с тем, что эти сети используют различные стеки коммуникационных протоколов. В каждом конкретном стеке протоколов, будь то стек DoD или Novell NetWare, средства, реализующие какой-либо уровень, обеспечивают интерфейс для вышележащего уровня своей системы и пользуются услугами интерфейсных функций нижележащего уровня. Например, средства реализации протокола Novell IPX в сервере предоставляют интерфейсные услуги протоколу NCP для приема запросов от рабочих станций и пересылки им ответов. В свою очередь протокол IPX пользуется интерфейсными функциями драйвера сетевого адаптера Ethernet, чтобы передать пакет для отправки в сеть.

Если бы в компьютерном мире существовал только один стек протоколов, то у независимых разработчиков сетевого и программного обеспечения не было бы никаких проблем: сетевые адаптеры вместе со своими драйверами подходили бы к любой сетевой ОС за счет единого интерфейса между канальным и сетевым уровнями, разработчики транспортных средств новых ОС могли бы использовать существующие реализации протокола сетевого уровня, а разработчики сетевых приложений использовали бы единый API для обращения к сервисным услугам прикладного уровня ОС.

К сожалению, в реальном мире компьютерных сетей существует несколько стеков протоколов, уже завоевавших свое место под солнцем и не собирающихся его уступать. Например, если на предприятии используются мейнфреймы IBM, то они скорее всего используют протоколы сетевой архитектуры SNA и аппаратуру Token Ring. Использование компьютеров DEC с операционной системой VAX означает, что используются протоколы DECnet и Ethernet. Сети локальных компьютеров используют чаще всего протоколы Novell NetWare, Banyan VINES, IBM LAN Server или Microsoft LAN Manager с аппаратурой Ethernet, Token Ring или ARCnet.

Существование многих стеков протоколов не вносит никаких проблем до тех пор, пока не появляется потребность в их взаимодействии, то есть потребность в доступе пользователей сети NetWare к мейнфрейму IBM или пользователей графических рабочих станций UNIX к компьютеру VAX. В этих случаях

проявляется несовместимость близких по назначению, но различных по форматам данных и алгоритмам протоколов.

Общность различных стеков протоколов проявляется только на нижних уровнях - физическом и канальном. Здесь в настоящее время почти нет проблем для взаимодействия, так как большинство стеков могут использовать общие протоколы Ethernet, Token Ring, FDDI. Исключение составляют только мейнфреймы IBM, которые на нижнем уровне в основном используют протоколы типа ведущий-ведомый с синхронной передачей данных, ориентированные на иерархическую соподчиненную структуру мейнфрейм - групповой контроллер - терминалы. Да и соединение двух компьютеров, использующих на нижнем уровне различные протоколы, а на верхних - одинаковые не составляет проблемы - эта задача решается аппаратно с помощью транслирующего моста или маршрутизатора.

Сложнее обстоит дело с сопряжением сетей, использующие различные протоколы верхних уровней, начиная с сетевого. Задачи согласования протоколов верхних уровней решить труднее из-за большей сложности этих протоколов и их разнообразия - чем большим интеллектом обладает протокол, тем больше у него аспектов и граней, по которым он может отличаться от своего собрата по функциональному назначению. Сложно осуществить трансляцию транспортных протоколов (таких, как IP и IPX), но гораздо сложнее совместить протоколы верхнего, прикладного уровня, с помощью которых клиенты получают сервис у серверов.

Если рассмотреть наиболее часто используемый в сетях сервис, а именно, файловый сервис, то различия в протоколах файлового сервиса в первую очередь связаны с различиями структур файловых систем. Например, пользователю MS-DOS непривычны приемы монтирования файловой системы UNIX в одно дерево, он хочет работать с разрозненными файловыми системами отдельных носителей, отображенными на буквы английского алфавита. Команды, используемые при работе с различными файловыми системами, также различны как по названию, так и по содержанию. Кроме того, даже для одной файловой системы в различных операционных системах предусмотрены различные удаленные сервисы. В ОС UNIX можно работать с удаленной файловой системой с помощью символьных команд протокола прикладного уровня FTP, переписывая файлы с удаленной машины на локальную по одному, а можно работать с протоколом NFS, который обеспечивает монтирование удаленной системы в локальную и требует других команд и приемов. Поэтому проблемы, возникающие на верхних уровнях, гораздо сложнее, чем проблемы замены заголовка пакета на канальном уровне.

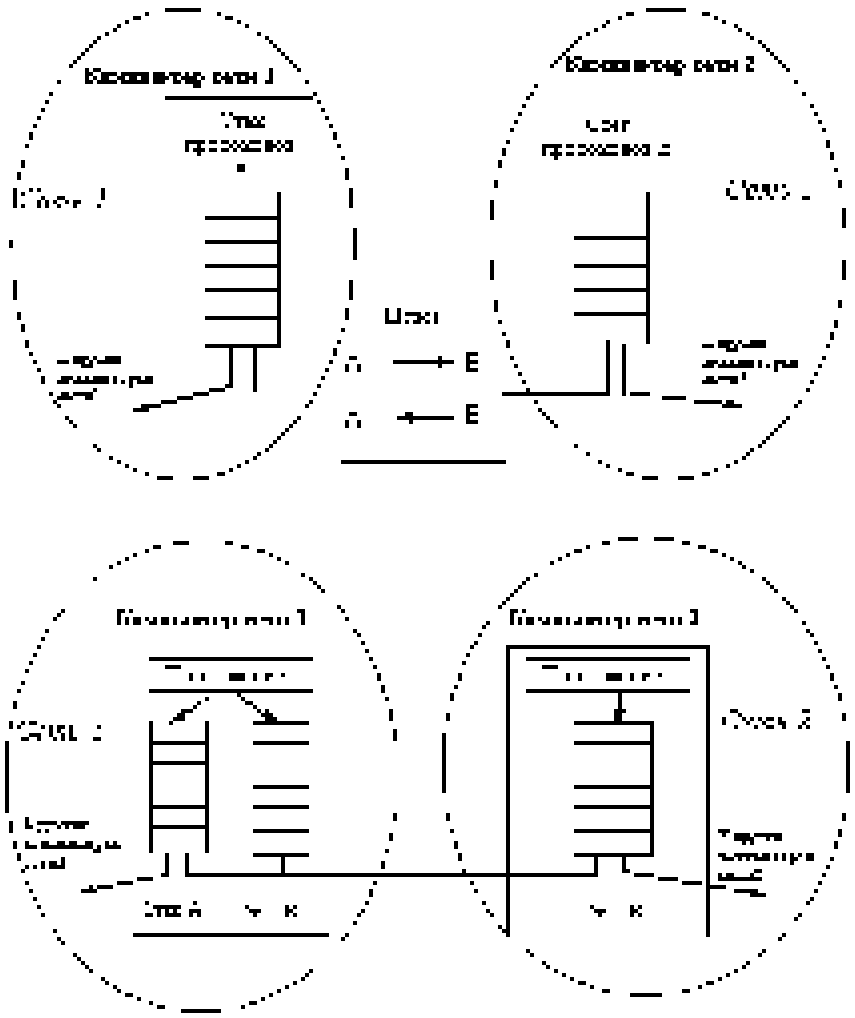


Рис. 3.13. Два основных варианта согласования протоколов
а - трансляция протоколов; б - мультиплексирование стеков протоколов

Для организации взаимодействия различных сетей в настоящее время используется два подхода.

Первый подход связан с использованием так называемых *шлюзов*, которые обеспечивают согласование двух стеков протоколов путем преобразования (трансляции) протоколов (рисунок 3.13,а). Шлюз размещается между взаимодействующими сетями и служит посредником, переводящим сообщения, поступающие от одной сети, в формат другой сети.

Второй подход заключается в том, что в операционные системы серверов и рабочих станций встраиваются несколько мирно сосуществующих наиболее популярных стеков протоколов. Такая технология получила название *мультиплексирования стеков протоколов*. За счет ее использования либо клиентские запросы используют стек протоколов той сети, к которой относятся нужные серверы, либо серверы подключают стек протоколов, соответствующий поступившему клиентскому запросу (рисунок 3.13,б).

Взаимодействие компьютеров, принадлежащих разным сетям, напоминает общение людей, говорящих на разных языках. Для достижения взаимопонимания

они также могут использовать два подхода: пригласить переводчика (аналог шлюза), или перейти на язык собеседника, если они им владеют (аналог мультиплексирования стеков протоколов).

Каждый из подходов имеет свои преимущества и недостатки, на которых мы остановимся позже.

Шлюзы

Итак, шлюз согласует коммуникационные протоколы одного стека с коммуникационными протоколами другого стека. Программные средства, реализующие шлюз, нет смысла устанавливать ни на одном из двух взаимодействующих компьютеров с разными стеками протоколов, гораздо рациональнее разместить их на некотором компьютере-посреднике. Прежде, чем обосновать это утверждение, рассмотрим принцип работы шлюза.

Рисунок 3.14 иллюстрирует принцип функционирования шлюза. В показанном примере шлюз, размещенный на компьютере 2, согласовывает протоколы клиентского компьютера 1 сети А с протоколами серверного компьютера 3 сети В. Допустим, что две сети используют полностью отличающиеся стеки протоколов. Как видно из рисунка, в шлюзе реализованы оба стека протоколов.

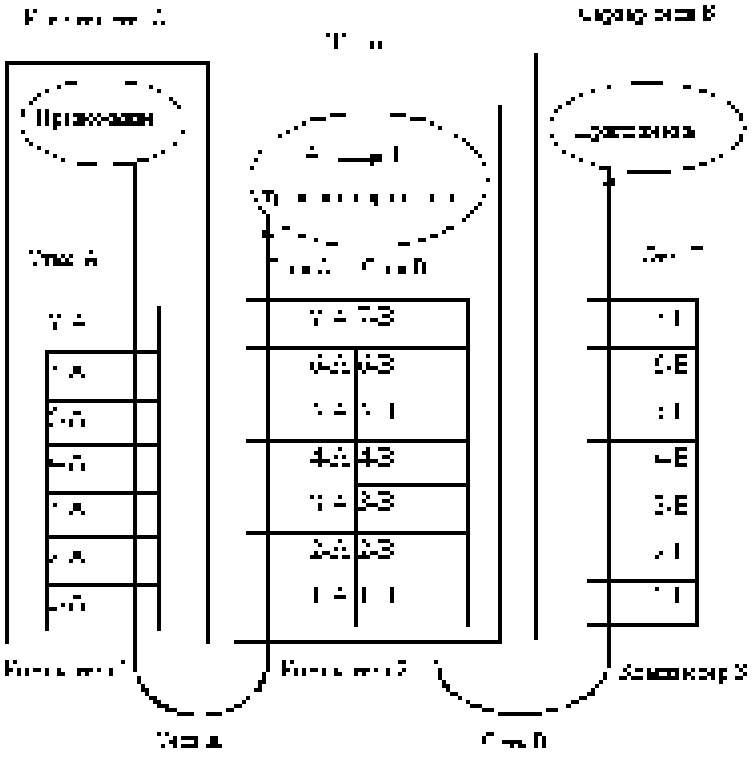


Рис. 3.14. Принципы функционирования шлюза

Запрос от прикладного процесса клиентского компьютера сети А поступает на прикладной уровень его стека протоколов. В соответствии с этим протоколом на прикладном уровне формируются соответствующий пакет (или несколько

пакетов), в которых передается запрос на выполнение сервиса некоторому серверу сети В. Пакет прикладного уровня передается вниз по стеку компьютера сети А, а затем в соответствии с протоколами канального и физического уровней сети А поступает в компьютер 2, то есть в шлюз.

Здесь он передается от самого нижнего к самому верхнему уровню стека протоколов сети А. Затем пакет прикладного уровня стека сети А преобразуется (транслируется) в пакет прикладного уровня серверного стека сети В. Алгоритм преобразования пакетов зависит от конкретных протоколов и, как уже было сказано, может быть достаточно сложным. В качестве общей информации, позволяющей корректно провести трансляцию, может использоваться, например, информация о символьном имени сервера и символьном имени запрашиваемого ресурса сервера (в частности, это может быть имя каталога файловой системы). Преобразованный пакет от верхнего уровня стека сети В передается к нижним уровням в соответствии с правилами этого стека, а затем по физическим линиям связи в соответствии с протоколами физического и канального уровней сети В поступает в другую сеть к нужному серверу. Ответ сервера преобразуется шлюзом аналогично.

Мультиплексирование стеков протоколов

Вторым использующимся в настоящее время на практике подходом является использование в рабочих станциях технологии мультиплексирования различных стеков протоколов.

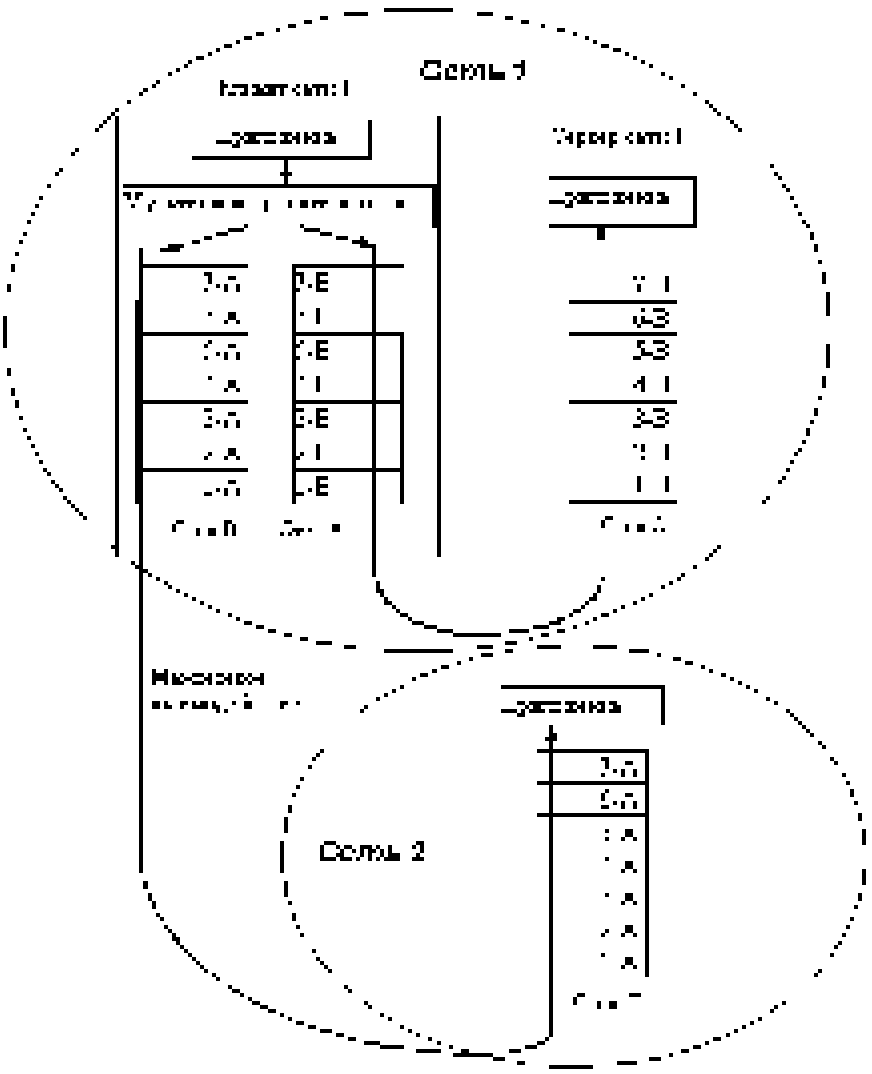


Рис. 3.15. Мультиплексирование стеков

При мультиплексировании стеков протоколов на один из двух взаимодействующих компьютеров с различными стеками протоколов помещается коммуникационный стек другого компьютера. На рисунке 3.15 приведен пример взаимодействия клиентского компьютера сети 1 с сервером своей сети и сервером сети 2, работающей со стеком протоколов, полностью отличающимся от стека сети 1. В клиентском компьютере реализованы оба стека. Для того, чтобы запрос от прикладного процесса был правильно обработан и направлен через соответствующий стек, в компьютер необходимо добавить специальный программный элемент - мультиплексор протоколов. Мультиплексор должен уметь определять, к какой сети направляется запрос клиента. Для этого может использоваться служба имен сети, в которой отмечается принадлежность того или иного ресурса определенной сети с соответствующим стеком протоколов.

При использовании технологии мультиплексирования структура коммуникационных средств операционной системы может быть и более сложной. В общем случае на каждом уровне вместо одного протокола появляется целый набор протоколов, а мультиплексоров может быть несколько,

выполняющих коммутацию между протоколами разных уровней (рисунок 3.16). Например, рабочая станция может получить доступ к сетям с протоколами NetBIOS, IP, IPX через один сетевой адаптер. Аналогично, сервер, поддерживающий прикладные протоколы NCP, SMB и NFS может без проблем выполнять запросы рабочих станций сетей NetWare, Windows NT и Sun одновременно.

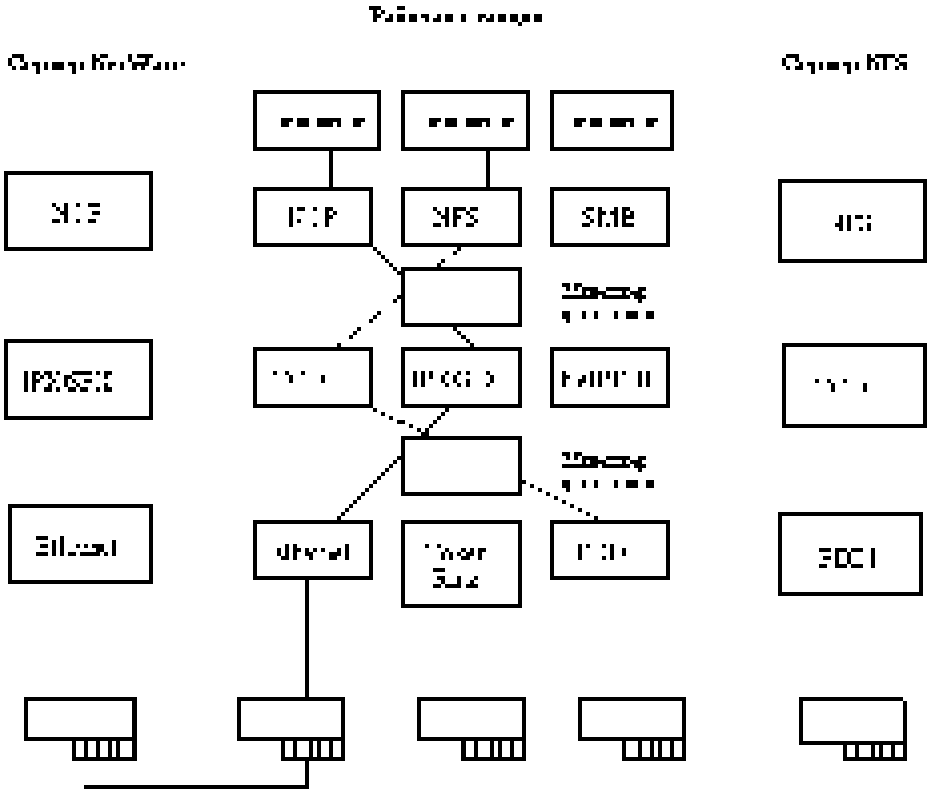


Рис. 3.16. Мультиплексирование протоколов

Предпосылкой для развития технологии мультиплексирования стеков протоколов стало строгое определения протоколов и интерфейсов различных уровней и их открытое описание, так, чтобы фирма при реализации "чужого" протокола или интерфейса могла быть уверена, что ее продукт будет правильно взаимодействовать с продуктами других фирм по данному протоколу.

Использование магистрального протокола

Хорошим решением был бы переход на единый стек протоколов, но вряд ли эта перспектива осуществится в ближайшем будущем. Попытка введения единого стека коммуникационных протоколов сделана в 1990 году правительством США, которое обнародовало программу GOSIP - Government OSI Profile, в соответствии с которой стек протоколов OSI должен стать общим знаменателем для всех сетей, устанавливаемых в правительственных организациях США. Но, понимая бесполезность силовых мер, программа GOSIP не ставит задачу немедленного перехода на стек OSI, а принуждает пока к использованию этого стека в качестве "второго языка" правительственных сетей, наряду с родным, первым.

Вопросы реализации

При объединении сетей различных типов в общем случае необходимо обеспечить двухстороннее взаимодействие сетей, то есть решить две задачи (рисунок 3.17):

1. Обеспечение доступа клиентам сети А к ресурсам и сервисам серверов сети В.

2. Обеспечение доступа клиентам сети В к ресурсам и сервисам сети А.

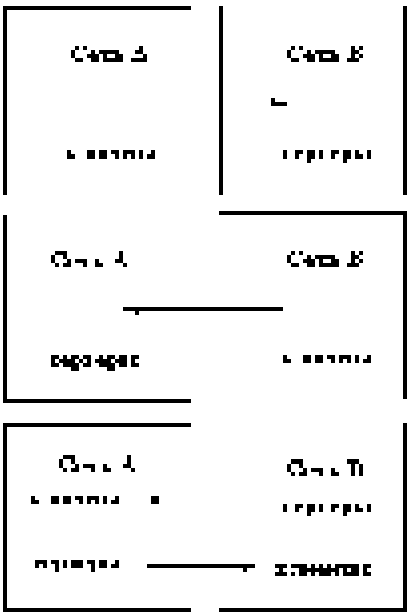


Рис. 3.17. Варианты сетевого взаимодействия

Эти задачи независимы и их можно решать отдельно. Прежде всего нужно понять, необходимо ли полное решение или достаточно и частичного, то есть нужно ли, чтобы пользователи, например, UNIX-машин имели доступ к ресурсам серверов сети NetWare, а пользователи персональных машин имели доступ к ресурсам UNIX-хостов, или же достаточно обеспечить доступ к ресурсам другой сети только одному виду пользователей.

Кроме того, каждую из этих задач можно в свою очередь разделить на части. В сети обычно имеются различные виды разделяемых ресурсов, и с каждым типом ресурсов могут предоставляться различные виды сервиса. Например, в UNIX-сетях файлы являются разделяемым ресурсом, и с ними связаны два вида сервиса - перемещение файлов между машинами по протоколу FTP и монтирование удаленной файловой системы по протоколу NFS. Поэтому при объединении сетей можно предложить пользователям набор средств, каждое из которых позволяет воспользоваться одним каким-либо сервисом чужой сети. Естественно, возможно объединение всех функций в рамках одного продукта.

При объединении сетей достаточно иметь средства взаимодействия сетей только в одной из сетей. Например, фирма Novell разработала ряд программных продуктов для связи с UNIX-сетями, которые достаточно включить в

программное обеспечение сети NetWare, чтобы решить обе указанные задачи взаимодействия сетей. При этом серверной части UNIX клиент NetWare представляется UNIX-клиентом, а клиент UNIX обращается с файлами и принтерами, управляемыми сервером NetWare, как с UNIX-файлами и UNIX-принтерами. Возможен перенос средств взаимодействия сетей и на сторону UNIX-сети. Тогда аналогичные функции будут выполнять программные средства на UNIX-машине.

В то время, как расположение программных средств, реализующих шлюз, уже было определено - они должны располагаться на компьютере, занимающем промежуточное положение между двумя взаимодействующими машинами, вопрос о размещении дополнительных стеков протоколов остался открытым. Заметим также, что шлюз реализует взаимодействие "многие-ко-многим" (все клиенты могут обращаться ко всем серверам).

Рассмотрим все возможные варианты размещения программных средств, реализующих взаимодействие двух сетей, которые основаны на мультиплексировании протоколов. Введем некоторые обозначения: С - сервер, К - клиент, (- дополнительный протокол или стек протоколов.

На рисунке 3.18 показаны оба возможных варианта *однонаправленного* взаимодействия A®B: а) путем добавления нового стека к клиентам сети А, либо б) путем присоединения "добавки" к серверам сети В.

В первом случае, когда средства мультиплексирования располагаются на клиентских частях, только клиенты, снабженные средствами мультиплексирования протоколов, могут обращаться к серверам сети В, при этом они могут обращаться ко всем серверам сети В. Во втором случае, когда набор стеков расположен на каком-либо сервере сети В, данный сервер может обслуживать всех клиентов сети А. Очевидно, что серверы сети В без средств мультиплексирования не могут быть использованы клиентами сети А.

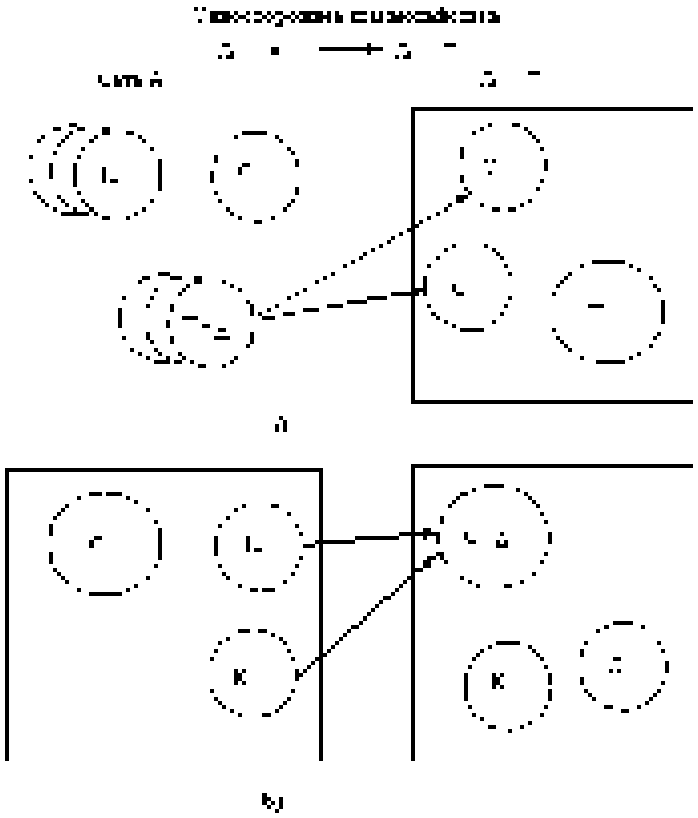


Рис. 3.18. Варианты размещения программных средств
(С - сервер, К - клиент, (- средства сетевого взаимодействия)

Примером "добавки", модифицирующей клиентскую часть, может служить популярное программное средство фирмы Novell LAN Workplace, которое превращает клиента NetWare в клиента UNIX. Аналогичным примером для модификации сервера могут служить другие продукты фирмы Novell: NetWare for UNIX, который делает возможным использование услуг сервера UNIX клиентами NetWare, или Novell NetWare for VMS, который служит для тех же целей в сети VMS.

Взаимодействие А (В реализуется симметрично.

Если же требуется реализовать взаимодействие *в обе стороны* одновременно, то для этого существует четыре возможных варианта, показанных на рисунке 3.19. Каждый вариант имеет свои особенности с точки зрения возможностей связи клиентов с серверами:

1. Средства обеспечения взаимодействия расположены только на клиентских частях обеих сетей. Для тех и *только тех* клиентов обеих сетей, которые оснащены "добавками", гарантируется возможность связи *со всеми* серверами из "чужой" сети.
2. Все средства обеспечения взаимодействия расположены на стороне сети А. *Все* клиенты сети В могут обращаться к серверам сети А (*не ко всем*, а только к тем, которые имеют сетевую "добавку").*Часть* клиентов сети А, которые обозначены как К+(, могут обращаться *ко всем* серверам сети В.

3. Средства межсетевого взаимодействия расположены только на серверных частях обеих сетей. *Всем*клиентам обеих сетей гарантируется возможность работы с серверами "чужих" сетей, но *не со всеми*, а только с серверами, обладающими сетевыми средствами мультиплексирования протоколов.
4. Все средства межсетевого взаимодействия расположены на стороне В. Двусторонний характер взаимодействия обеспечивается модификацией и клиентских, и серверных частей сети В. Все клиенты сети А могут обращаться за сервисом к серверам сети В, обозначенным как С+(, а все серверы сети А могут обслуживать клиентов сети В, обозначенных как К+(.

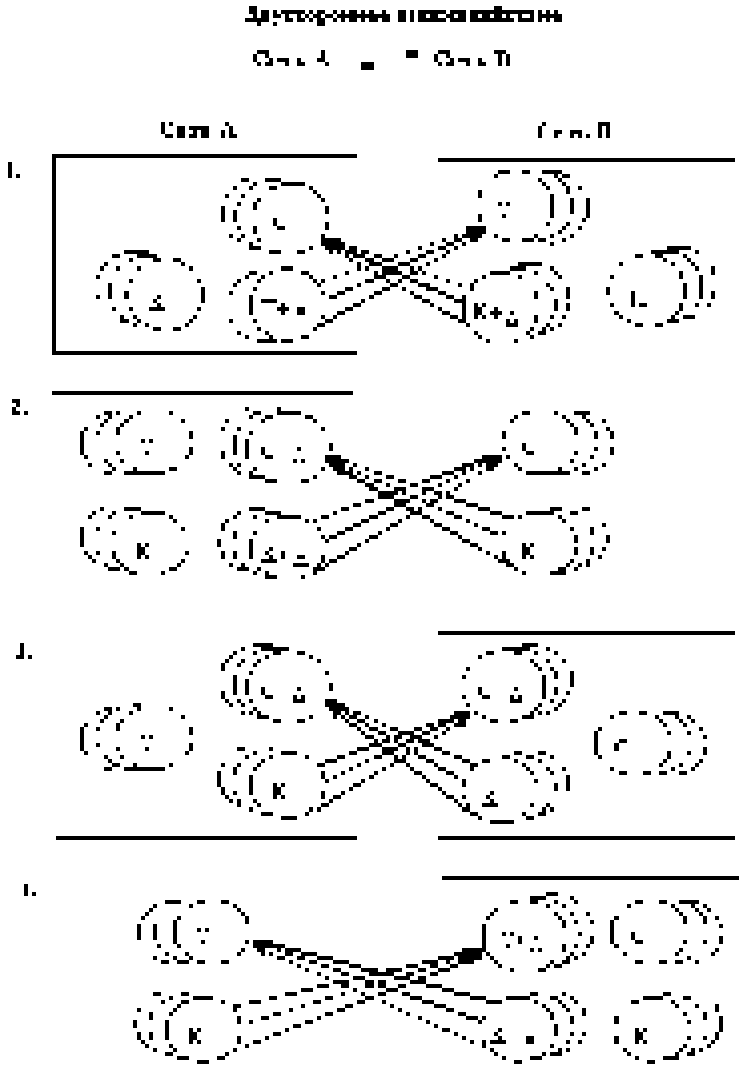


Рис. 3.19. Варианты размещения программных средств при двустороннем взаимодействии
(С - сервер, К - клиент, (- средства сетевого взаимодействия)

Очевидно, что наличие программных продуктов для каждого из рассмотренных вариантов сильно зависит от конкретной пары операционных систем. Для некоторых пар может вовсе не найтись продуктов межсетевого взаимодействия, а для некоторых можно выбирать из нескольких вариантов. Рассмотрим в качестве примера набор программных продуктов, реализующих взаимодействие

Windows NT и NetWare. В ОС Windows NT и в серверной части (Windows NT Server), и в клиентских частях (Windows NT Workstation) предусмотрены встроенные средства мультиплексирования нескольких протоколов, в том числе и стека IPX/SPX. Следовательно эта операционная система может поддерживать двустороннее взаимодействие (по варианту 2) с NetWare без каких-либо дополнительных программных средств. Аналогичным образом реализуется взаимодействие сетей Windows NT с UNIX-сетями.

Сравнение вариантов организации взаимодействия сетей

Возвращаясь к принципам организации взаимодействия сетей, сравним два основных подхода - мультиплексирование протоколов и трансляцию протоколов (шлюзы).

Встроенные в сетевую ОС средства мультиплексирования протоколов дают все те преимущества, которые присущи встроенным средствам:

- Эти средства не нужно отдельно приобретать;
- Нет проблем их совместимости с другими продуктами.

Основным недостатком этого подхода является *избыточность*. Хотя средства мультиплексирования обычно позволяют загружать и выгружать по желанию пользователя различные стеки протоколов, но если нужно одновременно работать с тремя различными сетями, то в каждую рабочую станцию необходимо загрузить все три стека одновременно.

Шлюз по своей природе является *выделенным* сервисом, разделяемым всеми источниками запросов к серверам другой сети. Использование шлюзов обеспечивает следующие преимущества:

- Позволяет сосредоточить все функции согласования протоколов в одном месте и разгрузить рабочие станции от дополнительного программного обеспечения, а их пользователей - от необходимости его генерации. Шлюз сохраняет в локальной сети ее родную среду протоколов, что повышает производительность, так как стек протоколов был специально спроектирован для данной операционной среды и наилучшим образом учитывает ее особенности.
- Возникающие проблемы легко локализуются.
- Обслуживающий персонал работает в привычной среде, где можно использовать имеющийся опыт по поддержанию сети. Шлюзы сохраняют различные, несовместимые сети в их первозданном виде. Если имеется несколько различных сетей, то для их совместной работы может понадобиться значительное количество шлюзов. Для доступа пользователей сети UNIX к мейнфрейму понадобится шлюз UNIX-SNA, для подключения пользователей NetWare к компьютерам UNIX и мейнфрейму нужно два шлюза - NetWare-UNIX и NetWare-SNA.

Недостатки использования шлюзов:

- Шлюзы работают, как правило, медленно; пользователи замечают уменьшение производительности при обращении к другой сети через шлюз.
- Шлюз как централизованное средство понижает надежность сети.

Службы именования ресурсов и проблемы прозрачности доступа

Подобно большой организации, большая 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, ведение которой поручить сетевой операционной системе. Данные из этой базы могут быть востребованы многими сетевыми системными приложениями, в первую очередь системами управления и администрирования. Кроме этого, такая база полезна при организации электронной почты, систем коллективной работы, службы безопасности, службы инвентаризации программного и аппаратного обеспечения сети, да и для практически любого крупного бизнес-приложения.

Хотя полезных применений единой справочной службы много, она нужна по крайней мере для эффективного решения задачи администрирования, то есть ведения учетной информации на пользователей сети и определения прав доступа этих пользователей к разделяемым ресурсам сети. Эта задача всегда решалась каким-либо способом во всех многопользовательских операционных системах, не обязательно сетевых. В локальных версиях UNIX имеются файлы с предопределенными именами, хранящие эту информацию - например, файл /etc/passwd хранит информацию о пользователях и их паролях, а также о группах пользователей.

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 операционных системах. 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.

Единая 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. Набор разрозненных баз данных не предоставляет такого прозрачного доступа к ресурсам сети, как это имеет место в случае использования ОС NetWare 3.x с ее базой bindery, локальной для каждого сервера. В последнем случае пользователь должен заранее знать, на каком сервере находится нужный ему ресурс и производить логическое подключение к этому серверу для получения доступа к этому ресурсу. Для того, чтобы получить доступ к ресурсам какого-нибудь сервера, пользователь должен иметь там свою учетную информацию, которая дублируется таким образом на всех серверах сети. В единой базе данных о каждом пользователе существует только одна запись.

Но за удобства приходится расплачиваться решением проблем распределенности, репликации и синхронизации, которые возникают при построении крупномасштабной базы данных для большой сети.

Реализация справочной службы над полностью централизованной базой данных, хранящейся только в виде одной копии на одном из серверов сети, не подходит для большой системы по нескольким причинам, и в первую очередь из-за низкой производительности и низкой надежности такого решения. Производительность будет низкой из-за того, что запросы на логический вход всех пользователей будут поступать в единственный сервер, который при большом количестве пользователей обязательно перестанет справляться с их обработкой, то есть такое решение плохо масштабируется в отношении количества пользователей и разделяемых ресурсов. Надежность также не может быть высокой в системе с единственной копией данных. Кроме снятия ограничений по производительности и надежности, желательно, чтобы структура базы данных позволяла производить логическое группирование ресурсов и пользователей по структурным подразделениям предприятия и назначать для каждой такой группы своего администратора.

Проблемы сохранения производительности и надежности при увеличении масштаба сети решаются за счет использования распределенных баз данных справочной информации. Разделение данных между несколькими серверами снижает нагрузку на каждый сервер, а надежность при этом достигается за счет наличия нескольких копий (называемых часто репликами) каждой части базы данных. Для каждой части базы данных можно назначить своего администратора, который обладает правами доступа только к объектам своей порции информации о всей системе. Для пользователя же (и для сетевых приложений) такая распределенная база данных представляется единой базой данных, которая

обеспечивает доступ ко всем ресурсам сети вне зависимости от того, с какой рабочей станции осуществил свой вход в сеть пользователь.

Существует два подхода к организации справочной службы сети: доменный и глобальный. Рассмотрим эти подходы на конкретных примерах - доменной справочной службе ОС Windows NT и глобальной справочной службе NDS ОС NetWare. Естественно, это не единственные операционные системы, где такие службы имеются - доменная служба реализована в Microsoft LAN Manager и IBM LAN Server, а глобальная справочная служба - в ОС Banyan VINES (служба Streettalk III). Более того, существует стандарт X.500, разработанный МККТТ для глобальной справочной службы почтовых систем, который с успехом может применяться и применяется для хранения любой справочной информации.

Доменный подход

Домен - это основная единица администрирования и обеспечения безопасности в Windows NT. Для домена существует общая база данных учетной информации пользователей (user accounts), так что при входе в домен пользователь получает доступ сразу ко всем разрешенным ресурсам всех серверов домена.

Доверительные отношения *(trust relationships)* обеспечивают транзитную аутентификацию, при которой пользователь имеет только одну учетную запись в одном домене, но может получить доступ к ресурсам всех доменов сети.

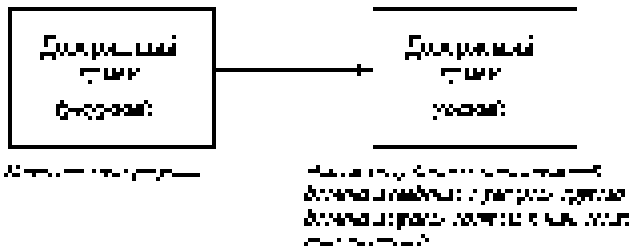


Рис. 3.20. Доверительные отношения между доменами

Пользователи могут входить в сеть не только из рабочих станций того домена, где хранится их учетная информация, но и из рабочих станций доменов, которые доверяют этому домену. Домен, хранящий учетную информацию, часто называют учетным, а доверяющий домен - ресурсным.

Доверительные отношения не являются транзитивными. Например, если домен А доверяет домену В, а В доверяет С, то это не значит, что А автоматически доверяет С.

Основной и резервные контроллеры домена

В домене должен находится сервер, выполняющий роль основного контроллера домена (primary domain controller). Этот контроллер хранит первичную копию базы данных учетной информации пользователей домена. Все изменения,

производимые в учетной информации, сначала производятся именно в этой копии. Основной контроллер домена всегда существует в единственном экземпляре. Пользователь, который администрирует домен, не должен явно задавать имя компьютера, который выполняет роль основного контроллера, утилита, в помощью которой осуществляется администрирование (в Windows NT это User Manager for Domains), должна по имени домена самостоятельно, в соответствии с заранее разработанным протоколом провести диалог с основным контроллером домена и сделать нужные изменения в его базе данных.

Кроме основного контроллера в домене могут существовать несколько резервных контроллеров (backup domain controllers). Эти контроллеры хранят реплики базы учетных данных. Все резервные контроллеры в дополнение к основному могут обрабатывать запросы пользователей на логический вход в домен.

Резервный контроллер домена решает две задачи:

1. Он становится основным контроллером при отказе основного.
2. Уменьшает нагрузку на основной контроллер по обработке логических входов пользователей.

Если сеть состоит из нескольких сетей, соединенных глобальными связями, то в каждой сети должен быть по крайней мере один резервный контроллер домена.

Обычный сервер (не основной или резервный контроллер домена) может быть членом домена, а может и не быть. Если он принимает участие в домене, то он пользуется учетной информацией, хранящейся на контроллере домена. Если же нет - то доступ ко всем его ресурсам имеют только пользователи, которые заведены в базе учетной информации этого сервера.

Четыре модели организации связи доменов

Механизм доменов можно использовать на предприятии различными способами. В зависимости от специфики предприятия можно объединить ресурсы и пользователей в различное количество доменов, а также по-разному установить между ними доверительные отношения.

Microsoft предлагает использовать четыре типовые модели использования доменов на предприятии:

1. Модель с одним доменом;
2. Модель с главным доменом;
3. Модель с несколькими главными доменами;
4. Модель с полными доверительными отношениями.

Модель с одним доменом

Эта модель подходит для организации, в которой имеется не очень много пользователей, и нет необходимости разделять ресурсы сети по организационным подразделениям. Главный ограничитель для этой модели - производительность, которая падает, когда пользователи просматривают домен, включающий много серверов.

Использование только одного домена также означает, что сетевой администратор всегда должен администрировать все серверы. Разделение сети на несколько доменов позволяет назначать администраторов, которые могут администрировать только отдельные серверы, а не всю сеть.

Таблица 3.1.

Преимущества и недостатки модели с одним доменом

Преимущества	Недостатки
Наилучшая модель для предприятий с небольшим числом пользователей и ресурсов Централизованное управление пользовательской учетной информацией Нет нужды в управлении доверительными отношениями Локальные группы нужно определять только однажды	Низкая производительность, если домен имеет слишком много пользователей и/или серверов Невозможность группирования ресурсов

Модель с главным доменом

Эта модель хорошо подходит для предприятий, где необходимо разбить ресурсы на группы в организационных целях, и в то же время количество пользователей и групп пользователей не очень велико. Эта модель сочетает централизацию администрирования с организационными преимуществами разделения ресурсов между несколькими доменами.

Главный домен удобно рассматривать как чисто учетный домен, основное назначение которого - хранение и обработка пользовательских учетных данных. Остальные домены в сети - это домены ресурсов, они не хранят и не обрабатывают пользовательскую учетную информацию, а поставляют ресурсы (такие как разделяемые файлы и принтеры) для сети. В этой модели пользовательскую учетную информацию хранят только основной и резервный контроллеры главного домена.

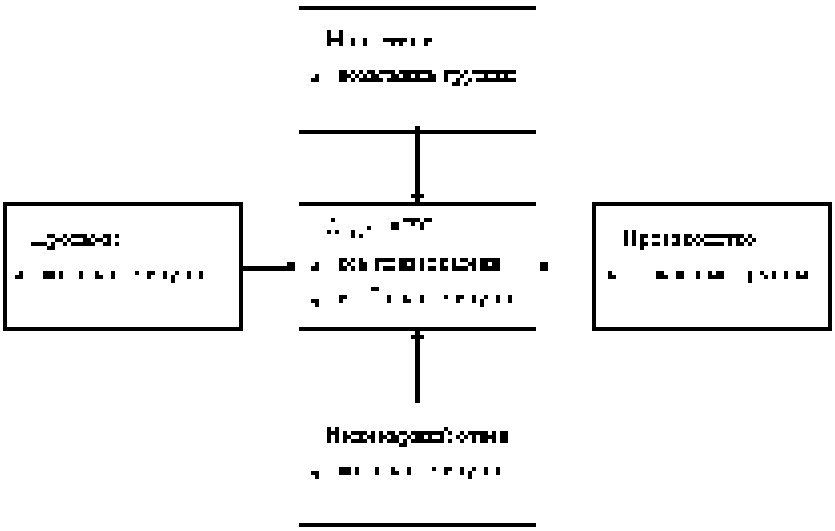


Рис. 3.21. Модель с главным доменом

Таблица 3.2.

Преимущества и недостатки модели с главным доменом

Преимущества	Недостатки
Наилучшая модель для предприятия, у которого не очень много пользователей, а разделяемые ресурсы должны быть распределены по группам Учетная информация может централизованно управляться Ресурсы логически группируются Домены отделов могут иметь своих администраторов, которые управляют ресурсами отдела Глобальные группы должны определяться только один раз (в главном домене)	Плохая производительность, если в главном домене слишком много пользователей и групп Локальные группы нужно образовывать в каждом домене, где они используются

Модель с несколькими главными доменами

Эта модель предназначена для больших предприятий, которые хотят поддерживать централизованное администрирование. Эта модель в наибольшей степени масштабируема.

В данной модели имеется небольшое число главных доменов. Главные домены используются как учетные домены, причем учетная информация каждого пользователя создается только в одном из главным доменов. Сотрудники отдела Автоматизированных Информационных Систем (АИС) предприятия могут администрировать все главные домены, в то время как ресурсные домены могут администрировать сотрудники соответствующих отделов.

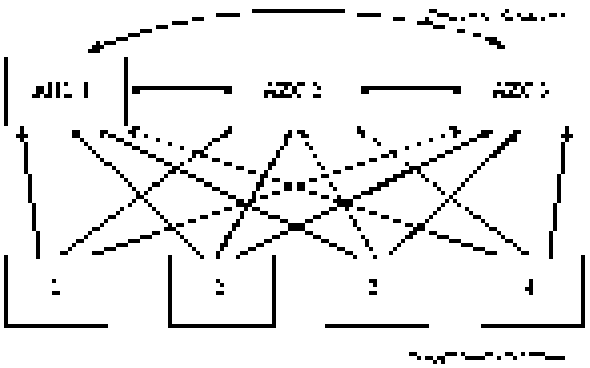


Рис. 3.22. Модель с несколькими главными доменами

Каждый главный домен доверяет всем остальным главным доменам. Каждый домен отдела доверяет всем главным доменам, но доменам отделов нет необходимости доверять друг другу.

Так как все ресурсные домены доверяют всем главным, то данные о любом пользователе могут использоваться в любом отделе предприятия.

Использование глобальных групп в этой модели несколько сложнее, чем в предыдущих. Если нужно образовать глобальную группу из пользователей, учетная информация которых хранится в разных главных доменах, то фактически приходится образовывать несколько глобальных групп - по одной в каждом главном домене. В модели с одним главным доменом нужно образовать только одну глобальную группу.

Чтобы упростить решение этой проблемы, целесообразно распределять пользователей по главным доменам по организационному принципу, а не по какому-либо иному, например, по алфавитному.

Таблица 3.3.

Преимущества и недостатки модели с несколькими главными доменами

Преимущества	Недостатки
Наилучшая модель для предприятия с большим числом пользователей, и центральным отделом АИС. Хорошо масштабируется. Ресурсы логически группируются. Домены отделов могут иметь своих администраторов, которые управляют ресурсами отдела.	Как локальные, так и глобальные группы должны определяться по нескольку раз в каждом учетном домене. Необходимо управлять большим количеством доверительных отношений. В одном домене локализуются не все данные о пользователях.

Модель с полными доверительными отношениями

Эта модель обеспечивает распределенное администрирование пользователей и доменов. В этой модели каждый домен доверяет каждому. Каждый отдел может управлять своим доменом, определяя своих пользователей и глобальные группы пользователей, и учетная информация о них может использоваться во всех доменах предприятия.

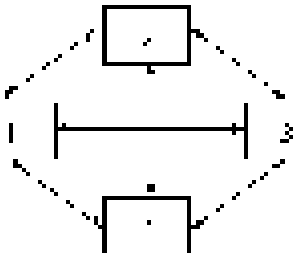


Рис. 3.23. Модель с полными доверительными отношениями

Из-за резкого увеличения числа доверительных отношений эта модель не подходит для больших предприятий. Для *n* доменов нужно установить n(n-1) доверительных отношений.

К этой модели полностью применим термин "доверие". Для создания доверительных отношений с другим доменом администратор действительно должен быть уверен, что он доверяет администратору того домена, особенно если он дает некоторые права глобальным группам другого домена. Как только такие права даны, местный администратор зависит от того, не добавит ли удаленный администратор в глобальную группу нежелательных или непроверенных пользователей в будущем. При администрировании главных доменов такая опасность также имеется. Но риск здесь ниже из-за того, что пользователей в главные домены добавляют сотрудники центрального отдела АИС, а не произвольно назначенный администратором сотрудник функционального отдела предприятия.

Таблица 3.4.

Преимущества и недостатки модели с полными доверительными отношениями

Преимущества	Недостатки
Наилучшим образом подходит для предприятий, на которых нет централизованного отдела АИС Хорошо масштабируется в отношении количества пользователей. Каждый отдел имеет полное управление над своими пользователями и ресурсами. Как ресурсы, так и пользователи группируются по отделам.	Модель не подходит для предприятий с централизованным отделом АИС. Нужно управлять очень большим количеством доверительных отношений. Каждый отдел должен довериться администраторам других отделов, что те не включат в состав своих глобальных групп нежелательных пользователей.

Служба NDS ОС NetWare 4.x - это глобальная служба справочников, использующая распределенную объектно-ориентированную базу данных сетевых ресурсов. База данных NDS содержит информацию о всех сетевых ресурсах, включая информацию о пользователях, группах пользователей, принтерах, томах и компьютерах. NetWare использует эту информацию для обеспечения доступа к этим ресурсам.

База данных NDS заменяет справочник bindery предыдущих версий NetWare. Справочник bindery - это "плоская" или одноуровневая база данных, разработанная для поддержки одного сервера. В ней также использовалось понятие "объект" для сетевого ресурса, но трактовка этого термина отличалась от общепринятой. Объекты bindery идентифицировались простыми числовыми значениями и имели определенные атрибуты. Однако для этих объектов не определялись явные взаимоотношения наследования классов объектов, поэтому взаимоотношения между объектами bindery устанавливались администратором произвольно, что часто приводило к нарушению целостности данных.

База данных службы NDS представляет собой многоуровневую базу данных, поддерживающую информацию о ресурсах всех серверов сети. Для совместимости с предыдущими версиями NetWare в службе NDS предусмотрен механизм эмуляции базы bindery.

Служба NDS представляет собой значительный шаг вперед по сравнению с предыдущими версиями за счет:

- распределенности;
- реплицируемости;
- прозрачности;
- глобальности.

Распределенность заключается в том, что информация не хранится на одном сервере, а разделена на части, называемые разделами (partitions). NetWare хранит эти разделы на нескольких серверах сети (рисунок 3.24). Это свойство значительно упрощает администрирование и управление большой сетью, так как она представляется администратору единой системой. Кроме этого, обеспечивается более быстрый доступ к базе данных сетевых ресурсов за счет обращения к ближайшему серверу.

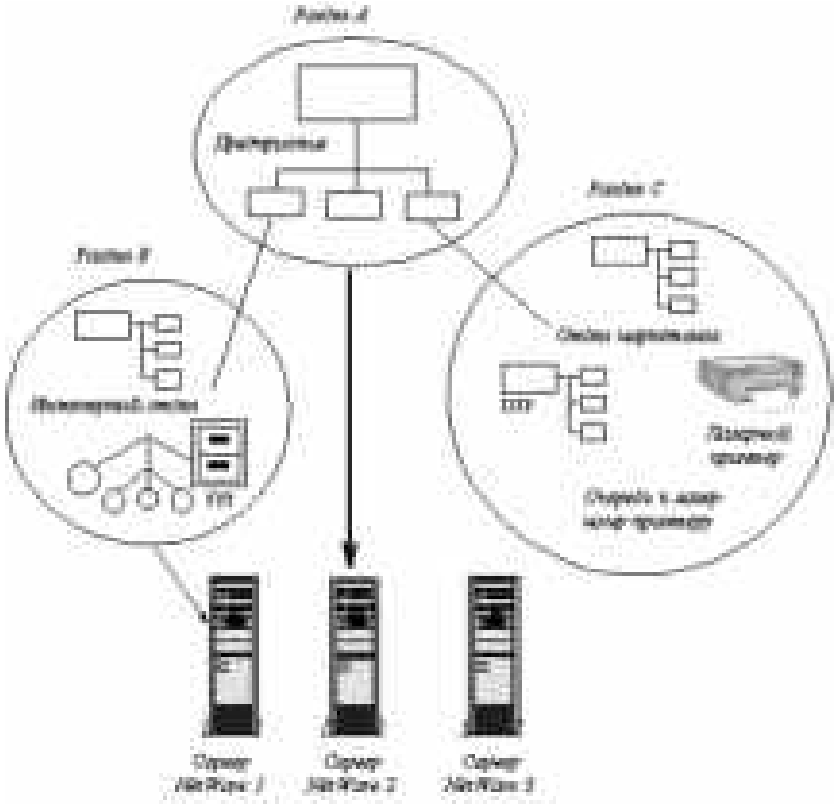


Рис. 3.24. Разделы базы данных NDS

Реплика - это копия информации раздела NDS. Можно создать неограниченное количество реплик каждого раздела и хранить их на разных серверах. Если один сервер останавливается, то копии этой информации могут быть получены с другого сервера. Это увеличивает отказоустойчивость системы, так как ни один из серверов не отвечает за всю информацию базы данных NDS.

Прозрачность заключается в том, что NDS автоматически создает связи между программными и аппаратными компонентами, которые обеспечивают пользователю доступ к сетевым ресурсам. NDS при этом не требует от пользователя знаний физического расположения этих ресурсов. Задавая сетевой ресурс по имени, вы получите к нему корректный доступ даже в случае изменения его сетевого адреса или места расположения.

Глобальность NDS заключается в том, что после входа вы получаете доступ к ресурсам всей сети, а не только одного сервера, как было в предыдущих версиях. Это достигается за счет процедуры глобального логического входа (global login). Вместо входа в отдельный сервер пользователь NDS входит в сеть. После чего он получает доступ к разрешенным для него ресурсам сети. Информация, предоставляемая во время логического входа, используется для процесса идентификации пользователя. Позже, при попытке пользователя получить доступ к ресурсам, таким как серверы, тома или принтеры, фоновый процесс идентификации проверяет, имеет ли пользователь право использовать данный ресурс.

Объектно-ориентированный подход

В NDS информация о сетевых ресурсах организована с помощью объектов. Каждый объект представляет собой ресурс, такой как принтер, том, пользователь или сервер.

Объекты организованы в иерархическую структуру, соответствующую структуре организации, отражающую реальные информационные потоки и потребности разделения ресурсов. Одни объекты представляют физические сущности, например объект-пользователь представляет пользователя, а объект-принтер представляет принтер. Другие объекты представляют логические понятия, такие как группы или очереди к принтерам.

Еще одна категория объектов, в которую входят, например, объекты типа отдела предприятия, помогают организовывать и упорядочивать другие объекты.

Объекты имеют атрибуты, в которых хранится индивидуальная для данного объекта информация, например, имя и телефон пользователя или место расположения факса и т.п.

Дерево каталогов

NDS использует для хранения информации логическую структуру, называемую деревом каталогов (Directory Tree, DT). Эта иерархическая структура имеет корневой элемент (root) и ветви (рисунок 3.25). Администратору сети NetWare 4.x предоставляется удобная графическая утилита NetWare Administrator, работающая в среде Windows, наглядно представляющая каждый объект дерева каталогов NDS в виде иконки и отражающая связи между объектами. Пользователи также получают удобства прозрачного доступа к ресурсам всей сети, если они пользуются оболочкой NetWare для Windows, которая поддерживает диалог с NDS и представляет доступные пользователю ресурсы в виде вложенных пиктограмм.

Дерево каталогов содержит объекты двух типов:

- объекты-контейнеры,
- объекты листья.

Объекты-контейнеры содержат или включают другие объекты. Они используются для логического упорядочивания и организации других объектов дерева.

NDS содержит объекты-контейнеры трех типов, которые можно использовать для организации дерева объектов:

- Страна (Country) - необязательный объект, который можно использовать в сетях, охватывающих несколько стран.

- Организация (Organization) - располагается уровнем ниже, чем объекты-страны (если они используются).
- Отдел или подразделение организации (Organizational Unit) - располагается уровнем ниже, чем объекты-организации. Помогает в дальнейшем упорядочивании остальных объектов.

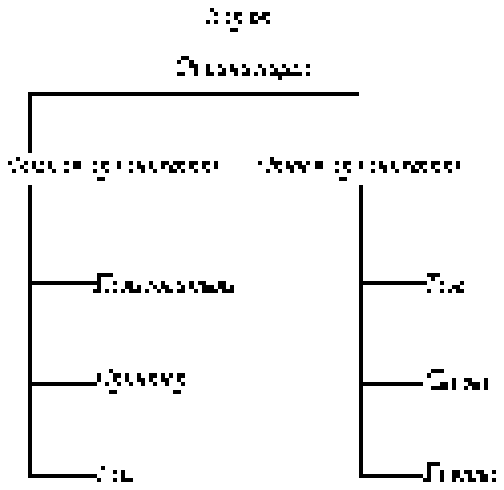


Рис. 3.25. Типичная структура дерева каталогов NDS

Объекты-листья не содержат другие объекты, они используются для представления конечных сетевых ресурсов, таких как компьютеры, тома, принтеры, пользователи и группы пользователей. В NDS определены еще и такие типы объектов-листьев, как:

- *организатор* - используется для таких руководящих лиц фирмы, как руководитель группы или вице-президент,
- *сервер*,
- *принт-сервер*,
- *очередь печати*,
- *карта каталогов на томе*,
- *профиль* - представляет командный файл входа Login Script для специальной группы пользователей, которые хотят разделять общие команды Login Script, но не обязательно входят в один объект-контейнер в дереве каталогов, или же для подмножества пользователей одного контейнера,
- *псевдоним* - указывает на оригинальное местонахождения объекта в дереве. Любой объект может быть расположенным в нескольких местах дерева с использованием псевдонимов. Псевдонимы делают использование NDS более гибким и удобным.

Служба NDS и файловая система

Служба NDS предназначена для управления такими сетевыми ресурсами, как серверы и тома NetWare, но она не обеспечивает управление файловой системой.

Файлы и каталоги не являются объектами службы NDS. Однако они представляются в виде иконок при использовании графической утилиты NetWare Administrator. Одним из атрибутов объекта-тома является месторасположение физического тома, который содержит файлы и каталоги. Таким образом, объект-том представляет собой связь между NDS и файловой системой.

Служба NDS предоставляет средства для поиска объектов в ее базе данных сетевых ресурсов. Можно делать запросы, типичные для баз данных, например, поиск пользователей, живущих на одной улице и т.п. Можно также сделать запрос о значениях всех атрибутов какого-либо конкретного объекта.

NDS также использует синхронизацию часов между всеми серверами сети для обеспечения правильного порядка событий в сети.

Имена и контексты

В именовании объектов служба NDS использует те же принципы, что и файловые системы MS DOS и UNIX. Объект-лист имеет краткое имя, называемое Common Name (CN). Оно может состоять максимум из 64 символов, включая пробелы. Поэтому имя принтера "Быстрый, но часто ломающийся Epson" является законным. Аналогом полного имени файловой системы MS DOS является так называемое "различимое имя" - Distinguished Name. Различимое имя представляет собой конкатенацию всех имен объектов, расположенных на пути этого объекта к корню дерева. Составляющие различимого имени отделяются друг от друга точкой. В отличие от полных имен MS DOS крайней левой составляющей различимого имени является краткое имя объекта, а крайней правой составляющей - имя корневого объекта. Например,

```
Vova S.NetProgrammers.Microsoft.US
```

представляет собой различимое имя объекта-пользователя с сетевым именем Vova S, работающего в сетевом отделе фирмы Microsoft, расположенной в США. Возможен и другой вариант записи различимого имени с указанием типов объектов:

```
CN=Vova s.Ou=NetProgrammers.O=Microsoft.C=US.
```

Такой вид записи более содержателен.

Средства защиты объектов в NDS

Служба NDS определяет права доступа одних сетевых объектов к другим. Различаются права доступа к объекту в целом и права доступа к его атрибутам.

По отношению к объектам существует следующий набор прав:

- *Browse* - просмотр;

- *Add* - добавление;
- *Delete* - удаление;
- *Rename* - переименование;
- *Supervisor* - обеспечивает все перечисленные выше права.

По отношению к атрибутам объектов используются такие права:

- *Compare* - сравнение значения атрибута;
- *Read* - чтение значения атрибута;
- *Write* - запись нового значения атрибута;
- *Self* - присвоение себя в качестве значения атрибута другого объекта, например, если объект-группа разрешает право Self для объекта User, то последний может сделать себя членом этой группы;
- *Supervisor* - все права по доступу к атрибутам.

С каждым объектом связан список управления доступом (ACL), в котором определяются права доступа к данному объекту со стороны других объектов.

Права доступа наследуются в дереве объектов сверху вниз, поэтому права объекта-контейнера наследуются входящими в него объектами. Для достижении необходимой гибкости в наделении объекта правами используется маска наследования (Inheritance Mask), с помощью которой можно заблокировать некоторые наследуемые права. С помощью наследования прав доступа главный администратор дерева, имеющий доступ ко всем его объектам, может назначить администратора поддерева, который получит права доступа ко всем объектам данного поддерева. Если поддерево соответствует какой-либо структурной единице предприятия (что и подразумевается), например отделу, то это будет администратор отдела, управляющий пользователями и ресурсами данного отдела.

После подробного рассмотрения свойств дерева каталогов NDS можно уточнить понятия раздела (partition) и реплики (replica). Раздел представляет собой поддерево общего дерева сети. Для определения раздела необходимо выбрать объект-контейнер в общем дереве, который будет корневым объектом данного раздела. Создание раздела уменьшает объем хранимой на сервере информации базы данных NDS за счет исключения редко используемой информации и делает доступ к локальным объектам более быстрым, хотя объекты, находящиеся в других разделах, также доступны всем клиентам сети.

Реплика - это точная копия определенного раздела, хранящаяся на различных серверах. Наличие нескольких реплик обеспечивает отказоустойчивость ОС NetWare 4.x, а также ускоряет доступ к информации при перенесении реплики с подключенного через глобальную сеть сервера на локальный сервер.

Существуют три типа реплик: главная реплика (master replica), вторичная реплика (read/write replica) и реплика только для чтения (read-only replica).

Главная реплика позволяет проводить над ней такие операции, как создание нового раздела, слияние разделов и удаление раздела.

Вторичная реплика разрешает обновлять информацию об объектах, добавлять новые объекты, но не разрешает создавать новые разделы.

Реплика только для чтения позволяет только читать информацию из ее базы и проводить операции поиска.

В сети может существовать произвольное количество реплик. При изменения информации в какой-либо реплике автоматически запускается процесс обновления всех остальных реплик. Этот процесс называется процессом синхронизации службы каталогов (DSS).

Любой сервер NetWare, поддерживающий службу NDS, называется сервером имен.

Современные концепции и технологии проектирования операционных систем

Требования, предъявляемые к ОС 90-х годов

Операционная система является сердцевиной сетевого программного обеспечения, она создает среду для выполнения приложений и во многом определяет, какими полезными для пользователя свойствами эти приложения будут обладать. В связи с этим рассмотрим требования, которым должна удовлетворять современная ОС.

Очевидно, что главным требованием, предъявляемым к операционной системе, 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, виртуальную память, свопинг, поддерживать многооконный интерфейс, а также выполнять многие другие, совершенно необходимые функции. Кроме этих функциональных требований к операционным системам предъявляются не менее важные рыночные требования. К этим требованиям относятся:

- *Расширяемость*. Код должен быть написан таким образом, чтобы можно было легко внести дополнения и изменения, если это потребуется, и не нарушить целостность системы.
- *Переносимость*. Код должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которая включает наряду с типом процессора и способ организации всей аппаратуры компьютера) одного типа на аппаратную платформу другого типа.

- *Надежность и отказоустойчивость*. Система должна быть защищена как от внутренних, так и от внешних ошибок, сбоев и отказов. Ее действия должны быть всегда предсказуемыми, а приложения не должны быть в состоянии наносить вред ОС.
- *Совместимость*. ОС должна иметь средства для выполнения прикладных программ, написанных для других операционных систем. Кроме того, пользовательский интерфейс должен быть совместим с существующими системами и стандартами.
- *Безопасность*. ОС должна обладать средствами защиты ресурсов одних пользователей от других.
- *Производительность*. Система должна обладать настолько хорошим быстродействием и временем реакции, насколько это позволяет аппаратная платформа.

Рассмотрим более подробно некоторые из этих требований.

Расширяемость

В то время, как аппаратная часть компьютера устаревает за несколько лет, полезная жизнь операционных систем может измеряться десятилетиями. Примером может служить ОС UNIX. Поэтому операционные системы всегда эволюционно изменяются со временем, и эти изменения более значимы, чем изменения аппаратных средств. Изменения ОС обычно представляют собой приобретение ею новых свойств. Например, поддержка новых устройств, таких как CD-ROM, возможность связи с сетями нового типа, поддержка многообещающих технологий, таких как графический интерфейс пользователя или объектно-ориентированное программное окружение, использование более чем одного процессора. Сохранение целостности кода, какие бы изменения не вносились в операционную систему, является главной целью разработки.

Расширяемость может достигаться за счет модульной структуры ОС, при которой программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс. Новые компоненты могут быть добавлены в операционную систему модульным путем, они выполняют свою работу, используя интерфейсы, поддерживаемые существующими компонентами.

Использование объектов для представления системных ресурсов также улучшает расширяемость системы. Объекты - это абстрактные типы данных, над которыми можно производить только те действия, которые предусмотрены специальным набором объектных функций. Объекты позволяют единообразно управлять системными ресурсами. Добавление новых объектов не разрушает существующие объекты и не требует изменений существующего кода.

Прекрасные возможности для расширения предоставляет подход к структурированию ОС по типу клиент-сервер с использованием микроядерной технологии. В соответствии с этим подходом ОС строится как совокупность

привилегированной управляющей программы и набора непривилегированных услуг-серверов. Основная часть ОС может оставаться неизменной в то время, как могут быть добавлены новые серверы или улучшены старые.

Средства вызова удаленных процедур (RPC) также дают возможность расширить функциональные возможности ОС. Новые программные процедуры могут быть добавлены в любую машину сети и немедленно поступить в распоряжение прикладных программ на других машинах сети.

Некоторые ОС для улучшения расширяемости поддерживают загружаемые драйверы, которые могут быть добавлены в систему во время ее работы. Новые файловые системы, устройства и сети могут поддерживаться путем написания драйвера устройства, драйвера файловой системы или транспортного драйвера и загрузки его в систему.

Переносимость

Требование переносимости кода тесно связано с расширяемостью. Расширяемость позволяет улучшать операционную систему, в то время как переносимость дает возможность перемещать всю систему на машину, базирующуюся на другом процессоре или аппаратной платформе, делая при этом по возможности небольшие изменения в коде. Хотя ОС часто описываются либо как переносимые, либо как непереносимые, переносимость - это не бинарное состояние. Вопрос не в том, может ли быть система перенесена, а в том, насколько легко можно это сделать. Написание переносимой ОС аналогично написанию любого переносимого кода - нужно следовать некоторым правилам.

Во-первых, большая часть кода должна быть написана на языке, который имеется на всех машинах, куда вы хотите переносить систему. Обычно это означает, что код должен быть написан на языке высокого уровня, предпочтительно стандартизованном, например, на языке С. Программа, написанная на ассемблере, не является переносимой, если только вы не собираетесь переносить ее на машину, обладающую командной совместимостью с вашей.

Во-вторых, следует учесть, в какое физическое окружение программа должна быть перенесена. Различная аппаратура требует различных решений при создании ОС. Например, ОС, построенная на 32-битовых адресах, не может быть перенесена на машину с 16-битовыми адресами (разве что с огромными трудностями).

В-третьих, важно минимизировать или, если возможно, исключить те части кода, которые непосредственно взаимодействуют с аппаратными средствами. Зависимость от аппаратуры может иметь много форм. Некоторые очевидные формы зависимости включают прямое манипулирование регистрами и другими аппаратными средствами.

В-четвертых, если аппаратно зависимый код не может быть полностью исключен, то он должен быть изолирован в нескольких хорошо локализуемых модулях. Аппаратно-зависимый код не должен быть распределен по всей системе. Например, можно спрятать аппаратно-зависимую структуру в программно-задаваемые данные абстрактного типа. Другие модули системы будут работать с этими данными, а не с аппаратурой, используя набор некоторых функций. Когда ОС переносится, то изменяются только эти данные и функции, которые ими манипулируют.

Для легкого переноса ОС при ее разработке должны быть соблюдены следующие требования:

- *Переносимый язык высокого уровня*. Большинство переносимых ОС написано на языке С (стандарт ANSI X3.159-1989). Разработчики выбирают С потому, что он стандартизован, и потому, что С-компиляторы широко доступны. Ассемблер используется только для тех частей системы, которые должны непосредственно взаимодействовать с аппаратурой (например, обработчик прерываний) или для частей, которые требуют максимальной скорости (например, целочисленная арифметика повышенной точности). Однако непереносимый код должен быть тщательно изолирован внутри тех компонентов, где он используется.
- *Изоляция процессора*. Некоторые низкоуровневые части ОС должны иметь доступ к процессорно-зависимым структурам данных и регистрам. Однако код, который делает это, должен содержаться в небольших модулях, которые могут быть заменены аналогичными модулями для других процессоров.
- *Изоляция платформы*. Зависимость от платформы заключается в различиях между рабочими станциями разных производителей, построенными на одном и том же процессоре (например, MIPS R4000). Должен быть введен программный уровень, абстрагирующий аппаратуру (кэши, контроллеры прерываний ввода-вывода и т. п.) вместе со слоем низкоуровневых программ таким образом, чтобы высокоуровневый код не нуждался в изменении при переносе с одной платформы на другую.

Совместимость

Одним из аспектов совместимости является способность ОС выполнять программы, написанные для других ОС или для более ранних версий данной операционной системы, а также для другой аппаратной платформы.

Необходимо разделять вопросы двоичной совместимости и совместимости на уровне исходных текстов приложений. Двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее на выполнение на другой ОС. Для этого необходимы: совместимость на уровне

команд процессора, совместимость на уровне системных вызовов и даже на уровне библиотечных вызовов, если они являются динамически связываемыми.

Совместимость на уровне исходных текстов требует наличия соответствующего компилятора в составе программного обеспечения, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция имеющихся исходных текстов в новый выполняемый модуль.

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

Обладает ли новая ОС двоичной совместимостью или совместимостью исходных текстов с существующими системами, зависит от многих факторов. Самый главный из них - архитектура процессора, на котором работает новая ОС. Если процессор, на который переносится ОС, использует тот же набор команд (возможно с некоторыми добавлениями) и тот же диапазон адресов, тогда двоичная совместимость может быть достигнута достаточно просто.

Гораздо сложнее достичь двоичной совместимости между процессорами, основанными на разных архитектурах. Для того, чтобы один компьютер выполнял программы другого (например, DOS-программу на Mac), этот компьютер должен работать с машинными командами, которые ему изначально непонятны. Например, процессор типа 680x0 на Mac должен исполнять двоичный код, предназначенный для процессора 80x86 в PC. Процессор 80x86 имеет свои собственные дешифратор команд, регистры и внутреннюю архитектуру. Процессор 680x0 не понимает двоичный код 80x86, поэтому он должен выбрать каждую команду, декодировать ее, чтобы определить, для чего она предназначена, а затем выполнить эквивалентную подпрограмму, написанную для 680x0. Так как к тому же у 680x0 нет в точности таких же регистров, флагов и внутреннего арифметико-логического устройства, как в 80x86, он должен имитировать все эти элементы с использованием своих регистров или памяти. И он должен тщательно воспроизводить результаты каждой команды, что требует специально написанных подпрограмм для 680x0, гарантирующих, что состояние эмулируемых регистров и флагов после выполнения каждой команды будет в точности таким же, как и на реальном 80x86.

Это простая, но очень медленная работа, так как микрокод внутри процессора 80x86 исполняется на значительно более быстродействующем уровне, чем эмулирующие его внешние команды 680x0. За время выполнения одной команды 80x86 на 680x0, реальный 80x86 может выполнить десятки команд. Следовательно, если процессор, производящий эмуляцию, не настолько быстр, чтобы компенсировать все потери при эмуляции, то программы, исполняющиеся под эмуляцией, будут очень медленными.

Выходом в таких случаях является использование так называемых прикладных сред. Учитывая, что основную часть программы, как правило, составляют вызовы библиотечных функций, прикладная среда имитирует библиотечные функции целиком, используя заранее написанную библиотеку функций аналогичного назначения, а остальные команды эмулирует каждую по отдельности.

Соответствие стандартам POSIX также является средством обеспечения совместимости программных и пользовательских интерфейсов. Во второй половине 80-х правительственные агентства США начали разрабатывать POSIX как стандарты на поставляемое оборудование при заключении правительственных контрактов в компьютерной области. POSIX - это "интерфейс переносимой ОС, базирующейся на UNIX". POSIX - собрание международных стандартов интерфейсов ОС в стиле UNIX. Использование стандарта POSIX (IEEE стандарт 1003.1 - 1988) позволяет создавать программы стиле UNIX, которые могут легко переноситься из одной системы в другую.

Безопасность

В дополнение к стандарту POSIX правительство США также определило требования компьютерной безопасности для приложений, используемых правительством. Многие из этих требований являются желаемыми свойствами для любой многопользовательской системы. Правила безопасности определяют такие свойства, как защита ресурсов одного пользователя от других и установление квот по ресурсам для предотвращения захвата одним пользователем всех системных ресурсов (таких как память).

Обеспечение защиты информации от несанкционированного доступа является обязательной функцией сетевых операционных систем. В большинстве популярных систем гарантируется степень безопасности данных, соответствующая уровню С2 в системе стандартов США.

Основы стандартов в области безопасности были заложены "*Критериями оценки надежных компьютерных систем*". Этот документ, изданный в США в 1983 году национальным центром компьютерной безопасности (NCSC - National Computer Security Center), часто называют Оранжевой Книгой.

В соответствии с требованиями Оранжевой книги безопасной считается такая система, которая "посредством специальных механизмов защиты контролирует доступ к информации таким образом, что только имеющие соответствующие полномочия лица или процессы, выполняющиеся от их имени, могут получить доступ на чтение, запись, создание или удаление информации".

Иерархия уровней безопасности, приведенная в Оранжевой Книге, помечает низший уровень безопасности как D, а высший - как А.

- В класс D попадают системы, оценка которых выявила их несоответствие требованиям всех других классов.

- Основными свойствами, характерными для С-систем, являются: наличие подсистемы учета событий, связанных с безопасностью, и избирательный контроль доступа. Уровень С делится на 2 подуровня: уровень С1, обеспечивающий защиту данных от ошибок пользователей, но не от действий злоумышленников, и более строгий уровень С2. На уровне С2 должны присутствовать*средства секретного входа,* обеспечивающие идентификацию пользователей путем ввода уникального имени и пароля перед тем, как им будет разрешен доступ к системе. *Избирательный контроль доступа*, требуемый на этом уровне позволяет владельцу ресурса определить, кто имеет доступ к ресурсу и что он может с ним делать. Владелец делает это путем предоставляемых прав доступа пользователю или группе пользователей. *Средства учета и наблюдения* (*auditing*) - обеспечивают возможность обнаружить и зафиксировать важные события, связанные с безопасностью, или любые попытки создать, получить доступ или удалить системные ресурсы.*Защита памяти* - заключается в том, что память инициализируется перед тем, как повторно используется. На этом уровне система не защищена от ошибок пользователя, но поведение его может быть проконтролировано по записям в журнале, оставленным средствами наблюдения и аудитинга.
- Системы уровня В основаны на помеченных данных и распределении пользователей по категориям, то есть реализуют *мандатный контроль доступа*. Каждому пользователю присваивается рейтинг защиты, и он может получать доступ к данным только в соответствии с этим рейтингом. Этот уровень в отличие от уровня С защищает систему от ошибочного поведения пользователя.
- Уровень А является самым высоким уровнем безопасности, он требует в дополнение ко всем требованиям уровня В выполнения формального, математически обоснованного доказательства соответствия системы требованиям безопасности.

Различные коммерческие структуры (например, банки) особо выделяют необходимость учетной службы, аналогичной той, что предлагают государственные рекомендации С2. Любая деятельность, связанная с безопасностью, может быть отслежена и тем самым учтена. Это как раз то, что требует С2 и то, что обычно нужно банкам. Однако, коммерческие пользователи, как правило, не хотят расплачиваться производительностью за повышенный уровень безопасности. А-уровень безопасности занимает своими управляющими механизмами до 90% процессорного времени. Более безопасные системы не только снижают эффективность, но и существенно ограничивают число доступных прикладных пакетов, которые соответствующим образом могут выполняться в подобной системе. Например для ОС Solaris (версия UNIX) есть несколько тысяч приложений, а для ее аналога В-уровня - только сотня.

Тенденции в структурном построении ОС

Как уже отмечалось выше, для удовлетворения требований, предъявляемых к современной ОС, большое значение имеет ее структурное построение. Операционные системы прошли длительный путь развития от монолитных систем к хорошо структурированным модульным системам, способным к развитию, расширению и легкому переносу на новые платформы.

Монолитные системы

В общем случае "структура" монолитной системы представляет собой отсутствие структуры (рисунок 4.1). ОС написана как набор процедур, каждая из которых может вызывать другие, когда ей это нужно. При использовании этой техники каждая процедура системы имеет хорошо определенный интерфейс в терминах параметров и результатов, и каждая вольна вызвать любую другую для выполнения некоторой нужной для нее полезной работы.

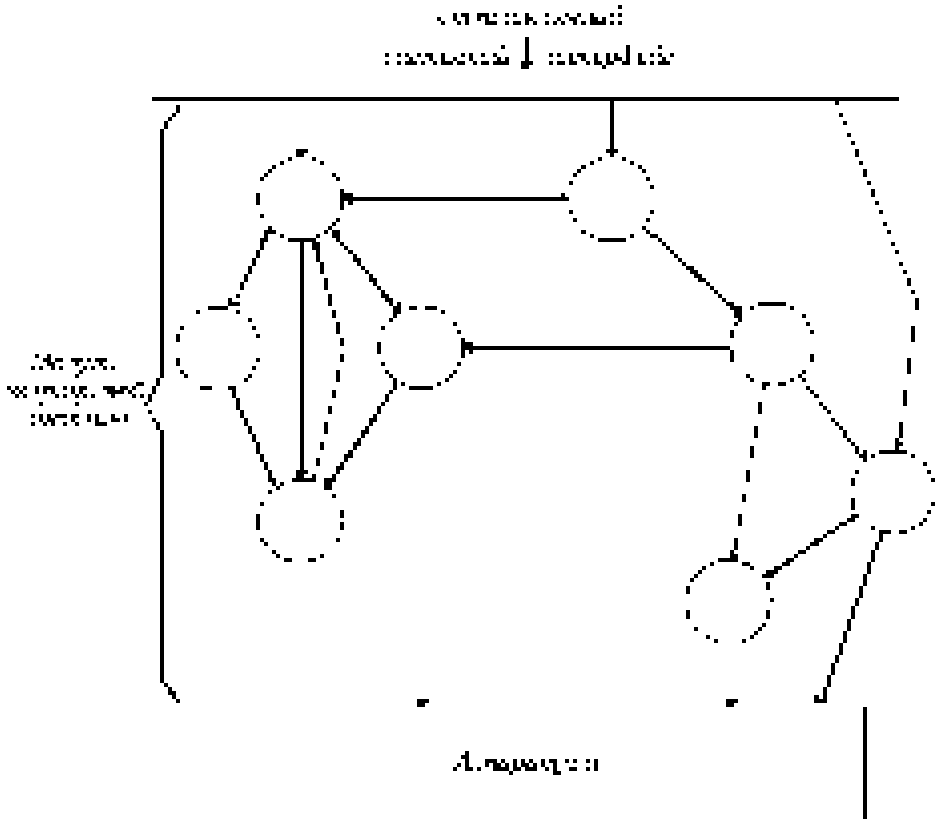


Рис. 4.1. Монолитная структура ОС

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 ядра UNIX или Novell NetWare). Каждая процедура видит любую другую процедуру (в отличие от структуры, содержащей модули, в которой большая часть информации является локальной для модуля, и процедуры модуля можно вызвать только через специально определенные точки входа).

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

1. Главная программа, которая вызывает требуемые сервисные процедуры.

2. Набор сервисных процедур, реализующих системные вызовы.

3. Набор утилит, обслуживающих сервисные процедуры.

В этой модели для каждого системного вызова имеется одна сервисная процедура. Утилиты выполняют функции, которые нужны нескольким сервисным процедурам. Это деление процедур на три слоя показано на рисунке 4.2.

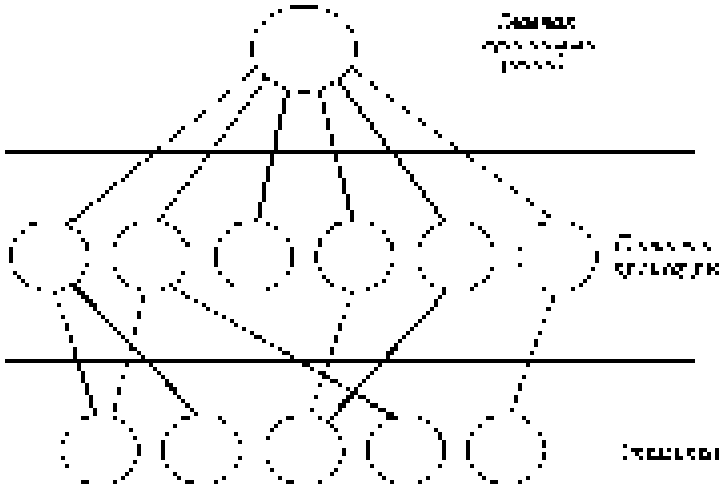


Рис. 4.2. Простая структуризация монолитной ОС

Многоуровневые системы

Обобщением предыдущего подхода является организация ОС как иерархии уровней. Уровни образуются группами функций операционной системы - файловая система, управление процессами и устройствами и т.п. Каждый уровень может взаимодействовать только со своим непосредственным соседом - выше- или нижележащим уровнем. Прикладные программы или модули самой операционной системы передают запросы вверх и вниз по этим уровням.

Первой системой, построенной таким образом была простая пакетная система THE, которую построил Дейкстра и его студенты в 1968 году.

Система имела 6 уровней. Уровень 0 занимался распределением времени процессора, переключая процессы по прерыванию или по истечении времени. Уровень 1 управлял памятью - распределял оперативную память и пространство на магнитном барабане для тех частей процессов (страниц), для которых не было места в ОП, то есть слой 1 выполнял функции виртуальной памяти. Слой 2 управлял связью между консолью оператора и процессами. С помощью этого уровня каждый процесс имел свою собственную консоль оператора. Уровень 3 управлял устройствами ввода-вывода и буферизовал потоки информации к ним и от них. С помощью уровня 3 каждый процесс вместо того, чтобы работать с конкретными устройствами, с их разнообразными особенностями, обращался к абстрактным устройствам ввода-вывода, обладающим удобными для пользователя характеристиками. На уровне 4 работали пользовательские программы, которым не надо было заботиться ни о процессах, ни о памяти, ни о консоли, ни об управлении устройствами ввода-вывода. Процесс системного оператора размещался на уровне 5.

В системе THE многоуровневая схема служила, в основном, целям разработки, так как все части системы компоновались затем в общий объектный модуль.

Дальнейшее обобщение многоуровневой концепции было сделано в ОС MULTICS. В системе MULTICS каждый уровень (называемый кольцом) является более привилегированным, чем вышележащий. Когда процедура верхнего уровня хочет вызвать процедуру нижележащего, она должна выполнить соответствующий системный вызов, то есть команду TRAP (прерывание), параметры которой тщательно проверяются перед тем, как выполняется вызов. Хотя ОС в MULTICS является частью адресного пространства каждого пользовательского процесса, аппаратура обеспечивает защиту данных на уровне сегментов памяти, разрешая, например, доступ к одним сегментам только для записи, а к другим - для чтения или выполнения. Преимущество подхода MULTICS заключается в том, что он может быть расширен и на структуру пользовательских подсистем. Например, профессор может написать программу для тестирования и оценки студенческих программ и запустить эту программу на уровне n, в то время как студенческие программы будут работать на уровне n+1, так что они не смогут изменить свои оценки.

Многоуровневый подход был также использован при реализации различных вариантов ОС UNIX.

Хотя такой структурный подход на практике обычно работал неплохо, сегодня он все больше воспринимается монолитным. В системах, имеющих многоуровневую структуру было нелегко удалить один слой и заменить его другим в силу множественности и размытости интерфейсов между слоями. Добавление новых функций и изменение существующих требовало хорошего знания операционной системы и массы времени. Когда стало ясно, что

операционные системы живут долго и должны иметь возможности развития и расширения, монолитный подход стал давать трещину, и на смену ему пришла модель клиент-сервер и тесно связанная с ней концепция микроядра.

Модель клиент-сервер и микроядра

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 понятным 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 ясного представления функций того или иного программного элемента в той или иной ситуации, нежели технологией. Эта модель успешно применяется не только при построении ОС, но и на всех уровнях программного обеспечения, и имеет в некоторых случаях более узкий, специфический смысл, сохраняя, естественно, при этом все свои общие черты.

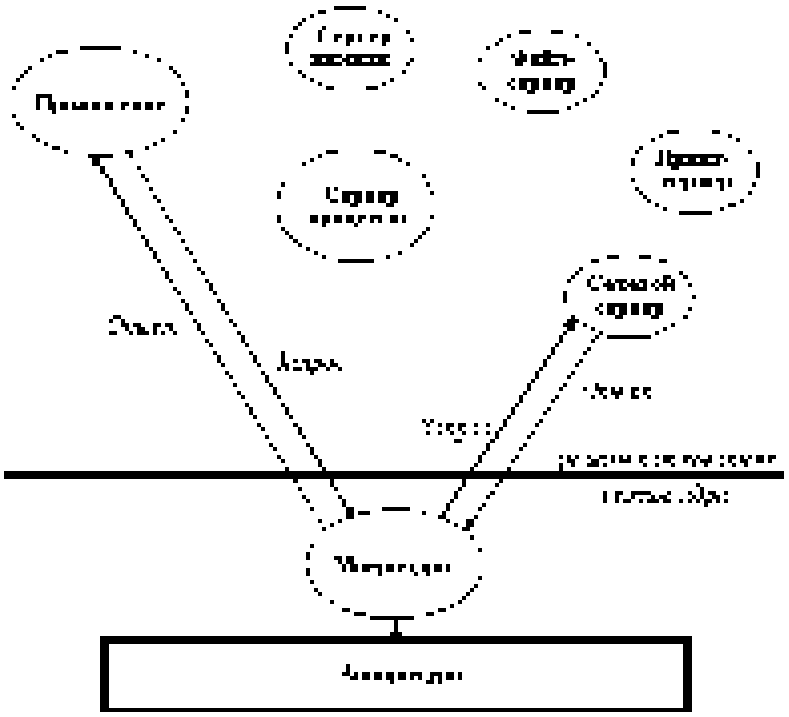


Рис. 4.3. Структура ОС клиент-сервер

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 (называемое здесь микроядром), работая в привилегированном режиме, доставляет сообщение нужному серверу, сервер выполняет операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения (рисунок 4.3).

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

Данная теоретическая модель является идеализированным описанием системы клиент-сервер, в которой ядро состоит только из средств передачи сообщений. В действительности различные варианты реализации модели клиент-сервер в структуре ОС могут существенно различаться по объему работ, выполняемых в режиме ядра.

На одном краю этого спектра находится разрабатываемая фирмой IBM на основе микроядра Mach операционная система Workplace OS, придерживающаяся чистой микроядерной доктрины, состоящей в том, что все несущественные функции ОС должны выполняться не в режиме ядра, а в непривилегированном (пользовательском) режиме. На другом - Windows NT, в составе которой имеется исполняющая система (NT executive), работающая в режиме ядра и выполняющая функции обеспечения безопасности, ввода-вывода и другие.

Микроядро реализует жизненно важные функции, лежащие в основе операционной системы. Это базис для менее существенных системных служб и приложений. Именно вопрос о том, какие из системных функций считать несущественными, и, соответственно, не включать их в состав ядра, является предметом спора среди соперничающих сторонников идеи микроядра. В общем случае, подсистемы, бывшие традиционно неотъемлемыми частями операционной системы - файловые системы, 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 и в привилегированном пространстве. Под этим обычно подразумеваются машиннозависимые 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. Эти функции операционной системы трудно, если не невозможно, выполнить программам, работающим в пространстве пользователя.

Есть два пути решения этой проблемы. Один путь - разместить несколько таких, чувствительных к режиму работы процессора, серверов, в пространстве ядра, что обеспечит им полный доступ к аппаратуре и , в то же время, связь с другими процессами с помощью обычного механизма сообщений. Такой подход был использован, например, при разработке Windows NT: кроме микроядра, в привилегированном режиме работает часть Windows NT, называемая executive управляющей программой. Она включает ряд компонентов, которые управляют виртуальной памятью, объектами, вводом-выводом и файловой системой (включая сетевые драйверы), взаимодействием процессов, и частично системой безопасности.

Другой путь, заключается в том, чтобы оставить в ядре только небольшую часть сервера, представляющую собой механизм реализации решения, а часть, отвечающую за принятие решения, переместить в пользовательскую область. В соответствии с этим подходом, например, в микроядре Mach, на базе которого разработана Workplace OS, размещается только часть системы управления процессами (и нитями), реализующая диспетчеризацию (то есть непосредственно переключение с процесса на процесс), а все функции, связанные с анализом приоритетов, выбором очередного процесса для активизации, принятием решения о переключении на новый процесс и другие аналогичные функции выполняются вне микроядра. Этот подход требует тесного взаимодействия между внешним планировщиком и резидентным диспетчером.

Здесь важно сделать различие - запуск процесса или нити требует доступа к аппаратуре, так что по логике - это функция ядра. Но ядру все равно, какую из нитей запускать, поэтому решения о приоритетах нитей и дисциплине постановки в очередь может принимать работающий вне ядра планировщик.

Кроме уже представленных соображений, перемещение планировщика на пользовательский уровень может понадобиться для чисто коммерческих целей. Некоторые производители ОС (например, IBM и OSF со своими вариантами микроядра Mach) планируют лицензировать свое микроядро другим поставщикам, которым может потребоваться заменить исходный планировщик на другой, поддерживающий, например, планирование в задачах реального времени или реализующий какой-то специальный алгоритм планирования. А вот другая ОС - Windows NT, также использующая микроядерную концепцию - воплотила понятие приоритетов реального времени в своем планировщике, резидентно расположенном в ядре, и это не дает возможности заменить ее планировщик на другой.

Как и управление процессами, управление памятью может распределяться между микроядром и сервером, работающим в пользовательском режиме. Например, в Workplace OS микроядро управляет аппаратурой страничной памяти. Пейджер

(система управления страничной памятью), работающий вне ядра, определяет стратегию замещения страниц (т.е. решает, какие страницы следует удалить из памяти для размещения страниц, выбранных с диска в ответ на прерывание по отсутствию необходимой страницы), а микроядро выполняет перемещение выбранных пейджером страниц. Как и планировщик процессов, пейджер является заменяемой составной частью.

Драйверы устройств также могут располагаться как внутри ядра, так и вне его. При размещении драйверов устройств вне микроядра для обеспечения возможности разрешения и запрещения прерываний, часть программы драйвера должна исполняться в пространстве ядра. Отделение драйверов устройств от ядра делает возможной динамическую конфигурацию ОС. Кроме динамической конфигурации, есть и другие причины рассматривать драйверы устройств в качестве процессов пользовательского режима. СУБД, например, может иметь свой драйвер, оптимизированный под конкретный вид доступа к диску, но его нельзя будет подключить, если драйверы будут расположены в ядре. Этот подход также способствует переносимости системы, так как функции драйверов устройств могут быть во многих случаях абстрагированы от аппаратной части.

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

Высокая степень переносимости обусловлена тем, что весь машинно-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. Абстрагируя интерфейсы прикладных программ от расположенных ниже операционных систем, микроядра позволяют гарантировать, что вложения в прикладные программы не пропадут в течение нескольких лет, даже если будут сменяться операционные системы и процессоры.

Расширяемость также является одним из важных требований к современным операционным системам. Является ли операционная система маленькой, как DOS, или большой, как UNIX, для нее неизбежно настанет необходимость приобрести свойства, не заложенные в ее конструкцию. Увеличивающаяся сложность монолитных операционных систем делала трудным, если вообще возможным, внесение изменений в ОС с гарантией надежности ее последующей работы. Ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС.

Обычно операционная система выполняется только в режиме ядра, а прикладные программы - только в режиме пользователя, за исключением тех случаев, когда они обращаются к ядру за выполнением системных функций. В отличие от обычных систем, система построенная на микроядре, выполняет свои серверные

подсистемы в режиме пользователя, как обычные прикладные программы. Такая структура позволяет изменять и добавлять серверы, не влияя на целостность микроядра.

Иногда имеется потребность и в сокращении возможностей ОС. На Windows NT или UNIX перешло бы большее число пользователей, если бы для этих операционных систем не требовалось 16 Мб оперативной памяти и 70 Мб и более пространства на жестком диске. Микроядро не обязательно подразумевает небольшую систему. Надстроенные службы, типа файловой системы и системы управления окнами, добавят к ней немало. Конечно же, не всем нужна безопасность класса C2 или распределенные вычисления. Если важные, но предназначенные для определенных потребителей свойства можно исключать из состава системы, то базовый продукт подойдет более широкому кругу пользователей.

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. При получении от процесса сообщения микроядро может обработать его самостоятельно или переслать другому процессу. Так как микроядру все равно, пришло ли сообщение от локального или удаленного процесса, подобная схема передачи сообщений является элегантным базисом для RPC. Однако такая гибкость не дается даром. Пересылка сообщений не так быстра, как обычные вызовы функций, и ее оптимизация является критическим фактором успеха операционной системы на основе микроядра. Windows NT, например, в некоторых случаях заменяет перенос сообщений на коммуникационные каналы с общей памятью, имеющие более высокую пропускную способность. Хотя это и стоит дороже в смысле потребления фиксированной памяти микроядра, данная альтернатива может помочь сделать модель пересылки сообщений более практичной.

Коммерческие версии микроядер

Одной из первых представила понятие микроядра фирма Next, которая использовала в своих компьютерах систему Mach, прошедшую большой путь развития в университете Карнеги-Меллона при помощи агентства Министерства обороны США DARPA. Теоретически, ее небольшое привилегированное ядро, окруженное службами пользовательского режима, должно было обеспечить беспрецедентную гибкость и модульность. Но на практике это преимущество было несколько уменьшено наличием монолитного сервера операционной

системы BSD 4.3, который выполнялся в пользовательском пространстве над микроядром Mach. Однако Mach дал Next возможность предоставить службу передачи сообщений и объектно-ориентированные средства, которые предстали перед конечными пользователями в качестве элегантного интерфейса пользователя с графической поддержкой конфигурирования сети, системного администрирования и разработки программного обеспечения.

Затем пришла Microsoft Windows NT, рекламировавшая в качестве ключевых преимуществ применения микроядра не только модульность, но и переносимость. Конструкция NT позволяет ей работать на системах на основе процессоров Intel, MIPS и Alpha (и последующих), и поддерживать симметричную многопроцессорность. Из-за того, что NT должна была выполнять программы, написанные для DOS, Windows, OS/2 и использующих соглашения POSIX, Microsoft использовала модульность, присущую микроядерному подходу для того, чтобы сделать структуру NT не повторяющей ни одну из существующих операционных систем. Вместо этого NT поддерживает каждую надстроенную операционную систему в виде отдельного модуля или подсистемы.

Более современные архитектуры микроядра были предложены Novell, USL, Open Software Foundation, IBM, Apple и другими. Одним из основных соперников NT на арене микроядер является микроядро Mach 3.0, которое и IBM и OSF взялись привести к коммерческому виду. (Next в качестве основы для NextStep пока использует Mach 2.5, но при этом внимательно присматривается к Mach 3.0.). Основной соперник Mach - микроядро Chorus 3.0 фирмы Chorus Systems, выбранный USL за основу для своих предложений. Это же микроядро будет использоваться в SpringOS фирмы Sun, объектно-ориентированном преемнике Solaris.

Сегодня стало ясно, что имеется тенденция движения от монолитных систем в сторону подхода с использованием небольших ядер. Именно такой подход уже использовался компаниями QNX Software и Unisys, в течение нескольких лет поставляющих пользующиеся успехом операционные системы на основе микроядра. QNX фирмы QNX Software обслуживает рынок систем реального времени, а CTOS фирмы Unisys популярна в области банковского дела.

Объектно-ориентированный подход

Хотя технология микроядер и заложила основы модульных систем, способных развиваться регулярным образом, она не смогла в полной мере обеспечить возможности расширения систем. В настоящее время этой цели в наибольшей степени соответствует объектно-ориентированный подход, при котором каждый программный компонент является функционально изолированным от других.

Основным понятием этого подхода является "объект". *Объект* - это единица программ и данных, взаимодействующая с другими объектам посредством приема и передачи сообщений. Объект может быть представлением как

некоторых конкретных вещей - прикладной программы или документа, так и некоторых абстракций - процесса, события.

Программы (функции) объекта определяют перечень действий, которые могут быть выполнены над данными этого объекта. Объект-клиент может обратиться к другому объекту, послав сообщение с запросом на выполнение какой-либо функции объекта-сервера.

Объекты могут описывать сущности, которые они представляют, с разной степенью детализации. Для обеспечения преемственности при переходе к более детальному описанию разработчикам предлагается механизм наследования свойств уже существующих объектов, то есть механизм, позволяющий порождать более конкретные объекты из более общих. Например, при наличии объекта "текстовый документ" разработчик может легко создать объект "текстовый документ в формате Word 6.0", добавив соответствующее свойство к базовому объекту. Механизм наследования позволяет создать иерархию объектов, в которой каждый объект более низкого уровня приобретает все свойства своего предка.

Внутренняя структура данных объекта скрыта от наблюдения. Нельзя произвольно изменять данные объекта. Для того, чтобы получить данные из объекта или поместить данные в объект, необходимо вызывать соответствующие объектные функции. Это изолирует объект от того кода, который использует его. Разработчик может обращаться к функциям других объектов, или строить новые объекты путем наследования свойств других объектов, ничего не зная о том, как они сконструированы. Это свойство называется инкапсуляцией.

Таким образом, объект предстает для внешнего мира в виде "черного ящика" с хорошо определенным интерфейсом. С точки зрения разработчика, использующего объект, пока внешняя реакция объекта остается без изменений, не имеют значения никакие изменения во внутренней реализации. Это дает возможность легко заменять одну реализацию объекта другой, например, в случае смены аппаратных средств; при этом сложное программное окружение, в котором находятся заменяемые объекты, не потребует никаких изменений.

С другой стороны, способность объектов представать в виде "черного ящика" позволяет упаковывать в них и представлять в виде объектов уже существующие приложения, ничего в них не изменяя.

Использование объектно-ориентированного подхода особенно эффективно при создании активно развивающегося программного обеспечения, например, при разработке приложений, предназначенных для выполнения на разных аппаратных платформах.

Полностью объектно-ориентированные операционные системы очень привлекательны для системных программистов, так как, используя объекты

системного уровня, программисты смогут залезать вглубь операционных систем для приспособления их к своим нуждам, не нарушая целостность системы.

Но особенно большие перспективы имеет этот подход в реализации распределенных вычислительных сред. В то время, как сейчас разные пакеты, работающие в данный момент в сети, представляют собой статически связанные наборы программ, в будущем, с использованием объектно-ориентированного подхода, они могут превратиться в единую совокупность динамически связываемых объектов, где каждый объект оперативно устанавливает и разрывает связи с другими объектами для выполнения актуальных в данный момент задач. Приложения, созданные для такой сетевой среды, основанной на объектах, могут выполняться, динамически обращаясь к множеству объектов, независимо от их местонахождения в сети и независимо от их операционной среды.

Поскольку любое объектно-ориентированное приложение представляет собой набор объектов, разработчику желательно иметь стандартные средства для управления объектами и организации их взаимодействия. При использовании и разработке объектно-ориентированных приложений в неоднородных распределенных средах, нужны также средства, упрощающие доступ к объектам сети. При возникновении запроса к какому-либо объекту распределенной среды, независимо от того, находится требуемый объект на том же компьютере или на одном из удаленных, прозрачным образом должен быть выполнен поиск объекта, передача ему сообщения, и возврат ответа. Для обеспечения прозрачного обнаружения объектов, все они должны быть снабжены ссылками, хранящимися в каталогах. Отсюда вытекает очень сложная проблема организации службы каталогов, позволяющей программистам именовать и искать объекты в сети, которая, вообще говоря, может быть разбросана по всему миру.

Однако, несмотря на упомянутые сложности и проблемы, объектно-ориентированный подход является одной из самых перспективных тенденций в конструировании программного обеспечения.

Коммерческие объектно-ориентированные средства

Объектно-ориентированный подход к построению операционных систем, придающий порядок процессу добавления модульных расширений к небольшому ядру был принят на вооружение многими известными фирмами, такими как Microsoft, Apple, IBM, Novell/USL (UNIX Systems Laboratories) и Sun Microsystems - все они развернули свои операционные системы в этом направлении. Taligent, совместное предприятие IBM и Apple, надеется опередить всех со своей от начала до конца объектно-ориентированной операционной системой. Тем временем Next поставляет Motorola- и Intel-версии NextStep, наиболее продвинутой объектно-ориентированной операционной системы из имеющихся. Хотя NextStep и не имеет объектной ориентированности сверху донизу, как это планируется в разработках Taligent, но она доступна уже сегодня.

Одним из первых применений объектных систем для большинства пользователей станут основанные на объектах прикладные программы. К объектно-ориентированным технологиям этого уровня, уже имеющимся сейчас или доступным в ближайшем будущем относятся:

- Microsoft OLE (Object Linking and Embedding - связывание и внедрение объектов),
- стандарт OpenDoc от фирм Apple, IBM, WordPerfect, Novell и Borland,
- DSOM (Distributed System Object Model - объектная модель распределенных систем) фирмы IBM,
- PDO (Portable Distributed Objects - переносимые распределенные объекты) фирмы Next,
- каркасы (frameworks) фирмы Taligent,
- архитектура CORBA объединения OMG.

Средства OLE

Для пользователей Windows объектно-ориентированный подход проявляется при работе с программами, использующими технологию OLE фирмы Microsoft. В первой версии OLE, которая дебютировала в Windows 3.1, пользователи могли вставлять объекты в документы-клиенты. Такие объекты устанавливали ссылку на данные (в случае связывания) или содержали данные (в случае внедрения) в формате, распознаваемом программой-сервером. Для запуска программы-сервера пользователи делали двойной щелчок на объекте, посредством чего передавали данные серверу для редактирования. OLE 2.0, доступная в настоящее время в качестве расширения Windows 3.1, переопределяет документ-клиент как контейнер. Когда пользователь щелкает дважды над объектом OLE 2.0, вставленным в документ-контейнер, он активизируется в том же самом месте. Представим, например, что контейнером является документ Microsoft Word 6.0, а вставленный объект представляет собой набор ячеек в формате Excel 5.0. Когда вы щелкнете дважды над объектом электронной таблицы, меню и управляющие элементы Word как по волшебству поменяются на меню Excel. В результате, пока объект электронной таблицы находится в фокусе, текстовый процессор становится электронной таблицей.

Инфраструктура, требуемая для обеспечения столь сложных взаимодействий объектов, настолько обширна, что Microsoft называет OLE 2.0 "1/3 операционной системы". Хранение объектов, например, использует docfile, который в действительности является миниатюрной файловой системой, содержащейся внутри обычного файла MS-DOS. Docfile имеет свои собственные внутренние механизмы для семантики подкаталогов, блокировок и транзакций (т.е. фиксации-отката).

Наиболее заметный недостаток OLE - отсутствие сетевой поддержки, и это будет иметь наивысший приоритет при разработке будущих версий OLE. Следующая

основная итерация OLE появится в распределенной, объектной версии Windows, называемой Cairo (Каир), ожидаемой в 1995 году.

Стандарт OpenDoc

Apple, совместно с WordPerfect, Novell, Sun, Xerox, Oracle, IBM и Taligent, известными вместе как Component Integration Laboratory (Лаборатория по объединению компонентов), также занимается архитектурой объектно-ориентированных составных документов, называемой OpenDoc. Создаваемый для работы на разных платформах, этот проект значительно отстает по степени готовности от OLE 2.0.

Ключевыми технологиями OpenDoc являются механизм хранения Бенто (названный так в честь японской тарелки с отделениями для разной пищи), технология сценариев (scripting), позаимствованная в значительной степени из AppleSript, и SOM фирмы IBM. В Бенто-документе каждый объект имеет постоянный идентификатор, перемещающийся вместе с ним от системы к системе. Хранение не только является транзакционным, как в OLE, но и позволяет держать и отслеживать многочисленные редакции каждого объекта. Если имеется несколько редакций документа, то реально храниться будут только изменения от одной редакции к другой. Верхняя граница числа сохраняемых редакций будет определяться пользователем.

Команда Apple планирует сделать OpenDoc совместимым с Microsoft OLE. Если план завершится успехом, система OpenDoc сможет окружать объекты OLE слоем программ трансляции сообщений. Контейнер OpenDoc будет видеть встроенный объект OLE как объект OpenDoc, а объект OLE будет представлять свой контейнер, как контейнер OLE. Утверждается, что будет допустима также и обратная трансляция по этому сценарию, когда объекты OpenDoc функционируют в контейнерах OLE. Слой трансляции разрабатывается WordPerfect при помощи Borland, Claris, Lotus и других.

В основе OLE и OpenDoc лежат две соперничающие объектные модели: Microsoft COM (Component Object Model - компонентная объектная модель) и IBM SOM. Каждая определяет протоколы, используемые объектами для взаимодействия друг с другом. Основное их различие заключается в том, что SOM нейтральна к языкам программирования и поддерживает наследование, тогда как COM ориентирована на C ++ и вместо механизма наследования использует альтернативный механизм, который Microsoft называет агрегацией.

Семейство CORBA

Hewlett-Packard, Sun Microsystems и DEC экспериментируют с объектами уже много лет. Теперь эти компании и много других объединились вместе, основав промышленную коалицию под названием OMG (Object Management Group), разрабатывающую стандарты для обмена объектами. OMG CORBA (Common Object Request Broker Architecture - Общая архитектура посредника обработки

объектных запросов) закладывает фундамент распределенных вычислений с переносимыми объектами. CORBA задает способ поиска объектами других объектов и вызова их методов. SOM согласуется с CORBA. Если вы пользуетесь DSOM под OS/2, вы сможете вызывать CORBA-совместимые объекты, работающие на HP, Sun и других архитектурах. Означает ли это возможность редактировать объект OpenDoc, сделанный на Macintosh, в документе-контейнере на RISC-рабочей станции? Вероятно, нет. CORBA может гарантировать только механизм нижнего уровня, посредством которого одни объекты вызывают другие. Для успешного взаимодействия требуется также понимание сообщений друг друга.

Множественные прикладные среды

В то время как некоторые идеи (например, объектно-ориентированный подход) непосредственно касаются только разработчиков и лишь косвенно влияют на конечного пользователя, концепция множественных прикладных сред приносит пользователю долгожданную возможность выполнять на своей ОС программы, написанные для других операционных систем и других процессоров.

И сейчас дополнительное программное обеспечение позволяет пользователям некоторых ОС запускать чужие программы (например, Mac и UNIX позволяют выполнять программы для DOS и Windows). Но в зарождающемся поколении операционных систем средства для выполнения чужих программ становятся стандартной частью системы. Выбор операционной системы больше не будет сильно ограничивать выбор прикладных программ. Хотя столкновение пользовательских интерфейсов программ для Mac, Windows и UNIX на одном и том же экране и заставит пользователя немного потрудиться, но все равно, множественные прикладные среды операционных систем скоро станут такими же стандартными, как мыши и меню.

Множественные прикладные среды обеспечивают совместимость данной ОС с приложениями, написанными для других ОС и процессоров, на двоичном уровне, а не на уровне исходных текстов. Для пользователя, купившего в свое время пакет (например, Lotus 1-2-3) для MS DOS, важно, чтобы он мог запускать этот полюбившийся ему пакет без каких-либо изменений и на своей новой машине, построенной, например, на RISC-процессоре, и работающей под управлением, например, Windows NT.

При реализации множественных прикладных сред разработчики сталкиваются с противоречивыми требованиями. С одной стороны, задачей каждой прикладной среды является выполнение программы по возможности так, как если бы она выполнялась на "родной" ОС. Но потребности этих программ могут входить в конфликт с конструкцией современной операционной системы. Специализированные драйверы устройств могут противоречить требованиям безопасности. Могут конфликтовать схемы управления памятью и оконные системы. Чисто экономические вопросы (например, стоимость лицензирования программ и угроза судебного преследования) также могут повлиять на дизайн

чужих прикладных сред. Но самой большой потенциальной проблемой является производительность - прикладная среда должна выполнять программы с приемлемой скоростью.

Этому требованию не могут удовлетворить широко используемые ранее эмулирующие системы. Для сокращения времени на выполнение чужих программ прикладные среды используют имитацию программ на уровне библиотек. Эффективность этого подхода связана с тем, что большинство сегодняшних программ работают под управлением GUI (графических интерфейсов пользователя) типа Windows, Mac или UNIX Motif, при этом приложения тратят большую часть времени, производя некоторые хорошо предсказуемые вещи. Они непрерывно выполняют вызовы библиотек GUI для манипулирования окнами и для других связанных с GUI действий. И это то, что позволяет прикладным средам возместить время, потраченное на эмулирование команды за командой. Тщательно сделанная прикладная среда имеет в своем составе библиотеки, имитирующие внутренние библиотеки GUI, но написанные на родном коде, то есть она совместима с программным интерфейсом другой ОС. Иногда такой подход называют трансляцией для того, чтобы отличать его от более медленного процесса эмулирования кода по одной команде за раз.

Например, для Windows-программы, работающей на Mac, при интерпретировании команд 80x86 производительность может быть очень низкой. Но когда производится вызов функции открытия окна, модуль прикладной среды может переключить его на перекомпилированную для 680x0 подпрограмму открытия окна. Так как библиотекам GUI не нужно дешифрировать и имитировать каждую команду, то в частях программы, относящихся к вызовам GUI ABI (Application Binary Interface - двоичный интерфейс прикладного программирования), производительность может резко вырасти. В результате на таких участках кода скорость работы программы может достичь (а возможно, и превзойти) скорость работы на своем родном процессоре.

Сегодня в типичных программах значительная часть кода занята вызовом GUI ABI. Apple утверждает, что программы для Mac тратят до 90 процентов процессорного времени на выполнение подпрограмм из Mac toolbox, а не на уникальные для этих программ действия. SunSelect говорит, что программы для Windows тратят от 60 до 80 процентов времени на работу в ядре Windows. В результате при эмуляции программы на основе GUI потери производительности могут быть значительно меньше. SunSelect заявляет, что его новая прикладная среда Windows, WABI (Windows Application Binary Interface - двоичный интерфейс прикладных программ Windows), благодаря сильно оптимизированным библиотекам, на некоторых платформах при исполнении одних и тех же тестов может обогнать настоящий Microsoft Windows.

С позиции использования прикладных сред более предпочтительным является способ написания программ, при котором программист для выполнения некоторой функции обращается с вызовом к операционной системе, а не пытается более эффективно реализовать эквивалентную функцию самостоятельно, работая

напрямую с аппаратурой. Отбить у программистов охоту "обращаться к металлу" сможет наличие в библиотеках мощных и сложных программ, к которым гораздо проще обращаться, чем писать самому.

Модульность операционных систем нового поколения позволяет намного легче реализовать поддержку множественных прикладных сред. В отличие от старых операционных систем, состоящих из одного большого блока для всех практических применений, разбитого произвольным образом на части, новые системы являются модульными, с четко определенными интерфейсами между составляющими. Это делает создание дополнительных модулей, объединяющих эмуляцию процессора и трансляцию библиотек, значительно более простым.

К усовершенствованным операционным системам, явно содержащим средства множественных прикладных сред, относятся: IBM OS/2 2.x и Workplace OS, Microsoft Windows NT, PowerOpen компании PowerOpen Association и версии UNIX от Sun Microsystems, IBM и Hewlett-Packard. Кроме того, некоторые компании переделывают свои интерфейсы пользователя в виде модулей прикладных сред, а другие поставщики предлагают продукты для эмуляции и трансляции прикладных сред, работающие в качестве прикладных программ.

Существует много разных стратегий по воплощению идеи множественных прикладных сред, и некоторые из этих стратегий диаметрально противоположны. В случае UNIX, транслятор прикладных сред обычно делается, как и другие прикладные программы, плавающим на поверхности операционной системы. В более современных операционных системах типа Windows NT или Workplace OS модули прикладной среды выполняются более тесно связанными с операционной системой, хотя и обладают по-прежнему высокой независимостью. А в OS/2 с ее более простой, слабо структурированной архитектурой средства организации прикладных сред встроены глубоко в операционную систему.

Использование множественных прикладных сред обеспечит пользователям большую свободу выбора операционных систем и более легкий доступ к более качественному программному обеспечению.

Сетевой пакет DCE фирмы OSF

Распределенные вычисления имеют дело с понятиями более высокого уровня, чем физические носители, каналы связи и методы передачи по ним сообщений. Распределенная среда должна дать пользователям и приложениям прозрачный доступ к данным, вычислениям и другим ресурсам в гетерогенных системах, представляющих собой набор средств различных производителей. Стратегические архитектуры каждого крупного системного производителя базируются сейчас на той или иной форме распределенной вычислительной среды (DCE). Ключом к пониманию выгоды такой архитектуры является прозрачность. Пользователи не должны тратить свое время на попытки выяснить, где находится тот или иной ресурс, а разработчики не должны писать коды для своих приложений, использующие местоположение в сети. Никто не

заинтересован в том, чтобы заставлять разработчиков приложений становиться гуру коммуникаций, или же в том, чтобы заставлять пользователей заботиться о монтировании удаленных томов. Кроме того, сеть должна быть управляемой. Окончательной картиной является виртуальная сеть: набор сетей рабочих групп, отделов, предприятий, объединенных сетей предприятий, которые кажутся конечному пользователю или приложению единой сетью с простым доступом.

Одной из технологий, которая будет играть большое значение для будущего распределенных вычислений является технология DCE (Distributed Computing Environment) некоммерческой организации Open Software foundation (OSF). DCE OSF - это интегрированный набор функций, независимых от операционной системы и сетевых средств, которые обеспечивают разработку, использование и управление распределенными приложениями. Из-за их способности обеспечивать управляемую, прозрачную и взаимосвязанную работу систем различных производителей и различных платформ, DCE является одной из самых важных технологий десятилетия.

Большинство из ведущих фирм-производителей ОС договорились о поставках DCE в будущих версиях своих системных и сетевых продуктов. Например, IBM, которая предлагает несколько DCE-продуктов, базирующихся на AIX, распространила их и на сектор сетей персональных компьютеров. В сентябре 1993 года IBM начала поставлять инструментальные средства для разработки DCE-приложений - DCE SDK для OS/2 и Windows, и в это же время выпустила на рынок свой первый DCE-продукт для пользователей персональных компьютеров - DCE Client for OS/2. Компания IBM достаточно далеко продвинулась в реализации спецификации DCE в своих продуктах, обогнав Microsoft, Novell и Banyan. Сейчас она продает как отдельный продукт клиентскую часть служб DCE для операционных сред MVS, OS/400, OS/2, AIX и Windows. IBM собирается реализовать отсутствующую в LAN Server единую справочную службу и новую службу безопасности в соответствии со спецификацией DCE и выпустить интегрированный вариант DCE/LAN Server в конце этого года.

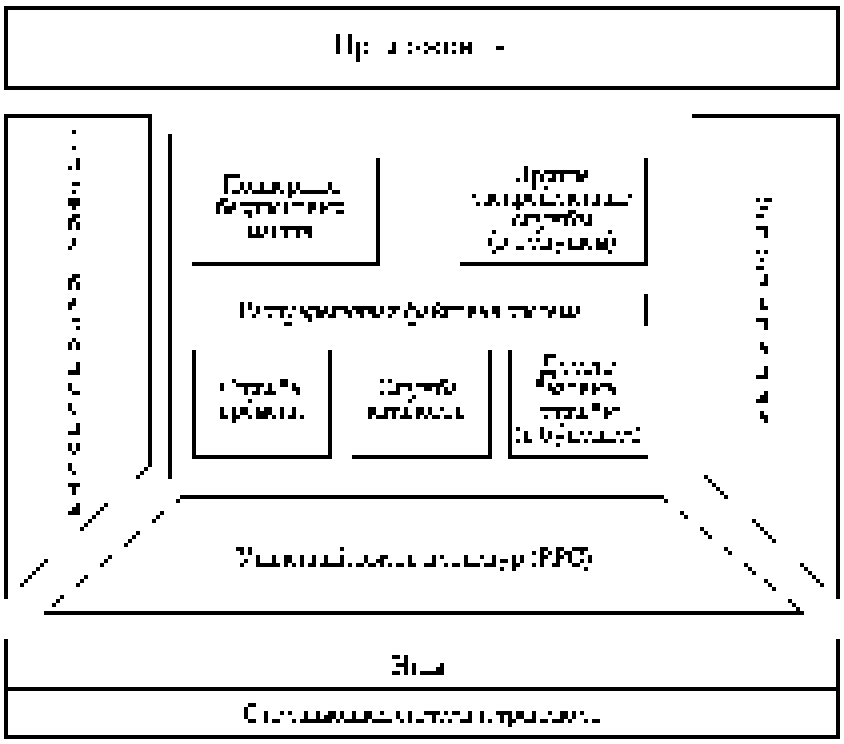


Рис. 4.4. Архитектура средств OSF DCE

Некоторые крупные фирмы-потребители программных продуктов обращаются непосредственно в OSF за лицензиями на первые версии DCE OSF.

В настоящее время DCE состоит из 7 средств и функций, которые делятся на базовые распределенные функции и средства разделения данных. Базовые распределенные функции включают:

1. нити,
2. RPC,
3. службу каталогов,
4. службу времени,
5. службу безопасности.

Функции разделения данных строятся над базовыми функциями и включают:

1. распределенную файловую систему (DFS),
2. поддержку бездисковых машин.

На рисунке 4.4 показана архитектура DCE OSF. Эта архитектура представляет собой многоуровневый набор интегрированных средств. Внизу расположены базисные службы, такие как ОС, а на самом верхнем уровне находятся потребители средств DCE приложения. Средства безопасности и управления пронизывают все уровни. OSF резервирует место для функций, которые могут появиться в будущем, таких как спулинг, поддержка транзакций и распределенная объектно-ориентированная среда.

Нити

Обычно приложения имеют дело с процессами, каждый из которых состоит из одной нити управления.

Нити - это важная модель для выражения параллелизма внутри процесса, особенно для распределенного окружения. Например, возможности многонитевой обработки становятся особенно важными в контексте RPC. RPC является синхронным механизмом по своей природе: клиент делает вызов удаленной функции и ожидает выполнения вызова. При использовании нитей одна нить может сделать запрос, а другая начать обрабатывать данные от другого запроса. Следовательно, использование нитей может существенно улучшить производительность распределенного приложения.

Нитевая модель предъявляет меньшие требования к искусству программиста, чем другие альтернативы параллелизма, такие как асинхронные операции или разделение памяти.

Поскольку поддержка нитей дает большой выигрыш в производительности, большинство современных операционных систем является многонитевыми. Однако многие используемые в настоящее время операционные системы таковыми не являются. DCE OSF предлагает специальный пакет, предназначенный для обеспечения многонитевости для организации распределенных приложений в ОС, не имеющих собственных встроенных средств поддержки многонитевости. Этот пакет выполнен как библиотека функций работы с нитями.

По сравнению со встроенными в ядро ОС функциями поддержки нитей, библиотечная их реализация имеет некоторые функциональные ограничения. С другой стороны, в этом случае предоставляются хорошие шансы для обеспечения совместимости свойств нитей в гетерогенных средах.

Все остальные службы DCE OSF - RPC, службы безопасности, каталогов и времени, распределенная файловая система - используют сервис этого пакета.

Рассмотрим пакет, реализующий нити в DCE OSF, более подробно. Как и большинство программного обеспечения DCE, этот пакет большой и сложный. Он состоит из 51 библиотечной функции, относящейся к нитям, которыми могут пользоваться прикладные программы. Многие из них не являются необходимыми, а разработаны только для удобства. Мы рассмотрим только основные.

Эти функции удобно разделить на 7 категорий, каждая из которых имеет дело с определенным аспектом работы с нитями. Первая категория имеет дело с управлением нитями. В нее входят вызовы:

CREATE - создает новую нить

EXIT - вызывается нитью при завершении

JOIN - подобен системному вызову WAIT в UNIX

DETACH - отсоединение нити-потомка

Эти вызовы позволяют создавать и завершать нити. Родительская нить может ждать потомка, используя вызов join, подобно вызову wait в UNIX'e. Если родительская нить не планирует этого, то она может отсоединиться от потомка, сделав вызов detach. В этом случае, когда нить-потомок завершается, ее память освобождается немедленно, вместо обычного ожидания родителя, который издал вызов join.

Вторая категория функций позволяет пользователю создавать, разрушать и управлять шаблонами для нитей, для семафоров (в данном случае имеются ввиду бинарные семафоры, имеющие два состояния, называемые мьютексами). Шаблоны могут использоваться с соответствующими начальными условиями. При создании объекта один из параметров вызова create является указателем на шаблон. Например, шаблон нити имеет по умолчанию размер стека 8 К. Все нити, созданные с помощью этого шаблона имеют размер стека 8 К. Смысл использования шаблонов в том, что они позволяют не задавать все параметры создаваемого объекта.

Существуют системные вызовы добавления новых атрибутов к шаблонам. Вызовы attr_create и attr_delete создают и удаляют шаблоны нитей. Другие вызовы позволяют программе читать и модифицировать атрибуты шаблона, такие как размер стека или параметры планирования. Имеются также шаблоны для создания и удаления семафоров.

Третья группа вызовов содержит пять функций для управления мьютексами: создание, удаление, блокирование, разблокирование.

При работе с семафорами возникает очевидный вопрос, что случится, если использовать операцию "разблокировать" по отношению к мьютексу, который не заблокирован. Сохранится ли эта операция или будет потеряна? Эта проблема и вынудила Дейкстру ввести в свое время семафоры, при использовании которых порядок выполнения операций блокирования и разблокирования не имеет значения. В результате не возникает эффекта гонок. Логика работы мьютекса в DCE зависит от реализации, что, конечно, не позволяет писать переносимые программы.

Существует два типа мьютексов в DCE - быстрые и дружественные. Быстрый мьютекс является аналогом блокировки в СУБД. Если процесс пытается заблокировать незаблокированную запись, то эта операция выполняется успешно. Однако, если он попытается применить эту блокировку второй раз, то он сам заблокируется, ожидая, пока кто-нибудь не снимет блокировку с мьютекса.

Дружественный мьютекс позволяет нити заблокировать уже заблокированный мьютекс. Предположим, что главная программа блокирует семафор, а затем вызывает функцию, которая также блокирует мьютекс. Чтобы избежать клинча, принимается и вторая блокировка. Если количество открытий и закрытий мьютекса совпадает, связанные операции блокирования-разблокирования могут иметь произвольную глубину вложения. Разработчики пакета, очевидно не пришли к согласию относительно того, что делать с повторной блокировкой от той же нити, и решили реализовать обе версии.

RPC

Хорошо известный механизм для реализации распределенных вычислений, RPC, расширяет традиционную модель программирования - вызов процедуры - для использования в сети. RPC может составлять основу распределенных вычислений. Теоретически программист не должен становиться экспертом по коммуникационным средствам для того, чтобы написать распределенное сетевое приложение. При использовании RPC программист будет применять специальный язык для описания операций. Данное описание обрабатывается компилятором, который создает код как для клиента, так и для сервера. Для обеспечения такого типа функций средства RPC должны быть простыми, прозрачными, надежными и высокопроизводительными.

Средства RPC пакета DCE OSF обладают простотой. Они приближаются к модели вызова локальных процедур насколько это возможно. Они почти не требуют изменения концептуального мышления программиста, и поэтому уменьшают время его переподготовки. Это особенно важно для коллективов программистов, работающих в фирмах, которые не являются производителями коммерческих программных продуктов.

Неизменность протокола - это другая особенность RPC DCE OSF. Этот протокол четко определен и не может изменяться пользователем (в данном случае разработчиком сетевых приложений). Такая неизменность, гарантируемая ядром, является важным свойством в гетерогенных средах, требующих согласованной работы. В отличие от OSF, некоторые другие разработчики средств RPC полагают, что гибкость и возможность приспосабливать эти средства к потребностям пользователей являются более важными.

Средства RPC DCE OSF поддерживают ряд транспортных протоколов и позволяют добавлять новые транспортные протоколы, сохраняя при этом свои функциональные свойства.

DCE RPC эффективно используется в других службах DCE: в службах безопасности, каталогов и времени, в распределенной файловой системе. DCE RPC интегрируется с системой идентификации для обеспечения защиты доступа и с нитями клиента и сервера для того, чтобы, сохраняя синхронный характер выполнения вызова, обеспечить параллелизм. Способность RPC посылать и

получать потоки типизированных данных неопределенной длины используется в распределенной файловой системе.

Распределенная служба каталогов

Задачей службы каталогов в распределенной сети является поиск сетевых объектов, то есть пользователей, ресурсов, данных или приложений. Служба каталогов (или иначе, имен) должна отобразить большое количество системных объектов (пользователей, организаций, групп, компьютеров, принтеров, файлов, процессов, сервисов) на множество понятных пользователю имен. Эта проблема сложна даже для гомогенных сетей, так как персонал и оборудование перемещается, изменяет свои имена, местонахождение и т.д. В гетерогенных глобальных сетях служба каталогов становится намного более сложной, из-за необходимости синхронизации различных баз данных каталогов. Более того, при появлении в сети распределенных приложений служба каталогов должна начать отслеживание всех таких объектов и всех их компонентов.

Хорошая служба каталогов делает использование распределенного окружения прозрачным для пользователя. Пользователям не нужно знать расположение удаленного принтера, файла или приложения.

OSF определяет двухярусную архитектуру для службы каталогов для целей адресации межячеечного и глобального взаимодействия. Ячейка (cell) - это фундаментальная организационная единица для систем в DCE OSF. Ячейки могут иметь социальные, политические или организационные границы. Ячейки состоят из компьютеров, которые должны часто взаимодействовать друг с другом - это могут быть, например, рабочие группы, отделы, или отделения компаний. В общем случае компьютеры в ячейке географически близки. Размер ячеек изменяется от 2 до 1000 компьютеров, хотя OSF считает наиболее приемлемым диапазон от десятков до сотен компьютеров.

Некоторые производители и пользователи агитируют за реализацию X.500 как общей службы каталогов на всех уровнях. Но OSF полагает, что использование X.500 на уровне рабочей группы (то есть ячейки) было бы слишком громоздким из-за требований к программному обеспечению по производительности - особенно, когда более гибкие средства службы каталогов уровня ячейки уже существуют на рынке.

Служба каталогов DCE состоит из 4-х элементов:

- CDS (Cell Directory Service) - служба каталогов ячейки. Ячейка сети - это группа систем, администрируемых как единое целое. CDS оптимизируется для локального доступа. Большинство запросов к службе каталогов относятся к ресурсам той же ячейки. Каждая ячейка сети нуждается по крайней мере в одной CDS.
- GDA (Global Directory Agent) - агент глобального каталога. GDA - это шлюз имен, который соединяет домен DCE с другими административными

доменами посредством глобальной службы каталогов X.500 и DNS (domain name service - сервис имен домена). GDA передает запрос на имя, которое он не смог найти в локальной ячейке, службе каталогов другой ячейки, или глобальной службе каталогов (в зависимости от места хранения имени). Для того, чтобы отыскать имя, клиент делает запрос к локальному агенту GDA. Затем GDA передает запрос на междоменное имя службе X.500. Эта служба возвращает ответ GDA, который в свою очередь передает его клиенту. OSF GDA может быть совместим с любой схемой глобального именования.

- GDS (Global Directory Service) - глобальная служба каталогов. Основанная на стандарте X.500, GDS работает на самом верхнем уровне иерархии и обеспечивает связь множества ячеек в множестве организаций.
- XDS (X/Open Directory Service) - обеспечивает поддержку X/Open API функций службы каталогов и позволяет разработчикам писать приложения, независимые от нижележащих уровней архитектуры службы каталогов. XDS-совместимые приложения будут работать одинаковым образом со службами каталогов DCE и X.500.

Распределенная служба безопасности

Имеется две больших группы функций службы безопасности: идентификация и авторизация. Идентификация проверяет идентичность объекта (например, пользователя или сервиса). Авторизация (или управление доступом) назначает привилегии объекту, такие как доступ к файлу.

Авторизация - это только часть решения. В распределенной сетевой среде обязательно должна работать глобальная служба идентификации, так как рабочей станции нельзя доверить функции идентификации себя или своего пользователя. Служба идентификации представляет собой механизм передачи третьей стороне функций проверки идентичности пользователя.

Служба безопасности OSF DCE базируется на системе идентификации Kerberos, разработанной в 80-е годы и расширенной за счет добавления элементов безопасности. Kerberos использует шифрование, основанное на личных ключах, для обеспечения трех уровней защиты. Самый нижний уровень требует, чтобы пользователь идентифицировался только при установлении начального соединения, предполагая, что дальнейшая последовательность сетевых сообщений исходит от идентифицированного пользователя. Следующий уровень требует идентификацию для каждого сетевого сообщения. На последнем уровне все сообщения не только идентифицируются, но и шифруются.

Система безопасности не должна сильно усложнять жизнь конечного пользователя в сети, то есть он не должен запоминать десятки паролей и кодов.

Весьма полезным сетевым средством для целей безопасности является служба прав доступа или, другими словами, авторизация. Служба авторизации OSF базируется на POSIX-совместимых списках прав доступа - ACL.

В то время как система Kerberos основана на личных ключах, в настоящее время широкое распространение получили методы, основанные на публичных ключах (например, метод RSA). OSF собирается сделать DCE-приложения переносимыми из Kerberos в RSA.

Распределенная файловая система DFS OSF

Распределенная файловая система DFS OSF предназначена для обеспечения прозрачного доступа к любому файлу, расположенному в любом узле сети. Главная концепция такой распределенной файловой системы - это простота ее использования.

Распределенная файловая система должна иметь единое пространство имен. Файл должен иметь одинаковое имя независимо от того, где он расположен. Другими желательными свойствами являются интегрированная безопасность, согласованность и доступность данных, надежность и восстанавливаемость, производительность и масштабируемость до очень больших конфигураций без уменьшения производительности и независимое от места расположения управление и администрирование.

Распределенная файловая система DFS OSF базируется на известной файловой системе AFS (The Andrew File System).

Файловая система AFS

AFS была разработана в университете Карнеги-Меллона и названа в честь спонсоров-основателей университета Andrew Carnegie и Andrew Mellon. Эта система, созданная для студентов университета, не является прозрачной системой, в которой все ресурсы динамически назначаются всем пользователям при возникновении потребностей. Несмотря на это, файловая система была спроектирована так, чтобы обеспечить прозрачность доступа каждому пользователю, независимо от того, какой рабочей станцией он пользуется.

Особенностью этой файловой системы является возможность работы с большим (до 10 000) числом рабочих станций.

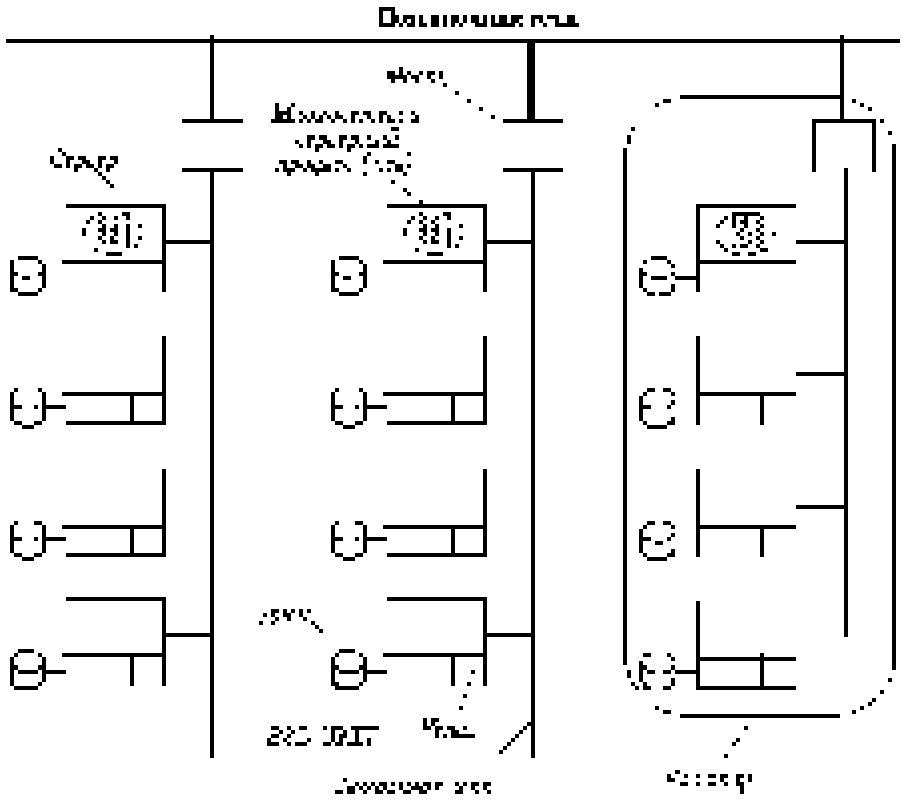


Рис. 4.5. Конфигурация системы, используемая AFS в университете Карнеги-Меллона

Конфигурация системы показана на рисунке 4.5. Она состоит из кластеров, каждый из которых включает файловый сервер и несколько десятков рабочих станций. Идея состоит в том, чтобы распределить большую часть трафика в пределах отдельных кластеров и тем самым уменьшить загрузку позвоночника сети.

Так как студенты могли входить в систему и в университете, и в общежитии, то иногда они оказывались далеко от сервера, который содержал их файлы. Несмотря на это, пользователь должен иметь возможность работать на произвольно выбранной рабочей станции, как на своем персональном компьютере.

Физически нет разницы между машиной клиента и сервера, и все они выполняют одну и ту же ОС BSD UNIX с его большим монолитным ядром. Однако над ядром выполняются совершенно различные программы серверов и клиентов. На клиент-машинах выполняются менеджеры окон, редакторы и другое стандартное программное обеспечение системы UNIX. Каждый клиент имеет также часть кода - venus, которая управляет интерфейсом между клиентом и серверной частью системы, называемой vice. Вначале venus выполнялся в пользовательском режиме, но позже он был перемещен в ядро для повышения производительности. Venus работает также в качестве менеджера кэша. В дальнейшем мы будем называть venus просто клиентом, а vice - сервером.

Пространство имен, видимое пользовательскими программами, выглядит как традиционное дерево в ОС UNIX с добавленным к нему каталогом /cmu (рисунок 4.5). Содержимое каталога /cmu поддерживается AFS посредством vice-серверов и идентично на всех рабочих станциях. Другие каталоги и файлы исключительно локальны и не разделяются. Возможности разделяемой файловой системы предоставляются путем монтирования к каталогу /cmu. Файл, который UNIX ожидает найти в верхней части файловой системы, может быть перемещен символьной связью из разделяемой файловой системы (например, /bin/sh может быть символьно связан с /cmu/bin/sh).

В основе AFS лежит стремление делать для каждого пользователя как можно больше на его рабочей станции и как можно меньше взаимодействовать с остальной системой. При открытии удаленного файла весь файл (или его значительная часть, если он очень большой) загружается на диск рабочей станции и кэшируется там, причем процесс, который сделал вызов OPEN, даже не знает об этом. По этой причине каждая рабочая станция имеет диск.

После загрузки файла на локальный диск он помещается в локальный каталог /cash, так что он выглядит для ОС как нормальный файл. Дескриптор файла, возвращаемый системным вызовом OPEN, соответствует именно этому файлу, так что вызовы READ и WRITE работают обычным путем, без участия клиента и сервера. Другими словами, хотя системный вызов OPEN значительно изменен, реализация READ и WRITE не изменилась.

Безопасность - это главный вопрос в системе с 10 000 пользователей. Так как пользователи вольны перегружать свои рабочие станции, когда захотят и могут выполнять на них модифицированные версии ОС, то главный принцип сервера - не доверять клиентским рабочим станциям. Все сообщения между рабочими станциями шифруются на уровне аппаратуры.

Защита выполнена несколько необычным путем. Каталоги защищаются списками прав доступа (ACL), но файлы имеют обычные биты RWX UNIX'a. Разработчики системы предпочитают механизм ACL, но так как многие UNIX-программы работают с битами RWX, то они оставлены для совместимости. Списки прав доступа могут содержать и отсутствие прав, так что можно, например, потребовать, чтобы доступ к файлу был разрешен для всех, кроме одного конкретного человека.

Диски рабочих станций используются только для временных файлов, кэширования удаленных файлов и хранения страниц виртуальной памяти, но не для постоянной информации. Это существенно упрощает управление системой, в этом случае нужно управлять и архивировать только файлы серверов, а рабочие станции не требуют никаких забот. Концептуально они могут начинать каждый рабочий день с чистого листа.

AFS сконструирована для расширения до масштабов национальной файловой системы. Система, показанная на рисунке, на самом деле представляет собой

отдельную ячейку (cell). Каждая ячейка - это административная единица, такая как отдел или компания. Ячейки могут быть соединены друг с другом с помощью монтирования, так что дерево разделяемых файлов может покрывать многие города.

В дополнение к концепциям файла, каталога и ячейки AFS поддерживает еще одно важное понятие - том. Том - это собрание каталогов, которые управляются вместе. Обычно все файлы какого-либо пользователя составляют том. Таким образом поддерево, входящее в /usr/john, может быть одним томом, поддерево, входящее в /usr/mary, может быть другим томом. Фактически, каждая ячейка представляет собой ничто иное, как набор томов, соединенных вместе некоторым, подходящим с точки зрения монтирования, образом. Большинство томов содержат пользовательские файлы, некоторые другие используются для двоичных выполняемых файлов и другой системной информации. Тома могут иметь признак "только для чтения".

Семантика, предлагаемая AFS, близка к сессионной семантике. Когда файл открывается, он берется у подходящего сервера и помещается в каталог /cash на локальном диске на рабочей станции. Все операции чтения-записи работают с кэшированной копией. При закрытии файла он выгружается назад на сервер. Следствием этой модели является то, что когда процесс открывает уже открытый файл, то версия, которую он видит, зависит от того, где находится процесс. Процесс на той же рабочей станции видит копию в каталоге /cash, при этом выполняется вся семантика UNIX.

В то же время процесс на другой рабочей станции продолжает видеть исходную версию файла на сервере. Только после того, как файл будет закрыт и отослан обратно на сервер, последующая операция открытия увидит новую версию. После того, как файл закрывается, он остается в кэше, на случай, если он скоро будет снова открыт. Как мы видели ранее, повторное открытие файла, который находится в кэше, порождает проблему: "Как клиент узнает, последняя ли это версия файла?" В первой версии AFS эта проблема решалась прямым запросом клиента к серверу. К сожалению эти запросы создавали большой трафик и впоследствии алгоритм был изменен. В новом алгоритме, когда клиент загружает файл в свой кэш, то он сообщает серверу, что его интересуют все операции открытия этого файла процессами на других рабочих станциях. В этом случае сервер создает таблицу, отмечающую местонахождение этого кэшированного файла. Если другой процесс где-либо в системе открывает этот файл, то сервер посылает сообщение клиенту, чтобы тот отметил этот вход кэша как недействительный. Если этот файл в настоящее время используется, то использующие его процессы могут продолжать делать это. Однако, если другой процесс пытается открыть его заново, то клиент должен свериться с сервером, действителен ли все еще этот вход в кэше, а если нет, то получить новую копию. Если рабочая станция терпит крах, а затем перезагружается, то все файлы в кэше отмечаются как недействительные.

Блокировка файла поддерживается с помощью системного вызова UNIX FLOCK. Если блокировка не снимается в течение 30 минут, то она снимается по тайм-ауту. Тома, предназначенные только для чтения, такие как системные двоичные файлы, реплицируются, а пользовательские файлы - нет.

Хотя прикладные программы видят традиционное пространство имен UNIX, внутренняя организация сервера и клиента использует совершенно другую схему имен. Они используют двухуровневую схему именования, при которой каталог содержит структуры, называемые fids (file identifiers), вместо традиционных номеров i-узлов.

Fid состоит из трех 32-х битных полей. Первое поле - это номер тома, который однозначно определяет отдельный том в системе. Это поле говорит, на каком томе находится файл. Второе поле называется vnode, это индекс в системных таблицах определенного тома. Оно определяет конкретный файл в данном томе. Третье поле - это уникальный номер, который используется для обеспечения повторного использования vnode. Если файл удаляется, то его vnode может быть повторно использован, но с другим значением уникального номера, для того, чтобы обнаружить и отвергнуть все старые fids.

Протокол между сервером и клиентом использует fid для идентификации файла. Когда fid поступает в сервер, по значению номера тома производится поиск в базе данных, управляемой всеми серверами, чтобы обнаружить нужный сервер. Тома могут перемещаться между серверами, но не части томов, так что эта база данных требует периодического обновления, но перемещения томов случается редко, так что трафик обновления невелик. Перемещение тома является неделимым - сначала на сервере назначения делается копия тома, а затем удаляется оригинал. Этот механизм также используется для репликации томов только для чтения за исключением того, что исходный том не удаляется после его копирования. Этот же алгоритм используется для резервного копирования. Когда делается копия, то она помещается в файловую систему как том только для чтения. В течение последующих 24 часов процесс скопирует этот том на ленту. Дополнительное преимущество этого метода - пользователь, который случайно удалил файл, все еще имеет доступ ко вчерашней копии.

Теперь рассмотрим общий механизм доступа к файлам в AFS. Когда приложение выполняет системный вызов OPEN, то он перехватывается оболочкой клиента, которая первым делом проверяет, не начинается ли имя файла с /cmu. Если нет, то файл локальный, и обрабатывается обычным способом. Если да, то файл разделяемый. Производится грамматический разбор имени, покомпонентно находится fid. По fid проверяется кэш, и здесь имеется три возможности:

1. Файл находится в кэше, и он достоверен.
2. Файл находится в кэше, и он не достоверен.
3. Файл не находится в кэше.

В первом случае используется кэшированный файл. Во втором случае клиент запрашивает сервер, изменялся ли файл после его загрузки. Файл может быть недостоверным, если рабочая станция недавно перезагружалась или же некоторый другой процесс открыл файл для записи, но это не означает, что файл уже модифицирован, и его новая копия записана на сервер. Если файл не изменялся, то используется кэшированный файл. Если он изменялся, то используется новая копия. В третьем случае файл также просто загружается с сервера. Во всех трех случаях конечным результатом будет то, что копия файла будет на локальном диске в каталоге /cash, отмеченная как достоверная.

Вызовы приложения READ и WRITE не перехватываются оболочкой клиента, они обрабатываются обычным способом. Вызовы CLOSE перехватываются оболочкой клиента, которая проверяет, был ли модифицирован файл, и, если да, то переписывает его на сервер, который управляет данным томом.

Помимо кэширования файлов, оболочка клиента также управляет кэшем, который отображает имена файлов в идентификаторы файлов fid. Это ускоряет проверку, находится ли имя в кэше. Проблема возникает, когда файл был удален и заменен другим файлом. Однако этот новый файл будет иметь другое значение поля "уникальный номер", так что fid будет выявлен как недостоверный. При этом клиент удалит вход (pass, fid) и начнет грамматический разбор имени с самого начала. Если дисковый кэш переполняется, то клиент удаляет файлы в соответствии с алгоритмом LRU.

Vice работает на каждом сервере как отдельная многонитевая программа. Каждая нить обрабатывает один запрос. Протокол между сервером и клиентом использует RPC и построен непосредственно на API. В нем есть команды для перемещения файлов в обоих направлениях, блокирования файлов, управления каталогами и некоторые другие. Vice хранит свои таблицы в виртуальной памяти, так что они могут быть произвольной величины.

Так как клиент идентифицирует файлы по их идентификаторам fid, то у сервера возникает следующая проблема: как обеспечить доступ к UNIX-файлу, зная его vnode, но не зная его полное имя. Для решения этой проблемы в AFS в UNIX добавлен новый системный вызов, позволяющий обеспечить доступ к файлам по их индексам vnode.

Реализация DFS на базе AFS дает прекрасный пример того, как работают вместе различные компоненты DCE. DFS работает на каждом узле сети совместно со службой каталогов DCE, обеспечивая единое пространство имен для всех файлов, хранящихся в DFS. DFS использует списки ACL системы безопасности DCE для управления доступом к отдельным файлам. Потоковые функции RPC позволяют DFS передавать через глобальные сети большие объемы данных за одну операцию.

Распределенная служба времени

В распределенных сетевых системах необходимо иметь службу согласования времени. Многие распределенные службы, такие как распределенная файловая система и служба идентификации, используют сравнение дат, сгенерированных на различных компьютерах. Чтобы сравнение имело смысл, пакет DCE должен обеспечивать согласованные временные отметки.

Сервер времени OSF DCE - это система, которая поставляет время другим системам в целях синхронизации. Любая система, не содержащая сервера времени, называется клерком (clerk). Распределенная служба времени использует три типа серверов для координации сетевого времени. Локальный сервер синхронизируется с другими локальными серверами той же локальной сети. Глобальный сервер доступен через расширенную локальную или глобальную сети. Курьер (courier) - это специальный локальный сервер, который периодически сверяет время с глобальными серверами. Через периодические интервалы времени серверы синхронизируются друг с другом с помощью протокола DTS OSF. Этот протокол может взаимодействовать с протоколом синхронизации времени NTP сетей Internet.

Многие фирмы-потребители программного обеспечения уже используют или собираются использовать средства DCE, поэтому ведущие фирмы-производители программного обеспечения, такие как IBM, DEC и Hewlett-Packard, заняты сейчас реализацией и поставкой различных элементов и расширений этой технологии.

Одной из главных особенностей и достоинств пакета DCE OSF является тесная взаимосвязь всех его компонентов. Это свойство пакета иногда становится его недостатком. Так, очень трудно работать в комбинированном окружении, когда одни приложения используют базис DCE, а другие - нет. В версии 1.1 совместимость служб пакета с аналогичными средствами других производителей улучшена. Например, служба Kerberos DCE в текущей версии несовместима с реализацией Kerberos MIT из-за того, что Kerberos DCE работает на базе средств RPC DCE, а Kerberos MIT - нет. OSF обещает полную совместимость с Kerberos MIT в версии 1.1. Имеются и положительные примеры совместимости пакета DCE со средствами других производителей, например со средствами Windows NT. Хотя Windows NT и не является платформой DCE, но их совместимость может быть достигнута за счет полной совместимости средств RPC. Поэтому, после достаточно тщательной работы на уровне исходных кодов, разработчики могут создать DCE-сервер, который сможет обслуживать Windows NT-клиентов, и Windows NT-сервер, который работает с DCE-клиентами.

Для того, чтобы стать действительно распространенным базисом для создания гетерогенных распределенных вычислительных сред, пакет DCE должен обеспечить поддержку двух ключевых технологий - обработку транзакций и объектно-ориентированный подход. Поддержка транзакций совершенно необходима для многих деловых приложений, когда недопустима любая потеря данных или их несогласованность. Две фирмы - IBM и Transarc - предлагают дополнительные средства, работающие над DCE и обеспечивающие обработку

транзакций. Что же касается объектно-ориентированных свойств DCE, то OSF собирается снабдить этот пакет средствами, совместимыми с объектно-ориентированной архитектурой CORBA, и работающими над инфраструктурой DCE. После достаточно тщательной работы на уровне исходных кодов, разработчики могут создать DCE-сервер, который сможет обслуживать Windows NT-клиентов, и Windows NT-сервер, который работает с DCE-клиентами.

Для того, чтобы стать действительно распространенным базисом для создания гетерогенных распределенных вычислительных сред, пакет DCE должен обеспечить поддержку двух ключевых технологий - обработку транзакций и объектно-ориентированный подход. Поддержка транзакций совершенно необходима для многих деловых приложений, когда недопустима любая потеря данных или их несогласованность. Две фирмы - IBM и Transarc - предлагают дополнительные средства, работающие над DCE и обеспечивающие обработку транзакций. Что же касается объектно-ориентированных свойств DCE, то OSF собирается снабдить этот пакет средствами, совместимыми с объектно-ориентированной архитектурой CORBA и работающими над инфраструктурой DCE.

Семейство операционных систем UNIX

История и общая характеристика семейства операционных систем UNIX

UNIX имеет долгую и интересную историю. Начавшись как несерьезный и почти "игрушечный" проект молодых исследователей, UNIX стал многомиллионной индустрией, включив в свою орбиту университеты, многонациональные корпорации, правительства и международные организации стандартизации.

UNIX зародился в лаборатории Bell Labs фирмы AT&T более 20 лет назад. В то время Bell Labs занималась разработкой многопользовательской системы разделения времени MULTICS (Multiplexed Information and Computing Service) совместно с MIT и General Electric, но эта система потерпела неудачу, отчасти из-за слишком амбициозных целей, не соответствовавших уровню компьютеров того времени, а отчасти и из-за того, что она разрабатывалась на языке PL/1, а компилятор PL/1 задерживался и вообще плохо работал после своего запоздалого появления. Поэтому Bell Labs вообще отказалась от участия в проекте MULTICS, что дало возможность одному из ее исследователей, Кену Томпсону, заняться поисковой работой в направлении улучшения операционной среды Bell Labs. Томпсон, а также сотрудник Bell Labs Денис Ритчи и некоторые другие разрабатывали новую файловую систему, многие черты которой вели свое происхождение от MULTICS. Для проверки новой файловой системы Томпсон написал ядро ОС и некоторые программы для компьютера GE-645, который работал под управлением мультипрограммной системы разделения времени GECOS. У Кена Томпсона была написанная им еще во времена работы над MULTICS игра "Space Travel" - "Космическое путешествие". Он запускал ее на

компьютере GE-645, но она работала на нем не очень хорошо из-за невысокой эффективности разделения времени. Кроме этого, машинное время GE-645 стоило слишком дорого. В результате Томпсон и Ритчи решили перенести игру на стоящую в углу без дела машину PDP-7 фирмы DEC, имеющую 4096 18-битных слов, телетайп и хороший графический дисплей. Но у PDP-7 было неважное программное обеспечение, и, закончив перенос игры, Томпсон решил реализовать на PDP-7 ту файловую систему, над который он работал на GE-645. Из этой работы и возникла первая версия UNIX, хотя она и не имела в то время никакого названия. Но она уже включала характерную для UNIX файловую систему, основанную на индексных дескрипторах inode, имела подсистему управления процессами и памятью, а также позволяла двум пользователям работать в режиме разделения времени. Система была написана на ассемблере. Имя UNIX (Uniplex Information and Computing Services) было дано ей еще одним сотрудником Bell Labs, Брайаном Керниганом, который первоначально назвал ее UNICS, подчеркивая ее отличие от многопользовательской MULTICS. Вскоре UNICS начали называть UNIX.

Первыми пользователями UNIX'a стали сотрудники отдела патентов Bell Labs, которые нашли ее удобной средой для создания текстов.

Большое влияние на судьбу UNIX оказала перепись ее на языке высокого уровня C, разработанного Денисом Ритчи специально для этих целей. Это произошло в 1973 году, UNIX насчитывал к этому времени уже 25 инсталляций, и в Bell Labs была создана специальная группа поддержки UNIX.

Широкое распространение UNIX получил с 1974 года, после описания этой системы Томпсоном и Ритчи в компьютерном журнале CACM. UNIX получил широкое распространение в университетах, так как для них он поставлялся бесплатно вместе с исходными кодами на C. Широкое распространение эффективных C-компиляторов сделало UNIX уникальной для того времени ОС из-за возможности переноса на различные компьютеры. Университеты внесли значительный вклад в улучшение UNIX и дальнейшую его популяризацию. Еще одним шагом на пути получения признания UNIX как стандартизованной среды стала разработка Денисом Ритчи библиотеки ввода-вывода stdio. Благодаря использованию этой библиотеки для компилятора C, программы для UNIX стали легко переносимыми.

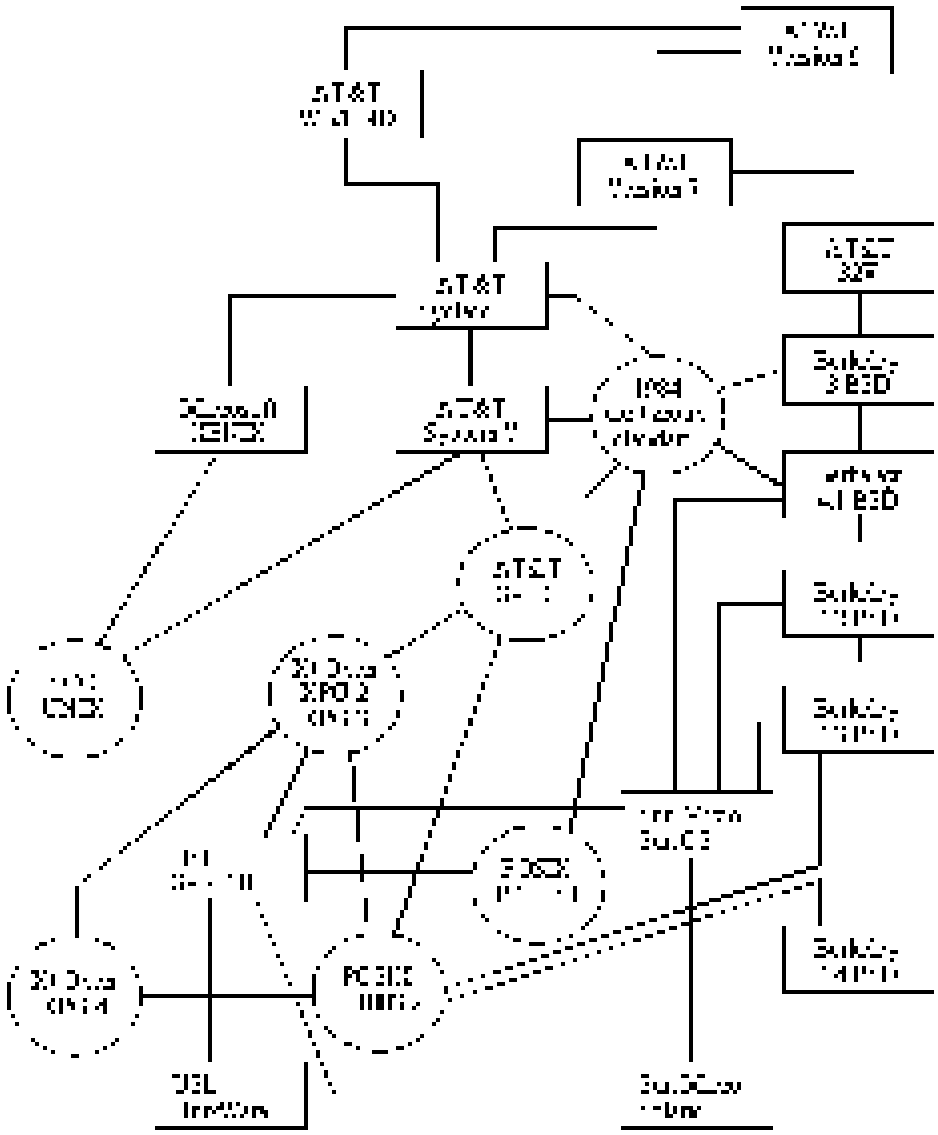


Рис. 5.1. История развития UNIX

Широкое распространение UNIX породило проблему несовместимости его многочисленных версий. Очевидно, что для пользователя весьма неприятен тот факт, что пакет, купленный для одной версии UNIX, отказывается работать на другой версии UNIX. Периодически делались и делаются попытки стандартизации UNIX, но они пока имели ограниченный успех. Процесс сближения различных версий UNIX и их расхождения носит циклический характер. Перед лицом новой угрозы со стороны какой-либо другой операционной системы различные производители UNIX-версий сближают свои продукты, но затем конкурентная борьба вынуждает их делать оригинальные улучшения и версии снова расходятся. В этом процессе есть и положительная сторона - появление новых идей и средств, улучшающих как UNIX, так и многие другие операционные системы, перенявшие у него за долгие годы его существования много полезного.

На рисунке 5.1 показана упрощенная картина развития UNIX, которая учитывает преемственность различных версий и влияние на них принимаемых стандартов. Наибольшее распространение получили две весьма несовместимые линии версий

UNIX: линия AT&T - UNIX System V, и линия университета Berkeley-BSD. Многие фирмы на основе этих версий разработали и поддерживают свои версии UNIX: SunOS и Solaris фирмы Sun Microsystems, UX фирмы Hewlett-Packard, XENIX фирмы Microsoft, AIX фирмы IBM, UnixWare фирмы Novell (проданный теперь компании SCO), и список этот можно еще долго продолжать.

Наибольшее влияние на унификацию версий UNIX оказали такие стандарты как SVID фирмы AT&T, POSIX, созданный под эгидой IEEE, и XPG4 консорциума X/Open. В этих стандартах сформулированы требования к интерфейсу между приложениями и ОС, что дает возможность приложениям успешно работать под управлением различных версий UNIX.

Независимо от версии, общими для UNIX чертами являются:

- многопользовательский режим со средствами защиты данных от несанкционированного доступа,
- реализация мультипрограммной обработки в режиме разделения времени, основанная на использовании алгоритмов вытесняющей многозадачности (preemptive multitasking),
- использование механизмов виртуальной памяти и свопинга для повышения уровня мультипрограммирования,
- унификация операций ввода-вывода на основе расширенного использования понятия "файл",
- иерархическая файловая система, образующая единое дерево каталогов независимо от количества физических устройств, используемых для размещения файлов,
- переносимость системы за счет написания ее основной части на языке С,
- разнообразные средства взаимодействия процессов, в том числе и через сеть,
- кэширование диска для уменьшения среднего времени доступа к файлам.

Далее мы подробно остановимся на основных концепциях версии UNIX System V Release 4, которая вобрала в себя лучшие черты линий UNIX System V и UNIX BSD.

Версия UNIX System V Release 4 - это незаконченная коммерческая версия операционной системы, т.к. в ее кодах отсутствуют многие системные утилиты, необходимые для успешной эксплуатации ОС, например утилиты администрирования или менеджер графического интерфейса. Версия SVR4 является скорее стандартной реализацией кода ядра, вобравшая в себя наиболее популярные и эффективные решения из различных версий ядра UNIX, такие как виртуальная файловая система VFS, отображаемые в память файлы и т.п. Код SVR4 (частично доработанный) лег в основу многих современных коммерческих версий UNIX, таких как HP-UX, Solaris, AIX и т.д.

Концепции UNIX System V Release 4

Управление процессами

Образ, дескриптор, контекст процесса

В основе UNIX лежит концепция процесса - единицы управления и единицы потребления ресурсов. Процесс представляет собой программу в состоянии выполнения, причем в UNIX в рамках одного процесса не могут выполняться никакие параллельные действия.

Каждый процесс работает в своем виртуальном адресном пространстве. Совокупность участков физической памяти, отображаемых на виртуальные адреса процесса, называется образом процесса.

При управлении процессами операционная система использует два основных типа информационных структур: дескриптор процесса (структура proc) и контекст процесса (структура user).

Дескриптор процесса содержит такую информацию о процессе, которая необходима ядру в течение всего жизненного цикла процесса, независимо от того, находится ли он в активном или пассивном состоянии, находится ли образ процесса в оперативной памяти или выгружен на диск. Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов. В дескрипторе прямо или косвенно (через указатели на связанные с ним структуры) содержится информация о состоянии процесса, расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, а также его итоговое значение - глобальный приоритет, идентификатор пользователя, создавшего процесс, информация о родственных процессах, о событиях, осуществления которых ожидает данный процесс и некоторая другая информация.

Контекст процесса содержит менее оперативную, но более объемную часть информации о процессе, необходимую для возобновления выполнения процесса с прерванного места: содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, информацию о всех открытых данным процессом файлов и незавершенных операциях ввода-вывода (указатели на структуры file) и другие данные, характеризующие состояние вычислительной среды в момент прерывания. Контекст, так же как и дескриптор процесса, доступен только программам ядра, то есть находится в виртуальном адресном пространстве операционной системы, однако он хранится не в области ядра, а непосредственно примыкает к образу процесса и перемещается вместе с ним, если это необходимо, из оперативной памяти на диск. В UNIX для процессов предусмотрены два режима выполнения: привилегированный режим - "система" и обычный режим - "пользователь". В режиме "пользователь" запрещено выполнение действий, связанных с управлением ресурсами системы, в частности,

корректировка системных таблиц, управление внешними устройствами, маскирование прерываний, обработка прерываний. В режиме "система" выполняются программы ядра, а в режиме "пользователь" - оболочка и прикладные программы. При необходимости выполнить привилегированные действия пользовательский процесс обращается с запросом к ядру в форме так называемого системного вызова. В результате системного вызова управление передается соответствующей программе ядра. С момента начала выполнения системного вызова процесс считается системным. Таким образом, один и тот же процесс может находиться в пользовательской и системной фазах. Эти фазы никогда не выполняются одновременно.

В данных версиях UNIX процесс, работающий в режиме системы, не мог быть вытеснен другим процессом. Из-за этого организация ядра, которое составляет привилегированную общую часть всех процессов, упрощалась, т.к. все функции ядра не были реентерабельными. Однако, при этом реактивность системы страдала - любой процесс, даже низкоприоритетный, войдя в системную фазу, мог оставаться в ней сколь угодно долго. Из-за этого свойства UNIX не мог использоваться в качестве ОС реального времени. В более поздних версиях, и в SVR4 в том числе, организация ядра усложнилась и процесс можно вытеснить и в системной фазе, но не в произвольный момент времени, а только в определенные периоды его работы, когда процесс сам разрешает это сделать установкой специального сигнала.

В SVR4 имеется несколько процессов, которые не имеют пользовательской фазы, например, процесс pageout, организующий выталкивание страниц на диск.

Порождение процессов

Порождение процессов в системе UNIX происходит следующим образом. При создании процесса строится образ порожденного процесса, являющийся точной копией образа породившего процесса. Сегмент данных и сегмент стека отца действительно копируются на новое место, образуя сегменты данных и стека сына. Процедурный сегмент копируется только тогда, когда он не является разделяемым. В противном случае сын становится еще одним процессом, разделяющим данный процедурный сегмент.

После выполнения системного вызова fork оба процесса продолжают выполнение с одной и той же точки. Чтобы процесс мог опознать, является ли он отцом или сыном, системный вызов fork возвращает в качестве своего значения в породивший процесс идентификатор порожденного процесса, а в порожденный процесс NULL. Типичное разветвление на языке С записывается так:

```
if( fork() )  {  действия отца }



else   { действия сына }
```

Идентификатор сына может быть присвоен переменной, входящей в контекст процесса-отца. Так как контекст процесса наследуется его потомками, то дети могут узнать идентификаторы своих старших братьев, так образом сумма знаний наследуется при порождении и может быть распространена между родственными процессами. Наследуются все характеристики процесса, содержащиеся в контексте.

На независимости идентификатора процесса от выполняемой процессом программы построен механизм, позволяющий процессу придти к выполнению другой программы с помощью системного вызова exec.

Таким образом в UNIX порождение нового процесса происходит в два этапа - сначала создается копия процесса-родителя, то есть дублируется дескриптор, контекст и образ процесса. Затем у нового процесса производится замена кодового сегмента на заданный.

Вновь созданному процессу операционная система присваивает целочисленный идентификатор, уникальный за весь период функционирования системы.

Планирование процессов

В системе UNIX System V Release 4 реализована вытесняющая многозадачность, основанная на использовании приоритетов и квантования.

Все процессы разбиты на несколько групп, называемых классами приоритетов. Каждая группа имеет свои характеристики планирования процессов.

Созданный процесс наследует характеристики планирования процесса-родителя, которые включают класс приоритета и величину приоритета в этом классе. Процесс остается в данном классе до тех пор, пока не будет выполнен системный вызов, изменяющий его класс.

В UNIX System V Release 4 возможно включение новых классов приоритетов при инсталляции системы. В настоящее время имеется три приоритетных класса: класс реального времени, класс системных процессов и класс процессов разделения времени. В отличие от ранних версий UNIX приоритетность (привилегии) процесса тем выше, чем больше число, выражающее приоритет. На рисунке 5.2 показаны диапазоны изменения приоритетов для разных классов. Значения приоритетов определяются для разных классов по разному.

Процессы системного класса используют стратегию фиксированных приоритетов. Системный класс зарезервирован для процессов ядра. Уровень приоритета процессу назначается ядром и никогда не изменяется. Заметим, что пользовательский процесс, перешедший в системную фазу, не переходит при этом в системный класс приоритетов.

Процессы реального времени также используют стратегию фиксированных приоритетов, но пользователь может их изменять. Так как при наличии готовых к выполнению процессов реального времени другие процессы не рассматриваются, то процессы реального времени надо тщательно проектировать, чтобы они не захватывали процессор на слишком долгое время. Характеристики планирования процессов реального времени включают две величины: уровень глобального приоритета и квант времени. Для каждого уровня приоритета имеется по умолчанию своя величина кванта времени. Процессу разрешается захватывать процессор на указанный квант времени, а по его истечении планировщик снимает процесс с выполнения.

Приоритетный класс	**Выбор планировщика**	**Глобальное значение приоритета**
Реальное время (real time)	первый	159 . . . 100
Системные процессы (system)	. . . .	99 60
Процессы разделения времени (time-shared)	. . . . последний	59 0
Возможно добавление новых классов		

Рис. 5.2. Приоритетные классы процессов

Процессы разделения времени были до появления UNIX System V Release 4 единственным классом процессов, и по умолчанию UNIX System V Release 4 назначает новому процессу этот класс. Состав класса процессов разделения времени наиболее неопределенный и часто меняющийся, в отличие от системных

процессов и процессов реального времени. Для справедливого распределения времени процессора между процессами, в этом классе используется стратегия динамических приоритетов, которая адаптируется к операционным характеристикам процесса.

Величина приоритета, назначаемого процессам разделения времени, вычисляется пропорционально значениям двух составляющих: пользовательской части и системной части. Пользовательская часть приоритета может быть изменена суперпользователем и владельцем процесса, но в последнем случае только в сторону его снижения.

Системная составляющая позволяет планировщику управлять процессами в зависимости от того, как долго они используют процессор, не уходя в состояние ожидания. Тем процессам, которые потребляют большие периоды времени без ухода в состояние ожидания, приоритет снижается, а тем процессам, которые часто уходят в состояние ожидания после короткого периода использования процессора, приоритет повышается. Таким образом, процессам, ведущим себя не по-джентльменски, дается низкий приоритет, что означает, что они реже выбираются на выполнение. Но процессам с низким приоритетом даются большие кванты времени, чем процессам с высокими приоритетами. Таким образом, хотя низкоприоритетный процесс и не работает так часто, как высокоприоритетный, но зато, когда он наконец выбирается на выполнение, ему отводится больше времени.

Планировщик использует следующие характеристики для процессов разделения времени:

ts_globpri	содержит величину глобального приоритета;
ts_quantum	определяет количество тиков системных часов, которые отводятся процессу до его вытеснения;
ts_tqexp	системная часть приоритета, назначаемая процессу при истечении его кванта времени;
ts_slpret	системная составляющая приоритета, назначаемая процессу после выхода его из состояния ожидания; ожидающим процессам дается высокий приоритет, так что они быстро получают доступ к процессору после освобождения ресурса;
ts_maxwaite	максимальное число секунд, которое разрешается потреблять процессу; если этот квант времени истекает до кванта ts_quantum, то, следовательно, считается, что процесс ведет себя по-джентльменски, и ему назначается более высокий приоритет;
ts_lwait	величина системной части приоритета, назначаемая процессу, если истекает ts_maxwait секунд.

Для процессов разделения времени в дескрипторе процесса proc имеется указатель на структуру, специфическую для данного класса процесса. Эта структура состоит из полей, используемых для вычисления глобального приоритета:

ts_timeleft	число тиков, остающихся в кванте процесса;
ts_cpupri	системная часть приоритета процесса;
ts_uprilim, ts_upri	верхний предел и текущее значение пользовательской части приоритета. Эти две переменные могут модифицироваться пользователем;
ts_nice	используется для обратной совместимости с системным вызовом nice. Она содержит текущее значение величины nice, которая влияет на результирующую величину приоритета. Чем выше эта величина, тем меньше приоритет.

В версии SVR4 нет поддержки многонитевой (multithreading) организации процессов на уровне ядра, хотя и есть два системных вызова для организации нитей в пользовательском режиме. Во многих коммерческих реализациях UNIX, базирующихся на кодах SVR4, в ядро включена поддержка нитей за счет собственной модификации исходных текстов SVR4.

Файловые системы UNIX System V Release 4

В UNIX System V Release 4 реализован механизм виртуальной файловой системы VFS (Virtual File System), который позволяет ядру системы одновременно поддерживать несколько различных типов файловых систем. Механизм VFS поддерживает для ядра некоторое абстрактное представление о файловой системе, скрывая от него конкретные особенности каждой файловой системы.

Типы файловых систем, поддерживаемых в UNIX System V Release 4:

- *s5* - традиционная файловая система UNIX System V, поддерживаемая в ранних версиях UNIX System V от AT&T;
- *ufs* - файловая система, используемая по умолчанию в UNIX System V Release 4, которая ведет происхождение от файловой системы SunOS, которая в свою очередь, происходит от файловой системы Berkeley Fast File System (FFS);
- *nfs* - адаптация известной файловой системы NFS фирмы Sun Microsystems, которая позволяет разделять файлы и каталоги в гетерогенных сетях;
- *rfs* - файловая система Remote File Sharing из UNIX System V Release 3. По функциональным возможностям близка к NFS, но требует на каждом компьютере установки UNIX System V Release 3 или более поздних версий этой ОС;

- *Veritas* - отказоустойчивая файловая система с транзакционным механизмом операций;
- *specfs* - этот новый тип файловой системы обеспечивает единый интерфейс ко всем специальным файлам, описываемым в каталоге /dev;
- *fifofs* - эта новая файловая система использует механизм VFS для реализации файлов FIFO, также известных как конвейеры (pipes), в среде STREAMS;
- *bfs* - загрузочная файловая система. Предназначена для быстрой и простой загрузки и поэтому представляет собой очень простую плоскую файловую систему, состоящую из одного каталога;
- */proc* - файловая система этого типа обеспечивает доступ к образу адресного пространства каждого активного процесса системы, обычно используется для отладки и трассировки;
- */dev/fd* - этот тип файловой системы обеспечивает удобный метод ссылок на дескрипторы открытых файлов.

Не во всех коммерческих реализациях поддерживаются все эти файловые системы, отдельные производители могут предоставлять только некоторые из них.

Традиционная файловая система s5

Типы файлов

Файловая система UNIX s5 поддерживает логическую организацию файла в виде последовательности байтов. По функциональному назначению различаются обычные файлы, каталоги и специальные файлы.

Обычные файлы содержат ту информацию, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ, то есть ОС не накладывает никаких ограничений на структуру и характер информации, хранимой в обычных файлах.

Каталог - файл, содержащий служебную информацию файловой системы о группе файлов, входящих в данный каталог. В каталог могут входить обычные, специальные файлы и каталоги более низкого уровня.

Специальный файл - фиктивный файл, ассоциируемый с каким-либо устройством ввода-вывода, используется для унификации механизма доступа к файлам и внешним устройствам.

Структура файловой системы

Файловая система s5 имеет иерархическую структуру, в которой уровни создаются за счет каталогов, содержащих информацию о файлах более низкого уровня. Каталог самого верхнего уровня называется корневым и имеет имя root.

Иерархическая структура удобна для многопользовательской работы: каждый пользователь локализуется в своем каталоге или поддереве каталогов, и вместе с тем все файлы в системе логически связаны. Корневой каталог файловой системы всегда располагается на системном устройстве (диск, имеющий такой признак). Однако это не означает, что и все остальные файлы могут содержаться только на нем. Для связи иерархий файлов, расположенных на разных носителях, применяется монтирование файловой системы, выполняемое системным вызовом mount. Пусть имеются две файловые системы, расположенные на разных дисках (рисунок 5.3). Операция монтирования заключается в следующем: в корневой файловой системе выбирается некоторый существующий каталог, содержащий один пустой файл, в данном примере каталог man, содержащий файл dummy. После выполнения монтирования выбранный каталог man становится корневым каталогом второй файловой системы. Через этот каталог смонтированная файловая система подсоединяется как поддерево к общему дереву (рисунок 5.4). При этом нет логической разницы между основной и монтированными файловыми системами.

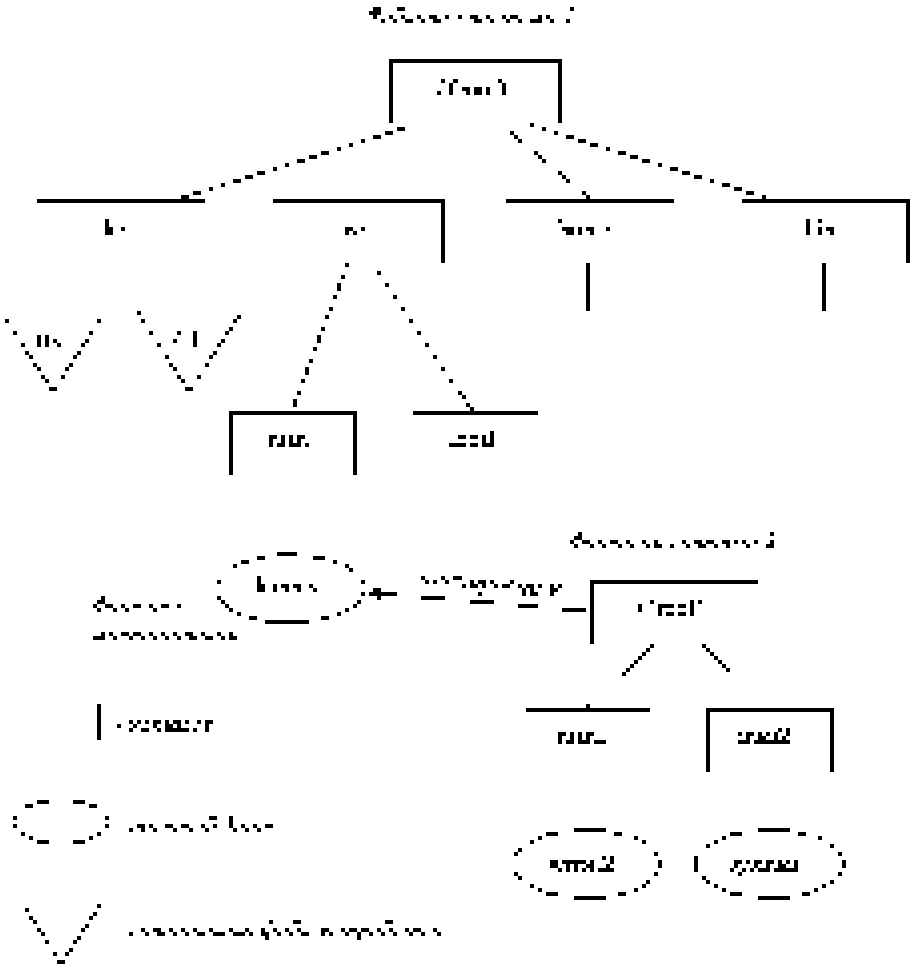


Рис. 5.3. Традиционная файловая система s5

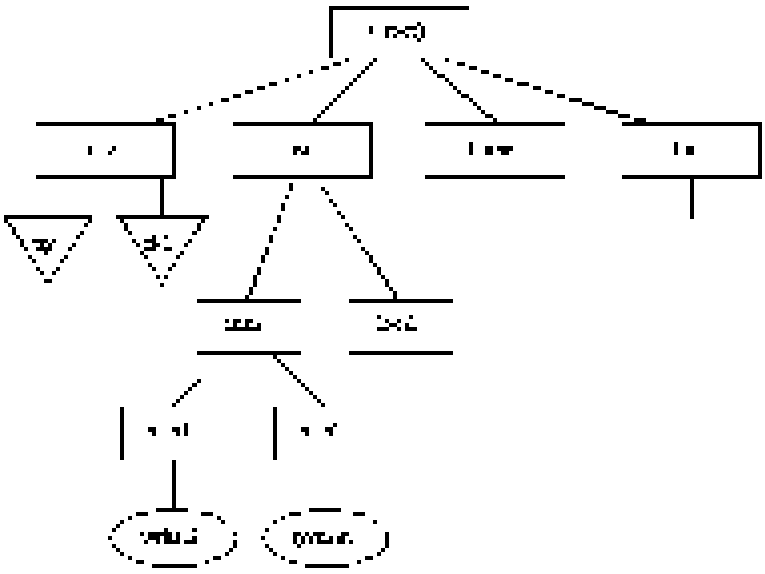


Рис. 5.4. Общая файловая система (после монтирования)

Имена файлов

В UNIX для файла существует три типа имени - краткое, полное и относительное. Краткое имя идентифицирует файл в пределах одного каталога. Оно может состоять не более чем из 14 символов и содержать так называемый суффикс, отделяемый точкой. Полное имя однозначно определяет файл. Оно состоит из цепочки имен каталогов, через которые проходит маршрут от корневого каталога до данного файла. Имена каталогов разделяются символами "/", при этом имя корневого каталога не указывается, например, /mnt/rk2/test.c, где mnt и rk2 - имена каталогов, а test.c - имя файла. Каждому полному имени в ОС соответствует только один файл, однако файл может иметь несколько различных имен, так как ссылки на один и тот же файл могут содержаться в разных каталогах (жесткие связи). Относительное имя файла связано с понятием "текущий каталог", то есть каталог, имя которого задавать не нужно, так как оно подразумевается по умолчанию. Имя файла относительно текущего каталога называется относительным. Оно представляет собой цепочку имен каталогов, через которые проходит маршрут от текущего каталога до данного файла. Относительное имя в отличие от полного не начинается с символа "/". Так, если в предыдущем примере принять за текущий каталог /mnt, то относительное имя файла test.c будет rk2/test.c.

Привилегии доступа

В UNIX s5 все пользователи по отношению к данному файлу делятся на три категории: владелец, член группы владельца и все остальные. Группа - это пользователи, которые объединены по какому-либо признаку, например, по принадлежности к одной разработке. Кроме этого, в системе существует суперпользователь, обладающий абсолютными правами по доступу ко всем файлам системы.

Определены также три вида доступа к файлу - чтение, запись и выполнение. Привилегии доступа к каждому файлу определены для каждой из трех категорий пользователей и для каждой из трех операций доступа. Начальные значения прав доступа к файлу устанавливаются при его создании операционной системой и могут изменяться его владельцем или суперпользователем.

Физическая организация файла

В общем случае файл может располагаться в несмежных блоках дисковой памяти. Логическая последовательность блоков в файле задается набором из 13 элементов. Первые 10 элементов предназначаются для непосредственного указания номеров первых 10 блоков файла. Если размер файла превышает 10 блоков, то в 11 элементе указывается номер блока, в котором содержится список следующих 128 блоков файла. Если файл имеет размер более, чем 10+128 блоков, то используется 12-й элемент, содержащий номер блока, в котором указываются номера 128 блоков, каждый из которых может содержать еще по 128 номеров блоков файла. Таким образом, 12-й элемент используется для двухуровневой косвенной адресации. В случае, если файл больше, чем $10+128+128^2$ блоков, то используется 13 элемент для трехуровневой косвенной адресации. При таком способе адресации предельный размер файла составляет 2 113 674 блока. Традиционная файловая система s5 поддерживает размеры блоков 512, 1024 или 2048 байт.

Структуры индексных дескрипторов и каталогов

Вся необходимая операционной системе информация о файле, кроме его символьного имени, хранится в специальной системной таблице, называемой индексным дескриптором (inode) файла. Индексные дескрипторы всех файлов имеют одинаковый размер - 64 байта и содержат данные о типе файла, о физическом расположении файла на диске (описанные выше 13 элементов), размере в байтах, о дате создания, последней модификации и последнего обращения к файлу, о привилегиях доступа и некоторую другую информацию. Индексные дескрипторы пронумерованы и хранятся в специальной области файловой системы. Номер индексного дескриптора является уникальным именем файла. Соответствие между полными символьными именами файлов и их уникальными именами устанавливается с помощью иерархии каталогов.

Каталог представляет собой совокупность записей обо всех файлах и каталогах, входящих в него. Каждая запись состоит из 16 байтов, 14 байтов отводится под короткое символьное имя файла или каталога, а 2 байта - под номер индексного дескриптора этого файла. В каталоге файловой системы s5 непосредственно не указываются характеристики файлов. Такая организация файловой системы позволяет с меньшими затратами перестраивать систему каталогов. Например, при включении или исключении файла из каталога идет манипулирование меньшими объемами информации. Кроме того, при включении одного и того же файла в разные каталоги не нужно иметь несколько копий как характеристик, так и самих файлов. С этой целью в индексном дескрипторе ведется учет ссылок на

этот файл из всех каталогов. Как только число ссылок становится равным нулю, индексный дескриптор данного файла уничтожается.

Расположение файловой системы s5 на диске показано на рисунке 5.5. Все дисковое пространство, отведенное под файловую систему, делится на четыре области:

- *загрузочный блок (boot),* в котором хранится загрузчик операционной системы;
- *суперблок (superblock)* - содержит самую общую информацию о файловой системе: размер файловой системы, размер области индексных дескрипторов, число индексных дескрипторов, список свободных блоков и список свободных индексных дескрипторов, а также другую административную информацию;
- *область индексных дескрипторов*, порядок расположения индексных дескрипторов в которой соответствует их номерам;
- *область данных*, в которой расположены как обычные файлы, так и файлы-каталоги. Специальные файлы представлены в файловой системе только записями в соответствующих каталогах и индексными дескрипторами специального формата, но места в области данных не занимают.

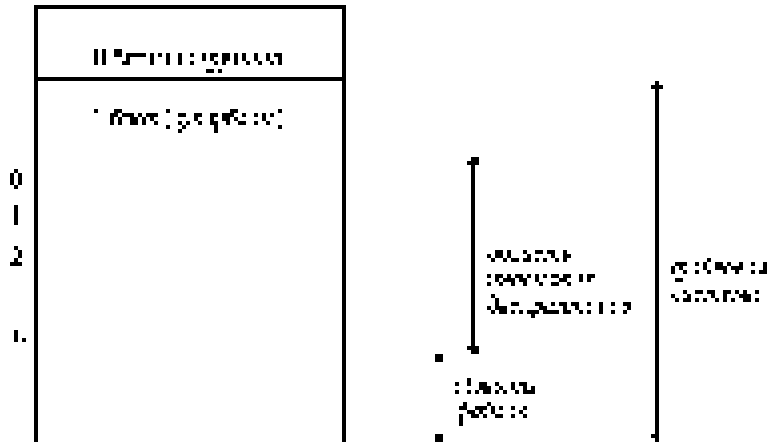


Рис. 5.5. Расположение файловой системы s5 на диске

Доступ к файлу осуществляется путем последовательного просмотра всей цепочки каталогов, входящих в полное имя файла, и соответствующих им индексных дескрипторов. Поиск завершается после получения всех характеристик из индексного дескриптора заданного файла. Эта процедура требует в общем случае нескольких обращений к диску, пропорционально числу составляющих в полном имени файла. Для уменьшения среднего времени доступа к файлу его дескриптор копируется в специальную системную область памяти. Копирование индексного дескриптора входит в процедуру открытия файла. При открытии файла ядро выполняет следующие действия:

1. Проверяет, существует ли файл; если не существует, то можно ли его создать. Если существует, то разрешен ли к нему доступ требуемого вида.
2. Копирует индексный дескриптор с диска в оперативную память; если с указанным файлом уже ведется работа, то новая копия индексного дескриптора не создается.
3. Создает в области ядра структуру, предназначенную для отображения текущего состояния операции обмена данными с указанным файлом. Эта структура, называемая file, содержит данные о типе операции (чтение, запись или чтение и запись), о числе считанных или записанных байтов, указатель на байт файла, с которым проводится операция.
4. Делает отметку в контексте процесса, выдавшего системный вызов на операцию с данным файлом.

Виртуальная файловая система VFS

В UNIX System V Release 3 был реализован механизм переключения файловых систем (File System Switch, FSS), позволяющий операционной системе поддерживать различные типы файловых систем. В соответствии с этим подходом информация о файловой системе и файлах разбивается на две части - зависимую от типа файловой системы и не зависимую. FSS обеспечивает интерфейс между ядром и файловой системой, транслируя запросы ядра в операции, зависящие от типа файловой системы. При этом ядро имеет представление только о независимой части файловой системы. Однако большее распространение получила схема, реализованная фирмой Sun Microsystems, использующая аналогичный подход. Эта схема называется переключателем виртуальной файловой системы - Virtual File System (VFS). В UNIX System V Release 4 реализован именно этот механизм.

VFS не ориентируется на какую-либо конкретную файловую систему, механизмы реализации файловой системы полностью скрыты как от пользователя, так и от приложений. В ОС нет системных вызовов, предназначенных для работы со специфическими типами файловой системы, а имеются абстрактные вызовы типа open, read, write и другие, которые имеют содержательное описание, обобщающее некоторым образом содержание этих операций в наиболее популярных типах файловых систем (например, s5, ufs, nfs и т.п.). VFS также предоставляет ядру возможность оперирования файловой системой, как с единым целым: операции монтирования и демонтирования, а также операции получения общих характеристик конкретной файловой системы (размера блока, количества свободных и занятых блоков и т.п.) в единой форме. Если конкретный тип файловой системы не поддерживает какую-то абстрактную операцию VFS, то файловая система должна вернуть ядру код возврата, извещающий об этом факте.

В VFS вся информация о файлах разделена на две части - не зависящую от типа файловой системы, которая хранится в специальной структуре ядра - структуре vnode, и зависящую от типа файловой системы - структура inode, формат которой на уровне VFS не определен, а используется только ссылка на нее в структуре

vnode. Имя inode не означает, что эта структура совпадает со структурой индексного дескриптора inode файловой системы s5. Это имя используется для обозначения зависящей от типа файловой системы информации о файле, как дань традиции.

Виртуальная файловая система VFS поддерживает следующие типы файлов:

- обычные файлы,
- каталоги,
- специальные файлы,
- именованные конвейеры,
- символьные связи.

Содержательное описание обычных файлов, каталогов и специальных файлов и связей не отличается от их описания в файловой системе s5.

Символьные связи

Версия UNIX System V Release 4 вводит новый тип связи - мягкая связь, называемая символьной связью и реализуемая с помощью системного вызова symlink. Символьная связь - это файл данных, содержащий имя файла, с которым предполагается установить связь. Слово "предполагается" использовано потому, что символьная связь может быть создана даже с несуществующим файлом. При создании символьной связи образуется как новый вход в каталоге, так и новый индексный дескриптор inode. Кроме этого, резервируется отдельный блок данных для хранения полного имени файла, на который он ссылается.

Многие системные вызовы пользуются файлом символьных связей для поиска реального файла. Связанные файлы не обязательно располагаются в той же файловой системе.

Имеются три системных вызова, которые имеют отношение к символьным связям:

- readlink - чтение полного имени файла или каталога, на который ссылается символьная связь. Эта информация хранится в блоке, связанном с символьной связью.
- lstat - аналогичен системному вызову stat, но используется для получения информации о самой связи.
- lchown - аналогичен системному вызову chown, но используется для изменения владельца самой символьной связи.

Именованные конвейеры

Конвейер - это средство обмена данными между процессами. Конвейер буферизует данные, поступающие на его вход, таким образом, что процесс, читающий данные на его выходе, получает их в порядке "первый пришел -

первый вышел" (FIFO). В ранних версиях UNIX для обмена данными между процессами использовались неименованные конвейеры - pipes, которые представляли собой очереди байт в оперативной памяти. Однако, из-за отсутствия имен, такие конвейеры могли использоваться только для передачи данных между родственными процессами, получившими указатель на конвейер в результате копирования сегмента данных из адресного пространства процесса-прародителя. Именованные конвейеры позволяют обмениваться данными произвольной паре процессов, т.к. каждому такому конвейеру соответствует файл на диске. Никакие данные не связываются с файлом-конвейером, но все равно в каталоге содержится запись о нем, и он имеет индексный дескриптор. В UNIX System V Release 4 конвейеры реализуются с использованием коммуникационных модулей STREAMS.

Файлы, отображенные в памяти

Побочным продуктом новой архитектуры виртуальной памяти UNIX System V Release 4 является возможность отображать содержимое файла (или устройства) в виде последовательности байтов в виртуальное адресное пространство процесса. Это упрощает процедуру доступа процесса к данным.

Реализация файловой системы VFS

UNIX System V Release 4 имеет массив структур vfssw [ ], каждая из которых описывает файловую систему конкретного типа, которая может быть установлена в системе. Структура vfssw состоит из четырех полей:

- символьного имени файловой системы;
- указателя на функцию инициализации файловой системы;
- указателя на структуру, описывающую функции, реализующие абстрактные операции VFS в данной конкретной файловой системе;
- флаги, которые не используются в описываемой версии UNIX.

Пример инициализированного массива структур vfssw:

```
struct vfssw vfssw[] = {



{0, 0 , 0 ,0 }, - нулевой элемент не используется



{"spec",  specint,  &spec_vfsops,    0},  - SPEC



{"vxfs",  vx_init,  &vx_vfsops,      0},  - Veritas


```

```
{"cdfs",  cdfsinit, &cdfs_vfsops,    0},  - CD ROM

{"ufs",   ufsinit,  &ufs_vfsops,     0},  - UFS

{"s5",    vx_init,  &vx_vfsops,      0},  - S5

{"fifo",  fifoinit, &fifovfsops,     0},  - FIFO

{"dos",   dosinit,  &dos_vfsops,     0},  - MS-DOS
```

Функции инициализации файловых систем вызываются во время инициализации операционной системы. Эти функции ответственны за создание внутренней среды файловой системы каждого типа.

Структура vfsops, описывающая операции, которые выполняются над файловой системой, состоит из 7 полей, так как в UNIX System V Release 4 предусмотрено 7 абстрактных операций над файловой системой:

VFS_MOUNT	монтирование файловой системы,
VFS_UNMOUNT	размонтирование файловой системы,
VFS_ROOT	получение vnode для корня файловой системы,
VFS_STATVFS	получение статистики файловой системы,
VFS_SYNC	выталкивание буферов файловой системы на диск,
VFS_VGET	получение vnode по номеру дескриптора файла,
VFS_MOUNTROOT	монтирование корневой файловой системы.

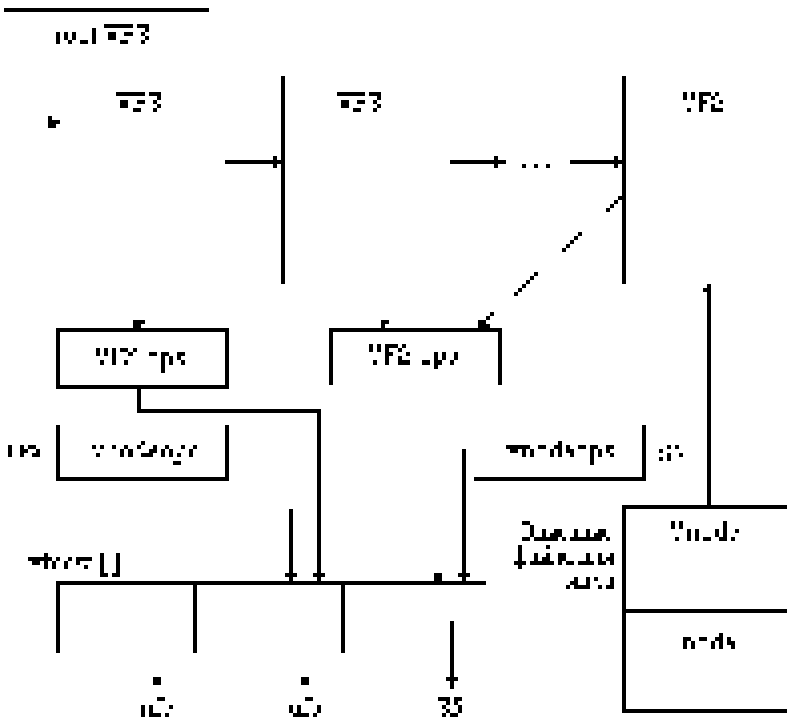


Рис. 5.6. Монтирование файловых систем в VFS

Операция VFS_MOUNT выполняет традиционное для UNIX монтирование файловой системы на указанный каталог уже смонтированной файловой системы для образования общего дерева, а операция VFS_UNMOUNT отменяет монтирование. Операция VFS_ROOT используется при разборе полного имени файла, когда встречается дескриптор vnode, который связан со смонтированной на него файловой системой. Операция VFS_ROOT помогает найти vnode, который является корнем смонтированной файловой системы. Операция VFS_STATVFS позволяет получить независимую от типа файловой системы информацию о размере блока файловой системы, о количестве блоков и количестве свободных блоков в единицах этого размера, о максимальной длине имени файла и т.п. Операция VFS_SYNC выталкивает содержимое буферов диска из оперативной памяти на диск. Операция VFS_MOUNTROOT позволяет смонтировать корневую файловую систему, то есть систему, содержащую корневой каталог / общего дерева. Для указания того, какая файловая система будет монтироваться как корневая, в UNIX System V Release 4 используется переменная rootfstype, содержащая символьное имя корневой файловой системы, например "ufs".

Таким образом, в UNIX System V Release 4 одновременно в единое дерево могут быть смонтированы несколько файловых систем различных типов, поддерживающих операцию монтирования (рисунок 5.6).

VOP_OPEN	- открыть файл
VOP_CLOSE	- закрыть файл

VOP_READ	- читать из файла
VOP_WRITE	- записать в файл
VOP_IOCTL	- управление в/в
VOP_SETFL	- установить флаги статуса
VOP_GETATTR	- получить атрибуты файла
VOP_SETATTR	- установить атрибуты файла
VOP_LOOKUP	- найти vnode по имени файла
VOP_CREATE	- создать файл
VOP_REMOVE	- удалить файл
VOP_LINK	- связать файл
VOP_MAP	- отобразить файл в память

Рис. 5.7. Абстрактные операции над файлами

Кроме операций над файловой системой в целом, для каждого типа файловой системы (s5, ufs и т.д.), установленной в ОС, необходимо описать способ реализации абстрактных операций над файлами, которые допускаются в VFS. Этот способ описывается для каждого типа файловой системы в структуре vnodeops, состав которой приведен на рисунке 5.7. Как видно из состава списка абстрактных операций, они образованы объединением операций, характерных для наиболее популярных файловых систем UNIX. Для того, чтобы обращение к специфическим функциям не зависело от типа файловой системы, для каждой операции в vnodeops определен макрос с общим для всех типов файловых систем именем, например, VOP_OPEN, VOP_CLOSE, VOP_READ и т.п. Эти макросы определены в файле <sys/vnode.h> и соответствуют системным вызовам. Таким образом, в структуре vnodeops скрыты зависящие от типа файловой системы реализации стандартного набора операций над файлами. Даже если файловая система какого-либо конкретного типа не поддерживает определенную операцию над своими файлами, она должна создать соответствующую функцию, которая выполняет некоторый минимум действий: или сразу возвращает успешный код завершения, или возвращает код ошибки. Для анализа и обработки полного имени файла в VFS используется операция VOP_LOOKUP, позволяющая по имени файла найти ссылку на его структуру vnode.

Работа ядра с файлами во многом основана на использовании структуры vnode, поля которой представлены на рисунке 5.8. Структура vnode используется ядром

для связи файла с определенным типом файловой системы через поле v_vfsp и конкретными реализациями файловых операций через поле v_op. Поле v_pages используется для указания на таблицу физических страниц памяти в случае, когда файл отображается в физическую память (этот механизм описан в разделе, описывающем организацию виртуальной памяти). В vnode также содержится тип файла и указатель на зависимую от типа файловой системы часть описания характеристик файла - структуру inode, обычно содержащую адресную информацию о расположении файла на носителе и о правах доступа к файлу. Кроме этого, vnode используется ядром для хранения информации о блокировках (locks), примененных процессами к отдельным областям файла.

Ядро в своих операциях с файлами оперирует для описания области файла парой vnode, offset, которая однозначно определяет файл и смещение в байтах внутри файла.

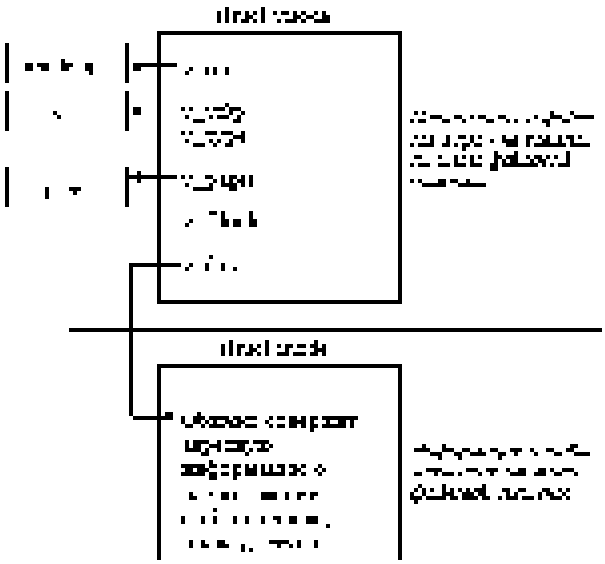


Рис. 5.8. Описатель файла - vnode

При каждом открытии процессом файла ядро создает в системной области памяти новую структуру типа file, которая, как и в случае традиционной файловой системы s5, описывает как открытый файл, так и операции, которые процесс собирается производить с файлом (например, чтение). Структура file содержит такие поля, как:

- flag - определение режима открытия (только для чтения, для чтения и записи и т.п.);
- struct vnode * f_vnode - указатель на структуру vnode (заменивший по сравнению с s5 указатель на inode);
- offset - смещение в файле при операциях чтения/записи;
- struct cred * f_cred - указатель на структуру, содержащую права процесса, открывшего файл (структура находится в дескрипторе процесса);

а также указатели на предыдущую и последующую структуру типа file, связывающие все такие структуры в список.

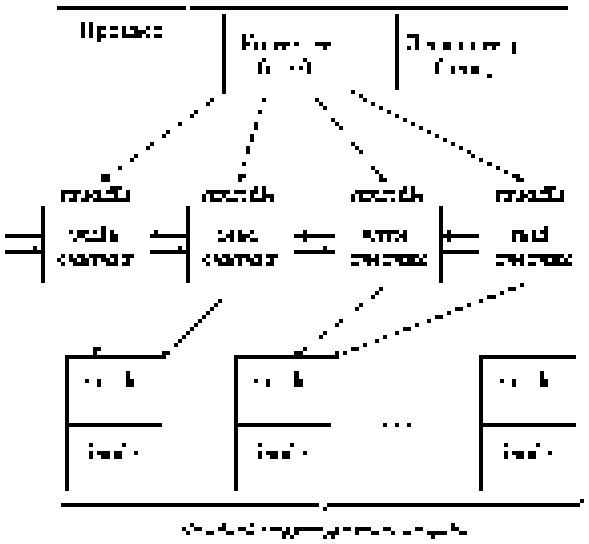


Рис. 5.9. Связь процесса с его файлами

Связь структур процесса с системным списком структур file показан на рисунке 5.9.

В отличие от структур типа file структуры типа vnode заводятся операционной системой для каждого активного (открытого) файла в единственном экземпляре, поэтому структуры file могут ссылаться на одну и ту же структуру vnode.

Структуры vnode не связаны в какой-либо список. Они появляются по требованию в системном пуле памяти и присоединяются к структуре данных, которая инициировала появление этого vnode, с помощью соответствующего указателя. Например, в случае структуры file в ней используется указатель f_vnode на соответствующую структуру vnode, описывающую нужный файл. Аналогично, если файл связан с образом процесса (то есть это файл, содержащий выполняемый модуль), то отображение сегмента памяти, содержащего части этого файла, осуществляется посредством указателя vp (в структуре segvn_data) на vnode этого файла.

Все операции с файлами в UNIX System V Release 4 производятся с помощью связанной с файлом структуры vnode. Когда процесс запрашивает операцию с файлом (например, операцию open), то независимая от типа файловой системы часть ОС передает управление зависимой от типа файловой системы части ОС для выполнения операции. Если зависимая часть обнаруживает, что структуры vnode, описывающей нужный файл, нет в оперативной памяти, то зависимая часть заводит для него новую структуру vnode.

Для ускорения доступа к файлам в UNIX System V Release 4 используется механизм быстрой трансляции имен файлов в соответствующие им ссылки на

структуры vnode. Этот механизм основан на наличии кэша, хранящего максимально 800 записей о именах файлов и указателях vnode.

Сетевая файловая система NFS

Одной из самых известных сетевых файловых систем является Network File System (NFS) фирмы Sun Microsystems. NFS была первоначально создана для UNIX-компьютеров. Сейчас она поддерживает как UNIX, так и другие ОС, включая MS DOS. NFS поддерживает неоднородные системы, например, MS-DOS-клиенты и UNIX-серверы.

Основная идея NFS - позволить произвольному набору пользователей разделять общую файловую систему. Чаще всего все пользователи принадлежат одной локальной сети, но не обязательно. Можно выполнять NFS и на глобальной сети. Каждый NFS-сервер предоставляет один или более своих каталогов для доступа удаленным клиентам. Каталог объявляется доступным со всеми своими подкаталогами. Список каталогов, которые сервер передает, содержится в файле /etc/exports, так что эти каталоги экспортируются сразу автоматически при загрузке сервера. Клиенты получают доступ к экспортируемым каталогам путем монтирования. Многие рабочие станции Sun - бездисковые, но и в этом случае можно монтировать удаленную файловую систему на корневой каталог, при этом вся файловая система целиком располагается на сервере. При выполнении программ почти нет различий, расположен ли файл локально или на удаленном диске. Если два или более клиента одновременно смонтировали один и тот же каталог, то они могут связываться путем разделения файла.

Так как одной из целей NFS является поддержка неоднородных систем с клиентами и серверами, выполняющими различные ОС на различной аппаратуре, то особенно важно, чтобы был хорошо определен интерфейс между клиентами и серверами. Только в этом случае станет возможным написание программного обеспечения клиентской части для новых операционных систем рабочих станций, которое будет правильно работать с существующими серверами. Это цель достигается двумя протоколами.

Первый NFS-протокол управляет монтированием. Клиент может послать полное имя каталога серверу и запросить разрешение на монтирование этого каталога на какое-либо место собственного дерева каталогов. При этом серверу не указывается, в какое место будет монтироваться каталог сервера, так как ему это безразлично. Получив имя, сервер проверяет законность этого запроса и возвращает клиенту описатель файла, содержащий тип файловой системы, диск, номер дескриптора (inode) каталога, информацию безопасности. Операции чтения и записи файлов из монтируемых файловых систем используют этот описатель файла. Монтирование может выполняться автоматически с помощью командных файлов при загрузке. Существует другой вариант автоматического монтирования: при загрузке ОС на рабочей станции удаленная файловая система не монтируется, но при первом открытии удаленного файла ОС посылает запросы

каждому серверу, и, после обнаружения этого файла, монтирует каталог того сервера, на котором этот файл расположен.

Второй NFS-протокол используется для доступа к удаленным файлам и каталогам. Клиенты могут послать запрос серверу для выполнения какого-либо действия над каталогом или операции чтения или записи файла. Кроме того, они могут запросить атрибуты файла, такие как тип, размер, время создания и модификации. Большая часть системных вызовов UNIX поддерживается NFS, за исключением open и close. Исключение open и close не случайно. Вместо операции открытия удаленного файла клиент посылает серверу сообщение, содержащее имя файла, с запросом отыскать его (lookup) и вернуть описатель файла. В отличие от вызова open, вызов lookup не копирует никакую информацию во внутренние системные таблицы сервера. Вызов read содержит описатель того файла, который нужно читать, смещение в уже читаемом файле и количество байтов, которые нужно прочитать. Преимуществом такой схемы является то, что сервер не должен запоминать ничего об открытых файлах. Таким образом, если сервер откажет, а затем будет восстановлен, информация об открытых файлах не потеряется, потому что ее нет. Серверы, подобные этому, не хранящие постоянную информацию об открытых файлах, называются stateless.

В противоположность этому в UNIX System V в удаленной файловой системе RFS требуется, чтобы файл был открыт, прежде чем его можно будет прочитать или записать. После этого сервер создает таблицу, сохраняющую информацию о том, что файл открыт, и о текущем читателе. Недостаток этого подхода состоит в том, что когда сервер отказывает, после его перезапуска все его открытые связи теряются, и программы пользователей портятся. В NFS такого недостатка нет.

К сожалению метод NFS затрудняет блокировку файлов. В UNIX файл может быть открыт и заблокирован так, что другие процессы не имеют к нему доступа. Когда файл закрывается, блокировка снимается. В stateless-системах, подобных NFS, блокирование не может быть связано с открытием файла, так как сервер не знает, какой файл открыт. Следовательно, NFS требует специальных дополнительных средств управления блокированием.

Реализация кодов клиента и сервера в NFS имеет многоуровневую структуру (рисунок 5.10).

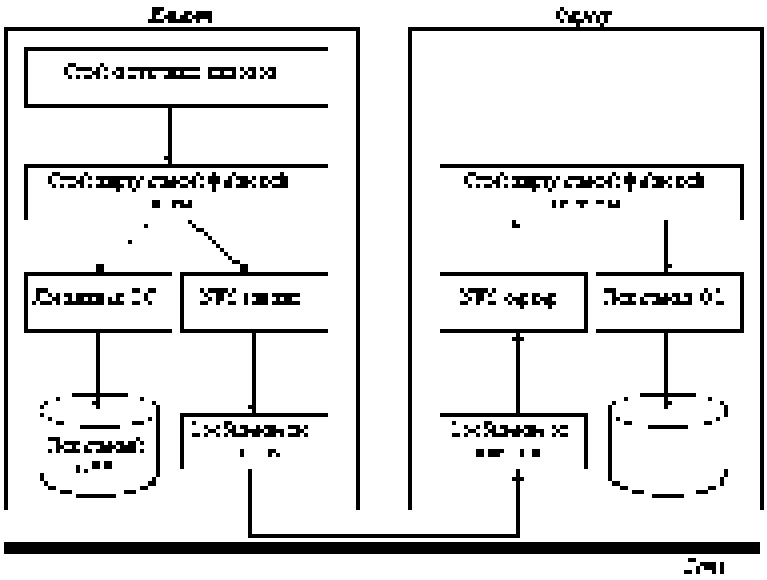


Рис. 5.10. Многоуровневая структура NFS

Верхний уровень клиента - уровень системных вызовов, таких как OPEN, READ, CLOSE. После грамматического разбора вызова и проверки параметров, этот уровень обращается ко второму уровню - уровню виртуальной файловой системы (VFS). В структуре vnode имеется информация о том, является ли файл удаленным или локальным. Чтобы понять, как используются vnode, рассмотрим последовательность выполнения системных вызовов MOUNT, READ, OPEN. Чтобы смонтировать удаленную файловую систему, системный администратор вызывает программу монтирования, указывая удаленный каталог, локальный каталог, на который должен монтироваться удаленный каталог, и другую информацию. Программа монтирования выполняет грамматический разбор имени удаленного каталога и определяет имя машины, где находится удаленный каталог. Если каталог существует и является доступным для удаленного монтирования, то сервер возвращает описатель каталога программе монтирования, которая путем выполнения системного вызова MOUNT передает этот описатель в ядро.

Затем ядро создает vnode для удаленного каталога и обращается с запросом к клиент-программе для создания rnode (удаленного inode) в ее внутренних таблицах. Каждый vnode указывает либо на какой-нибудь rnode в NFS клиент-коде, либо на inode в локальной ОС.

Управление памятью. Свопинг

В UNIX System V Release 4 реализована сегментно-страничная модель памяти в ее традиционном виде. Наряду с механизмом управления страницами используется и механизм свопинга, когда на диск выталкиваются все страницы какого-либо процесса. Свопинг применяется в "предаварийных" ситуациях, когда размер свободной оперативной памяти уменьшается до некоторого заданного порога, так что работа всей системы очень затрудняется.

На рисунке 5.11 показаны основные структуры, описывающие виртуальное адресное пространство отдельного процесса. В дескрипторе процесса proc содержится указатель на структуру as, с помощью которой описываются все виртуальные сегменты, которыми обладает данный процесс. Элемент a_seg в структуре as указывает на первый дескриптор сегмента процесса. Каждый дескриптор сегмента (структура seg) описывает один виртуальный сегмент процесса. Дескрипторы сегментов процесса связаны в двунаправленный список. Дескриптор сегмента содержит базовый адрес начала сегмента в виртуальном адресном пространстве процесса, размер сегмента, а также указатели на операции, которые допускаются над этим сегментом (дублирование, освобождение, отображение и т.д.).

Имеются следующие типы виртуальных сегментов:

- *Текст (text)* - содержит коды команд исполняемого модуля процесса. Он обычно обозначается "только для чтения", так чтобы ни сам процесс, ни другие процессы не могли изменить его кодовую часть. Текстовый сегмент может разделяться многими процессами, например, всеми пользователями, работающими с одним редактором.
- *Данные (data)* - содержит данные, используемые и модифицируемые процессом во время выполнения. К сегменту данных обычно разрешается иметь доступ для чтения и записи. В отличие от текстового сегмента, сегмент данных никогда не разделяется другими процессами.
- *Стек (stack)* - содержит стек процесса. Он помечается доступным для чтения и записи и, подобно сегменту данных, не может разделяться другими процессами.

Есть еще два типа сегментов:

- *Разделяемая память (shared memory)* - область памяти, доступная для чтения и записи нескольким процессам.
- *Отображенный файл (mapped file)* - сегменты отображенного файла используются для того, чтобы отобразить части файлов в адресное пространство процесса, и использовать стандартные механизмы ОС управления виртуальной памятью для ускорения доступа к файлам.

Поле s_data дескриптора сегмента указывает на структуру данных segvn_dat, в которой содержится специфическая для сегмента информация:

- *type:* признак, является ли сегмент разделяемым или личным;
- *vp и offset:* указатель на vnode файла и смещение в этом файле, которые задают адрес, начиная с которого расположены на диске данные этого сегмента;
- *amp:* указатель на карту анонимных страниц сегмента.

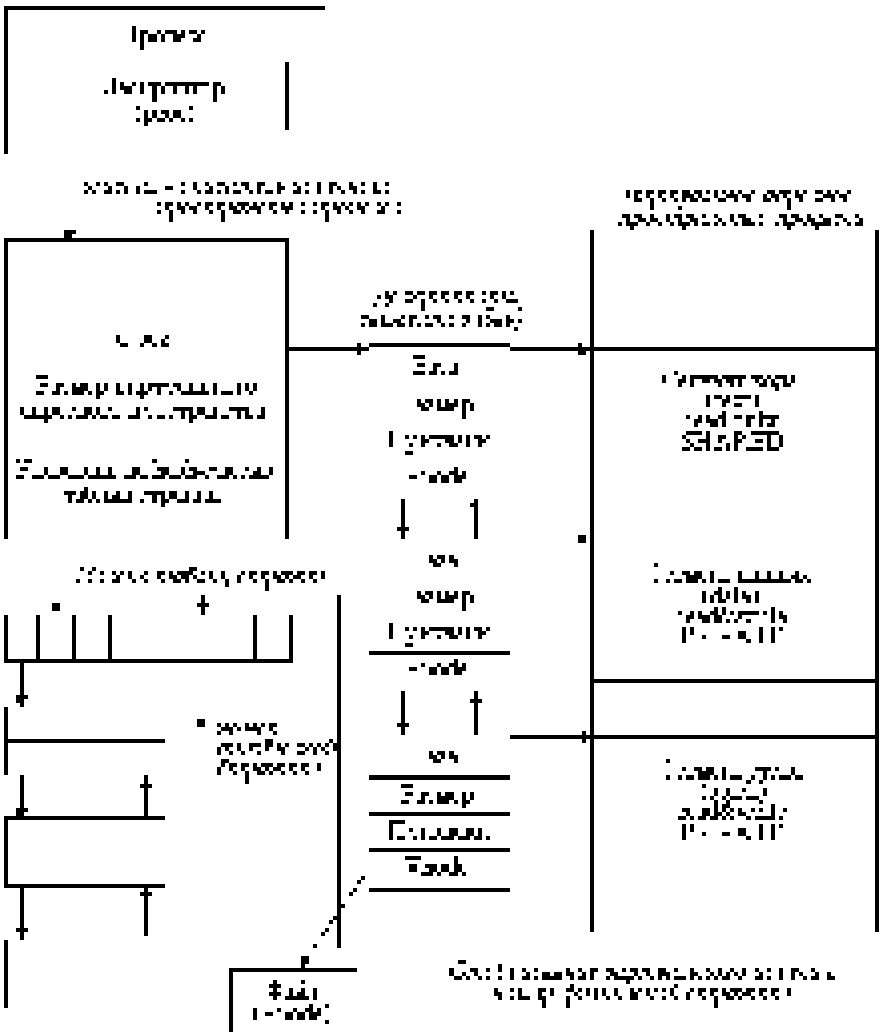


Рис. 5.11. Сегментно-страничная модель виртуальной памяти UNIX

Каждый сегмент имеет связь с дисковым пространством, на котором хранятся данные, отображаемые в данный сегмент виртуального адресного пространства. Это может быть файл или часть файла на диске, или же это может быть область свопинга, которая файлом не является. Сегмент кода или сегмент инициализированных данных обычно связан с файлом, в котором хранится исполняемая программа. Под связью с файлом понимается отображение виртуального сегмента и его страниц на определенную область диска, из которой загружаются данные виртуальных страниц сегмента при их перемещении в оперативную память, а также куда помещаются данные при вытеснении виртуальных страниц на диск. Виртуальные страницы, которые были изначально взяты из определенного файла (который описывается на уровне ядра структурой vnode), называются vnode-страницами, а страницы, которые появились только при развертывании процесса (а это обычно страницы стека или неинициализированных сегментов данных) - анонимными страницами. Однако анонимные страницы также имеют связь с файлом, в который они выталкиваются при их вытеснении из физической памяти (так называемое свопинг-устройство). На свопинг-устройство также указывает vnode, поэтому в этом качестве может выступать любой файл, а перемещение страниц из свопинг-устройства в память осуществляется теми же функциями, что используются для vnode-страниц.

Отображение виртуальных страниц сегмента на физические задается с помощью таблицы HAT (Hardware Address Translation), указатель на которую имеется в структуре адресного пространства процесса as. Структура таблицы HAT зависит от аппаратной платформы, но в любом случае с ее помощью можно найти таблицу или таблицы страниц, содержащих дескрипторы страниц (структуры типа pte). Дескриптор страницы содержит признак наличия данной виртуальной страницы в физической памяти, номер соответствующей физической страницы, а также ряд признаков типа "модификация", "была ссылка", помогающих операционной системе планировать процесс вытеснения виртуальных страниц на диск.

В UNIX System V Release 4 используется алгоритм перемещения виртуальных страниц процесса в физическую память по запросу. Обычно при запуске процесса в физическую память помещается только небольшая часть страниц, необходимая для старта процесса, а остальные страницы загружаются при страничных сбоях. Очевидно, что начальный период работы любого процесса порождает повышенную нагрузку на систему. Если при поиске виртуального адреса в соответствующем дескрипторе обнаруживается признак отсутствия этой страницы в физической памяти, то происходит страничное прерывание, и ядро перемещает эту страницу с диска в физическую память. Для поиска страницы на диске используется информация из структуры s_data сегмента - либо vnode и offset, если страница типа vnode, либо информация о расположении анонимной страницы в области свопинга с помощью информации о ее расположении там по карте amp.

Если в физической памяти недостаточно места для размещения затребованной процессом страницы, то ОС выгружает некоторые страницы на диск. Этот процесс осуществляется специальным процессом ядра, "выталкивателем страниц", имеющем в UNIX System V Release 4 имя pageout. Для принятия решения о том, какую виртуальную страницу нужно переместить на диск, процессу pageout нужно иметь информацию о текущем состоянии физической памяти.

Структура физической памяти

На рисунке 5.12 показан пример распределения физической памяти. В самых нижних адресах находится текстовый сегмент ядра, затем располагается сегмент данных ядра, далее может идти динамический сегмент данных ядра, в котором отводится место для структур ядра, например, для дескрипторов процессов. В оставшейся физической памяти могут располагаться в общем случае несколько областей для хранения страниц пользовательских процессов. Эти области описываются таблицей pageac_table[ ], каждый элемент которой содержит номера начальной и конечной страницы области, указатели на дескрипторы первой и последней страниц, размещенных в этой области.

Для каждой физической страницы имеется дескриптор страницы - структура page, в котором содержится информация о том, свободна или занята страница,

загружена ли в нее в данный момент виртуальная страница, модифицировано ли ее содержимое, сколько процессов хотят сохранить эту страницу в памяти и другая информация. В каждый момент времени дескриптор физической страницы может состоять в одном из следующих списков:

- Список хэшированных виртуальных дескрипторов файла. Каждый отображаемый или выполняемый файл описывается виртуальным дескриптором (vnode). Страницы, относящиеся к отдельному vnode, связываются в список. При этом используется хэширование для быстрого поиска в случае, когда произойдет страничное прерывание.
- Список свободных страниц.
- Список страниц, образующих кэш страниц. Этот список подобен списку свободных станиц, но данные в этих страницах остаются действительными. Страницы обычно помещаются в список кэша процессом pageout. Если происходит поиск какой-либо страницы и эта страница находится в списке страниц кэша, то она повторно используется. Например, страница может быть изъята процессом pageout и помещена в кэш-список. Процесс может затем повторно использовать эту страницу перед тем, как операционная система назначит ее другому процессу. Таким образом, кэшированная страница может быть повторно назначена процессу без перемещения данных с диска.

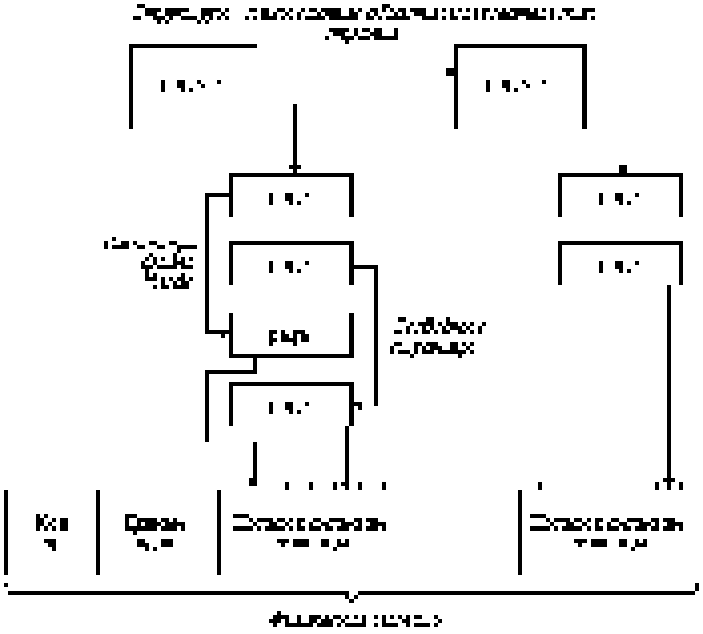


Рис. 5.12. Структуры, описывающие физическую память

Для вытеснения виртуальных страниц с целью освобождения физической памяти в UNIX System V Release 4 используется несколько констант, описывающих размер свободной памяти. Эти константы используются как пороговые значения для действий по освобождению физической памяти (рисунок 5.13). Если свободная память в системе превышает порог lotsfree, то процесс pageout не вызывается вовсе. Если размер свободной физической памяти находится в

пределах от desfree до lotsfree, то pagefree вызывается 4 раза в секунду, а если ее размер становится меньше порога desfree, то pagefree вызывается при каждом цикле работы функции clock(). Если же свободная память становится меньше порога GPGSLO, то в действие вступает процесс свопинга, который в UNIX System V Release 4 называется shed. Этот процесс выбирает определенный процесс, а затем выгружает все его страницы на диск, освобождая тем самым сразу значительное место в памяти. Таким образом UNIX System V Release 4 использует механизм свопинга процессов, который был основным механизмом в ранних версиях UNIX для освобождения физической памяти для других процессов. Процесс, выгруженный на диск, исключается из претендентов на выполнение. Через некоторое время процесс shed вызывается снова. Если количество свободной памяти превысило GPGSL, то процесс загружается с диска в память и включается в очередь готовых к выполнению процессов.

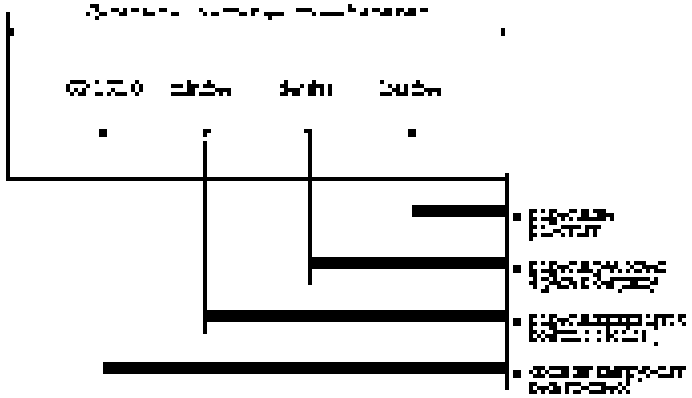


Рис. 5.13. Пороговые значения для действий по освобождению физической памяти

Процесс вытеснения страниц pageout использует при поиске страниц для вытеснения алгоритм NRU (Not Recently Used), выбирающий для вытеснения не используемые в последнее время страницы. Этот алгоритм использует признаки модификации и доступа страниц. Процесс pageout периодически очищает эти признаки у тех страниц, которые не свободны. Если при следующем вызове процесс pageout видит, что эти признаки равны нулю, то значит доступа к этим страницам с момента предыдущего вызова процесса pageout не было, поэтому эти страницы вытесняются на диск. Процесс pageout циклически проверяет все страницы физической памяти, поэтому он называется часовым алгоритмом, что отражает просмотр страниц как бы по часовой стрелке. Было замечено, что обход всех страниц при их большом количестве занимает слишком много времени, поэтому в UNIX System V Release 4 применяется модифицированный часовой алгоритм. Он хорошо иллюстрируется часами с двумя стрелками, которые движутся синхронно, то есть угол между ними сохраняется постоянным. Первая стрелка указывает на виртуальные страницы, признаки которых обнуляются, а вторая - на страницы, признаки которых проверяются и, в случае их равенства нулю, страница вытесняется их физической памяти. При каждом вызове процесс pageout делает лишь часть полного оборота, поэтому при небольшом зазоре

между стрелками в памяти остаются только страницы, к которым идет интенсивное обращение.

Система ввода-вывода

Основу системы ввода-вывода ОС UNIX составляют драйверы внешних устройств и средства буферизации данных. ОС UNIX использует два различных интерфейса с внешними устройствами: байт-ориентированный и блок-ориентированный.

Подсистема буферизации

Любой запрос на ввод-вывод к блок-ориентированному устройству преобразуется в запрос к подсистеме буферизации, которая представляет собой буферный пул и комплекс программ управления этим пулом.

Буферный пул состоит из буферов, находящихся в области ядра. Размер отдельного буфера равен размеру блока данных на диске.

С каждым буфером связана специальная структура - заголовок буфера, в котором содержится следующая информация:

- Данные о состоянии буфера:
- занят/свободен,
- чтение/запись,
- признак отложенной записи,
- ошибка ввода-вывода.
- Данные об устройстве - источнике информации, находящейся в этом буфере:
- тип устройства,
- номер устройства,
- номер блока на устройстве.
- Адрес буфера.
- Ссылка на следующий буфер в очереди свободных буферов, назначенных для ввода-вывода какому-либо устройству.

Упрощенный алгоритм выполнения запроса к подсистеме буферизации приведен на рисунке 5.14. Данный алгоритм реализуется набором функций, наиболее важные из которых рассматриваются ниже.

Функция bwrite - синхронная запись. В результате выполнения данной функции немедленно инициируется физический обмен с внешним устройством. Процесс, выдавший запрос, ожидает результат выполнения операции ввода-вывода. В данном случае в процессе может быть предусмотрена собственная реакция на ошибочную ситуацию. Такой тип записи используется тогда, когда необходима гарантия правильного завершения операции ввода-вывода.

Функция bawrite - асинхронная запись. При таком типе записи также немедленно инициируется физический обмен с устройством, однако завершения операции ввода-вывода процесс не дожидается. В этом случае возможные ошибки ввода-вывода не могут быть переданы в процесс, выдавший запрос. Такая операция записи целесообразна при поточной обработке файлов, когда ожидание завершения операции ввода-вывода не обязательно, но есть уверенность в повторении этой операции.

Функция bdwrite - отложенная запись. При этом передача данных из системного буфера не производится, а в заголовке буфера делается отметка о том, что буфер заполнен и может быть выгружен, если потребуется освободить буфер.

Функции bread и getblk - получить блок. Каждая из этих функций ищет в буферном пуле буфер, содержащий указанный блок данных. Если такого блока в буферном пуле нет, то в случае использования функции getblk осуществляется поиск любого свободного буфера, при этом возможна выгрузка на диск буфера, содержащего в заголовке признак отложенной записи. В случае использования функции bread при отсутствии заданного блока в буферном пуле организуется его загрузка в какой-нибудь свободный буфер. Если свободных буферов нет, то также производится выгрузка буфера с отложенной записью. Функция getblk используется тогда, когда содержимое зарезервированного блока не существенно, например, при записи на устройство данных, объем которых равен одному блоку.

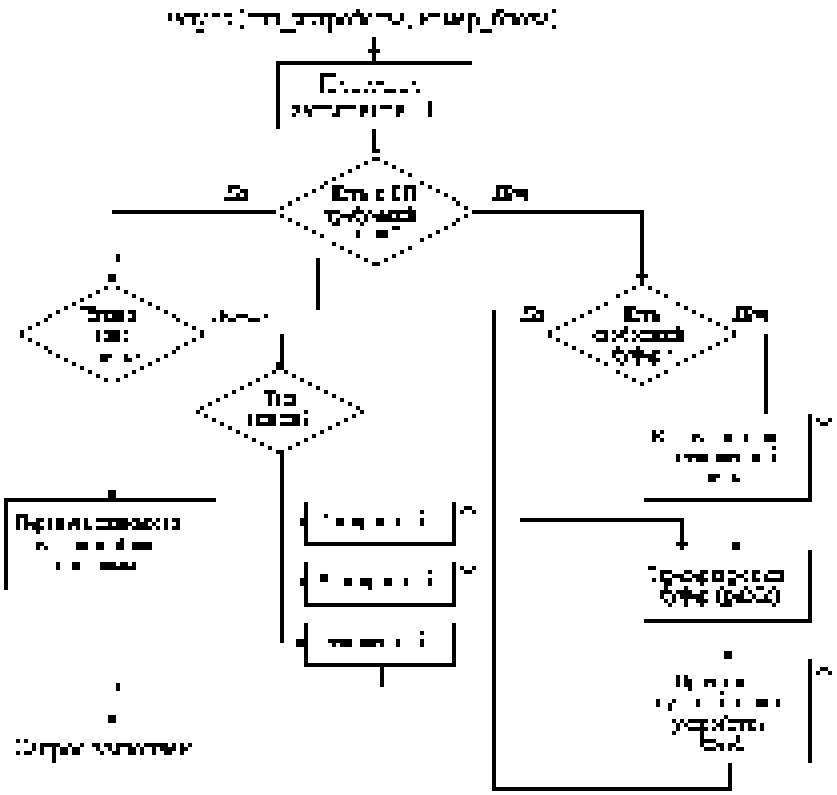


Рис. 5.14. Упрощенная схема выполнения запросов подсистемой буферизации

Таким образом, за счет отложенной записи в системном буферном пуле задерживается некоторое число блоков данных. При возникновении запроса к внешней памяти просматривается содержимое буферного пула. При этом вероятность обнаружения данных в системном пуле достаточно велика. Это обусловлено объективными свойствами пространственной и временной локальности данных. В соответствии с описанным алгоритмом буферизации, в системном буферном пуле оседает наиболее часто используемая информация. Таким образом, система буферизации выполняет роль кэш-памяти по отношению к диску. Кэширование диска уменьшает среднее время доступа к данным на диске, однако при этом снижается надежность файловой системы, так как в случае внезапной потери питания или отказа диска может произойти потеря блоков, содержащихся в системном буфере. Этот недостаток частично компенсируется регулярной (каждую секунду) принудительной записью всех блоков из системной области на диск.

Выше был описан механизм старого буферного кэша, использовавшегося в предыдущих версиях UNIX System V в качестве основного дискового кэша. В UNIX System V Release 4 используется новый механизм, основанный на отображении файлов в физическую память. Однако старый механизм кэширования также сохранен, так как новый кэш используется только для блоков данных файлов, но непригоден для кэширования административной информации диска, такой как inode, каталог и т.д.

Новый буферный кэш

Расположение данных в файле характеризуется их смещением от начала файла. Так как ядро ссылается на любой файл с помощью структуры vnode, то расположение данных в файле определяется парой vnode/offset. Доступ к файлу по адресу vnode/offset достигается с помощью сегмента виртуальной памяти типа segmap, который подобен сегменту segvn, используемому системой виртуальной памяти. Метод доступа к файлам, основанный на сегментах segmap, называется новым буферным кэшем. Этот способ кэширования использует модель страничного доступа к памяти для ссылок на блоки файлов. Размер страницы, используемой новым буферным кэшем, машиннозависим. Для отображения блоков файлов используется адресное пространство ядра, которое также как и пользовательское виртуальное пространство описывается структурой as. Одним из сегментов в адресном пространстве ядра является сегмент segkmap, который используется для страничного доступа к области файлов.

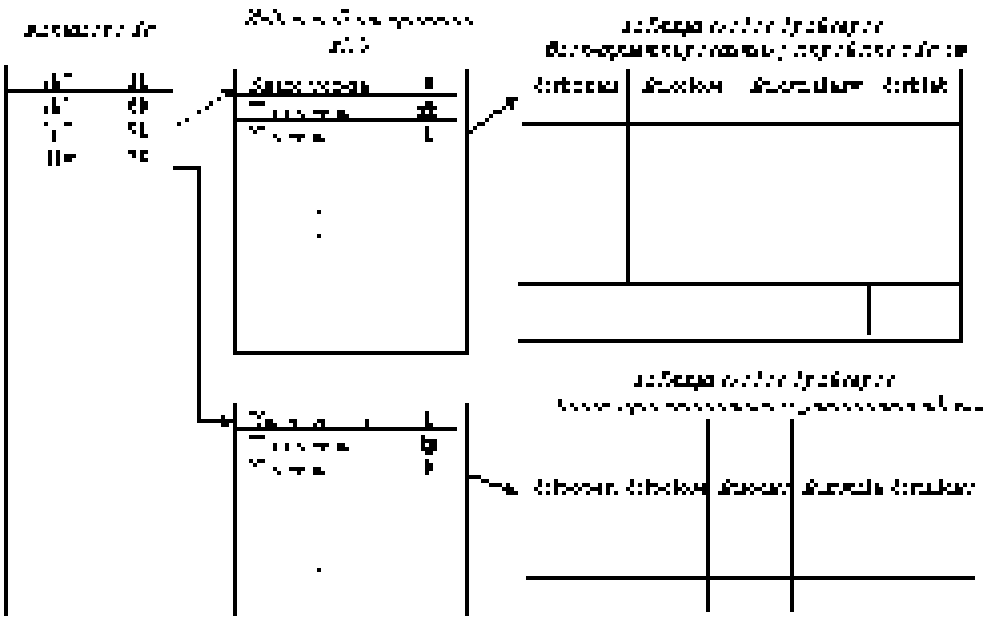


Рис. 5.15. Организация связи ядра с драйверами

Драйверы

Драйвер - это совокупность программ (секций), предназначенная для управления передачей данных между внешним устройством и оперативной памятью.

Связь ядра системы с драйверами (рисунок 5.15) обеспечивается с помощью двух системных таблиц:

- bdevsw - таблица блок-ориентированных устройств и
- cdevsw - таблица байт-ориентированных устройств.

Для связи используется следующая информация из индексных дескрипторов специальных файлов:

- класс устройства (байт-ориентированное или блок-ориентированное),
- тип устройства (лента, гибкий диск, жесткий диск, устройство печати, дисплей, канал связи и т.д.)
- номер устройства.

Класс устройства определяет выбор таблицы bdevsw или cdevsw. Эти таблицы содержат адреса программных секций драйверов, причем одна строка таблицы соответствует одному драйверу. Тип устройства определяет выбор драйвера. Типы устройств пронумерованы, т.е. тип определяет номер строки выбранной таблицы. Номер устройства передается драйверу в качестве параметра, так как в ОС UNIX драйверы спроектированы в расчете на обслуживание нескольких устройств одного типа.

Такая организация логической связи между ядром и драйверами позволяет легко настраивать систему на новую конфигурацию внешних устройств путем модификации таблиц bdevsw и cdevsw.

Драйвер байт-ориентированного устройства в общем случае состоит из секции открытия, чтения и записи файлов, а также секции управления режимом работы устройства. В зависимости от типа устройства некоторые секции могут отсутствовать. Это определенным образом отражено в таблице cdevsw. Секции записи и чтения обычно используются совместно с модулями обработки прерываний ввода-вывода от соответствующих устройств.

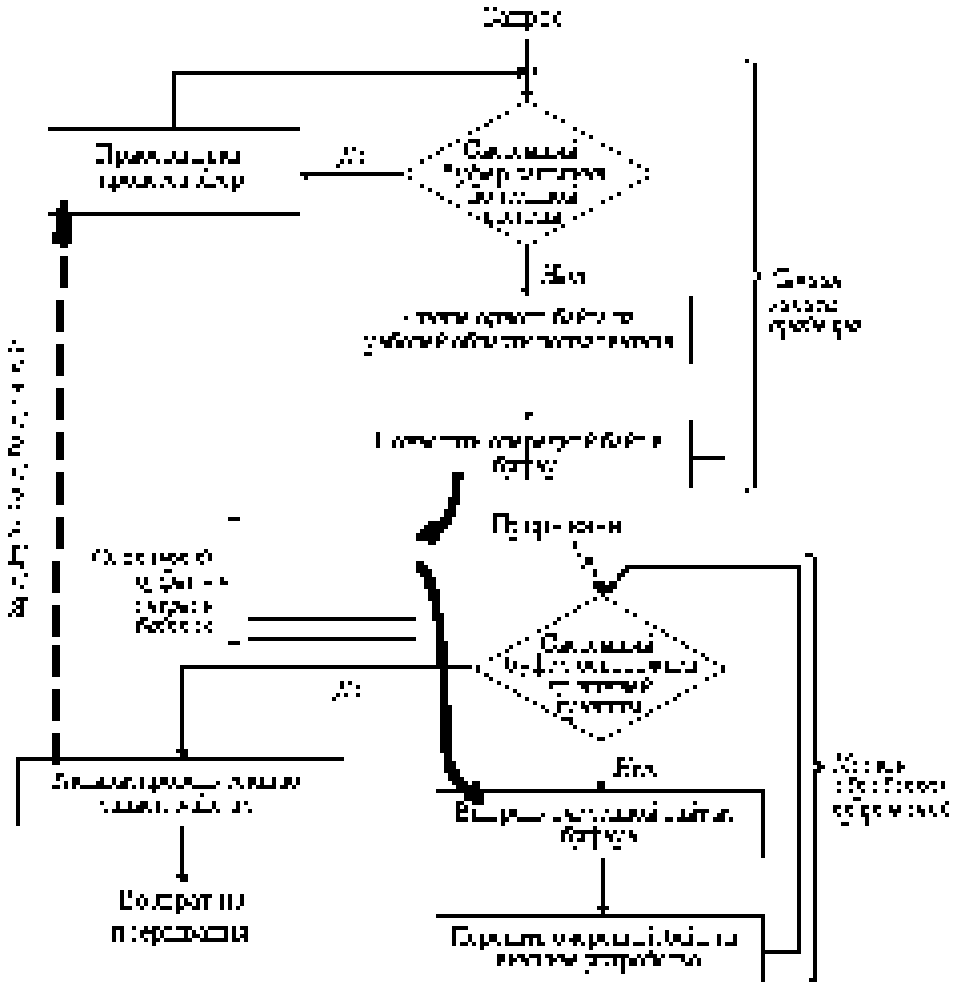


Рис. 5.16. Взаимодействие секции записи драйвера с модулем обработки прерывания

На рисунке 5.16 показано взаимодействие секции записи драйвера байт-ориентированного устройства с модулем обработки прерываний. Секция записи осуществляет передачу байтов из рабочей области программы, выдавшей запрос на обмен, в системный буфер, организованный в виде очереди байтов. Передача байтов идет до тех пор, пока системный буфер не заполнится до некоторого, заранее определенного в драйвере, уровня. В результате секция записи драйвера приостанавливается, выполнив системную функцию sleep (аналог функций типа wait). Модуль обработки прерываний работает асинхронно секции записи. Он

вызывается в моменты времени, определяемые готовностью устройства принять следующий байт. Если при очередном прерывании оказывается, что очередь байтов уменьшилась до определенной нижней границы, то модуль обработки прерываний активизирует секцию записи драйвера путем обращения к системной функции wakeup.

Аналогично организована работа при чтении данных с устройства.

Драйвер блок-ориентированного устройства состоит в общем случае из секций открытия и закрытия файлов, а также секции стратегии. Кроме адресов этих секций, в таблице bdevsw указаны адреса так называемых таблиц устройств (rktab). Эти таблицы содержат информацию о состоянии устройства - занято или свободно, указатели на буфера, для которых активизированы операции обмена с данным устройством, а также указатели на цепочку буферов, в которых находятся блоки данных, предназначенные для обмена с данным устройством.

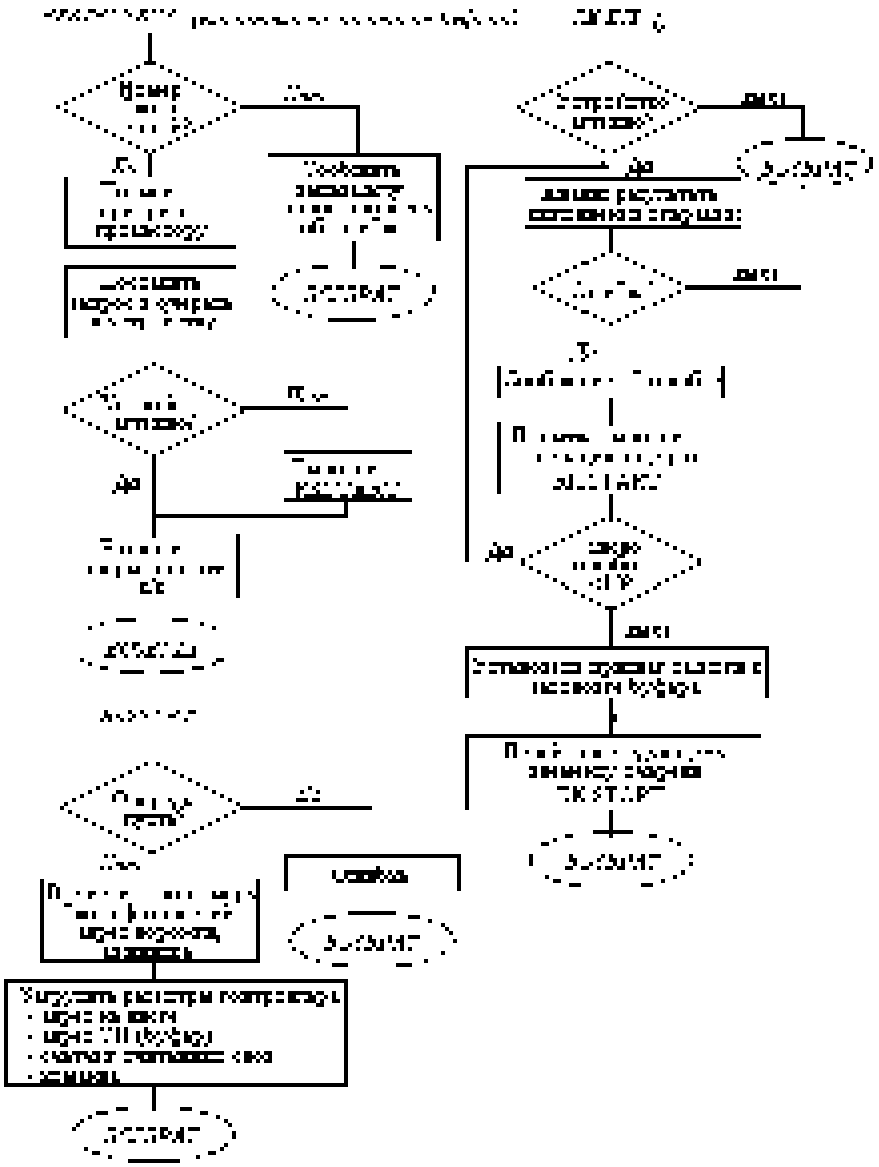


Рис 5.17. Структурная схема драйвера диска типа RK

На рисунке 5.17 приведена упрощенная схема драйвера жесткого диска. Секция стратегии - rkstrategy - выполняет постановку запроса на ввод-вывод в очередь к устройству путем присоединения указанного буфера к цепочке буферов, уже предназначенных для обмена с данным устройством. В случае необходимости секция стратегии запускает устройство (программа rkstart) для выполнения чтения или записи блока с устройства. Вся информация о требуемой операции может быть получена из заголовка буфера, указатель на который передается секции стратегии в качестве аргумента.

После запуска устройства управление возвращается процессу, выдавшему запрос к драйверу.

Об окончании ввода-вывода каждого блока устройство оповещает операционную систему сигналом прерывания. Первое слово вектора прерываний данного устройства содержит адрес секции драйвера - модуля обработки прерываний rkintr. Модуль обработки прерываний проводит анализ правильности выполнения ввода-вывода. Если зафиксирована ошибка, то несколько раз повторяется запуск этой же операции, после чего драйвер переходит к вводу-выводу следующего блока данных из очереди к устройству.

Коммерческие реализации UNIX

UnixWare

UnixWare представляет собой полную реализацию наиболее современной версии системы UNIX для Intel-совместимых платформ - UNIX System V Release 4.2 (SVR4.2). Система сочетает высокую производительность, удобный графический интерфейс и возможности гибкой интеграции с сетями NetWare. Реализованная в ядре поддержка протокола IPX предоставляет пользователям UnixWare прозрачный доступ к сетевым ресурсам NetWare. DOS-клиенты сети получают при этом терминальный доступ к приложениям на сервере UnixWare и возможность коллективного использования файлов, хранящихся на сервере NetWare. Система выпускается в двух вариантах: UnixWare Personal Edition для работы в качестве клиента и однорангового сервера на 2 соединения, UnixWare Application Server, для построения мощного многопользовательского сервера приложений.

Версия UNIX SVR4.2 была создана фирмой UNIX System Laboratories (USL) в 1992 году как развитие версии UNIX System V Release 4. Для совместимости этой версии с наиболее популярными в секторе локальных сетей операционными системами Novell NetWare было создано совместное предприятие USL и Novell Univel, которое разработало и выпустило на рынок операционную систему UnixWare.

Дополнительные свойства UnixWare по сравнению с UNIX System V Release 4

A. Уменьшение требований к оперативной памяти и повышение производительности ядра

Одной из важнейших особенностей UNIX SVR4.2 является возможность эффективно работать на ЭВМ с процессором 386SX и 6 МБ оперативной памяти. Эта возможность появилась в результате работы, направленной на уменьшение размера и увеличение скорости важнейших программных компонентов системы, включая ядро и средства графики. Была проделана работа по улучшению программ загрузчика системы и закрытия системы, а также и драйверов устройств SCSI.

Изменения в структуре ОС и повышение производительности снизили минимальные требования к оперативной памяти на 30%. Преимущества UNIX SVR4.2 по требованиям к объему оперативной памяти еще более заметны по сравнению с системами с аналогичными возможностями. Так, для работы ПО Solaris фирмы SUN требуется минимум 12 МБ памяти, причем для нормальной работы SUN рекомендует использовать 16 МБ ОЗУ.

В ОС UNIX SVR4.2 производительность при нормальной загрузке, при "грязной" загрузке после неаккуратного закрытия, а также при закрытии системы значительно увеличилась по сравнению с предыдущими версиями. В частности, время закрытия системы сократилось на 58% (с 38 до 17 секунд) на типичной аппаратной конфигурации ЭВМ. Загрузка системы при нормальных условиях эквивалентна физическому включению машины после аккуратного закрытия. Время нормальной загрузки сократилось на 48% (с 65 до 38 секунд). При "грязной" загрузке эти времена составляют соответственно 140 и 40 секунд (71%).

B. Отказоустойчивая файловая система Veritas

В дополнение к стандартным файловым системам (BFS, UFS, S5) UnixWare поддерживает: CD-ROM File System (CDFS), NetWare UNIX Client File System (NUCFS) и Veritas Fault Resilient File System. Система Veritas, основанная на транзакционном механизме операций с файловой системой, обеспечивает не только улучшенную производительность, но и высокую устойчивость к отказам системы.

C. Переносимость приложений

Унифицированная программная среда UnixWare обеспечивает поддержку широкого спектра приложений различных систем UNIX, включая SCO, ISC UNIX System V R3, SCO XENIX и BSD UNIX. Совместимость приложений обеспечивается строгим соблюдением промышленных стандартов UNIX System V Application Binary Interface (ABI), System V Interface Definition (SVID), iBSC2 и др.

D. Графический интерфейс

Стандарт X-Window, на основе которого построен мощный и удобный графический пользовательский интерфейс (GUI) UnixWare, в сочетании с сетевыми возможностями системы, позволяет эффективно использовать перспективные архитектуры типа "клиент-сервер". Графическая среда Desktop Manager позволяет выбирать одну из двух стандартных систем графического интерфейса (OSF/Motif или OPEN LOOK) и обеспечивает при работе с графическими объектами на экране доступ к приложениям, большинству системных программ и развитой системе подсказок. Предусмотрена также возможность непосредственного программирования функций Desktop Manager.

E. Поддержка национальных алфавитов

В UnixWare предусмотрен широкий набор средств "интернационализации", включающий поддержку различных раскладок клавиатуры, наборов символов и языков пользовательского интерфейса.

F. Масштабируемые шрифты

В комплект поставки UnixWare входит система Adobe Type Manager, обеспечивающая доступ к тысячам существующих масштабируемых шрифтов формата Type 1.

G. Средства управления доступом

В дополнение к средствам идентификации пользователей по имени и паролю UnixWare имеет развитые средства управления доступом к ресурсам системы. Имеется возможность полного протоколирования работы системы, включая регистрацию выполняемых команд и доступа к информации.

H. Интеграция с NetWare

UnixWare обеспечивает полную интеграцию с сетью NetWare, благодаря которой рабочие станции UnixWare имеют доступ к ресурсам (файловая система, принтеры, почта) сети NetWare, как и другие ее клиенты, а остальные пользователи локальной сети получают также терминальный доступ к серверу приложений UnixWare. При этом как на уровне клиента, так и на уровне сервера приложений операционная система UnixWare использует традиционный для NetWare сетевой протокол IPX. Пользователям UnixWare в локальной сети NetWare предоставляются следующие виды поддержки:

- прозрачный доступ к файлам, принтерам и электронной почте;
- протоколы IPX, SPX и NCP, встроенные в ядро операционной системы;
- поддержка протоколов SAP (Service Advertising Protocol) и RIP (Routing Information Protocol);
- графический пользовательский интерфейс с функциями NetWare.

I. Поддержка мультипроцессирования

Начиная с версии 2.0, UnixWare поддерживает симметричное мультипроцессирование (SMP). Оба варианта UnixWare 2.01 (Application Server и Personal Edition) поддерживают в базовой поставке 2 симметричных процессора Intel. UnixWare Application Server может поддерживать (за счет добавления модулей поддержки дополнительных процессоров) до 8 процессоров Intel. UnixWare 2.01 является многонитевой операционной системой.

Solaris

Solaris 2.x - это операционная система компании Sun, базирующаяся на UNIX System V Release 4. Она включает:

- базовую операционную систему SunOS 5.x и систему сетевой поддержки ONC (Open Network Computing);
- оконную систему Open Windows версии 3.x (построенную на базе X11R5) с интерфейсом в стандарте OPEN LOOK;
- набор вспомогательных утилит (диспетчер файлов, почту, печать, календарь и другие) DeskSet версии 3.x.

Компания Sun доработала исходный код UNIX System V Release 4 в соответствии со своими потребностями. Новая ОС Solaris 2.x имеет несколько основных отличий от базовой операционной системы:

- реализована многонитевость,
- поддерживается симметричная многопроцессорная обработка,
- предусмотрен режим реального времени: допускаются прерывания процессов в системной фазе, что обеспечивает гарантированное время ответа на запросы.

Сетевая среда Solaris 2.x включает в себя известную и уже ставшую стандартом сетевую файловую систему NFS, глобальную справочную службу и средства разработки распределенных приложений. Сегодня Solaris стал одной из самых распространенных версий UNIX. Эта ОС работает на платформах SPARC, Intel x86, а, возможно, в скором времени будет работать и на PowerPC.

Осенью 1994 года компания Sun Microsystems объявила о выпуске новой версии операционной системы Solaris для платформ SPARC и Intel x86 - Solaris 2.4. Эта версия появилась в результате тщательного и долгого тестирования предыдущих версий. На сегодняшний день Solaris 2.4 является наиболее стабильной и качественной версией Solaris. Новое качество выражается не только в устранении всех замеченных в ходе тестирования недостатков, но и в более высокой производительности, чем у Solaris 2.x. В частности, увеличена средняя производительность при работе с СУБД за счет реализации асинхронных операций ввода-вывода в ядре, а не в библиотеках. Производительность

файлового сервера NFS увеличилась в результате более эффективного использования механизма многонитевой обработки. Кроме того, гораздо быстрее стали работать протоколы TCP/IP и программы, реализующие пользовательский интерфейс. Важным свойством Solaris 2.4 является переносимость - программы, написанные для SPARC, могут выполняться на x86 и наоборот.

SCO UNIX System V/386

Варианты ОС UNIX, производимые компанией SCO и предназначенные исключительно для использования на Intel-платформах, до сих пор базируются на лицензированных исходных текстах System V 3.2. Однако SCO довела свои продукты до уровня полной совместимости со всеми основными стандартами (в тех позициях, для которых существуют стандарты). Консерватизм компании объясняется прежде всего тем, что ее реализация ОС UNIX включает наибольшее количество драйверов внешних устройств и поэтому может быть установлена практически на любой Intel-платформе. Естественно, при переходе на другой вариант опорных исходных текстов ядра системы могла бы потребоваться массовая переделка драйверов. Тем не менее SCO имеет соглашение с французской компанией Chorus Systems о разработки новой версии SCO UNIX, базирующейся на микроядре Chorus и предназначенной для использования в системах реального времени.

В настоящее время компания SCO приобрела у Novell ОС UnixWare и работает над версией UNIX, совмещающей особенности SCO UNIX и UnixWare в рамках одной системы.

Микроядро Mach

Ядро любой современной коммерческой версии UNIX представляет собой набор очень большого количества функций, с запутанными взаимосвязями и очень расплывчатыми границами между основными подсистемами. В результате любая модификация организованной таким образом системы дается тяжело и приводит к появлению в новых версиях большого количества ошибок. Кроме того, не во всех инсталляциях нужны все компоненты ядра, а при монолитном его построении удаление ненужных функций затруднено. Недостатки, присущие операционным системам с большим монолитным ядром (а это в первую очередь различные версии UNIX'а), породили интерес к системам, построенным на основе микроядра.

Напомним, что микроядерный подход заключается в том, что базовые функции ядра оформляются в виде отдельной небольшой компоненты, выполняемой в привилегированном режиме, а остальные функции ОС выполняются в пользовательском режиме с использованием примитивов микроядра. Ввиду больших потенциальных преимуществ, которые сулит этот подход, можно предположить, что в ближайшее время большинство новых операционных

систем будет строиться на основе микроядра. Наиболее известными реализациями этого подхода являются микроядра Mach и Chorus.

Основной сложностью использования микроядерного подхода на практике является замедление скорости выполнения системных вызовов при передаче сообщений через микроядро по сравнению с классическим подходом.

Мы достаточно подробно рассмотрим принципы организации и функции микроядра Mach по двум причинам. Во-первых, микроядро по определению содержит базовые механизмы, имеющиеся внутри любой операционной системы, поэтому знакомство с этими механизмами в чистом виде полезно и для изучения любой конкретной ОС.

Во-вторых, микроядра лицензируются и используются как готовый программный продукт в качестве основы для построения коммерческой сетевой операционной системы. Сейчас имеется несколько коммерческих реализаций операционных систем на основе микроядра Mach (NextStep фирмы Next, UNIX BSD, OSF/1 v.1.3), а также проводится ряд работ по использованию этого ядра. Так как свойства микроядра в значительной степени определяют свойства ОС, построенной на его основе, то знание микроядра помогает в оценке характеристик использующей его ОС.

Введение в Mach

История Mach

Система Mach имела в качестве предшественницы систему RIG - Rochester Intelligent Gateway, начало разработки которой пришлось на 1975 год. RIG была написана для 16-битового мини-компьютера компании DataGeneral под названием Elipce. Целью этой разработки была демонстрация возможностей структурирования операционной системы и представления ее в виде набора процессов, которые могут взаимодействовать между собой путем передачи сообщений, в том числе и по сети. Затем эта операционная система была улучшена путем добавления средств защиты и средств прозрачной работы в сети и получила название Accent (в 1981 году, в университете Карнеги-Меллона). В 1984 году она уже использовалась на 150 компьютерах PERQ - ранних графических станциях, но проиграла соревнование с UNIX'ом. Это обстоятельство побудило создать третье поколение ОС, использующей механизм обмена сообщениями. Этот проект и был назван Mach. В связи с тем, что Mach проектировалась как система, совместимая с UNIX, планировалась поддержка большого количества приложений для UNIX. Кроме совместимости с UNIX, в Mach были введены и другие усовершенствования, включая нити, улучшенные механизмы межпроцессного взаимодействия, поддержка многопроцессорных систем, улучшенная виртуальная память и др. В это время агентство DARPA искало операционную систему для поддержки мультипроцессоров. Выбор был сделан в пользу университета Карнеги-Меллона, и работы над ОС Mach были продолжены. Было решено сделать эту систему совместимой с 4.2BSD путем

комбинации Mach и 4.2BSD в виде единого ядра. Хотя этот подход привел к большому ядру, он гарантировал абсолютную совместимость. Первая версия Mach была реализована в 1986 году для VAX11/784, 4-х процессорной машины. Вскоре эта ОС была перенесена на IBM PC RT и Sun 3. К 1987 году Mach выполнялась также на мультипроцессорах Encore и Sequent. Хотя Mach и имела сетевые средства, ее скорее можно было отнести к ОС отдельной машины или мультипроцессора, а не к сетевой распределенной прозрачной системе. Вскоре была создана организация производителей компьютеров OSF (IBM, DEC, Hewlett Packard) для того, чтобы отобрать контроль над ОС UNIX у ее собственника AT&T. Они выбрали Mach 2.5 в качестве основы для их первой операционной системы OSF/1. Хотя Mach 2 и OSF/1 содержали большое количество кода Berkeley и AT&T, была надежда, что OSF, по крайней мере, сможет контролировать направление развития UNIX. В 1988 году ядро Mach 2.5 было большим и монолитным из-за того, что содержало большое количество кода Berkeley UNIX. А в 1989 году университет Карнеги-Меллона удалил весь код BSD UNIX из ядра и поместил его в пользовательское пространство. То, что осталось, было микроядром, состоящим из чистого кода Mach. Эта версия 3.0 и используется как основа последующих версий OSF.

Цели Mach

ОС Mach значительно изменилась со времени ее первой реализации в виде RIG. Цели проекта также изменились со временем. На текущий момент основные цели выглядят так:

1. Обеспечение базовых функций для создания других операционных систем (например, UNIX).
2. Поддержка больших разреженных адресных пространств.
3. Обеспечение прозрачного доступа к сетевым ресурсам.
4. Поддержка параллелизма как в системе, так и в приложениях.
5. Обеспечение переносимости Mach на различные типы компьютеров.

Основные концепции Mach

Микроядро Mach было разработано в качестве основы, на базе которой можно эмулировать UNIX и другие ОС. Эта эмуляция осуществляется программным уровнем, который работает вне ядра, в пользовательском пространстве (рис. 6.1). Следует отметить, что несколько эмуляторов могут работать одновременно, так что можно выполнять программы 4.3BSD, System V и MS-DOS на одной машине в одно и то же время.

Ядро Mach, подобно другим микроядрам, обеспечивает управление процессами, управление памятью, коммуникации и функции ввода-вывода. Функции управления файлами, каталогами и другие традиционные для операционных систем функции выполняются в пользовательском пространстве. Идея построения ядра Mach состоит в обеспечении механизмов, необходимых для

работы системы, но стратегия использования этих механизмов реализуется на уровне пользовательских процессов.

Ядро управляет пятью главными абстракциями:

1. Процессы
2. Нити
3. Объекты памяти
4. Порты
5. Сообщения

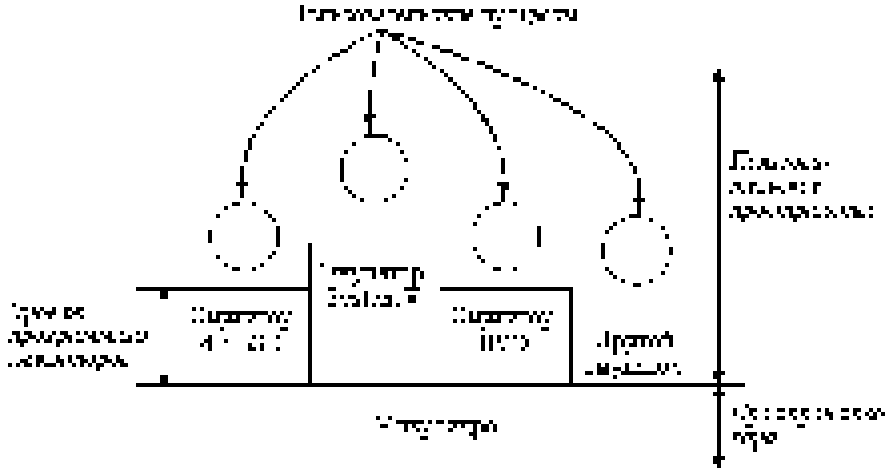


Рис. 6.1. Абстрактная модель эмуляции UNIX на основе Mach

Кроме этого, ядро работает и с некоторыми другими абстракциями, или связанными с указанными, или менее важными.

Процесс - это, в основном, среда, в которой происходит выполнение. Он имеет адресное пространство, содержащее текст программы и данные, и обычно один или более стеков. Процесс - это базисная единица для распределения ресурсов. Например, коммуникационный канал всегда принадлежит одному процессу.

Нить в Mach является единицей выполнения. Она имеет счетчик команд и набор регистров, связанных с ней. Каждая нить является частью точно одного процесса. Процесс, состоящий из одной нити, подобен традиционному (например, как в UNIX) процессу.

Концепцией, уникальной для Mach, является введение понятия *объект памяти* (memory object), представляющий собой структуру данных, которая может быть отображена в адресное пространство процесса. Объекты памяти занимают одну или несколько страниц и образуют основу для системы управления виртуальной памятью Mach. Когда процесс ссылается на объект памяти, который не представлен в физической памяти, это вызывает страничное прерывание. Как и в других ОС, ядро перехватывает страничное прерывание. Однако в отличие от других систем, ядро Mach для загрузки отсутствующей

страницы посылает сообщение серверу пользовательского режима, а не самостоятельно выполняет эту операцию.

Межпроцессное взаимодействие в Mach основано на передаче сообщений. Для того, чтобы получить сообщение, пользовательский процесс просит ядро создать защищенный почтовый ящик, который называется *порт*. Порт хранится внутри ядра и способен поддерживать очередь упорядоченного списка сообщений. Очереди не имеют фиксированной длины, но в целях управления потоком для каждого порта отдельно устанавливается пороговое значение в *n сообщений*, так что всякий процесс, пытающийся послать еще одно сообщение в очередь длины *n*, приостанавливается для того, чтобы дать порту возможность очиститься.

Процесс может предоставить другому процессу возможность посылать (или получать) сообщения в один из принадлежащих ему портов. Такая возможность реализуется в виде *мандата* (capability), который включает не только указатель на порт, но и список прав, которыми другой процесс обладает по отношению к данному порту (например, право выполнить операцию ПОСЛАТЬ - SEND). Все коммуникации в Mach используют этот механизм.

Сервер Mach BSD UNIX

Как уже было сказано выше, разработчики системы Mach модифицировали Berkeley UNIX для работы в пользовательском пространстве в форме прикладной программы. Такая структура имеет несколько преимуществ по сравнению с монолитным ядром. Во-первых, система упрощается за счет разделения на часть, которая выполняет управление ресурсами (ядро), и часть, которая обрабатывает системные вызовы (UNIX-сервер), и ею становится легче управлять. Такое разделение напоминает разделение труда в операционной системе VM/370 мейнфреймов IBM, где ядро эмулирует набор "голых" 370-х машин, на каждой из которых реализована однопользовательская операционная система.

Во-вторых, за счет помещения UNIX'а в пользовательское пространство его можно сделать в высокой степени машинно-независимым. Все машинно-зависимые части могут быть удалены из UNIX'а и скрыты внутри ядра Mach.

В-третьих, как уже было упомянуто выше, несколько ОС могут работать одновременно. Например, на процессоре Intel 386 Mach может выполнять программу UNIX и программу MS-DOS одновременно. Аналогично возможно одновременное тестирование новой экспериментальной ОС и работа с основной ОС.

В-четвертых, в систему могут быть введены операции реального времени, потому что все традиционные препятствия для работы в реальном времени, такие как, например, запрет прерываний на время обновления критических таблиц, могут быть либо исключены, либо перенесены в пользовательское пространство. Ядро может быть тщательно структурировано, для того чтобы не препятствовать

работе приложений реального времени. Наконец, такое построение системы может быть использовано для обеспечения лучшей защиты между процессами, если она нужна. Если каждый процесс работает со своей версией UNIX'а, то для одного процесса очень трудно что-либо разузнать о файлах другого процесса.

Управление процессами в Mach

Процессы

Процесс в Mach - это, в первую очередь, адресное пространство и набор нитей, которые выполняются в этом адресном пространстве. Таким образом, выполнение связано с нитями, а процессы являются пассивными объектами и служат для сбора всех ресурсов, использующихся группой взаимосвязанных нитей, в удобные контейнеры.

Рисунок 6.2 иллюстрирует процесс в Mach. Кроме адресного пространства и нитей, процесс характеризуется используемыми им портами и некоторыми параметрами. Все порты, показанные на рисунке, имеют специальное назначение. *Порт процесса* используется для взаимодействия с ядром. Многие из функций ядра процесс может вызывать путем отправки сообщения на порт процесса, а не с помощью системного вызова. Этот механизм используется в Mach всюду для уменьшения количества системных вызовов до возможного минимума. Некоторые из системных вызовов будут далее обсуждены. Вообще говоря, программист может даже не знать, выполняется ли сервис с помощью системного вызова или нет. Всем сервисам, вызываемым как с помощью системных вызовов, так и с помощью передачи сообщений, соответствуют эрзац-процедуры (заглушки, или стабы) в библиотеке. Это именно те процедуры, которые описаны в руководствах и которые вызываются прикладными программами. Эти процедуры генерируются на основании описания сервиса с помощью компилятора MIG (Mach Interface Generator).

Порт загрузки используется при инициализации, когда система стартует. Самый первый процесс читает из порта загрузки, чтобы узнать имена портов ядра, которые обеспечивают наиболее важные сервисы. Процессы UNIX'а также используют этот порт для взаимодействия с UNIX-эмулятором.

Порт особых ситуаций используется системой для передачи сообщений об ошибках процессу. Примерами особых ситуаций является деление на ноль, неправильный код команды. Этот порт также используют отладчики.

Зарегистрированные порты обычно используются для обеспечения возможностей взаимодействия между процессами и стандартными системными серверами.

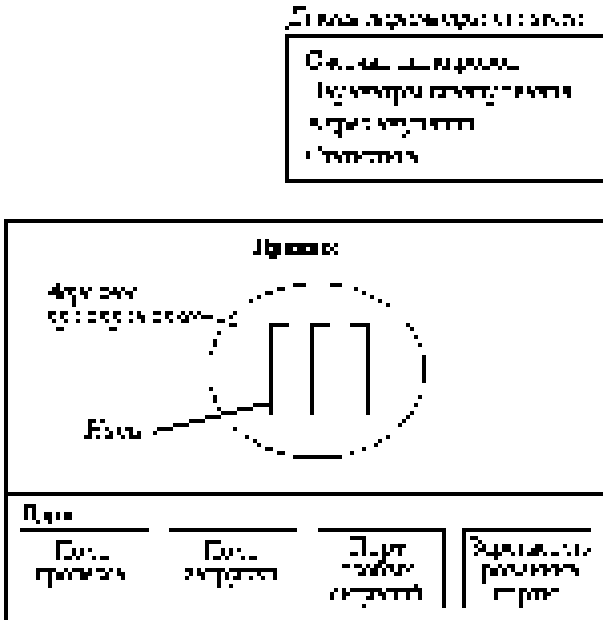


Рис. 6.2. Процесс Mach

Процессы также обладают и другими свойствами. Процесс может быть выполняемым или заблокированным, независимо от состояния его нитей. Если процесс является выполняемым, то те его нити, которые также готовы к выполнению, могут планироваться и выполняться. Если процесс заблокирован, то все его нити не могут выполняться независимо от их состояния.

Параметры планирования, помимо нитей, определяются также для процессов. Эти параметры включают возможность определения того, на каких процессорах могут выполняться нити данного процесса. Это свойство наиболее полезно для многопроцессорных систем. Например, процесс может использовать это свойство для того, чтобы заставить выполняться нить на разных процессорах, или заставить все нити работать на одном процессоре, или реализовать какой-нибудь промежуточный вариант. Кроме того, каждый процесс имеет приоритет, который можно устанавливать. При создании нити ей присваивается этот приоритет. Также имеется возможность изменять приоритет всех существующих нитей.

Наконец, каждый процесс имеет статистику, например, о количестве используемой памяти, временах выполнения нитей и т.д. Любой другой процесс, который интересуется этой статистикой, может получить ее, послав сообщение на порт данного процесса.

Также полезно упомянуть, чем не обладает процесс Mach по сравнению с процессами UNIX'a. Процесс не содержит: идентификатор пользователя, групповой идентификатор, маску сигналов, корневой каталог, рабочий каталог, массив дескрипторов файлов. Вся эта информация содержится в параметрах процесса на уровне сервера пользовательского режима.

Примитивы управления процессами

Mach предусматривает небольшое количество примитивов управления процессами. Большинство из них выполняется путем посылки сообщения ядру через порт процесса. Наиболее важные из этих вызовов приведены в таблице 6.1.

Первые два вызова предназначены для создания и уничтожения процессов. Вызов Create определяет прототип процесса, которым не обязательно является вызывающий процесс. Потомок является копией прототипа, за исключением случая наследования родительского адресного пространства, который определяется заданием параметра. Если адресное пространство не наследуется, то объекты (например, текст, инициализированные данные и стек) могут быть отображены в него позже. Изначально потомок не имеет нитей. Однако он автоматически наследует порты загрузки, процесса и особых ситуаций. Другие порты автоматически не наследуются.

Таблица 6.1.

Вызов	Описание
Create	Создает новый процесс, наследующий некоторые свойства
Terminate	Завершает определенный процесс
Suspend	Наращивает счетчик приостановок
Resume	Уменьшает счетчик приостановок; если он равен 0, то разблокирует процесс
Priority	Устанавливает приоритет для существующих или будущих нитей
Assign	Говорит, на каком процессоре должны выполняться новые нити
Info	Возвращает информацию о времени выполнения, используемой памяти и т.д.
Threads	Возвращает список нитей процесса

Процессы могут быть приостановлены и возобновлены с помощью программного управления. Каждый процесс имеет счетчик, наращиваемый вызовом *Suspend* и уменьшаемый вызовом *Resume*, которые могут блокировать и разблокировать его. Когда счетчик равен 0, то процесс может выполняться. Наличие счетчика позволяет избежать гонок.

Вызовы *Priority* и *Assign* позволяют программисту управлять тем, как и где нити выполняются в многопроцессорной системе. Планирование CPU выполняется на основе приоритетов, так что программист может определять, какие нити более важные, а какие - менее важные.

Нити

Активными объектами в Mach являются нити. Все нити процесса разделяют одно адресное пространство и имеют общие в пределах процесса ресурсы, показанные на рисунке 6.2. Кроме того, каждая нить имеет и свои особые ресурсы. Одним из таких ресурсов является *порт нити*, который является аналогом порта процесса и который используется нитью для того, чтобы вызывать специальные, ориентированные на нити, сервисы ядра, например, функцию завершения нити. Так как порты являются общими ресурсами для всех нитей одного процесса, каждая нить имеет доступ к портам своих нитей-братьев, таким образом каждая нить может управлять другой нитью, если это необходимо. Нити Mach управляются ядром, то есть они являются тем, что иногда называют "тяжеловесными" нитями, в отличие от "легковесных" нитей (нитей, полностью выполняющихся в пользовательском пространстве). Создание и уничтожение нитей осуществляется ядром и включает обновление структур данных ядра. Нити ядра предоставляют базовые механизмы для управления множеством "активностей" в общем адресном пространстве. Что делать с этими механизмами - это дело пользователя.

На однопроцессорной системе нити выполняются в режиме мультипрограммирования. На многопроцессорной системе нити выполняются параллельно. Этот параллелизм делает более важными вопросы синхронизации, взаимного исключения и планирования. Поскольку Mach ориентирована для выполнения на многопроцессорных системах, этим вопросам уделяется особое внимание. Подобно процессу, нить может быть активной или заблокированной. Механизм блокировки очень простой: просто у каждой нити есть счетчик, который наращивается или уменьшается. Когда он равен нулю - нить готова к выполнению. Когда он положителен, нить должна ждать, пока другая нить не уменьшит его до нуля. Такой механизм позволяет нитям управлять поведением друг друга. Имеется некоторое множество примитивов, связанных с нитями. Базовый интерфейс ядра обеспечивает примерно 2 десятка примитивов для нитей. Многие из них связаны с планированием. На базе этих примитивов можно строить различные пакеты нитей.

Мы уже рассматривали один пакет нитей в главе 4, а именно, пакет нитей DCE. Гораздо более скромные возможности предоставляет так называемый C-пакет нитей, поддерживаемый Mach (который был положен в основу пакета OSF). Этот пакет предназначен для того, чтобы дать возможность пользователям использовать нитевые примитивы ядра в простой и доступной форме. Он не предоставляет всю мощность, которую предлагает интерфейс ядра, однако он достаточен для реализации задач среднего уровня сложности. Этот пакет обладает свойством переносимости на широкий круг операционных систем и архитектур.

C-пакет содержит 6 вызовов, предназначенных для непосредственного манипулирования нитями. Эти вызовы приведены в таблице 6.2.

Таблица 6.2.

Вызов	Описание
Fork	Создает новую нить, выполняющую тот же код, что и родительская нить
Exit	Завершает нить
Join	Приостанавливает вызывающую нить до тех пор, пока существует некоторая указанная нить
Detach	Объявляет, что нить никогда не будет присоединена (ее не нужно ждать)
Yield	Отдает управление процессором по собственной инициативе
Self	Возвращает нити ее идентификатор

Первый вызов, *Fork***,** создает новую нить в том же адресном пространстве, в котором работает вызывающая нить. Новая нить выполняет процедуру, указанную в качестве параметра вызова *Fork,* а не процедуру родительской нити. После выполнения вызова родительская нить продолжает выполняться параллельно с порожденной нитью. Нить запускается с приоритетом и назначается на процессор в соответствии с параметрами процесса, к которому она относится.

Когда нить завершает свою работу, она выполняет вызов *Exit.* Если родительская нить заинтересована фактом завершения данной нити и выполнила вызов *Join,* то родительская нить блокирует себя до тех пор, пока данная нить, порожденная ею, не завершится. Если эта нить уже завершилась, родительская нить немедленно продолжает свое выполнение. Эти три вызова является грубыми аналогами системных вызовов UNIX'a FORK, EXIT и WAITPID.

Четвертый вызов, *Detach,* в UNIX'е отсутствует. Он обеспечивает способ, с помощью которого какая-либо нить может объявить, что ее не следует ждать. Обычно очистка стека и другой информации о состоянии родительской нити происходит только после того, как эта нить, присоединившая нить-потомок с помощью вызова *Join*, будет уведомлена о завершении нити-потомка. Для нити, выполнившей вызов*Detach*, такая очистка происходит немедленно после ее завершения (с помощью вызова *Exit*). В сервере это может быть полезно, когда для обслуживания каждого поступающего запроса инициализируется новая нить, а не используется уже существующая.

Вызов *Yield* говорит планировщику о том, что у нити нет полезной работы в данный момент времени, и она отдает управление процессором по собственной инициативе. В Mach, где обычно используется вытесняющая многозадачность,

вызов *Yield* используется только для оптимизации. В системах с невытесняющей многозадачностью этот вызов - основной для планирования работы нитей.

Синхронизация осуществляется с помощью мьютексов и переменных состояния. Примитивами мьютексов являются вызовы *Lock, Trylock* и *Unlock*. Также есть примитивы для распределения и освобождения мьютексов. Примитивы мьютексов в C-пакете те же, что и в пакете DCE.

Реализация C-нитей в Mach

В Mach существует несколько реализаций C-нитей. Оригинальная реализация выполняет все нити в пользовательском пространстве внутри одного процесса. Этот подход основан на разделении во времени всеми C-нитями одной нити ядра, как это показано на рисунке 6.3, а. Этот подход может также использоваться в UNIX'е или любой другой системе, в которой нет поддержки нитей ядром. Такие нити работают как разные подпрограммы одной программы, что означает, что они планируются невытесняющим образом. Например, в случае типовой задачи "производитель-покупатель" производитель должен заполнить буфер, а затем заблокироваться, чтобы дать шанс покупателю на выполнение.

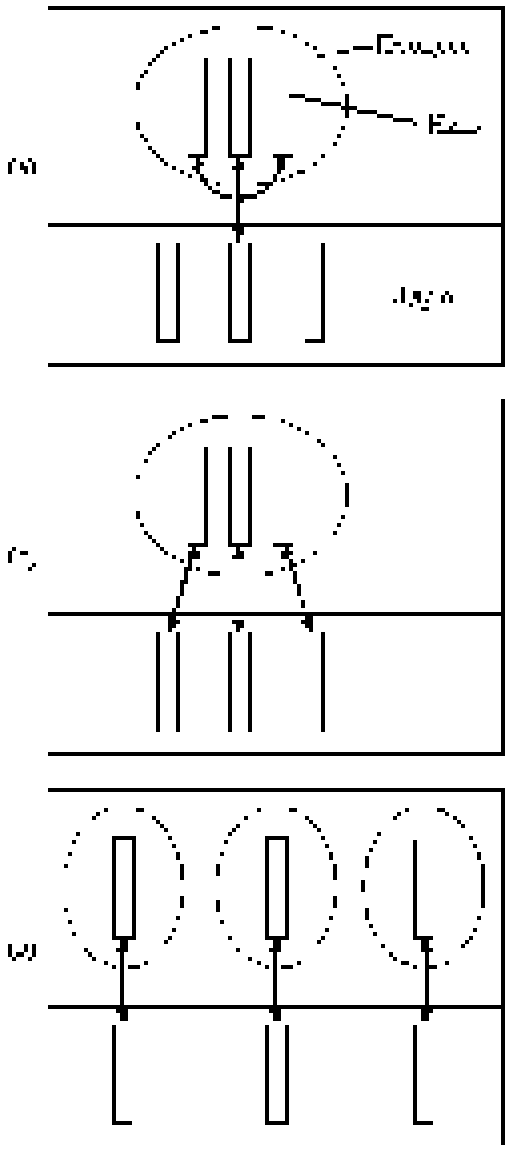


Рис. 6.3. Варианты реализаций С-нитей
(а) - Все С-нити используют одну нить ядра;
(б) - Каждая С-нить имеет свою собственную нить ядра;
(в) - Каждая С-нить имеет свой собственный однонитевый процесс

Данная реализация имеет недостаток, присущий всем нитевым пакетам, работающим в пользовательском пространстве без поддержки ядра. Если одна нить выполняет блокирующий системный вызов, например, чтение с терминала, то блокируется весь процесс. Чтобы избежать этой ситуации, программист должен избегать использовать блокирующие системные вызовы. В BSD UNIX имеется системный вызов SELECT, который может быть использован для того, чтобы узнать, имеется ли символ в буфере терминала. Но в целом операция остается запутанной.

Вторая реализация использует одну Mach-нить для одной С-нити, как показано на рисунке 6.3, б. Эти нити планируются с вытеснением. В многопроцессорных системах нити могут действительно работать параллельно на различных

процессорах. Фактически возможно мультиплексирование m пользовательских нитей с n нитями ядра, хотя наиболее общим случаям является n=m.

В третьей реализации для каждого процесса выделяется одна нить ядра, как показано на рисунке 6.3, в. Процессы реализуются так, что их адресные пространства отображаются в одну и ту же физическую память, позволяя разделять ее тем же способом, что и в предыдущих реализациях. Данный способ поддержки нитей используется только тогда, когда требуется специализированное использование виртуальной памяти. Недостатком этого метода является то, что порты, UNIX-файлы и другие ресурсы процесса не могут разделяться.

Основное практическое значение первого подхода состоит в том, что в условиях отсутствия истинного параллелизма последовательное выполнение нити позволяет воспроизводить промежуточные результаты, облегчая процесс отладки.

Планирование

Планирование в Mach в значительной степени определяется тем, что она предназначена для работы на многопроцессорных системах. Так как система с одним процессором является частным случаем многопроцессорной системы, мы сконцентрируем свое внимание на планировании в многопроцессорных системах.

Все процессоры многопроцессорной системы могут быть приписаны к некоторому *набору процессоров*. Каждый процессор относится точно к одному набору. Таким образом, каждый процессорный набор имеет в своем распоряжении несколько процессоров и несколько нитей, которые нуждаются в вычислительной мощности. В обязанности планирующего алгоритма входит работа по назначению нитей процессорам справедливым и эффективным образом. При планировании каждый процессорный набор представляется замкнутым миром со своими собственными ресурсами и со своими потребителями и является независимым от других процессорных наборов.

Такой механизм предоставляет процессам значительный контроль над своими нитями. Процесс может назначить важную нить процессорному набору, состоящему из одного процессора и не имеющему более ни одной прикрепленной нити, что гарантирует, что эта важная нить будет выполняться все время. Он может также динамически перераспределять нити между процессорными наборами во время их работы, балансируя нагрузку. В то время как средний компилятор обычно не использует этот механизм, система управления базами данных или система реального времени может его использовать.

Планирование нитей в Mach основано на приоритетах. Приоритеты - это целые числа от 0 до 31, причем 0 соответствует наивысшему приоритету, а 31 - самому низшему. Каждой нити присваиваются три значения, влияющих на его

приоритет. Первое значение - базовый приоритет, который нить может назначить сама с определенными ограничениями. Второе число - это наименьшее числовое значение, которое может принять базовый приоритет. Третье значение - это текущий приоритет, используемый в целях планирования. Он вычисляется ядром путем прибавления к базовому приоритету значения функции, учитывающей использование процессора нитью в последнюю итерацию.

При использовании второй модели, в которой каждой нити пользователя соответствует одна нить ядра, ядро Mach видит С-нити. Каждая нить борется за процессорное время со всеми другими нитями, независимо от того, к какому процессу относится та или иная нить. С каждым процессорным набором связано 32 очереди готовых к выполнению нитей, по одной для каждого значения приоритета (рисунок 6.4). Совокупность этих 32 очередей называется глобальной очередью процессорного набора. С каждой глобальной очередью связано три переменных. Первая переменная - это мьютекс, который используется для блокирования структур данных глобальной очереди. Он используется для того, чтобы гарантировать, что с очередью в данный момент времени работает только один процессор. Вторая переменная является счетчиком количества нитей во всех 32 очередях. Третья переменная-ссылка содержит указание о том, где найти наиболее приоритетную нить. Она позволяет искать наиболее приоритетную нить, начиная с некоторого уровня, не просматривая все пустые очереди наверху.

Помимо глобальной очереди, каждый процессор имеет свою собственную локальную очередь, в которой находятся нити, постоянно приписанные к этому процессору, например, потому, что они являются драйверами устройств ввода-вывода, которые закреплены за этим процессором. Эти нити не стоит помещать в глобальную очередь, так как они могут выполняться только на одном определенном процессоре.

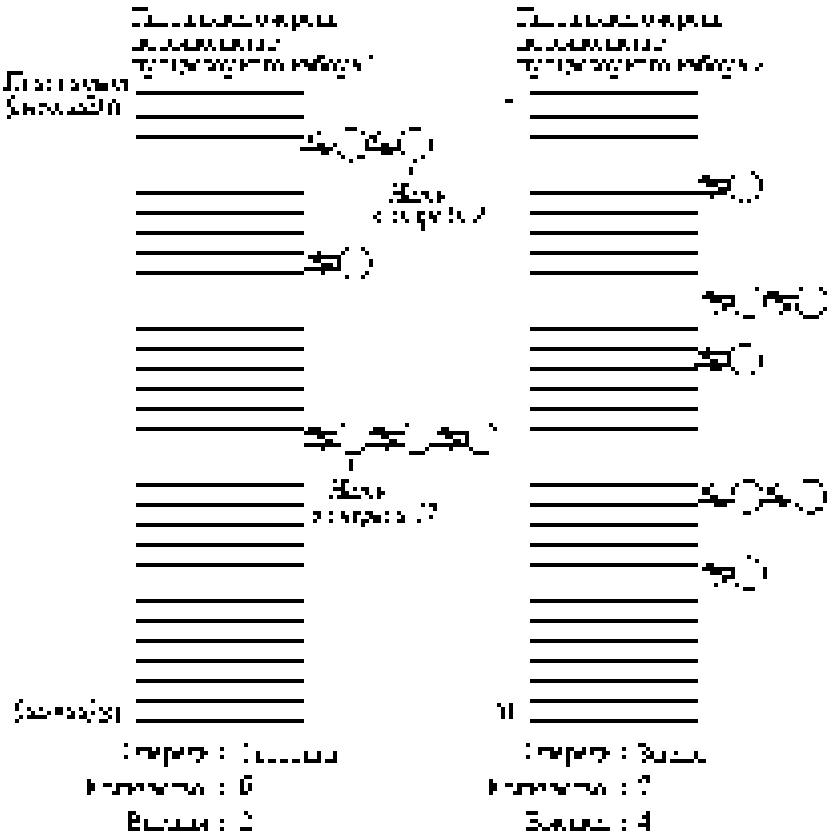


Рис. 6.4. Глобальные очереди на выполнение для ситуации с двумя процессорными наборами

Основной алгоритм планирования заключается в следующем. Когда нить блокируется, завершается или исчерпывает свой квант, процессор, на котором она выполнялась, прежде всего просматривает свою локальную очередь, на предмет того, есть ли в ней какие-либо нити. Для этого анализируется счетчик нитей, связанный с этой очередью. Если он не равен 0, процессор начинает искать наиболее приоритетную нить, начиная с очереди, указанной в переменной-ссылке. Если локальная очередь пуста, такой же алгоритм применяется для глобальной очереди, с той разницей, что глобальная очередь должна быть заблокирована перед тем, как начать выполнять поиск. Если ни в одной очереди нет готовых нитей, то запускается и выполняется специальная нить для простоев до тех пор, пока не появится какая-нибудь готовая нить.

Если готовая нить обнаружена, то она получает возможность выполняться в течение одного кванта. По завершении кванта, обе очереди - локальная и глобальная - просматриваются, чтобы узнать, нет ли еще нитей с таким же или более высоким приоритетом, с учетом того, что все нити из локальной очереди имеют более высокий приоритет, чем нити в глобальной очереди. Если подходящий кандидат найден, то происходит переключение нити. Если нет, то активная нить выполняется еще в течение одного кванта. Нить может быть также вытеснена. На многопроцессорной системе длина кванта может быть переменной, в зависимости от числа нитей, которые готовы к выполнению. Увеличение числа готовых нитей и уменьшение числа процессоров укорачивает

квант. Такой алгоритм дает хорошее время ответа для коротких запросов, даже если система сильно загружена, и обеспечивает высокую эффективность (длинный квант) для мало загруженных систем.

При каждом тике таймера процессор наращивает приоритетный счетчик текущей нити на небольшую величину. Чем больше становится значение приоритета, тем ниже приоритет нити, и нить в конце концов перемещается в очередь с самым большим номером, то есть с самым низким приоритетом.

В некоторых приложениях над решением одной проблемы работает большое количество нитей, поэтому для них важно иметь возможность управлять планированием в деталях. Mach предоставляет нитям дополнительное средство управления планированием (кроме процессорных наборов и приоритетов). Таким средством является системный вызов, который позволяет нити снижать свой приоритет до абсолютного минимума в течение указанного количества секунд. Это дает возможность выполняться другим нитям. После того как указанный интервал истечет, приоритету возвращается его прежнее значение.

Этот системный вызов имеет другое интересное свойство: он может назвать своего наследника, если захочет. Например, после отсылки сообщения другой нити посылающая нить освобождает процессор и требует, чтобы следующей выполнялась нить-получатель. Этот механизм, называемый планированием с передачами, минует все очереди. Если его использовать с умом, то можно повысить производительность. Ядро также использует этот механизм при некоторых обстоятельствах в качестве оптимизации.

Ядро Mach может быть сконфигурировано так, чтобы реализовывать так называемое аффинное планирование, но обычно такая опция не устанавливается. Когда она установлена, ядро планирует нить на процессор, на котором эта нить последний раз выполнялась в надежде на то, что часть ее адресного пространства все еще сохранилась в кэше данного процессора. Аффинное планирование применимо только для многопроцессорных систем.

Управление памятью в Mach

Ядро Mach имеет мощную, тщательно разработанную и в высшей степени гибкую систему управления памятью, основанную на страничном механизме и обладающую многими редкими свойствами. В частности, в нем машинно-зависимая часть кода отделена от машинно-независимой части чрезвычайно ясным и необычным способом. Это разделение делает управление памятью более мобильным, чем в других системах. Кроме того, система управления памятью тесно взаимодействует с коммуникационной системой.

Одной из основных особенностей системы управления памятью Mach является то, что ее код разбит на три части. Первая часть называется *pmap,* который работает в ядре и занимается работой с устройством отображения виртуальных адресов в физические (Memory Management Unit, MMU). Эта часть устанавливает

значения регистров MMU и аппаратных страничных таблиц, а также перехватывает все страничные прерывания. Эта часть кода зависит от архитектуры MMU и должна быть переписана для каждой новой машины всякий раз, когда Mach на нее переносится. Вторая часть - это машинно-независимый код ядра, и он связан с обработкой страничных сбоев, управляет отображением областей памяти и заменой страниц.

Третья часть кода работает в пользовательском пространстве в качестве процесса, называемого "менеджер памяти" (memory manager) или иногда "внешний менеджер страниц" (external pager). Эта часть имеет дело с логическими аспектами (в отличие от физических) системы управления памятью, в основном, с управлением хранения образов памяти на диске. Например, менеджер памяти отслеживает информацию о том, какие виртуальные страницы используются, какие находятся в оперативной памяти и где они хранятся на диске, когда не находятся в оперативной памяти. Ядро и менеджер памяти взаимодействуют с помощью хорошо определенного протокола, что дает возможность для пользователей ядра Mach писать свои собственные менеджеры памяти. Такое разделение обязанностей позволяет реализовывать страничные системы специльного назначения. При этом ядро становится потенциально меньше и проще, так как значительная часть кода работает в пользовательском пространстве. С другой стороны, это и источник усложнения ядра, так как при этом оно должно защищать себя от ошибок или некорректной работы менеджера памяти. Кроме того, при наличии двух активных частей системы управления памятью возможно возникновение гонок между ними.

Виртуальная память

Концептуально модель памяти в Mach выглядит для пользовательских процессов в виде большого линейного виртуального адресного пространства. Для большинства 32-х разрядных процессоров адресное пространство занимает диапазон от 0 до 2^{32} - 1 (4 Гбайта). Адресное пространство поддерживается страничным механизмом.

Mach обеспечивает очень тонкое управление механизмом использования виртуальных страниц (для тех процессов, которые интересуются этим). Для начала скажем о том, что адресное пространство может использоваться разреженным способом. Например, процесс может иметь дюжину секций виртуального адресного пространства, каждая из которых находится на расстоянии сотен мегабайт от ближайшего соседа, с большими зазорами неиспользуемого адресного пространства между этими секциями.

Теоретически любое виртуальное адресное пространство может быть разреженным, так что способность использовать большое количество разбросанных секций не является в действительности свойством архитектуры виртуальной памяти. Другими словами, любая 32-х битная машина должна позволять процессу иметь 50 000 секций данных, разделенных промежутками по 100 Мбайт, в пределах от 0 до 4 Гбайт. Однако, во многих реализациях линейная

страничная таблица от 0 до самой старшей используемой страницы хранится в памяти ядра. На машине с размером страницы в 1 Кб эта конфигурация требует 4 миллиона элементов таблицы страниц, делая такую схему очень дорогой, если не невозможной. Даже при многоуровневой организации страничной таблицы такая разреженность в лучшем случае неприемлема. В ядре Mach при его разработке изначально ставилась задача полной поддержки разреженных адресных пространств.

Для того, чтобы определить, какие виртуальные адреса используются, а какие нет, Mach обеспечивает способ назначения и отмены секций виртуального адресного пространства, называемых *областями*(regions). В вызове для назначения области можно определить базовый адрес и размер, и область выделяется там, где это указано. В другом случае может быть задан только размер области, а система сама находит подходящий диапазон адресов и возвращает базовый адрес. Виртуальный адрес является действительным только в том случае, когда он попадает в назначенный диапазон. Попытка использовать адрес из промежутков между назначенными областями вызывает прерывание, которое может быть перехвачено самим процессом, если он этого пожелает.

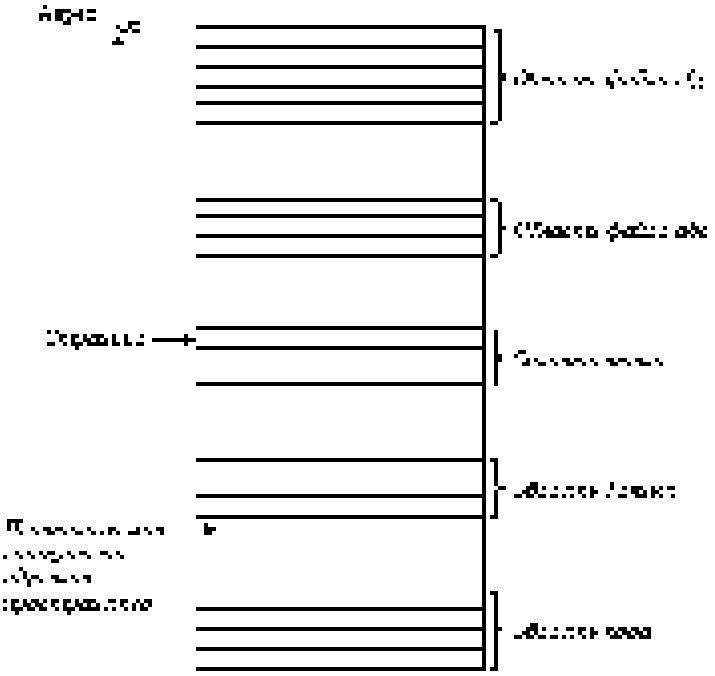


Рис. 6.5. Адресное пространство с назначенными областями, отображенными объектами и неиспользуемыми адресами

Ключевым понятием, связанным с использованием виртуального адресного пространства, является*объект памяти* (memory object). Объект памяти может быть страницей или набором страниц, а также может быть файлом или другой, более специальной, структурой данных, например, записью базы данных. Объект памяти может быть отображен в неиспользуемую часть виртуального адресного пространства, формируя новую область, как это показано на рисунке 6.5. Когда файл отображен в виртуальное адресное пространство, то его можно читать или

в него можно писать с помощью обычных машинных команд. Отображенные (mapped) файлы постранично вытесняются из памяти обычным образом. Когда процесс завершается, то его отображенные в память файлы автоматически возвращаются в файловую систему вместе со всеми изменениями, произошедшими с их содержанием в то время, когда они были отображены в память. Также возможно отменить отображение файла или другого объекта памяти, освобождая его адресное пространство и делая его доступным для последовательного его распределения или отображения.

Конечно, отображение файлов в память не единственный способ доступа к ним. Их содержимое можно читать и обычным способом. Однако и в этом случае библиотечные функции могут отображать файлы в память помимо желания пользователя, а не использовать систему ввода-вывода. Такой способ позволяет страницам файлов использовать систему виртуальной памяти, а не специально выделенные буфера где-либо в системе.

Mach поддерживает некоторое количество вызовов для работы с виртуальным адресным пространством. Основные приведены в таблице 6.3. Они не являются истинными системными вызовами. Вместо этого все они используют запись сообщения в порт процесса для вызова соответствующих функций ядра.

Таблица 6.3.

Примитивы для управления виртуальным адресным пространством

Вызов	Описание
Allocate	Делает область виртуального адресного пространства используемой
Deallocate	Освобождает область виртуального адресного пространства
Map	Отображает объект памяти в виртуальное пространство
Copy	Копирует область в другой диапазон виртуальных адресов
Inherit	Устанавливает атрибут наследования для области
Read	Читает данные из адресного пространства другого процесса
Write	Записывает данные в адресное пространство другого процесса

Первый вызов, *Allocate,* делает область виртуальных адресов используемой. Процесс может наследовать назначенное адресное пространство, а может и назначить дополнительное, но любая попытка обратиться к неназначенному адресу будет вызывать ошибку. Второй вызов, D*eallocate,* освобождает область (то есть удаляет ее из карты памяти), что дает возможность назначить ее заново или отобразить что-нибудь в нее, используя вызов M*ap*.

Вызов C*opy* копирует объект памяти в новую область. Оригинальная область остается неизменной. Операция C*opy* выполняется оптимизированным способом с использованием разделяемых страниц, чтобы избежать физического копирования.

Вызов *Inherit* влияет на способ наследования областью свойств при создании нового процесса. Адресное пространство может быть сконфигурировано так, что некоторые области будут наследовать свойства, а некоторые нет. Это будет обсуждаться ниже.

Вызовы *Read* и *Write* позволяют нити получить доступ к виртуальной памяти, относящейся к другому процессу. Эти вызовы требуют, чтобы вызывающая нить имела права доступа к порту другого процесса, которые процесс может предоставить, если захочет этого.

В дополнение к вызовам, перечисленным в таблице 6.3, существует несколько других. Эти вызовы связаны, в основном, с установкой и чтением атрибутов, режимами защиты и различными видами статистической информации.

Разделение памяти

Разделение играет важную роль в Mach. Для разделения объектов нитями одного процесса никаких специальных механизмов не требуется: все они автоматически видят одно и то же адресное пространство. Если одна из них имеет доступ к части данных, то и все его имеют. Более интересной является возможность разделять двумя или более процессами одни и те же объекты памяти, или же просто разделять страницы памяти для этой цели. Иногда разделение полезно в системах с одним процессором. Например, в классической задаче производитель-потребитель может быть желательным реализовать производителя и потребителя разными процессами, но иметь общий буфер, в который производитель мог бы писать, а потребитель - брать из него данные.

В многопроцессорных системах разделение объектов между процессами часто более важно, чем в однопроцессорных. Во многих случаях задача решается за счет работы взаимодействующих процессов, работающих параллельно на разных процессорах (в отличие от разделения во времени одного процессора). Этим процессам может понадобиться непрерывный доступ к буферам, таблицам или другим структурам данных, для того, чтобы выполнить свою работу. Существенно, чтобы ОС позволяла такой вид разделения. Ранние версии UNIX'а не предоставляли такой возможности, хотя потом она была добавлена.

Рассмотрим, например, систему, которая анализирует оцифрованные спутниковые изображения Земли в реальном масштабе времени, в том темпе, в каком они передаются со спутника. Такой анализ требует больших затрат времени, и одно и то же изображение может проверяться на возможность его использования в предсказании погоды, урожая или отслеживании загрязнения среды. Каждое полученное изображение сохраняется как файл.

Для проведения такого анализа можно использовать мультипроцессор. Так как метеорологические, сельскохозяйственные и экологические программы очень отличаются друг от друга и разрабатываются различными специалистами, то нет причин делать их нитями одного процесса. Вместо этого каждая программа является отдельным процессом, который отображает текущий снимок в свое адресное пространство, как показано на рисунке 6.6. Заметим, что файл, содержащий снимок, может быть отображен в различные виртуальные адреса в каждом процессе. Хотя каждая страница только один раз присутствует в памяти, она может оказаться в картах страниц различных процессов в различных местах. С помощью такого способа все три процесса могут работать над одним и тем же файлом в одно и то же время.

Другой важной областью разделяемости является создание процессов. Как и в UNIX'e, в Mach основным способом создания нового процесса является копирование существующего процесса. В UNIX'e копия всегда является двойником процесса, выполнившего системный вызов FORK в то время как в Mach потомок может быть двойником другого процесса (прототипа).

Одним из способов создания процесса-потомка состоит в копировании всех необходимых страниц и отображения копий в адресное пространство потомка. Хотя этот способ и работает, но он необоснованно дорог. Коды программы обычно используются в режиме только-для-чтения, так что их нельзя изменять, часть данных также может использоваться в этом режиме. Страницы с режимом только-для-чтения нет необходимости копировать, так как отображение их в адресные пространства обоих процессов выполнит необходимую работу. Страницы, в которые можно писать, не всегда могут разделяться, так как семантика создания процесса (по крайней мере в UNIX) говорит, что хотя в момент создания родительский процесс и процесс-потомок идентичны, последовательные изменения в любом из них невидимы в другом адресном пространстве.

Кроме этого, некоторые области (например, определенные отображенные файлы) могут не понадобиться процессу-потомку. Зачем связываться с массой забот, связанных с отображением, если эти области просто не нужны?

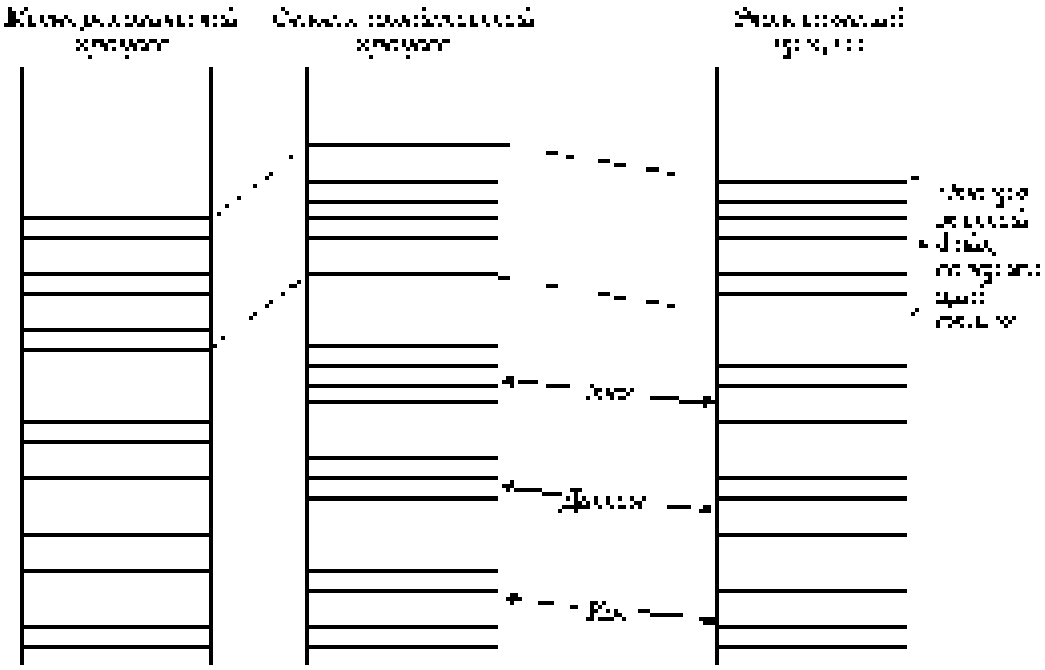


Рис. 6.6. Три процесса, разделяющие отображенный файл

Для обеспечения гибкости Mach позволяет процессу назначить *атрибут наследования* каждой области в адресном пространстве. Различные области могут иметь различные значения атрибута. Предусмотрены три значения атрибута:

1. Область не используется в процессе-потомке.
2. Область разделяется между процессом-прототипом и потомком.
3. Область в процессе-потомке является копией прототипа.

Если область имеет первое значение, то в процессе-потомке она отсутствует. Ссылки на нее рассматриваются как и любые другие ссылки на неназначенную процессу память - они вызывают прерывания. Потомок может назначить эту область для своих собственных целей, или отобразить в нее объект памяти.

Второй вариант представляет собой истинное разделение памяти. Страницы области присутствуют как в адресном пространстве прототипа, так и потомка. Изменения, сделанные в одном пространстве, видны в другом. Этот вариант не используется при реализации системного вызова FORK, но часто бывает полезен для других целей.

Третья возможность - копирование всех страниц области и отображение копий в адресное пространство потомка. FORK использует этот вариант. В действительности Mach не выполняет реального копирования страниц, а использует вместо этого стратегию, называемую *копирование-при-записи* (copy-on-write). Она состоит в том, что все необходимые страницы помещаются в карту виртуальной памяти потомка, но помечаются при этом как предназначенные только для чтения, как это показано на рисунке 6.7. Пока потомок только читает из этих страниц, все работает замечательно.

Однако, если потомок пытается писать в одну из этих страниц, происходит прерывание по защите памяти. Затем операционная система делает копию этой страницы и отображает копию в адресное пространство потомка, заменяя ей защищенную от записи страницу. Новая страница помечается как доступная для чтения и записи. На рисунке 6.7, б потомок пытается произвести запись в страницу 7. Это действие приводит к тому, что страница 7 копируется в страницу 8, а страница 8 отражается в адресное пространство вместо страницы 7. Страница 8 помечается для чтения и записи, так что запись в нее не вызовет прерывание.

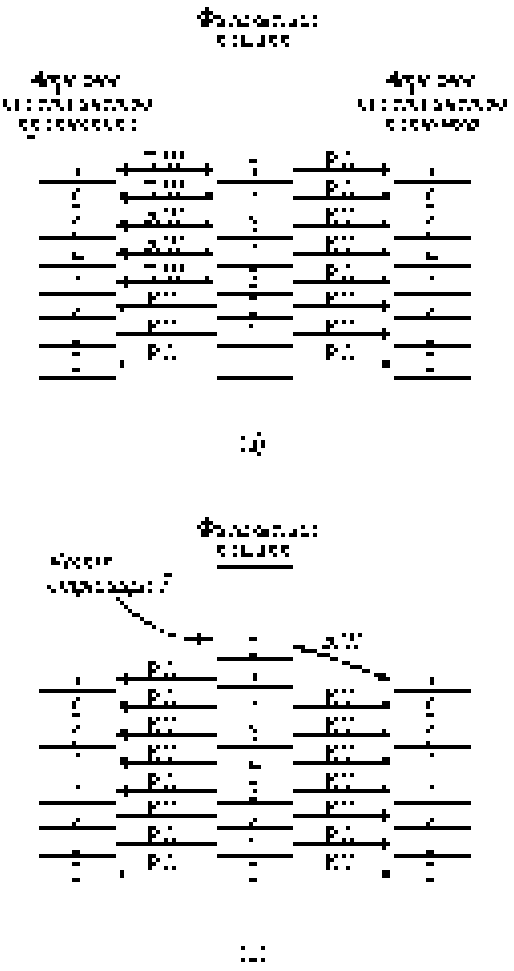


Рис. 6.7. Операция "копирование-при-записи"
а - после вызова FORK все страницы потомка помечаются "только-для-чтения"
б - при записи потомка в страницу 7 делается ее копия

Механизм *копирование-при-записи* имеет несколько преимуществ по сравнению со способом прямого копирования всех страниц в момент создания нового процесса. Во-первых, некоторые страницы имеют режим только-для-чтения, поэтому в их копировании нет необходимости. Во-вторых, к другим страницам обращений может никогда и не быть, поэтому, хотя потенциально в них может производиться запись, копировать их также не нужно. В третьих, хотя некоторые страницы и доступны для записи, но процесс-потомок может удалить их из своего

адресного пространства, не использовав. Поэтому при использовании описанного механизма копируются только те страницы, копирование которых необходимо.

Механизм *копирование-при-записи* обладает и недостатками. С одной стороны, он требует более сложного администрирования, так как система должна хранить информацию о том, какие страницы действительно защищены от записи (так что обращение к ним является ошибкой), а какие нужно копировать при записи. С другой стороны, *копирование-при-записи* приводит к большому количеству прерываний ядра, по одному на каждую страницу, в которую происходит запись. В зависимости от аппаратуры, одно прерывание ядра, сопровождаемое копированием большого количества страниц, может оказаться эффективнее, чем многочисленные прерывания ядра, свойственные механизму *копирование-при-записи*. Наконец, механизм*копирование-при-записи* плохо работает через сеть, потому что в этом случае всегда (то есть при доступе к разделяемым страницам) нужен физический транспорт, так что преимущество этого механизма, связанное с невыполнением копирования только читаемых данных, теряется.

Внешние менеджеры памяти

В начале нашего обсуждения средств управления памятью в Mach мы кратко упомянули о менеджерах памяти пользовательского режима. Теперь рассмотрим их более детально. Каждый объект памяти, который отображается в адресное пространство процесса, должен иметь внешний менеджер памяти, который им управляет. Объекты памяти различных классов управляются различными менеджерами памяти. Каждый из них может реализовывать свою собственную семантику, может решать, где хранить данные страниц, которые не находятся в физической памяти, и следовать своим собственным правилам при решении вопроса о том, что делать с объектами после того, как их отображение в память отменяется.

Для того, чтобы отобразить какой-либо объект в адресное пространство процесса, этот процесс посылает сообщение менеджеру памяти, запрашивая операцию отображения. Для того, чтобы выполнить эту работу, нужны три порта. Первый, *порт объекта* (object port), создается менеджером памяти и используется в дальнейшем ядром для информирования менеджера памяти о страничных отказах и других событиях, связанных с объектом. Второй порт, *порт управления*, создается самим ядром, так что менеджер памяти может реагировать на эти события (многие из них требуют проведения некоторых действий менеджером памяти). Использование различных портов связано с тем, что порты являются однонаправленными. В порт объекта пишет ядро, а читает менеджер памяти, а порт управления работает в обратном направлении.

Третьим является *порт имени* (name port), который используется как разновидность имени, с помощью которого идентифицируется объект. Например, нить может передать ядру виртуальный адрес и спросить, к какой области он относится. Ответом будет указатель на порт имени. Если адреса

относятся к одной и той же области, то они будут идентифицироваться одним и тем же портом имени.

Когда менеджер памяти отображает объект, он обеспечивает создание порта объекта как одного из параметров. Затем ядро создает два других порта и посылает инициализирующее сообщение на порт объекта, несущее информацию о портах управления и имени. Затем менеджер памяти посылает ответ, сообщая ядру о том, какими являются атрибуты объекта, а также о том, нужно или нет хранить объект в кэше после того, как отображение будет отменено. Изначально все страницы объекта помечаются как запрещенные по чтению и записи, чтобы вызвать прерывание при первом использовании.

В этом месте менеджер памяти выполняет чтение порта объекта и блокирует себя. Нить, которая отобразила объект, с этого момента разблокируется и может выполняться. Менеджер памяти простаивает до тех пор, пока ядро не попросит его что-либо сделать, записав сообщение в порт объекта.

Раньше или позже, нить несомненно попытается прочитать страницу или записать в страницу, относящуюся к объекту памяти. Ядро при этом пошлет сообщение менеджеру памяти через порт объекта, информируя его о том, к какой странице произошло обращение, и попросит менеджер обеспечить эту страницу. Это сообщение является асинхронным, так как ядро не решается заблокировать любую из своих нитей в ожидании ответа от пользовательского процесса, которого может и не быть. Во время ожидания ответа ядро приостанавливает вызвавшую обращение к странице нить и ищет другие нити для выполнения.

Когда менеджер памяти узнает о страничном отказе, он проверяет, является ли обращение разрешенным. Если нет, то он посылает ядру сообщение об ошибке. Если же отображение разрешено, менеджер памяти получает страницу тем методом, который подходит для этого объекта. Если объект является файлом, то менеджер памяти ищет корректный адрес и читает страницу в собственное адресное пространство. Затем он посылает ответ ядру, снабжая его указателем на эту страницу.

Ядро отображает страницу в адресное пространство вызвавшего обращение процесса. Теперь нить может быть разблокирована. Этот процесс повторяется до тех пор, пока все нужные страницы не загрузятся.

Чтобы быть уверенным в том, что существует необходимый запас свободных страничных кадров, нить страничного демона время от времени просыпается и проверяет состояние памяти. Если свободных страничных кадров физической памяти недостаточно, то она собирает информацию о старых "грязных" страницах и отсылает ее менеджеру памяти, ответственному за объект, которому принадлежат страницы. Ожидается, что менеджер памяти запишет эти страницы на диск и сообщит, когда эта операция будет завершена. Если страница относится к файлу, то менеджер сначала найдет смещение страницы в файле, а затем запишет ее туда. Используемый при этом алгоритм замены - это второй вопрос.

Стоит заметить, что страничный демон является частью ядра. Хотя алгоритм замены страниц полностью машинно-независим, в ситуации, когда память заполнена страницами, которые управляются разными менеджерами, нет приемлемого способа разрешить одному из них принимать решение о том, какую страницу нужно вытеснить из памяти. Единственным приемлемым способом является статическое распределение страничных кадров между различными менеджерами и разрешение каждому из них заменять страницы в пределах этого набора. Однако из-за того, что глобальный алгоритм, вообще говоря, более эффективен, чем локальный, такой подход не используется.

В дополнение к особым менеджерам памяти, которые отображают файлы и другие специальные объекты, имеется менеджер памяти "по умолчанию" (default) для "обычного" отображения страниц. Когда процесс просит выделить ему область виртуального адресного пространства с помощью вызова *Allocate,* то фактически происходит отображение объекта, управляемого менеджером памяти "по умолчанию". Этот менеджер поставляет заполненные нулями страницы, если это необходимо. Он использует временный файл для образования пространства на диске для свопинга страниц, а не специальную область на диске, как это делается в UNIX'e.

Чтобы концепция внешнего менеджера памяти заработала, должен существовать и использоваться строго определенный протокол взаимодействия между ядром и менеджером памяти. Этот протокол состоит из небольшого количества типов сообщений, которыми обмениваются ядро и менеджер. Любое взаимодействие между ними инициируется ядром в форме асинхронных сообщений, посылаемых в порт объекта некоторого объекта памяти. В ответ на него менеджер посылает асинхронный ответ на порт управления.

В таблице 6.4 приведен список типов сообщений, которые ядро посылает менеджеру памяти.

Таблица 6.4.

Некоторые типы сообщений, посылаемых ядром менеджеру сообщений

Вызов	Описание
Init	Инициализировать новый отображенный в память объект
Data_request	Передать ядру определенную страницу для обработки страничного отказа
Data_write	Взять страницу из памяти и переписать ее
Data_unlock	Разблокирует страницу, так что ядро может ее использовать
Lock_completed	Завершено выполнение предшествующий запрос Lock_request

Terminate	Информирование о том, что данный объект больше не используется

Когда объект отображается в память с использованием вызова *Map,* то ядро посылает сообщение *Init*соответствующему менеджеру памяти, чтобы позволить ему инициализировать себя. Это сообщение определяет порты, которые должны будут использоваться позже в диалоге с объектом. Запросы от ядра на получение страницы и поставку страницы осуществляются с помощью вызовов *Data_request* и *Data_write*соответственно. Эти сообщения управляют трафиком страниц в обоих направлениях, и поэтому являются наиболее важными вызовами.

Сообщение *Data_unlock* представляет собой запрос от ядра менеджеру памяти на разблокирование заблокированной страницы, так что ядро может использовать ее для другого процесса. Сообщение*Lock_completed* оповещает о завершении последовательности *Lock_request*, и будет рассмотрен ниже. Наконец, сообщение *Terminate* сообщает менеджеру памяти, что объект, имя которого указано в вызове, больше не используется и может быть удален из памяти. Существуют вызовы, определенные для менеджера памяти "по умолчанию".

Сообщения, перечисленные в таблице 6.5, передаются от внешнего менеджера памяти ядру.

Таблица 6.5.
Сообщения, посылаемые менеджером памяти ядру

Вызов	**Описание**
Set_attributes	Ответ на вызов Init
Data_provided	Ответ на вызов Data_request - здесь: запрошенная страница доставлена
Data_unavailable	Ответ на вызов - страницы нет в наличии
Lock_request	Запрашивает ядро для выполнения очистки, вытеснения или блокировки страниц
Destroy	Разрушить объект, который больше не нужен

Первое сообщение, *Set_attributes,* является ответом на сообщение *Init.* Оно сообщает ядру, что менеджер готов обрабатывать вновь отображенный объект. Ответ также содержит биты режима для объекта и говорит ядру, следует или нет кэшировать объект, даже если ни один процесс в какой-либо момент времени не отображает его. Следующие два сообщения являются ответами на сообщение *Data_request*. Это сообщение запрашивает страницу у менеджера

памяти. Ответ менеджера зависит от того, может он предоставить страницу или нет. Первое сообщение говорит о том, что может, второе - о том, что нет.

Сообщение *Lock_request* позволяет менеджеру памяти попросить ядро очистить некоторые страницы, то есть послать ему эти страницы, чтобы они могли быть записаны на диск. Это сообщение может быть использовано также и для того, чтобы изменить режим защиты страниц (чтение, запись, выполнение). Наконец, сообщение *Destroy* служит для уведомления ядра о том, что некоторый объект больше не нужен.

Необходимо обратить внимание на то, что когда ядро посылает сообщение менеджеру памяти, оно в действительности выполняет вызов. Хотя при этом способе гибкость повышается, некоторые разработчики систем считают, что это неэлегантно - ядру вызывать пользовательский процесс для выполнения для ядра некоторого сервиса. Эти люди обычно верят в иерархические системы, когда нижние уровни обеспечивают сервис для верхних, а не наоборот.

Распределенная разделяемая память в Mach

Концепция внешних менеджеров памяти в Mach хорошо подходит для реализации распределенной страничной разделяемой памяти. В этом разделе будут кратко описаны некоторые из работ, выполненные в этой области. Основная идея состоит в том, чтобы получить единое, линейное виртуальное адресное пространство, которое разделяется между процессами, работающими на компьютерах, которые не имеют какой-либо физической разделяемой памяти. Когда нить ссылается на страницу, которой у нее нет, то происходит страничное прерывание. Требуемая страница отыскивается на одном из дисков сети, доставляется на ту машину, где произошел страничный сбой, и помещается в физическую память, так что нить может продолжать работать.

Так как Mach уже имеет менеджеры памяти для различных классов объектов, то естественно ввести новый объект памяти - *разделяемую страницу*. Разделяемые страницы управляются одним или более специальными менеджерами памяти. Одной из возможностей является существование единого менеджера памяти, который управляет всеми разделяемыми страницами. Другой вариант состоит в том, что для каждой разделяемой страницы или набора разделяемых страниц используется свой менеджер памяти, чтобы распределить нагрузку.

Еще одна возможность состоит в использовании различных менеджеров памяти для страниц с различной семантикой. Например, один менеджер памяти мог бы гарантировать полную согласованность памяти, означающую, что любое чтение, следующее за записью, всегда будет видеть самые свежие данные. Другой менеджер памяти мог бы предоставлять более слабую семантику, например, что чтение никогда не возвращает данные, которые устарели более, чем на 30 секунд.

Рассмотрим наиболее общий случай: одна разделяемая страница, централизованное управление и полная согласованность памяти. Все остальные

страницы локальны для каждой отдельной машины. Для реализации этой модели нам понадобится один менеджер памяти, который обслуживает все машины системы. Назовем его сервером распределенной разделяемой памяти (Distributed Shared Memory, DSM). DSM-сервер обрабатывает ссылки на разделяемую страницу. Обычные менеджеры памяти управляют остальными страницами. До сих пор мы молчаливо подразумевали, что менеджер или менеджеры памяти, которые обслуживают данную машину, должны быть локальными для этой машины. Фактически, из-за того, что коммуникации прозрачны в Mach, менеджер памяти не обязательно располагается на той машине, чьей памятью он управляет.

Разделяемая страница всегда доступна для чтения или записи. Если она доступна для чтения, то она может копироваться (реплицироваться) на различные машины. Если же она доступна для записи, то должна быть только одна ее копия. DSM-сервер всегда знает состояние разделяемой страницы, а также на какой машине или машинах она в настоящий момент находится. Если страница доступна для чтения, то DSM-сервер сам имеет ее действительную копию.

Предположим, что страница является доступной для чтения, и нить, работающая на какой-то машине, пытается ее прочитать. DSM-сервер просто посылает этой машине копию страницы, обновляет свои таблицы, чтобы зафиксировать еще одного читателя, и завершает на этом данную работу. Страница будет отображена в новой машине и помечена "для чтения".

Теперь предположим, что один из читателей пытается осуществить запись в эту страницу. DSM-сервер посылает сообщение ядру или ядрам, которые имеют эту страницу, с просьбой вернуть ее. Сама страница не должна передаваться, так как DSM-сервер имеет ее действительную копию. Все, что требуется, это подтверждение, что страница больше не используется. После того, как все ядра освободят страницу, писатель получает ее копию вместе с разрешением использовать ее для записи.

Если теперь кто-либо еще хочет использовать эту страницу (которая теперь доступна для записи), DSM-сервер говорит текущему ее владельцу, что ее нужно прекратить использовать и вернуть. Когда страница возвращается, она может быть передана одному или нескольким читателям или одному писателю. Возможны многие вариации этого централизованного алгоритма, например, не запрашивать возвращение страницы до тех пор, пока машина, владеющая ею, не попользуется в течение некоторого минимального времени. Возможно также и распределенное решение.

Коммуникации в ядре Mach

Основной целью, которую ставили перед собой разработчики средств коммуникации ядра Mach, являлась поддержка различных стилей коммуникаций в сочетании с надежностью и гибкостью. Коммуникационные средства ядра Mach могут поддерживать асинхронную передачу сообщений, RPC, потоки байт

(streams), а также другие способы. Механизм взаимодействия процессов Mach базируется на соответствующих механизмах своих предшественников - RIG и Accent. Из-за своего эволюционного развития этот механизм приспособлен больше для локального использования, а не для распределенных систем.

Сначала рассмотрим случай одного узла, а затем расширим его для сети. Следует отметить, что мультипроцессор - это тоже один узел, поэтому взаимодействие процессов, работающих на разных процессорах многопроцессорной машины, также относится к локальному случаю.

Порты

Основой всех коммуникаций в Mach является структура данных ядра, называемая *порт.* В сущности порт представляет собой защищенный почтовый ящик. Когда нить одного процесса хочет взаимодействовать с нитью другого процесса, то нить-отправитель записывает сообщение в такой порт, а нить-получатель извлекает его оттуда. Каждый порт имеет средства защиты, которые гарантируют, что только процессы, имеющие соответствующие права, могут передавать и получать через него сообщения.

Порт поддерживает взаимодействие подобно конвейерам (pipes) в UNIX. Порт, который может быть использован для отправки запроса от клиента серверу, не может использоваться для отправки ответа от сервера клиенту. Для этого нужен второй порт. У каждого порта может быть только один процесс, читающий из него сообщения, и несколько процессов, пишущих в порт.

Порты поддерживают надежный и последовательный обмен сообщениями. Если нить посылает сообщения в порт, то система гарантирует, что оно будет доставлено. Сообщения никогда не теряются из-за ошибок, переполнения или других причин (по крайней мере, если нет отказов аппаратуры). Также гарантируется, что сообщения, отправленные одной нитью, будут получены в том же порядке. Если же две нити пишут в один и тот же порт попеременно, то система не дает никаких гарантий о последовательности сообщений, так как в ядре сообщения буферизуются. В отличие от конвейера, порты поддерживают поток не байтов, а сообщений. Несколько сообщений никогда не соединяются вместе в одно сообщение.

Структура порта показана на рисунке 6.8.

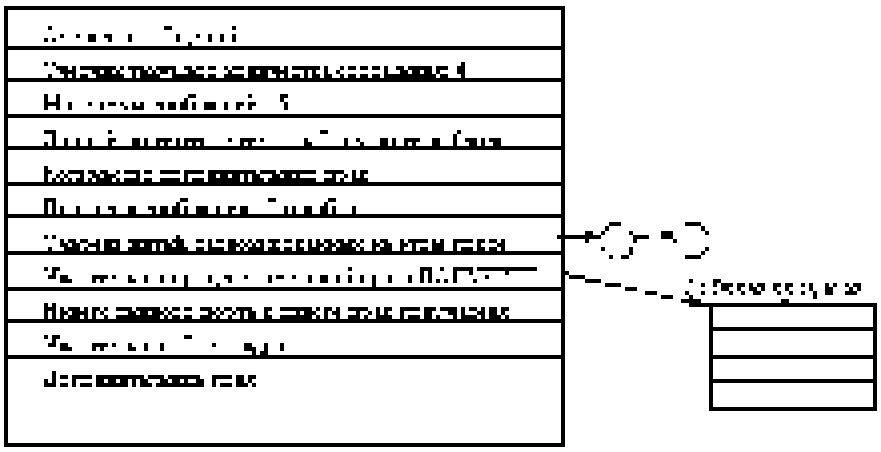


Рис 6.8. Структура порта

Когда порт создается, выделяется 64 байта из пространства ядра и они используются до тех пор, пока порт не разрушен, либо явно, либо косвенно, при определенных обстоятельствах, например, когда не окажется в наличии ни одного процесса, который использует этот порт. Порт содержит поля, указанные на рисунке 6.8, а также еще несколько второстепенных.

Сами сообщения хранятся не в порте, а в другой структуре данных ядра, называемой *очередь сообщений*. Порт содержит счетчик, в котором хранится текущее количество сообщений, находящихся в очереди, и максимально возможное число сообщений в очереди. Если порт относится к какому-либо набору портов, то порт хранит указатель на структуру данных набора портов. Как уже было кратко упомянуто выше, процесс может предоставить другим процессам права на использование его портов. По различным причинам ядро должно знать, сколько прав каждого типа предоставлено, поэтому порт хранит их количество.

Если при использовании порта возникает ошибка, то о ней сообщается путем посылки сообщений на другие порты, чьи права хранятся здесь. Нити могут быть заблокированы при чтении из порта, поэтому в структуру порта включен указатель на список заблокированных нитей. Также важна возможность нахождения прав на чтение из порта (он может быть только один), поэтому такая информация тоже хранится в структуре данных порта. Если порт представляет собой порт процесса, то следующее поле содержит указатель на процесс, к которому порт относится. Если это порт нити, то это поле содержит указатель на структуру данных ядра для нити, и так далее. Имеется также несколько дополнительных полей, не описанных здесь.

Когда нить создает порт, то в ответ она получает целое число, идентифицирующее порт, аналогичное дескриптору файла в UNIX. Это число используется в последовательных вызовах, с помощью которых посылаются сообщения порту или принимаются сообщения от него, для того, чтобы идентифицировать, какой порт нужно использовать. Порты хранятся для процессов, а не для нитей, так что если одна нить создает порт и получает в ответ

идентификатор 3, то другая нить того же процесса никогда не получит этот идентификатор при создании порта. Фактически ядро не запоминает информацию о том, какая нить какой порт создала.

Нить может передать права доступа к порту другой нити другого процесса. Ясно, что нельзя проделать это просто помещением соответствующего целого числа в сообщение, так же как и UNIX-процесс не может передать дескриптор файла для стандартного устройства вывода путем записи 1 в конвейер. Используемый для этого механизм защищен ядром, мы обсудим его позже. В данный момент времени важно только знать, что это можно сделать.

На рисунке 6.9 мы видим ситуацию, в которой два процесса, A и B, имеют доступ к одному и тому же порту. Процесс A послал сообщение для данного порта, а процесс B прочитал это сообщение. Заголовок и тело сообщения физически копируются из A в порт, а затем из порта в B.

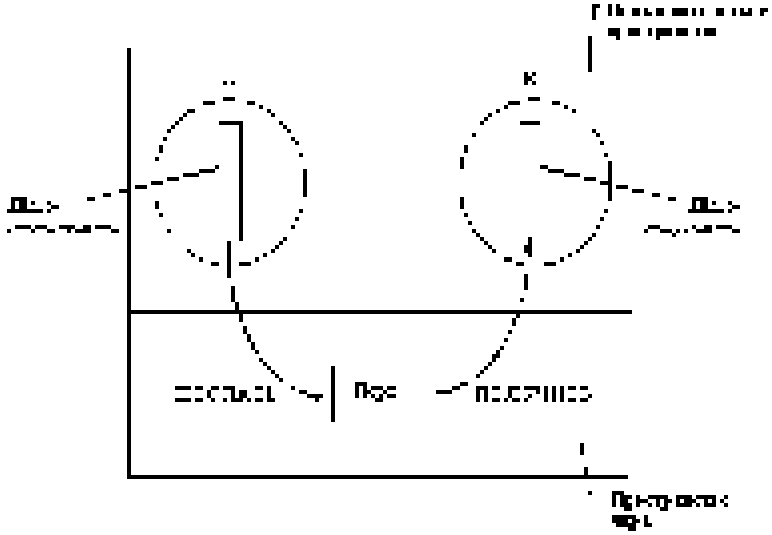


Рис. 6.9. Передача сообщений через порт

Для удобства порты можно группировать в *наборы портов*. Порт может относится максимум к одному набору портов. Можно произвести операцию чтения из набора портов (но записать сообщение можно только в один порт). Например, сервер может использовать этот механизм для одновременного чтения из большого количества портов. Ядро возвращает одно сообщение от одного порта набора. В отношении того, какой порт будет выбран, не дается никаких обещаний. Если все порты пусты, то сервер блокируется. В этом случае сервер может поддерживать разные порты для каждого из многих поддерживаемых им объектов и получать сообщения для каждого из них без необходимости выделять нить для каждого объекта. В соответствии с текущей реализацией все сообщения для набора портов ставятся в общую очередь, так что на практике нет особой разницы между получением из порта и получением из набора портов.

Некоторые порты используются особым образом. Каждый процесс имеет специальный порт - *порт процесса***,** который нужен для связи процесса с ядром.

Большая часть "системных вызовов", связанных с процессами (смотрите табл. 6.1), делается путем посылки сообщения в этот порт. Аналогично, каждая нить имеет свой собственный порт для выполнения "системных вызовов", связанных с нитями. Взаимодействие с драйверами ввода-вывода также реализуется с использованием механизма портов.

Права доступа

В первом приближении, для каждого процесса ядро поддерживает таблицу всех портов, к которым он имеет доступ. Эта таблица хранится в ядре, так что пользовательский процесс не может получить к ней доступ. Процесс ссылается на порты, указывая номер их позиции в этой таблице, например, 1, 2 и так далее. Входы этой таблицы содержат указатели на порты и определяют разрешенные над ними действия, и называются мандатами. Назовем таблицу *списком прав доступа* (списком мандатов).

Каждый процесс имеет ровно один список прав доступа. Когда нить обращается к ядру с запросом на создание для нее порта, ядро делает это и создает запись в таблице прав доступа того процесса, к которому относится данная нить. Вызывающая нить и все другие нити того же процесса имеют равные права доступа. Целое число, возвращаемое нити, чтобы идентифицировать права доступа, является обычно индексом в списке прав. Далее мы будем называть это целое число *именем права доступа*(именем мандата).

Все нити одного процесса имеют равные права по доступу к портам процесса. Права определяются по отношению к трем возможным операциям: ПОЛУЧИТЬ, ПОСЛАТЬ и ПОСЛАТЬ-ОДИН-РАЗ. Право ПОЛУЧИТЬ дает возможность обладателю этого права прочитать сообщение из порта. Ранее упоминалось, что связи в Mach однонаправленные. Это означает, что в любой произвольный момент только один процесс может иметь право ПОЛУЧИТЬ для данного порта. Мандат с правом ПОЛУЧИТЬ можно передавать другому процессу, но это означает, что он будят изъят у исходного процесса. Таким образом, для каждого порта имеется только один потенциальный получатель.

Мандат с правом ПОСЛАТЬ позволяет его владельцу отсылать сообщения определенному порту. Таким правом могут обладать многие процессы. Эта ситуация является грубым аналогом банковской системы во многих странах: каждый, кто знает номер банковского счета, может положить на него деньги, но только владелец может снять деньги со счета.

Право ПОСЛАТЬ-ОДИН-РАЗ также позволяет отослать сообщение, но только однократно. После того, как сообщение отослано, ядро отбирает это право. Этот механизм используется для протоколов типа запрос-ответ. Например, клиент хочет, чтобы сервер выполнил какой-то его запрос, поэтому он создает порт для получения ответа. Затем он отсылает сообщение-запрос серверу, содержащее защищенный мандат для порта ответа с правом ПОСЛАТЬ-ОДИН-РАЗ. После

того, как сервер отошлет ответ, мандат отзывается из его списка прав доступа, и мандат с таким именем может быть только вновь внесен в список прав в будущем.

Имена прав доступа имеют действие только внутри процесса. Два процесса могут иметь доступ к одному и тому же порту, но использовать для этого разные имена. На рисунке 6.10 оба процесса имеют право посылать сообщения на порт *Y,* но для процесса A это мандат номер 3, а для процесса B - мандат 4.

Список прав доступа привязан к определенному процессу. Когда этот процесс завершается или уничтожается, его список удаляется. Порты, для которых он содержал право на операцию ПОЛУЧИТЬ, больше не могут использоваться, и они уничтожаются, даже если в них есть недоставленные сообщения.

Если различные нити процесса получают одни и те же права несколько раз, то в списке прав доступа делается только по одной записи. Для того, чтобы фиксировать данные о том, сколько раз присутствует каждое право, ядро использует счетчик ссылок для каждого порта. Когда права доступа удаляются, счетчик ссылок уменьшается. Право удаляется из списка, только когда он становится равным 0. Этот механизм важен, так как различные нити могут получать и отдавать права без уведомления друг друга.

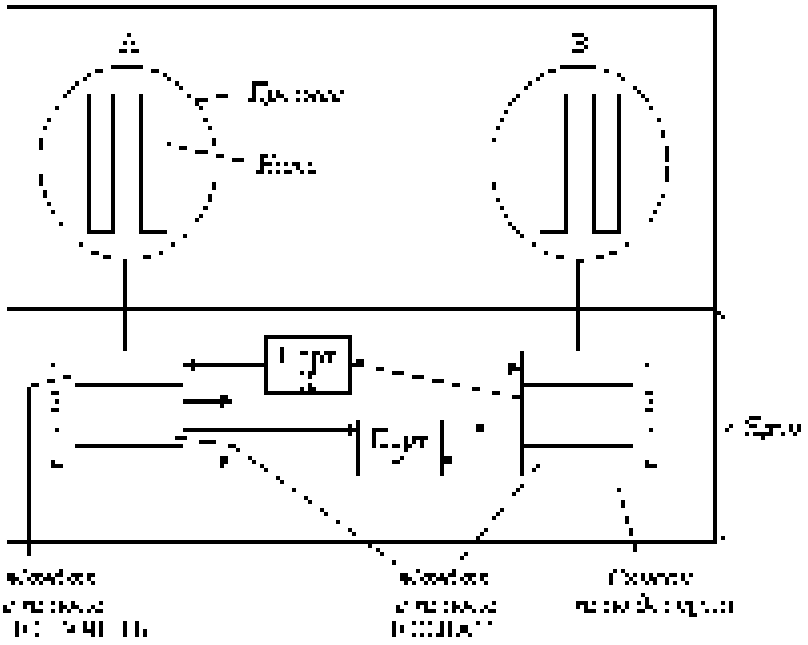


Рис. 6.10. Списки прав доступа

Каждая запись в списке прав может представлять собой одно из четырех значений:

1. Право для порта.
2. Право для набора портов.
3. Нулевая (пустая) запись.

4. Код, показывающий, что порт, который был здесь записан, теперь не существует.

Первая возможность уже детально пояснена. Вторая позволяет нити читать из набора портов, даже не зная о том, что имя права относится к набору портов, а не к отдельному порту. Пустая запись отмечает неиспользуемую строку списка.

Наконец, четвертая возможность отмечает порты, которые уже не существуют, но для которых права ПОСЛАТЬ еще существуют. Например, когда порт удаляется из-за того, что процесс, имеющий право ПОЛУЧИТЬ для него, завершен, ядро отслеживает все права ПОСЛАТЬ и помечает их как несуществующие. Попытка послать сообщение с нулевым и несуществующим правом приводит к ошибке с соответствующим кодом. Когда все права ПОСЛАТЬ для порта отменены, неважно по каким причинам, ядро может (опционально) отослать сообщение, уведомляющее получателя, что не осталось ни одного отправителя, и можно не ожидать сообщений.

Примитивы для управления портами

Mach имеет свыше 20 вызовов для управления портами. Все они вызываются путем отправки сообщения на порт процесса. Наиболее важные из них представлены в таблице 6.6.

Таблица 6.6.
Некоторые вызовы управления портами в Mach

Вызов	Описание
Allocate	Создать порт и включить его права в список прав
Destroy	Разрушить порт и удалить его права из списка
Deallocate	Удалить право из списка прав
Extract_right	Извлечь *n*-ое право другого процесса
Insert_right	Включить право в список другого процесса
Move_member	Переместить порт в набор портов
Set_qlimit	Установить максимальное количество сообщений, которые порт может хранить

Первый вызов, *Allocate*, создает новый порт и вводит его право в список прав процесса, выполнившего этот вызов. Это право является правом ПОЛУЧИТЬ из этого порта. Имя права возвращается, так что порт можно использовать.

Следующие два вызова отменяют действие первого. *Destroy* удаляет право. Если это право ПОЛУЧИТЬ, то порт разрушается и все остальные права для него во всех процессах помечаются как недействительные.*Deallocate* уменьшает счетчик ссылок, связанный с правом. Если он равен нулю, то право удаляется, но порт остается существовать. Вызов *Deallocate* может использоваться только для удаления прав ПОСЛАТЬ и ПОСЛАТЬ_ОДИН_РАЗ или недействительных прав.

Вызов *Extract_right* позволяет нити выбрать право из списка прав другого процесса и вставить это право в свой собственный список. Конечно, вызывающая нить должна иметь доступ к порту процесса, управляющего другим процессом (например, своим потомком). *Insert_right* работает иначе. Он позволяет процессу взять одно из своих прав и добавить его в список прав другого процесса (например, своего потомка).

Вызов *Move_member* используется для управления набором портов. Он может добавить порт к набору портов или удалить порт. Наконец, вызов *Set_qlimit* определяет максимальное количество сообщений, которые порт может хранить. Когда порт создается, то их по умолчанию 5, но с помощью вызова*Set_qlimit* это количество может быть увеличено или уменьшено. Сообщения могут иметь любую длину, так как они сами не хранятся портом.

Отправка и получение сообщений

Конечной целью создания портов является возможность отправки в них сообщений. В этом разделе поясняется, как сообщения отправляются, как получаются и что они содержат. Mach поддерживает один системный вызов для отправки и получения сообщений. Этот вызов содержится внутри библиотечной процедуры, называемой *mach_mes*. Она имеет семь параметров и большое количество опций. Чтобы дать представление о ее сложности, нужно отметить, что существует 35 различных кодов возврата ошибочных ситуаций. Ниже дается упрощенный обзор некоторых возможностей этой процедуры. К счастью, она используется в функциях, генерируемых чаще компилятором заглушек (stub compiler), а не вручную.

Итак, вызов *mach_mes* используется и для отправки и для получения сообщений. Он может отослать сообщение в порт, а затем вернуть управление вызвавшей *mach_mes* функции немедленно, так что она может модифицировать буфер сообщения, не влияя на посланные данные. Он может также использоваться для попытки получения сообщения из порта, причем, если порт пуст, он либо блокирует вызвавшую нить, либо отказывается от попытки по истечении некоторого тайм-аута. Наконец, можно объединять операции отправки и получения, сначала отсылая сообщение, а затем блокируя нить до получения ответа. В последнем режиме вызов *mach_mes* можно использовать для реализации RPC.

Типичный вызов функции *mach_mes* выглядит так:

mach_mes(&hdr, options, send_size, rcv_size, rcv_port, timeout, notify_port);

Первый параметр, *hdr*, является указателем на сообщение, которое нужно отослать или на место, куда нужно поместить приходящее сообщение, или на то и другое. Сообщение начинается с фиксированного заголовка, непосредственно за которым следует тело сообщения. Эта структура показана на рисунке 6.11.

Детали формата сообщения будут рассмотрены ниже, а сейчас необходимо отметить, что заголовок содержит имя права доступа для порта назначения. Эта информация нужна ядру, так как ядро из нее узнает о том, куда нужно отправить сообщение. Когда выполняется операция только ПОЛУЧИТЬ, заголовок не заполняется, так как он будет полностью переписан пришедшим сообщением.

Второй параметр, *options*, содержит бит, определяющий, что сообщение должно быть отправлено, и другой бит, который говорит о том, что оно должно быть получено. Если оба бита включены, то выполняется RPC. Еще один бит разрешает тайм-аут, величина которого указана в параметре *timeout* в миллисекундах. Если требуемая операция не может быть выполнена за время тайм-аута, то вызов возвращает код ошибки. Если часть ПОСЛАТЬ вызова RPC не выполняется за отведенное время (например, порт назначения заполнен в течение слишком большого времени), то часть ПОЛУЧИТЬ просто не выполняется.

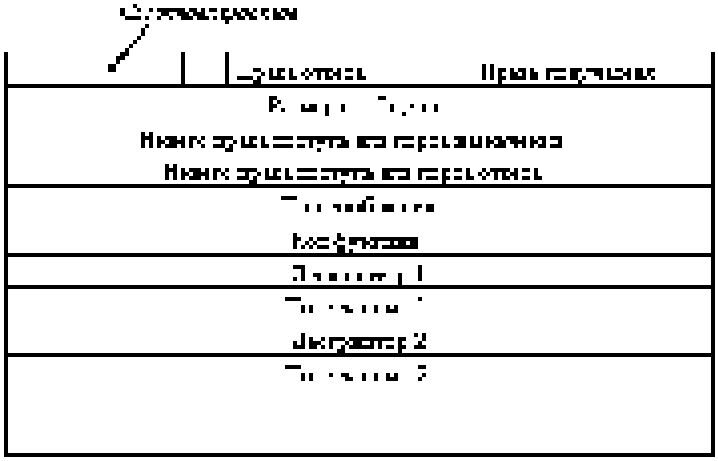


Рис. 6.11. Формат сообщения Mach

Остальные биты поля *options* позволяет операции ПОСЛАТЬ, которая не может завершиться немедленно, вернуть управление, а сообщение о статусе завершения посылается на *notify_port* позже.

Параметры *send_size* и *rcv_size* определяют длину отсылаемого сообщения и количество байт, отводимых для хранения приходящего сообщения, соответственно. *Rcv_port* используется для приема сообщений. Это имя прав доступа порта или набора портов, которые должны получить сообщение.

Теперь рассмотрим формат тела сообщения. Первое слово содержит бит, говорящий о том, является сообщение простым или сложным. Разница состоит в том, что простое сообщение не может содержать прав доступа или защищенных указателей, а сложное может. Простое сообщение требует меньших усилий со стороны ядра и, следовательно, более эффективно. Оба типа сообщения имеют определенную системой структуру, описанную ниже.

Поле *Размер сообщения* указывает общую длину заголовка и тела сообщения. Эта информация нужна как передающей, так и принимающей сторонам.

Далее идут имена двух мандатов доступа (то есть индексы в списке прав доступа передающей стороны). Первый относится к порту назначения, а второй - к порту ответа.

Последние два поля заголовка ядром не используются. Их могут использовать более высокие по иерархии слои программного обеспечения. По соглашению они используются для указания типа сообщения и кода функции или операции (например, для сервера нужно пояснить, это запрос на чтение или на запись). Это назначение может измениться в будущем.

Когда сообщение успешно отослано и получено, оно копируется в адресное пространство адресата. Однако может случиться, что порт назначения уже полностью заполнен. Одной из возможностей тогда будет блокирование отправителя и просто ожидание, пока порт не сможет принять сообщение. Другой вариант связан с организацией тайм-аута. В некоторых случаях можно превысить предел максимального числа сообщений для порта.

Следует упомянуть о некоторых вопросах, связанных с приемом сообщений. Во-первых, что делать, если поступившее сообщение имеет длину больше, чем длина буфера? Предусмотрено два варианта: удалить это сообщение или же вернуть вместе с кодом ошибки данный размер, чтобы позволить вызвавшей функцию *mach_mes* нити повторить передачу с буфером большего размера.

Если существует несколько нитей, блокированных на операции ПОЛУЧИТЬ из одного и того же порта, и сообщение поступает в порт, то системой выбирается одна из них для получения сообщения. Остальные остаются блокированными.

Форматы сообщений

Как уже было сказано, тело сообщения может быть простым или сложным, в зависимости от значения управляющего бита заголовка. Структура сложного сообщения показана на рисунке 6.11. Тело сложного сообщения состоит из последовательности пар (дескриптор, поле данных). Каждый дескриптор несет информацию о том, что находится в поле данных, следующим непосредственно за дескриптором. Дескриптор может иметь два формата, различающихся только количеством бит в каждом его поле. Обычный формат дескриптора приведен на рисунке 6.12. Дескриптор определяет тип последующих данных, их длину, и

количество однотипных данных (поле может содержать некоторое число данных одного типа). Возможны такие типы данны, как последовательность бит и байт, целое число различной длины, неструктурированное машинное слово, булевское значение, число с плавающей запятой, строка и мандат (право доступа). Имея такую информацию, система может попытаться выполнить преобразование данных между машинами, если они имеют различные внутренние представления данных. Это преобразование выполняется не ядром, а сервером сетевых сообщений (описан ниже). Преобразование данных выполняется и для простых сообщений (также этим сервером).

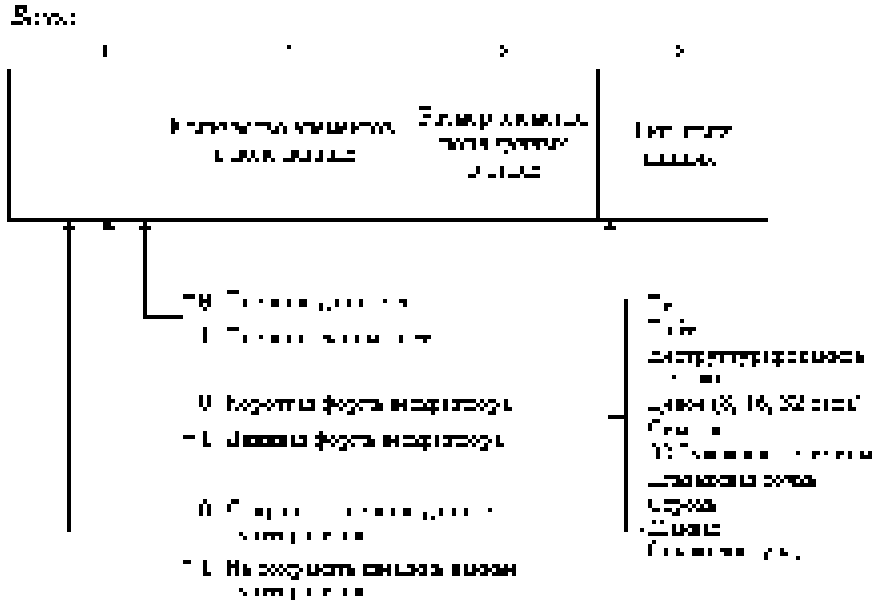


Рис. 6.12. Дескриптор комплексного сообщения

Одним из наиболее интересных типов данных, которые могут содержаться в поле данных сообщения, является мандат. Используя сложные сообщения, можно копировать или передавать мандаты доступа от одного процесса другому. Так как в Mach права доступа являются защищенными объектами ядра, то для их передачи нужен защищенный механизм.

Этот механизм состоит в следующем. Дескриптор может определить, что слово, стоящее в сообщении непосредственно за ним, содержит имя одного из прав доступа отправителя, и что это право нужно переслать процессу-получателю и вставить в его список прав. Этот дескриптор также определяет, должно ли право копироваться (то есть оригинал остается нетронутым) или же перемещаться (оригинал удаляется).

Кроме того, некоторые значения характеристики дескриптора *Тип поля данных* указывают ядру на то, что нужно модифицировать право при копировании или передаче. Например, право ПОЛУЧИТЬ может трансформироваться в право ПОСЛАТЬ или ПОСЛАТЬ-ОДИН-РАЗ, так что получатель сможет посылать ответ через порт, к которому у отправителя есть только право ПОЛУЧИТЬ. Действительно, нормальным способом установления

взаимодействия между двумя процессами является создание одним из них порта, а затем отправка мандата ПОЛУЧИТЬ для этого порта другому процессу, превращая его в мандат ПОСЛАТЬ по дороге.

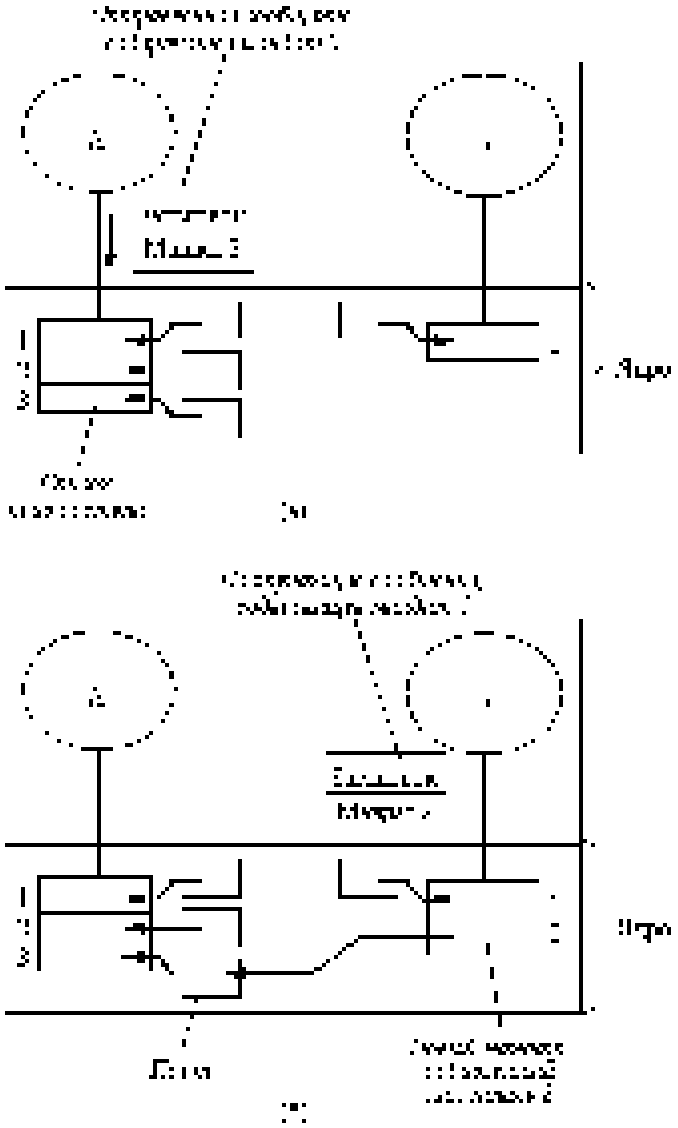


Рис. 6.13. (а) Ситуация перед отправкой мандата; (б) Ситуация после его прибытия

Чтобы увидеть, как работает механизм передачи прав, рассмотрим случай, представленный на рисунке 6.13,а.

Мы видим два процесса, A и B, имеющие мандат 3 и 1 соответственно. Все права являются правом только ПОЛУЧИТЬ. Нумерация начинается с 1, так как значение 0 обозначает нуль-порт. Одна из нитей процесса A отправляет сообщение процессу B, и в этом сообщении содержится мандат 3.

Когда сообщение прибывает, ядро проверяет заголовок и видит, что это сложное сообщение. Затем ядро начинает обрабатывать дескрипторы в теле сообщения, один за другим. В этом примере сообщение содержит только один дескриптор,

описывающий мандат и инструкции ядру, что его нужно превратить в мандат ПОСЛАТЬ (или, может быть ПОСЛАТЬ-ОДИН-РАЗ). Ядро выделяет одну свободную запись в списке права доступа получателя, например, запись номер 2 в данном случае, и модифицирует сообщение таким образом. что слово, следующее за дескриптором, теперь содержит значение 2, а не 3. Когда приемник получает сообщение, то он видит, что оно содержит новое право, с именем (индексом) 2. Он может использовать его немедленно, для отправки ответа.

Имеется еще один интересный тип данных, которые может передавать сообщение: внешние данные (out-of-line). Эти данные не содержатся непосредственно в сообщении, а описаны как область виртуального адресного пространства отправителя. Тем самым в Mach обеспечивается возможность передачи в пределах машины большого количества данных без их физического копирования. Слово, следующее за дескриптором внешних данных, содержит адрес, а размер и количество полей дает 20-битный счетчик байтов. Все вместе это и определяет область виртуального адресного пространства. Для больших областей используется увеличенная форма дескриптора.

Когда сообщение прибывает к получателю, ядро выбирает нераспределенную часть виртуального адресного пространства того же размера, что и внешние данные, и отображает страницы отправителя в адресное пространство получателя, помечая их как страницы с копированием-при-записи. Адрес, следующий за дескриптором, изменяется и теперь указывает на область в адресном пространстве получателя. В зависимости от значения бита дескриптора область может быть удалена из адресного пространства отправителя или нет.

Хотя этот метод очень эффективен при копировании данных между процессами в пределах одной машины, он непригоден для использования в сети, так как страницы придется копировать, даже если они имеют режим только-для-чтения. То есть возможность логического копирования без физического теряется. Метод копирование-при-записи требует, чтобы сообщение было выровнено по границе страницы и составляло целое число страниц. Частично использованные страницы также позволяют получателю видеть данные, которые ему не следует видеть.

Сервер сетевых сообщений

Все коммуникационные механизмы Mach, которые до сих пор были рассмотрены, относились к случаю отдельной машины. Коммуникациями по сети управляют серверы пользовательского режима, называемые серверами сетевых сообщений, аналогами которых являются внешние менеджеры памяти, которые уже были рассмотрены. Каждая машина в распределенной системе Mach имеет сервер сетевых сообщений. Эти серверы работают вместе, обрабатывая межмашинные сообщения.

Сервер сетевых сообщений (Network Message Server) - это многонитевый процесс, который реализует многие функции. Они включают взаимодействие с локальными нитями, передачу сообщений через сеть, трансляцию типов данных

из представления одной машины в представление другой машины, управление права доступа защищенным образом, удаленное уведомление, поддержку простого сервиса поиска сетевых имен, аутентификацию других сетевых серверов. Серверы сетевых сообщений должны поддерживать различные сетевые протоколы, в зависимости от сетей, к которым они присоединены.

Основной метод, с помощью которого сообщения пересылаются через сеть, иллюстрируется рисунком 6.14. На нем изображена машина-клиент А и машина-сервер В. Прежде, чем клиент сможет взаимодействовать с сервером, на машине А должен быть создан порт, чтобы работать как передаточное звено для сервера. Сервер сетевых сообщений имеет право ПОЛУЧИТЬ для этого порта. Нить сервера всегда прослушивает этот порт (и другие удаленные порты, которые вместе образуют набор портов). Этот порт показан как небольшой квадрат внутри ядра машины А.

Передача сообщения от клиента серверу требует пяти шагов, пронумерованных на рисунке 6.14 от 1 до 5. Во-первых, клиент посылает сообщение порту-посреднику сервера сетевых сообщений своей машины. Во-вторых, сервер сетевых сообщений получает это сообщение. Так как это сообщение сугубо локальное, то с помощью него могут быть посланы внешние данные или данные режима копирование-при-записи. В-третьих, сервер сетевых сообщений ищет локальный порт, в нашем примере 4, в таблице, которая отображает порты-посредники на *сетевые порты*. Когда сетевой порт становится известен, сервер сетевых сообщений начинает искать его в других таблицах. Затем он конструирует сетевое сообщение, содержащее локальное сообщение плюс любые внешние данные, и посылает его по локальной сети серверу сетевых сообщений на машине-сервере. В некоторых случаях трафик между серверами сетевых сообщений должен быть зашифрован для защиты информации. Ответственность за разбивку сообщения на пакеты и инкапсуляцию их в формат соответствующего протокола несет транспортный модуль.

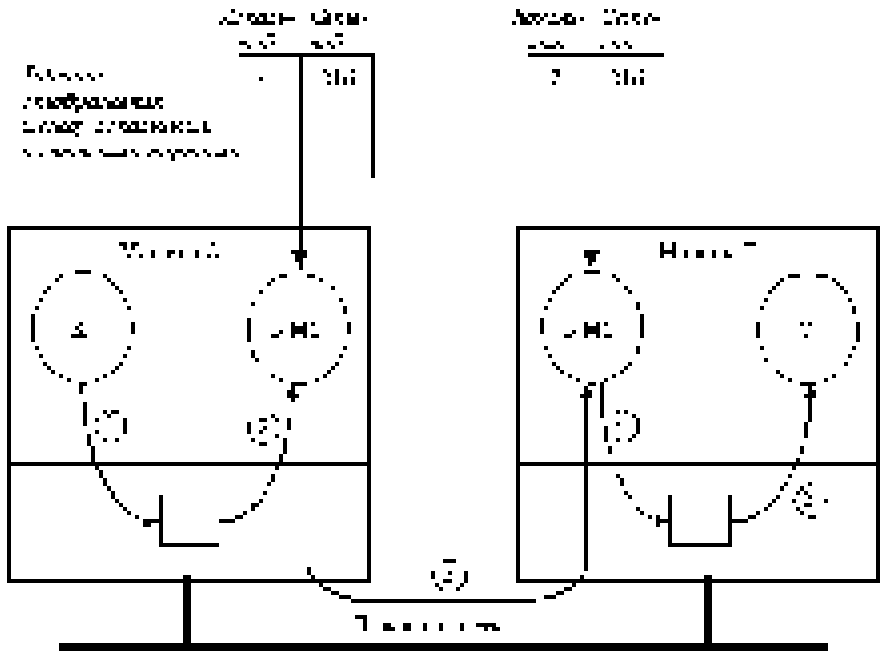


Рис. 6.14. Межмашинное взаимодействие в Mach выполняется за пять шагов

Когда сервер сообщений удаленной сети получает сообщение, он берет из него номер сетевого порта и отображает его на номер локального порта. На шаге 4 сервер записывает сообщение в этот локальный порт. Наконец, локальный сервер машины B читает сообщение из локального порта и выполняет запрос клиента машины A. Ответ проходит по этому же пути в обратном направлении.

Сложное сообщение требует небольшой дополнительной работы. Для обычных полей данных сервер сетевых сообщений на машине-сервере выполняет преобразование форматов данных, если это необходимо (например, изменяет порядок байт в слове). Обработка прав доступа при пересылке через сети также усложняется. Когда мандат пересылается через сеть, должен быть назначен номер сетевого порта, и оба сервера сетевых сообщений, отправитель и получатель, должны сделать соответствующие записи в своих таблицах отображения. Если эти машины не доверяют друг другу, необходимы тщательно разработанные процедуры аутентификации, чтобы каждая из машин убедилась в идентичности другой.

Хотя механизм передачи сообщений от одной машины к другой через сервер пользовательского режима обладает гибкостью, за это платится высокая цена, выражающаяся в снижении производительности по сравнению с реализацией межмашинного взаимодействия полностью внутри ядра, как это делается в большинстве других распределенных систем.

Эмуляция BSD UNIX в Mach

В ядре Mach имеется различные серверы, которые работают над ним. Наверно, наиболее важным сервером является программа, которая содержит большое количество кодов BSD UNIX (например, весь код файловой системы). Этот сервер представляет собой основной эмулятор UNIX. Такая конструкция - отражение реальной истории Mach как модифицированной версии BSD UNIX.

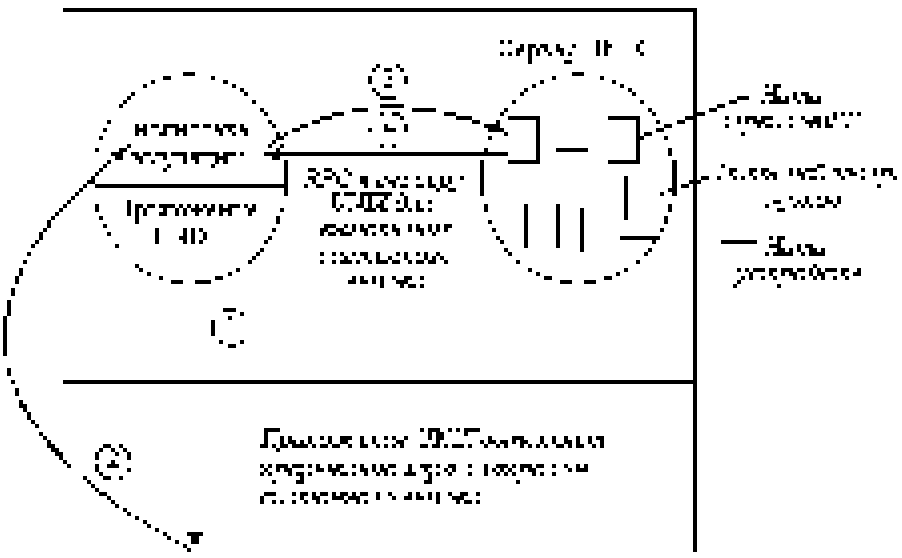


Рис. 6.15. Эмуляция UNIX в Mach

Реализация механизма эмуляции UNIX в среде Mach состоит из двух частей, сервера UNIX и библиотеки эмуляции системных вызовов, как это показано на рисунке 6.15. Когда система стартует, сервер UNIX инструктирует ядро, чтобы оно перехватывало все прерывания системных вызовов и отображало вектора этих прерываний на адреса внутри библиотеки эмуляции процесса UNIX'а, по которым расположены обрабатывающие данные вызовы функции. Любой системный вызов, который делается UNIX-процессом, приводит к кратковременной передаче управления ядру, а затем к немедленной передаче управления библиотеке эмуляции. Значения машинных регистров в момент передачи управления библиотеке становятся теми же, что и в момент прерывания. Такой метод иногда называют методом батута.

Как только библиотека эмуляции получает управление, она проверяет регистры для того, чтобы определить, какой системный вызов нужно обработать. Затем библиотека делает вызов RPC другого процесса, сервера UNIX, который и должен выполнить эту работу. После завершения обработки вызова пользовательская программа снова получает управление. Эта передача управления не проходит через ядро.

Когда процесс *init* порождает потомков с помощью системного вызова fork, то они автоматически наследуют как библиотеку эмуляции, так и механизм батута, поэтому они могут выполнять системные вызовы UNIX.

Сервер UNIX реализован в виде набора C-нитей. Хотя некоторые нити управляют таймерами, работают с сетью и другими устройствами ввода-вывода, большинство нитей обрабатывают системные вызовы BSD. Библиотека эмуляции взаимодействует с этими нитями с использованием обычного механизма межпроцессного взаимодействия Mach.

Когда сообщение поступает на UNIX-сервер, его принимает свободная простаивающая нить, определяет, от какого процесса пришел вызов, извлекает номер системного вызова и параметры, выполняет системный вызов, и, наконец, отсылает ответ. Большинство сообщений соответствует точно одному системному вызову BSD.

Существует один набор системных вызовов, выполняющихся по другому - это вызовы операций ввода-вывода с файлами. Они могли бы выполняться и по описанной схеме, но из-за соображений производительности был реализован другой подход. Когда файл открывается, то он отображается непосредственно в адресное пространство вызывающего процесса, так что библиотека эмуляции получает к нему доступ непосредственно, без необходимости делать вызов RPC сервера UNIX. Например, чтобы выполнить системный вызов READ, библиотека эмуляции определяет место расположения байтов в отображенном файле, которые нужно прочитать, определяет место расположения буфера и просто копирует байты в буфер с максимально возможной скоростью.

Во время цикла копирования может случиться страничный отказ, если некоторые страницы файла не находятся в памяти. Каждый отказ приводит к тому, что Mach посылает сообщение внешнему менеджеру памяти, управляющему отображением файла. Этот менеджер памяти представляет собой нить внутри UNIX-сервера, называемую пейджером i-узла (i-node pager). Менеджер читает с диска нужную страницу файла и отображает ее в адресное пространство прикладной программы. Он также синхронизирует операции над файлами, которые открыты несколькими UNIX-процессами одновременно.

Хотя описанный метод выполнения программ UNIX и кажется запутанным, многочисленные измерения показали, что он работает лучше, чем традиционные монолитные реализации ядра. В дальнейшем работы над Mach будут фокусироваться на разделении сервера UNIX на несколько серверов с более специфическими функциями.

Сетевые продукты фирмы Novell

История и версии сетевой ОС NetWare

Novell - это крупнейшая фирма, которой принадлежит, согласно различным источникам, от 65% до 75% рынка сетевых операционных систем для локальных вычислительных сетей. Наибольшую известность фирма Novell приобрела благодаря своим сетевым операционным системам семейства NetWare. Эти системы реализованы как системы с выделенными серверами.

Основные усилия Novell были затрачены на создание высокоэффективной серверной части сетевой ОС, которая за счет специализации на выполнении функций файл-сервера обеспечивала бы максимально возможную для данного класса компьютеров скорость удаленного доступа к файлам и повышенную безопасность данных. Для серверной части своих ОС Novell разработала специализированную операционную систему, оптимизированную на файловые операции и использующую все возможности, предоставляемые процессорами Intel x386 и выше. За высокую производительность пользователи сетей Novell NetWare расплачиваются стоимостью - выделенный файл-сервер не может использоваться в качестве рабочей станции, а его специализированная ОС имеет весьма специфический API, что требует от разработчиков дополнительных серверных модулей особых знаний, специального опыта и значительных усилий.

Для рабочих станций Novell выпускает две собственные ОС со встроенными сетевыми функциями: Novell DOS 7 с входящей в нее сетевой одноранговой компонентой Personal Ware, а также ОС UnixWare, являющейся реализацией UNIX System V Release 4.2 со встроенными возможности работы в сетях NetWare. (Осенью этого года права на систему UnixWare проданы компании Santa Cruz Operations.) Для популярных ОС персональных компьютеров других производителей Novell выпускает сетевые оболочки с клиентскими функциями по отношению к серверу NetWare.

Первоначально операционная система NetWare была разработана фирмой Novell для сети Novell S-Net, имеющей звездообразную топологию и патентованный сервер с микропроцессором Motorola MC68000. Когда фирма IBM выпустила персональные компьютеры типа PC XT, Novell решила, что NetWare может быть легко перенесена в архитектуру микропроцессоров семейства Intel 8088, и тогда она сможет поддерживать практически все имеющиеся на рынке сети персональных компьютеров.

Первая версия NetWare была выпущена фирмой Novell в начале 1983 года.

В 1985 году появилась система **Advanced NetWare v1.0**, которая расширяла функциональные возможности операционной системы сервера**.**

Версия 1.2 системы Advanced NetWare, выпущенная также в 1985 году, стала первой операционной системой для процессора Intel 80286, работающей в защищенном режиме.

Версия 2.0 системы Advanced NetWare, выпущенная в 1986 году, отличалась от предыдущих версий более высокой производительностью и возможностью объединения разнородных на канальном уровне сетей. Полностью используя возможности защищенного режима процессора 80286, Advanced NetWare обеспечила такую производительность сети, которая была недоступна операционным системам, работающим в реальном режиме и ограниченным 640 Кбайтами памяти. Версия 2.0 впервые обеспечила возможность подключения к одному серверу до четырех сетей с различной топологией, таких как Ethernet, ArcNet и Token Ring.

В 1987 году Novell выпустила систему **SFT NetWare**, в которой были предусмотрены специальные средства обеспечения надежности системы и расширены возможности управления сетью. Такие средства, как учет используемых ресурсов и защита от несанкционированного доступа, позволили администраторам сети определять, когда и как пользователи осуществляют доступ к информации и ресурсам сети. Разработчики впервые получили возможность создавать многопользовательские прикладные программы, которые могут выполняться на сервере в качестве дополнительных процессов сетевой операционной системы и использовать ее функциональные возможности.

Операционная система **NetWare v2.15** появилась на рынке в декабре 1988 года, добавив в NetWare средства поддержки компьютеров семейства Macintosh. У пользователей Macintosh появилась возможность подключать свои компьютеры в качестве клиентов серверов NetWare, получая доступ к ресурсам сети и осуществляя прозрачный поиск и хранение информации на сервере. При этом на пользователей Macintosh распространяются все основные свойства NetWare, включая устойчивость к сбоям и защиту от несанкционированного доступа.

В сентябре 1989 года Novell выпустила свою первую версию 32-разрядной операционной системы для серверов с микропроцессором 80386, которая

получила название **NetWare 386 v3.0**. Она обладала значительно более высокой производительностью по сравнению с предыдущими версиями, усовершенствованной системой защиты от несанкционированного доступа, гибкостью в применении, а также поддержкой различных сетевых протоколов. Она отвечала самым передовым требованиям к среде функционирования распределенных прикладных программ.

В июне 1990 года появилась версия **NetWare 386 v.3.1**, в которой были усовершенствованы средства обеспечения надежности и управления сетью, повышена производительность, улучшены инструментальные средства для независимых разработчиков.

В 1991 году фирмой Novell операционные системы для процессоров 80286 (SFT, Advanced и ELS NetWare) были заменены на более мощную и удобную систему **NetWare v2.2**, функционально превосходящую предыдущие версии 2.1x.

Одновременно была выпущена система **NetWare v3.11**, существенно расширившая возможности NetWare 386. NetWare v3.11 стала первой сетевой операционной системой, обеспечивающей доступ к сетевым ресурсам с рабочих станций DOS, Windows, OS/2, UNIX и Macintosh.

В 1993 году после длительных испытаний начались поставки системы **NetWare SFT III v3.11**. NetWare System Fault Tolerance Level III (SFT III) v3.11 - сетевая операционная система, разработанная специально для использования в системах, требующих наивысшего уровня надежности. В дополнение к средствам повышения надежности, имеющимся в составе NetWare v3.11, SFT III обеспечивает работу двух серверов в "зеркальном" режиме. При этом один из серверов всегда является активным, а второй находится в горячем резерве, обеспечивая у себя такое же состояние памяти и дисков, как и у основного сервера.

В 1993 году фирма Novell выпустила ОС **NetWare v4.0**, явившуюся во многих отношениях революционно новым продуктом. Эта система была разработана специально для построения вычислительных сетей "масштаба предприятия" с несколькими файл-серверами, большим количеством сетевых ресурсов и пользователей. Одним из основных нововведений явилась служба каталогов NetWare Directory Services (NDS), хранящая в распределенной по нескольким серверам базе данных информацию о всех разделяемых сетевых ресурсах и пользователях, что обеспечило возможность при одном логическом входе в систему получать прозрачный доступ ко всем ресурсам многосерверной сети.

В сентябре 1993 года Novell выпустила систему **NetWare v3.12**, представляющую собой усовершенствованный вариант самой популярной сетевой ОС фирмы Novell - NetWare v3.11. В версии NetWare 3.12 были устранены замеченные за время эксплуатации версии NetWare 3.11 ошибки и добавлены новые средства: усеченная версия электронной почты Global MHS, средства поддержки клиентов Macintosh и клиентская оболочка для DOS и Windows по технологии VLM,

позволяющая динамически загружать и выгружать необходимые для рабочей станции сетевые компоненты.

Последней на сегодня версией NetWare является ориентированная на корпоративное использование сетевая операционная система **NetWare v4.1**.

Версии 4.0, 4.01 и 4.02

Версии ОС NetWare 4.x существенно отличаются от версий семейства 3.x их очевидной корпоративной направленностью. Если версии 3.x были рассчитаны на сети небольших и средних предприятий, то уже первая ОС из нового семейства - NetWare 4.0 - имеет много новых свойств, направленных на обеспечение успешной работы в больших гетерогенных сетях. Версии 4.01 и 4.02 лишь незначительно отличаются от версии 4.0 за счет небольших улучшений в глобальной службе каталогов, которая оказалась в версии 4.0 не слишком удобной для использования, что и привело к весьма медленному старту ОС нового поколения. Кроме уже отмеченной глобальной службы каталогов, в этих версиях улучшены:

- система управления оперативной памятью - уменьшилась фрагментация в процессе динамической выгрузки и загрузки модулей NLM;
- система управления внешней памятью - уменьшена фрагментация дисков, а также появились средства прозрачной компрессии дисковых разделов и прозрачной миграции файлов с диска на стриммер и обратно;
- система управления сетью;
- система безопасности;
- транспортная система.

Глобальная служба справочников сетевых ресурсов

Главным отличием ОС NetWare v 4.0x от предыдущих версий является введение единого для всех файл-серверов сетевого каталога (справочника сетевых ресурсов) - NetWare Directory Services (NDS), имеющего иерархическую древовидную структуру и основанного на международном стандарте X.500. В предыдущих версиях NetWare база данных сетевых ресурсов, называемая Bindery, была уникальна для каждого файл-сервера. Поэтому для получения доступа к нужным ресурсам пользователь должен был подключаться к предоставляющему этот ресурс файл-серверу.

В NetWare v 4.0x все сетевые ресурсы, такие как файлы, принтеры, прикладные программы и т.д. составляют единую логическую сущность, не зависящую от их физического размещения. Пользователю достаточно один раз подключиться к сети, чтобы получить доступ ко всем ее ресурсам, которыми он имеет право пользоваться. Пользователи и прикладные программы, которые обращаются к NDS для получения доступа к необходимым ресурсам, могут и не знать, как распределены эти ресурсы по серверам и подсетям (в отличие от предыдущих версий, где эти ресурсы были жестко "привязаны" к серверам). В NetWare 4.02

значительно расширена и улучшена служба NDS по сравнению с предыдущими версиями NetWare 4.x. Изменена процедура установки NDS, а для простой или не очень сложной структуры описания ресурсов сети обеспечен автоматический режим установки.

Управление оперативной и дисковой памятью сервера

Новые средства управления оперативной памятью сервера, используемые в NetWare 4.0x, значительно улучшают, по сравнению с предыдущими версиями NetWare, использование оперативной памяти сервера. После выгрузки NLM оперативная память освобождается более эффективно, поэтому многократная загрузка и выгрузка NLM в NetWare 4.0x не приводит к накоплению так называемого "мусора" в оперативной памяти сервера. В NetWare 4.0x обеспечивается также защита областей оперативной памяти, используемой загружаемыми модулями, от искажения в результате работы других модулей. Такое искажение может возникнуть при использовании некорректно написанных NLM (например, разрабатываемых самим пользователем). Защита оперативной памяти сервера позволяет снизить вероятность отказов сети при тестировании новых NLM.

NetWare 4.0x позволяет значительно экономить дисковое пространство серверов. Это достигается возможностью автоматического переноса редко используемых файлов с дисковых накопителей на ленточные и оптические накопители большой емкости (доступ к файлам при этом не изменяется). Другая возможность - компрессия файлов на дисковых накопителях, выполняемая в фоновом режиме.

В NetWare 3.x информация записывается на диск только целыми блоками, минимальный размер которых составляет 4 К. NetWare 4.0x может распределять дисковое пространство не только целыми блоками, но и подблоками размером 512 байт. Эта функция, называемая Disk Suballocation, позволяет существенно уменьшить потери дискового пространства.

NetWare v.4.0x позволяет подключать накопители большой емкости, такие как CD-ROM, WORM, перезаписываемые оптические диски и другие, непосредственно к файловой системе NetWare в качестве томов.

Управление сложной сетью

NetWare 4.x обеспечивает широкий набор функций по управлению сложной сетью, позволяющих контролировать доступ к файлам, каталогам, очередям, томам, генерировать разнообразные отчеты о событиях, происходящих в сети. Новая утилита "NetWare Administrator", работающая в среде Windows 3.1 или OS/2 2.x, обеспечивает графический интерфейс для управления сетью. В версии 4.02 эта утилита проще устанавливается и примерно втрое быстрее работает, чем в предыдущих версиях NetWare 4.0x. Появилась также возможность вывода дерева сетевого каталога на печать.

Улучшения в системе безопасности

Версии 4.0x, как и версии 3.x, имеют многоуровневую систему защиты каталогов и файлов, а также осуществляют контроль доступа пользователей к сети. В NetWare 4.0x предусмотрены дополнительные уровни безопасности. В частности, в предыдущих версиях обязательно есть пользователь с именем Supervisor, обладающий всеми административными полномочиями в сети. Поэтому постороннему лицу достаточно подобрать пароль, чтобы затем войти в сеть под именем Supervisor. В версиях NetWare 4.0x существует возможность назначить администратору сети любое имя, что уменьшает риск входа взломщика в сеть под видом администратора.

Кроме того, в NetWare 4.0x используется новая технология передачи пароля по сети, основанная на разделении ключей. При входе пользователя в сеть сервер направляет рабочей станции запрос на идентификацию, зашифрованный с помощью пароля пользователя, случайного ключа и личного ключа пользователя. Рабочая станция расшифровывает этот запрос, используя случайный ключ и пароль, и получает значение личного ключа пользователя, который в дальнейшем используется при доступе ко всем сетевым ресурсам. Таким образом, ни личный ключ пользователя, ни пароль не передаются в явном виде по сети, что исключает возможность их перехвата и подделки.

Еще одна отличительная особенность версий 4.0x, повышающая безопасность - возможность контролировать изменения в NDS и файловой системе. Пользователь-аудитор может, например, следить за тем, кто создает и модифицирует сетевые объекты, кто и как использует те или иные файлы.

Улучшения в транспортной системе

Штатные средства NetWare 4.0x обеспечивают режимы ускорения передачи информации в сети: режим форсированной передачи пакетов Packet Burst Mode, когда пакеты передаются группами без подтверждения приема на каждый переданный пакет, и режим передачи межсетевых пакетов большого размера без их разбиения на сегменты.

В программном обеспечении рабочей станции NetWare 4.02 улучшены утилиты печати, утилита просмотра электронной документации, изменена система службы управления сохранением файлов Storage Management Services (SMS), упрощена и убыстрена процедура установки CD-ROM.

Версия NetWare 4.1

Версия 4.1 является последней выпущенной в свет версией NetWare семейства 4.x на момент написания данного пособия.

Некоторые обозреватели считают, что версия 4.1 - эта та версия, которой должна была быть версия 4.0, имея в виду многочисленные неудобства в реализации

службы NDS и некоторые другие недочеты, помешавшие версиям 4.0x завоевать рынок (на конец 1994 года только 31% пользователей NetWare в мире работало с различными версиями 4.0x). Теперь, с выходом версии 4.1, положение может существенно измениться.

Во-первых, Novell значительно упростила процедуру инсталляции.

Во-вторых, возможности версии 4.1 существенно расширены. Как и в NetWare 4.0x, здесь поддерживается улучшенная файловая система и средства управления памятью, увеличено максимальное число обслуживаемых пользователей (свыше 250), реализованы службы сжатия и перемещения редко используемых файлов, а также более совершенные средства защиты информации и управления правами пользователей.

Новыми элементами NetWare 4.1 являются средства IPX Mac и NetWare IP, интегрированная с NDS служба обработки сообщений MHS и очень полезная программа DS-Standard фирмы Preffered Systems, облегчающая конфигурирование NetWare и переход от одной версии к другой.

В-третьих, фирма Novell снизила цену на свой продукт и предусмотрела различные варианты его лицензирования.

Упрощенная процедура инсталляции при использовании программы DS-Standard

При переходе с NetWare 3.x на NetWare 4.1 Novell рекомендует вместо собственных средств использовать программу DS-Standard. Главное преимущество этого продукта состоит в том, что он позволяет собирать информацию с существующих серверов и дает возможность администратору создавать и конфигурировать NDS-дерево в автономном режиме. Как только администратор решает, что полученная структура дерева его устраивает, DS-Standard экспортирует все данные о новой структуре в справочник NDS. Работа с DS делится на три основных этапа: сбор информации, моделирование и конфигурирование. На первом этапе осуществляется сбор информации с существующих серверов 3.x и 4.x., при этом могут быть собраны все сведения, включая учетные ограничения, данные об эквивалентности прав доступа, конфигурации заданий на печать, доверенные права, сценарии входа в сеть, ограничения на станцию, а также информация о системе печати. Единственное, что не импортируется с серверов 3.x - это пароли пользователей (средства миграции Novell импортируют пароли).

После сбора программой DS-Standard данных с серверов, администратор может приступить к моделированию дерева NDS. Программа позволяет добавлять, перемещать и удалять объекты, а также изменять их свойства. В этом отношении она во многом напоминает утилиту NetWare Admin.

После завершения автономного моделирования, DS-Standard формирует новую конфигурацию, внося изменения в "живое" дерево NDS.

С помощью процедуры инсталляции можно перейти на NetWare 4.1 на всех серверах сети, причем каждый сервер получает свой собственный контекст. Такой контекст представляет собой отдельную ветвь дерева NDS. Позднее можно воспользоваться специальными средствами для удаления и перемещения ветвей.

В NetWare 4.1 расширены функции эмуляции bindery. В предыдущей версии процедура эмуляции могла использовать только одну ветвь дерева, а в 4.1 - до 16 ветвей.

В состав новой версии включена утилита NetSync, позволяющая управлять с сервера 4.1 12-ю серверами NetWare 3.x. При инсталляции NetSync на сервер NetWare 3.x вся информация из его базы bindery копируется в контекст bindery сервера NetWare 4.1, после чего можно управлять сервером NetWare 3.x с помощью утилит NetAdmin NWAdmin версии 4.1. Любые изменения, сделанные на сервере NetWare 4.1, автоматически переносятся на сервер NetWare 3.x. Недостаток такой системы состоит в том, что, если на сервере 3.x пользователь изменяет свой сценарий входа в сеть, то внесенные им изменения не переносятся автоматически на сервер 4.1.

Конфигурирование NDS

В версии NetWare 4.1 появились, наконец, средства для удаления, перемещения и переименования ветвей дерева NDS. Это повышает гибкость системы, поскольку вовсе не обязательно строить дерево в окончательном виде с первой попытки.

Несколько деревьев можно объединить в одно с помощью утилиты DSMerge. Раньше приходилось проектировать дерево NDS в масштабах всего предприятия, что для большинства организаций было очень неудобно. Сегодня каждый отдел предприятия может самостоятельно строить свои деревья, чтобы позднее слить их в единое дерево NDS.

Усовершенствованные средства передачи сообщений

Служба сообщений MHS в версии 4.1 теперь тесно интегрирована со службой NDS и включена в комплект поставки. Это позволило, во-первых, значительно уменьшить размер модулей NLM MHS (примерно с 2 Мбайт до примерно 200 Кбайт) и, во-вторых, обеспечить в системе поддержку только одной базы данных с информацией о пользователях - NDS.

В комплект поставки MHS не включены шлюзы к другим почтовым системам, поэтому их необходимо приобретать отдельно, причем для каждого дерева NDS потребуется свой шлюз.

Поддержка клиентских станций

Novell улучшила оболочку для клиентов на основе компьютеров Macintosh, заменив прежнюю 5-пользовательскую версию, поставлявшуюся с NetWare 3.x и 4.02, "неограниченной" версией NetWare for Macintosh. Теперь компьютеры Macintosh могут легко подключаться к сети NetWare и работать с NDS в качестве клиента. Правда утилиты администрирования NDS для Mac пока не созданы, хотя Novell и собирается из выпустить в будущем на основе технологии OpenDoc фирмы Apple.

Для клиентов NetWare выпущена новая версия оболочки на основе VLM-технологии - версия 1.2, в которой устранены ошибки первой версии.

Поддержка коммуникационных протоколов

В комплект поставки NetWare 4.1 фирма Novell включила купон на получение NetWare/IP. При наличии NetWare/IP можно использовать протоколы TCP/IP для передачи сообщений прикладного протокола клиент-сервер NetWare - протокола NCP. На клиентских станциях, работающих под управлением DOS, загрузка стека протоколов TCP/IP требует лишь небольшого дополнительного объема памяти.

В версии 4.1 можно использовать новых протокол обмена маршрутной информацией NLSP вместо протокола RIP. Протокол NLSP основан на алгоритме "состояния связей" (Link State Algorithms), хорошо работающем в сетях больших размеров за счет того, что маршрутизаторы обмениваются только информацией о изменениях в состоянии связей с ближайшими соседями, что существенно уменьшает служебный трафик по сравнению с дистанционно-векторными протоколами, к которым относится протокол RIP. К протоколам "состояния связей" относится и сравнительно новый протокол OSPF, разработанный как часть стека Internet.

Одновременно с разработкой протокола NLSP фирма Novell предприняла и другие шаги по улучшению своего стека протоколов в условиях работы в крупных сетях с глобальными связями. Ведутся работы по улучшению работы протокола IPX в глобальных сетях, при поддержке всеми узлами сети службы NDS отпадает необходимость в другом широковещательном протоколе - протоколе SAP.

Концепции построения NetWare

Структура NetWare и обзор особенностей

NetWare - это специализированная операционная система, а не ОС общего назначения. ОС общего назначения обеспечивают сервис, который удовлетворяет потребностям многих различных приложений, к тому же такая ОС обычно очень устойчива к поведению своих приложений за счет специальных ограничительных мер. Приложения могут разрабатываться почти без заботы о их

взаимодействии с другими программами. Они также могут быть написаны без учета фактора разделения ресурсов компьютера, таких как память или CPU.

В ОС общего назначения проблемы взаимодействия, разделения ресурсов и т.д. решаются операционной системой. Приложениям, которые пытаются решать их самостоятельно, ОС может запретить это делать. Это обеспечивает некоторый уровень защиты приложений и ОС.

NetWare - это специализированная ОС, которая с самого начала проектировалась для оптимизации сетевого сервиса и, в первую очередь, доступа к удаленным файлам. Такие приложения, как электронные таблицы и текстовые процессоры, будут лучше работать под управлением ОС общего назначения, а приложения типа сервера печати, сервера баз данных и коммуникационного сервера, которые обеспечивают управление разделяемыми ресурсами, будут лучше работать под NetWare. Но, чтобы добиться такого эффекта, приложения для NetWare нужно писать тщательно, осознавая последствия их совместной работы на сервере, чтобы одно приложение не подавляло другие из-за слишком интенсивного захвата процессорного времени.

Кроме повышения производительности - основной цели разработки семейства ОС NetWare 3.x и 4.x, разработчики ставили перед собой цели создания открытой, расширяемой и высоконадежной операционной системы, обеспечивающей высокий уровень защиты информации.

Способы повышения производительности

Плоская модель памяти

NetWare работает в защищенном режиме CPU (protected mode), используя все преимущества 386, 486 процессоров и Pentium, связанные с 32-разрядной адресацией памяти.

В защищенном режиме память адресуется непрерывным диапазоном адресов. Эта так называемая "плоская" (flat) модель памяти делает управление памятью более удобным и гибким. В этом случае нет необходимости переключать сегменты памяти, так как вся память состоит из одного сегмента. При работе в "реальном" режиме CPU отдельная операция по выделению памяти ограничена размером 64 К, так как 64 К - это максимальный размер сегмента. Работа в 32-разрядном режиме значительно повышает скорость выполнения всех компонентов и модулей ОС.

Нити и невытесняющая многозадачность

Другим преимуществом защищенного режима является возможность выполнять несколько программ одновременно. Часто это называют многозадачностью (multitasking). В NetWare реализован механизм "нитей" (thread), который позволяет использовать все преимущества расщепления одного процесса на

несколько параллельно выполняемых нитей. Этот механизм описан в разделе 1.2.4 главы 1. NetWare обеспечивает удобные средства для реализации многонитевых процессов.

Существует несколько вариантов реализации алгоритма диспетчирования нитей. NetWare использует метод невытесняющей многозадачности (nonpreemptive multitasking). Это означает, что обычно невозможно прерывание приложений и их нитей другими приложениями и нитями. Иногда этот метод называют "окружением хороших парней", так как ожидается, что приложения будут вести себя вежливо по отношению к системным ресурсам. Фактически, если приложение не отдает периодически управление CPU, чтобы дать возможность другим приложениям выполняться, то будет работать только это приложение. Следовательно, при работе в таком режиме очень важно понимать последствия захвата CPU и быть "хорошим парнем" среди равных. Главным же преимуществом невытесняющей многозадачности является более быстрое переключение с нити на нить по сравнению с вытесняющей многозадачностью (preemptive multitasking), когда нить процесса прерывается в неожиданный и часто неудобный для нее момент времени, и ОС приходится сохранять гораздо больше информации о прерванном состоянии нити, чем в случае, когда нить сама отдает управление ОС.

Из-за того, что NetWare использует режим невытесняющей многозадачности, она не очень заботится о управлении поведением нитей, которые выполняются. NetWare хранит информацию о том, какая нить выполняется, с каким приоритетом и как долго это происходит, но навязывает нитям свои ограничения только в экстремальных ситуациях. Обычно NetWare считает, что все нити справедливо разделяют процессор, достаточно часто отдавая ему управление. Это позволяет NetWare самой работать более эффективно.

Кэширование диска

Вся оперативная память, оставшаяся после загрузки ОС и дополнительных модулей, используется для кэширования диска, что, файлам при соответствующих размерах оперативной памяти, естественно, существенно повышает скорость обращения к дисковым.

Элеваторный поиск

В ОС NetWare предусмотрен отдельный процесс чтения с диска, который считывает данные с жестких дисков сервера и размещает их в кэш-буферах. Этот процесс сортирует поступающие запросы на чтение и располагает их в порядке приоритетов, в зависимости от текущего положения головок дисковода. Такой метод обслуживания запросов, называемый элеваторным поиском (elevator seeking), оптимизирует перемещение головок и в результате позволяет значительно увеличить пропускную способность дисковой подсистемы при большой интенсивности запросов.

Параллельный поиск

Если на сервере имеется несколько дисковых каналов, то NetWare может параллельно осуществлять поиск данных на нескольких дисках (по одному диску на канал). Это существенно повышает производительность.

Способы обеспечения открытости и расширяемости

Все сетевые сервисы, утилиты сервера или работающие на сервере приложения выполнены в NetWare в виде загружаемых модулей - NetWare Loadable Modules, NLMs, которые могут динамически загружаться и выгружаться в любое время без остановки сервера. Структура ОС NetWare приведена на рисунке 7.1.

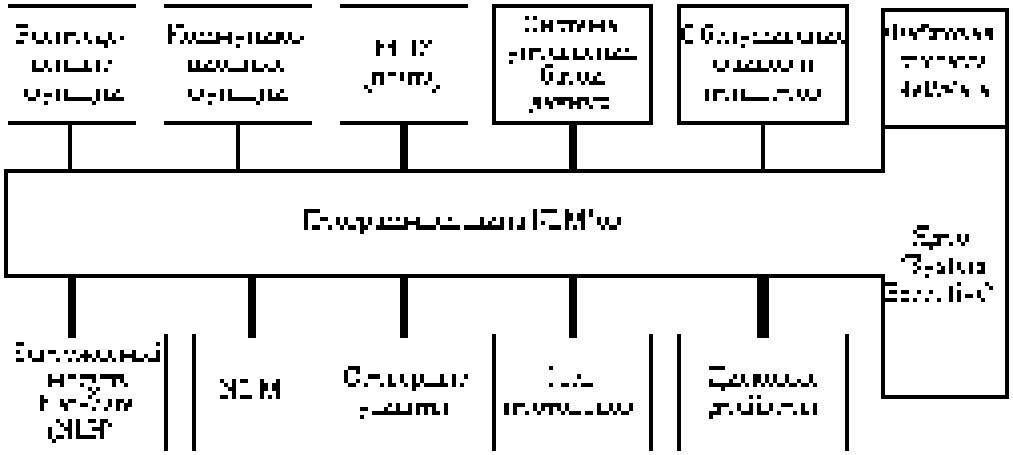


Рис. 7.1. Структура ОС NetWare

Ядро системы, называемое System Executive, выполняет базовые задачи ОС по управлению памятью, планированию и диспетчированию нитей, управлению файловой системой, также поддерживает программную шину для интерфейса NLM'ов. Каждый NLM выполняет либо функции операционной системы (драйвер диска или сетевого адаптера, утилита пространства имен, файловый сервер или модуль почтового сервиса), либо является пользовательским модулем, реализующим дополнительный сетевой сервис - например, сервис SQL-сервера или сервера печати. Для ядра системы все модули NLM равноправны, поэтому расширение или сужение функций системы осуществляется путем добавления или выгрузки соответствующего NLM'а.

Novell обеспечивает расширяемость системы NetWare за счет предоставления программистам набора инструментальных средств и строго описанных интерфейсов API для разработки собственных NLM-приложений, использующих все возможности 32-разрядного окружения. В настоящее время существует большое количество программных систем третьих фирм, реализованных в виде NLM-приложений, для серверов NetWare - серверы баз данных, коммуникационные серверы и т.п.

Открытость ОС NetWare обеспечивается поддержкой ею наиболее популярных стеков протоколов в строгом соответствии с существующими стандартами. NetWare поддерживает такие популярные сетевые протоколы, как IPX/SPX, TCP/IP, Apple Talk, и средства их мультиплексирования, такие как STREAMS и TLI. Стандарт ODI позволяет независимым разработчикам сетевых адаптеров легко включать свои NLM-драйверы в состав серверной ОС NetWare. Кроме того, фирма Novell разработала для NetWare большое количество программных средств - шлюзов к другим широко распространенным сетям, таким, как сети Internet и SNA.

Способы обеспечения надежности

В системах NetWare предусмотрен ряд функций, обеспечивающих надежность системы и целостность данных. Ниже перечислены функции, которые обеспечивают защиту всех частей сервера: от устройств хранения данных до критичных файлов прикладных программ. Наличие таких функций позволяет NetWare обеспечить очень высокий уровень надежности сети.

Средства обеспечения надежности SFT I:

- *Проверка чтением после записи*. После записи на диск каждый блок данных немедленно считывается в память для проверки читаемости. Первоначальное содержание блока не стирается до окончания проверки. Если данные не читаются, они записываются в другой блок диска, а первый блок помечается как дефектный.
- *Дублирование каталогов*. NetWare хранит копию структуры корневого каталога. Если портится основная структура, то начинает использоваться резервная.
- *Дублирование таблицы размещения файлов FAT*. NetWare хранит копию FAT и использует ее при порче основной таблицы FAT.
- *Оперативное исправление 1 (Hot Fix 1).* Запись или перезапись данных из дефектных блоков в специальную резервную область диска.
- *Контроль UPS.*

Средства обеспечения надежности SFT II:

- Зеркальное отображение дисков, подключенных к одному дисковому контроллеру (Disk Mirroring).
- Дуплексирование дисков, подключенных к различным дисковым контроллерам (Disk Duplexing).
- Оперативное исправление 2 (Hot Fix 2). Повторное выполнение операции чтения на отраженном диске и запись данных в резервную область.
- Система отслеживания транзакций (TTS).

Средства обеспечения надежности SFT III заключаются в полном динамическом зеркальном отображении двух серверов, которые могут находится на

значительном удалении друг от друга (при использовании оптоволоконного кабеля для межсерверной связи - до 4 км).

Средства обеспечения надежности уровней SFT I и SFT II реализованы в NetWare v3.11, NetWare v3.12 и NetWare v4.x. Уровень надежности SFT III реализован пока только в NetWare SFT III v3.11.

Защита информации

Средства защиты информации встроены в NetWare на базовых уровнях операционной системы, а не являются надстройкой в виде какого-либо приложения. Поскольку NetWare использует на файл-сервере особую структуру файлов, то пользователи не могут получить доступ к сетевым файлам, даже если они получат физический доступ к файл-серверу.

Операционные системы NetWare содержат механизмы защиты следующих уровней:

- защита информации о пользователе;
- защита паролем;
- защита каталогов;
- защита файлов;
- межсетевая защита.

В 1983 году фирма Novell ввела в систему концепций локальной сети понятия имени пользователя, пароля и характеристики пользователя (user profile). Характеристика пользователя содержит перечень ресурсов, к которым пользователь имеет доступ, и права, которыми он обладает при работе с этими ресурсами. Администратор сети может ограничить права пользователя по входу в сеть датой, временем и конкретными рабочими станциями. Средства обнаружения нарушений защиты и блокировки действий нарушителя извещают администратора сети о попытках несанкционированного доступа.

В версии NetWare 3.12 пароли хранятся на сервере в зашифрованном виде. Пароль, задаваемый пользователем, передается по кабелю также в зашифрованном виде, что обеспечивает защиту от попыток узнать пароль путем "прослушивания" сети.

В версии NetWare 4.x использована более надежная схема идентификации пользователя при логическом входе в сеть, основанная на использовании технологии защиты RSA public key/private key. При использовании этой технологии пароль и личный ключ пользователя никогда не передаются по кабелям, что полностью исключает возможность узнать чужой пароль. В службу каталогов NDS также введен новый уровень управления доступом, который может быть введен в действие администратором в любой части сети.

С точки зрения защиты ОС NetWare не делает различия между операционными системами рабочих станций. Станции, работающие под управлением DOS, Windows, OS/2, Macintosh и UnixWare, обслуживаются совершенно одинаково, и все функции защиты применяются ко всем операционным системам, которые могут использоваться в сети NetWare.

Управление процессами

Каждый NLM стартует в ОС NetWare как по крайней мере одна нить (ее можно считать процессом в традиционном понимании этого термина), которая создается неявным образом при запуске функции main( ) программы, но однонитевые процессы в NetWare являются редким исключением. Функциональные особенности файл-сервера предопределяют большие выгоды при расщеплении процесса обслуживания параллельных запросов к файлам, поступающих одновременно от нескольких пользователей, на несколько нитей. Концепция нитей упрощает программирование этого параллелизма, так как подпроцесс обслуживания одного запроса внутри одной нити программируется как последовательный процесс, то есть процесс, в котором все стадии выполнения протекают в естественной последовательности. Параллелизм же выполнения нескольких запросов реализуется на уровне планировщика операционной системы. NetWare сама запускает много внутренних процессов, таких как процесс обновления кэша или процесс обработки командной строки консоли, которые являются многонитевыми. Все нити одного NLM разделяют общее адресное пространство.

При переключении нитей операционная система использует контекст нити, который является мгновенным снимком рабочей среды нити. В общем случае в контекст нити входят различные переменные, содержание экранов, указатели, счетчики, ссылки, управляющая информация и другие параметры. При переключении с одной нити на другую контекст снимаемой с выполнения нити запоминается ОС, а контекст запускаемой нити восстанавливается. Время переключения во многом зависит от размера контекста, и алгоритм невытесняющей многозадачности работает быстрее вытесняющего алгоритма и из-за того, что в первом случае контекст нити или процесса (в ОС, поддерживающих только концепцию процесса) имеет меньшие размеры.

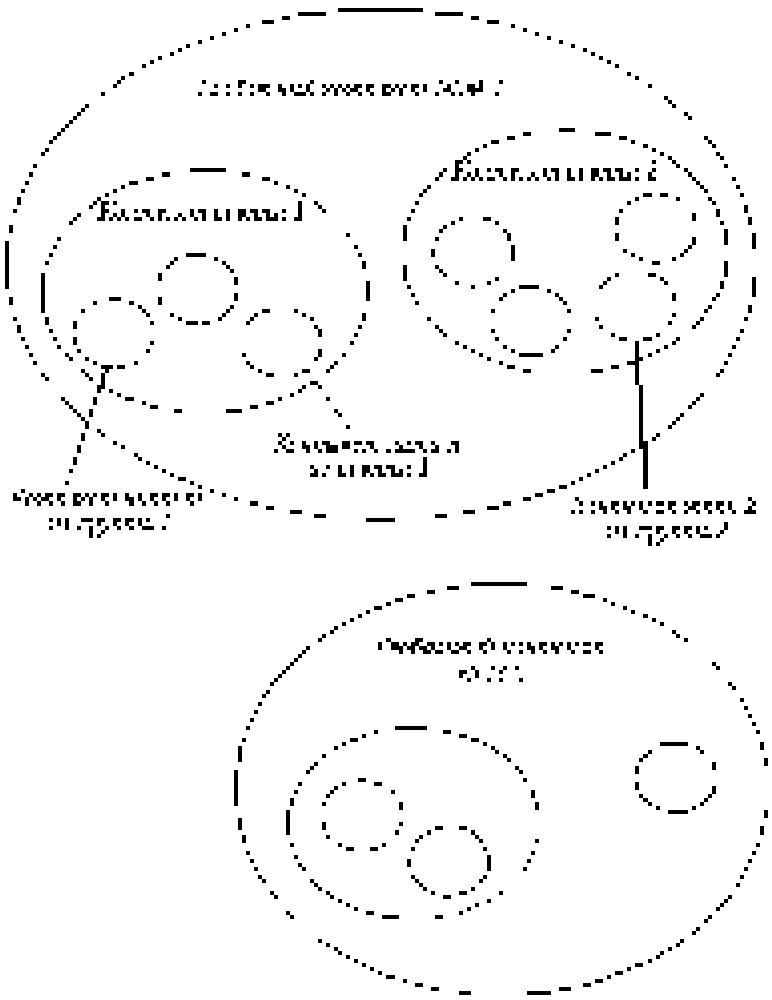


Рис. 7.2. Соотношение между глобальным, групповым и индивидуальным контекстами нитей в NetWare

В среде NetWare различается три вида контекстов нитей: глобальный контекст, контекст группы нитей и контекст отдельной нити. Эти контексты вложены друг в друга, как это показано на рисунке 7.2. Соотношение между данными этих контекстов напоминает соотношение глобальных и локальных переменных в программе, написанной на языке С. Глобальный контекст является общим для всех нитей данного NLM'a, и все его переменные видны для всех его нитей. В NetWare можно создавать несколько групп нитей внутри одного NLM-процесса, и эти группы имеют свой групповой контекст. Все нити группы видят переменные контекста своей группы, но не видят переменных контекста другой группы. И, наконец, каждая отдельная нить имеет свой собственный контекст. Содержимое этого контекста доступно только для данной нити.

Очевидно, что такая иерархическая организация контекстов ускоряет переключение нитей, так как при переключении с нити на нить в пределах одной группы нет необходимости заменять контексты групп или глобальные контексты, достаточно лишь заменить контексты нитей, которые меньше по объему, чем контексты других видов. Аналогично, при переключении с нити одной группы на нить другой группы в пределах одного NLM глобальный контекст не изменяется,

а заменяется только контекст группы. Переключение же глобальных контекстов происходит только при переходе с нити одного NLM на нить другого NLM.

Программный код в NetWare для работы с нитями может пользоваться различными библиотечными функциями, такими как:

- *BeginThread* - создать новую нить.
- *BeginThreadGroup* - создать новую группу нитей, которая содержит одну нить. Функция аналогична BeginThread, но нить приписывается к новой группе нитей.
- *ThreadSwitch*- с помощью этой функции нить отдает управление планировщику ОС, который выполняет переключение на новую нить. Нить, вызвавшая эту функцию, считает себя готовой к дальнейшему выполнению, но отдает управления для того, чтобы могли выполняться и другие нити.
- *ThreadSwitchWithDelay* - функция аналогична предыдущей, но нить считает, что она будет готова к выполнению только через определенное количество переключений с нити на нить.
- *Delay* - функция, аналогичная предыдущей, но задержка дается в миллисекундах.
- *ThreadSwitchLowPriority* - функция передачи управления, отличается от ThreadSwitch тем, что нить просит поместить ее в очередь готовых к выполнению, но низкоприоритетных нитей.
- *SheduleWorkToDo* - вместо создания новой нити для выполнения определенной работы (выраженной функцией языка С), поручает эту работу уже созданной заранее нити из резервного пула нитей ОС NetWare, который создается при старте системы для системных целей и срочных работ NLM'ов. Эта функция появилась только в версии NetWare 4.0.

Кроме этих функций NetWare предоставляет средства синхронизации нитей с помощью семафоров и сигналов.

Планировщик NetWare использует несколько очередей для организации невытесняющей дисциплины обслуживания нитей (рисунок 7.3).

При создании нити с помощью функций BeginThread или BeginThreadGroup нить попадает в конец очереди RunList, которая содержит готовые к выполнению нити. После того, как выполнявшаяся на CPU нить завершает свою очередную итерацию с помощью одного из вызовов передачи управления (или вызова ожидания на семафоре), планировщик выбирает для выполнения стоящую первой в очереди RunList нить и запускает ее на выполнение. Нить, завершившая свою очередную итерацию, помещается в конец одной из очередей в зависимости от того, какой вызов передачи управления она использовала: в конец очереди RunList при вызове ThreadSwitch, в конец очереди DelayedWorkToDoList при вызовах ThreadSwitchWithDelay или Delay или же в конец очереди LowPriorityRunList при вызове ThreadSwitchLowPriority. Если же нить вообще

завершила свою работу, выполнив функцию return() в главной функции нити (при создании нити в качестве параметра указывается функция, которая является главной функцией нити), то данная нить уничтожается.

Нити, находящиеся в очереди DelayedWorkToDoList, после завершения условия ожидания перемещаются в конец очереди RunList.

Нити, находящиеся в очереди LowPriorityRunList, запускаются на выполнения только в том случае, когда очередь RunList пуста. Обычно в эту очередь назначаются нити, выполняющую несрочную фоновую работу.

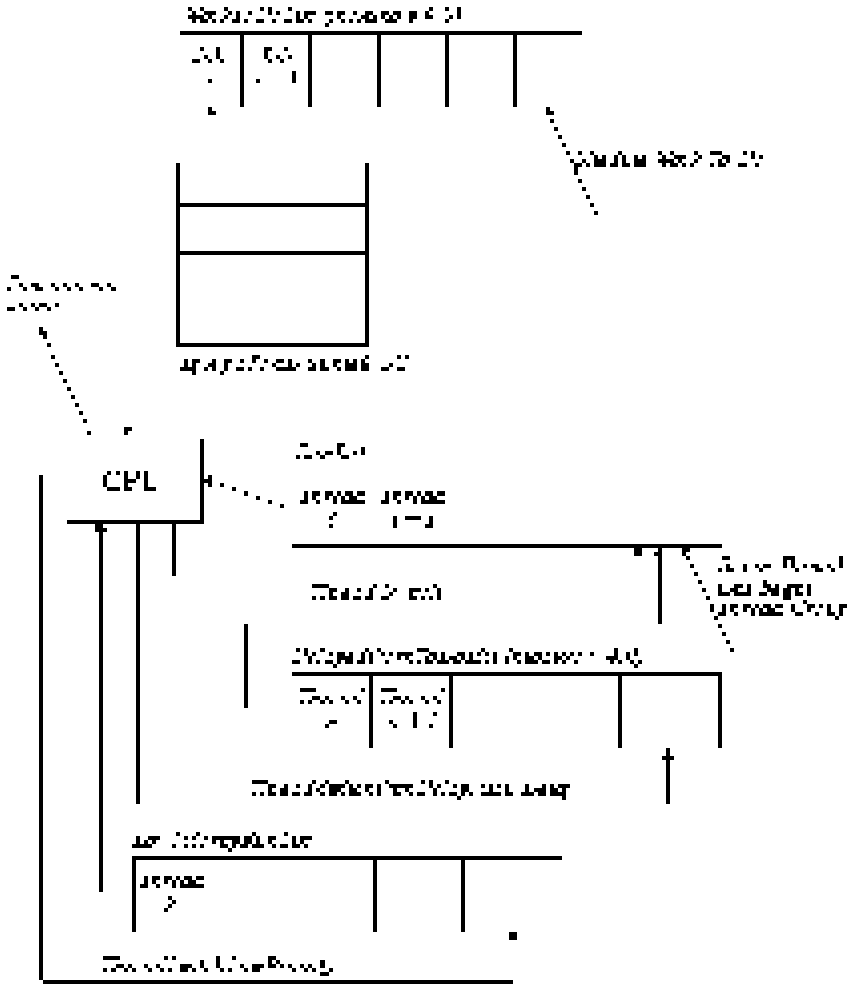


Рис. 7.3. Система очередей планирования NetWare

Очередь WorkToDoList является в системе самой приоритетной. Рабочие нити ОС выбирают работы из этой очереди, и эти нити обладают наивысшим приоритетом, то есть попадают на выполнение перед нитями из очереди RunList. Рабочие нити должны использоваться для выполнения очень срочных работ. Планировщик разрешает выполниться подряд только определенному количеству нитей из очереди WorkToDoList, а затем запускает нить из очереди RunList. Очередь WorkToDoList и связанные с ней функции, появившиеся в версии NetWare 4.0, значительно повышают производительность NLM-приложений.

Описанный механизм организации многонитевой работы в ОС NetWare v3.x и NetWare 4.x в сочетании со средствами синхронизации нитей (семафоры и сигналы) представляет собой современный подход к организации параллелизма и многопоточной обработки. Этот подход потенциально очень производителен, так как отличается небольшими накладными расходами ОС на диспетчирование нитей за счет простых алгоритмов планирования и иерархии контекстов Но для достижения высокой производительности к разработчикам NLM-приложений предъявляются высокие требования, так как распределение процессорного времени между различными NLM будет зависеть в конечном счете от реализации приложения и способа использования описанных в этом разделе средств. Кроме того, общая производительность сервера будет определяться всем набором выполняемых на нем NLM'ов и их взаимной способностью к сосуществованию.

Файловая система

Файловая система NetWare значительно отличается от файловых систем ОС общего назначения следующими ключевыми свойствами:

- в ней предприняты дополнительные меры по сохранению целостности данных;
- достигнута высокая производительность;
- обеспечена емкость файловых систем класса мейнфреймов;
- обеспечивается широкий набор функций файловых API для серверных приложений.

Файловая система NetWare 4.x обратно совместима с файловой системой NetWare 3.x, но имеет несколько новых свойств, включая интерфейс монитора файловой системы.

Тома и жесткие диски

Том - это первичная структура данных файловой системы NetWare. Том включает физическое хранилище данных, логическую информацию о файлах (файлы и каталоги), информацию пространства имен (Name Space) для поддержки не-DOS'овских форматов файлов и системы отказоустойчивости - систему оперативного исправления (Hot Fix) и систему отслеживания транзакций (TTS).

Сервер NetWare 3.12 или 4.x может иметь до 64 томов, монтируемых одновременно. Каждый том может обеспечивать хранение до 32 ТБ (терабайт), если сервер имеет достаточный кэш для хранения структур данных тома, включая FAT (File Allocation Table) тома.

Том NetWare - это аналог понятия "файловая система" в UNIX. То есть том можно монтировать и демонтировать, как и файловую систему UNIX. Однако внутренняя структура тома NetWare существенно отличается от структуры файловой системы UNIX.

Физическая структура тома

Физический носитель, который доступен для приложений с помощью средств тома NetWare, состоит из блоков. Блок тома соответствует последовательности секторов физического носителя. Стандартный размер блока тома - 4K (8 секторов), но возможны блоки и больших размеров. Том NetWare - это массив блоков, а каждый блок - это массив секторов.

Блоки тома должны быть связаны с реальным физическим носителем. Этот носитель состоит из сегментов областей физического носителя, которые являются разделами (partitions), подготовленными для использования как части тома NetWare.

Таким образом, базовая структура тома NetWare включает:

- Сегмент физического носителя, который подготовлен как раздел NetWare;
- Секторы физического носителя, поддерживаемые контроллером диска;
- Блоки, каждый из которыхпредставляет собой массив секторов;
- Том, представляет собой массив блоков.

Том NetWare может быть многосегментным. Поэтому физический носитель тома может состоять из нескольких дисководов.

Многосегментные тома имеют следующую структуру:

- Том может включать до 32 сегментов;
- Отдельный физический носитель может состоять максимум из 8 сегментов, относящихся к одному или нескольким томам.

Размещение сегментов одного тома на разных дисках позволяет осуществлять операции чтения и записи различных частей этого тома одновременно, что повышает скорость доступа к данным. Однако при размещении сегментов тома на нескольких дисках требуется зеркальное отображение дисков для защиты информации при отказе какого-либо диска, иначе такой отказ приведет к потере одного или нескольких томов.

Таблица, которая описывает сегмент, называется таблицей определения тома Volume Definition Table (VDT). В этой таблице содержится имя тома, размер тома и информация о расположении сегментов тома на различных дисках. Каждый том NetWare содержит четыре копии (для обеспечения отказоустойчивости) таблицы VDT в каждом разделе NetWare диска. Кроме таблиц VDT раздел NetWare содержит область переназначения дефектных блоков Hot Fix, остальная часть раздела NetWare отводится под сегменты, которые могут принадлежать различным томам.

На сервере NetWare должен быть один диск, содержащий раздел DOS. Этот раздел является активным и с него после выполнения стартового командного файла DOS autoexec.bat автоматически стартует ОС NetWare.

Логическая структура тома

Каждый том имеет таблицу распределения блоков файлов FAT и таблицу входов в каталог DET (Directory Entry Table). Таблица FAT по назначению аналогична таблице FAT MS-DOS, а таблица DET - корневому каталогу диска MS-DOS. Отличие DET от корневого каталога DOS состоит в том, что для каждого файла в нем может находиться несколько записей - входов, если файл имеет не DOS'овский формат.

Таблицы FAT и DET кэшируются в оперативной памяти сервера. FAT кэшируется всегда, а DET - динамически, кэшируются только те входы, которые используются. Входы DET могут выгружаться из памяти, если они долгое время не используются.

NetWare всегда оперирует с избыточным числом копий FAT и DET для надежности.

Кэширование файлов

В NetWare для достижения высокой производительности файловой системы реализован обширный динамический кэш файлов в оперативной памяти. Этот кэш построен на блочной основе. Когда приложение читает или пишет в файл, NetWare копирует нужные блоки данных файла в кэш (если они не находятся уже там). Когда файловая кэш-память полностью заполняется, NetWare выполняет процедуру выгрузки в соответствии с алгоритмом "наименее используемый в последнее время" (Least Recently Used, LRU).

NetWare конфигурирует файловую кэш-память во время инсталляции ОС. После распределения памяти для структур данных операционной системы и инициализации динамических таблиц для стартовой конфигурации, NetWare превращает всю оставшуюся память в файловый кэш. Если NLM'ы динамически запрашивают память, то она берется из памяти файлового кэша. В версиях NetWare 4.x NLM может вернуть эту память файловому кэшу, когда она ему больше не нужна (в предыдущих версиях такой возможности нет).

NetWare кэширует данные файлов поблочно. Это позволяет NetWare достигать высокой степени синхронизации между буферами файлового кэша и блоками тома. Фактически, система кэширования файлов представляет собой часть логической файловой системы NetWare. Такая тесная интеграция между файловым кэшем и физическими носителями помогают сохранить целостность данных в файлах при значительном выигрыше в производительности.

В NetWare в буферах кэш-системы хранятся не только блоки данных файлов, но и такие элементы файловой системы, как FAT, Turbo FAT, кэш-таблица и входы каталогов. Turbo FAT представляет собой таблицу, в которой непосредственно перечислены все блоки файла, если их количество превышает 64. Это обеспечивает быстрый доступ к большим файлам.

При разработке серверных приложений при использовании стандартных функций API работы с файлами программисту нет необходимости задумываться об особенностях реализации системы кэширования файлов. Однако NetWare предоставляет разработчику специальные функции чтения данных непосредственно из буферов кэша (API асинхронного чтения AsyncRead API). Этот API позволяет увеличить производительность NLM-приложений.

Основные направления развития NetWare

Поддержка мультипроцессирования

В версию NetWare 4.1 средства поддержки многопроцессорных платформ не попали, хотя Novell объявила о своей трехэтапной стратегии внедрения средств мультипроцессирования еще на конференции BrainShare'94. Эту стратегию Novell назвала *распределенной параллельной обработкой* (Distributed Parallel Processing, DPP).

На первой стадии Novell будет поддерживать симметричные аппаратные платформы не совсем симметричным способом. Схематически этот вариант представлен на рисунке 7.4. Ядро системы и существующие модули NLM будут выполняться постоянно на одном из процессоров системы, в то время как модули, занимающиеся обработкой ввода-вывода (например, драйверы сетевых адаптеров, диска), и специально разработанные NLM будут динамически распределяться между остальными процессорами.

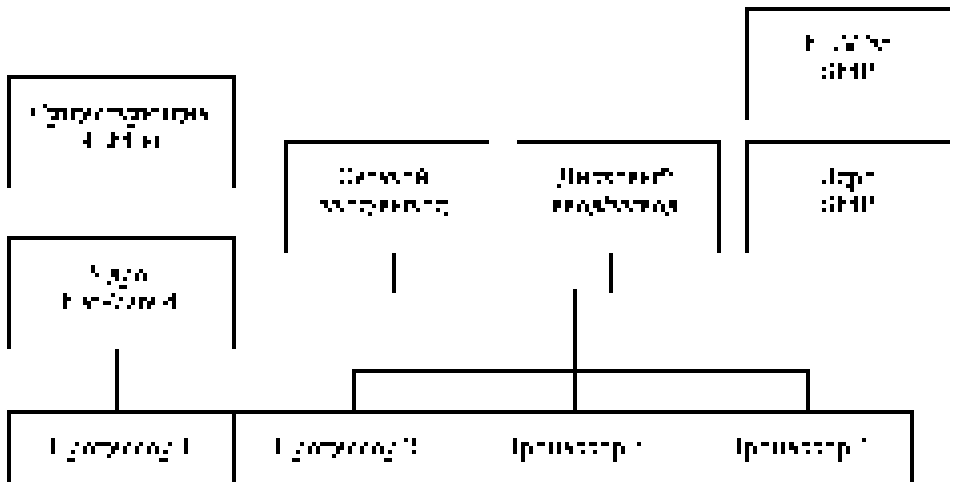


Рис. 7.4. Первая стадия стратегии Novell по внедрению распределенной параллельной обработки (Byte, 02, 1995)

В настоящее время первая стадия поддержки мультипроцессорных платформ реализована Novell в версии NetWare 4.1 SMP, которая поставляется только производителями некоторых симметричных мультиплексоров, например, компанией Tricord.

Обеспечение процессорной независимости

Помимо поддержки многопроцессорного режима, в число приоритетных направлений развития NetWare входит обеспечение процессорной независимости.

Делаются попытки переноса NetWare на RISC-платформы. Для этого Novell переписала NetWare на C и отделила ее аппаратно-зависимые части. Так как ранее Novell уже использовала название Portable NetWare для обозначения версий NetWare, работающих в среде VMS и UNIX, то эта действительно переносимая версия NetWare была названа PIN (Processor Independent NetWare). Она будет работать как "родная" на процессорах PowerPC и поддерживать NLM'ы.

Усилия по программе PIN не только отрывают NetWare от команд x86, но и уводят ее от шин PC, архитектуры памяти и системы прерывания. Такое отделение осуществляется с помощью слоя NSI (NetWare Systems Interface), эквивалента Novell слоя HAL в ОС Windows NT. NSI ведет свое происхождение из работы, проведенной фирмой NetFrame Systems, которая с 1989 года занимается адаптацией NetWare для работы на своих суперсерверах, которые хотя и построены на процессорах Intel, но имеют архитектуру более близкую к мейнфреймам, чем к персональным компьютерам.

"Мы купили лицензию на код NetWare и удалили оттуда все ссылки на контроллер прерывания, функции BIOS и все остальное, что было непосредственно связано с процессором Intel" - рассказывает Карл Амдал (Carl Amdahl). Позже эта работа была использована в NetWare 3.11, в которой зависимости от платформы изолированы в модуле, загружающем ядро NetWare. А теперь эти же результаты используются при разработке NSI.

Однако главная проблема состоит в том, нужен ли вообще многоплатформенный вариант NetWare. Поскольку узким местом сервера NetWare, нацеленного в основном на операции с файлами, являются возможности подсистемы ввода-вывода, а не вычислительные операции, то есть сомнения в целесообразности переноса NetWare на платформы с более мощным процессором. Действительно, в существующих NetWare-серверах процессоры семейства Intel, как правило, являются недозагруженными. Этот вопрос очень болезненен для Novell, особенно после того, как ее основной партнер по программе PIN - Hewlett-Packard приостановил работы по переносу NetWare на PA-RISC, а перенос на процессор Alpha отложен на неопределенный срок.

Операционные системы рабочих станций фирмы Novell

После продажи компанией Novell операционной системы UnixWare компании Santa Cruz Operations в ассортименте операционных систем, предлагаемых Novell для установки на рабочих станциях, остались система DOS 7 и система Personal Ware, построенная на основе DOS 7. Кроме этого, Novell предлагает большое количество сетевых оболочек, состоящих из драйверов коммуникационных протоколов, редиректора и пользовательской утилиты, для большинства популярных настольных ОС: OS/2, Windows 3.1, Macintosh (оболочки для Windows NT и Windows 95 должны появиться в ближайшее время).

Novell DOS 7 объединяет в себе прогрессивные технологии в области DOS-совмес-тимых операционных систем, средства полной поддержки сетей NetWare и построения одноранговой локальной сети для рабочей группы. Будучи полностью совместимой с используемыми приложениями и драйверами для DOS и Microsoft Windows, DOS 7 расширяет стандартные возможности DOS многозадачностью и поддержкой защищенного режима.

Novell DOS 7 - это первая версия DOS со встроенными функциями для организации одноранговых сетей, и, по мнению фирмы Novell - это главная отличительная особенность этой ОС.

Встроенные сетевые возможности

Novell DOS 7 объединяет в себе средства построения одноранговой локальной сети **Personal Ware**(доступной и в виде отдельного продукта), универсального клиента NetWare (Universal NetWare Client) и стандартные средства сетевого управления.

Сервер одноранговой сети Personal Ware (называемый Personal Ware Server или Desktop Server) обеспечивает легкое в использовании разделение файлов, принтеров, накопителей на CD-ROM для приложений DOS и MS Windows и является эффективным решением для рабочей группы в сети NetWare или для начинающих построение сети пользователей. Сервер Personal Ware не является выделенным.

Универсальный клиент NetWare - оболочка для перенаправления сетевых запросов - обеспечивает доступ к ресурсам файл-серверов под управлением NetWare 2.x, 3.x, 4.x и к серверам Personal Ware. Универсальный клиент реализован как набор VLM-модулей. Унифицированный полный набор сетевых утилит реализован для DOS и MS Windows. Доступ ко всем ресурсам сети осуществляется по единому имени и паролю.

Особенности сетевых возможностей Personal Ware:

- сервер Personal Ware может выполняться в защищенном режиме процессора с использованием DPMS;
- возможна загрузка DOS с сервера Personal Ware для бездисковых станций на основе стандартной технологии BOOT ROM / DosGen;

- Novell DOS 7 включает агента SNMP, MIB (Management Information Base), утилиты для наблюдения и управления разделяемыми ресурсами;
- автоматическое переподключение к серверу после его перезагрузки;
- база данных о пользователях сети дублируется на всех серверах.

Прогрессивные DOS-технологии

Novell DOS 7 - первая DOS с драйверами защищенного режима и многозадачностью.

Сервис защищенного режима DPMS (DOS Protected Mode Services) позволяет написанным в соответствии со спецификацией DPMS резидентным программам и драйверам устройств выполняться в защищенном режиме процессора и использовать расширенную (extended) память, сохраняя обычную память (conventional, upper и high) для прикладных программ. Дисковый кэш, дисковый компрессор Stacker, сервер Personal Ware и другие компоненты DOS 7 используют DPMS. DPMS совместим с MS Windows и программами управления памятью третьих фирм.

Novell DOS 7 реализует вытесняющую многозадачность (preemtive multitasking), позволяя нескольким программам DOS выполняться одновременно. Поддерживаются приложения - клиенты DPMI, такие как Lotus 1-2-3 v3.1, Paradox, сессия MS Windows в стандартном режиме. В фоновом режиме могут выполняться и графические программы. Предлагается полный набор программных интерфейсов поддержки многозадачности для разработчиков (разделение памяти, каналы, очереди, семафоры и др.)

Novell DOS 7 включает Stacker, наиболее эффективную и надежную программу сжатия информации на дисках, включая гибкие. Stacker включает утилиты конвертации дисков, обслуживаемых программами MS-DOS Double Space и DR DOS SuperStor. Stacker использует спецификацию DPMS.

В Novell DOS 7 имеется полный набор эффективных утилит, поддерживающих DPMS:

- дисковый кэш;
- дисковый оптимизатор;
- отслеживание удалений файлов и восстановление удаленных файлов;
- версии утилит фирмы Fifth Generation для DOS и MS Windows: архивирования файлов Fastback Express и антивирус Search&Destroy.

Novell DOS 7 предлагает различные средства обеспечения безопасности:

- загрузка по паролю;
- защита жесткого диска;
- контроль доступа к ресурсам за определенный период времени;
- вход в компьютер и в сеть по единому паролю.

Другие особенности Novell DOS 7:

- включает совместимый с MSCDEX доступ к накопителям на CD-ROM, драйвер поддерживает DPMS;
- расширенные командные файлы и CONFIG.SYS: условное выполнение, меню, приглашения;
- полный электронный справочник по всем возможностям;
- утилиты DOS с полноэкранным пользовательским интерфейсом.

Novell DOS 7 - новая версия популярной системы DR DOS 6.0, а продукт Personal NetWare заменил на рынке одноранговых сетей NetWare Lite 1.1.

Сетевые системные утилиты

NetWare Connect 1.0 фирмы Novell

NetWare Connect - программный продукт для создания в сетях NetWare мощных коммуникационных центров, решающих большинство задач асинхронных коммуникаций как для пользователей сети, так и для сторонних пользователей, которым необходим доступ в сеть NetWare.

NetWare Connect обеспечивает удаленным от сети пользователям DOS, Windows и Macintosh прозрачный доступ в сеть в качестве рабочих станций NetWare, как будто бы они непосредственно включены в сеть (режим remote node). Кроме того пользователи сети NetWare могут коллективно использовать такие сетевые ресурсы, как модемы, асинхронные порты и каналы связи, оборудование и каналы X.25. NetWare Connect полностью заменяет продукт NetWare Asynchronous Communication Server (NACS) v.3.0.

Реализованный в виде NLM-модуля ОС NetWare 3.x и 4.x, NetWare Connect использует такие ее преимущества, как высокая производительность, безопасность и удобства администрирования. Возможны выгрузка либо динамическое переконфигурирование NetWare Connect без перезагрузки файл-сервера NetWare.

NetWare Connect может предоставить пользователям сети три полезные функции:

- коллективный доступ к коммуникационной аппаратуре (сервер модемов и другого коммуникационного оборудования);
- сервис удаленного узла сети;
- поддержку сервиса удаленного управления.

NetWare Connect предоставляет эффективное решение проблемы *коллективного доступа к коммуникационной аппаратуре*, например, совместного использования модемов. Без использования NetWare Connect каждый пользователь, которому необходимо работать с модемом, должен был установить его на свой компьютер. Модем и телефонный канал выделялись каждому

пользователю и большую часть времени простаивали. NetWare Connect превращает коммуникационное оборудование в разделяемый сетевой ресурс, захватываемый пользователем только на время работы. Администратор сети оценивает потребности пользователей в одновременно работающих модемах и устанавливает их в сети NetWare Connect на соответствующее число портов.

С любой станции сети становится возможным, используя совместимые с NetWare Connect коммуникационные программы, осуществлять доступ к центральным ЭВМ, электронным доскам объявлений (BBS), сетям X.25 и серверам доступа (например, к NetWare Access Services или Citrix WinView for Networks). Совместимость коммуникационных пакетов третьих фирм с NetWare Connect достигается использованием промышленных стандартов NetWare Asynchronous Services Interface (NASI) или BIOS Int 14. Обеспечивается доступ в локальную сеть удаленных ПК, использующих программы удаленного доступа третьих фирм (например, pcAnywhere).

Важной принципиально новой способностью NetWare Connect является поддержка *сервиса удаленного узла* сети. Удаленные пользователи DOS, Windows и Macintosh могут звонить на NetWare Connect с удаленной рабочей станции и работать в сети так же, как они делают это при локальном подключении. Сетевой трафик протоколов IPX, TCP/IP и AppleTalk направляется по асинхронному каналу связи. Для пользователей DOS и Windows в составе NetWare Connect поставляется программа NetWare Remote Node (NRN), которая взаимодействует с компонентом Remote Node Services (RNS) на сервере NetWare Connect (рисунок 7.5). Перед предоставлением такой связи RNS проверяет имя и пароль удаленного пользователя. После этого поверх NRN на удаленном компьютере могут загружаться программы IPXODI и сетевая оболочка NETX или оболочка выполненная по VLM-технологии, обеспечивая тем самым тот же сервис для удаленного узла, что и для локального. Программа NRN с точки зрения программы IPXODI работает аналогично ODI-драйверу.

Для пользователей Macintosh, которым необходим доступ к сети AppleTalk, NetWare Connect поддерживает программное обеспечение AppleTalk Remote Access фирмы Apple Computer. В сервере NetWare Connect функции главной машины для удаленного пользователя компьютера Macintosh поддерживает компонента Apple Remote Access Service (ARAS).

Пользователи могут обмениваться почтой, работать с файлами и любыми программами, хотя низкая пропускная способность асинхронного канала может приводить к слишком большому времени реакции системы, и накладывает определенные ограни-чения на спектр эффективно используемых прикладных программ. При работе в диапазоне 1200 - 9600 б/с наиболее эффективно использование программ копирования файлов и приложений клиент-сервер.

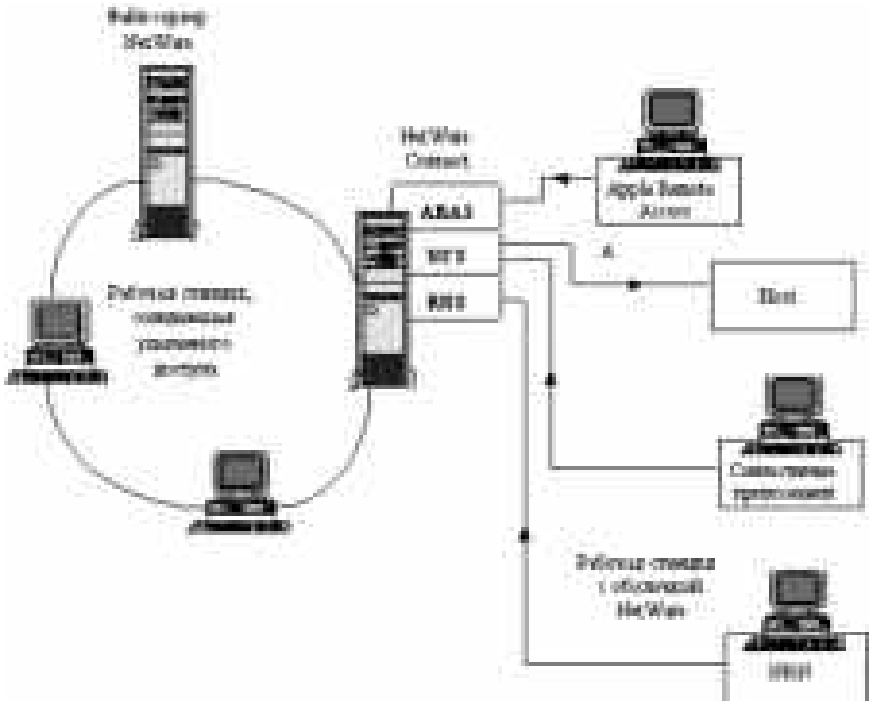


Рис. 7.5. Использование NetWare Connect

NetWare Connect также управляет входящими соединениями для приложений *удаленного управления*. Удаленные пользователи с помощью какого-либо пакета удаленного управления (например, pcAnywhere фирмы Symantec) могут позвонить на подключенный к NetWare Connect модем и получить удаленное управление рабочей станцией в сети. Например, уезжающий в деловую поездку служащий, может оставить свой офисный компьютер в ожидании соединения, и затем работать на нем удаленно. Компонента NetWare Connect, называемая NASI Connection Service (NCS), управляет как входящими, так и исходящими соединениями.

Серверы приложений (например, NetWare Access Services или WinView for Networks фирмы Citrix) могут использовать NetWare Connect в качестве коммуникационного ресурса для приема входящих звонков. Кроме того, с серверами приложений удаленный пользователь может работать после входа в сеть через NetWare Connect в режиме удаленного узла.

Исходящие соединения могут осуществляться с любой станции сети с использованием NASI-совместимых коммуникационных программ других фирм, которые, как правило, эмулируют терминал, осуществляют доступ к центральным ЭВМ, электронным доскам объявлений (BBS), сетям X.25 и серверам доступа.

Такие компоненты сервера NetWare Connect, как RNS, ARAS и NCS, разделяют его коммуникационные порты между собой. В отличие от NACS, сервер NetWare Connect не требует ручной работы при установлении входящего NASI-соединения или при его разрыве. Вместо этого доступные для удаленного управления компьютеры локальной сети регистрируются на сервере NetWare

Connect, но реальное NASI-соединение не выполняется (на назначенном порте) до тех пор, пока не поступит запрос от удаленного пользователя на доступ к какому-либо из этих компьютеров. Это позволяет использовать коммуникационный порт для выполнения других функций до образования NCS-соединения.

WinView for Networks v2.2 фирмы Citrix Systems

Фирмы Novell и Citrix Systems подписали соглашение о совместной разработке нового поколения средств удаленного доступа для сетей NetWare. Разработки будут основываться на созданной Citrix технологии сервера приложений WinView for Networks.

Данный программный продукт предназначен для реализации высокоэффективного сервера приложений, поддерживающего подключение пользователей по обычным телефонным линиям или по связям локальной сети. WinView, минимально загружая линии связи и обеспечивая необходимую надежность, позволяет работать с Windows-приложениями и текстовыми приложениями DOS и OS/2. Это достигается за счет использования разработанного Citrix коммуникационного протокола Intelligent Console Architecture. В рамках этого протокола сообщения, пересылаемые между сервером приложений и клиентом, состоят из сообщений Windows и уведомлений о событиях типа нажатия клавиш или кнопки мыши. Также возможна работа удаленных пользователей, использующих какой-либо из распространенных терминалов.

По сравнению с NetWare Access Services (NAS) WinView for Networks обладает целым рядом особенностей. Одна из них - возможность подключения удаленных пользователей не только через многопортовые адаптеры и интерфейс NASI, но и через стандартные COM-порты. В отличие от NAS, WinView может функционировать не только в интеграции с Novell NetWare, но и автономно. Очень интересна возможность эффективного общения с MS Windows по медленным каналам и полноценная работа с MS Windows при использовании в качестве клиентов машин класса AT 286 с 640 Кб оперативной памяти. За счет реализации на базе OS/2 WinView обеспечивает очень высокую надежность и эффективность исполнения разнообразных приложений. По этой же причине WinView предъявляет более высокие требования к персональному компьютеру, используемому в качестве сервера приложений. Рекомендуемая конфигурация - 486 компьютер с шиной EISA или MCA, 16 мегабайт оперативной памяти и 80 мегабайтный жесткий диск, многопортовый адаптер семейства DigiChannel, хотя возможно функционирование и на более скромных платформах. WinView for Networks версии 2.2 поставляется в версии на 5 и 10 одновременных подключений.

В соответствии с соглашением фирме Citrix передана линия продуктов NetWare Access Services (NAS). Начиная с 1994 года распространение и сопровождение

линии NetWare Access Services обеспечивает именно фирма Citrix. Существует Upgrade с NAS 1.3 на WinView.

Шлюзы IP-сетей

За последние годы локальные сети масштаба предприятия увеличились в размерах и в настоящее время поддерживают жизненно необходимые для этих предприятий программы. Находясь на различных стадиях процесса интеграции сетей персональных компьютеров, организации-пользователи будут по разным причинам сталкиваться с необходимостью соединить хост-системы UNIX со своими сетями персональных компьютеров. На первых стадиях интеграции эти организации обычно просто хотят, чтобы пользователи UNIX-систем могли использовать недорогое периферийное оборудование, подключенное к сетям NetWare - принтеры, стриммеры и т.п. Важны также возможности перемещения и совместного использование файлов. Те организации, которые уже зашли достаточно далеко в процессе интеграции сетей, обычно начинают использовать хост-системы UNIX как файл-серверы или как системы конечного хранения и обработки баз данных. Рабочие станции персональных компьютеров в сети NetWare используют приложения предварительной обработки данных, имеющие доступ к базам данных хостов. И, наконец, в настоящее время многие организации хотели бы интегрировать приложения, работающие в их сетях персональных компьютеров, с приложениями хост-системы.

Проблемы соединения этих двух типов сетевых сред заключаются в том, что они используют различные стеки протоколов на всех уровнях, начиная с сетевого. Сети Novell NetWare используют протоколы IPX/SPX на сетевом уровне и протокол NCP на уровне взаимодействия верхних уровней клиентской и серверной частей ОС. В UNIX-сетях, как правило, используются протоколы стека DoD с сетевыми протоколам TCP/IP и такими протоколами уровня клиент-сервер, как telnet (эмуляция терминала), FTP (пересылка файлов) и популярный протокол NFS фирмы Sun, позволяющий монтировать удаленную файловую систему в локальное дерево каталогов.

На рисунке 7.6 показаны продукты фирмы Novell для взаимодействия с UNIX-сетями и основные выполняемые ими функции.

LAN WorkPlace for DOS обеспечивает поддержку широко распространенного сетевого протокола TCP/IP и реализует на его основе доступ пользователя рабочей станции NetWare к таким сервисам UNIX-машины, как эмуляция терминала по протоколу telnet и обмен файлами по протоколу FTP. Это дает возможность получить доступ к большому количеству микро- и мини- ЭВМ с различной архитектурой и используемой OS (UNIX, VMS, OS/2), работающих с TCP/IP, к локальным и глобальным сетям (Internet), построенным на этом протоколе.

Реализация TCP/IP в LAN WorkPlace использует ODI (Open Data-link Interface) драйверы, применяемые в NetWare, что автоматически обеспечивает

совместимость с широким спектром сетевых адаптеров для различных типов сетей: Ethernet, Token Ring, Arcnet, FDDI, Packet Radio. ODI-технология позволяет при работе в сети NetWare одновременно использовать IPX/SPX и TCP/IP на одном и том же сетевом адаптере.

Реализация TCP/IP дополнена двумя наборами приложений (для DOS и для Windows), которые обеспечивают традиционный UNIX-сервис для передачи файлов (FTP) и эмуляции терминалов (telnet). В дополнение к этому LAN WorkPlace for DOS обеспечивает открытый набор API, на которых базируется большое количество программных систем третьих фирм. В их числе такие "нетривиальные" сетевые продукты, как: DeskView/X фирмы Quarterdeck, X-Vision фирмы VisionWare, SQL*Net for MS-DOS фирмы Oracle, PC Net-Library for DOS фирмы Sybase, система TUXEDO фирмы USL.

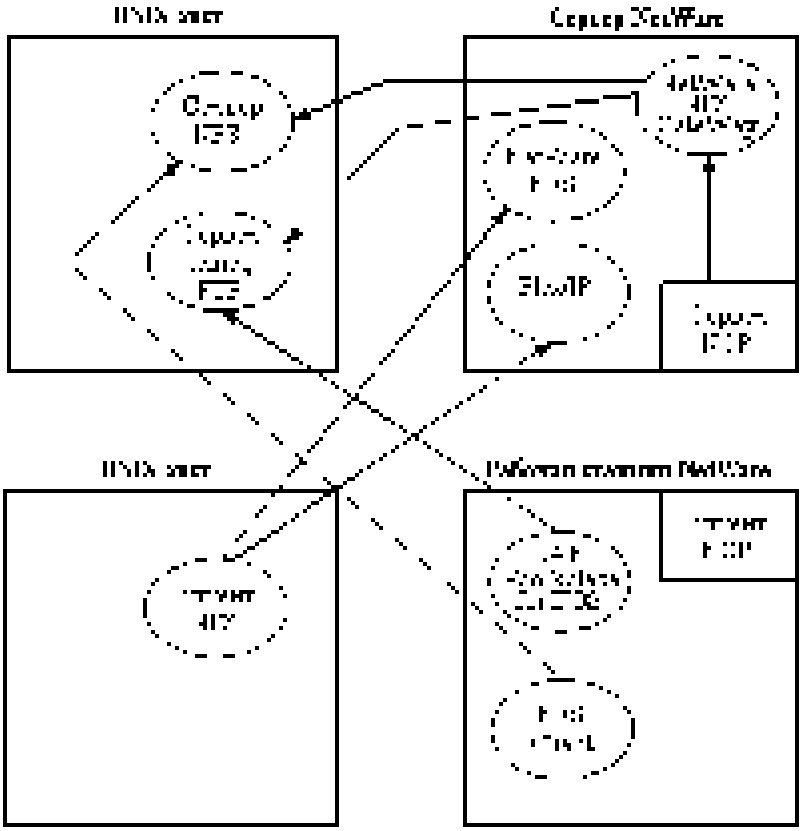


Рис. 7.6. Взаимодействие сетей NetWare и UNIX

LAN WorkPlace позволяет также DOS и MS Windows клиентам сети NetWare непосредственно связываться с удаленными серверами NetWare 3.11, которые доступны только через TCP/IP маршрутизаторы, или же работать в сети, где доступным является только TCP/IP протокол. Для этого используется механизм, упаковывающий IPX-датаграммы в TCP/IP пакеты - так называемый "IP Tunneling". Версия 4.1 дополнена поддержкой IP на последовательных каналах по протоколам SLIP, CSLIP и PPP, что существенно расширяет область применения пакета.

NFS Client for LAN WorkPlace/LAN WorkGroup v2.3 является дополнением к LAN WorkPlace/LAN WorkGroup и представляет собой реализацию клиентской части широко распространенного протокола NFS фирмы SUN. В силу того, что NFS Client базируется на стандартном TCP/IP протоколе, он позволяет DOS-рабочим станциям непосредственно монтировать файловые системы различных UNIX-систем, поддерживающих NFS.

Функциональные возможности:

- Прозрачный доступ к файлам хост-машин NFS путем отображения их файловых систем на логические диски DOS.
- Поддержка символических связей UNIX-файловой системы. Возможность запуска многопользовательских DOS-приложений, таких как DBase IV, Paradox, Clipper с NFS сервера, что позволяет существенно уменьшить требуемое пространство на локальном диске рабочих станций.
- Возможность поддерживать до 16 ID групп, используемых для доступа к файлам на NFS-сервере, что позволяет работать одновременно с большим количеством файлов, чем при работе с другими PC NFS продуктами.
- Механизм блокировки и разделения файлов, доступный в DOS 3.1 и выше, может быть использован для файлов на NFS-сервере, вне зависимости от наличия подобного механизма в используемой реализации NFS.
- Поддержка простой схемы маппирования для ID-пользователей, ID-групп и NFS hostnames, которая позволяет не использовать более сложную систему Network Information Service.
- Совместное использование принтеров, подключенных к хостам NFS.
- Поддержка сетевых возможностей MS Windows. В поставку входят конфигурационные файлы, требуемые для программы SETUP в MS Windows.
- Эффективное использование доступной верхней памяти. Для загрузки драйверов и клиентской программы LWPNFS в верхнюю память могут быть использованы известные программы QEMM-386, 386 Max и другие. Предусмотрена также возможность удаления LWPNFS из памяти командой unload.

Пакет Flex/IP предназначен для использования в сетях, где требуется доступ пользователей UNIX к ресурсам (принтеры, файлы) NetWare. В качестве коммуникационного транспортного протокола Flex/IP использует широко распространенный в UNIX протокол TCP/IP. На основе этого протокола в UNIX-системах имеются утилиты для передачи файлов в сети, следующие стандарту FTP (File Transfer Protocol). Для интеграции с имеющимися UNIX-системами фирма Novell выбрала путь реализации этих сетевых протоколов на платформе NetWare. Так, поддержка TCP/IP на уровне маршрутизации IP-пакетов представлена в NetWare версии 3.11 как одна из базовых функций. В дополнение к этому пакет Flex/IP предоставляет NetWare-серверу дополнительные функции сервера FTP, доступного любым FTP-клиентам, использующим протокол TCP/IP.

Возможности удаленного доступа к файлам NetWare и совместного использования сетевых принтеров, осуществляемые пакетом Flex/IP, позволяют считать этот вариант интеграции NetWare и UNIX весьма эффективным с точки зрения его простоты, надежности и невысокой стоимости.

Функциональные возможности Flex/IP:

- *File Transfer.* Flex/IP реализует FTP-сервер, поддерживающий параллельные FTP-соединения. Любой пользователь UNIX-рабочей станции, имеющий необходимые права доступа, может устанавливать соединения с одним или несколькими NetWare-серверами, просматривать удаленные директории и пересылать файлы в обоих направлениях. Более того, Flex/IP обеспечивает функцию FTP-gateway, давая пользователям UNIX возможность FTP-доступа к другим серверам NetWare (даже не поддерживающим TCP/IP, как, например, в случае с NetWare v2.15).
- *Sharing printers.* В рамках FLex/IP поставляется так называемый "bidirectional print gateway", позволяющий как UNIX-TCP/IP-клиентам использовать принтеры, подключенные к NetWare, так и рабочим станциям NetWare использовать принтеры, подключенные к UNIX-машинам.
- *Managing NetWare 3.11 Servers from X-Windows.* В состав FLex/IP входит X-Window приложение. XCONSOLE, которое позволяет на X-Windows-станции иметь удаленный доступ к консоли NetWare, давая возможность пользователю диагностировать и конфигурировать систему NetWare, не покидая своего рабочего места.

NetWare NFS v1.2

NetWare NFS обеспечивает прозрачную интеграцию UNIX-систем с файловой системой и ресурсами NetWare 3.11, предоставляя пользователям UNIX доступ к среде NetWare из их "родного" программного окружения с использованием привычной для них процедуры монтирования удаленной файловой системы сервера NetWare. UNIX-клиенты могут использовать NetWare NFS для доступа к файлам NetWare и другим сетевым ресурсам совместно с другими NetWare-клиентами, такими как DOS, OS/2 и Macintosh-компьютеры.

NetWare NFS использует коммуникационный транспортный протокол TCP/IP, широко применяемый в UNIX-системах и стандартный для NFS. Благодаря этому не возникает необходимости в дополнительном программном обеспечении на стороне UNIX-клиентов.

Возможности NetWare NFS:

- *NFS-сервер.* Позволяет пользователям UNIX рассматривать файловую систему NetWare как расширение их естественной распределенной файловой системы NFS. Для доступа к томам NetWare используются

стандартные команды монтирования UNIX/NFS. NFS-Server поддерживает: transparent support of authentication, отображение атрибутов и имен файлов, блокировку файлов, преобразование прав доступа UNIX и NetWare, удаленное сетевое администрирование.

- *NFS Namespace NLM.* Привносит в файловую систему NetWare file attributes and naming conventions, характерные для UNIX-систем, что обеспечивает прозрачный обмен файлами с DOS, Macintosh и OS/2 станциями.
- *LPD NLM* - сервер печати, позволяющий UNIX-рабочим станциям перенаправлять принтерные запросы в стандартные очереди печати NetWare.
- *Line Printer Gateway* - дает возможность пользователям сети NetWare печатать на принтерах, подключенных к UNIX-машинам.
- *FTPSERV NLM* - сервер FTP протокола, позволяющий любому FTP-клиенту, имеющему соответствующие права доступа, передавать файлы с и на любой сервер NetWare. Lock Manager NLM поддерживает операции блокировки файлов и записей в среде NFS. UNIX-приложения, использующие блокировки, полностью поддерживаются программой Lock Manager.
- *XCONSOLE* - программа для удаленного управления сервером NetWare, дающая возможность администраторам UNIX'а открывать X-Windows сессии с одновременным доступом к консолям нескольких NetWare-серверов.
- *Management utilities.* Широкий набор программ, позволяющий следить за состоянием NFS-сервера, и, при необходимости, реконфигурировать его.

NetWare NFS Gateway предоставляет пользователям рабочих станций сети NetWare доступ к файловой системе удаленных NFS-серверов и поддерживает протоколы NFS, FTP и telnet. Gateway, реализованный как NLM, устанавливается на сервере NetWare и позволяет последнему связываться по протоколу TCP/IP с любым UNIX NFS сервером и монтировать его файловую систему, создавая при этом в дополнение к локальным томам на диске NetWare-сервера том сетевой файловой системы. Эти функции на NetWare-сервере осуществляются в многозадачном режиме, параллельно с выполнением основных функций сервера. С точки зрения рабочих станций NetWare доступ ко всем (NetWare и NFS) томам на сервере абсолютно одинаков. Соединение станций по прежнему осуществляется по протоколу IPX/SPX, и поэтому дополнительного программного обеспечения на рабочих станциях не требуется.

NetWare NFS Gateway, таким образом, производит преобразование протоколов между рабочей станцией и сервером NetWare (NCP и IPX/SPX) с одной стороны и между NetWare и NFS-серверами (NFS и TCP/IP) с другой. В этом заключается существенное отличие данного решения проблемы доступа к удаленной файловой системе от решения, предоставляемого NFS Client for LAN WorkPlace.

Системы обработки сообщений MHS и GroupWise

Для пользователей сетей NetWare сейчас доступны две мощные системы обработки сообщений: Message Handling System (MHS) и GroupWise. Некоторое пересечение функций этих систем объясняется тем, что первая система была разработана в недрах фирмы Novell уже достаточно давно и дошла в своем развитии до версии 2.0, а система GroupWise перешла к Novell как наследство компании Word Prefect, приобретенной Novell. У этих систем имеются достаточно четкие отличия: система MHS представляет собой основу для построения мощных систем электронной почты, а система GroupWise - это уже законченная электронная почта плюс построенная на ней надстройка для групповой работы над документами.

Средства MHS фирмы Novell уже доступны в течение нескольких лет; на сегодняшний день более 1000 зарегистрированных корпоративных и независимых разработчиков программного обеспечения разработали приложения, использующие MHS. Последняя версия этой системы управления сообщениями по схеме "запоминание-и-передача" называется Global MHS. Она представляет собой работающее на сервере приложение, выполненное в форме NLM-модулей.

Доступ к MHS обеспечивается способом, очень отличающимся от распространенных программных интерфейсов электронной почты типа MAPI или VIM. MHS не имеет программируемого интерфейса. Файл-сообщение отсылается путем добавления текстового (ASCII) заголовка, который содержит инструкции о адресе и способе доставки. Этот файл с присоединенными сообщениями в двоичном или любом другом формате сохраняется в каталоге, который периодически опрашивается сервером MHS. В этом процессе не участвуют ни вызовы процедур, ни библиотечные функции. Хотя и с некоторой натяжкой, этот подход очень похож на способ адресации сообщений в протоколе SMTP, используемом в сетях Internet.

MHS-серверы имеют такие развитые свойства, как автоматическая маршрутизация по критерию стоимости или функции синхронизации справочников, которые обновляют справочники MHS во всей корпорации, независимо от того, где были сделаны изменения. В новой версии NetWare 4.1 справочная служба MHS интегрирована со службой каталогов NetWare NDS, что значительно снижает административные издержки - в сети существует единая глобальная справочная служба.

Для создания программируемого интерфейса к MHS третьи фирмы разработали динамические библиотеки среды Windows, функции которых автоматизируют адресацию сообщений в формате MHS, используя наглядные меню выбора адреса в стиле Windows. Фирма Novell сама недавно выпустила набор библиотек DLL, которые обеспечивают доступ к функциям MHS для приложений, использующих VIM, MAPI или CMC.

Для приложений управления документами MHS поддерживает встроенную трансляцию форматов документов. Для удаленных пользователей Novell поставляет продукт Remote MHS, который делает интерфейсы MHS, VIM, Simple MAPI и CMC доступными для любого удаленного Windows-приложения.

Как и в случае продуктов, основанных на моделях синхронного взаимодействия клиент-сервер и асинхронной передачи сообщений, корпоративные свойства средств модели "запоминание-и-передача" следует воспринимать с некоторой долей скептицизма. Несмотря на многочисленные обещания, продукты этой модели еще не обеспечивают поддержку гетерогенности, необходимую для полномасштабной циркуляции документов, изображений и форм по корпоративной сети. Легкая синхронизация справочных служб различных систем модели "запоминание-и-передача" все еще только ожидается, равно как и следование стандарту X.500. Пока магистрали таких служб непригодны для транзакционных приложений, если только сервисы "запоминание-и-передача" не интегрированы в транзакционные журнальные системы (очень мало поставщиков говорит сегодня об этом).

Несмотря на необходимость улучшений, разработчики корпоративных сетей уже начали использовать системы передачи сообщений модели "запоминание-и-передача" для критических приложений, включая транспортировку запросов к базам данных и обновление баз данных. Действительно, организациям, которые интенсивно используют Windows, cc:Mail или NetWare, было бы странно игнорировать ресурсы модели "запоминание-и-передача" при рассмотрении вариантов корпоративной интеграции. Сегодня магистрали сервисов "запоминание-и-передача" наилучшим образом подходят широкому кругу приложений обработки файлов и документов, и у них есть все шансы стать чем то более значительным, чем они есть, в недалеком будущем.

GroupWise - это в основном пакет электронной почты, но в то же время он предоставляет из себя и нечто большее. В него включены средства индивидуального, группового и ресурсного планирования. Пользователи могут составлять собственные списки неотложных дел и "заводить" напоминающие о них будильники. С помощью прекрасно работающего механизма правил можно, например, автоматически удалить все сообщения заданного пользователя или послать ответ по обратному адресу.

Пакет теперь поддерживает 12 клиентских платформ и использует для запуска серверных процессов серверы NetWare и несколько различных UNIX-серверов. Ранее сетевые администраторы вынуждены были некоторым задачам WordPerfect Office отводить выделенные DOS и OS/2 серверы.

Недостатком версии GroupWise 4.1 является то обстоятельство, что шлюзы для связи с другими почтовыми системами и доменами GroupWise по-прежнему работают только на DOS. Поэтому асинхронные коммуникации необходимо распределять по DOS-машинам. Таким образом, на пути к тому, чтобы продукту было достаточно для работы одного сервера, сделан заметный шаг.

В состав GroupWise входит предназначенная для удаленных пользователей клиентская часть, имеющая тот же интерфейс, что и версия для локальных рабочих станций.

Для большинства организаций существенно соответствие GroupWise стандарту X.500.

Серверная часть GroupWise состоит из нескольких NLM-модулей, отвечающих за пересылку сообщений между пользовательскими почтовыми ящиками, почтовыми отделениями и удаленными доменами. Вплоть до версии 4.1 эти функции выполнялись только на компьютерах, работающих под управлением DOS, либо в сеансе DOS под управлением OS/2. Реализация в виде NLM-модулей оказалась значительно удобнее и проще в эксплуатации.

Перед NLM-модулями, входящими в пакет GroupWise, на сервер NetWare следует загрузить несколько других NLM-модулей, и, прежде всего, разработанный Novell вспомогательный NLM-модуль протокола SNMP.

Поддержка данного протокола включена в GroupWise, однако модуль должен быть загружен даже в том случае, когда поддержка SNMP не нужна.

Одним из основных NLM-модулей является сервер сообщений MS. Этот модуль отвечает за пересылку корреспонденции между входящими в состав домена почтовыми отделениями. Модуль Office Server занимается доставкой корреспонденции от сервера сообщений к почтовым ящикам пользователей. Для каждого почтового отделения, входящего в состав домена, загружается своя копия модуля Office Server.

Административный сервер ADS отвечает за передачу запросов на синхронизацию каталогов между доменами GroupWise.

В состав пакета входит модуль BDS, осуществляющий синхронизацию каталогов GroupWise и NetWare. Он выявляет любые изменения в базе данных сетевых ресурсов среды NetWare 3.x и 4.x и в соответствие с ними обновляет различную служебную информацию GroupWise, в частности списки пользователей.

Количество шлюзов, предлагаемых в составе пакета GroupWise, впечатляет, однако это достижение частично обесценивается отсутствием поддержки различных платформ. Все 23 имеющихся в настоящее время шлюза являются DOS-ориентированными. Набор обслуживаемых систем весьма широк - от OV/VM компании IBM и стандарта X.400 до cc:mail компании Lotus и Massage Router компании DEC. Связь с удаленными доменами осуществляется с помощью шлюза Async Gateway. В дальнейшем предполагается использовать для этих целей пакет NetWare Connect.

Диапазон услуг, предлагаемых GroupWise, очень широк. Интегрированные в пакет средства планирования допускают составление графиков личных встреч и

деловых совещаний с учетом занятости таких специфических ресурсов, как конференц-залы или аудиовизуальное оборудование. В области электронных конференций Novell реализовала популярную в Internet технология обслуживания по спискам, которая дает возможность подписаться на электронный бюллетень и получать все посылаемые в него сообщения.

Семейство сетевых ОС компании Microsoft

Сетевые продукты Microsoft

В 1984 году Microsoft выпустила свой первый сетевой продукт, называемый Microsoft Networks, который обычно неформально называют MS-NET. Некоторые концепции, заложенные в MS-NET, такие как введение в структуру базовых компонент - редиректора и сетевого сервера - успешно перешли в LAN Manager, а затем и в Windows NT.

Microsoft все еще поставляет свою сетевую *ОС LAN Manager*. Большое количество независимых поставщиков имеют лицензии на эту ОС и поддерживают свои собственные версии LAN Manager как часть своих сетевых продуктов. В число этих компаний входят такие известные фирмы как AT&T и Hewlett-Packard. LAN Manager требует установки на файл-сервере операционной системы OS/2, рабочие станции могут работать под DOS, Windows или OS/2. OS/2 - это операционная система, реализующая истинную многозадачность, работающая в защищенном режиме микропроцессоров x86 и выше. LAN Manager использует 32-х битную версию файловой системы OS/2, называемую HPFS, которая оптимизирована для работы на файл-сервере за счет кэширования каталогов и данных. LAN Manager - это первая сетевая ОС, разработанная для поддержки среды клиент-сервер. Ключевыми компонентами LAN Manager являются редиректор и сервер. Особенно эффективно LAN Manager поддерживает архитектуру клиент-сервер для систем управления базами данных. LAN Manager разрешает рабочим станциям под OS/2 поддерживать сетевой сервис по технологии "равный-с-равным". Это означает, что рабочая станция может выполнять функции сервера баз данных, принт-сервера или коммуникационного сервера. Ограничением является то, что только один пользователь, кроме владельца этой рабочей станции, имеет доступ к такому одноранговому сервису.

Для работы в небольшой сети фирма Microsoft предлагает компактную, не требующую значительных аппаратных или программных затрат операционную систему *Windows for Workgroups*. Эта операционная система позволяет организовать сеть по схеме "равный-с-равным", при этом нет необходимости приобретать специальный компьютер для работы в качестве сетевого сервера. Эта операционная система особенно подходит для решения сетевых задач в коллективах, члены которого ранее широко использовали Windows 3.1. В Windows for Workgroups достигнута высокая производительность сетевой

обработки за счет того, что все сетевые драйверы являются 32-х разрядными виртуальными драйверами.

С середины 1993 года Microsoft начала выпуск новых операционных систем "новой технологии" (New Technology - NT) *Windows NT*.

В сентябре 1995 года компания Microsoft выпустила еще одну новую операционную систему Windows 95 (кодовое название Chicago), предназначенную для замены Windows 3.1 и Windows for Workgroups 3.11 в настольных компьютерах с процессорами Intel x86.

История Windows NT

В конце 88-го года Microsoft поручила Дэвиду Катлеру (David Cutler) возглавить новый проект в области программного обеспечения: создать новую ОС фирмы Microsoft для 90-х годов. (Дэвид Катлер - главный консультант фирмы DEC, который 17 лет проработал там, разрабатывая ОС и компиляторы: VAX/ VMS, ОС для MicroVAX I, OS RSX-11M, компиляторы VAX PL/1, VAX C). Он собрал команду инженеров для разработки ОС новой технологии (New Technology - NT).

Первоначально планировалось разработать NT с пользовательским и программным (API) интерфейсами в стиле OS/2, однако OS/2 плохо продавалась, а Windows 3.0 имела большой и постоянный успех на рынке. Увидев рыночные ориентиры и сложности, связанные с развитием и поддержкой двух несовместимых систем, Microsoft решила изменить свой курс и направить своих инженеров в сторону стратегии единой цельной операционной системы. Эта стратегия состоит в том, чтобы разрабатывать семейство базирующихся на Windows операционных систем, которые охватывали бы множество типов компьютеров, от самых маленьких ноутбуков до самых больших мультипроцессорных рабочих станций. Windows NT, как было названо следующее поколение Windows-систем, занимает самое высокое место в семействе Windows. Она поддерживает графический интерфейс (GUI) пользователя Windows, а также является первой базирующейся на Windows операционной системой фирмы Microsoft, поддерживающей Win32 API, 32-х битный программный интерфейс для разработки новых приложений. Win32 API делает доступными для приложений улучшенные свойства ОС, такие как многонитевые процессы, синхронизацию, безопасность, I/O, управление объектами.

В июле 1993 года появились первые ОС семейства NT - Windows NT 3.1 и Windows NT Advanced Server 3.1.

Версии Windows NT

Windows NT 3.1

Операционная система Windows NT с самого начала проектировалась с учетом всех требований, предъявляемых к современным ОС: расширяемости, переносимости, надежности, совместимости, производительности. Эти свойства были достигнуты за счет применения передовых технологий структурного проектирования, таких как клиент-сервер, микроядра, объекты.

В отличие от Windows, в которой реализована многозадачность без вытеснения (non-preemptive multitasking), в Windows NT используется механизм *многозадачности с вытеснением* (preemptive multitasking).

Windows NT поддерживает *симметричную многопроцессорную организацию* вычислительного процесса, в соответствии с которой ОС может выполняться на любом свободном процессоре или на всех процессорах одновременно, разделяя память между ними. Учитывая, что *многозадачность реализуется на уровне нитей*, разные части одного и того же процесса могут действительно выполняться параллельно. Следовательно, многонитевые серверы могут обслуживать более одного клиента.

Для управления нитями Windows NT Server использует *механизм приоритетов*. В определенные моменты производятся оценка приоритетов и перераспределение нитей по процессорам, в результате чего последовательные стадии одного потока программы могут выполняться разными процессорами или откладываться до высвобождения очередного процессора.

Windows NT Server поддерживает до 16 параллельных процессоров, что актуально для таких серверов, как Symmetry 750 фирмы Sequent с 16 процессорами Intel 486/50 МГц. Следует, однако, иметь в виду, что реализация СМП в Windows NT Server нацелена на оптимизацию производительности и не обеспечивает резервирования в целях повышения отказоустойчивости. В случае выхода из строя одного из процессоров система останавливается.

В Windows NT Server в полной мере реализован потенциал масштабируемости архитектуры СМП. Однопроцессорную систему можно легко развивать, наращивая число процессоров, без замены версии ОС или приложений.

При управлении устройствами ввода/вывода Windows NT Server использует асинхронный подход. Для завершения процесса и начала выполнения новой задачи не нужно ждать поступления сигнала об окончании таких операций, как чтение или запись. Каждый процесс создается с использованием одной нити, которая служит специфическим отображением выполнения программы процессором. Впоследствии программа может создавать новые нити, и Windows NT Server будет распределять их и управлять ими, не привлекая к этому приложения высокого уровня.

Для того, чтобы прикладная программа могла использовать несколько потоков, не нужно предусматривать этого в ее алгоритме. Отдельный поток создается для каждой операции. Например, в одном потоке программа может воспроизводить сложную графическую форму, а другой использовать для редактирования объемного чертежа. Каждый из этих потоков (или, с точки зрения пользователя, операций) работает на отдельном процессоре, не требуя никаких управляющих вмешательств со стороны приложения. Потоки внутри процесса используют общую область памяти и, следовательно, не должны специально обмениваться данными.

В соответствии с требованием совместимости, Windows NT обеспечивает среду выполнения не только для приложений с исходным программным интерфейсом Win32 API. При выполнении на процессорах фирмы Intel защищенные подсистемы Windows NT обеспечивают *двоичную совместимость существующих приложений* фирмы Microsoft, включая MS-DOS, Win16, OS/2. На MIPS RISC процессорах двоичная совместимость достигается для приложений MS-DOS и 16-битных Windows-приложений (с использованием эмуляции). Windows NT обеспечивает также совместимость на уровне исходных текстов для POSIX-приложений, которые твердо придерживаются интерфейса, определенного в стандарте IEEE 1003.1.

Помимо совместимости программных интерфейсов, Windows NT поддерживает существующие файловые системы, включая файловую систему MS-DOS (FAT), файловую систему CD-ROM, файловую систему OS/2 (HPFS) и собственную новую файловую систему (NTFS).

В отличие от большинства других операционных систем, Windows NT изначально разрабатывался с учетом возможности работы в сети. В результате этого функции совместного использования файлов, устройств и объектов встроены в интерфейс с пользователем. Администраторы могут централизованно управлять и контролировать работу сетей в масштабах крупных предприятий. Особенно важно отметить возможность распространения работы приложений типа клиент-сервер на многокомпьютерные системы.

Windows NT 3.5

Версия Windows NT 3.5, как и предыдущая Windows NT 3.1, разработана в двух конфигурациях: для рабочей станции Windows NT Workstation 3.5 и для сервера - Windows NT Server 3.5. Windows NT 3.5 имеет многочисленные усовершенствования и нововведения по сравнению с Windows NT 3.1:

Улучшенное автораспознавание аппаратуры, возможность ручного выбора и конфигурирования сетевых адаптеров, если автоматическое распознавание не дает положительного результата.

Встроенная совместимость с NetWare. Возможность выполнения роли шлюза к сетям NetWare, так что Windows NT-компьютеры могут получать доступ к

файлам, принтерам и серверам приложений NetWare. В Windows NT начиная с версии 3.5 входит Microsoft Compatible Workstation Service for NetWare, который позволяет осуществлять доступ к файлам, каталогам и принтерам на сервере Novell NetWare. Транспортный протокол Microsoft NWLink IPX/SPX обеспечивает связь между компьютером с Windows NT и NetWare файл-сервером и сервером печати. Он поддерживает работу с файлами и с очередями печати на NetWare сервере.

Встроенная поддержка TCP/IP. Новая высокопроизводительная Microsoft-реализация протоколов TCP/IP, которая обеспечивает простое, мощное решение для межсетевого взаимодействия.Microsoft поддерживает протокол TCP/IP, начиная с 1991 года, когда был выпущен первый стек для Microsoft LAN Manager 2.1. В Windows NT также имеется поддержка этого протокола, начиная с самой первой версии этой операционной системы. Помимо этого, имеются базовые утилиты, такие как ftp, tftp, telnet, команды r*, arp, route и finger. С выходом версии 3.5 появились новые ключевые свойства, которые, с одной стороны, упростили конфигурирование и обслуживание, а с другой - улучшили свойства TCP/IP.

Значительные улучшения *средств удаленного доступа* RAS, включающие поддержку IPX/SPX и TCP/IP, использование стандартов Point to Point Protocol (PPP) и Serial Line IP (SLIP). Сервер RAS может теперь поддерживать до 256 соединений (вместо 64 в версии 3.1).

Поддержка *длинных имен файлов* в файловой системе FAT. Windows NT поддерживает работу с тремя файловыми системами: NTFS, FAT и HPFS. Таким образом, если до установки Windows NT на компьютере были установлены MS-DOS или OS/2, то нет никакой необходимости переформатировать диск. Система преобразует FAT или HPFS в NTFS, сохранив всю информацию на диске. Обратное преобразование невозможно. Здесь уместно заметить, что если вы хотите установить NTFS только затем, чтобы использовать длинные (до 255 символов) имена файлов, то для этих целей прекрасно подойдут и FAT и HPFS. Если для последней это естественное свойство, то возможность использования длинных имен файлов на FAT была введена только в версии Windows NT начиная с 3.5. Вы можете спокойно называть файлы и каталоги именами, выходящими за пределы традиционного для MS-DOS правила "8.3", нисколько не опасаясь, что эти файлы не будут доступны при работе в MS-DOS. Для таких файлов и каталогов будут назначены вторые, "короткие" имена.

Полная поддержка хранения *встроенных объектов* OLE 2.x и поиска составных документов. Эти возможности включают связывание, встраивание, связывание со встроенными объектами, технологии "drag-and-drop" и OLE-Automation.

В операционную систему Windows NT 3.5 встроены графические возможности *трехмерной графики OpenGL API*. OpenGL - это независимая от операционной системы промышленно-стандартная библиотека графических функций, разработанная фирмой Silicon Graphics для своих рабочих станций. В

настоящее время OpenGL признана Architecture Review Board, включающей такие фирмы, как DEC, IBM, Intel, Microsoft и Silicon Graphics. Технология OpenGL была лицензирована Microsoft для предоставления этого мощного 32-разрядного API пользователям Windows NT. Развитые функции этой библиотеки требуются в том случае, когда необходима визуализация крупных проектов и данных. Типичные задачи, требующие ее использования, - это САПР, системы механического и промышленного дизайна, программы статистического и научного анализа.

Приложения, разработанные для MS Windows 3.x и MS-DOS, выполняются более надежно, так как каждое приложение теперь работает в *своем адресном пространстве*.

Доменная организация. В сетях на основе Windows NT Server рабочие станции подключаются к выделенным серверам. Именованные собрания серверов могут быть сгруппированы в *домены*. Такой метод организации сети упрощает централизованное управление сетью и позволяет использовать Windows NT Server в качестве сетевой операционной системы масштаба предприятия. Если администратор однажды завел учетную информацию о пользователе домена, то последний имеет возможность зарегистрироваться на любой рабочей станции в этом домене. Для этого достаточно ввести имя, имя домена и пароль при регистрации, и Windows NT Workstation опознает пользователя и воссоздаст его рабочую среду. В серверных сетях, как правило, все совместно используемые каталоги располагаются на выделенных серверах, а совместно используемые принтеры подключены к специализированным серверам печати. Однако это ни в коей мере не ограничивает возможностей пользователя по предоставлению ресурсов его рабочей станции в совместное использование так, как это обычно делается в одноранговых сетях. Windows NT Server предоставляет возможность пользователям различных доменов совместно использовать ресурсы путем установления *доверительных отношений* между доменами. Если домен А и домен Б полностью доверяют друг другу, то пользователь домена А может зарегистрироваться в домене Б и осуществлять доступ к ресурсам его сервера. Аналогично, пользователь домена Б может использовать ресурсы любого из серверов домена А.

Клиентами в сети с Windows NT Server могут являться компьютеры с различными операционными системами. Стандартно поддерживаются: MS-DOS, OS/2, Windows for Workgroups, клоны UNIX, Macintosh, Windows NT Workstation. Программное обеспечение возможных клиентов включается в стандартную поставку Windows NT Server.

Microsoft является одним из лидеров в установлении общественных стандартов на socket-интерфейсы для Windows. Windows Sockets является открытой спецификацией, определяющей программный интерфейс Windows к сетевым протоколам. Этот интерфейс также является частью Microsoft Windows Open Services Architecture (WOSA). Он уже знаком сетевым программистам, работающим под UNIX с расширениями на базе Windows, и стал стандартным

методом разработчиков, пишущих Windows-приложения для обеспечения удаленного вызова процедур (RPC) не только через TCP/IP, но и через IPX и NetBEUI. В 1993 году Microsoft создал свой Internet FTP сервер, работающий на базе Windows NT Advanced Server. Этот сервер можно найти в Internet как ftp.microsoft.com. В настоящее время в среднем 25 000 пользователей подключаются к этому серверу еженедельно и загружают с него общим числом около 75 000 файлов в неделю.

Взаимодействие с UNIX. В Windows NT обеспечивается посредством поддержки общих стандартных сетевых протоколов (включая TCP/IP), стандартных способов распределенной обработки, стандартных файловых систем и совместного использования данных, а также благодаря простоте переноса приложений. Несмотря на то, что система Windows NT была разработана для поддержки работы по схеме клиент-сервер, для совместимости с UNIX-хостами встроена эмуляция терминалов.

SNMP. В Windows NT имеется ряд средств для интеграции в системы, использующие протокол SNMP (Simple Network Management Protocol), что позволяет выполнять удаленное администрирование Windows NT с помощью, например, SUN Net Manager и HP OpenView. Поддержка графических и текстовых терминалов.

В Windows NT входят мощные *API гибкой поддержки сред распределенных вычислений:*

- DCE совместимый *RPC* (Remote Procedure Call) - критическая составная часть, необходимая при построении распределенных приложений;
- *Windows Sockets* - API, совместимый с сокетами типа Berkeley, популярным в UNIX механизмом распределенных вычислений;
- *WOSA (Windows Open Services Architecture)* - этот набор API позволяет объединять системы Windows с широким рядом приложений-поставщиков данных, выпускаемых самыми разными производителями.

Сети SNA. Доступ к мэйнфреймам IBM и системам IBM AS400 возможен при установке Microsoft SNA Server. SNA Sever является шлюзом, позволяющим осуществлять доступ с рабочей станции как к серверам локальной сети, так и к мэйнфреймам без необходимости использования двух сетевых карт или нескольких стеков сетевых протоколов. Это приводит к снижению стоимости оборудования и объема требуемой оперативной памяти. Обеспечивая прозрачный доступ к мэйнфреймам, SNA Server, будучи интегрированным с системой безопасности NT Server, обеспечивает авторизацию доступа к хосту. SNA Server может работать с любым из протоколов, поддерживаемых NT Server: IPX/SPX, TCP/IP или NetBEUI.

Требования к аппаратуре

Для работы с Windows NT Workstation (Server) 3.5 требуется:

- Компьютер с процессорами i386, i486 или Pentium с тактовой частотой от 33 Мгц и оперативной памятью от 12 (16) Мбайт; либо с процессором DEC Alpha и оперативной памятью от 16 (24) Мбайт; либо с MIPS-процессором; либо компьютер с несколькими процессорами.
- Не менее 70 (90) Мб свободного пространства на жестком диске.
- Желательно иметь устройство чтения компакт дисков с интерфейсом SCSI (CD-ROM) и мышь.
- При работе в сети, требуется наличие 16- или 32-разрядной сетевой карты Ethernet или Token Ring.

Дополнительно могут быть установлены звуковая карта, накопители на магнитных лентах, принтеры, графопостроители, модемы и другие периферийные устройства.

Полный список техники, прошедшей тестирование на совместимость с Windows NT, прилагается к системе.

Windows NT 4.0

При разработке Windows NT 4.0 Microsoft решила пожертвовать стабильностью ради производительности. С этой целью были внесены изменения в архитектуру: библиотеки менеджера окон и GDI, а также драйверы графических адаптеров были перенесены из пользовательского режима в режим ядра. Это изменение означает некоторый отход от принятой в предыдущих версиях Windows NT 3.x концепции микроядра.

Перенос графической библиотеки и драйверов в область ядра повышает скорость выполнения графического ввода-вывода. Эти изменения особенно сказались на скорости выполнения приложений Win32, в то время как приложения Windows-16 и DOS-ские графические приложения работают примерно также, как и в версии 3.5.

В то же время описанные изменения делают операционную систему в принципе менее надежной. Действительно, поскольку программное обеспечение графических адаптеров, как правило, разрабатывается фирмами-производителями этого оборудования и это программное обеспечение часто меняется (вместе с оборудованием), то от него трудно ожидать той надежности, которая требуется для модулей операционной системы.

Кроме архитектурных в Windows NT 4.0 имеются и другие не менее кардинальные изменения:

- Средства взаимодействия с NetWare модифицированы - Gateway и клиент NCP поддерживают теперь NDS.
- В стандартную поставку включен Internet Information Server и сервер DNS. DNS взаимодействует с WINS и DHCP-серверами. Эта комбинация

реализует Dynamic DNS, который разрешает верхние уровни доменного имени и передает имя для окончательного разрешения службе WINS.
- Поддержка многопротокольной маршрутизации.
- Сервер может работать как транслирующий агент протокола BOOTP/DHCP , что позволяет компьютеру передавать сообщения BOOTP/DHCP по IP-сети.
- Новые административные средства Windows NT могут работать удаленно на клиентах Windows 95. Кроме того, Windows NT Server обеспечивает сервис удаленной загрузки для клиентов Windows 95. (Это полезно для бездисковых рабочих станций.)
- Интерфейс в стиле Windows 95.
- Подсистема обработки сообщений Microsoft Windows Messaging Subsystem позволяет получать и отправлять почту из приложений.
- В Windows NT 4.0 появился эмулятор Intel'овских процессоров для RISC-платформ.

Но не известно как скажется на быстродействии распределенная версия OLE, названная Distributed COM (в Windows 95 добавление OLE снизило производительность).

Microsoft добавила в Windows NT 4.0 много технических средств, чтобы сделать эту операционную систему пригодной для использования в качестве платформы для Web-сервера.

Одно из усовершенствований связано с тем, что повышающаяся роль Internet'a и клиент-серверных систем ведет к росту числа мобильных пользователей. Microsoft в связи с этим улучшила RAS (улучшила поддержку ISDN) и предоставила средства безопасной работы с RAS через Internet. В RAS реализованы протоколы PPTP (создает зашифрованный трафик через Internet) и Multilink PPP (позволяет объединять несколько каналов в один). Клиентами могут быть Windows NT 4.0 Workstation или Windows 95. Важным аргументом в борьбе за Internet является включение в стандартную поставку Windows NT 4.0 Web-сервера производства Microsoft - Internet Information Server, возможности которого сравнимы, а по ряду тестов и превосходят аналогичный популярный продукт Server Netscape для NT.

Области использования Windows NT

Windows NT Workstation, прежде всего, может использоваться как клиент в сетях Windows NT Server, а также в сетях NetWare, UNIX, Vines. Она может быть рабочей станцией и в одноранговых сетях, выполняя одновременно функции и клиента, и сервера. Windows NT Workstation может применяться в качестве ОС автономного компьютера при необходимости обеспечения повышенной производительности, секретности, а также при реализации сложных графических приложений, например, в системах автоматизированного проектирования.

Windows NT Server может быть использован прежде всего как сервер в корпоративной сети. Здесь весьма полезной оказывается его возможность выполнять функции контроллера доменов, позволяя структурировать сеть и упрощать задачи администрирования и управления. Он используется также в качестве файл-сервера, принт-сервера, сервера приложений, сервера удаленного доступа и сервера связи (шлюза). Кроме того, Windows NT Server может быть использован как платформа для сложных сетевых приложений, особенно тех, которые построены с использованием технологии клиент-сервер.

Так, под управлением Windows NT Server может работать сервер баз данных Microsoft SQL Server, а также серверы баз данных других известных фирм, такие как Oracle и Sybase, Adabas и InterBase.

На платформе Windows NT Server может быть установлена новая мощная система администрирования Microsoft System Management Server, функцией которой является инвентаризация аппаратной и программной конфигурации компьютеров сети, автоматическая установка программных продуктов на рабочие станции, удаленное управление любым компьютером и мониторинг сети.

Windows NT Server может использоваться как сервер связи с мейнфреймам. Для этого создан специальный продукт Microsoft SNA Server, позволяющий легко объединить в одной сети IBM PC-совместимые рабочие станции и мощные мейнфреймы.

Наконец, Windows NT Server является платформой для нового производительного почтового сервера Microsoft Exchange.

Концепции Windows NT

Структура: NT executive и защищенные подсистемы

При разработке структуры Windows NT была в значительной степени использована концепция микроядра. В соответствии с этой идеей ОС разделена на несколько подсистем, каждая из которых выполняет отдельный набор сервисных функций - например, сервис памяти, сервис по созданию процессов, или сервис по планированию процессов. Каждый сервер выполняется в пользовательском режиме, выполняя цикл проверки запроса от клиента на одну из его сервисных функций. Клиент, которым может быть либо другая компонента ОС, либо прикладная программа, запрашивает сервис, посылая сообщение на сервер. Ядро ОС (или микроядро), работая в привилегированном режиме, доставляет сообщение нужному серверу, затем сервер выполняет операцию, после этого ядро возвращает результаты клиенту с помощью другого сообщения.

Структурно Windows NT может быть представлена в виде двух частей: часть операционной системы, работающая в режиме пользователя, и часть операционной системы, работающая в режиме ядра (рисунок 8.1).

Часть Windows NT, работающая в режиме ядра, называется executive - исполнительной частью. Она включает ряд компонент, которые управляют виртуальной памятью, объектами (ресурсами), вводом-выводом и файловой системой (включая сетевые драйверы), взаимодействием процессов и частично системой безопасности. Эти компоненты взаимодействуют между собой с помощью межмодульной связи. Каждая компонента вызывает другие с помощью набора тщательно специфицированных внутренних процедур.

Вторую часть Windows NT, работающую в режиме пользователя, составляют серверы - так называемые защищенные подсистемы. Серверы Windows NT называются защищенными подсистемами, так как каждый из них выполняется в отдельном процессе, память которого отделена от других процессов системой управления виртуальной памятью NT executive. Так как подсистемы автоматически не могут совместно использовать память, они общаются друг с другом посредством посылки сообщений. Сообщения могут передаваться как между клиентом и сервером, так и между двумя серверами. Все сообщения проходят через исполнительную часть Windows NT. Ядро Windows NT планирует нити защищенных подсистем точно так же, как и нити обычных прикладных процессов.

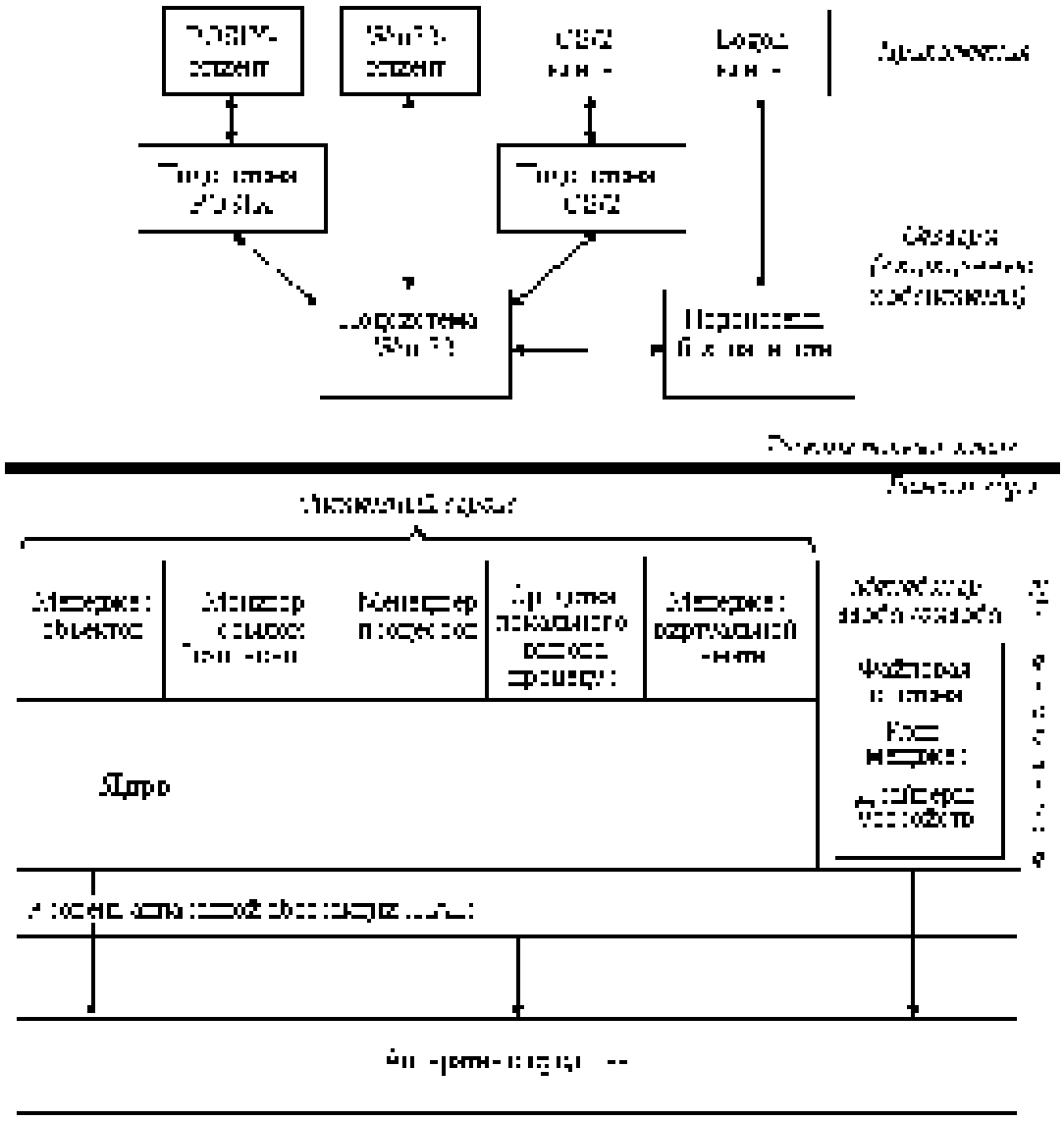


Рис. 8.1. Структура Windows NT

Поддержку защищенных подсистем обеспечивает исполнительная часть - Windows NT executive, которая работает в пространстве ядра и никогда не сбрасывается на диск. Ее составными частями являются:

- Менеджер объектов. Создает, удаляет и управляет объектами NT executive - абстрактными типами данных, используемых для представления ресурсов системы.
- Монитор безопасности. Устанавливает правила защиты на локальном компьютере. Охраняет ресурсы операционной системы, выполняет защиту и регистрацию исполняемых объектов.
- Менеджер процессов. Создает и завершает, приостанавливает и возобновляет процессы и нити, а также хранит о них информацию.
- Менеджер виртуальной памяти.
- Подсистема ввода-вывода. Включает в себя следующие компоненты:
    1.

- менеджер ввода-вывода, предоставляющий средства ввода-вывода, независимые от устройств;
- файловые системы - NT-драйверы, выполняющие файл-ориентированные запросы на ввод-вывод и транслирующие их в вызовы обычных устройств;
- сетевой редиректор и сетевой сервер - драйверы файловых систем, передающие удаленные запросы на ввод-вывод на машины сети и получающие запросы от них;
- драйверы устройств NT executive - низкоуровневые драйверы, которые непосредственно управляют устройством;
- менеджер кэша, реализующий кэширование диска.

Исполнительная часть, в свою очередь, основывается на службах нижнего уровня, предоставляемых ядром (его можно назвать и микроядром) NT. В функции ядра входит:

- планирование процессов,
- обработка прерываний и исключительных ситуаций,
- синхронизация процессоров для многопроцессорных систем,
- восстановление системы после сбоев.

Ядро работает в привилегированном режиме и никогда не удаляется из памяти. Обратиться к ядру можно только посредством прерывания. Ядро расположено над уровнем аппаратных абстракций (Hardware Abstraction Level HAL), который концентрирует в одном месте большую часть машинно-зависимых процедур. HAL располагается между NT executive и аппаратным обеспечением и скрывает от системы такие детали, как контроллеры прерываний, интерфейсы ввода/вывода и механизмы взаимодействия между процессорами. Такое решение позволяет легко переносить Windows NT с одной платформы на другую путем замены только слоя HAL.

При создании NT разработчики руководствовались задачами улучшения производительности и сетевых возможностей, а также требованием поддержки определенного набора прикладных сред. Эта цель была достигнута продуманным разделением функций между модулями ядра и остальными модулями. Например, передача данных в файловую систему и по сети производится быстрее в пространстве ядра, поэтому внутри ядра NT выделены буфера для небольших по объему (от 16 до 32 Кб) операций чтения и записи, являющихся типичными для приложений клиент-сервер и распределенных приложений. Размещение этих функций ввода-вывода внутри ядра, может, и портит академическую чистоту микроядра NT, но соответствует цели создания NT.

Защищенные подсистемы Windows NT работают в пользовательском режиме и создаются Windows NT во время загрузки операционной системы. Сразу после создания они начинают бесконечный цикл своего выполнения, отвечая на сообщения, поступающие к ним от прикладных процессов и других подсистем. Среди защищенных подсистем можно выделить подкласс, называемый подсистемами окружения. Подсистемы окружения реализуют интерфейсы приложений операционной системы (API). Другие типы подсистем, называемые интегральными подсистемами, исполняют необходимые операционной системе задачи. Например, большая часть системы безопасности Windows NT реализована в виде интегральной подсистемы, сетевые серверы также выполнены как интегральные подсистемы.

Наиболее важной подсистемой окружения является Win32 - подсистема, которая обеспечивает доступ для приложений к 32-bit Windows API. Дополнительно эта система обеспечивает графический интерфейс с пользователем и управляет вводом/выводом данных пользователя. Также поддерживаются подсистемы POSIX, OS/2,16-разрядная Windows и MS-DOS.

Каждая защищенная подсистема работает в режиме пользователя, вызывая системный сервис NT executive для выполнения привилегированных действий в режиме ядра. Сетевые серверы могут выполняться как в режиме пользователя, так и в режиме ядра, в зависимости от того, как они разработаны.

Подсистемы связываются между собой путем передачи сообщений. Когда, например, пользовательское приложение вызывает какую-нибудь API-процедуру, подсистема окружения, обеспечивающая эту процедуру, получает сообщение и выполняет ее либо обращаясь к ядру, либо посылая сообщение другой подсистеме. После завершения процедуры подсистема окружения посылает приложению сообщение, содержащее возвращаемое значение. Посылка сообщений и другая деятельность защищенных подсистем невидима для пользователя.

Основным средством, скрепляющим все подсистемы Windows NT в единое целое, является механизм вызова локальных процедур (Local Procedure Call - LPC). LPC представляет собой оптимизированный вариант более общего средства - удаленного вызова процедур (RPC), которое используется для связи клиентов и серверов, расположенных на разных машинах сети.

Средства LPC поддерживают несколько способов передачи данных между клиентами и серверами: один обычно используется для передачи коротких сообщений, другой - для длинных сообщений, а третий оптимизирован специально для использования подсистемой Win32. Каждая подсистема устанавливает порт - канал связи, посредством которого с ней могут связываться другие процессы. Порты реализуются как объекты.

Windows NT использует защищенные подсистемы для того, чтобы:

- Обеспечить несколько программных интерфейсов (API), по возможности не усложняя при этом базовый программный код (NT executive).
- Изолировать базовую операционную систему от изменений или расширений в поддерживаемых API.
- Объединить часть глобальных данных, требующихся всем API, и в то же время отделить данные, использующиеся каждым отдельным API от данных, использующихся другими API.
- Защитить окружение каждого API от приложений, а также от окружений других API, и защитить базовую операционную систему от различных окружений.
- Позволить операционной системе расширяться в будущем за счет новых API.

Таким образом, реализация частей ОС в виде серверов, выполняющихся в режиме пользователя, является важнейшей частью проекта Windows NT и оказывает глубокое воздействие на все функционирование системы.

Микроядро NT служит, главным образом, средством поддержки для переносимой основной части ОС - набора пользовательских сред. Концентрация машинно-зависимых программ внутри микроядра делает перенос NT на разнообразные процессоры относительно легким. Но в то время, как некоторые микроядра (Mach и Chorus)

предполагается поставлять в качестве самостоятельного программного продукта, из операционной системы Windows NT ядро вряд ли может быть вычленено для отдельного использования. Это является одной из причин того, что некоторые специалисты не считают Windows NT истинно микроядерной ОС в том смысле, в котором таковыми являются Mach и Chorus. Те же критики отмечают также, что NT не исключает, как это положено, все надстроенные службы из пространства ядра и что драйверы устройств в NT по минимуму взаимодействуют с ядром, предпочитая работать непосредственно с лежащим ниже слоем аппаратной абстракции HAL.

Множественные прикладные среды

При разработке NT важнейшим рыночным требованием являлось обеспечение поддержки по крайней мере двух уже существующих программных интерфейсов OS/2 и POSIX, а также возможности добавления других API в будущем.

Заметим, что для того, чтобы программа, написанная для одной ОС, могла быть выполнена в рамках другой ОС, недостаточно лишь обеспечить совместимость API. Кроме этого, необходимо обеспечить ей "родное" окружение: структуру процесса, средства управления памятью, средства обработки ошибок и исключительных ситуаций, механизмы защиты ресурсов и семантику файлового доступа. Отсюда ясно, что поддержка нескольких прикладных программных сред является очень сложной задачей, тесно связанной со структурой операционной системы. Эта задача была успешно решена в Windows NT, при этом в полной мере был использован опыт разработчиков ОС Mach из университета Карнеги-Меллона, которые смогли в своей клиент-серверной реализации UNIX'a отделить базовые механизмы операционной системы от серверов API различных ОС.

Windows NT поддерживает пять прикладных сред операционных систем: MS-DOS, 16-разрядный Windows, OS/2 1.x, POSIX и 32-разрядный Windows (Win32). Все пять прикладных сред реализованы как подсистемы окружения. Каждая работает в собственном защищенном пользовательском пространстве. Подсистема Win32 обеспечивает поддержку дисплея, клавиатуры и мыши для четырех оставшихся подсистем.

16-битовые приложения DOS и Windows работают на VDM (virtual DOS machines - виртуальные машины DOS), каждая из которых эмулирует полный 80x86 процессор с MS-DOS. В NT VDM является приложением Win32, значит, как и обычные модули прикладных сред для UNIX, приложения DOS и 16-битовой Windows расположены в слое непосредственно над подсистемой Win32.

Подсистемы OS/2 и POSIX построены по-другому. В качестве полноценных подсистем NT они могут взаимодействовать с подсистемой Win32 для получения доступа к вводу и выводу, но также могут обращаться непосредственно к исполнительной системе NT за другими средствами операционной системы. Подсистема OS/2 может выполнять многие имеющиеся приложения OS/2 символьного режима, включая OS/2 SQL Server, и поддерживает именованные каналы и NetBIOS.

Однако возможности подсистемы POSIX весьма ограничена, несмотря на непосредственный доступ ее к службам ядра. Приложения POSIX должны быть откомпилированы специально для Windows NT. NT не поддерживает двоичный код, предназначенный для других POSIX-совместимых систем, таких как UNIX. К тому же подсистема POSIX NT не поддерживает непосредственно печать, не поддерживает

сетевой доступ, за исключением доступа к удаленным файловым системам, и не поддерживает многие средства Win32, например, отображение на память файлов и графику.

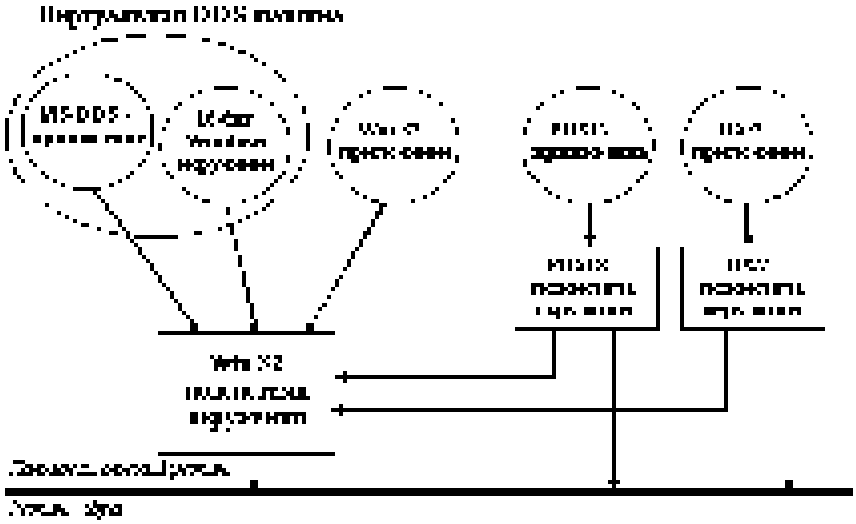


Рис. 8.2. Реализация множественных прикладных сред в Windows NT

На рисунке 8.2 показана структура, обеспечивающая в Windows NT поддержку множественных прикладных сред.

NT executive выполняет базовые функции операционной системы и является той основой, на которой подсистемы окружения реализуют поддержку своих приложений. Все подсистемы равноправны и могут вызвать "родные" функции NT для создания соответствующей среды для своих приложений.

Каждая подсистема окружения имеет свое представление о том, что такое, например, процесс или описатель файла, поэтому структуры данных, используемые в каждом окружении, могут не совпадать. Следовательно, как только подсистема Win32 передала прикладной процесс другой подсистеме окружения, данное приложение становится клиентом этой подсистемы вплоть до завершения процесса. При этом подсистема Win32 перенаправляет входные сообщения от пользователя этому приложению, а также отображает вывод приложения на экране.

Объектно-ориентированный подход

Хотя NT и не является полностью объектно-ориентированной, в ее основе лежат объекты. Единообразная форма именования, совместного использования и учета системных ресурсов, простой и дешевый способ обеспечения безопасности системы и ее модификации - все эти преимущества могут быть достигнуты при использовании объектной модели.

В Windows NT любой ресурс системы, который одновременно может быть использован более чем одним процессом, включая файлы, совместно используемую память и физические устройства, реализован в виде объекта и управляется рядом функций. Такой подход сокращает число изменений, которые необходимо внести в операционную систему в процессе ее эксплуатации. Если, скажем, изменилось что-то в аппаратуре, то все, что необходимо сделать - заменить соответствующий объект. Аналогично, если требуется

поддержка новых ресурсов, то надо добавить только новый объект, не изменяя при этом остального кода операционной системы.

Наиболее фундаментальное отличие между объектом и обыкновенной структурой данных заключается в том, что внутренняя структура данных объекта скрыта от наблюдения. Вы должны вызвать объектную функцию для того, чтобы получить данные из объекта или поместить данные в объект. Вы не можете непосредственно изменять данные, находящиеся внутри объекта. Это отделяет средства реализации объекта от кода, который только использует его, такая техника позволяет легко изменять в последствии реализацию объектов.

Группа разработчиков NT executive решила использовать объекты для представления системных ресурсов, потому что объекты обеспечивают централизованные средства для выполнения трех важных (и часто утомительных) задач ОС:

- Поддержка воспринимаемых человеком имен системных ресурсов;
- Разделение ресурсов и данных между процессами;
- Защита ресурсов от несанкционированного доступа.

Не все структуры данных в NT executive являются объектами. Объектами сделаны только такие данные, которые нужно разделять, защищать, именовать или делать видимыми для программ пользовательского режима (с помощью системных функций). Структуры, которые используются только одним компонентом executive для выполнения внутренних функций, не являются объектами.

Несмотря на всестороннее использование объектов для представления разделяемых ресурсов, Windows NT не является объектно-ориентированной системой в строгом смысле. Большая часть кода операционной системы написана на С с целью обеспечения переносимости. Несмотря на то, что С не поддерживает непосредственно объектно-ориенти-
рованные конструкции, такие как динамическое связывание типов данных, полиморфные функции или наследование классов, эти инструментальные средства были использованы из-за их широкой распространенности.

Менеджер объектов - это компонента NT executive, которая ответственна за создание, удаление, защиту и слежение за NT-объектами. Менеджер объектов централизует операции управления ресурсами, которые в противном случае будут разбросаны по всей ОС.

Менеджер объектов NT выполняет следующие функции:

- Выделяет память для объекта.
- Присоединяет к объекту так называемый дескриптор безопасности, который определяет, кому разрешено использовать объект, и что они могут с ним делать.
- Создает и манипулирует структурой каталога объектов, в котором хранятся имена объектов.
- Создает описатель объекта и возвращает его вызывающему процессу.

Процессы пользовательского режима, включая подсистемы окружения, должны иметь описатель объекта перед тем, как их нити смогут использовать этот объект. Использование описателей для работы с системными ресурсами не является новой идеей. Например, библиотеки С и Паскаля (а также других языков) возвращают описатели для

открытых файлов. Аналогично приложения Win32 используют различные типы описателей для управления окнами, курсором мыши, иконками. В обоих случаях описатели служат косвенными указателями на системные ресурсы; эта косвенность предохраняет прикладные программы от рутинной работы непосредственно с системными структурами данных.

Каждый NT-объект является объектом определенного типа. Тип определяет данные, которые хранит объект, и "родные" системные функции, которые могут к нему применяться. Для того, чтобы управлять различными объектами единообразно, менеджер объектов требует, чтобы каждый объект содержал несколько полей стандартной информации в определенном месте объекта. До тех пор, пока эти данные имеются, менеджер объектов не заботится о том, что еще хранится в объекте. Каждый объект состоит из двух частей - заголовка объекта и тела объекта, которые содержат стандартные и переменные данные объекта соответственно. Менеджер объектов работает с заголовком объекта, а другие компоненты executive работают с телами объектов тех типов, которые они сами создают. Заголовок объекта используется менеджером без учета типа объекта. В заголовке объекта любого типа содержится имя, каталог, дескриптор безопасности, квоты на использование ресурсов, счетчик открытых описателей, база данных открытых описателей, признак постоянный/временный, режим пользователя/ядра, указатель на тип объекта.

Кроме заголовка объекта, каждый объект имеет тело объекта, формат и содержание которого уникально определяется типом этого объекта; у всех объектов одного и того же типа одинаковый формат тела. При создании объекта исполнительная часть может оперировать данными в телах всех объектов этого типа.

Процессы и нити

В разных ОС процессы реализуются по-разному. Эти различия заключаются в том, какими структурами данных представлены процессы, как они именуются, какими способами защищены друг от друга и какие отношения существуют между ними. Процессы Windows NT имеют следующие характерные свойства:

- Процессы Windows NT реализованы в форме объектов, и доступ к ним осуществляется посредством службы объектов.
- Процесс Windows NT имеет многонитевую организацию.
- Как объекты-процессы, так и объекты-нити имеют встроенные средства синхронизации.
- Менеджер процессов Windows NT не поддерживает между процессами отношений типа "родитель-потомок".

В любой системе понятие "процесс" включает следующее:

- исполняемый код,
- собственное адресное пространство, которое представляет собой совокупность виртуальных адресов, которые может использовать процесс,
- ресурсы системы, такие как файлы, семафоры и т.п., которые назначены процессу операционной системой.
- хотя бы одну выполняемую нить.

Адресное пространство каждого процесса защищено от вмешательства в него любого другого процесса. Это обеспечивается механизмами виртуальной памяти. Операционная система, конечно, тоже защищена от прикладных процессов. Чтобы выполнить какую-

либо процедуру ОС или прочитать что-либо из ее области памяти, нить должна выполняться в режиме ядра. Пользовательские процессы получают доступ к функциям ядра посредством системных вызовов. В пользовательском режиме выполняются не только прикладные программы, но и защищенные подсистемы Windows NT.

В Windows NT процесс - это просто объект, создаваемый и уничтожаемый менеджером объектов. Объект-процесс, как и другие объекты, содержит заголовок, который создает и инициализирует менеджер объектов. Менеджер процессов определяет атрибуты, хранимые в теле объекта-процесса, а также обеспечивает системный сервис, который восстанавливает и изменяет эти атрибуты.

В число атрибутов тела объекта-процесса входят:

- Идентификатор процесса - уникальное значение, которое идентифицирует процесс в рамках операционной системы.
- Токен доступа - исполняемый объект, содержащий информацию о безопасности.
- Базовый приоритет - основа для исполнительного приоритета нитей процесса.
- Процессорная совместимость - набор процессоров, на которых могут выполняться нити процесса.
- Предельные значения квот - максимальное количество страничной и нестраничной системной памяти, дискового пространства, предназначенного для выгрузки страниц, процессорного времени - которые могут быть использованы процессами пользователя.
- Время исполнения - общее количество времени, в течение которого выполняются все нити процесса.

Напомним, что нить является выполняемой единицей, которая располагается в адресном пространстве процесса и использует ресурсы, выделенные процессу. Подобно процессу нить в Windows NT реализована в форме объекта и управляется менеджером объектов.

Объект-нить имеет следующие атрибуты тела:

- Идентификатор клиента - уникальное значение, которое идентифицирует нить при ее обращении к серверу.
- Контекст нити - информация, которая необходима ОС для того, чтобы продолжить выполнение прерванной нити. Контекст нити содержит текущее состояние регистров, стеков и индивидуальной области памяти, которая используется подсистемами и библиотеками.
- Динамический приоритет - значение приоритета нити в данный момент.
- Базовый приоритет - нижний предел динамического приоритета нити.
- Процессорная совместимость нитей - перечень типов процессоров, на которых может выполняться нить.
- Время выполнения нити - суммарное время выполнения нити в пользовательском режиме и в режиме ядра, накопленное за период существования нити.
- Состояние предупреждения - флаг, который показывает, что нить должна выполнять вызов асинхронной процедуры.
- Счетчик приостановок - текущее количество приостановок выполнения нити.

Кроме перечисленных, имеются и некоторые другие атрибуты.

Как видно из перечня, многие атрибуты объекта-нити аналогичны атрибутам объекта-процесса. Весьма сходны и сервисные функции, которые могут быть выполнены над объектами-процессами и объектами-нитями: создание, открытие, завершение,

приостановка, запрос и установка информации, запрос и установка контекста и другие функции.

Алгоритм планирования процессов и нитей

В Windows NT реализована вытесняющая многозадачность, при которой операционная система не ждет, когда нить сама захочет освободить процессор, а принудительно снимает ее с выполнения после того, как та израсходовала отведенное ей время (квант), или если в очереди готовых появилась нить с более высоким приоритетом. При такой организации разделения процессора ни одна нить не займет процессор на очень долгое время.

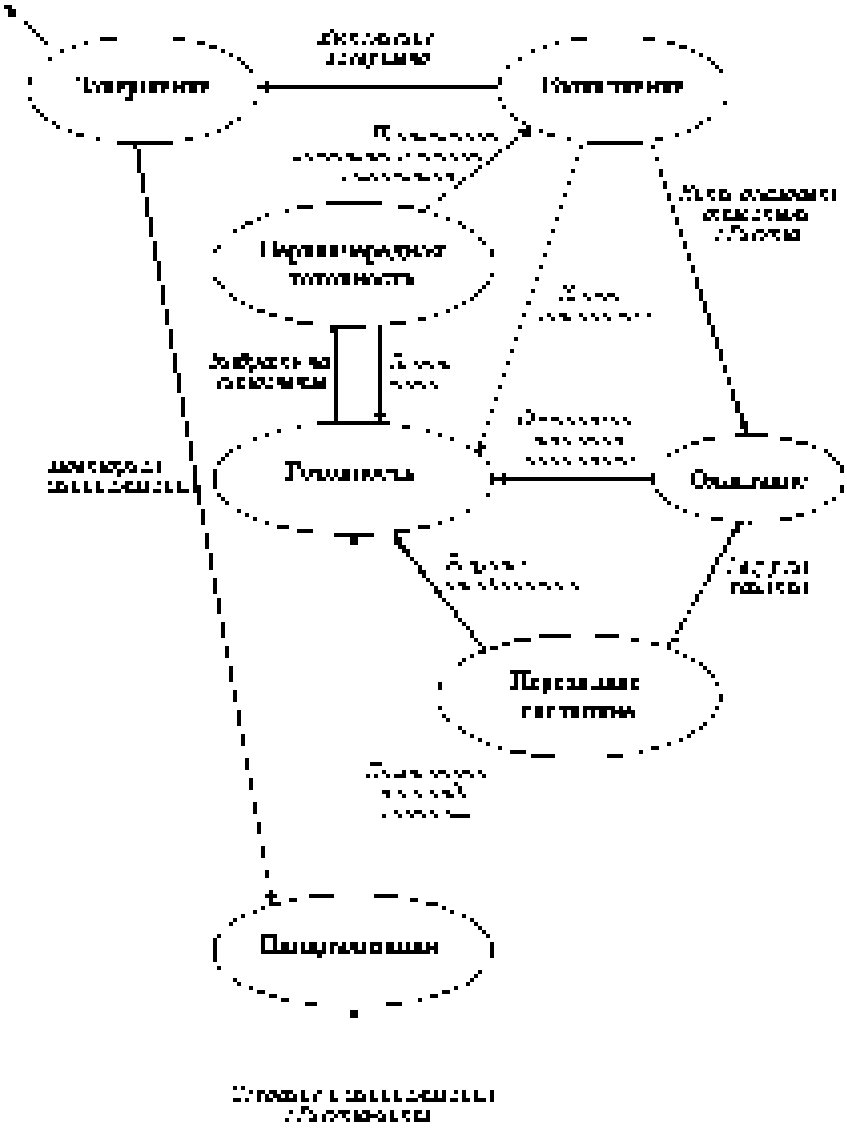


Рис. 8.3. Граф состояний нити

В ОС Windows NT нить в ходе своего существования может иметь одно из шести состояний (рисунок 8.3). Жизненный цикл нити начинается в тот момент, когда программа создает новую нить. Запрос передается NT executive, менеджер процессов выделяет память для объекта-нити и обращается к ядру, чтобы инициализировать объект-нить ядра. После инициализации нить проходит через следующие состояния:

- *Готовность*. При поиске нити на выполнение диспетчер просматривает только нити, находящиеся в состоянии готовности, у которых есть все для выполнения, но не хватает только процессора.
- *Первоочередная готовность* (standby). Для каждого процессора системы выбирается одна нить, которая будет выполняться следующей (самая первая нить в очереди). Когда условия позволяют, происходит переключение на контекст этой нити.
- *Выполнение*. Как только происходит переключение контекстов, нить переходит в состояние выполнения и находится в нем до тех пор, пока либо ядро не вытеснит ее из-за того, что появилась более приоритетная нить или закончился квант времени, выделенный этой нити, либо нить завершится вообще, либо она по собственной инициативе перейдет в состояние ожидания.
- *Ожидание*. Нить может входить в состояние ожидания несколькими способами: нить по своей инициативе ожидает некоторый объект для того, чтобы синхронизировать свое выполнение; операционная система (например, подсистема ввода-вывода) может ожидать в интересах нити; подсистема окружения может непосредственно заставить нить приостановить себя. Когда ожидание нити подойдет к концу, она возвращается в состояние готовности.
- *Переходное состояние*. Нить входит в переходное состояние, если она готова к выполнению, но ресурсы, которые ей нужны, заняты. Например, страница, содержащая стек нити, может быть выгружена из оперативной памяти на диск. При освобождении ресурсов нить переходит в состояние готовности.
- *Завершение*. Когда выполнение нити закончилось, она входит в состояние завершения. Находясь в этом состоянии, нить может быть либо удалена, либо не удалена. Это зависит от алгоритма работы менеджера объектов, в соответствии с которым он и решает, когда удалять объект. Если executive имеет указатель на объект-нить, то она может быть инициализирована и использована снова.

Диспетчер ядра использует для определения порядка выполнения нитей алгоритм, основанный на приоритетах, в соответствии с которым каждой нити присваивается число - приоритет, и нити с более высоким приоритетом выполняются раньше нитей с меньшим приоритетом. В самом начале нить получает приоритет от процесса, который создает ее. В свою очередь, процесс получает приоритет в тот момент, когда его создает подсистема той или иной прикладной среды. Значение базового приоритета присваивается процессу системой по умолчанию или системным администратором. Нить наследует этот базовый приоритет и может изменить его, немного увеличив или уменьшив. На основании получившегося в результате приоритета, называемого приоритетом планирования, начинается выполнение нити. В ходе выполнения приоритет планирования может меняться.

Windows NT поддерживает 32 уровня приоритетов, разделенных на два класса - класс реального времени и класс переменных приоритетов. Нити реального времени, приоритеты которых находятся в диапазоне от 16 до 31, являются более приоритетными процессами и используются для выполнения задач, критичных ко времени.

Каждый раз, когда необходимо выбрать нить для выполнения, диспетчер прежде всего просматривает очередь готовых нитей реального времени и обращается к другим нитям, только когда очередь нитей реального времени пуста. Большинство нитей в системе попадают в класс нитей с переменными приоритетами, диапазон приоритетов которых от 0 до 15. Этот класс имеет название "переменные приоритеты" потому, что диспетчер настраивает систему, выбирая (понижая или повышая) приоритеты нитей этого класса.

Алгоритм планирования нитей в Windows NT объединяет в себе обе базовых концепции - квантование и приоритеты. Как и во всех других алгоритмах, основанных на *квантовании*,

каждой нити назначается квант, в течение которого она может выполняться. Нить освобождает процессор, если:

- блокируется, уходя в состояние ожидания;
- завершается;
- исчерпан квант;
- в очереди готовых появляется более приоритетная нить.

Использование *динамических приоритетов*, изменяющихся во времени, позволяет реализовать адаптивное планирование, при котором не дискриминируются интерактивные задачи, часто выполняющие операции ввода-вывода и недоиспользующие выделенные им кванты. Если нить полностью исчерпала свой квант, то ее приоритет понижается на некоторую величину. В то же время приоритет нитей, которые перешли в состояние ожидания, не использовав полностью выделенный им квант, повышается. Приоритет не изменяется, если нить вытеснена более приоритетной нитью.

Для того, чтобы обеспечить хорошее время реакции системы, алгоритм планирования использует наряду с квантованием концепцию *абсолютных приоритетов*. В соответствии с этой концепцией при появлении в очереди готовых нитей такой, у которой приоритет выше, чем у выполняющейся в данный момент, происходит смена активной нити на нить с самым высоким приоритетом.

В многопроцессорных системах при диспетчеризации и планировании нитей играет роль их процессорная совместимость: после того, как ядро выбрало нить с наивысшим приоритетом, оно проверяет, какой процессор может выполнить данную нить и, если атрибут нити "процессорная совместимость" не позволяет нити выполняться ни на одном из свободных процессоров, то выбирается следующая в порядке приоритетов нить.

Сетевые средства

Средства сетевого взаимодействия Windows NT направлены на реализацию взаимодействия с существующими типами сетей, обеспечение возможности загрузки и выгрузки сетевого программного обеспечения, а также на поддержку распределенных приложений.

Windows NT с точки зрения реализации сетевых средств имеет следующие особенности:

- Встроенность на уровне драйверов. Это свойство обеспечивает быстродействие.
- Открытость - обуславливается легкостью динамической загрузки-выгрузки, мультиплексируемостью протоколов.
- Наличие RPC, именованных конвейеров и почтовых ящиков для поддержки распределенных приложений .
- Наличие дополнительных сетевых средств, позволяющих строить сети в масштабах корпорации: дополнительные средства безопасности централизованное администрирование отказоустойчивость (UPS, зеркальные диски).

Windows NT унаследовала от своих предшественников редиректор и сервер, протокол верхнего уровня SMB и транспортный протокол NetBIOS (правда, с новым "наполнением" - NetBEUI). Как и в сети MS-NET редиректор перенаправляет локальные запросы ввода-вывода на удаленный сервер, а сервер принимает и обрабатывает эти запросы.

Сначала редиректор и сервер были написаны на ассемблере и располагались над существующим системным программным обеспечением MS-DOS. Новые редиректор и сервер встроены в Windows NT, они не зависят от архитектуры аппаратных средств, на которых работает ОС. Они написаны на С и выполнены как загружаемые драйверы файловой системы, которые могут загружаться или выгружаться в любое время. Они также могут сосуществовать с редиректорами и серверами других производителей.

Реализация редиректора и сервера как драйверов файловой системы делают их частью NT executive. Следовательно, они имеют доступ к специализированным интерфейсам, которые менеджер ввода-вывода обеспечивает для драйверов. Эти интерфейсы, в свою очередь, были разработаны с учетом нужд сетевых компонент. Доступ к интерфейсам драйверов плюс возможности непосредственного вызова кэш-менеджера дают значительный вклад в повышение производительности редиректора и сервера. Многоуровневая модель драйверов менеджера ввода-вывода отражает многоуровневую модель сетевых протоколов. Так как редиректор и сервер являются драйверами, то они могут быть размещены на верхнем уровне, под которым располагаются все необходимые драйверы транспортных протоколов. Такая структура обеспечивает модульность сетевых компонент и создает эффективный путь от уровня редиректора или сервера вниз к транспортному и физическому уровням сети.

Сетевой редиректор обеспечивает средства, необходимые одному компьютеру Windows NT для доступа к файлам и принтерам другого компьютера. Так как он поддерживает SMB-протокол, то он работает с существующими серверами MS-NET и LAN Manager, обеспечивая доступ к системам MS-DOS, Windows и OS/2 из Windows NT. Механизмы безопасности обеспечивают защиту данных Windows NT, разделяемых по сети, от несанкционированного доступа.

Редиректор имеет одну основную задачу: поддержку распределенной файловой системы, которая ведет себя подобно локальной файловой системе, хотя и работает через ненадежную среду (сеть). Когда связь отказывает, редиректор ответственен за восстановление соединения, если это возможно, или же за возврат кода ошибки, чтобы приложение смогло повторить операцию.

Подобно другим драйверам файловой системы, редиректор должен поддерживать асинхронные операции ввода-вывода, если они вызываются. Когда пользовательский запрос является асинхронным, то редиректор должен вернуть управление немедленно, независимо от того, завершилась ли удаленная операция ввода-вывода или нет. При этом редиректор выполняется в контексте этой нити. Вызывающая нить должна продолжить свою работу, а редиректор должен ждать завершения запущенной операции. Есть два варианта решения этой проблемы: или редиректор сам создает новую нить, которая будет ждать, или он может передать эту работу уже готовой нити, существующей в системе. В Windows NT реализован второй вариант.

Редиректор отправляет и получает блоки SMB для выполнения своей работы. Протокол SMB является протоколом прикладного уровня, включающим сетевой уровень и уровень представления.

SMB реализует:

- установление сессии,
- файловый сервис,
- сервис печати,

- сервис сообщений.

Интерфейс, в соответствии с которым редиректор посылает блоки SMB, называется *интерфейсом транспортных драйверов* (transport driver interface - TDI). Редиректор вызывает функции TDI для передачи блоков SMB различным транспортным драйверам, загруженным в Windows NT. Для вызова функций TDI редиректор должен открыть канал, называемый виртуальной связью (virtual circuit), к машине назначения, а затем послать SMB-сообщение через эту виртуальную связь. Редиректор создает только одну виртуальную связь для каждого сервера, с которым соединена система Windows NT, и мультиплексирует через нее запросы к этому серверу. Транспортный уровень определяет, каким образом реализовать виртуальную связь, и пересылает данные через сеть.

Как и редиректор, сервер Windows NT на 100% совместим с существующими SMB-протоколами MS-NET и LAN Manager. Эта полная совместимость позволяет серверу обрабатывать запросы, исходящие не только от систем Windows NT, но и от других систем, работающих с программным обеспечением LAN Manager. Как и редиректор, сервер выполнен в виде драйвера файловой системы.

Может показаться странным, что сервер в соответствии с микроядерной концепцией не реализован как серверный процесс. Было бы логично ожидать, что сетевой сервер будет функционировать как защищенная подсистема - процесс, чьи нити ожидают поступления запросов по сети, выполняют их, а затем возвращают результаты по сети. Этот подход, как наиболее естественный, был тщательно рассмотрен при проектировании Windows NT, однако, учитывая опыт построения сетей VAX/VMS и опыт использования RPC, было решено выполнить сервер как драйвер файловой системы. Хотя сервер и не является драйвером в обычном смысле, и он не управляет файловой системой на самом деле, использование модели драйвера обеспечивает некоторые преимущества.

Главное из них состоит в том, что драйвер реализован в среде NT executive и может вызывать кэш-менеджер NT непосредственно, что оптимизирует передачу данных. Например, когда сервер получает запрос на чтение большого количества данных, он вызывает кэш-менеджер для определения места расположения этих данных в кэше (или для загрузки этих данных в кэш, если их там нет) и для фиксации данных в памяти. Затем сервер передает данные непосредственно из кэша в сеть, минуя доступ к диску. Аналогично, при запросе на запись данных сервер вызывает кэш-менеджер для резервирования места для поступающих данных. Затем сервер пишет данные непосредственно в кэш. Записывая данные в кэш, сервер возвращает управление клиенту гораздо быстрее; затем кэш-менеджер записывает данные на диск в фоновом режиме (используя страничные средства менеджера виртуальной памяти).

Будучи драйвером файловой системы, сервер несколько более гибок по сравнению с его реализацией в виде процесса. Например, он может регистрировать функции завершения ввода-вывода, что позволяет ему получать управление немедленно после завершения работы драйверов нижнего уровня. Хотя сервер Windows NT реализован как драйвер файловой системы, другие серверы могут быть реализованы и как драйверы, и как серверные процессы.

Асинхронные вызовы обрабатываются сервером аналогично, с использованием пула рабочих нитей.

И редиректоры, и серверы, и транспортные драйверы могут быть в любое время загружены и выгружены.

Открытая архитектура сетевых средств Windows NT обеспечивает работу своих рабочих станций (и серверов) в гетерогенных сетях не только путем предоставления возможности динамически загружать и выгружать сетевые средства, но и путем непосредственного переключения с программных сетевых средств, ориентированных на взаимодействие с одним типом сетей, на программные средства для другого типа сетей в ходе работы системы. Windows NT поддерживает переключение программных средств на трех уровнях:

- на уровне редиректоров - каждый редиректор предназначен для своего протокола (SMP, NCP, NFS, VINES);
- на уровне драйверов транспортных протоколов, предоставляя для них и для редиректоров стандартный интерфейс TDI;
- на уровне драйверов сетевых адаптеров - со стандартным интерфейсом NDIS 3.0.

Для доступа к другим типам сетей в Windows NT, помимо встроенного, могут загружаться дополнительные редиректоры. Специальные компоненты Windows NT решают, какой редиректор должен быть вызван для обслуживания запроса на удаленный ввод-вывод. За последние десятилетия получили распространение различные протоколы передачи информации по сети. И хотя Windows NT поддерживает не все эти протоколы, она, по крайней мере, разрешает включать их поддержку.

После того, как сетевой запрос достигает редиректора, он должен быть передан в сеть. В традиционной системе каждый редиректор жестко связан с определенным транспортным протоколом. В Windows NT поставлена задача гибкого подключения того или иного транспортного протокола, в зависимости от типа транспорта, используемого в другой сети. Для этого во всех редиректорах нижний уровень должен быть написан в соответствии с определенными соглашениями, которые и определяют единый программный интерфейс, называемый *интерфейсом транспортных драйверов* (TDI).

TDI позволяет редиректорам оставаться независимым от транспорта. Таким образом, одна версия редиректора может пользоваться любым транспортным механизмом. TDI обеспечивает набор функций, которые редиректоры могут использовать для пересылки любых типов данных с помощью транспортного уровня. TDI поддерживает как связи с установлением соединения (виртуальные связи), так и связи без установления соединения (датаграммные связи). Хотя LAN Manager использует связи с установлением соединений, Novell IPX является примером сети, которая использует связь без установления соединения. Microsoft изначально обеспечивает транспорты - NetBEUI (NetBIOS Extended User Interface), TCP/IP, IPX/SPX, DECnet и AppleTalk.

Сетевые адаптеры поставляются вместе с сетевыми драйверами, которые раньше часто были рассчитаны на взаимодействие с определенным типом транспортного протокола. Так как Windows NT позволяет загружать драйверы различных транспортных протоколов, то производители сетевых адаптеров, использующие такой подход, должны были писать различные варианты одного и того же драйвера, рассчитанные на связь с разными протоколами транспортного уровня.

Чтобы помочь производителям избежать этого, Windows NT обеспечивает интерфейс и программную среду, называемые "*спецификация интерфейса сетевого драйвера*" (NDIS), которые экранируют сетевые драйверы от деталей различных транспортных протоколов.

Самый верхний уровень драйвера сетевого адаптера должен быть написан в соответствии с рекомендациями NDIS. В этом случае пользователь может работать с сетью TCP/IP и сетью NetBEUI (или DECnet, NetWare, VINES и т.п.), используя один сетевой адаптер и один сетевой драйвер. Среда NDIS использовалась в сетях LAN Manager, но для Windows NT она была обновлена.

Через свою нижнюю границу драйвер сетевого адаптера обычно взаимодействует непосредственно с адаптером или адаптерами, которые он обслуживает. Драйвер сетевого адаптера, реализованный для среды NDIS, управляет адаптером не непосредственно, а использует для этого функции, предоставляемые NDIS (например, для запуска ввода-вывода или обработки прерываний). Таким образом, среда NDIS образует некую оболочку, которая позволяет достаточно просто переносить драйверы сетевых адаптеров из одной ОС в другую. NDIS позволяет сетевым драйверам не содержать встроенных знаний о процессоре или операционной системе, на которых он работает.

Совместимость Windows NT с NetWare

Совместимость сетевых операционных систем предполагает использование одинакового стека коммуникационных протоколов, в том числе и верхнего прикладного уровня. Протоколы верхнего уровня (NCP, SMB, NFS, FTP, telnet) включают две части - клиентскую и серверную. При взаимодействии двух компьютеров на каждой стороне могут присутствовать как обе части прикладного протокола, так и по одной его части, в зависимости от этого образуется или одна, или две пары "клиент-сервер".

Для клиентской части протокола верхнего уровня, реализованного в виде модуля операционной системы, используются разные названия - редиректор (redirector), инициатор запросов или запросчик (requester). Эти компоненты получают запросы от приложений на доступ к удаленным ресурсам, расположенным на серверах, и ведут диалог с сервером в соответствии с каким-либо протоколом прикладного уровня. Совокупность функций, которая может использовать приложение для обращения к редиректору, называется прикладным интерфейсом (API) редиректора.

Существующая версия Windows NT 3.51 имеет встроенную поддержку стека протоколов Novell, а именно протоколов IPX/SPX и клиентской части NCP. При разработке первой версии Windows NT 3.1 между Microsoft и Novell существовало соглашение о том, что редиректор, реализующий клиентскую часть протокола NCP, будет написан силами сотрудников Novell и передан Microsoft в течение 60 дней после выпуска коммерческой версии Windows NT 3.1. Однако первая версия редиректора от Novell появилась только спустя четыре месяца и обладала существенными ограничениями: не поддерживался полностью API редиректора NetWare, в частности, поддерживались только 32-х разрядные вызовы, что означало невозможность работы старых 16 разрядных приложений клиента NetWare.

Через некоторое время Microsoft разработала свою собственную версию редиректора для NetWare, проведя большую работу по освоению NCP. Этот вариант оказался гораздо лучше, однако и он имеет недостатки: в нем отсутствует поддержка входных сценариев NetWare и службы каталогов NetWare Directory Services. Отсутствие поддержки входных сценариев означает, что администратору сети будет сложно автоматизировать создание индивидуальной операционной среды NetWare для пользователей, использующих Windows NT в качестве клиентской машины серверов NetWare.

Организация, использующая NetWare, может добавить Windows NT в качестве:

- клиентской рабочей станции,
- файлового сервера или сервера печати наряду с NetWare,
- файлового сервера или сервера печати вместо NetWare,
- сервера баз данных или других приложений.

Сеть с файловыми серверами различных типов (NetWare и Windows NT) порождает сложные технические проблемы. Даже если серверы используют одинаковые транспортные протоколы, в данном случае протокол IPX (в реализации Microsoft имеющий название NWLink), клиентским рабочим станциям все равно придется загружать два разных инициатора запросов. У клиента, работающего в среде MS-DOS, для этого может просто не хватить памяти.

Для смягчения перехода от NetWare к Windows NT Server разработано несколько инструментальных программ, в том числе утилита Migration Tool, которая включена в комплект поставки Windows NT Server. Эта утилита переносит учетную информацию пользователей (имена пользователей, ограничения и права доступа) и данные с одного или нескольких файловых серверов NetWare на сервер Windows NT. Migration Tool подбирает наилучшее соответствие между возможностями NetWare и возможностями Windows NT. Однако имеется ряд существенных различий в том, как обрабатываются такие вещи, как ограничения. В NetWare подобная информация обрабатывается для каждого пользователя в отдельности, а в Windows NT она общая для целого сервера.

Компания Beame and Whiteside Software создала первый NFS сервер для Windows NT, а также продукт под названием BW-Multiconnect, который превращает сервер Windows NT в сервер NetWare. Системы Windows NT с установленным продуктом BW-Multiconnect посылают широковещательные сообщения по протоколу SAP (протокол объявления сервисов и серверов по сети - Service Advertising Protocol, с помощью которого клиенты NetWare узнают о наличии в сети серверов и о тех услугах, которые они предоставляют). BW-Multiconnect должен облегчить сосуществование и миграцию от NetWare к Windows NT. Хотя он и может работать как единственный NCP-сервер сети, он не предназначен для этой роли, так как предоставляет лишь ограниченный набор утилит под Windows и DOS, и не обрабатывает входных командных файлов NetWare. Но когда в сети есть "настоящий" файловый сервер NetWare, то пользователи могут войти в этот сервер, выполнить системный входной командный файл, а затем подсоединиться к серверу Windows NT. Этот продукт превращает в сервер NetWare как Windows NT Server, так и Windows NT Workstation.

Microsoft ведет работу над созданием своих собственных файл- и принт-серверов NetWare для Windows NT. Кроме этого, скоро должен появиться редиректор NetWare для Windows NT, поддерживающий NDS.

Рассмотренные способы организации взаимодействия сетей построены на использовании принципа мультиплексирования протоколов. Другим подходом является использование шлюза. Шлюз действует как транслятор, что позволяет получать доступ к файлам и ресурсам печати на файловом сервере NetWare, не пользуясь ничем, кроме загруженного редиректора Windows NT. Шлюз преобразовывает SMB-сообщения, посланные каким-либо Windows NT-клиентом, в NCP-сообщения, которые посылаются на серверы NetWare. В этом случае имеется экономия памяти на клиентских машинах, так как не требуется загружать дополнительные редиректоры.

Вариант шлюза подходит только для приложений, использующих для запросов к серверу NetWare только стандартный API, а при использовании специфического для NetWare API нельзя обойтись без установки дополнительного редиректора.

Если NetWare-шлюз загружен, Windows NT Server может подсоединиться к одному или нескольким файловым серверам NetWare и подключиться к любому дисковому тому, очереди на печать или каталогу. После того, как сервер подключился к ресурсам, их можно начинать использовать совместно с другими пользователями через File Manager или Print manager, как если бы они были локальными ресурсами. То есть пользователи, вошедшие в домен, на сервере которого установлен шлюз к NetWare, получают доступ к серверам NetWare.

Трансляция протоколов в шлюзе замедляет доступ к серверу NetWare по сравнению с доступом через редиректор клиента. При тестировании замедление в малозагруженном шлюзе составило от 10% до 15%.

Имя пользователя, используемое шлюзом для входа в сервер NetWare, должно входить в группу NTGateway на сервере Windows NT. Разрешение на доступ к ресурсам NetWare предоставляется пользователям сервером Windows NT точно так же, как если бы это были его локальные ресурсы.

Средства BackOffice

Компания Microsoft объединяет под названием BackOffice набор своих серверов: сервер Windows NT Server, составляющий основу для построения остальных специализированных серверов: сервера баз данных Microsoft SQL Server, почтового сервера Microsoft Mail Server и сервера интегрированной службы обработки сообщений Microsoft Exchange, шлюз к SNA-сетям Microsoft SNA Server и сервер управления вычислительной системой Microsoft System Management Server. Все эти продукты хорошо работают вместе, образуя интегрированную и управляемую систему специализированных серверов офиса. В этой интегрированной среде можно построить приложения модели клиент-сервер практически любого масштаба, используя серверные возможности базы данных, почты, средств организации групповой работы.

Сервер баз данных SQL Server

Этот сервер представляет собой систему управления реляционными базами данных высшего класса, построенную в архитектуре клиент-сервер.

Интегрируемость и открытость

SQL Server достаточно легко интегрируется со всеми существующими на сей день клиентами - настольными компьютерами и системами на базе хостов. В перечень поддерживаемых клиентов входят: Windows 3.1, Windows for Workgroups 3.11, Windows NT Workstation, MS-DOS, OS/2 и Apple Macintosh. Для поддержки клиентов, работающих на UNIX и VMS, можно воспользоваться программным обеспечением Open Client Software компании Sybase. Сетевая поддержка включает Microsoft Windows NT Server, Microsoft LAN Manager,

Novell NetWare, сети с протоколами стека TCP/IP, IBM LAN Server, Banyan VINES, DEC PATHWORKS и Apple AppleTalk. Все сети поддерживаются с помощью их родных протоколов.

Конечные пользователи могут получить доступ к данным, хранящимся на сервере, и оперативно составлять отчеты и проводить анализ данных с помощью таких средств, как Microsoft Access и Microsoft Excel.

Для перехода от баз данных других форматов к SQL Server компания Microsoft разработала ряд полезных утилит миграции: Access Upsizing Tool для перехода от архитектуры приложений модели файл-сервер, Transfer Manager для переноса данных из баз данных Sybase или SQL Server, работающих в среде UNIX или OS/2, на платформу Windows NT.

Компания Microsoft предусмотрела также ряд средств для того, чтобы предлагаемое ей решение было открытым. Среди них технология ODBC, интерфейс DB-Library, шлюз ODS и язык ANSI SQL.

- *Технология ODBC (Open Database Connectivity)* - это открытый и независимый от производителя прикладной программный интерфейс (API) между клиентами и сервером. Технологию ODBC поддерживают свыше 130 независимых производителей приложений, драйверов и сервисов баз данных, среди которых IBM, Lotus Development Corporation, Novell и Word Perfect.
- *DB-Library* - это родной интерфейс для Microsoft SQL Server, который поддерживается большим количеством коммерческих утилит и приложений. SQL Server также поддерживает и интерфейс Open Client компании Sybase.
- *ODS (Open Data Services)* - это API для разработки шлюзов, работающих на базе сервера Windows NT, для предоставления клиентам доступа к любым источникам информации.
- *Transact-SQL* - язык, который разработан специально для Microsoft SQL Server. Эта реализация SQL полностью совместима со стандартом SQL 1989 и дополнена возможностями для создания таких компонент базы данных, как триггеры, правила, хранимые процедуры, и некоторых других.

Разработка приложений

Для быстрой разработки пользовательских приложений можно воспользоваться одним из настольных приложений компании Microsoft : настольной СУБД Microsoft Access, электронной таблицей Microsoft Excel, настольной СУБД Microsoft FoxPro, обладающей более высоким быстродействием по сравнению с Microsoft Access, системой программирования Microsoft Visual Basic, которая сочетает простоту, графические средства проектирования приложений и доступ к данным с помощью встроенных средств ODBC, или системой

программирования Microsoft Visual C++ с графической средой программирования.

Кроме этого, для реализации логики работы приложения можно воспользоваться средствами, которые работают в серверной части приложения, то есть средствами системы SQL Server: проекциями (views), хранимыми процедурами и триггерами.

Надежность системы SQL Server определяется надежностью операционной системы Windows NT, а также собственными средствами - механизмом транзакций, системой автоматического восстановления после сбоев и отказов, компонентами целостности данных (правилами, хранимыми процедурами и триггерами).

Производительность и масштабируемость обеспечиваются за счет полного использования широких возможностей в этих аспектах сервера Windows NT Server. SQL Server использует средства создания и диспетчирования нитей, управления их приоритетами, средства безопасности, управления событиями и их мониторинга Windows NT, исключая тем самым ненужное дублирование кода в операционной системе и СУБД. SQL Server не создает для каждого пользователя отдельного процесса, а работает как единый процесс, создающий для каждого пользовательского соединения отдельную нить. На SMP платформах каждая нить назначается на свободный или малозагруженный центральный процессор, обеспечивая динамическую балансировку загрузки.

Масштабируемость достигается за счет того, что сама операционная система Windows NT изначально разрабатывалась как переносимая система, способная работать на широком ряде аппаратных платформ - от простых однопроцессорных Intel-серверов до мощных многопроцессорных серверов на RISC-процессорах Alpha или MIPS.

Администрирование SQL Server обеспечивается за счет поставляемых утилит с графическим интерфейсом, предназначенных для работы под управлением Windows 3.1, Windows for Workgroups 3.11 или Windows NT. Эти утилиты поставляются в 32-х битном и 16-ти битном вариантах. С помощью утилит администрирования можно управлять несколькими SQL серверами в сети. SQL Server поддерживает опцию интегрированной безопасности, которая обеспечивает один логический вход как в сеть, так и в сервер баз данных. При этом доступ к SQL серверу управляется привилегиями, которые устанавливаются для пользователей и групп пользователей в Windows NT. Специальная компонента, называемая SQL Monitor, позволяет составить и отработать расписание автоматического копирования данных из базы на устройства резервного копирования, такие как стриммеры.

Перспективы развития компания Microsoft связывает с версией SQL Server 6.0. Эта версия предназначена для крупных распределенных корпоративных систем и отличается следующими особенностями:

- Распределенные данные и приложения будут поддерживаться за счет встроенной системы репликации с удобными графическими средствами управления.
- Трехуровневая архитектура администрирования, основанная на технологии OLE, будет обеспечивать централизованное администрирование распределенными серверами.
- Объектная ориентация на основе технологии OLE предназначена для превращения SQL сервера из пассивного внешнего источника данных в активного участника пользовательских приложений. SQL Server 6.0 будет поддерживать богатый интерфейс OLE Automation для связей с настольными приложениями с помощью языка Visual Basic for Applications и интерфейса MAPI. Например, используя OLE, он может отослать пользователям результаты запроса по почте как встроенные объекты электронной таблицы Microsoft Excel.
- Повышение производительности до 20 Гбайт в час за счет новой методики параллельного архивирования и технологии компрессирования данных.

Шлюз SNA Server

В традиционной среде мейнфреймов вся обработка данных осуществляется на хост-машинах, причем мейнфреймы IBM традиционно считаются надежными средствами централизованного управления и администрирования такого рода обработки. Однако терминалы для доступа к мейнфреймам обычно не имеют средств пользовательского графического интерфейса, что усложняет работу пользователей, а также не могут производить обработку данных, как это делают персональные компьютеры. Кроме того, мейнфреймам недостает средств, необходимых для быстрой разработки приложений.

С другой стороны, настольные приложения, такие как Microsoft Access и Microsoft Excel, обеспечивают простоту использования данных, но не обладают достаточной мощностью для поддержки ответственных корпоративных данных. Чтобы получить преимущества от использования всех корпоративных данных, необходимо объединить данные, используемые в настольных компьютерах, с данными, хранящимися на серверах сетей и на мейнфреймах. Для западных пользователей эта проблема особенно актуальна, так как там около 80% всех компьютерных данных хранятся на мейнфреймах и других хостах.

Одним из средств объединения локальных сетей с мейнфреймами является Microsoft SNA Server for Windows NT, который обеспечивает для пользователей локальных сетей доступ к мейнфреймам и мини-компьютерам фирмы IBM.

По сравнению с прямым подключением персональных компьютеров к мейнфрейму, использование шлюза SNA Server экономит производительность как мейнфрейма, так и персоналок, обеспечивает централизованное управление взаимодействием, защищает корпоративные данные на уровне безопасности C2,

обеспечивает высокую готовность доступа к мейнфрейму за счет средств отказоустойчивости и архивирования данных.

SNA Server обеспечивает:

- Соединение со всеми популярными мейнфреймами архитектуры SNA (например, 3090 или ES/9000) и SNA-мини-компьютеров (семейства AS/400).
- Использование всех типов SNA-каналов: SDLC, X.25, 802.2.
- Взаимодействие с серверами локальной сети, работающими под управлением операционных систем: Microsoft, NetWare, Banyan VINES, AppleTalk.
- Взаимодействие с клиентами сети, работающими под управлением: MS-DOS, Windows 3.x, Windows NT Workstation, Windows NT Server и Macintosh (UNIX через дополнительный шлюз TN3270).
- Взаимодействие с клиентами через мосты, маршрутизаторы или сервер удаленного доступа RAS для Windows NT.

Шлюз SNA Server может использоваться в различных конфигурациях - один шлюз для доступа к одному хосту, один шлюз для доступа к нескольким хостам, много шлюзов для доступа к одному хосту, много шлюзов для доступа к нескольким хостам.

Шлюз поддерживает до 250 одновременных соединений в любых комбинациях восходящих соединений (с хостом), равноправных или нисходящих соединений. Шлюз поддерживает до 2 000 пользователей, а в одном домене Windows NT может быть до 50 шлюзов SNA Server. Шлюз использует ту учетную пользовательскую информацию, которая хранится на сервере Windows NT, а для доступа к хостам пользователям шлюза должны быть предоставлены специальные права. Шлюз можно администрировать и с хоста с помощью системы NetView.

Почтовые системы Microsoft Mail и система коллективной работы Microsoft Exchange

Компания Microsoft предлагает в настоящее время три системы обработки сообщений - Microsoft Mail Server 3.2, Microsoft Mail Server 3.5 и Microsoft Exchange. Почтовая система Microsoft Mail 3.2 выпускается достаточно давно и состоит из почтового агента передачи сообщений (Message Transfer Agent, MTA), работающего на выделенном персональном компьютере под управлением MS-DOS или OS/2, и почтовых клиентов, которые могут работать в локальной сети под управлением DOS, Windows, OS/2 или Macintosh. Кроме этого, на любом невыделенном компьютере располагается база данных почтовой системы - почтовое отделение (Post Office, PO). В почтовую систему входят также шлюзы к другим типам почтовых систем, в том числе к системам, основанным на стандарте X.400, системам обмена сообщениями мейнфреймов PROFS и SNADS,

почтовой системе SMTP сетей TCP/IP, системе обмена сообщений MHS фирмы Novell и некоторым другим. Все эти шлюзы работают на выделенных компьютерах под управлением DOS.

Недавно компания Microsoft объявила о выпуске новой версии почтовой системы Microsoft Mail Server 3.5, включающей новую версию многозадачной программы-агента передачи сообщений (Multitasking Message Transfer Agent, MMTA), которая работает в среде ОС Windows NT Server. Таким образом, старый вариант MMTA переводится из OS/2 в родную среду Windows NT. Кроме этого, новый пакет включает новые административные утилиты, позволяющие усовершенствовать управление почтовыми ящиками пользователей, личными адресными книгами и глобальными списками адресов.

Интеграция Microsoft Mail с Windows NT Server является частью стратегии Microsoft по обеспечению возможности функционирования в рамках одной организации двух систем - Microsoft Mail и Microsoft Exchange Server. Exchange Server предназначен не только для поддержки почтового обмена сообщениями, но и содержит надстройки для организации работы в группе. Система Microsoft Mail Server 3.5 полностью совместима с клиентским ПО Microsoft Exchange, которое вошло в комплект поставки Windows 95. Для использования этого ПО совместно с почтовыми отделениями Microsoft Mail не требуется дополнительных лицензий.

Система Microsoft Exchange интегрирует электронную почту, планирование работы пользователей, электронные формы, средства разделение документов и содержит некоторые приложения, например, отслеживание активности покупателей. По сути, это объединение электронной почты и системы разделения информации на основе технологии клиент-сервер. Система состоит из сервера Microsoft Exchange Server и клиентов для различных популярных операционных систем. Система организует для своих клиентов доступ к различным источникам информации, например, к серверам баз данных, на основе механизма обмена сообщениями в технологии store-and-forward, присущей почтовым системам. Сервер поддерживает не только локальные, но и глобальные связи, а также обеспечивает шифрацию передаваемых сообщений по алгоритму LSA.

Система управления компьютерами System Management Server

Эта система предназначена для централизованного управления объединенными в сеть персональными компьютерами.

В функции System Management Server входит:

Учет используемых аппаратных и программных средств. Система автоматически собирает информацию об обследованных ПК и создает записи в базе данных об аппаратных и программных ресурсах. После этого администратор может быстро выяснить, чем он располагает и где это находится. Например,

узнать о том, на каких ПК нужно обновить драйверы принтеров, какие ПК обладают достаточным количеством памяти и дискового пространства и т. п.

Распределение и установка программного обеспечения. После завершения обследования администратор может создать пакеты рассылки программного обеспечения - очень эффективный способ для уменьшения стоимости такой процедуры. SMS также позволяет централизованно устанавливать и администрировать приложения, которые запускаются с файловых серверов, а также дает возможность конечным пользователям запускать такие приложения с любой рабочей станции сети.

Анализ сетевых протоколов. SMS позволяет анализировать сетевые протоколы в целях обнаружения узких мест в сети и управления сетевым трафиком.

Удаленный анализ производительности и возникающих проблем. База данных SMS хранит детальную информацию о конфигурации всех ПК в сети для того, чтобы можно было выполнять удаленный анализ возникающих проблем. Администратор может удаленно управлять мышью, клавиатурой и видеть экран любого ПК, работающего в сети под управлением MS-DOS или Windows.

Windows 95 и будущее семейства Windows

Windows 95 представляет собой замечательный пример эволюционного развития архитектуры Windows. Система имеет новый интерфейс, который можно легко настраивать и с помощью которого можно легко путешествовать по ресурсам системы. Эта ОС выполняет 16- и 32-х разрядные приложения, поддерживает технологию "plug-and-play" и содержит встроенные средства для сетевой работы. У многих обозревателей нет сомнений в том, что эта система улучшит жизнь многим миллионам пользователей персональных компьютеров.

Несмотря на все преимущества, Windows 95 - это по-прежнему вариация на тему Windows 3.1. Для большинства пользователей это означает, что заложенные в ней архитектурные анахронизмы могут приводить к неожиданному краху системы. Даже по мнению специалистов Microsoft, для важных бизнес-приложений более предпочтительным является использование Windows NT, которая обеспечивает защиту данных и устойчивость к некорректной работе приложений.

Windows 95 обладает некоторыми уникальными возможностями. Вероятно, она обеспечивает наилучшую среди всех операционных систем *поддержку мобильных пользователей*, спектр ее коммуникационных средств чрезвычайно широк и гибок, она наконец предоставляет решение застарелой проблемы недостатка памяти для выполнения приложений. Именно для мобильных пользователей, работающих на портативных компьютерах с шиной PCMCIA, наилучшим образом проявляет себя технология "plug-and-play", хотя для персональных компьютеров с шиной ISA могут возникать некоторые проблемы. Windows 95 автоматически отслеживает подключение портативного компьютера к рабочей станции сети и загружает или выгружает соответствующие драйверы.

Например, когда пользователь приезжает в офис и подключает свой ноутбук к своей же рабочей станции, то Windows 95 узнает, что она присоединена теперь к внешнему монитору и начинает работать с графикой более высокого разрешения.

Windows 95 также является отличным коммуникационным средством: она поддерживает практически все сетевые протоколы и адаптеры. Для коммуникаций, использующих коммутируемые телефонные линии, Windows 95 предлагает использовать средства Microsoft Exchange. Встроенный клиент Microsoft Exchange имеет подсистемы для пересылки факсов, сообщений электронной почты и сети Microsoft Network, которая занимается рассылкой программного обеспечения и технической поддержкой пользователей в режиме on-line.

Наконец, переделка средств управления оперативной памятью позволит пользователям загрузить больше приложений прежде, чем система выдаст сообщение о нехватке памяти. Например, в Windows 3.1 обычно можно загрузить от 3 до 5 приложений, а для Windows 95 эти границы составляют 6 - 12 приложений.

Стабильность Windows 95, развитые средства сетевой работы и новый удобный пользовательский интерфейс представляют собой безусловные преимущества по сравнению с DOS или Windows 3.1, но все же не дотягивают до того уровня, к которому привыкли пользователи таких ОС, как OpenVMS, UNIX или Windows NT.

Одной из проблем Windows 3.1 является способность приложения вызвать крах системы, вынудив сделать перезагрузку. В Windows 95 осталось много старого кода, с помощью которого осуществляется выполнение приложений. Например, такие критические компоненты операционной системы, как USER и GDI, которые соответственно обеспечивают управление окнами и предоставляют средства графического интерфейса, являются по-прежнему 16-разрядными и работают в том же адресном пространстве, что и 16-разрядные приложения. Поэтому 16-разрядное приложение, содержащее ошибки, может потенциально "подвесить" виртуальную машину, на которой работают подсистемы USER и GDI, или, что еще хуже, заставить USER или GDI неверно работать, что может привести к краху всей ОС. Даже 32-разрядные приложения могут вызвать останов системы. Большая часть нижней памяти размером в 1 Мбайт, принадлежащая адресному пространству системного кода Windows 95 (то есть системной виртуальной машине System VM), открыта для операций приложения Win32.

Многозадачность - это еще одно потенциально слабое место. Windows 95 пересылает все вызовы USER API через 16-разрядную системную виртуальную машину System VM, которая размещается там же , где и выполняемое 16-разрядное приложение. Если 16-разрядное приложение "подвешивает" машину System VM, отказываясь обрабатывать сообщение (встречающийся чаще всего тип ошибки в существующих приложениях Windows), то все остальные процессы

приостанавливаются. Пока пользователь не завершит в принудительном порядке зависшее 16-разрядное приложение (в Windows 95 есть хорошее средство для выполнения этой операции) и тем самым не освободит машину System VM, другие выполняемые программы, даже 32-разрядные, будут заблокированы.

Рисунки 8.4, 8.5 и 8.6 дают возможность сравнить архитектурные решения, положенные в основу операционных систем Windows 3.1, Windows 95 и Windows NT.

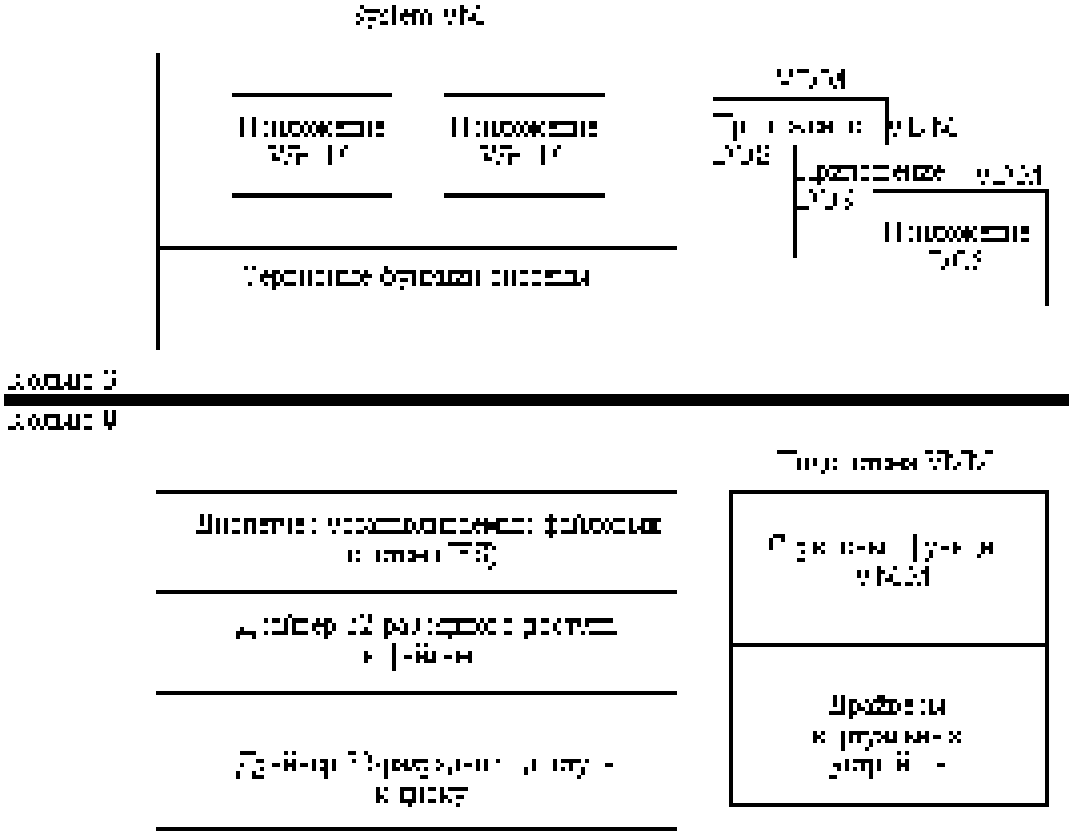


Рис. 8.4. Архитектура ОС Windows 3.1

В состав операционной системы Windows 3.1 (рис 8.4) входит системная виртуальная машина System VM, внутри которой размещаются все 16-разрядные приложения Win16, а также код и данные системных DLL, которые обеспечивают выполнение сервисных функций ОС. Приложения Win16 выполняются в общем адресном пространстве внутри системной виртуальной машины. Программы Win16 выполняются в режиме невытесняющей многозадачности. Системные библиотеки USER, GDI и KERNEL предоставляют сервисные функции операционной системы приложениям и отображаются в адресное пространство, совместно используемое приложениями Win16. Приложения DOS запускаются на отдельных виртуальных DOS-машинах (VDM), работающих в режиме вытесняющей многозадачности. Диспетчер устанавливаемых файловых систем (IFS) и драйвер 32-разрядного доступа к файлам (только в Windows for Workgroups 3.11) осуществляют большинство файловых операций в защищенном

режиме, что ускоряет доступ к файлам. Драйвер 32-разрядного доступа к диску управляет обменом с диском на физическом уровне.

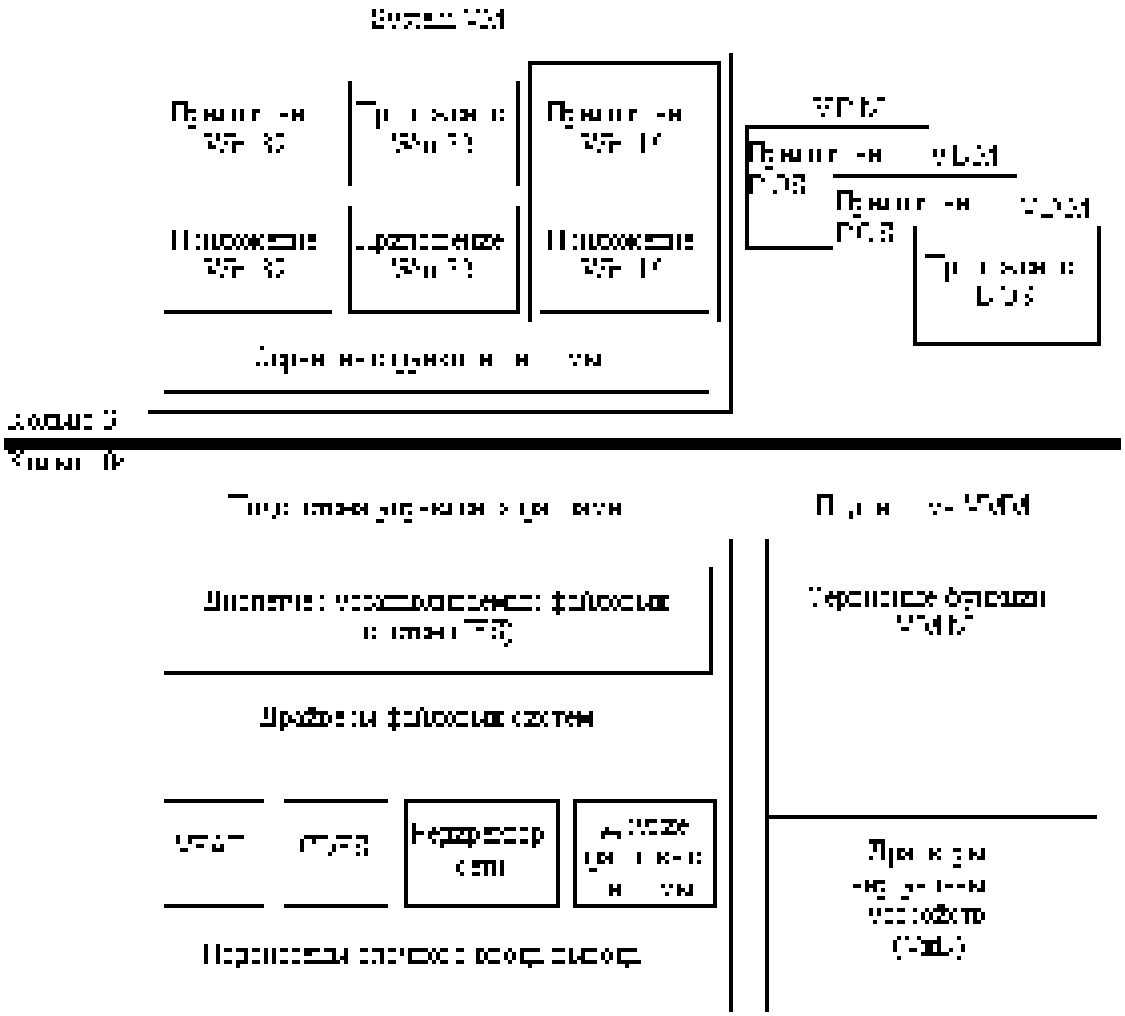


Рис. 8.5. Архитектура ОС Windows 95

Подсистема управления виртуальными машинами (VM Manager, VMM) предоставляет сервисные функции низкого уровня, такие как распределение процессорного времени между VM и управление виртуальной памятью. Сюда также относятся драйверы виртуальных устройств (VxD) для аппаратуры.

Архитектура Windows 95 (рисунок 8.5) представляет собой немного улучшенную версию архитектуры Windows 3.1. Внутри системной VM выполняются приложения Win16 и Win32. Большая часть кода операционной системы и данных также размещается здесь. Приложения Win32 работают на основе алгоритма вытесняющей многозадачности в отдельных адресных пространствах. Все приложения Win16 выполняются как единый процесс в общем адресном пространстве на основе алгоритма невытесняющей многозадачности. Библиотеки динамической компоновки USER, USER32, GDI, GDI32, KERNEL и KERNEL32, которые предоставляют системные сервисы всем приложениям, загружаются в системную VM и отображаются в адресные пространства каждого прикладного процесса. Это повышает производительность за счет устранения

затрат времени на переходы между кольцами защиты при вызове системных функций. Однако с другой стороны, это также ставит под угрозу целостность системы, открывая доступ к частям ОС для прикладных программ. На виртуальных DOS-машинах (VDM) выполняются DOS-программы. Они работают в режиме вытесняющей многозадачности.

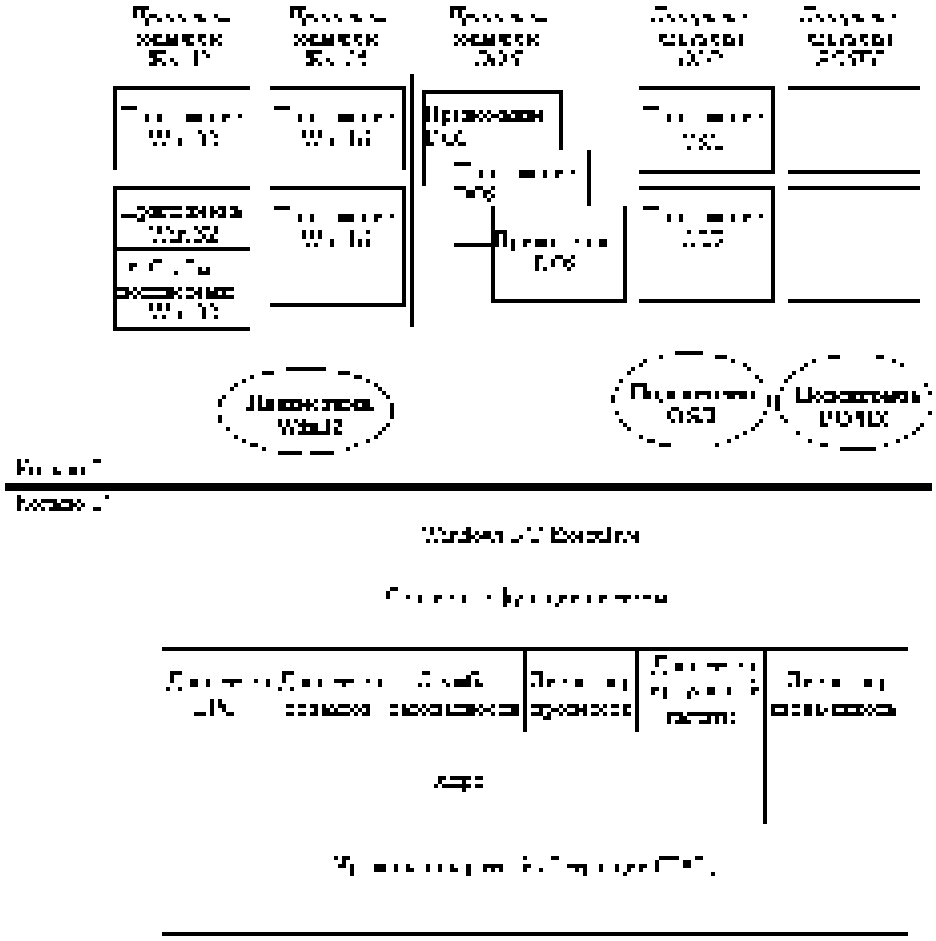


Рис. 8.6. Архитектура ОС Windows NT

Подсистема управления файлами Windows 95 работает в нулевом кольце защиты и обрабатывает все вызовы, связанные с вводом-выводом. Большинство вызовов обрабатывается в защищенном режиме, но некоторые по-прежнему приводят к переключению в режим Virtual 86, и обрабатываются в реальном режиме DOS. Диспетчер устанавливаемых файловых систем IFS передает вызовы файлового ввода-вывода драйверу соответствующей файловой системы. Драйвер файловой системы VFAT реализует собственную VFAT-систему Windows 95, которая похожа на файловую систему FAT с добавленными средствами обработки длинных имен файлов. Драйвер CDFS заменяет MSCDEX и управляет операциями по вводу данных с накопителей CD ROM. Редиректор, выполненный в виде драйвера файловой системы, обеспечивает обращение к сетевым накопителям. Можно устанавливать дополнительные драйверы файловых систем. Подсистема блочного ввода-вывода выполняет соответствующие операции на физическом уровне в ответ на запросы драйверов файловых систем.

Подсистема управления виртуальными машинами (VMM) предоставляет низкоуровневые сервисные функции, например, планирование нитей и управление памятью. Сюда также относятся драйверы виртуальных устройств (VxD) для аппаратуры.

На рисунке 8.6 представлена уже знакомая структура Windows NT, в которой каждое из приложений обращается к сервисным функциям (серверам) косвенно, через вызовы локальных процедур (LPC), реализованных в диспетчере LPC, являющемся частью NT Executive и работающем в привилегированном режиме. Приложения Win32 исполняются как отдельные многонитевые процессы. Программы Win16 могут запускаться как однонитевые процессы на общей виртуальной машине, или на собственной виртуальной машине, что обеспечивает им большую степень защищенности от других программ Win16. Приложения DOS выполняются как отдельные процессы на отдельных виртуальных DOS-машинах (VDM). Среда машины в рамках VDM конструируется таким образом, чтобы как можно более точно имитировать среду реального режима DOS. Подсистемы OS/2 и POSIX обеспечивают работу соответствующих прикладных программ в текстовом режиме.

Windows NT Executive предоставляет сервисные функции ОС, необходимые для подсистем пользовательского режима и реализует внутренние механизмы системы, такие, например, как планирование нитей и управление памятью. Слой системных сервисных функций служит интерфейсом между программами пользовательского режима и NT Executive.

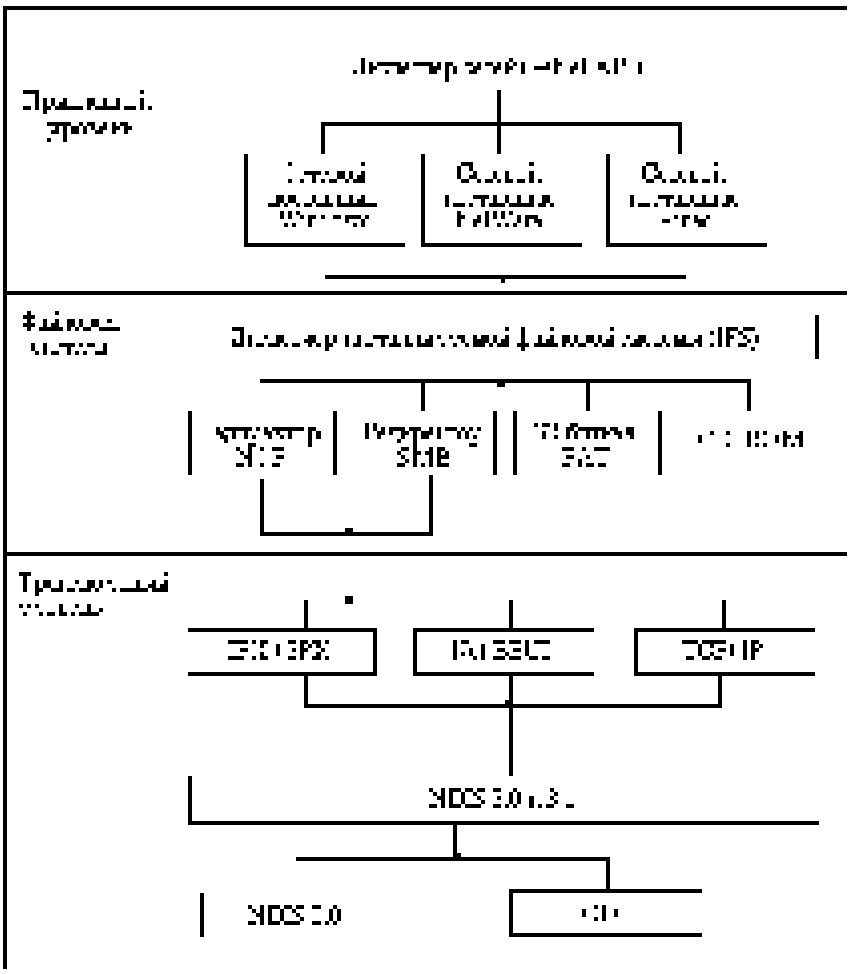


Рис. 8.7. Структура сетевых средств Windows 95

Ядро обрабатывает прерывания и исключительные ситуации, переключает нити, синхронизирует процессоры в многопроцессорных системах и выполняет другие низкоуровневые функции, используемые при работе NT Executive.

На рисунке 8.7 представлена организация сетевых средств Windows 95, очень напоминающая аналогичную структуру сетевых средств Windows NT. Это и не удивительно - обе системы поддерживают один и тот же программный интерфейс WNet, с помощью которого приложения и системные утилиты получают доступ к просмотру, отображению и потреблению сетевых ресурсов.

Как видно из следующей таблицы, разбиение версий Windows на два семейства - NT и 95 - явление временное. Оно вызвано скорее не стратегическими соображениями, а тактикой борьбы за пользователей в условиях, когда мощность большей части персональных компьютеров, установленных в настоящее время у пользователей, оказалась явно недостаточной для эффективной работы Windows NT. Ввиду угрозы перетекания конечных пользователей на более компактную и менее ресурсоемкую (по сравнению с Windows NT) OS/2 Warp Connect компания Microsoft и выпустила Windows 95, как некоторую временную ОС с ограниченным сроком годности - не более 5 лет, как это видно из таблицы. И хотя Microsoft планирует выпустить еще 2 версии, улучшающие свойства Windows 95, наличие некоторых общих свойств у Windows NT и Windows 95, а также

очевидные слабости Windows 95, говорят о том, что долговременная стратегия Microsoft связана с линией Windows NT, многие из новых свойств которой будут отрабатываться и в версиях линии Windows 95 (как сейчас это произошло с пользовательским интерфейсом и некоторыми системными утилитами типа клиента Microsoft Exchange).

Таблица. 8.1.

Существующие и будущие версии операционных систем компании Microsoft

	Корпоративные пользователи (работающие на достаточных аппаратных ресурсах)	**Домашние пользователи** (работающие на ограниченных аппаратных ресурсах)
1994	*Windows 3.1* *Windows for Workgroups 3.11*	*Windows 3.1*
1995	*Windows NT 3.51* Работа на платформе PowerPC Общие органы управления и панели диалога с Windows 95	*Windows 95* Новый пользовательский интерфейс Вытесняющая многозадачность Многонитевость Защита памяти Поддержка службы Microsoft Network Win32 API
1996	*Windows NT 3.52* Поддержка службы Microsoft Network Пользовательский интерфейс Windows 95	*Nashville* Незначительно улучшенная версия Windows 95
1997-1998	*Windows NT 4.0* Объектно-ориентированная архитектура Объектно-ориентированная файловая система Сетевые средства OLE Служба каталогов Встроенные сетевые средства ATM Аутентификация с помощью службы Kerberos	*Memphis* Последняя основная версия обычной Windows
1999-2000	Windows NT Единая, унифицированная ОС для всех пользователей	

Операционная система OS/2

История развития OS/2 и ее место на рынке

Аналитики, занимающиеся 32-х битными операционными системами для персональных компьютеров, всегда концентрируют свое внимание на битве между Microsoft Windows и IBM OS/2, предполагая, что Microsoft имеет преимущество. Но не все согласны с такой точкой зрения. OS/2 v.2.0 была первой доступной и работающей 32-х битной операционной системой для персональных компьютеров. И она первой начала очередной круг состязаний - версия OS/2

Warp, предназначенная для клиентских машин сетей клиент-сервер и одноранговых сетей, появилась на рынке раньше Windows 95, позиционированной аналогичным образом. OS/2 Warp была также первой системой, включившей набор средств поддержки Internet, а также средств объектной ориентации.

Битва Microsoft - IBM на рынке настольных ОС

Когда бета-тестеры получили Chicago, первую публичную версию Windows 95, те, кто уже использовал OS/2, отметили чрезвычайную схожесть двух систем. Например, обе начинают работу с показа красивой заставки, а затем приглашают пользователя к работе за вместительным рабочим столом; обе системы рассматривают иконки и программы как объекты; обе используют правую кнопку мыши для управления поведением объектов; обе используют более 20 дискет для инсталляции. Пользовательский интерфейс обеих систем имеет одинаковый уровень изощренности, требования к аппаратным ресурсам компьютера похожи, и они обе основаны на использовании одинакового набора лежащих в основе системы технологий. Эти технологии включают многозадачность и многонитевость, способность выполнять DOS-программы с помощью виртуальных машин процессоров Intel 80x86, полную 32-х битную организацию.

И это не случайность. С тех пор, как IBM выпустила версию 2.0 OS/2, а Microsoft решила позиционировать Windows NT как корпоративную ОС, стала ясно видна важная брешь в линии операционных систем Microsoft, которую и заполнила IBM. Попытки Microsoft выдвинуть Windows 3.1 на ту же роль наиболее развитой ОС для настольных систем, что и OS/2, имели ограниченный успех. Аналитики считают, что корпорация Microsoft действительно хотела, чтобы Windows NT заняла на рынке то же место, что и OS/2, но OS/2 уже заняла его к тому времени, когда вышла Windows NT.

В результате Microsoft стала нести потери в объемах продаж, и, что более важно, терять твердую почву для своих операционных систем. Когда стало ясно, что Windows NT вряд ли в полной мере станет лидером настольных ОС высшего класса, маркетинговая машина Microsoft стала меньше говорить о возможностях Windows NT и начала говорить о возможностях Windows 95. Ясно, что IBM и OS/2 оказали значительное влияние на стратегию Microsoft в области операционных систем.

IBM, в свою очередь, постоянно создает здоровую конкуренцию для линии Windows. Windows 95 не сравнима с OS/2 2.2. Скорее конкурировать будут Windows 95 и OS/2 Warp 3/0. Warp - это выстрел с дальним прицелом, направленный на вытеснение Windows. И, хотя Warp имеет некоторые исходные преимущества и как система выглядит "лучше", Windows по прежнему является надежным выбором.

Имена операционных систем могут измениться, но равновесие в битве IBM/Microsoft останется тем же. Через два года Microsoft и IBM смогут

обмениваться аналогичными выстрелами в сражении Cairo - OS/2 вместо Windows 95 - Warp.

Существуют две причины - фактическая и эмоциональная - которые мешают установлению перемирия между этими двумя компаниями:

- *Фактически*, IBM была в этой области первой. OS/2 превратилась в работающий продукт со свей версией 2.0 в 1992 году. С этого времени она стала многозадачной, многонитевой системой с удобным объектно-ориентированным интерфейсом. Усилия по развитию OS/2 были неторопливыми и постоянными, и система получала похвалы и поддержку на всем пути своего развития. Однако Windows по прежнему держала наибольшую долю рынка. Преимущества OS/2 были не всесторонними, и, несмотря на усилия технических и маркетинговых специалистов IBM, система не стала вполне совершенной.

В отношении управления системой, с OS/2 работать не проще, чем с Windows. Конфликты с аппаратной и программной совместимостью могут по прежнему вызывать проблемы, и их решение не выглядит универсальным и интуитивным.

- *Эмоционально*, IBM чувствует себя "преданной" Microsoft, которая сбежала из рядов разработчиков OS/2. Это не совсем справедливо по отношению к Microsoft, так как компания вправе вкладывать свои капиталы с ту сферу деятельности, которая по ее мнению принесет наибольшую прибыль. Хотя Microsoft могла бы вести себя более тактично и продолжать партнерство по OS/2.

Хотя сейчас IBM далеко не та компания, какою она была в те далекие дни, когда она доминировала на рынке персональных компьютеров, ей тоже не хватает такта. Эта компания была первой так долго, что она не умеет выступать на вторых ролях. Первоначальная стратегия игнорирования общественных потребностей и навязывания дорогих, но не всегда обоснованных решений, быстро потерпела неудачу. С появлением клонов персональных компьютеров отпала необходимость платить больше только за марку IBM. Поэтому с момента появления версии OS/2 2.0 IBM изменила свою стратегию. Она стала играть по тем же правилам, по которым играют остальные компании.

OS/2 - постепенные улучшения

Операционная система OS/2 начиналась как совместная разработка IBM и Microsoft (хотя большую часть работы должна была выполнить Microsoft). Изначально она была задумана как замена DOS. Уже тогда было ясно, что DOS с ее ограничениями по памяти и по возможностям файловой системы не может воспользоваться вычислительной мощностью появляющихся компьютеров. OS/2 была хорошо продуманной системой. Она должна была поддерживать вытесняющую многозадачность, виртуальную память, графический

пользовательский интерфейс, виртуальную машину для выполнения DOS-приложений. Фактически она выходила за пределы простой многозадачности с ее концепцией, названной многонитевостью.

Первые версии OS/2 не оказали значительного влияния на рынок. Версия OS/2 1.0, выпущенная в 1987 году, содержала большинство технических свойств, необходимых для многозадачной ОС. Однако у нее не было менеджера графического представления (presentation manager, PM), а также отсутствовали драйверы для многих популярных принтеров и других устройств. Версия OS/2 1.1, появившаяся в 1989 году, включала рудиментарную версию PM, которая, наконец, делала возможным использование графических приложений в нескольких окнах. Однако в этой версии PM не хватало многих свойств, которые присущи развитому графическому интерфейсу, кроме того, по прежнему отсутствовали многие драйверы принтеров. Выпущенная в 1990 году версия 1.2 имела улучшенный PM, хотя он и не следовал общепринятым концепциям графического интерфейса. Появились драйверы для большинства принтеров и других периферийных устройств.

Однако дискредитация OS/2 уже произошла. Версия 1.2 не была существенно лучше предыдущих версий и все еще предъявляла значительные требования к аппаратуре. К этому времени многие пользователи решили перейти на новую платформу Windows 3.0 или подождать, пока не появится что-нибудь принципиально лучшее. Продажи OS/2 по-прежнему были вялыми и рынок не интересовался ею. Это объяснялось наличием у OS/2 ряда существенных недостатков:

- Виртуальная машина DOS, которая должна была бы обладать способностью выполнять немодифицированные приложения DOS, с самого начала имела технические изъяны. Эта виртуальная машина была разработана на базе виртуальных возможностей процессора i286, который позволял выделять сегмент памяти в 640 Кб для отдельного DOS-приложения. Однако процессор i286 в этом виртуальнои режиме работал слишком медленно, поэтому виртуальная DOS-машина была реализована на основе реального режима процессора. При этом требовался перезапуск процессора для переключения между реальным и защищенным режимами. Хотя эта операция и выполнялась очень быстро и незаметно для пользователя, она была сложной и вносила путаницу.
- Microsoft и IBM не смогли в полной мере реализовать концепцию виртуальной обработки в режиме I8086: в этом режиме DOS-приложения, которые непосредственно читали или писали в аппаратные порты, переставали работать. В связи с этим не могли использоваться и популярные сетевые операционные системы на базе DOS.
- Память в этом режиме использовалась нерационально - если пользователь конфигурировал OS/2 с возможностью DOS-совместимости, то 640 КБ

памяти всегда выделялись для этих целей и не могли использоваться для задач OS/2.
- Еще одним недостатком было отсутствие возможности обмена данными между DOS- и OS/2-приложениями.
- В каждый момент времени могло выполняться только одно DOS-приложение, и это приложение не могло использовать расширенную память.

В результате для пользователей OS/2 многие популярные DOS-приложения оказались недоступными, а те, что были доступны, не могли вообще взаимодействовать со средой OS/2. Время показало, что для пользователей это обстоятельство оказалось весьма важным, так как многие отказались от покупки OS/2, оставаясь с проверенной, хотя и не очень совершенной DOS.

OS/2 Warp

Общая характеристика

В конце 1994 года IBM выпустила третью главную версию OS/2, которую назвала OS/2 Warp 3 (warp - основа). Его демонстрации и развернутая рекламная компания напоминали рекламную компанию 1992 года, когда была выпущена OS/2 2.0. Во всяком случае один лозунг был точным повторением: в этой системе есть много преимуществ, которые пользователи и корпорации могут извлечь немедленно из 32-х разрядной операционной среды.

OS/2 Warp имеет хорошо продуманный объектно-ориентированный интерфейс с применением техники drug-and-drop при выполнении операций копирования, удаления, печати, а также некоторых других. Перечни свойств объектов легко доступны в меню, вызываемых щелчком правой клавиши мыши. Имеется специальная панель для размещения часто используемых документов или прикладных программ.

В состав OS/2 Warp входит набор утилит BonusPack, который содержит IBM Works - интегрированный программный пакет начального уровня, и Internet Access Kit - самый полный набор средств для сети Internet из всех средств, поставляемых в составе операционных систем, Web Browser и почта Internet Mail. В публикациях встречаются утверждения, что он более совершенен, чем набор для доступа к Internet, реализованный в Windows 95. В феврале 1995 года IBM начала продавать пакет OS/2 Warp 3 Full Pack, который содержит библиотеки Win-OS/2. Эти библиотеки дают возможность выполнять Windows-программы, не приобретая лицензионных копий Microsoft Windows.

Одним из часто критикуемых недостатков OS/2 Warp является то, что она не поддерживает 32-х битные приложения Windows (точнее, она поддерживает API Win32s, но не поддерживает полный API Windows NT, который называется Win32 и который почти полностью поддерживает Windows 95). Однако в

ближайшее время этот недостаток не будет критическим, так как приложений Win32 пока немного, зато с приложениями Win16 у OS/2 Warp проблем нет. IBM говорит, что она может обеспечить поддержку приложений Win32, если этого пожелают пользователи.

В то же время в OS/2 Warp ощущается недостаток сетевых функциональных возможностей. Положение должно измениться, так как летом 1995 года IBM начала продавать следующую версию OS/2 - Warp Connect, которая содержит важнейшие драйверы и утилиты. В число новых средств входят редиректоры для операционных систем NetWare 3.x и 4.1 и OS/2 LAN Server. Версия OS/2 Warp Connect работает с протоколами IPX и NetBIOS, а также с новой реализацией протоколов TCP/IP. Этот новый комплект устанавливает двухточечное соединение по протоколу PPP вместо соединений SLIP, предусмотренных в базовом пакете OS/2 Warp. Этот комплект понизит нагрузку на центральный процессор и обеспечит одновременный доступ к локальной сети и сети Internet.

Кроме того, Warp Connect предоставляет давно ожидаемые в OS/2 средства одноранговой сетевой связи. Согласно сообщению фирмы IBM, в эту версию входит большое число собственных драйвером, которые смогут работать более чем с 70% существующих адаптеров Ethernet и более чем с 90% адаптеров Token Ring. То же самое программное обеспечение дает возможность клиенту Warp Connect подключаться в серверу LAN Server 4.0.

Warp Connect содержит также программу Lan Distance фирмы IBM, которая позволит соединяться через связной сервер с любым подключенным к сети устройством. В отличие от Windows 95 ОС Warp Connect не содержит средств, поддерживающих удаленный доступ через коммутируемые телефонные сети. Еще одним нововведением является справочная база данных ASK PSP на компакт-диске с интерфейсом запросов на языке, близком к естественному английскому.

Что касается почтовых услуг, то IBM выбрала для Warp Connect пакет Lotus Notes Express, а не свой собственный Ultimedia Mail/2. Notes Express позволяет соединиться с любым сервером Notes.

Как и другие версии Warp, Warp Connect тоже будет поставляться в двух версиях: одна без Windows-библиотек, другая, подобно Full Pack, с библиотеками Win-OS/2.

Внутренняя организация OS/2 Warp

На рисунке 9.1 показана структура операционной системы OS/2 Warp 3.0. В OS/2 имеется несколько видов виртуальных машин для прикладных программ. Собственные 32- и 16-разрядные программы OS/2 выполняются на отдельных виртуальных машинах в режиме вытесняющей многозадачности и могут общаться между собой с помощью средств DDE OS/2. Прикладные программы DOS и Win16 могут запускаться на отдельных виртуальных машинах в

многозадачном режиме. При этом они поддерживают полноценные связи DDE и OLE 2.0 друг с другом и связи DDE с 32-х разрядными программами OS/2. Кроме того, можно запустить несколько программ Win16 на общей виртуальной машине Win16, где они работают в режиме невытесняющей многозадачности. Разнообразные сервисные функции API OS/2, в том числе SOM (модель системных объектов), обеспечиваются с помощью системных динамических библиотек DLL, к которым можно обращаться без требующих затрат времени переходов между кольцами защиты. Ядро OS/2 предоставляет многие базовые сервисные функции API, обеспечивает поддержку файловой системы, управление памятью, и имеет диспетчер аппаратных прерываний. В ядре виртуальных DOS-машин (VDM-ядре) осуществляется эмуляция DOS и процессора 8086, а также управление VDM. Драйверы виртуальных устройств обеспечивают уровень аппаратной абстракции. Драйверы физических устройств напрямую взаимодействуют с аппаратурой.

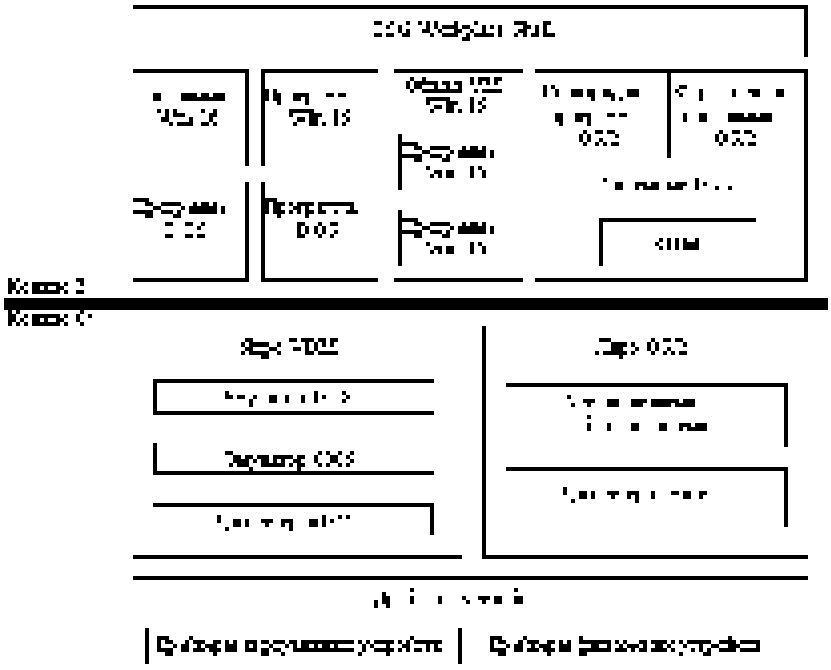


Рис. 9.1. Структура OS/2

На рисунке 9.2 изображены сетевые средства OS/2 Warp Connect. Они делятся на четыре уровня. Прикладной уровень включает программные интерфейсы приложений операционной системы. Компоненты на уровне файловой системы отвечают за выполнение файловых операций. Транспортный уровень реализует коммуникационные протоколы. Имеется компонента *Общая транспортная семантика*(Common Transport Semantic), которая позволяет использовать любую файловую систему (а точнее ее редиректор) в сочетании с любым протоколом транспортного уровня.

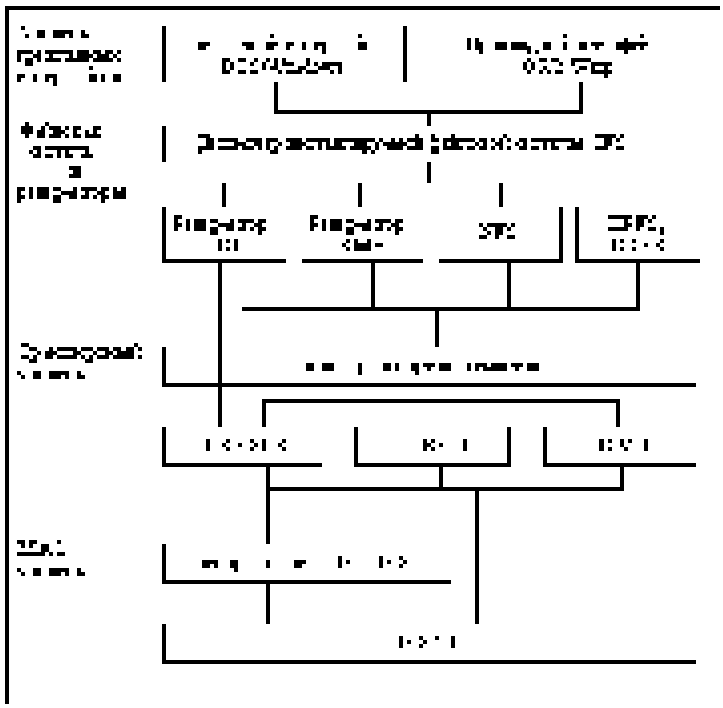


Рис. 9.2. Структура сетевых средств OS/2 Warp Connect

Программное обеспечение MAC-уровня включает драйверы сетевых адаптеров и диспетчерский слой в стандарте NDIS 2.01, который позволяет различным сетевым протоколам работать через один адаптерам, и различным адаптерам связываться через общий протокол. Существует модуль преобразования ODI-NDIS, который позволяет использовать модули транспортных протоколов, реализованные в расчете на работу с диспетчерским слоем ODI компании Novell.

Диспетчер инсталлируемой файловой системы (IFS) теоретически позволяет любой прикладной программе работать с любой файловой системой.

Файловая система HPFS

HPFS - сокращенное название высокопроизводительной файловой системы (high performance file system), совместно разработанной в 1989 году корпорациями IBM и Microsoft.

Эта система была разработана, чтобы преодолеть некоторые недостатки FAT, к числу которых относятся:

- ограничения, налагаемые на размер файлов и дискового пространства;
- ограничение длины имени файла;
- фрагментация файлов, приводящая к снижению быстродействия системы и износу оборудования;
- непроизводительные затраты памяти, вызванные большими размерами кластеров;
- подверженность потерям данных.

Проблема непроизводительных потерь дискового пространства связана с тем, что место на диске выделяется целыми блоками - кластерами. *Кластер* - это единица дискового пространства, которыми оперирует файловая система при выделении места для файла. В среднем половина выделяемого кластера для каждого файла будет затрачиваться в пустую. Это может быть одной из причин нерационального использования памяти диска. Например, при емкости диска 510 Мбайт число размещенных на нем файлов может составить около 1,5 тысяч. В этом случае FAT приведет к потере 6 Мбайт пространства, обусловленной только размером выделяемого блока. Для очень распространенных сейчас дисков емкостью 850 Мбайт ситуация может оказаться еще более критической. На таком диске может разместиться около 2 тысяч файлов, что повлечет за собой потерю 20 Мбайт. Для сетевых дисков емкостью в несколько гигабайт потери достигают астрономических цифр. Чем больше размер раздела жесткого диска, тем больше объем минимальной неделимой области памяти, выделяемой файлу, тем больше потери.

Эти потери можно существенно сократить внедрением более эффективных файловых систем. Простой переход на HPFS, работающую в среде OS/2, позволяет вновь вернуться к первоначальному размеру выделяемого блока - 512 байт, причем для любых размеров диска. Размер вероятного выигрыша для диска емкостью 512 Мбайт, содержащего 8 000 файлов, составит около 30 Мбайт. Этот выигрыш связан с тем, что на каждом файле в среднем теряется не 4096 байт (половина размера кластера в FAT для диска данной емкости), а всего 256 байт.

В OS/2 положение осложняется применяемым методом хранения расширенных атрибутов (extended attributes). В разделе FAT файл, содержащий единственный символ, занял бы целый кластер для размещения собственно файла и еще один кластер для расширенных атрибутов.

Так как расширенные атрибуты почти всегда имеют объем меньше 300 байт, размер теряемого впустую дискового пространства изменяется от примерно половины кластера при использовании малых разделов до львиной доли объема кластер при больших разделах. В сумме на каждом файле теряется примерно кластер.

Переход на HPFS позволит сэкономить дисковое пространство. HPFS распределяет пространство, основываясь на физических 512-байтовых секторах, а не на кластерах, независимо от размера раздела. Система HPFS позволяет уменьшить и непроизводительные потери, так как в ней предусмотрено хранение до 300 байт расширенных атрибутов в F-узле файла, без захвата для этого дополнительного сектора.

Другая проблема связана с *фрагментацией файлов*, которая наиболее характерна для емких дисков с большим числом файлов. Фрагментация существенно сказывается на времени доступа к файлу. Другой негативный эффект фрагментации - повышенный износ диска. О серьезности этой проблемы говорит обилие утилит для дефрагментации дисков, использующих FAT.

Файловая система HPFS обеспечивает гораздо более низкий уровень фрагментации. Хотя избавиться полностью от нее не удается, снижение производительности, возникающее по этой причине, почти незаметно для пользователя.

Первые 16 секторов раздела HPFS составляют загрузочный блок. Эта область содержит метку диска и код начальной загрузки системы. Сектор 16, известный под названием суперблок, содержит много общей информации о файловой системе в целом: размер раздела, указатель на корневой каталог, счетчик элементов каталогов, номер версии HPFS, дату последней проверки и исправления раздела при помощи команды CHKDSK, а также дату последнего выполнения процедуры дефрагментации раздела. Он также содержит указатели на список испорченных блоков на диске, таблицу дефектных секторов и список доступных секторов.

Сектор 17 носит название SpareBlock (запасной блок). Он содержит указатель на список секторов, которые можно использовать для "горячего" исправления ошибок, счетчик доступных секторов для "горячего" исправления ошибок, указатель на резерв свободных блоков, применяемых для управления деревьями каталогов, и информацию о языковых наборах символов. Система HPFS использует информацию о языковых наборах, чтобы дать возможность пересылать файлы, составленные на разных языках, даже в том случае, когда имена файлов содержат уникальные для какого-либо языка символы. SpareBlock также содержит так называемый "грязный" флаг. Этот новый флаг сообщает операционной системе о том, было ли завершение предыдущего сеанса работы нормальным, либо произошло в результате сбоя электропитания, либо файлы не были закрыты должным образом по какой-то другой причине. Если этот флаг обнаружен во время начальной загрузки, то операционная система автоматически запускает утилиту CHKDSK, пытаясь обнаружить и исправить все ошибки, внесенные в файловую систему из-за неправильного выключения системы.

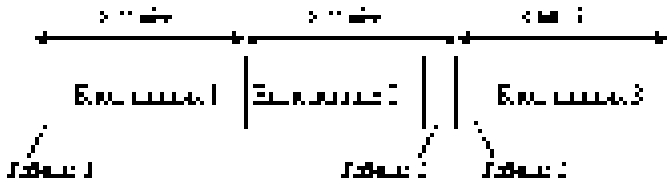


Рис. 9.3. Прием увеличения доступного непрерывного пространства

Во время форматирования раздела HPFS делит его на полосы по 8 Мбайт каждая. Каждая полоса - ее можно представить себе как виртуальный "мини-диск" - имеет отдельную таблицу объемом 2 Кбайт, в которой указывается , какие секторы полосы доступны, а какие заняты. Чтобы максимально увеличить протяженность непрерывного пространства для размещения файлов, таблицы попеременно располагаются в начале и в конце полос (рисунок 9.3). Этот метод позволяет файлам размером до 16 Мбайт (минус 4 Кбайта, отводимые для размещения таблицы) храниться в одной непрерывной области.

Затем файловая система HPFS оценивает размер каталога и резервирует необходимое пространство в полосе, расположенной ближе всего к середине диска. Сразу же после форматирования объем диска в HPFS кажется меньше, чем в FAT, так как заранее резервируется место для каталогов в центре диска. Место резервируется в середине диска для того, чтобы физические головки, считывающие данные, никогда не проходили более половины ширины диска.

Тот факт, что все пространство заранее распределено, также позволяет HPFS использовать специально оптимизированное программное обеспечение для более быстрой и эффективной работы с каталогами. Сравните это с системой FAT, где головкам требуется пройти весь путь к началу диска и прочитать таблицу размещения файлов, затем найти кластер, вновь пройти к началу диска, чтобы определить в FAT местонахождения следующего кластера, и так далее. Эта процедура становится еще более неудобной по мере нарастания фрагментации. Поэтому ясно, что размещение каталогов в середине диска повышает производительность системы. Вместе с тем, такое предварительное распределение не накладывает ограничений на число файлов, которые могут быть размещены на жестком диске. В редких случаях, когда системе HPFS потребуется больше пространства, чем изначально было отведено под каталоги, она может выделить дополнительное пространство из любой доступной области диска.

Число файлов в каждом блоке каталога - переменная величина, зависящая от длины имен файлов, которые содержатся в нем. Имена файлов в HPFS могут иметь длину до 254 символов, они сортируются в порядке, определяемом последовательностью символов в текущей кодовой странице системы.

Скорость работы увеличивается также благодаря способу хранения элементов каталогов. Система FAT последовательно просматривает каждый элемент каталога, чтобы отыскать нужный файл. Поэтому в самом худшем случае приходится перебирать все файлы в каталоге, прежде, чем найдется нужный. Но HPFS использует для хранения элементов каталогов структуру данных, называемую B-деревом. Каждый элемент каталога начинается с числа, представляющего длину элемента, которая изменяется в зависимости от длины имени файла. Затем следуют время и дата создания файла, его размер и атрибуты (только для чтения, архивный, скрытый и системный), а также указатель на F-узел файла. Каждый файл (и каталог) имеет F-узел - структуру данных, занимающую один сектор и содержащую принципиально важную информацию о файле.

F-узел содержит указатель на начало файла, первые 15 символов имени файла, дополнительные временные маркеры последней записи и последнего доступа, журнал, хранящий информацию о предыдущих обращениях к файлу, структуру распределения, описывающую размещение файла на диске, и первые 300 байт расширенных атрибутов файла. (Расширенные атрибуты редко занимают более 300 байт, что фактически означает, что HPFS для получения этой информации приходится читать на один сектор меньше, чем FAT.) Программы LAN Server и

LAN Manager фирмы IBM также сохраняют в F-узле информацию об управлении пользовательским доступом (Access Control). Заметьте, что F-узлы хранятся в смежных с представляемыми ими файлами секторах, поэтому, когда файл открывается, то четыре автоматически считываемых в кэш сектора содержат F-узел и три первых сектора файла.

Структура размещения HPFS имеет дополнительные преимущества по сравнению с FAT благодаря техническому приему, называемому кодированием по длине выполнения (Run Length Encoding, RLE). Вместо того, чтобы определять в таблице каждый используемый сектор, HPFS сохраняет указатель на первый сектор и число последовательно расположенных используемых секторов. Каждая область дискового пространства, описываемая парой (сектор, длина), называется экстентом. Хотя HPFS и сводит фрагментацию к минимуму, файлы все же могут быть в некоторой степени фрагментированными. В таких ситуациях пары, описывающие экстенты, добавляются к F-узлу файла. Один F-узел может хранить до 8 экстентов, обеспечивая достаточное пространство для большинства файлов.

А если все же потребуется еще большее пространство, то HPFS изменяет структуру таким образом, что F-узел становится корнем B+-дерева секторов размещения. B+-дерево является вариантом бинарного B-дерева. Созданное как структура для более быстрого обнаружения данных по сравнению с методом последовательного перебора, бинарное дерево состоит из ветвей, каждая из которых представляет выбор одного из двух возможных продолжений. Короткое дерево территориальных телефонных кодов может выглядеть так, как показано на рисунке 9.4,а. Здесь левая ветвь соответствует числам с меньшими значениями, чем значение в точке разветвления, а правая - с большими. Пусть выполняется поиск, например, кода 513. Вначале анализируется код в вершине дерева, поскольку 513 больше 212, то дальнейший поиск осуществляется по правой ветви. Так как 513 больше 407, то вновь поиск идет по правой ветви, где и находится нужный элемент данных. Для того, чтобы найти данные с помощью этого метода, потребовалось выполнить только два сравнения, в то время как для последовательного перебора могло бы потребоваться пять сравнений.

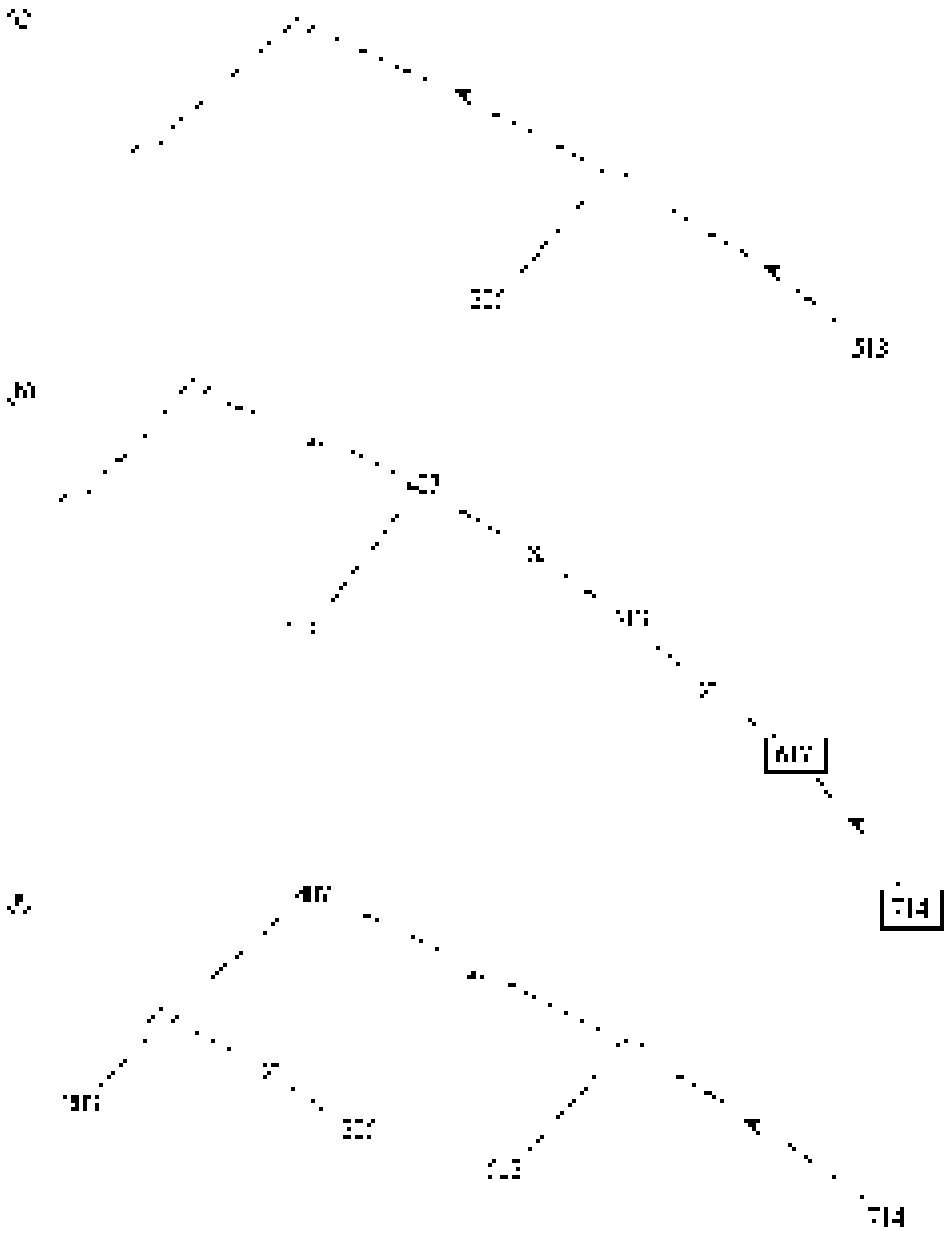


Рис. 9.4. Бинарные древовидные структуры

Эффективность бинарных деревьев зависит от последовательности, в которой в них добавляются новые элементы данных. Если, например, добавить код 617, то он будет следовать за кодом 513, а если добавить еще один код 714, то он последует за кодом 617. Поэтому, если элементы добавляются в порядке возрастания, то результирующее дерево становится все более похожим на последовательную структуру (рис. 9.4,б).

Структура B-дерева была разработана в целях предотвращения этой проблемы. Методы управления B-деревьями обеспечивают сбалансированность дерева. Структуру на рисунке 9.4 (б) лучше реорганизовать так, чтобы она приняла вид, показанный на рисунке 9.4 (в). Это делает дерево более эффективным, но приводит к дополнительным затратам, так как его балансировка выполняется всякий раз при добавлении или удалении элемента, либо при изменении значения элемента.

Возвращаясь к методу описания физической структуры файла, основанному на экстентах, следует учесть, что многие современные контроллеры дисков могут читать за одно обращение сразу несколько секторов. Применяемая в HPFS схема значительно повышает шансы использовать эту возможность, при этом происходит еще большее уменьшение числа требуемых операций взаимодействия между программой, файловой системой, драйвером дискового устройства и физическим диском.

HPFS имеет и другие оптимизирующие функции. Так при открытии или создании файла интеллектуальный алгоритм выделяет наиболее подходящую полосу. Программный интерфейс, используемый для создания файла, позволяет программисту сообщить операционной системе предполагаемый размер файла. С помощью этой информации HPFS может заранее выбрать для размещения файла полосу, имеющую непрерывную область наибольшего размера. Именно поэтому HPFS наиболее эффективно работает в больших разделах - больше число полос предоставляет большие возможности выбора.

Предположим, что многонитевая операционная система одновременно создает четыре новых файла на диске, использующем FAT. Так как для каждого файла нужен новый сектор, то он занимает ближайший доступный сектор в таблице размещения файлов. Это приводит к значительной фрагментации, так как кластеры между файлами распределяются вперемежку. HPFS выделила бы каждому из четырех файлов отдельную полосу, чтобы их содержимое оставалось непрерывным.

Как уже упоминалось, при открытии файла F-узел и первые три сектора считываются и помещаются в кэш. Если открываемый файл - исполняемый или если по данным журнала доступа к файлам в F-узле видно, что файл после открытия часто читается целиком, то многие секторы будут предварительно автоматически прочитаны и помещены в кэш.

Операции записи в кэш осуществляются особым образом, который называется "ленивой" записью. Когда программа посылает команду записи, HPFS помещает данные в кэш и немедленно сообщает программе, что операция выполнена, и только потом в фоновом режиме данные перемещаются из оперативной памяти на устройство. Это исключает длительную задержку, сопровождающую действительную операцию записи данных на устройство ввода-вывода. Однако при этом существует риск нарушения целостности данных. Например, уже после того, как программа получила от ОС сообщение об успешном завершении операции ввода-вывода, при попытке записать данные из кэша на диск драйвер этого устройства может сообщить об ошибке обращения к диску. В таком случае весьма полезным является список блоков "горячего" исправления.

Если попытка записи на диск заканчивается неудачно, то HPFS отыскивает в SpareBlock блок, который можно использовать для "горячего" исправления. Данные записываются в область "горячего" исправления, а таблицы неисправных блоков обновляются, указывая испорченный сектор и блок. HPFS будет

автоматически перенаправлять запросы чтения по новому адресу. Во время очередного выполнения утилиты CHKDSK файл будет скопирован в новое место, где он может храниться в непрерывной области. При обращении к нему нет необходимости переходить к блоку "горячего" исправления и обратно. Блок будет освобожден для использования в случае возникновения другой подобной проблемы. Таким образом, проблема решается автоматически без участи пользователя.

Для повышения эффективности система HPFS также предоставляет многоуровневые кэши. Например, она сохраняет в кэше подкаталоги, а также полное составное имя, записав в памяти контрольную сумму, однозначно определяющую путь к файлу. Поэтому при обращении к файлу, расположенному в глубоко вложенном каталоге, скорее всего будет возможен быстрый доступ сразу в нужный каталог без поиска по дереву каталогов.

HPFS обладает повышенной отказоустойчивостью по сравнению с FAT. Если на диске с FAT оказалась стертой таблица распределения файлов, то скорее всего окажутся потерянными все данные, которые находятся вне корневого каталога. В системе HPFS вместо таблицы размещения файлов применяется битовый массив, который содержит флаг, помечающий используемые секторы. Если область битового массива будет разрушена, пользователь этого не заметит, даже если это случится во время работы системы. F-узел файла также содержит информацию о размещении каждого файла. Поэтому область битового массива может быть восстановлена после поиска этой информации в F-узлах. Пользователь не увидит даже предупреждения - поиск выполняется автоматически. Этот процесс может быть запущен и с помощью утилиты CHKDSK, которая сравнивает битовый массив с информацией для файла о принадлежащих ему секторах. Если при чтении битового массива обнаруживается ошибка, то создается новый битовый массив.

В системе FAT при порче каталогов теряются указатели на начало цепочки кластеров каждого файла. Можно соединить отдельные кластеры в файл, но многое придется делать в ручную. Так как утилиты, подобные CHKDSK, не знают имени файлов, то для того, чтобы восстановить их старые имена, придется загружать файлы в текстовый редактор и пытаться определить, что они из себя представляют.

При работе с HPFS в случае потери каталога у каждого файла из этого каталога теряется лишь дата последней операции записи в файл и иных изменений, дата создания и длинное имя файла (символы, следующие за первыми пятнадцатью). Элемент каталога - это всего лишь указатель на F-узел. В F-узле хранятся первые 15 символов имени файла (плюс информация о том, имелись ли в имени файла другие символы, кроме первых 15) и прочая информация, нужная для доступа к файлу. Утилиты восстановления могут впоследствии найти в F- узле сведения о том или ином файле. Эта избыточность, обеспечиваемая каталогом и F-узлами, значительно увеличивает шансы на восстановление данных. CHKDSK в

настоящее время - единственная утилита восстановления, поставляемая с OS/2, которая, к сожалению, пока не использует всю имеющуюся информацию.

HPFS не налагает ограничений на максимальный размер файла, но OS/2 в настоящее время устанавливает предел в 2 Гбайта на один файл. Цель HPFS - доведение размера раздела до 2 Тбайт, но сегодня имеется ограничение в 64 Гбайта, поскольку некоторые части системы HPFS до сих пор остаются 16-разрядными.

LAN Server 4.0

Общая характеристика

Фирма IBM существенно улучшила свою серверную операционную систему LAN Server в последней версии 4.0. Упрощенная процедура установки этой системы по удобству сравнима с процедурой установки Windows NT Server в экспресс-режиме. LAN Server 4.0 удачно использует объектно-ориентированный интерфейс OS/2 для создания мощного набора графических средств администратора.

Производительность LAN Server 4.0 при работе с файлами и средствами сетевой печати при средних нагрузках значительно повысилась. LAN Server 4.0 может работать с новой версией OS/2 2.11, поддерживающей симметричное мультипроцессирование, что обеспечивает его хорошую масштабируемость. В тестах, проведенных сотрудниками PC Magazine Labs, версия OS/2 2.11 for SMP превзошла по производительности SCO Open Server в системах с 3 или 4 процессорами при большой нагрузке более чем в 1.5 раза (хотя и уступила Windows NT Server 3.5 при тех же условиях на 15% - 20%). Версия OS/2 2.11 for SMP по прежнему поддерживает существующие DOS и Windows приложения. Для систем с одним процессором однопроцессорная версия OS/2 на 10% быстрее, чем версия SMP. Худшее по сравнению с Windows NT увеличение производительности при увеличении числа процессоров может объясняться и принятой в OS/2 стратегией распределения процессоров между нитями - нить в течение всего периода своей жизни назначается всегда на один и тот же процессор, что не позволяет динамически оптимизировать загрузку процессоров.

LAN Server 4.0 работает как с файловой системой HPFS, так и с FAT. Существует упрощенная и более дешевая версия LAN Server 4.0, которая предназначена для небольших сетей и которая поддерживает только FAT.

Сетевые возможности

Система имеет средства поддержки протокола TCP/IP в составе пакета многопротокольной транспортной службы, что упрощает работу со стеками протоколов. Эта новая версия протоколов TCP/IP обеспечивает хорошую производительность. В отличие от Windows NT Server, LAN Server не содержит программы динамической конфигурации хоста DHCP, но ее появление

планируется в будущих версиях. Для доступа к Internet LAN Server использует утилиты, входящие в состав Warp 3.0. Сама по себе OS/2 Warp позволяет подключаться к Internet только по модему, но LAN Server активно участвует в работе этих утилит, делая их доступными и по локальной сети.

Управление сервером LAN Server 4.0

Практически все составляющие сети представлены в LAN Server представлены в виде объектов, для которых существует быстрый и эффективный механизм создания и управления. Возможность "перетаскивать" объекты существенно ускоряет и упрощает управление учетной информацией пользователей и правами их доступа к разделяемым ресурсам. Для пользователей, которые предпочитают работать с командной строкой, а не с мышью и пиктограммами, предоставляется мощный командный язык Rexx.

В дополнение к уровням пользователя и администратора IBM предлагает достаточное количество уровней привилегий, но ни один из них нельзя настраивать, а контроль администратора над использованием паролей если и обеспечивается, то минимальный. Кроме того, графические утилиты администратора системы не позволяют регламентировать время входа пользователя в сеть, хотя это можно делать с помощью утилит командной строки.

Особенно следует отметить возможность задания предельного объема дискового пространства, используемого пользователем. При достижении этого предела система предупреждает как администратора, так и самого пользователя (тем не менее она по-прежнему позволяет записывать данные пользователя на диск, что исключает риск их потери).

LAN Server позволяет администратору создавать профили пользователей, регулируя их доступ к определенным системным ресурсам. Можно создать единую процедуру входа пользователя в сеть, а также организовать централизованное управление сетевыми ресурсами с помощью концепции доменов. Процедуру входа в домен сформировать достаточно просто, но затем необходимо это сделать для каждого домена (здесь нет доверительных отношений между доменами). Конечно, такой вариант далек от идеала, но он все же лучше, чем вход в каждый сервер в отдельности. Если идентификаторы пользователя и его пароли в разных доменах совпадают, то в различных процедурах входа для каждого домена нет необходимости (при желании можно заказать программу Net Signon, осуществляющую согласование идентификаторов пользователя и его паролей между доменами). Служба псевдонимов системы LAN Server работает в масштабах всей сети, что позволяет серверам одного домена видеть ресурсы другого. Возможности централизованного управления сетью в этой системе ограничены. Графический интерфейс обеспечивает управление только шестью доменами, однако для утилит командной строки такого ограничения не существует.

Имея дело с Windows NT Server, приходится выбирать между файловыми системами: быстрой FAT и NTFS, обеспечивающей более надежную защиту информации. Используемая же в LAN Server файловая система HPFS оказалась лучше по обоим параметрам. Она работает эффективно и гарантирует безопасность файлов и каталогов. HPFS использует нулевое кольцо защиты микропроцессора Intel x86 и пытается обработать запросы на ввод-вывод в рамках одного прерывания. Кроме того, HPFS необходима для выполнения полного спектра задач администрирования сети, таких как защита файлов о несанкционированного доступа на самом сервере (это важно, поскольку LAN Server функционирует под управлением ОС, не содержащей процедуры аутентификации пользователя при входе).

В отличие от NetWare 4.1 и Windows NT Server, система LAN Server не удовлетворяет спецификациям уровня защиты С2, однако подобно этим сетевым операционным системам она использует зашифрованный ключ для аутентификации пользователей. Система регистрирует появляющиеся в сети ошибки и предупреждения и сопровождает их краткими комментариями. Тем не менее утилита ведения журнала контроля ограничивается лишь фиксацией событий, происходящих на сервере. В системе отсутствуют средства контроля ситуаций, возникающих в сети, регистрации транзакций и автоматической перезагрузки.

Термин "сервер приложений" приобретает в LAN Server второе значение. Администраторы могут выбирать приложения для DOS, Windows или OS/2, которые будут выполняться на сервере. Отдельные пользователи и группы с соответствующими правами осуществляют доступ к приложению, не заботясь о фактическом месте его выполнения. Администратор может незаметно для пользователей производить изменения на сервере, так как последний полностью прозрачен для них. Допустимо управлять сервером с рабочей станции.

В LAN Server 4.0 не предусмотрена возможность динамической настройки параметров производительности системы. Однако в ее состав входит утилита LS 4.0 Tuning Assistant, предоставляющая администраторам доступ к большому числу параметров. Правда, работая с этой утилитой, нужно хорошо знать особенности системы LAN Server и сетевых технологий низкого уровня.

Фирма IBM снабдила свою сетевую операционную систему и другими полезными утилитами, в том числе LAN Specialist, помогающей отслеживать ошибки (для этого требуется установить соответствующую программу на машину клиента), а также утилитой синхронизации файлов для удаленных пользователей (станции которых должны работать под управлением OS/2). На установочном компакт-диске имеется набор дополнительных программ под названием LAN Server Productivity Aids, включающий несколько утилит для данной ОС. Эти программы, в особенности Access Control Manager, который упрощает разделение ресурсов в сети, очень полезны.

LAN Server поддерживает только RAID-уровень 1, хотя существуют продукты третьих фирм для работы с системами RAID других уровней. Имеются также средства поддержки источников бесперебойного питания.

Совместимость с NetWare

Сетевая ОС LAN Server не содержит средств прямой связи с серверами NetWare (то есть не имеет стека протоколов Novell). Отчасти, это результат архитектурного решения: LAN Server работает под управлением OS/2, для которой пользователи могут приобрести отдельную программу - редиректор для NetWare. Можно создать шлюз между серверами по схеме "сервер под управлением базовой ОС", хотя результат получается более скромный, чем у системы Windows NT Server. Для этого нужно запустить редиректор NetWare на машине, функционирующей под управлением LAN Server. Пропускная способность такого соединения невелика, поэтому применять его надо с осторожностью.

Обзор сетевых операционных систем

Большое разнообразие типов компьютеров, используемых в вычислительных сетях, влечет за собой разнообразие операционных систем: для рабочих станций, для серверов сетей уровня отдела и серверов уровня предприятия в целом. К ним могут предъявляться различные требования по производительности и функциональным возможностям, желательно, чтобы они обладали свойством совместимости, которое позволило бы обеспечить совместную работу различных ОС.

Сетевые ОС могут быть разделены на две группы: масштаба отдела и масштаба предприятия. ОС для отделов или рабочих групп обеспечивают набор сетевых сервисов, включая разделение файлов, приложений и принтеров. Они также должны обеспечивать свойства отказоустойчивости, например, работать с RAID-массивами, поддерживать кластерные архитектуры. Сетевые ОС отделов обычно более просты в установке и управлении по сравнению с сетевыми ОС предприятия, у них меньше функциональных свойств, они меньше защищают данные и имеют более слабые возможности по взаимодействию с другими типами сетей, а также худшую производительность.

Сетевая операционная система масштаба предприятия прежде всего должна обладать основными свойствами любых корпоративных продуктов, в том числе:

- масштабируемостью, то есть способностью одинаково хорошо работать в широком диапазоне различных количественных характеристик сети,
- совместимостью с другими продуктами, то есть способностью работать в сложной гетерогенной среде интерсети в режиме plug-and-play.

Корпоративная сетевая ОС должна поддерживать более сложные сервисы. Подобно сетевой ОС рабочих групп, сетевая ОС масштаба предприятия должна

позволять пользователям разделять файлы, приложения и принтеры, причем делать это для большего количества пользователей и объема данных и с более высокой производительностью. Кроме того, сетевая ОС масштаба предприятия обеспечивает возможность соединения разнородных систем - как рабочих станций, так и серверов. Например, даже если ОС работает на платформе Intel, она должна поддерживать рабочие станции UNIX, работающие на RISC-платформах. Аналогично, серверная ОС, работающая на RISC-компьютере, должна поддерживать DOS, Windows и OS/2. Сетевая ОС масштаба предприятия должна поддерживать несколько стеков протоколов (таких как TCP/IP, IPX/SPX, NetBIOS, DECnet и OSI), обеспечивая простой доступ к удаленным ресурсам, удобные процедуры управления сервисами, включая агентов для систем управления сетью.

Важным элементом сетевой ОС масштаба предприятия является централизованная справочная служба, в которой хранятся данные о пользователях и разделяемых ресурсах сети. Такая служба, называемая также службой каталогов, обеспечивает единый логический вход пользователя в сеть и предоставляет ему удобные средства просмотра всех доступных ему ресурсов. Администратор, при наличии в сети централизованной справочной службы, избавлен от необходимости заводить на каждом сервере повторяющийся список пользователей, а значит избавлен от большого количества рутинной работы и от потенциальных ошибок при определении состава пользователей и их прав на каждом сервере.

Важным свойством справочной службы является ее масштабируемость, обеспечиваемая распределенностью базы данных о пользователях и ресурсах.

Такие сетевые ОС, как Banyan Vines, Novell NetWare 4.x, IBM LAN Server, Sun NFS, Microsoft LAN Manager и Windows NT Server, могут служить в качестве операционной системы предприятия, в то время как ОС NetWare 3.x, Personal Ware, Artisoft LANtastic больше подходят для небольших рабочих групп.

Критериями для выбора ОС масштаба предприятия являются следующие характеристики:

- Органичная поддержка многосерверной сети;
- Высокая эффективность файловых операций;
- Возможность эффективной интеграции с другими ОС;
- Наличие централизованной масштабируемой справочной службы;
- Хорошие перспективы развития;
- Эффективная работа удаленных пользователей;
- Разнообразные сервисы: файл-сервис, принт-сервис, безопасность данных и отказоустойчивость, архивирование данных, служба обмена сообщениями, разнообразные базы данных и другие;
- Разнообразные программно-аппаратные хост-платформы: IBM SNA, DEC NSA, UNIX;

- Разнообразные транспортные протоколы: TCP/IP, IPX/SPX, NetBIOS, AppleTalk;
- Поддержка многообразных операционных систем конечных пользователей: DOS, UNIX, OS/2, Mac;
- Поддержка сетевого оборудования стандартов Ethernet, Token Ring, FDDI, ARCnet;
- Наличие популярных прикладных интерфейсов и механизмов вызова удаленных процедур RPC;
- Возможность взаимодействия с системой контроля и управления сетью, поддержка стандартов управления сетью SNMP.

Конечно, ни одна из существующих сетевых ОС не отвечает в полном объеме перечисленным требованиям, поэтому выбор сетевой ОС, как правило, осуществляется с учетом производственной ситуации и опыта. В таблице приведены основные характеристики популярных и доступных в настоящее время сетевых ОС.

Таблица. 10.1.

Основные характеристики сетевых операционных систем

Novell NetWare 4.1	Специализированная операционная система, оптимизированная для работы в качестве файлового сервера и принт-сервера Ограниченные средства для использования в качестве сервера приложений: не имеет средств виртуальной памяти и вытесняющей многозадачности, а поддержка симметричного мультипроцессирования отсутствовала до самого недавнего времени. Отсутствуют API основных операционных сред, используемых для разработки приложений, - UNIX, Windows, OS/2 Серверные платформы: компьютеры на основе процессоров Intel, рабочие станции RS/6000 компании IBM под управлением операционной системы AIX с помощью продукта NetWare for UNIX Поставляется с оболочкой для клиентов: DOS, Macintosh, OS/2, UNIX, Windows (оболочка для Windows NT разрабатывается компанией Novell в настоящее время, хотя Microsoft уже реализовала клиентскую часть NetWare в Windows NT) Организация одноранговых связей возможна с помощью ОС PersonalWare Имеет справочную службу NetWare Directory Services (NDS), поддерживающую централизованное управление, распределенную, полностью реплицируемую, автоматически синхронизируемую и обладающую отличной масштабируемостью Поставляется с мощной службой обработки сообщений Message Handling Service (MHS), полностью интегрированную (начиная с версии 4.1) со справочной службой Поддерживаемые сетевые протоколы: TCP/IP, IPX/SPX, NetBIOS, Appletalk Поддержка удаленных пользователей: ISDN, коммутируемые телефонные линии, frame relay, X.25 - с помощью продукта NetWare Connect (поставляется отдельно) Безопасность: аутентификация с помощью открытых ключей метода шифрования RSA; сертифицирована по уровню C2 Хороший сервер коммуникаций

	Встроенная функция компрессии диска Сложное обслуживание
Banyan VINES 6.0 и ENS (Enterprise Network Services) 6.0	Серверные платформы: ENS for UNIX: работает на RISC-компьютерах под управлением SCO UNIX, HP-UX, Solaris, AIX ENS for NetWare: работает на Intel-платформах под управлением NetWare 2.x, 3.x, 4.x VINES работает на Intel-платформах Клиентские платформы: DOS, Macintosh, OS/2, UNIX, Windows for Workgroups, Windows NT Хороший сервер приложений: поддерживаются вытесняющая многозадачность, виртуальная память и симметричное мультипроцессирование в версии VINES и в ENS-версиях для UNIX. Поддерживаются прикладные среды UNIX, OS/2, Windows Поддержка одноранговых связей - отсутствует Справочная служба - Streettalk III, наиболее отработанная из имеющихся на рынке, с централизованным управлением, полностью интегрированная с другими сетевыми службами, распределенная, реплицируемая и автоматически синхронизируемая, отлично масштабируемая Согласованность работы с другими сетевыми ОС: хорошая; серверная оболочка работает в средах NetWare и UNIX; пользователи NetWare, Windows NT и LAN Server могут быть объектами справочной службы Streettalk III Служба сообщений - Intelligent Messaging, интегрирована с другими службами Поддерживаемые сетевые протоколы: VINES IP, TCP/IP, IPX/SPX, Appletalk Поддержка удаленных пользователей: ISDN, коммутируемые телефонные линии, X.25 Служба безопасности: поддерживает электронную подпись (собственный алгоритм), избирательные права доступа, шифрацию; не сертифицирована Простое обслуживание Хорошо масштабируется Отличная производительность обмена данными между серверами, хуже - при обмене сервер-ПК
Microsoft LAN Manager	широкая распространенность работает под OS/2 и UNIX поддерживает мощные серверные платформы один сервер может поддерживать до 2 000 клиентов
Microsoft Windows NT Server 3.51 и 4.0	Серверные платформы: компьютеры на базе процессоров Intel, PowerPC, DEC Alpha, MIPS Клиентские платформы: DOS, OS/2, Windows, Windows for Workgroups, Macintosh Организация одноранговой сети возможна с помощью Windows NT Workstation и Windows for Workgroups Windows NT Server представляет собой отличный сервер приложений: он поддерживает вытесняющую многозадачность, виртуальную память и симметричное мультипроцессирование, а также прикладные среды DOS, Windows, OS/2, POSIX Справочные службы: доменная для управления учетной информацией пользователей (Windows NT Domain Directory service), справочные службы имен WINS и DNS

	Хорошая поддержка совместной работы с сетями NetWare: поставляется клиентская часть (редиректор) для сервера NetWare (версий 3.x и 4.x в режиме эмуляции 3.x, справочная служба NDS поддерживается, начиная с версии 4.0), выполненная в виде шлюза в Windows NT Server или как отдельная компонента для Windows NT Workstation; недавно Microsoft объявила о выпуске серверной части NetWare как оболочки для Windows NT Server Служба обработки сообщений - Microsoft Mail, основанная на DOS-платформе, в этом году ожидается версия для платформы Windows NT - Microsoft Message Exchange, интегрированная с остальными службами Windows NT Server Поддерживаемые сетевые протоколы: TCP/IP, IPX/SPX, NetBEUI, Appletalk Поддержка удаленных пользователей: ISDN, коммутируемые телефонные линии, frame relay, X.25 - с помощью встроенной подсистемы Remote Access Server (RAS) Служба безопасности: мощная, использует избирательные права доступа и доверительные отношения между доменами; узлы сети, основанные на Windows NT Server, сертифицированы по уровню C2 Простота установки и обслуживания Отличная масштабируемость
IBM LAN Server 4.0	Серверные платформы: операционные системы MVS и VM для мейнфреймов; AS/400 с OS/400, рабочие станции RS/6000 с AIX, серверы Intel 486 или Pentium под OS/2 Поставляется с оболочками для клиентов: DOS, Macintosh, OS/2, Windows, Windows NT, Windows for Workgroups Серверы приложений могут быть организованы с помощью LAN Server 4.0 в операционных средах MVS, VM, AIX, OS/2, OS/400. В среде OS/2 поддерживаются: вытесняющая многозадачность, виртуальная память и симметричное мультипроцессирование Организация одноранговых связей возможна с помощью ОС Warp Connect Справочная служба - LAN Server Domain, то есть основа на доменном подходе Поддерживаемые сетевые протоколы: TCP/IP, NetBIOS, Appletalk Безопасность - избирательные права доступа, система не сертифицирована Служба обработки сообщений - отсутствует Высокая производительность Недостаточная масштабируемость
IBM и NCR LAN Manager	LAN Manager for UNIX хорошо распространена (15% объема мировых продаж сетевых ОС) LAN Manager for AIX поддерживает RISC компьютеры System/6000 в качестве файлового сервера Работает под UNIX, имеет все преимущества, связанные с использованием этой ОС