Е.М. Карчевский, О.В. Панкратова

Лекции по операционным системам (общий курс)

Учебное пособие

Казанский университет

2011

УДК 004.45 (075.8)
ББК 32.953.26–018.2
К 21

Печатается по решению Редакционно-издательского совета ФГАОУВПО
«Казанский (Приволжский) федеральный университет»

Методической комиссии института
Вычислительной математики и информационных технологий
Протокол №10 от 16 июня 2011 года

Заседания кафедры Вычислительной математики
Протокол №10 от 15 июля 2011 года

Авторы-составители
доктор физ. мат. наук Карчевский Е.М.
кандидат физ. мат. наук Панкратова О.В.

Научный редактор
кандидат физ. мат. наук Еникеев А.И.

Рецензенты
кандидат физ. мат. наук Якупов З.Я.
кандидат физ. мат. наук Ахтямов Р.Б.

Карчевский Е.М.

К21 Лекции по операционным системам (общий курс): учебное пособие / Е.М. Карчевский, О.В. Панкратова — Казань: Казан. ун-т, 2011. — 255 с.

Рассмотрены основные принципы построения и функционирования операционных систем.

Для студентов, специализирующихся в области прикладной математики и информатики.

УДК 004.45 (075.8)
ББК 32.953.26–018.2

Оглавление

Предисловие

Книга является изложением лекций по операционным системам, которые в течение нескольких лет читаются авторами для студентов института вычислительной математики и информационных технологий Казанского университета, специализирующихся в области прикладной математики и информатики.

Знание основ организации операционных систем и принципов их функционирования позволяет использовать компьютеры более эффективно. Глубокое изучение операционных систем помогает применять эти знания при создании программного обеспечения. К большому сожалению, в нашей стране в последние годы не создаются коммерческие операционные системы. Однако разработки сложных информационных систем, комплексов программ и отдельных приложений, предназначенных для работы в широко распространенных операционных системах, ведутся достаточно интенсивно. И здесь знание операционных систем, принципов их функционирования, методов организации вычислений является не только желательным, но обязательным.

Изучению операционных систем посвящен один из разделов курса "Системное и прикладное программное обеспечение", который входит в программу обучения студентов по специальности "Прикладная математика и информатика". Два других — "Системы программирования" и "Прикладные программные системы" — имеют хорошее учебно-методическое обеспечение. Для студентов нашего института в последние годы был подготовлен ряд пособий, которые интенсивно используются при проведении занятий по данному предмету. Однако лекции по операционным системам до сих пор не имели необходимой поддержки.

Учебный материал, ставший основой для настоящей книги, содержится в замечательных учебниках по операционным системам [1]–[3], изданных в России во второй половине двухтысячных годов. Мы ре-

комендуем их студентам в качестве дополнительной литературы для чтения.

Первая из упомянутых книг — это издание [1], допущенное Министерством образования Российской Федерации в качестве учебника для студентов высших учебных заведений, обучающихся по специальности "Информатика и вычислительная техника". В плане подготовки специалистов по этому направлению предусмотрено изучение отдельного предмета "Операционные системы". Поэтому многие разделы этого учебника выходят за рамки нашего курса, а изложение, более подробно, чем предусмотрено программой дисциплины "Системное и прикладное программное обеспечение".

Две другие книги — это западные учебники [2], [3], предназначенные для подготовки разработчиков системного программного обеспечения. Авторы этих изданий принимали участие в создании ряда известных операционных систем. Книги содержат бесценный материал и в свое время стали мировыми бестселлерами. Объем учебников позволил рассмотреть все основные вопросы проектирования операционных систем. Конечно, большинство из этих вопросов не рассматривается в рамках нашей дисциплины.

Цель настоящего курса — ввести студентов в круг понятий и проблем, связанных с общей структурой информационного обеспечения задач обработки данных на компьютерах, с тем чтобы студенты могли самостоятельно отвечать на теоретические и практические вопросы, возникающие при использовании операционных систем. В задачу курса входит ознакомление студентов с принципами построения и структурой аппаратно-программного окружения, в рамках которого протекают процессы выполнения программ, происходит управление взаимодействием программных процессов. В книге содержатся фактические данные, а именно, определения, схемы и диаграммами, снабженные примерами и комментариями. Каждый параграф завершается вопросами для самопроверки, позволяющими систематизировать приобретенные знания.

Многие вопросы, затронутые в книге, активно обсуждались с сотрудниками кафедр прикладной и вычислительной математики Казанского университета. Авторы приносят им свою искреннюю признательность.

Часть I

Введение

Глава 1

КРАТКАЯ ИСТОРИЯ ЭВМ

§ 1. Первое поколение ЭВМ (1945–1955): электронные лампы и коммутационные панели

Электронная вычислительная машина (ЭВМ, компьютер, computer) — это комплекс электронных устройств, предназначенных для хранения, обработки и передачи данных.

Когда появилась первая ЭВМ

- Первый компьютер на основе электронных ламп, в котором были воплощены принципы Джона фон Неймана (John von Neumann), был построен в 1948 году Морисом Уилксом (Morris Wilkes), Великобритания
- Во время Второй мировой войны разрабатывались вычислительные машины на основе электромеханических реле
  - Конрад Цузе (Konrad Zuse), 1941 год, Германия
  - Говард Айкен (Howard Aiken), 1943 год, США

Историческая справка: Джон фон Нейман

Янош Нейман родился в 1903 году в Будапеште в семье преуспевающего банкира. В 1913 году его отец получил дворянский титул, и Янош стал называться Янош фон Нейман. Позже, после переселения в США, его имя на английский манер изменилось на Джон.

В 1930 году фон Нейман был приглашен на преподавательскую должность в американский Принстонский университет. Он был одним из первых приглашённых на работу в основанный в 1930 году научно-исследовательский Институт Перспективных Исследований (Institute for Advanced Study), также располагавшийся в Принстоне, где с 1933 года и до самой смерти занимал профессорскую должность.

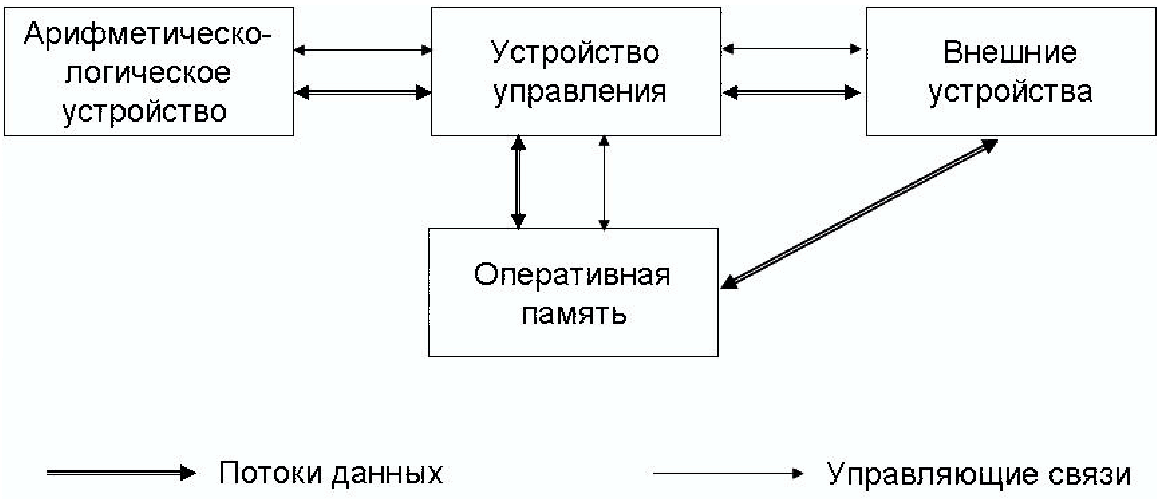


Рис. 1. Архитектура ЭВМ: принципы Джона фон Неймана

Наиболее известен как праотец современной архитектуры компьютеров (см. рис. 1 и 2), применением теории операторов к квантовой механике, а также как участник Манхэттенского проекта и как создатель теории игр и концепции клеточных автоматов.

В середине 40-х проект компьютера, хранящего свои программы в памяти был разработан в Муровской школе электрических разработок (The Moore School of Electrical Engineering) в Университете штата Пенсильвания (The University of Pennsylvania).

Подход, описанный в этом документе, стал известен как архитектура фон Неймана, по имени единственного из названных авторов проекта Джона фон Неймана, хотя на самом деле авторство проекта было коллективным. Информация о проекте стала доступна другим исследователям в 1946 году.

По плану предполагалось осуществить проект силами Муровской школы в машине “EDVAC”, однако “EDVAC” не был запущен до 1953 года из-за технических трудностей в создании надёжной компьютерной памяти.

Другие научно-исследовательские институты, получившие копии проекта фон Неймана, сумели решить эти проблемы гораздо раньше группы разработчиков из Муровской школы и реализовали их в собственных компьютерных системах.

В 1957 году фон Нейман заболел раком кости, возможно, вызван-

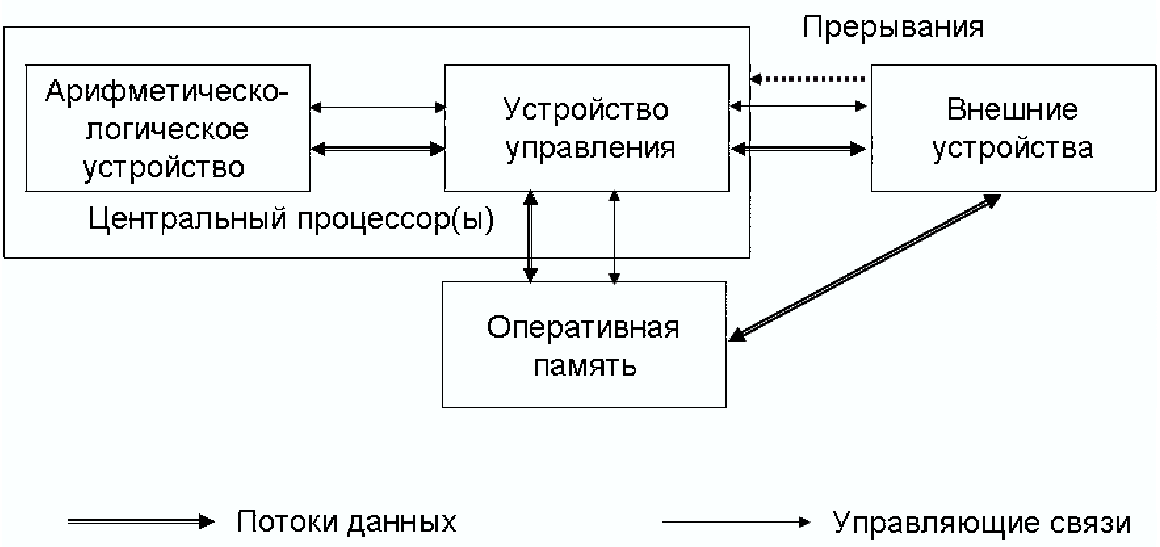


Рис. 2. Архитектура современных ЭВМ

ным радиоактивным облучением при исследовании атомной бомбы в Тихом океане или, может быть, при последующей работе в Лос-Аламосе, штат Нью-Мексико. Через несколько месяцев после постановки диагноза фон Нейман умер в тяжёлых мучениях.

Первое поколение ЭВМ

- Десятки тысяч электронных ламп (вероятность отказа)
- Программирование на абсолютном машинном языке
- Не было ОС, языков программирования (даже ассемблера)
- Машину разрабатывала, программировала и эксплуатировала одна команда
- На смену коммутационным панелям в начале 50-х годов пришли перфокарты

Вопросы для самопроверки

1. Существовали ли ЭВМ во время Второй мировой войны? (Да/Нет)

2. Имели ли ЭВМ первого поколения операционные системы? (Да/Нет)

Ответы на вопросы

1. Нет. Вычислительные машины во время Второй мировой войны работали на электромеханических реле. Электронные вычислительные машины на электронных лампах появились в конце 40-х годов.

2. Нет. В первых ЭВМ операционных систем не было. Программисты вводили команды на машинном языке при помощи коммутационных панелей или механических переключателей.

§ 2. Второе поколение ЭВМ (1955–1965): транзисторы и системы пакетной обработки

Транзистор (transistor) — миниатюрный полупроводниковый переключатель, который пропускает или не пропускает ток, позволяя процессорам выполнять операции, а памяти ЭВМ хранить данные по битам. Изобретен в США в 1948 году.

Задача (job) — совокупность работ, которая должна быть выполнена компьютером.

Однопоточная система пакетной обработки данных (single-stream batch-processing system) — разновидность ранних компьютерных систем, которая последовательно выполняла ряд неинтерактивных задач, по одной за раз (см. рис. 3).

Второе поколение ЭВМ (1955–1965)

- Применение транзисторов сделало ЭВМ более надежными
- Однопоточные системы пакетной обработки
- Первая операционная система для компьютера IBM 701 (середина 1950-х)
- Первые языки программирования (Язык ассемблера, Фортран)

Машинный язык (machine language) — язык, который определяется структурой аппаратных средств компьютера и может быть непосредственно воспринят ими. Программа на машинном языке записывается в двоичных кодах.

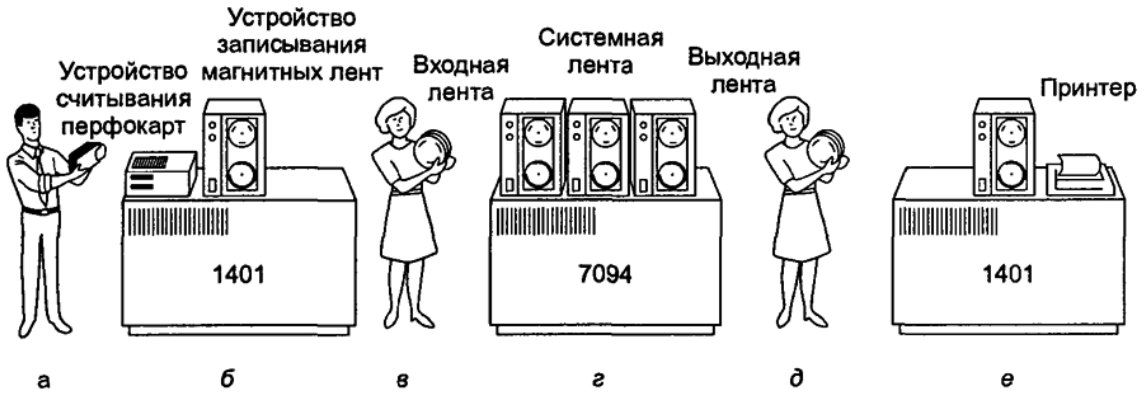


Рис. 3. Ранняя система пакетной обработки: а) программист приносит карты для IBM 1401; б) IBM 1401 записывает пакет заданий на магнитную ленту; в) оператор приносит входные данные на ленте к IBM 7094; г) IBM 7094 загружает компилятор с системной ленты, выполняет вычисления и записывает результат на выходную ленту; д) оператор переносит ленту с выходными данными на IBM 1401; е) IBM 1401 печатает выходные данные.

Язык ассемблера (assembly language) — язык низкого уровня, который представляет основные операции компьютера в виде английских сокращений названий команд.

Пример

```
LOAD basePay
ADD overTimePay
STORE grossPay
```

Ассемблер (assembler) — программа транслятор, которая переводит программы с языка ассемблера на машинный язык. Обычно является подпрограммой операционной системы.

Язык программирования высокого уровня (high-level language) — язык программирования, использующий английские идентификаторы и простой синтаксис для написания программы с применением меньшего количества операторов, чем при программировании на языках ассемблера.

Пример

```
grossPay = basePay + overTimePay
```

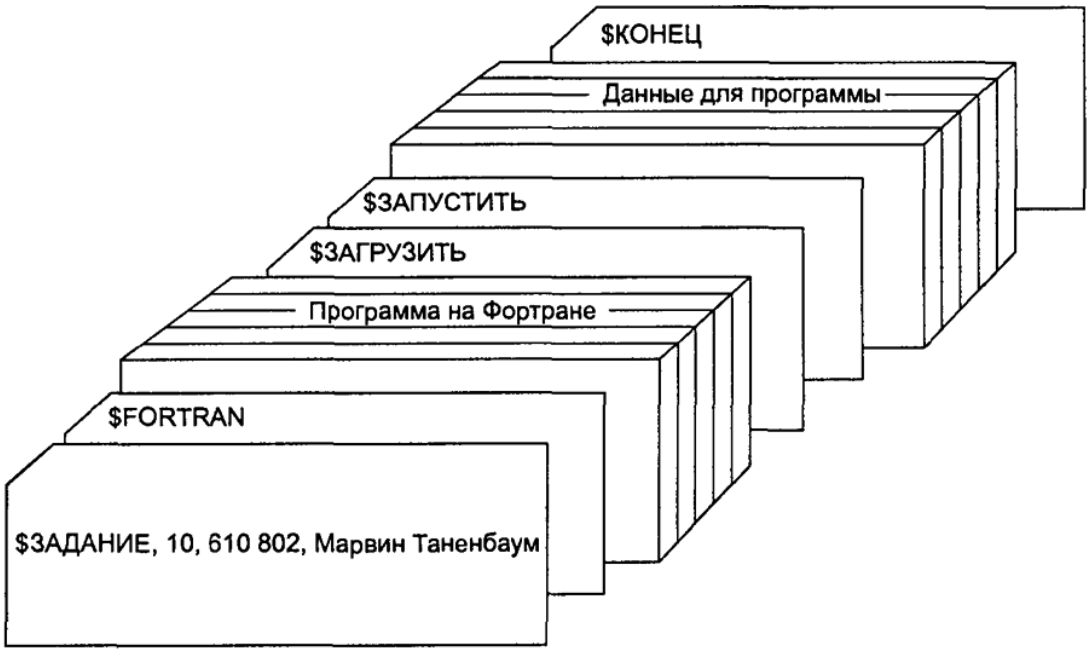


Рис. 4. Структура типичного задания FMS (Операционная система Fortran Monitor System)

Компилятор (compiler) — приложение, которое транслирует исходный код на языке высокого уровня в машинный код.

Язык Фортран (Fortran) — язык программирования высокого уровня, разработанный компанией IBM в середине 1950-х годов для научных приложений, которые требовали сложных математических вычислений (см. рис. 4).

Вопросы для самопроверки

1. Верно ли, что компьютеры выполняют программу непосредственно на языке ассемблера? (Да/Нет)

2. Можно ли писать программное обеспечение на переносимом машинном языке? (Да/Нет)

Ответы на вопросы

1. Нет. Программы-трансляторы осуществляют перевод программы с языка ассемблера на машинный язык, прежде чем ее можно будет выполнить.

2. Нет. Машинные языки являются машинно-зависимыми, таким

образом, программное обеспечение, написанное на определенном машинном языке, воспринимается только компьютерами одного типа.

§ 3. Третье поколение ЭВМ (1965–1980): интегральные схемы и многозадачность

Интегральная схема (integral circuit) — электронное устройство, состоящее из множества транзисторов.

IBM/360 (позднее 370, 390, zSeries)

- Первая линия компьютеров на интегральных схемах, 1964 год
- Серия программно совместимых машин разной производительности
- OS/360 состояла из миллиона строк, написанных на ассемблере тысячами программистов и содержала тысячи ошибок

Многозадачность (мультипрограммность, multiprogramming) — возможность одновременного хранения в памяти большого количества программ, так что они могут выполняться одновременно.

Ориентированный на ввод/вывод (I/O–bound) **процесс** (или задача), как правило, использует процессор в течение короткого промежутка времени, чтобы сформировать запрос на ввод/вывод, а затем освобождает процессор (см. рис. 5).

Ориентированный на вычисления (processor–bound) **процесс** (или задача), в полной мере использует свой квант процессорного времени при исполнении. Такие процессы, как правило, выполняют интенсивные вычисления и выдают мало запросов на ввод/вывод (см. рис. 5).

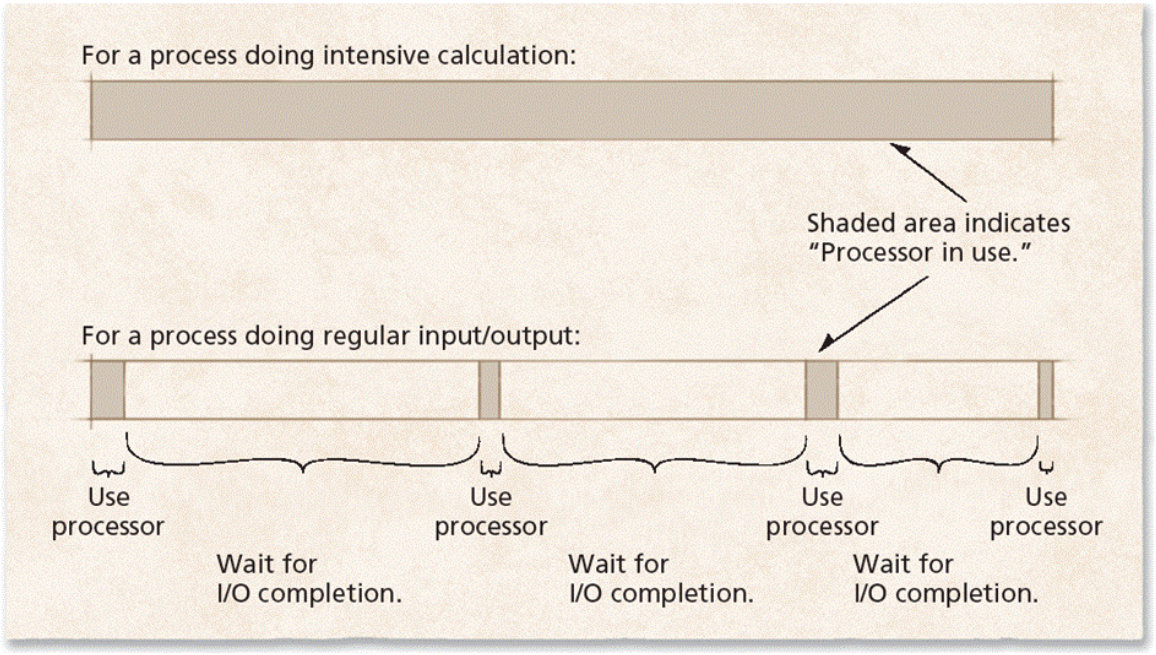


Рис. 5. Использование процессора в однозадачной системе ориентированным на вычисления процессом (вверху) и ориентированным на ввод/вывод процессом (внизу). Темная область означает "Процессор используется"

Многозадачность ОС третьего поколения

- В разных разделах оперативной памяти одновременно находятся ОС и несколько задач, одни из которых ориентированы на вычисления, а другие — на ввод/вывод
- Пока одна задача ожидает завершения операции ввода/вывода, другая использует центральный процессор

Спулинг (Simultaneous Peripheral Operation On Line, совместная периферийная операция в интерактивном режиме) — подкачка данных в оперативную память на фоне вычислений процессора.

Спулинг ОС третьего поколения

- ОС считывает задания с перфокарт на диск в фоновом режиме
- Когда текущее задание заканчивается ОС загружает новое задание с диска в освободившийся раздел оперативной памяти

Интерактивные пользователи (interactive user) — пользователи, которые находятся непосредственно возле машины, пока та решает их задачи. Интерактивные пользователи взаимодействуют с задачами в процессе их решения с помощью диалоговых терминалов.

Система с разделением времени (timesharing system) — операционная система, которая делает возможным одновременную работу многочисленных интерактивных пользователей.

CTSS

- CTSS (Compatible TimeSharing System) — совместимая система разделения времени
- Первая система с режимом разделения времени
- Разработана в 1960-х годах в Массачусетском технологическом институте (MTI) на специально переделанном компьютере IBM 7094

Виртуальная память (virtual memory) — способность операционной системы предоставлять программам доступ к большему пространству адресов, чем фактически существует в оперативной памяти.

Процесс (process) — выполняемая операционной системой программа.

MULTICS

- MULTICS (MULTiplexed Information and Computing Service) — мультиплексная информационная и вычислительная служба
- Совместная разработка MTI, исследовательской лаборатории Bell Labs и корпорации General Electric на мейнфрейме GE-645 (вторая половина 1960-х)
- Компьютерное предприятие общественного пользования в районе Бостона
- Первая крупная ОС, написанная на языке высокого уровня (PL/1)

- Одна из первых ОС с виртуальной памятью
- В системе MULTICS впервые использован термин процесс для описания программы в ходе ее выполнения

Миникомпьютеры

- PDP–1 выпущен корпорацией DEC в 1961 году (позднее PDP–2, . . . PDP–11)
- PDP–1 продавался по цене 120000 $ (5 % цены IBM 7094)
- Кен Томпсон написал UNIX — однопользовательскую версию MULTICS для PDP–7

Историческая справка: UNIX

Во времена, когда не было Windows, Macintosh, Linux и даже DOS, операционные системы, как правило, предназначались для работы на одной определенной модели компьютера: для управления системными ресурсами, выполнения потоковой обработки пакетов и, возможно, каких-то дополнительных функций.

С 1965 по 1969 год группа исследователей из Bell Labs, General Electric и MTI занималась разработкой OS MULTICS для компьютеров общего назначения под девизом “Все возможные функции для любого пользователя”. Эта система была дорогостоящей, громоздкой и сложной.

В 1969 году Bell Labs отстранилась от участия в этом проекте, собрав собственную небольшую команду разработчиков под руководством Кена Томпсона, которая взялась за разработку более практичной операционной системы для компьютеров Bell Labs. Она была названа UNICS в противовес MULTICS, затем обозначение было изменено на UNIX.

Через несколько лет система была переписана на интерпретируемом языке Томпсона B, а вскоре после этого — на более быстром компилируемом языке C (создателем которого является Денис Ритчи).

Группа студентов Калифорнийского университета в Беркли, которой руководил Билл Джой (позднее ставший соучредителем Sun

Microsystems), модифицировав исходные коды UNIX, создала BSD UNIX (Berkley Software Distribution UNIX).

Отраслевые разработчики программного обеспечения переходили на UNIX, привлеченные ее бесплатностью, компактностью и широкими возможностями настройки. Чтобы работать в UNIX разработчики должны были учить язык C, который им явно нравился. Большинство разработчиков обучались в колледжах, где в курсах по программированию предпочтения понемногу смещались от Pascal в сторону языка C.

Компания Sun Microsystems взяла за основу своей SunOS именно версию BSD UNIX и только позднее в сотрудничестве с AT&T разработала операционную систему Solaris на основе версии UNIX от AT&T.

Другая группа разработчиков UNIX, обеспокоенная тем, что сотрудничество Sun и AT&T ставит компанию Sun в неравные условия перед другими разработчиками UNIX, сформировала Фонд открытого программного обеспечения (Open Software Foundation, OSF) для создания собственной открытой версии UNIX под названием OSF/1. Жесткая конкуренция между OSF и AT&T, за спиной которой стояла Sun, получила название "войн UNIX".

На основе технологии UNIX было создано несколько известных операционных систем. В 1987 году профессор Эндрю Таненбаум из университета Врейе (Vreje) в Амстердаме создал операционную систему Minix — урезанную версию ОС, предназначенную специально для обучения основам проектирования операционных систем, которая до сих пор используется в этих целях.

Линус Торвальдс, финский студент, использовал Minix в качестве основы при создании хорошо известной ныне операционной системы Linux, распространяемой по принципу "открытого кода". Linux представляет собой наиболее популярную систему с открытыми кодами, и такие компании, как IBM, Hewlett Packard, Sun Microsystems, Intel предлагают версии Linux для своих серверов.

Другим проектом с открытыми кодами стала операционная система OpenBSD, которая считается самой защищенной операционной системой в мире. Еще одна операционная система с открытыми кодами — FreeBSD, которая славится простотой использования. Другой

потомок BSD, NetBSD ориентирована в первую очередь на переносимость между различными системами.

Вопросы для самопроверки

1. Повысило ли производительность труда программистов применение интерактивного режима работы? (Да/Нет)

2. Была ли реализована в системе MULTICS концепция виртуальной памяти? (Да/Нет)

Ответы на вопросы

1. Да. Время между постановкой задачи и возвратом результатов ее выполнения было сокращено от часов до минут. Это позволило программистам вводить, компилировать, редактировать, тестировать и отлаживать свои программы в интерактивном режиме, вплоть до устранения всех ошибок.

2. Да. Эта система включала виртуальную память, которая позволяет приложениям обращаться к большему пространству памяти, чем физически существует в системе. Это предоставило программистам возможность разработки более мощных приложений и избавило от необходимости решать сложные задачи управления памятью.

§ 4. Четвертое поколение ЭВМ (1980–н.в.): микросхемы и персональные компьютеры

Микросхема (большая интегральная схема, LSI, Large Scale Integration) — кремниевое микроминиатюрное электронное устройство, содержащее тысячи транзисторов на одном квадратном сантиметре.

Пример. Intel 8080 — первый универсальный 8-разрядный центральный процессор на микросхеме, 1974 год.

CP/M

- CP/M (Control Program for Microcomputers) — программа управления для микрокомпьютеров
- Первая операционная система для персональных компьютеров с Intel 8080

- Разработана в 1977 году Гэри Килдэллом (компания Digital Research)
- Лидировала среди ОС до 1983 года

MS-DOS

- MS-DOS (MicroSoft Disk Operation System) — дисковая операционная система фирмы Microsoft
- Первая операционная система для персональных компьютеров IBM PC
- Разработана в 1984 году Тимом Патерсоном (компания Microsoft)

Историческая справка: MS-DOS

В начале 80-х корпорация IBM разработала персональный компьютер IBM PC с процессором Intel 8080 и начала искать для него программное обеспечение. Сотрудники IBM связались с Билом Гейтсом, чтобы получить лицензию на право использования его интерпретатора языка Бейсик. Они также поинтересовались, не знает ли он операционную систему, которая работала бы на PC.

Гейтс посоветовал обратиться к Digital Research, тогда главенствующей компании по операционным системам. Но Килдэлл отказался встречаться с IBM, послав вместо себя подчиненного. Что еще хуже, его адвокат даже отказался подписывать соглашение о неразглашении, касающееся еще не выпущенного PC, чем полностью испортил дело. Корпорация IBM снова обратилась к Гейтсу с просьбой обеспечить ее операционной системой.

После повторного запроса IBM Гейтс выяснил, что у местного изготовителя компьютеров, Seattle Computer Products, есть подходящая операционная система DOS. Он направился в эту компанию с предложением выкупить DOS за 50 000 $, которое компания Seattle Computer Products с готовностью приняла.

Затем Гейтс создал пакет программ DOS/BASIC, и пакет был куплен IBM. Когда корпорация IBM захотела некоторых усовершенствований в операционной системе, Бил Гейтс пригласил для этой работы Тима Патерсона, человека, написавшего DOS, ставшего первым служащим еще не оперившейся компании Гейтса Microsoft.

Видоизмененная система была переименована в MS-DOS и быстро заняла доминирующее положение на рынке IBM PC. Самым важным оказалось решение Гейтса (как оказалось, чрезвычайно мудрое) продать MS-DOS компьютерным компаниям для установки вместе с их оборудованием, в отличие от попыток Килдэлла продавать CP/M конечным пользователям.

Когда в 1983 году появился компьютер IBM PC/AT с центральным процессором Intel 80286, система MS-DOS уже прочно стояла на ногах, а CP/M доживала свои последние дни. Позже система MS-DOS широко использовалась на компьютерах с процессорами 80386 и 80486. Хотя первоначальная версия MS-DOS была довольно примитивна, последующие версии системы выходили со все лучше разработанными свойствами, включая многое, позаимствованное от UNIX.

Графический интерфейс пользователя (GUI, Graphical User Interface) — удобное для пользователя средство доступа к операционной системе, включающее в себя графические элементы, такие как окна, значки и меню, предназначенные для упрощения работы с программами и файлами.

Историческая справка: Дуглас Энгельбарт

Дуглас Энгельбарт изобрел компьютерную мышь и был одним из первых разработчиков оригинальных графических дисплеев и окон.

Энгельбарт специализировался в сфере электроники. Во времена Второй мировой войны он работал специалистом по различным системам, например, радиолокационным и гидролокационным станциям.

Покинув ряды вооруженных сил, он вернулся в штат Орегон, чтобы получить степень по электротехнике в 1948 году. Затем он окончил аспирантуру и поступил на работу в Стенфордский Исследовательский Институт (SRI, Stanford Research Institute), где впервые и столкнулся с компьютерами.

В 1968 году на Объединенной конференции по компьютерам в Сан-Франциско Энгельбарт и его сотрудники представили свою компьютерную систему NLS (oNLine System — это название можно перевести как “Диалоговая система”). Ее отличительными элементами были мышь и графический интерфейс с окнами.

Первая компьютерная мышь, которая называлась “Индикатором координат X–Y для системы вывода изображений”, имела только одну кнопку. На ее нижней поверхности находились два колесика, горизонтальное и вертикальное. Мышь и графические окна были взаимосвязаны. Благодаря мыши переключение с одного окна на другое значительно упрощалось, в то время как без окон мышь уже не приносила столь значимой пользы.

Дуглас Энгельбарт посвятил свою жизнь повышению уровня человеческого интеллекта. Его изначальным замыслом, лежащим и в основе NLS, было создание системы, которая могла бы повысить уровень интеллекта и помочь людям быстрее решать задачи. Он основал Институт Совершенствования (Bootstrap Institute) с целью способствовать всеобщему пониманию его миссии. Совершенствование, согласно Энгельбарту, это развитие методов самого процесса развития. Он верит в то, что этот путь является наилучшим для повышения уровня человеческого интеллекта.

На сегодняшний день Энгельбарт все еще сотрудничает с Институтом Совершенствования. За свои труды он получил широкое общественное признание и множество наград.

GUI

- Apple Macintosh, 1984 год:
  - первая операционная система для персонального компьютера с GUI, поддерживающая мышь и окна;
  - разработана Стивом Джобсом.
- Windows 1.0, 1985 год:
  - графическая оболочка для MS-DOS корпорации Microsoft;
  - развивалась до 1995 года.

Вопросы для самопроверки

1. Верно ли, что Macintosh была первой операционной системой с графическим интерфейсом пользователя? (Да/Нет)

2. Верно ли, что MS-DOS разработал Бил Гейтс? (Да/Нет)

Ответы на вопросы

1. Нет. Macintosh была первой операционной системой с GUI для персонального компьютера. Впервые GUI был реализован в 1968 году в системе NLS Дугласа Энгельбарта.

2. Нет. MS-DOS разработал в 1984 году Тим Патерсон.

§ 5. История сети Интернет и всемирной паутины

Сеть ARPA net

- Прародитель сети Интернет
- Разработана в 1969 году ARPA (Advanced Research Projects Agency, управление перспективных исследовательских программ Министерства обороны США)
- Соединяла около дюжины компьютерных систем университетов и научно-исследовательских организаций, финансируемых ARPA
- Выделенные линии связи со скоростью 56 Кбит/сек.
- Передача информации посредством электронной почты (e-mail)
- Сеть ARPA net была децентрализована, перенаправляла трафик в обход поврежденных узлов
- Для ARPA net было разработано семейство протоколов TCP/IP

Протокол управления передачей / протокол Интернет (Transmission Control Protocol, TCP/IP) — семейство протоколов, которые определяют способы обмена информацией в сети Интернет.

Интернет (Internet) — сеть каналов связи, которая является технологической основой для телекоммуникаций и всемирной паутины (см. рис. 6). Каждый компьютер сети Интернет определяет, какими услугами сети он пользуется и какие услуги он предоставляет другим компьютерам, подключенным к Интернету.

Всемирная паутина (WWW, World Wide Web) — совокупность гипертекстовых документов, доступных через сеть Интернет

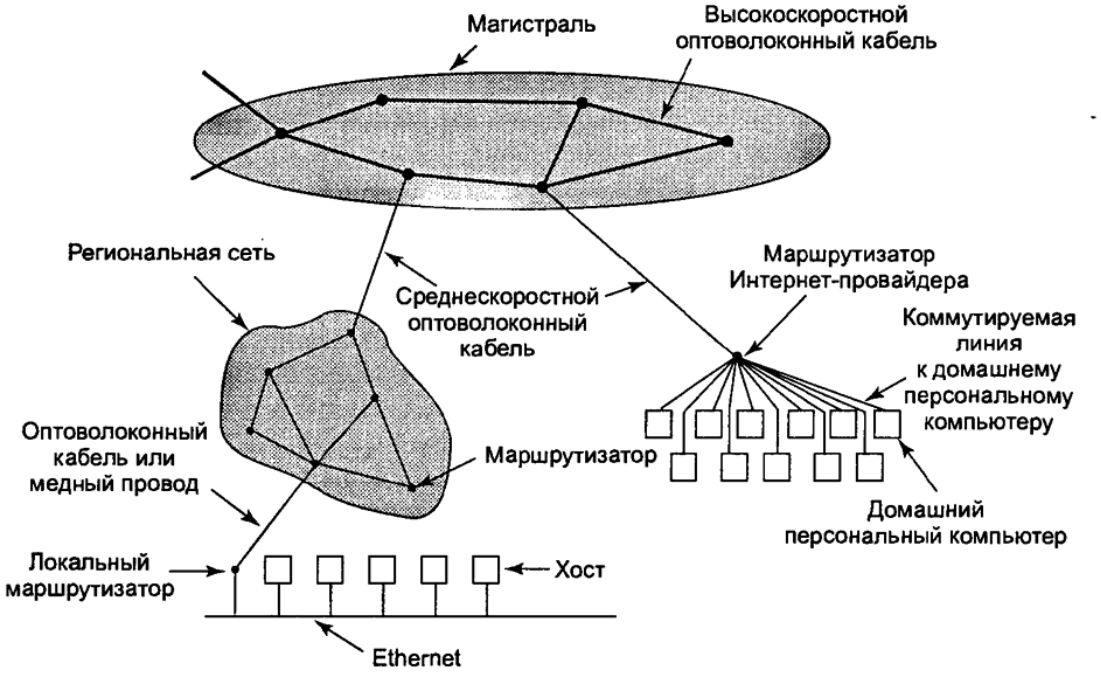


Рис. 6. Схема части Интернета. Здесь Ethernet — локальная сеть, хост — клиентский компьютер, маршрутизатор — коммутирующий компьютер, принимающий пакеты с одной или нескольких линий и отправляющий их дальше по одной или нескольким линиям

при помощи протокола передачи гипертекстовых файлов (HTTP). Веб-документы, как правило, пишутся на таких языках, как HTML.

Протокол передачи гипертекстовых файлов (HyperText Transfer Protocol, HTTP) — сетевой протокол, используемый для обмена документами на языке HTML и данными других форматов между клиентом и сервером. Это основной протокол всемирной паутины.

Гипертекстовый язык разметки (HyperText Markup Language, HTML) — язык, который описывает содержимое и порядок отображения содержимого веб-страницы, а также предоставляет гиперссылки для доступа к другим страницам.

Историческая справка: Тим Бернерс-Ли

Всемирная паутина была разработана Тимом Бернерсом-Ли в 1989 году. Она дает возможность пользователям компьютеров размещать и просматривать мультимедиа-документы (т.е. текстовые, гра-

фические, анимационные, аудио- и видеофайлы) практически любой тематики.

Тим Бернерс-Ли окончил Королевский колледж при Оксфордском университете со степенью по физике в 1976 году. В 1980-м он написал программу, которая использовала гипертекстовые ссылки для облегчения работы с многочисленными документами большого проекта.

В 1984 году он стал сотрудником Европейской организации ядерных исследований (European Center for Nuclear Research, CERN), где и получил опыт работы с коммуникационным программным обеспечением для сетевых систем, работающих в режиме реального времени.

Работая в Европейской организации ядерных исследований, в 1989 году Бернерс-Ли создал Протокол передачи гипертекстовых файлов (HTTP), язык HTML, а также первый сервер и навигатор для Всемирной паутины.

Он хотел, чтобы Паутина стала механизмом для открытого и свободного доступа ко всеобщим знаниям и опыту. До 1993 года Бернерс-Ли лично занимался изменениями и предложениями для протокола передачи гипертекстовых файлов и языка HTML, которые поступали от первых пользователей Паутиной.

К 1994 году количество последних стало настолько большим, что ему пришлось учредить Консорциум Всемирной Паутины (World Wide Web Consortium — W3C, www.w3c.org) для создания стандартов веб-технологий и контроля над ними. Будучи директором этой организации, он активно пропагандирует принципы свободного доступа к информации, предоставленной открытыми технологиями.

Вопросы для самопроверки

1. Имела ли сеть ARPA net центральный управляющий компьютер? (Да/Нет)

2. Использовала ли сет ARPA net протокол HTTP? (Да/Нет)

Ответы на вопросы

1. Нет. Сеть ARPA net была целиком децентрализована, таким образом, она не теряла способность передавать информацию в случае выхода из строя некоторых ее узлов.

2. Нет. Для ARPA net было разработано семейство протоколов TCP/IP. Протокол HTTP создал Тим Бернерс-Ли в 1989 году.

§ 6. Девяностые годы

- Экспоненциальный рост производительности аппаратных средств
- Распределенные вычисления стали обычным явлением среди домашних ПК, подключенных к Интернет
- MS Windows 95, 98, NT заняли лидирующее место на рынке ПО для ПК

Объектно-ориентированные языки программирования

- C++ — расширение языка C, разработанное Бьярне Страуструпом (Bell Labs) в начале 1980-х годов
- Java — разработан компанией Sun Microsystems в 1995 году. Упрощает переносимость, обладая возможностью исполнения на виртуальной машине
- C# (Си-шарп) — разработан корпорацией Microsoft в 2000 году. Обеспечивает доступ к библиотекам .NET

Объектно-ориентированное программирование (object-oriented programming, OOP) — концепция программирования, позволяющая разработчикам программного обеспечения быстро создавать сложные программные системы, используя компоненты многоразового использования — объекты, созданные на основе шаблонов — классов.

Объект (object) — программный компонент многоразового использования, который может моделировать элементы реального мира посредством своих свойств и действий.

Класс (class) — тип объекта. Определяет атрибуты и методы объекта.

Объектно-ориентированная операционная система (object-oriented operating system, OOOS) — операционная система, в которой

ресурсы и компоненты представлены в виде объектов. Наследование и интерфейсы помогают создавать модульные операционные системы, которые являются более простыми в обслуживании и расширении. Многие ОС используют объекты, но не многие были полностью написаны на объектно-ориентированных языках.

Открытое программное обеспечение (open-source software) — программное обеспечение, к которому прилагается исходный код и которое обычно распространяется согласно открытому лицензионному соглашению (General Public License, GPL). Такое программное обеспечение, как правило, разрабатывается группами независимых программистов по всему миру.

Открытое программное обеспечение

- Может быть просмотрено и модифицировано любым членом сообщества
- Существует большая вероятность того, что ошибки будут выявлены и удалены
- ПО может быть модифицировано под специфические потребности отдельных организаций
- Производители могут взыскивать плату за предоставление ПО с GPL-лицензией, но не могут препятствовать дальнейшему распространению открытого ПО
- GNU — проект свободного распространения программного обеспечения (www.gnu.org):
  - запущен Ричардом Столлменом в 1980-х годах;
  - был ориентирован на создание открытой операционной системы с возможностями и инструментарием UNIX.
- OSI — инициативная группа по распространению открытого программного обеспечения (Open Source Initiative) — организация, которая занимается поддержкой и популяризацией открытого ПО (www.opensorce.com)

Историческая справка: Линус Торвальдс

Линус Торвальдс родился в 1969 году в Хельсинки, Финляндия. Будучи ребенком он учился программированию, играя с Commodore VIC-20. В 1988 году он поступил в Хельсинский университет для изучения вычислительной техники. Там он написал версию системы UNIX на основе Minix профессора Эндрю Таненбаума для работы на своем новом ПК. В 1991 году он завершил первую версию основного ядра Linux, которое работало на процессоре Intel 80386.

Когда Линус Торвальдс создавал оригинальную версию операционной системы Linux, он бесплатно использовал многие из средств, опубликованных в рамках проекта свободного распространения программного обеспечения (GNU). В свою очередь он сам распространял Linux согласно открытому лицензионному соглашению (GPL) в качестве открытого кода, с благодарностью включив в него предоставленные другими программистами исправления, дополнения и прочие программы.

К 1994 году система Linux набрала достаточное количество приложений для того, чтобы стать полнофункциональной и доступной для использования операционной системой, — была выпущена версия 1.0. Программистам и профессорам, использовавшим UNIX для работы с большими системами, OS Linux очень понравилась, поскольку она смогла перенести возможности и ресурсы операционной системы UNIX на недорогие настольные системы без денежных затрат.

На сегодняшний день Торвальдс является сотрудником Лаборатории по разработке открытого программного обеспечения (Open Source Development Labs, OSDL), которая финансирует отнимающую всю его время работу над ядром. Он продолжает руководить проектом, который посвящен открытой операционной системе Linux, занимаясь изменениями и выпуском новых версий ядра.

Операционная система Linux стала одной из наиболее известных разработок в области открытого ПО за всю историю существования компьютеров и пользуется особым успехом на рынке серверов.

Вопросы для самопроверки

1. Верно ли, что многие современные операционные системы являются объектно-ориентированными? (Да/Нет)

2. Верно ли, что открытое лицензионное соглашение позволяет свободно просматривать, модифицировать и распространять открытое программное обеспечение? (Да/Нет)

Ответы на вопросы

1. Да. Разработчики ОС могут повторно использовать объекты при разработке новых компонентов. Увеличение модульности благодаря применению объектно-ориентированной технологии способствует поддержке новых архитектур.

2. Да. Конечные пользователи могут свободно модифицировать и распространять любое ПО, удовлетворяющее GPL. ПО, распространяемое согласно этой лицензии, должно содержать полный исходный код, сведения о всех его модификациях и сопровождаться общедоступной лицензией.

§ 7. Двухтысячные годы

- Производительность процессоров отстает от экспоненциального роста числа транзисторов в них
- Многопроцессорные системы:
  - мультипроцессоры (специальные ОС);
  - многомашинные системы (несколько одинаковых ОС);
  - распределенные системы (разные ОС на разных машинах).
- Обработка данных на мобильных устройствах

Стандартизация интерфейсов пользователей и приложений

- Windows XP объединила потребительскую и профессиональные линии OS Windows (Windows Me и Windows 2000)
- Linux и другие открытые ОС используются шире и применяют стандартные программные интерфейсы приложений (напр., POSIX) для повышения совместимости с другими ОС на базе UNIX

Программный интерфейс приложений (Application Programming Interface, API) — спецификация, которая позволяет приложениям запрашивать услуги у ядра посредством системных вызовов (System call interface, см. рис. 7).

Пример. Интерфейс переносимых операционных систем (Portable Operating System Interface, POSIX) — программный интерфейс приложений, первоначально разработанный для ранней ОС Unix. Применяется для совместимости ОС, основанных на Unix.

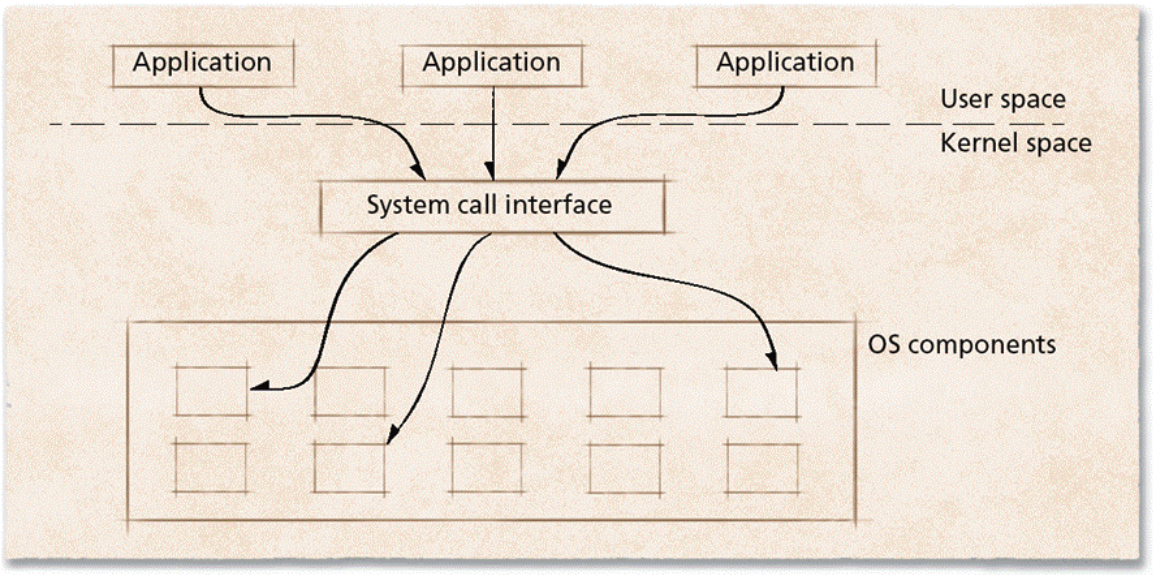


Рис. 7. Взаимодействие приложений с операционной системой. В пространстве пользователя (User space) показаны компоненты программного обеспечения, не обладающие непосредственным доступом к физическим ресурсам системы. В пространстве ядра (Kernel space) показаны компоненты ОС, обладающие неограниченным доступом к системным ресурсам.

Виртуальная машина (virtual machine) — приложение, эмулирующее функциональные возможности компьютерной системы (см. рис. 8).

Виртуальные машины

- Дают многочисленным пользователям возможность работы с аппаратными средствами в режиме коллективного пользования

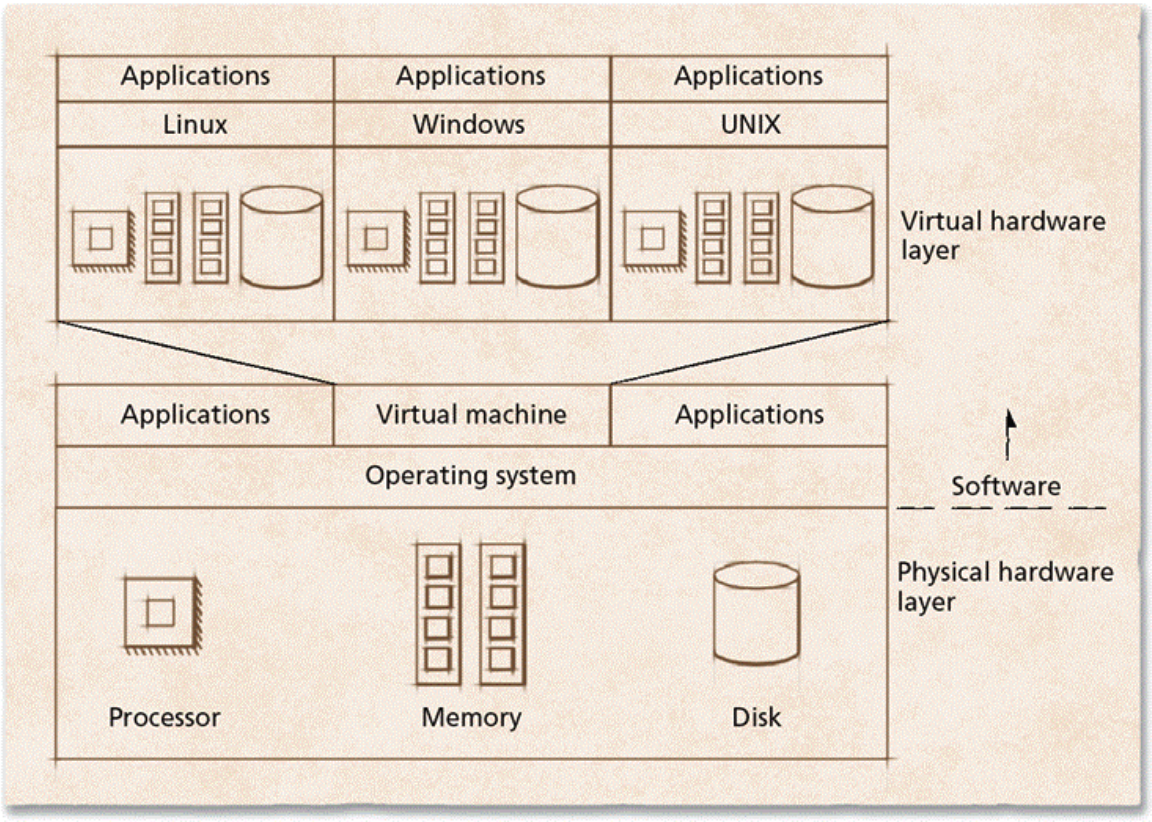


Рис. 8. Схема виртуальной машины. Пунктирная линия разделяет уровни реальной и виртуальной аппаратуры, эмулируемой программным обеспечением

при иллюзии непосредственного доступа к ресурсам базового компьютера

- Эта концепция впервые была реализована в системе VM/370 на IBM 370 и до сих пор широко используется на еще оставшихся мэйнфреймах
- Могут выполнять приложения, не полностью совместимые с физической системой, на которой функционирует виртуальная машина
- Например, виртуальные машины DOS в Windows
- Повышают мобильность программного обеспечения, т.е. возможность функционировать на различных платформах
- Например, виртуальные машины Java

Виртуальная машина Java (Java Virtual Machine, JVM) — виртуальная машина, которая позволяет выполнять программы, написанные на языке Java, на различных платформах, не компилируя заново эти программы на машинный язык компьютеров, где они выполняются.

Виртуальная машина Java

- Повышает мобильность приложений, т.е. возможность программного обеспечения функционировать на различных платформах
- Упрощает программирование, освобождая программиста от необходимости изучать архитектурные спецификации различных платформ

Вопросы для самопроверки

1. Верно ли, что виртуальные машины в большинстве случаев обладают более низкой производительностью, чем реальные машины? (Да/Нет)

2. Оказывает ли распределенная обработка данных помощь вычислениям, выполняемым на мобильных устройствах? (Да/Нет)

Ответы на вопросы

1. Да. Виртуальные машины, как правило, не обладают столь высокой производительностью, как реальные машины, поскольку им приходится выполнять программные действия, эмулирующие работу аппаратных средств. Зато они способствуют повышению мобильности программ, позволяя им функционировать на различных платформах.

2. Да. Распределенная обработка данных позволяет мобильным устройствам передавать задачи другим машинам, обладающим большим запасом ресурсов. Мобильное устройство, имея ограниченные ресурсы и срок службы аккумуляторной батареи, может запрашивать данные и вычислительную мощность по сети у более мощного компьютера.

Глава 2

ОПЕРАЦИОННЫЕ СИСТЕМЫ

§ 1. Что такое операционная система

Операционная система (ОС, Operating System, OS) — это программное обеспечение (см. рис. 1), которое управляет аппаратными средствами, приложениями и другими программными объектами компьютера (логическими ресурсами).

Ядро (kernel) — это комплекс программ, составляющих основу операционной системы.

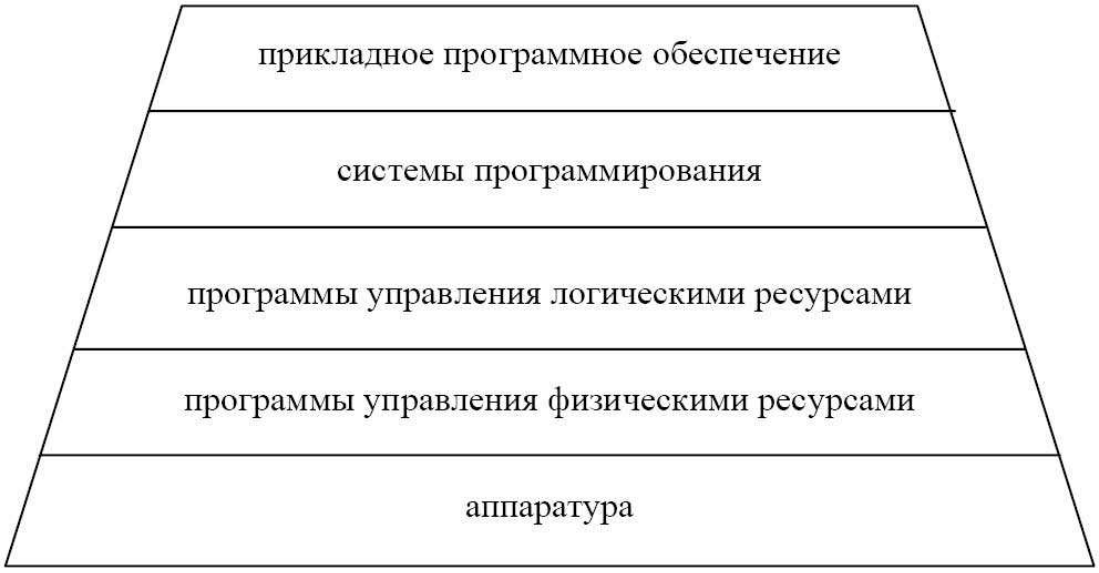


Рис. 1. Иерархия программно-аппаратного обеспечения

Основные задачи операционных систем

- Обеспечение взаимодействия между приложениями и аппаратными средствами компьютера
- Обеспечение взаимодействия между приложениями

- Распределение программных и аппаратных ресурсов системы:
  - распределение во времени (процессор, принтер);
  - распределение в пространстве (оперативная память, жесткий диск).

Примеры операционных систем

- ОС для персональных компьютеров (Windows Vista Home, Linux, Macintosh) предоставляют удобный интерфейс для одного пользователя
- Встроенные ОС (Palm OS, Windows Consumer Electronics) управляют работой карманных компьютеров, мобильных телефонов
- ОС для смарт-карт управляют работой процессора и ПЗУ смарт-карты, интерпретатора виртуальной машины Java
- ОС реального времени:
  - жесткие системы реального времени (системы управления воздушным движением);
  - гибкие системы реального времени (мультимедийные ОС, QNX, VxWorks).
- Серверные ОС (Unix, Linux, Windows XP Professional, Windows 2000) одновременно обслуживают множество пользователей и позволяют им делить между собой программные и аппаратные ресурсы (работа интернет-провайдеров)
- ОС мэйнфреймов (OS/390) ориентированы на обработку множества одновременных заданий, большинству из которых требуется огромное количество операций ввода-вывода:
  - пакетная обработка (исков в страховых компаниях);
  - обработка транзакций — групповых операций (бронирование авиабилетов);
  - разделение времени (электронно-коммерческий сайт).

- ОС суперкомпьютеров из списков Top 500, Top 50 (Unix, Linux, специальные модификации Windows)
- Earth Simulator (Сеймур Крэй, Иокогама, Япония, Июль 2003):
  - 5120 процессоров 500 МГц;
  - 35,6 Тфлоп (тэтафлоп — триллион операций над числами с плавающей запятой в секунду);
  - 10 терабайт ОП (1 Тб = 1024 Гб = 2^{40} Байт);
  - 640 Тб дисковая система;
  - 1,5 Петабайт ленточные накопители (1 Пб = 1024 Тб = 2^{50} Байт);
  - 1500 миль сетевого кабеля.

Вопросы для самопроверки

1. Операционные системы управляют только аппаратными средствами? (Да/Нет)

2. Верно ли, что основной задачей операционных систем является организация удобного интерфейса пользователя? (Да/Нет)

Ответы на вопросы

1. Нет. Операционные системы управляют также приложениями и другими программными объектами, такими как виртуальные машины.

2. Нет. Многие ОС предоставляют минимальный интерфейс пользователя. Основными задачами ОС являются обеспечение взаимодействия между приложениями и аппаратными средствами компьютера, а также распределение программных и аппаратных ресурсов системы.

§ 2. Компоненты операционных систем

- Ядро (см. рис. 2)
- Драйверы
- Оболочка

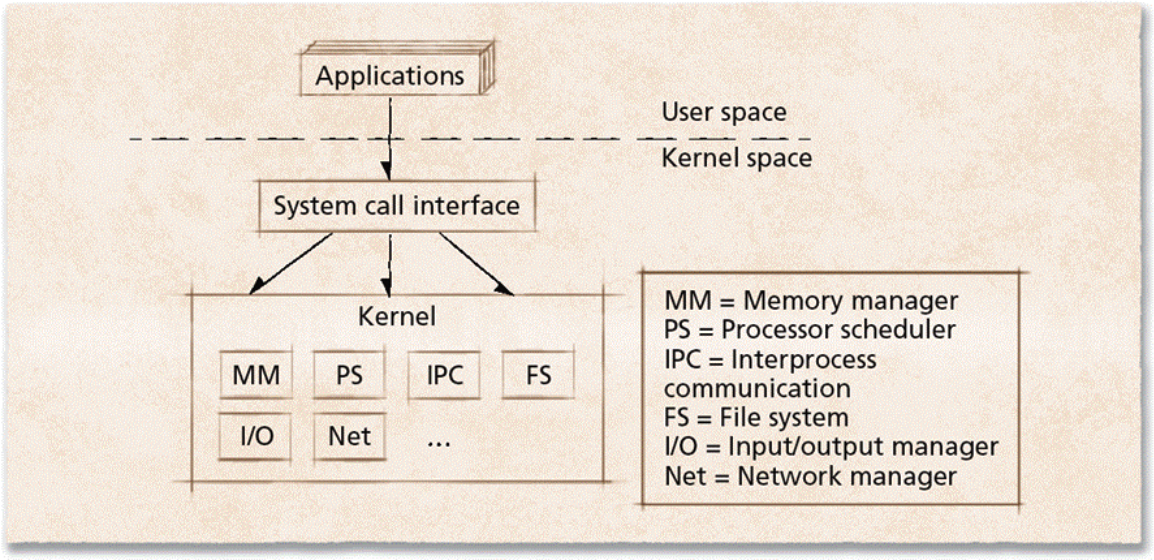


Рис. 2. Типичные компоненты ядра ОС. Здесь MM — диспетчер памяти, PS — планировщик процессов, IPC — диспетчер межпроцессного взаимодействия, FS — диспетчер файловой системы, I/O — диспетчер ввода/вывода, Net — сетевой диспетчер.

Диспетчер памяти (Memory Manager, MM) — компонент операционной системы, управляющий физической и виртуальной памятью. Определяет, когда и каким образом память распределяется между процессами и что следует предпринять, если основной памяти недостаточно.

Физическая (основная) память (main memory) — оперативная память, реально существующая в системе. Ее объем определяется возможностями оборудования компьютера. Все адреса основной памяти обычно могут напрямую адресоваться процессором.

Виртуальная память (virtual memory) — способность операционной системы предоставлять программам доступ к большему пространству адресов, чем фактически существует в оперативной памяти. Системы с виртуальной памятью позволяют облегчить труд программистов в части управления памятью, давая им возможность сосредоточиться на разработке приложений.

Планировщик процессов (Process Scheduler, PS) — компонент операционной системы, определяющий, какой процесс (или поток) будет получать доступ к процессору и на какое время.

Процесс (process) — выполняемая операционной системой программа.

Поток (thread) — независимо выполняемая последовательность программных команд (также называемая потоком выполнения, или потоком управления). Потоки упрощают выполнение параллельных действий внутри процесса.

Диспетчер межпроцессного взаимодействия (InterProcess Communication manager, IPC) — компонент операционной системы, управляющий взаимодействием между процессами.

Диспетчер файловой системы (File System manager, FS) — компонент операционной системы, управляющий размещением именованных объектов на запоминающих устройствах и предоставляющий интерфейс для доступа к данным на этих устройствах.

Диспетчер ввода/вывода (I/O manager) — компонент операционной системы, который принимает, анализирует и выполняет запросы ввода/вывода данных с аппаратных устройств.

Сетевой диспетчер (Network manager, Net) — компонент операционной системы, который обеспечивает взаимодействие компьютеров, связанных между собой посредством сети.

Драйвер устройства (device driver) — программное обеспечение, посредством которого ядро взаимодействует с аппаратурой (см. рис. 3).

Драйверы устройств

- Обычно изготавливаются и поставляются производителями устройств
- Хорошо знакомы со спецификой устройств, которыми они управляют, например, дисковые драйверы знают способ размещения данных на дисках
- Пользуются зависящим от устройств набором операций, например, для дисков это операции чтения и записи данных, открытия и закрытия накопителя

Рис. 3. Логическое расположение драйверов устройств

- Подчиняются принципу модульности, так что они могут быть установлены и удалены при изменении состава аппаратных компонентов системы
- Предоставляют пользователям возможность легко подсоединять новые типы устройств
- Обеспечивают расширяемость системы

Оболочка (shell) — приложение (как правило, текстовое или на базе графического интерфейса пользователя), позволяющее пользователю “общаться” с операционной системой.

Вопросы для самопроверки

1. Планировщик процессов определяет область памяти, в которую будет помещен новый процесс? (Да/Нет)

2. Драйвер устройства это программное обеспечение, посредством которого ядро операционной системы взаимодействует с аппаратурой? (Да/Нет)

Ответы на вопросы

1. Нет. Это определяет диспетчер памяти. Планировщик процессов распределяет между процессами процессорное время.

2. Да. Ядро операционной системы взаимодействует с аппаратурой посредством драйверов устройств.

§ 3. Свойства операционных систем

- Эффективность
- Живучесть
- Масштабируемость
- Расширяемость
- Мобильность
- Защищенность
- Интерактивность
- Практичность

Эффективная операционная система (efficient operating system) — операционная система, которая демонстрирует высокую производительность и малую длительность жизненного цикла задачи.

Производительность (throughput) — объем работы, выполненной за единицу времени.

Длительность жизненного цикла задачи (turnaround time) — время, которое проходит от подачи запроса до возвращения системой результатов его выполнения (иногда применяют термин “оборотное время”).

Живучая операционная система (robust operating system) — надежная и отказоустойчивая операционная система.

Отказоустойчивость (fault tolerance) — способность операционной системы справляться с программными или аппаратными ошибками.

Живучая ОС

- Не даст сбоя в работе из-за неожиданных программных или аппаратных ошибок
- Если отказ все-таки произойдет, он будет амортизирован
- Будет предоставлять услуги приложениям, пока аппаратные средства, необходимые для этого, не выйдут из строя

Масштабируемая операционная система (scalable operating system) — операционная система, которая может использовать ресурсы по мере их наращивания. Например, при подключении большего числа процессоров, производительность должна расти пропорционально.

Расширяемая операционная система (extensible operating system) — операционная система, которая без затруднений может быть наделена новыми возможностями.

Расширяемая ОС

- Может решать задачи изначально не предусмотренные при ее разработке
- Например, драйверы позволяют поддерживать работу устройств, не существовавших в момент разработки ОС

Мобильная операционная система (portable operating system) — операционная система, ориентированная на работу с различными конфигурациями аппаратуры. Мобильность ОС — ключевой момент для мобильности приложений.

Мобильность (portability) — возможность программного обеспечения функционировать на различных платформах.

Защищенная операционная система (secure operating system) — операционная система, препятствующая пользователям и программному обеспечению в получении несанкционированного доступа к услугам и данным.

Интерактивная операционная система (interactive operating system) — операционная система, которая позволяет приложениям мгновенно реагировать на действия пользователя.

Практичная операционная система (usable operating system) — операционная система, способная удовлетворять широкому спектру пользовательских запросов путем предоставления удобного интерфейса и поддержки множества разнообразных ориентированных на пользователя приложений.

Вопросы для самопроверки

1. Верно ли, что пользователи не могут получить доступ к данным и услугам без соответствующего разрешения благодаря живучести ОС? (Да/Нет)

2. Верно ли, что использование драйверов устройств прежде всего положительно сказывается на эффективности ОС? (Да/Нет)

Ответы на вопросы

1. Нет. Пользователи не могут получить несанкционированный доступ данным и услугам благодаря защищенности ОС.

2. Нет. Использование драйверов устройств прежде всего положительно сказывается на расширяемости ОС. Драйверы позволяют разработчикам ОС наделять ОС способностью к поддержке аппаратных средств, не существовавших в момент разработки ОС.

§ 4. Архитектура операционных систем

Типы архитектур операционных систем

- Монолитная
- Многоуровневая
- Микроядерная

- Сетевая
- Распределенная

Монолитная операционная система (monolithic operating system) — операционная система ядро которой содержит все компоненты операционной системы (см. рис. 4). Ядро в таких системах, как правило, обладает неограниченным доступом к ресурсам компьютерной системы.

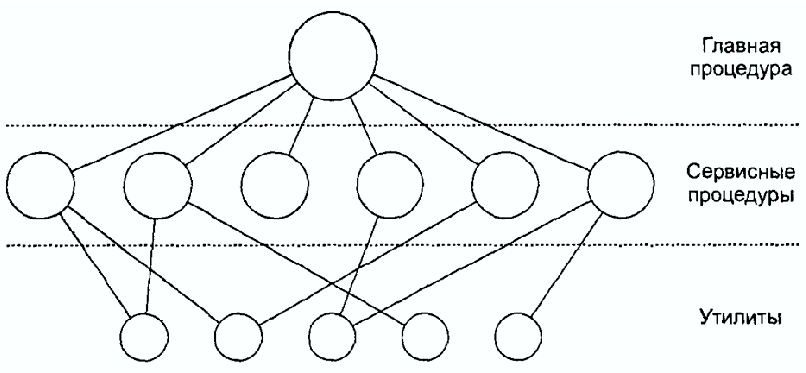


Рис. 4. Ядро монолитной ОС

Ядро монолитной ОС

- Главная процедура ядра принимает системный вызов от приложения и вызывает требуемую сервисную процедуру
- Сервисные процедуры выполняют системные вызовы
- Для каждого системного вызова имеется одна сервисная процедура
- Набор утилит обслуживает сервисные процедуры

Монолитные ОС

- OS/360, Linux
- Достоинство — эффективность вследствие малого количества вызовов, пересекающих границу пространств пользователя и ядра

- Недостаток — легко уязвима для ошибочного кода, поскольку весь код ОС работает в режиме неограниченного доступа к ресурсам компьютера

Многоуровневая (многослойная) операционная система (layered operating system) — модульная операционная система, которая располагает сходные компоненты на отдельных уровнях. Каждый уровень прибегает к услугам соседнего нижнего уровня, а результаты передает соседнему верхнему уровню.

Многоуровневые ОС

- MULTICS
- THE (Technische Hogeschool Eindhoven, см. рис. 5):
  - Э. Дейкстра, 1968 год;
  - пакетная система для голландского компьютера Electrologica X8;
  - поддерживала консоль оператора и многозадачность.

Многоуровневые ОС

- Достоинство — модульность позволяет проводить модифицирование и отладку каждого уровня отдельно от других
- Недостатки:
  - менее эффективны, чем монолитные, т.к. запрос на обслуживание должен пройти через многие уровни, прежде чем он будет удовлетворен;
  - уязвимы, как и монолитные, для ошибочного кода.

Операционная система на основе микроядра (microkernel operating system) — масштабируемая операционная система, которая включает в ядро минимальное количество служб и требует от программ пользовательского уровня реализации тех служб, которые в других типах операционных систем, как правило, делегированы ядру (см. рис. 6).

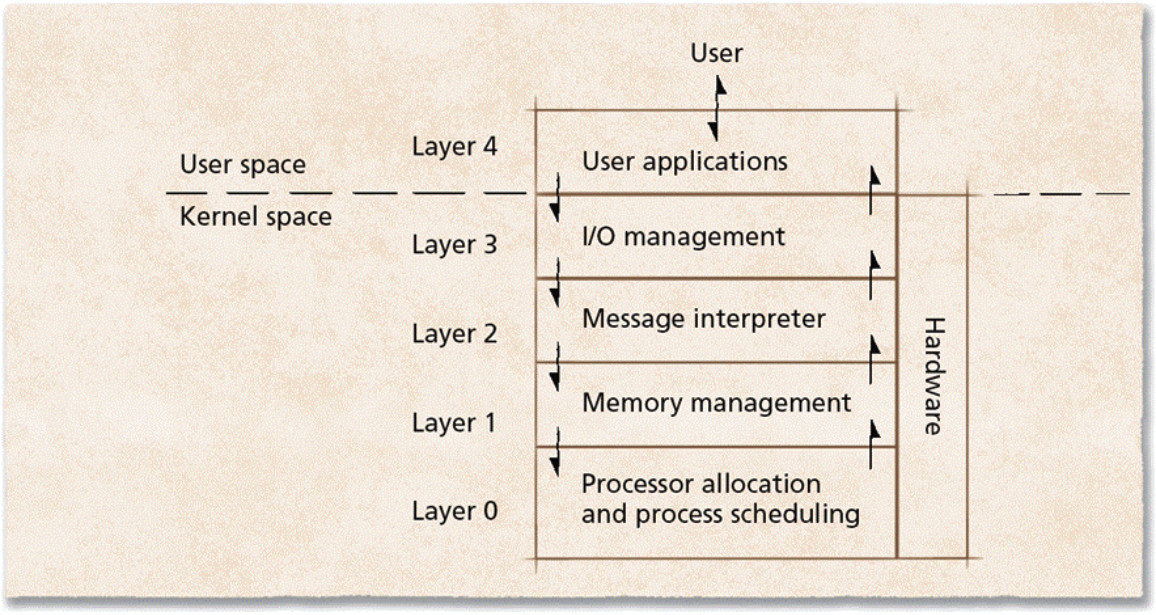


Рис. 5. Уровни операционной системы THE. Уровень 0 — распределение процессорных ресурсов и планирование процессов, 1 — управление памятью, 2 — интерпретатор сообщений, 3 — управление вводом/выводом, 4 — пользовательское приложение.

QNX

- Software Systems Limited, 1989 год
- Создана по заказу министерства обороны США
- Сетевая ОС реального времени на основе микроядра
- Может работать в сети из компьютеров на базе процессоров Intel начиная с 386
- Размер ядер разных версий: 10 Кб, 32 Кб, 64 Кб

Микроядерные ОС

- Достоинства:
  - микроядра демонстрируют высокую степень модульности, что делает их расширяемыми, мобильными и масштабируемыми;
  - надежность повышается за счет переноса части модулей в пространство пользователя.

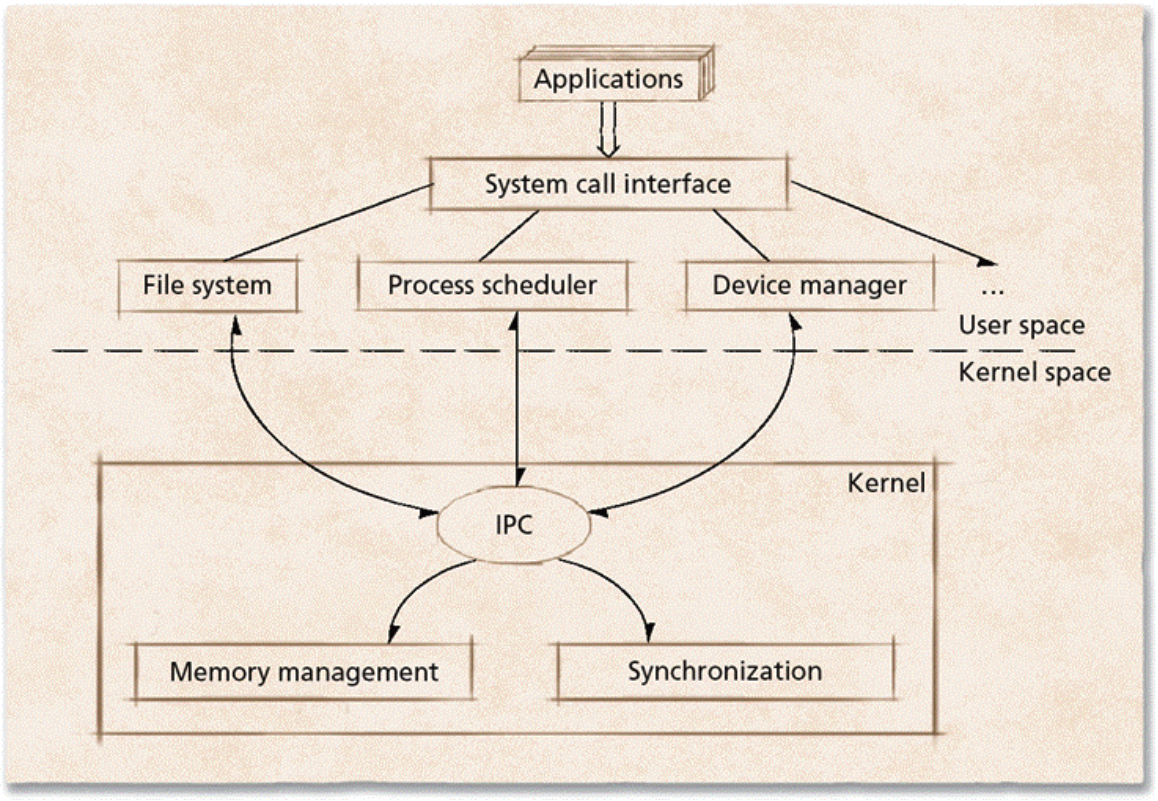


Рис. 6. Архитектура микроядерной ОС

- Недостаток — модульность достигается за счет повышения межмодульного взаимодействия, что может привести к снижению производительности

Сетевая операционная система (network operating system) — позволяет получать доступ к ресурсам на удаленных компьютерах, работающих под управлением отдельных независимых операционных систем.

Сетевые ОС

- Большинство современных ОС являются в большей или меньшей степени сетевыми
- Структура сетевых систем, как правило, основана на модели “клиент–сервер” (см. рис. 7)

Распределенная операционная система (distributed operating system) — единая операционная система, предоставляющая доступ к

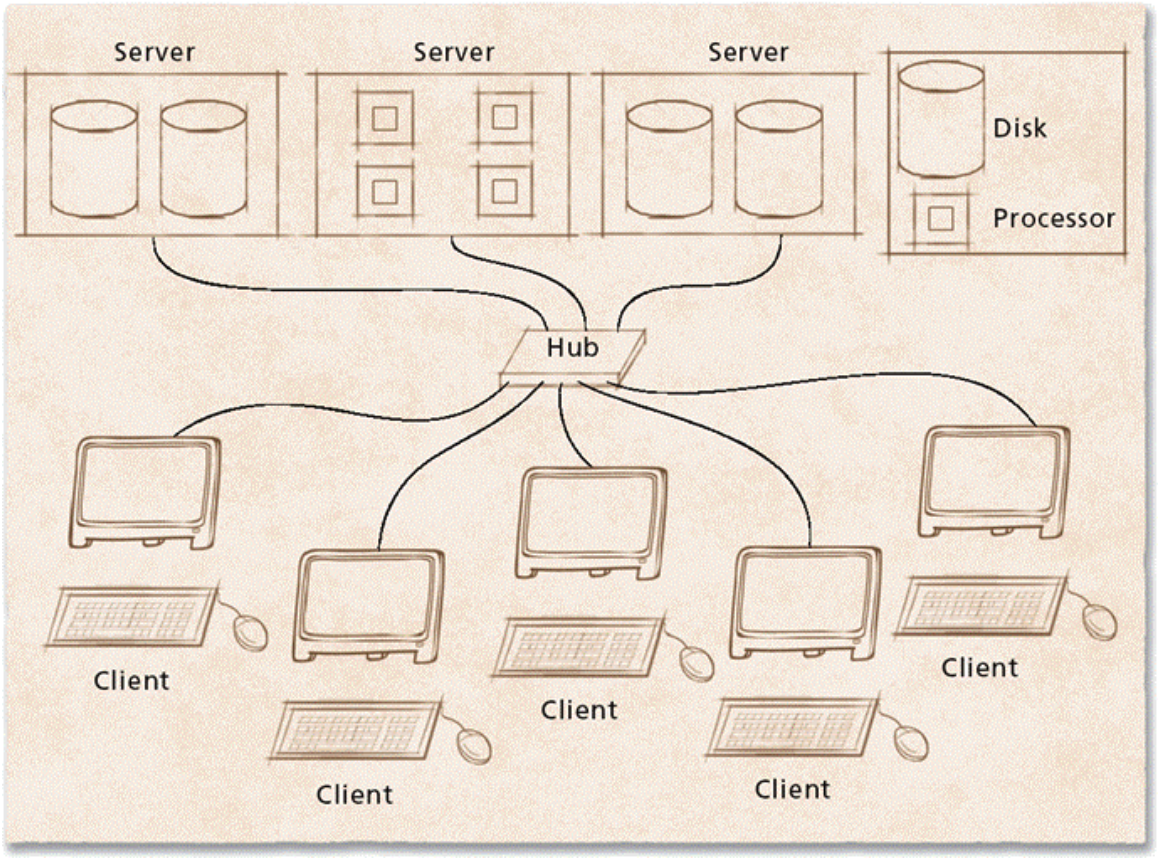


Рис. 7. Модель “клиент–сервер” сетевой ОС

ресурсам, расположенным на множестве компьютеров. При этом системные объекты обычно не знают, какие компьютеры выполняют их запросы.

Sprite

- Калифорнийский университет Беркли
- Обслуживает большое число персональных рабочих станций, объединенных локальной сетью
- Центральный сервер направляет процессы с загруженных станций на простаивающие
- Процессы не знают, какие именно станции их обслуживают

Вопросы для самопроверки

1. Верно ли, что сетевая операционная система управляет одним компьютером? (Да/Нет)

2. Способствуют ли мобильности микроядра? (Да/Нет)

Ответы на вопросы

1. Да. Сетевая ОС управляет одним компьютером, но взаимодействует с другими компьютерами сети. Распределенная ОС управляет многими компьютерами, объединенными в сеть.

2. Да. Микроядро может не зависеть от каких-либо определенных базовых аппаратных средств. Поддержка новых аппаратных средств при этом может быть обеспечена посредством загрузки нового модуля.

Глава 3

АППАРАТНЫЕ СРЕДСТВА

§ 1. Процессоры

Процессор (processor) — аппаратный компонент, выполняющий команды на машинном языке.

Центральный процессор (Central Processing Unit, CPU) — процессор, ответственный за общие вычисления в компьютере.

Сопроцессор (coprocessor) — специализированный процессор, разработанный для выполнения ограниченного набора команд специального назначения.

Сопроцессоры

- Графический сопроцессор (graphics coprocessor) — для трехмерной обработки изображений
- Цифровой процессор сигналов (Digital Signal Processor, DSP) — для преобразования цифрового сигнала в аналоговый аудисигнал

Такт (cycle) — один полный период электрического сигнала.

Тактовая частота (clock frequency) — количество тактов, генерируемых в секунду. Определяет частоту работы устройства (напр., процессора, памяти и шины). Эта величина может быть использована системой для измерения времени. Измеряется в герцах, 1 Гц = 1 такт/сек., 1 ГГц = 10^9 Гц.

Закон Мура

- Закон Мура (Moor's law) — число транзисторов в процессоре с каждым годом удваивается (см. рис. 1):

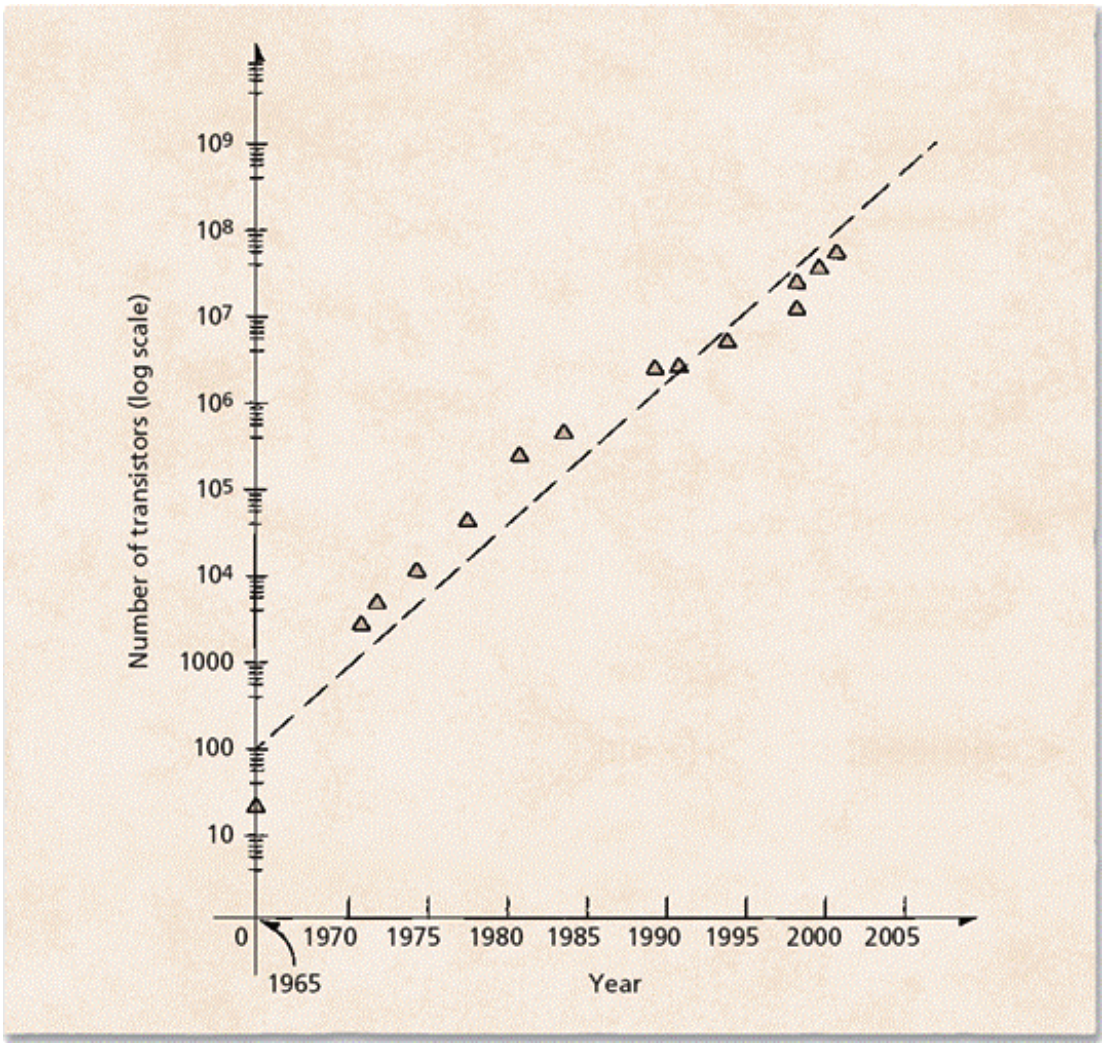


Рис. 1. Число транзисторов в процессорах Intel

  - "Cramming More Components onto Integrated Circuits", 1965 год;
  - Гордон Мур, почетный председатель совета директоров, соучредитель корпорации Intel.

- В 2000-х годах рост тактовой частоты отстает от экспоненциального роста числа транзисторов в процессорах

Компоненты процессора (см. рис. 2)

- Арифметико-логическое устройство (Arithmetic and Logic Unit, ALU) — компонент процессора, который выполняет основные арифметические и логические операции

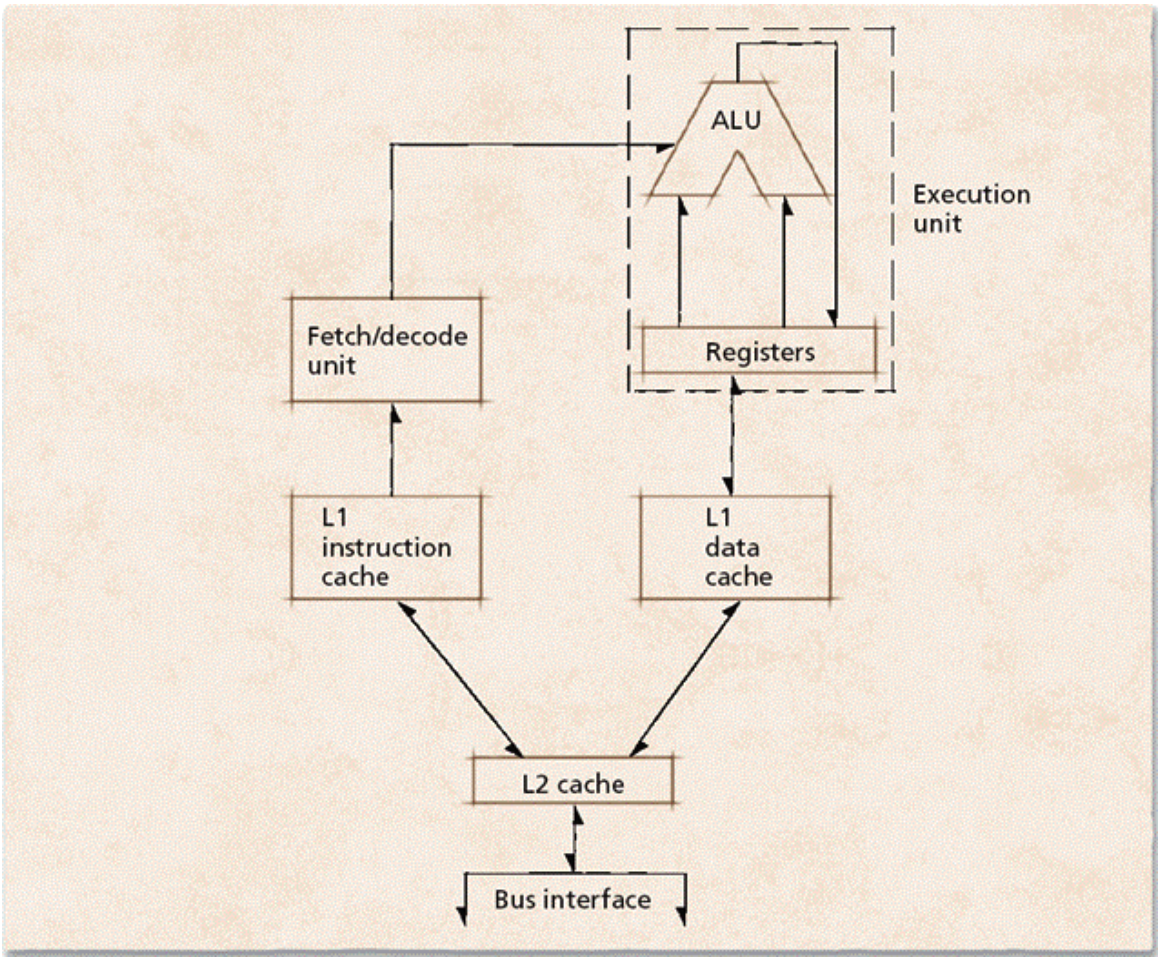


Рис. 2. Компоненты процессора

- Регистры (registers) — быстрая память, расположенная в процессоре, в которой хранятся данные, непосредственно обрабатываемые процессором
- Операционный блок (execution unit) — компонент процессора, объединяющий арифметико-логическое устройство и регистры
- Блок выборки команд (instruction fetch unit) — компонент процессора, который загружает команды из кэша команд, чтобы они могли быть дешифрованы и выполнены
- Дешифратор команд (instruction decode unit) — интерпретирует команды, осуществляет ввод необходимых данных в операционный блок и генерирует управляющие сигналы, заставляя процессор выполнять команды

- Ядро (kernel) — компонент процессора, объединяющий операционный блок, дешифратор команд и блок выборки команд
- Кэш-память (cache memory) — быстродействующая, дорогая и малая по объему память, в которой хранятся копии данных и команд для ускорения доступа к ним
- Кэш-память первого уровня (L1) — самые быстрые дорогие и наименьшие по объему блоки кэш-памяти, расположенные в процессоре
- Кэш-память второго уровня (L2) — менее быстрые и дорогие, большие по объему блоки кэш-памяти, расположенные либо в процессоре, либо на материнской плате

Регистры процессора

- Регистры общего назначения
- Управляющие регистры:
  - счетчик команд;
  - указатель стека;
  - слово состояния процессора;
  - ...

Регистры общего назначения (general-purpose registers) — регистры, которые могут быть использованы процессором для хранения данных и значений указателей пользовательских процессов.

Регистры общего назначения

- К регистрам общего назначения имеют доступ процессы пользователя
- Intel Pentium имеет 16 регистров общего назначения по 32 бита в каждом регистре
- К управляющим регистрам имеют доступ только компоненты операционной системы

Счетчик команд (Program Counter, PC) — управляющий регистр, содержащий адрес следующей, стоящей в очереди на выполнение команды. После того, как команда выбрана из памяти, регистр команд корректируется и указатель переходит к следующей команде.

Указатель стека (Stack Pointer) — управляющий регистр, содержащий адрес вершины стека в памяти.

Стек (stack) — область памяти, содержащая по одной области данных для каждой процедуры, которая уже начала выполняться, но еще не закончена. В стековой области данных процедуры хранятся ее входные параметры, локальные и временные переменные, не хранящиеся в регистрах.

Слово состояния процессора (Processor Status Word, PSW) — управляющий регистр, содержащий бит режима работы процессора: пользовательский режим, или режим ядра.

Пользовательский режим (user mode) — режим работы процессора, в котором доступно лишь ограниченное число из системы команд. В этом режиме процессам не позволяется обращаться непосредственно к системным ресурсам.

Система команд (instruction set) — набор машинных команд, которые процессор способен выполнить.

Режим ядра (kernel mode) — режим работы процессора, в котором доступны все команды, в том числе привилегированные. Они выполняют операции, обладающие доступом к защищенным системным ресурсам (напр., переключение процессора с процесса на процесс, или обращение к дисковому накопителю).

Вопросы для самопроверки

1. Может ли процессор выполнить одну команду за один такт? (Да/Нет)

2. Верно ли, что кэш-память первого уровня имеет меньший объем, чем кэш-память второго уровня? (Да/Нет)

Ответы на вопросы

1. Нет. Это обусловлено архитектурой процессора. Кроме того, команды после дешифрования часто представляют собой последовательность инструкций.

2. Да. Кэш-память первого уровня расположена ближе к ядру, более быстрая и дорогая, но меньшая по объему, чем кэш-память второго уровня.

§ 2. Методы повышения производительности процессоров

- Конвейерная обработка данных
- Суперскалярная архитектура
- Многоядерная архитектура

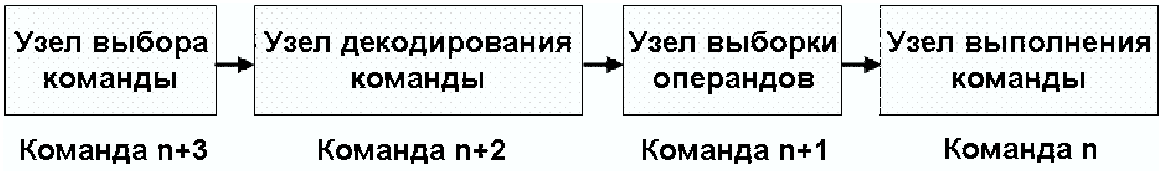


Рис. 3. Схема работы процессора с конвейерной обработкой данных

Конвейерная обработка данных (см. рис. 3)

- IBM 7030, начало 1960-х годов
- Одновременно выполняются инструкции нескольких команд
- В течение одного такта на каждом узле выполняется одна инструкция
- С началом каждого такта инструкции передвигаются к следующему узлу

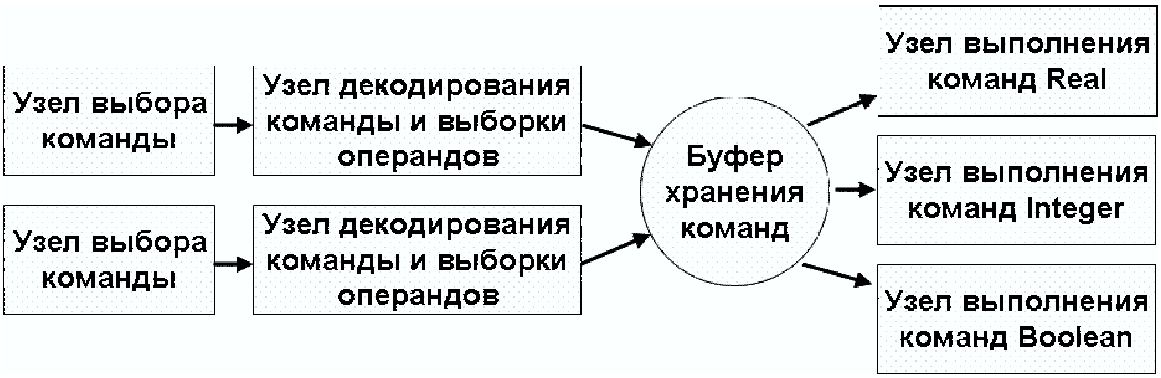


Рис. 4. Схема работы процессора с суперскалярной архитектурой

Суперскалярная архитектура (см. рис. 4)

- Intel Pentium
- За один такт считывается несколько команд
- Специализированные выполняющие узлы для различных операций
- Когда узел выполнения освобождается, он считывает из буфера команду, которую он может выполнить и выполняет ее

Многоядерная архитектура (см. рис. 5)

- Intel Core
- Множество вычислительных ядер на одном процессорном кристалле
- Параллельно может выполнятся несколько потоков команд

Вопросы для самопроверки

1. Конвейерная обработка данных впервые была реализована в процессорах Intel Pentium? (Да/Нет)

2. Должна ли быть выполнена считанная команда процессором с конвейерной обработкой данных, если в предыдущей команде был принят условный переход? (Да/Нет)

3. Гарантирует ли многоядерная архитектура процессора увеличение производительности? (Да/Нет)

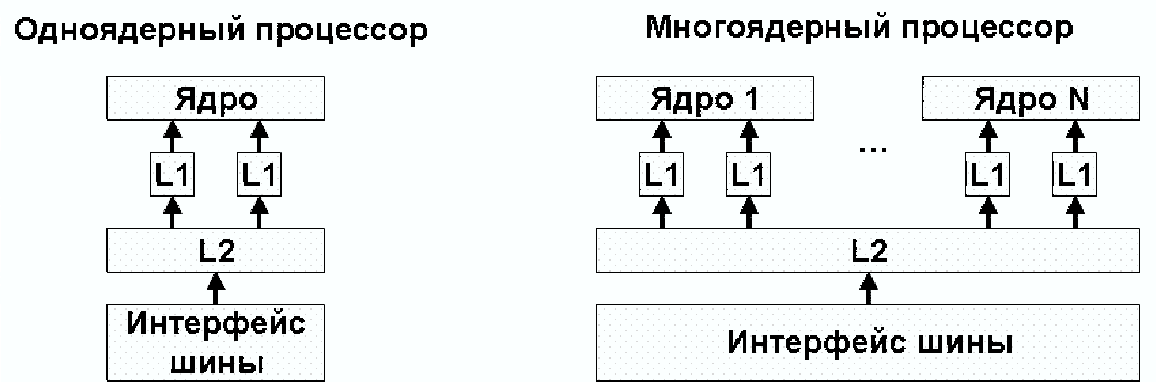


Рис. 5. Схема работы одноядерного (слева) и многоядерного (справа) процессоров

Ответы на вопросы

1. Нет. Конвейерная обработка данных впервые была реализована в начале 1960-х в ЭВМ IBM 7030.

2. Да. Это основная проблема, которую должна устранять операционная система для компьютера с конвейерной обработкой данных.

3. Нет. Многоядерная архитектура процессора обеспечивает увеличение производительности системы, только в случае, если программное обеспечение реализует параллельное выполнение вычислительных потоков.

§ 3. Память

Основная память (main memory) — энергозависимая память, в которой хранятся команды и данные; это самая медленная память в системной иерархии, к которой процессор может обращаться непосредственно (см. рис. 6).

Память с произвольной выборкой (Random Access Memory, RAM) — память, доступ к содержимому которой может осуществляться в произвольном порядке.

Пример. Кэш память первого и второго уровня, основная память.

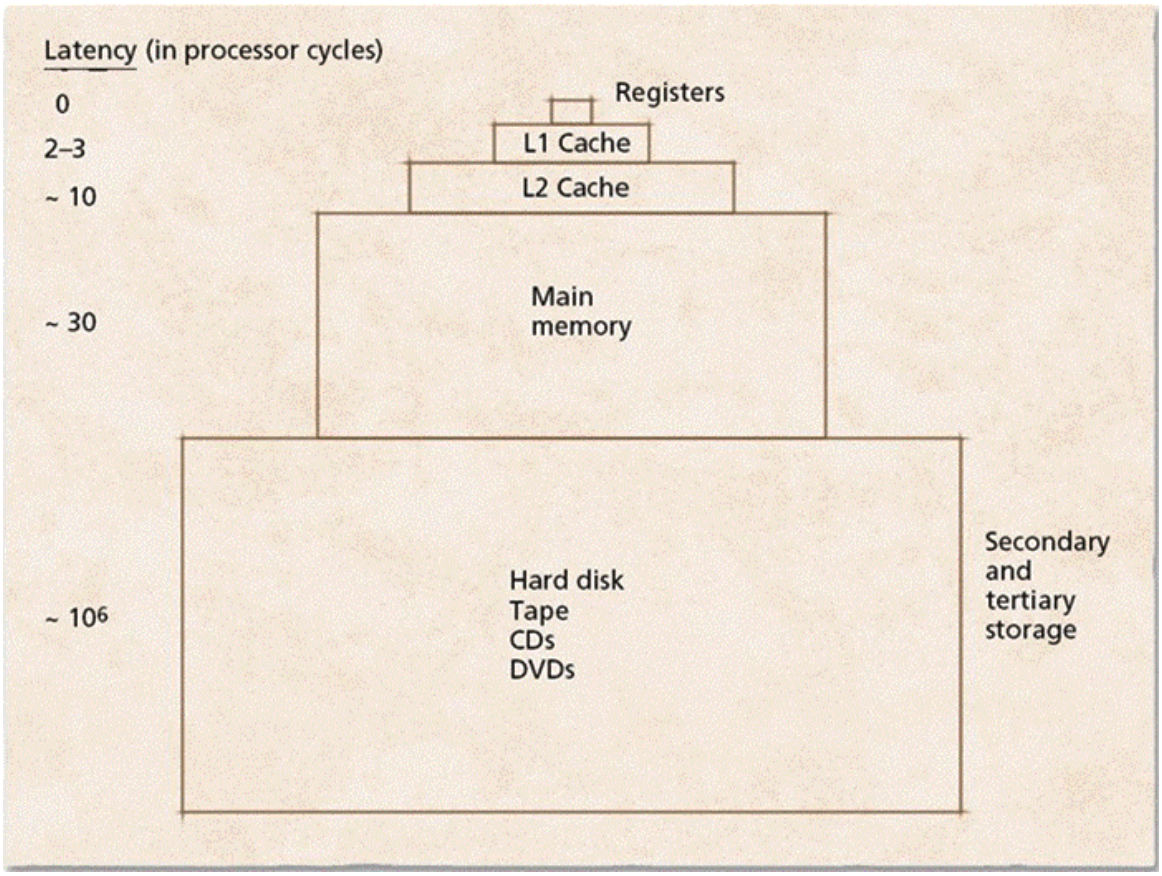


Рис. 6. Иерархия памяти. Слева указана задержка доступа к памяти в тактах

Вторичное запоминающее устройство (secondary storage) — память, которая, как правило, используется для долговременного хранения больших объемов данных и программ (см. рис. 6).

Пример. Дисковый накопитель, CD, DVD.

Устройство блочного ввода/вывода (block device) — устройство, которое передает данные группами байтов (как правило, от сотни байт до десятков килобайт).

Пример. Дисковый накопитель, CD, DVD.

Третичное запоминающее устройство (tertiary storage) — память, которая, как правило, используется для архивирования данных и программ (см. рис. 6).

Пример. Накопитель на магнитной ленте.

Защита памяти (memory protection) — механизм, препятствующий процессам в получении доступа к основной памяти, используемой другими процессами, или операционной системой.

Ограничительный регистр (bounds register) — управляющий регистр центрального процессора, который хранит информацию о диапазоне адресов, доступных активному процессу.

Виртуальная память (virtual memory) — концепция, позволяющая решить проблему ограниченной емкости основной памяти за счет предоставления каждому процессу виртуального адресного пространства (большего, чем основная память) для хранения данных и исполняемых инструкций.

Пример. Файл обмена (swap file) на дисковом накопителе.

Физический адрес (real address) — адрес ячейки в оперативной памяти.

Виртуальный адрес (virtual address) — адрес, по которому процесс обращается к системе виртуальной памяти; виртуальные адреса динамически преобразуются в физические в ходе выполнения программ.

Устройство управления памятью (Memory Management Unit, MMU) — специализированное аппаратное устройство, выполняющее, в частности, трансляцию виртуальных адресов в физические.

Вопросы для самопроверки

1. Верно ли, что схема иерархии памяти имеет вид пирамиды? (Да/Нет)

2. Используют ли процессы пользователя физические адреса? (Да/Нет)

Ответы на вопросы

1. Да. Если запоминающее устройство дешевле, пользователь может позволить себе купить такое устройство большей емкости, следовательно емкость памяти увеличивается.

2. Нет. Процессы используют виртуальные адреса. Трансляцию виртуальных адресов в физические осуществляет устройство управления памятью (MMU).

§ 4. Прямой доступ к памяти

Прерывание (interrupt) — аппаратный сигнал, сообщающий о наступлении определенного события.

Программируемый ввод/вывод (Programmed I/O, PIO) — реализация ввода/вывода для устройств, которые не поддерживают прерывания и в которых передача каждого слова в память (или обратно) должна контролироваться процессором. PIO применялся в ранних системах.

Прямой доступ к памяти (Direct Memory Access, DMA) — механизм передачи данных с внешнего устройства в основную память (или обратно) посредством контроллера ввода/вывода, требующий только прерывания процессора по окончании передачи данных.

Схема работы прямого доступа к памяти (см. рис. 7):

1) процессор посылает запрос ввода/вывода контроллеру ввода/вывода, который в свою очередь посылает запрос диску; процессор продолжает выполнять инструкции;

2) диск передает данные контроллеру ввода/вывода; данные размещаются в ячейке памяти с адресом, указанным командой прямого доступа к памяти;

3) диск посылает процессору прерывание для уведомления его о завершении выполнения операции ввода/вывода.

Вопросы для самопроверки

1. Обладает ли прямой доступ к памяти преимуществом перед программируемым вводом/выводом? (Да/Нет)

2. Используются ли прерывания для реализации программируемого ввода/вывода? (Да/Нет)

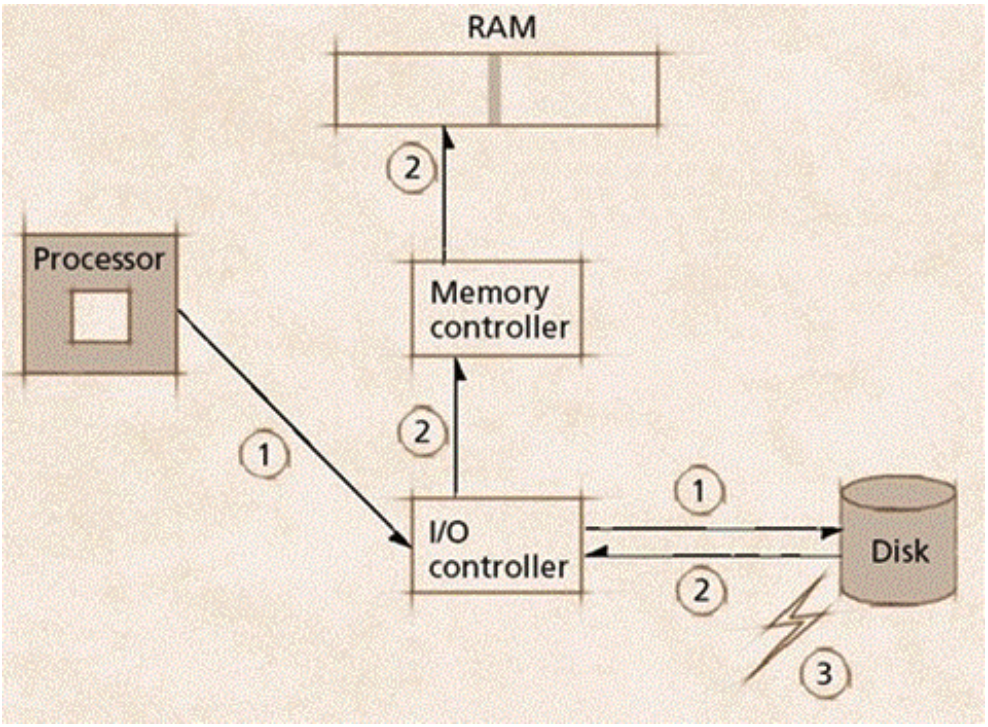


Рис. 7. Схема работы прямого доступа к памяти

Ответы на вопросы

1. Да. В системах, использующих PIO, процессор ожидает в состоянии простоя завершения каждой операции обмена данными внешних устройств с памятью. DMA позволяет процессору выполнять программные инструкции во время операции ввода/вывода.

2. Нет. Прерывания используются для реализации прямого доступа к памяти. Диск посылает процессору прерывание для уведомления его о завершении выполнения операции ввода/вывода.

§ 5. Начальная загрузка

Базовая система ввода/вывода, БИОС (Basic Input/Output System, BIOS) — программные средства низкого уровня, которые контролируют инициализацию операционной системы и управление основными аппаратными средствами.

Загрузочный сектор (boot sector) — определенное место на диске, где хранятся начальные команды (загрузчик) операционной систе-

мы; БИОС выдает указания аппаратным средствам загружать эти начальные команды при включении компьютера.

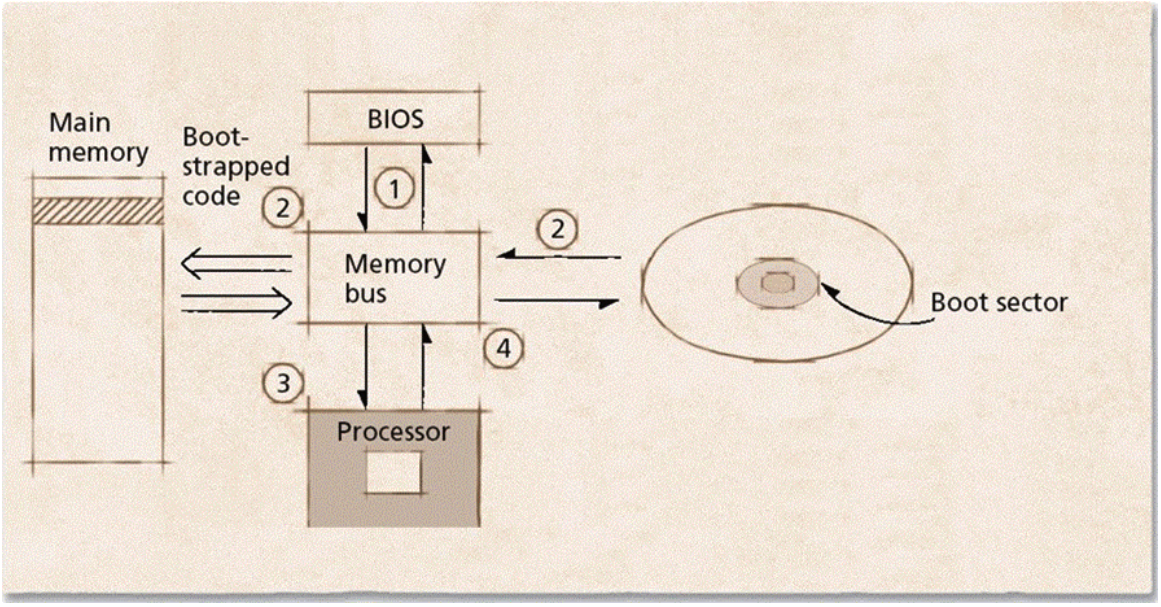


Рис. 8. Схема начальной загрузки

Схема начальной загрузки (см. рис. 8):

1) БИОС собирает информацию об аппаратных средствах и приводит систему в исходное состояние;
2) БИОС загружает загрузчик операционной системы (bootstrapped code) из загрузочного сектора в основную память;
3) процессор выполняет код начальной загрузки;
4) загрузчик операционной системы загружает операционную систему с диска в основную память.

Технология Plug-and-Play

- Упрощает установку драйверов и настройку аппаратных средств при наличии PnP BIOS
- PnP устройства могут определять необходимый драйвер и позволять ОС использовать его для настройки устройства

- PnP устройство можно подключать к работающему компьютеру и немедленно использовать

Вопросы для самопроверки

1. Должна ли ОС препятствовать процессам пользователя в получении доступа к загрузочному сектору? (Да/Нет)

2. Является ли BIOS частью операционной системы? (Да/Нет)

Ответы на вопросы

1. Да. Если бы пользователь был наделен возможностью доступа к загрузочному сектору, то он мог бы изменить код операционной системы, что привело бы систему в негодность.

2. Нет. BIOS записывается в постоянное запоминающее устройство до инсталляции на компьютер операционной системы и может обеспечивать работу различных операционных систем.

§ 6. Шины

Шина (bus) — совокупность проводников, образующих высокоскоростной канал связи для обмена данными между различными устройствами на материнской плате (см. рис. 9, 10).

Контроллер (controller) — аппаратный компонент, который управляет доступом устройства к шине (см. рис. 9).

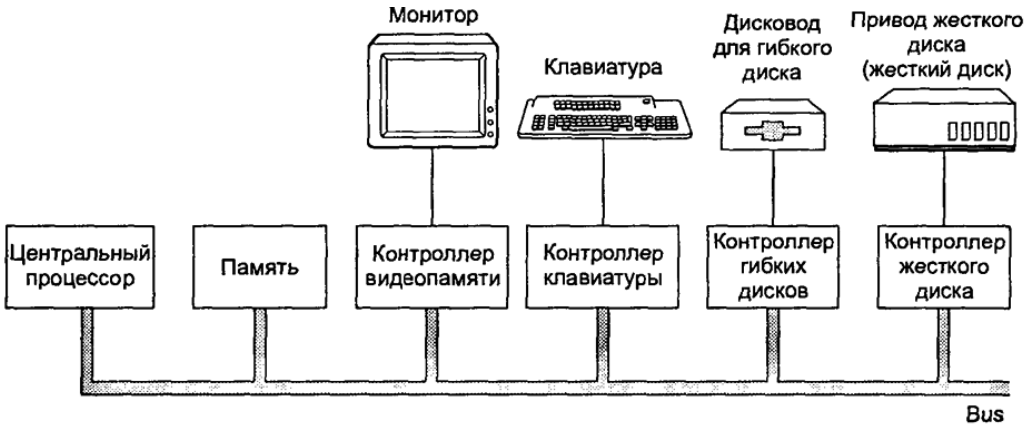


Рис. 9. Некоторые компоненты персонального компьютера

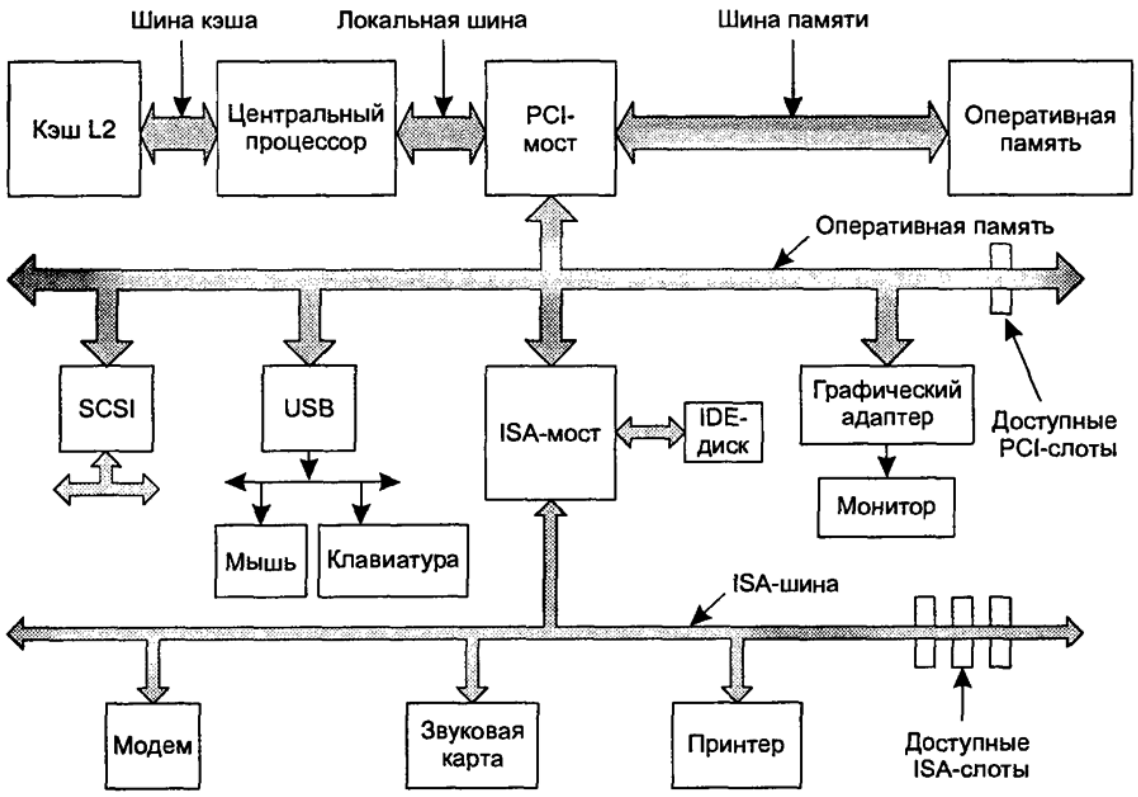


Рис. 10. Структура системы Pentium

Вопросы для самопроверки

1. Влияет ли разрядность шины на производительность системы? (Да/Нет)

2. Способствовало ли использование шины и контроллеров реализации принципа открытой архитектуры IBM PC? (Да/Нет)

Ответы на вопросы

1. Да. Разрядность шины определяет объем данных, которым могут обмениваться основная память и процессор за один такт. Если процессор генерирует запросы на большее количество данных, чем можно передать за один такт, то это приводит к задержке работы процессора.

2. Да. Открытая архитектура обеспечивает легкую замену и установку новых устройств IBM PC, в том числе, благодаря использованию шины и контроллеров.

Часть II

Процессы и потоки

Глава 4

КОНЦЕПЦИИ ПРОЦЕССА

§ 1. Определение процесса

Процесс

- Впервые термин процесс для описания программы в ходе ее выполнения был использован в системе MULTICS (вторая половина 1960-х годов)
- Иногда заменяется термином “задача” (task) и наоборот

Процесс (process) — логический объект, описывающий программу в стадии ее выполнения.

Адресное пространство процесса

- Область команд
- Область данных
- Область стека

Область команд (text region) — часть адресного пространства процесса, содержащая инструкции, предназначенные для выполнения процессором.

Область данных (data region) — часть адресного пространства процесса, содержащая данные (а не инструкции). Эта область доступна для изменений.

Область стека (stack region) — область адресного пространства процесса, содержащая инструкции и данные для открытых процедур. Размер стека увеличивается в случае вызова вложенных процедур и уменьшается по их завершении.

Вопросы для самопроверки

1. Правда ли, что термины “процесс” и “программа” являются синонимами? (Да/Нет)

2. Правда ли, что адресное пространство процесса поделено на непересекающиеся области? (Да/Нет)

Ответы на вопросы

1. Нет. Процессом называется программа в стадии выполнения, тогда как сама программа представляет собой неодушевленный логический объект.

2. Да. Каждая область адресного пространства предназначена для хранения данных, схожих по способу доступа (доступ запрещен, открыт, по принципу стека). Память поделена на области для того, чтобы система могла обеспечить соблюдение этих правил.

§ 2. Состояния процесса

Основные состояния процесса

- Состояние выполнения
- Состояние блокировки
- Состояние готовности

Состояние выполнения (running state) — состояние процесса, если ему в данный момент выделен процессор.

Состояние блокировки (blocked state) — состояние процесса, в котором он ожидает наступления определенного события, например, завершения операции ввода/вывода. Процесс в этом состоянии не может использовать процессор, даже если он свободен.

Состояние готовности (ready state) — состояние процесса, в котором он сразу мог бы использовать процессор, предоставленный в его распоряжение.

Список готовых к выполнению процессов (ready list) — структура данных ядра, в которой в упорядоченном виде хранятся

указатели на все готовые к выполнению процессы. Сортировка списка готовых к выполнению процессов обычно осуществляется в соответствии с их приоритетом.

Список заблокированных процессов (blocked list) — структура данных ядра, содержащая указатели на все заблокированные процессы. Данный список не упорядочен по приоритету.

Вопросы для самопроверки

1. Правда ли, что в каждый конкретный момент времени в системе может выполняться только один процесс? (Да/Нет)

2. Верно ли, что список заблокированных процессов хранится в упорядоченном виде? (Да/Нет)

Ответы на вопросы

1. Нет. В многопроцессорной системе в каждый конкретный момент времени может выполняться столько процессов, сколько процессоров установлено в системе.

2. Нет. Список заблокированных процессов хранится в неупорядоченном виде, так как никакого приоритетного порядка разблокировки не предусматривается — она осуществляется в том порядке, в котором происходят ожидаемые процессами события.

§ 3. Переходы процесса из состояния в состояние

Диспетчер (dispatcher) — компонент операционной системы, отбирающий для запуска на процессоре первый процесс из списка готовых к выполнению процессов. Является частью планировщика процессов.

Квант (quantum) — промежуток времени, в течение которого процессор остается выделенным одному процессу. Использование квантов позволяет предотвратить монопольный захват процессора одним процессом.

Таймер прерываний (interrupting clock) — аппаратно реализованный таймер, вырабатывающий сигналы прерывания через определенные промежутки времени (называемые квантами) для того, чтобы не допустить монопольного захвата процессора одним процессом.

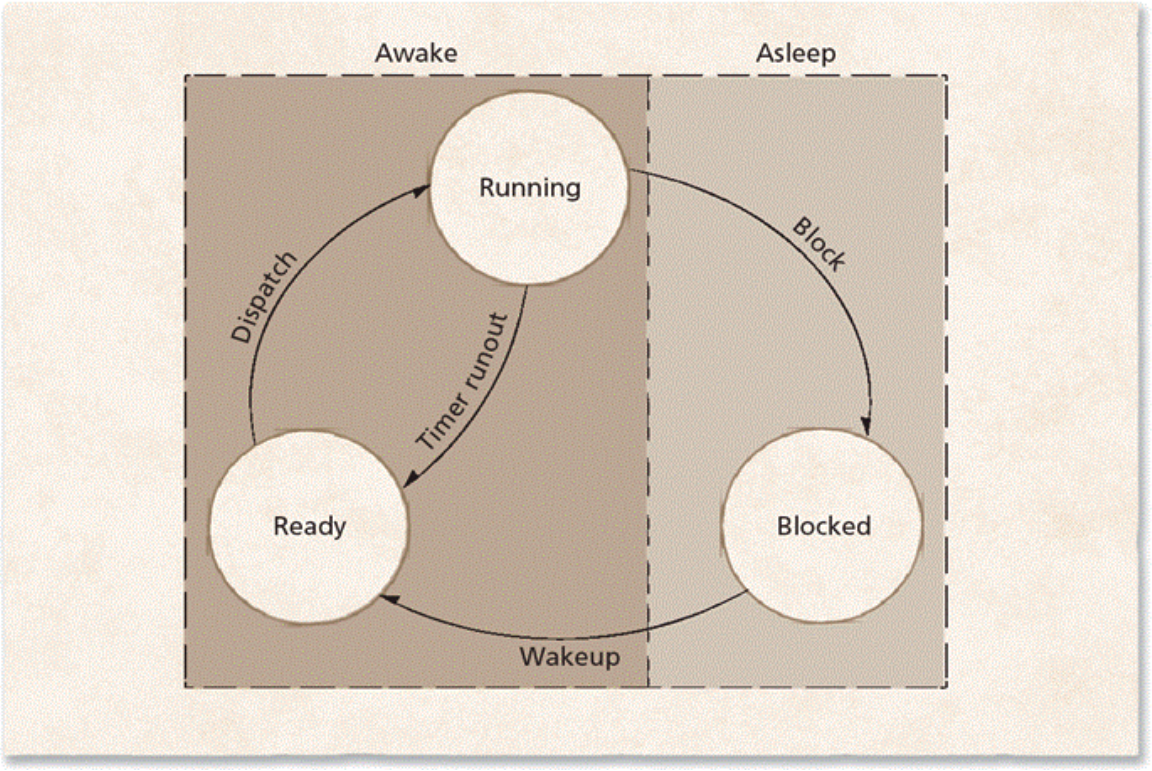


Рис. 1. Диаграмма переходов процесса из состояния в состояние. Awake — бодрствование; бодрствующие процессы постоянно соревнуются за процессорное время. Asleep — спячка; спящий процесс не сможет использовать процессор, даже если тот снова станет доступен.

Переходы процесса из состояния в состояние (см. рис. 1)

- Истечение кванта (timer run out) — по истечении кванта планировщик процессов переводит процесс в состояние готовности из состояния выполнения
- Блокирование (block) — процесс блокирует сам себя; если до истечения выделенного ему кванта времени он начнет операцию ввода/вывода, то добровольно освободит процессор
- Пробуждение (wake up) — по завершении операции ввода/вывода, ожидаемом процессом, планировщик процессов перемещает указатель на данный процесс из списка заблокированных в список готовых к выполнению процессов

Вопросы для самопроверки

1. Верно ли, что все переходы между состояниями процесса осуществляет планировщик процессов? (Да/Нет)

2. Возможен ли переход процесса, из состояния готовности в состояние блокировки? (Да/Нет)

3. Возможен ли переход процесса, из состояния блокировки в состояние выполнения? (Да/Нет)

Ответы на вопросы

1. Нет. Планировщик процессов осуществляет запуск, пробуждение и перевод процесса из состояния выполнения в состояние готовности по истечении кванта времени. Блокирование осуществляется самим процессом.

2. Нет. Процесс не может заблокировать себя из состояния готовности, так как для того чтобы сгенерировать запрос ввода/вывода, он должен выполняться.

3. Нет. По завершении операции ввода/вывода процесс не может быть сразу направлен на выполнение, так как в это время процессор может быть занят более приоритетным процессом.

§ 4. Блоки управления процессами (дескрипторы процессов)

Таблица процессов (process table) — структура данных, в которой хранятся указатели на все процессы системы.

Идентификационный номер процесса (Process Identification Number, PID) — числовое значение, уникальным образом идентифицирующее процесс.

Блок управления процессом (Process Control Block, PCB) — структура данных, содержащая информацию о процессе (состояние, адресное пространство и пр.). Другое название — дескриптор процесса.

Обычное содержимое PCB (см. рис. 2)

- Программный счетчик (program counter)

- Контекст выполнения (registers)
- Состояние процесса (state)
- Приоритет процесса (priority)
- Указатели на адресное пространство (address space)
- Указатель на родительский процесс (parent)
- Указатели на дочерние процессы (children)
- Указатели на открытые файлы (open files)
- ...

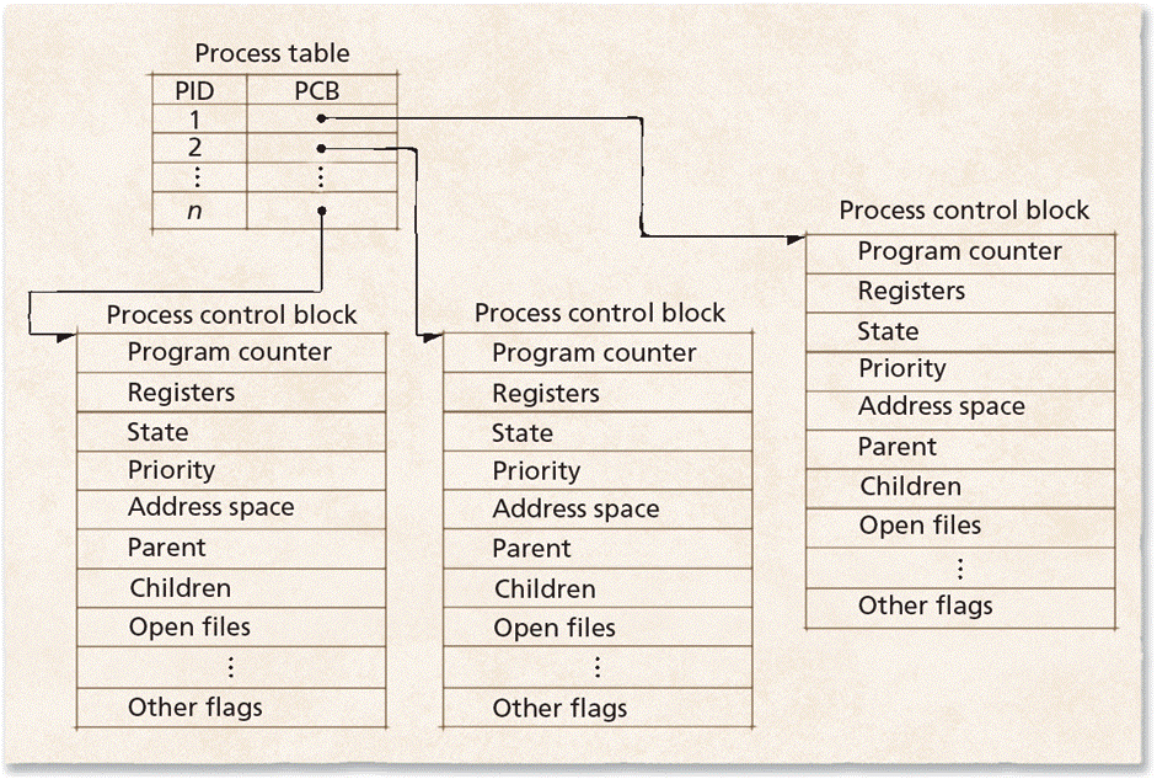


Рис. 2. Таблица процессов и блоки управления процессами

Программный счетчик (program counter) — указатель на следующую инструкцию процесса, которая должна быть выполнена процессором. После выполнения этой инструкции, значение программно-

го счетчика корректируется таким образом, чтобы тот ссылался на следующую инструкцию процесса.

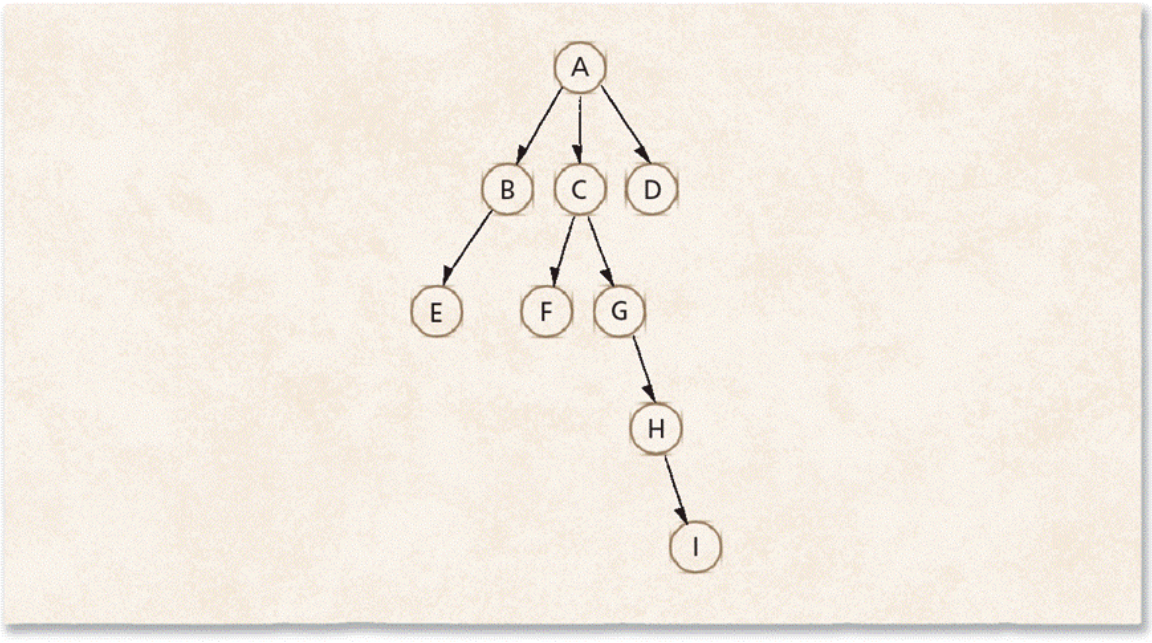


Рис. 3. Иерархия создания процессов

Контекст выполнения (registers, execution context) — содержимое регистров общего назначения и некоторых управляющих регистров перед выходом процесса из состояния выполнения. Контекст выполнения зависит от архитектуры системы. Благодаря этим данным операционная система может восстанавливать значения регистров в случае возвращения процесса в состояние выполнения.

Состояние процесса (process state) — статус процесса (выполняется, готов, блокирован).

Приоритет процесса (process priority) — значение, определяющее важность данного процесса по сравнению с другими процессами.

Адресное пространство (address space) — области памяти (команд, данных и стека), с которыми может работать процесс.

Родительский процесс (parent process) — процесс, породивший один или несколько дочерних процессов (см. рис. 3).

Дочерний процесс (child process) — новый процесс, созданный родительским процессом. Дочерний процесс располагается на один уровень ниже родительского в иерархии создания процессов (см. рис. 3).

Вопросы для самопроверки

1. Правда ли, что таблица процессов нужна для их упорядочивания? (Да/Нет)

2. Правда ли, что структура блока управления процессом зависит от реализации операционной системы? (Да/Нет)

3. Правда ли, что процесс может вообще не иметь родительского процесса? (Да/Нет)

Ответы на вопросы

1. Нет. Таблица процессов позволяет находить блоки управления процессами.

2. Да. Структура блока управления процессом определяется архитектурой вычислительной системы и существенно зависит от реализации операционной системы.

3. Да. Первый создаваемый операционной системой процесс, например, init в Unix не имеет родительского процесса.

§ 5. Переключение контекста

Переключение контекста (context switching) — выполняемая операционной системой операция отбора процессора у одного процесса и его выделения другому процессу. При этом ОС обязана сохранить состояние прерываемого процесса. Аналогично, система должна восстановить состояние процесса, выбираемого для запуска из списка готовых к выполнению.

Переключение контекста (см. рис. 4):

1) до начала переключения контекста процесс P_1 выполняется на процессоре;

2) после поступления сигнала прерывания ядро принимает решение о выборе нового процесса и инициирует переключение контекста;

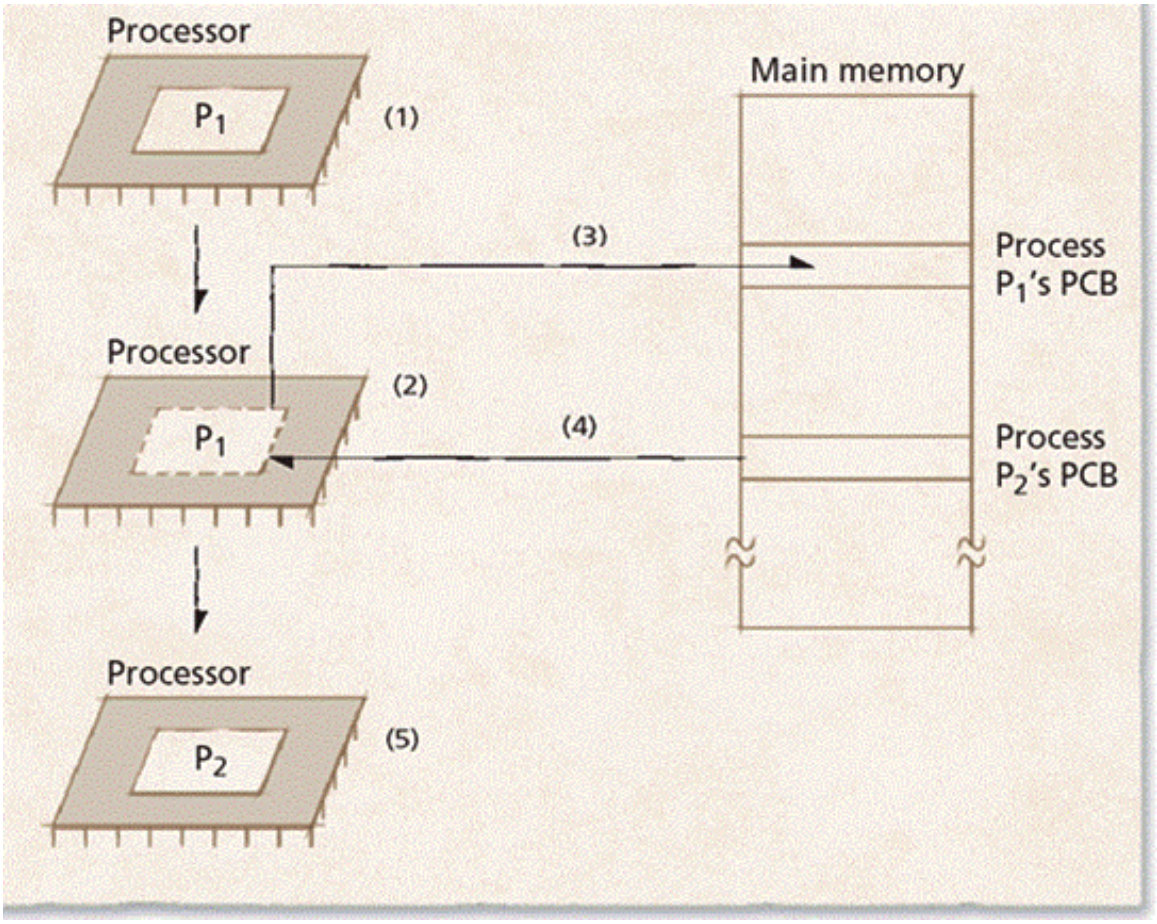


Рис. 4. Переключение контекста.

3) ядро сохраняет контекст выполнения процесса P_1 в его PCB в оперативной памяти;

4) ядро загружает контекст выполнения процесса P_2 из его PCB из оперативной памяти;

5) после переключения контекста процесс P_2 выполняется на процессоре.

Вопросы для самопроверки

1. Верно ли, что во время переключения контекста процессор не может выполнять инструкции от имени процессов? (Да/Нет)

2. Верно ли, что операционная система сохраняет контекст прерванного процесса в таблице процессов? (Да/Нет)

Ответы на вопросы

1. Да. Во время переключения контекста процессор не может выполнять "полезные вычисления". Поэтому операционная система должна минимизировать время, затрачиваемое на переключение контекста.

2. Нет. Во время переключения контекста ОС сохраняет контекст прерванного процесса в его блоке управления процессом (PCB).

§ 6. Прерывания

Прерывание (interrupt) — аппаратный сигнал, сообщающий о наступлении определенного события. Прерывания заставляют процессор выполнять последовательность программных инструкций, называемых обработчиком прерываний.

Обработчик прерываний (interrupt handler) — код ядра, выполняемый в ответ на прерывание.

Вектор прерываний (interrupt vector) — защищенный массив в памяти, в котором хранятся указатели на местоположение обработчиков прерываний.

Контроллер прерываний (interrupt controller) — микросхема на материнской плате либо аппаратный компонент процессора, сортирующий прерывания в соответствии с их приоритетом, обеспечивая первоочередную обработку высокоприоритетных прерываний.

Обработка прерывания таймера прерываний (см. рис. 5):

1) процесс P_1 выполняется на процессоре до поступления сигнала прерывания;

2) таймер прерываний по истечении кванта времени генерирует прерывание, чтобы исключить захват процессора процессом P_1;

3) процессор обращается к записи вектора прерываний, соответствующей прерываниям таймера прерываний;

4) процессор сохраняет контекст выполнения процесса P_1 в его PCB в оперативной памяти;

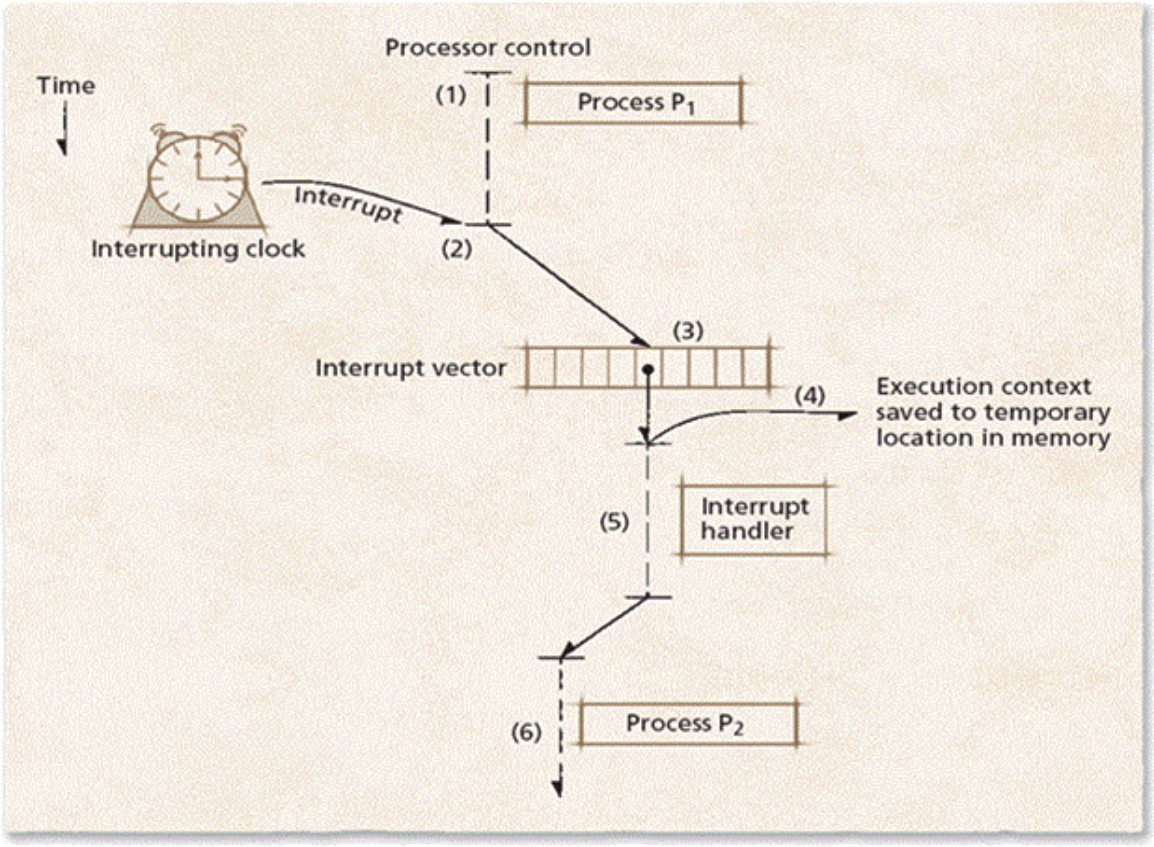


Рис. 5. Обработка прерывания таймера прерываний

5) процессор выполняет обработчик прерываний, тот вызывает планировщик процессов, который загружает контекст выполнения процесса P_2 из его PCB из оперативной памяти;

6) процесс P_2 выполняется на процессоре после загрузки его контекста выполнения.

Вопросы для самопроверки

1. Верно ли, что при запуске обработчика прерываний контекст выполнения прерванного процесса сохраняется в памяти? (Да/Нет)

2. Может ли прерванный процесс продолжить свое выполнение сразу после обработки прерывания таймера прерываний? (Да/Нет)

Ответы на вопросы

1. Да. Если не сохранить в памяти контекст выполнения процес-

са, обработчик прерываний может перезаписать значения регистров, используемых процессом.

2. Да. Планировщик процессов может загрузить контекст выполнения прерванного процесса, и он продолжит свое выполнение, если в списке готовых к выполнению процессов он окажется первым.

§ 7. Классы прерываний

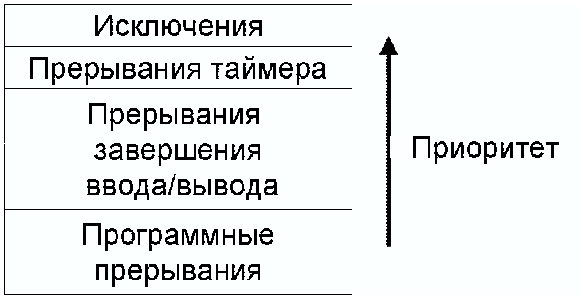


Рис. 6. Классы прерываний, упорядоченные по приоритетам

Исключение (exception) — аппаратный сигнал, вызванный ошибкой. Исключения делятся на

- аварийные завершения;
- ловушки;
- промахи.

Аварийное завершение (abort) — операция досрочного прекращения выполнения процесса, например, в результате аппаратного сбоя. Невосстановимая ошибка процесса. Имеет наивысший приоритет.

Ловушка (trap) — исключение, возникающее, например, в результате ошибок переполнения (когда значение, которое должно быть сохранено в регистре, превышает его разрядность). Ловушка перезапускает процесс, начиная с инструкции следующей за той, которая вызвала исключение.

Промах (fault) — вид исключения, вызванного, например, запрещенной операцией доступа к памяти. Некоторые промахи могут быть исправлены с помощью соответствующих обработчиков исключений (например, страничный промах). Тогда возобновление работы процесса начинается с инструкции, вызвавшей промах.

Прерывания таймера прерываний (interval timer interrupt) — сигнал, вызываемый таймером прерываний по истечении выделенного процессу кванта времени.

Прерывания завершения ввода/вывода (I/O completion interrupt) — сигнал, выдаваемый устройством по окончании обслуживания запроса ввода/вывода.

Программные прерывания (software-generated interrupt) — командные инструкции, использующиеся процессами пользователя для вызова системных функций. Переключают процессор из пользовательского режима в режим ядра.

Маскирование прерываний (mask interrupt) — при маскировании определенного типа прерываний, прерывания указанного типа не доставляются процессу, замаскировавшему их. В этом случае прерывания либо устанавливаются в очередь для последующей доставки, либо удаляются процессором.

Вопросы для самопроверки

1. Верно ли, что программные прерывания имеют самый низкий приоритет? (Да/Нет)

2. Верно ли, что ядро системы всегда обязано реагировать на прерывания, даже если сильно перегружено? (Да/Нет)

Ответы на вопросы

1. Да. Сигналы, поступающие от оборудования, приоритетнее вызова процессами пользователя системных функций, так как если оборудование занято или работает не правильно, эти вызовы могут быть не обслужены.

2. Нет. Большинство процессоров позволяют ядру маскировать прерывания определенных типов. Тогда процессор просто игнорирует

прерывания данного типа либо сохраняет их в очереди отложенных прерываний.

§ 8. Взаимодействие процессов сигналами

Сигналы (signals) — программные прерывания, уведомляющие процесс о наступлении определенного события либо возникновении ошибки.

Сигналы

- Не позволяют передавать данные
- При поступлении сигнала операционная система определяет:
  - кому предназначен данный сигнал;
  - как этот процесс должен на него реагировать.

Варианты реакции процесса на сигнал

- Перехват
- Игнорирование
- Маскирование

Перехват (catch) — процесс определяет процедуру, вызываемую операционной системой в случае поступления сигнала.

Игнорирование (ignore) — процесс перекладывает ответственность за выполнение действия по умолчанию по обработке сигнала на операционную систему. Чаще всего — это аварийное завершение процесса либо приостановка его выполнения.

Маскирование (masking) — процесс маскирует сигнал определенного типа (например, сигнал приостановки) и операционная система блокирует сигналы данного типа до тех пор, пока маскирование не будет отключено.

Вопросы для самопроверки

1. Позволяют ли сигналы передавать данные между процессами? (Да/Нет)

2. Верно ли, что процесс может отреагировать на сигнал одним из трех возможных способов? (Да/Нет)

Ответы на вопросы

1. Нет. Это основной недостаток использования сигналов при взаимодействии процессов.

2. Да. Процесс может перехватывать, игнорировать либо маскировать сигналы определенного типа.

§ 9. Взаимодействие процессов путем передачи сообщений

Передача сообщений (massage passing) — механизм, позволяющий процессам общаться путем обмена данными. Прием и отправка сообщений обычно реализуется в виде вызова системных функций вида

```
send(reciverProcess, message)
receive(senderProcess, message)
```

Передача сообщений

- Блокирующая
- Неблокирующая

Блокирующая передача (blocking send) — процесс вынужден ожидать до тех пор, пока сообщение не будет доставлено получателю, требуя подтверждения приема.

Неблокирующая передача (nonblocking send) — процесс отправитель может продолжить выполнение других операций даже если сообщение еще не было доставлено получателю.

Пример. Отсылка процессом данных на занятый сервер печати.

Канал (pipe) — механизм передачи сообщений, формирующий прямой поток данных между двумя процессами. Реализуется в виде

защищенной операционной системой области памяти, которая выступает в качестве буфера.

Вопросы для самопроверки

1. Верно ли, что в распределенных системах вместо сигналов обычно используется технология передачи сообщений? (Да/Нет)

2. Может ли процесс, выполнивший блокирующую передачу, не получить подтверждения приема сообщения? (Да/Нет)

Ответы на вопросы

1. Да. Как правило, типы передаваемых сигналов зависят от архитектуры системы, что чревато несовместимостью сигналов разных компьютеров. Кроме того, сигналы не позволяют компьютерам обмениваться данными, без чего не может обойтись большинство распределенных систем.

2. Да. Он может не получить подтверждения о доставке, что приведет к бесконечному блокированию процесса. Для решения этой проблемы используется механизм тайм-аута — если подтверждение не поступит через определенный промежуток времени, передача сообщения будет повторена.

Глава 5

КОНЦЕПЦИИ ПОТОКА

§ 1. Определение потока

Поток (thread) — логический объект, описывающий последовательность независимо выполняемых программных инструкций внутри процесса. Потоки позволяют воспользоваться преимуществами параллельного выполнения операций в рамках процесса. Каждый процесс имеет как минимум один поток выполнения.

Многопоточность (multithreading) — технология, позволяющая включать в состав процесса несколько работающих потоков для выполнения параллельных операций, возможно даже одновременно (для многопроцессорных / многоядерных систем).

Элементы процесса

Совместно используемые всеми потоками процесса	Индивидуальные для каждого потока процесса
Адресное пространство Родительский процесс Дочерние процессы Открытые файлы Необработанные аварийные сигналы Сигналы и их обработчики Информация об использовании ресурсов	Состояние (готов, выполняется, блокирован) Программный счетчик Контекст выполнения Стек процедур потока

Мотивы использования потоков

- Архитектура системы программирования обеспечивает написание фрагментов кода, которые должны выполняться параллельно

- Производительность многопроцессорных / многоядерных систем
- Взаимодействие потоков через общее адресное пространство

Вопросы для самопроверки

1. Верно ли, что создание потока требует меньшего числа тактов, чем создание процесса? (Да/Нет)

2. Верно ли, что взаимодействие процессов более эффективно, чем взаимодействие потоков одного процесса? (Да/Нет)

3. Могут ли многопоточные приложения выполняться быстрее однопоточных? (Да/Нет)

Ответы на вопросы

1. Да. Многие элементы процесса совместно используются всеми его потоками. По этому при создании нового потока они уже существуют.

2. Нет. Потоки, принадлежащие одному процессу, могут обмениваться друг с другом данными с помощью общего адресного пространства, обходясь без механизма взаимодействия процессов, подразумевающего обращение к ядру.

3. Да. Большинство приложений содержат фрагменты кода, которые могут выполняться независимо от остальной части приложения. Если выделить эти фрагменты в отдельные потоки, можно будет выполнять их отдельно на разных процессорах / ядрах.

§ 2. Асинхронное параллельное выполнение

Асинхронные параллельные потоки (asynchronous concurrent threads) — потоки, которые существуют в системе одновременно и выполняются независимо друг от друга, но периодически должны синхронизироваться и взаимодействовать.

Критический участок (critical section, критическая область) — фрагмент кода, выполняющий операции над разделяемым ресурсом (например, запись значения в разделяемую переменную). Чтобы добиться корректной работы программы, в своем критическом участке в любой момент времени должен находиться только один поток.

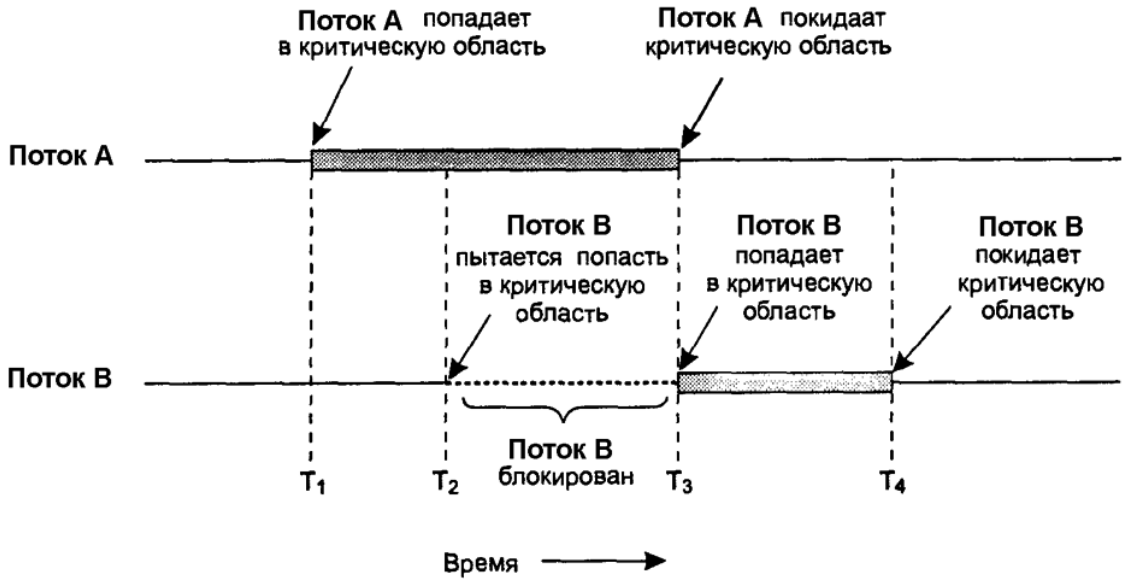


Рис. 1. Взаимоисключение с использованием критических участков

Взаимоисключение (mutual exclusion) — ограничение, в соответствии с которым выполнение одного потока внутри своего критического участка исключает выполнение других потоков внутри своих критических участков (см. рис. 1). Взаимоисключение жизненно важно для обеспечения корректной работы нескольких потоков, обращающихся к одним и тем же разделяемым данным для их модификации.

Пример: необходимость взаимоисключения. Пусть потоки A и B — счетчики, использующие общую глобальную переменную count. Рассмотрим следующую последовательность событий:

1) поток A считывает в регистр R значение переменной `count`=n: R=n;
2) поток A увеличивает в регистре R значение на единицу: R=n+1;
3) по истечении кванта времени процессор передается потоку B;
4) поток B увеличивает значение переменной count на единицу: count=n+1;
5) процессор возвращается потоку A;

6) поток A сохраняет значение n+1 из регистра R в переменную count: count=n+1.

В результате, переменная count имеет значение n+1, а при корректной работе потоков A и B, должно быть: count=n+2.

Вопросы для самопроверки

1. Верно ли, что синхронизироваться и взаимодействовать должны только асинхронные параллельные потоки? (Да/Нет)

2. Необходимо ли реализовывать взаимоисключение, если потоки считывают значение разделяемой переменной? (Да/Нет)

Ответы на вопросы

1. Нет. Синхронизироваться и взаимодействовать должны также асинхронные параллельные процессы.

2. Нет. Взаимоисключение необходимо реализовывать только в случае, если потоки модифицируют разделяемую переменную.

§ 3. Семафоры

Двоичный семафор (binary semaphore) — абстракция, используемая при реализации взаимоисключения, в которой применяются две атомарные операции (P и V) для доступа к защищенной переменной S, определяющей, могут ли потоки входить в свои критические участки (S=1 — могут, S=0 — нет).

Защищенная переменная S (protected variable S) — бинарное значение S, в котором хранится состояние семафора. Опросить либо изменить это значение можно только с помощью атомарных операций P и V. При создании семафора S присваивается значение 1.

Атомарная операция (atomic operation) — операция непрерывно выполняемая от начала до конца.

Операция P (операция ожидания) — одна из двух операций, позволяющих менять значение семафора S. Если S=0, то операция P блокирует вызывающий поток. Если S>0, то операция P уменьшит

значение S на единицу и позволит вызывающему потоку продолжить работу.

Операция V (операция оповещения) — одна из двух операций, позволяющих менять значение семафора S. Если у данного семафора есть заблокированные потоки, операция V будит один из них и увеличивает значение S на единицу, если заблокированных потоков нет, то просто увеличивает значение S на единицу.

Пример: фрагмент кода потока с использованием двоичного семафора

```
Некритический участок
P(S)
Критический участок
V(S)
Некритический участок
```

Считающий семафор (counting semaphore) — семафор, в котором переменная S целочисленная и может принимать значения больше 1. Считающие семафоры применяются, когда необходимо выделить ресурс из пула идентичных ресурсов.

Считающий семафор

- При создании считающего семафора, переменной S присваивается значение числа n ресурсов в пуле
- Каждая операция P уменьшает значение S на единицу, показывая, что некоторому потоку выделен один ресурс из пула
- Каждая операция V увеличивает значение S на единицу, показывая, что поток возвратил один ресурс в пул

Вопросы для самопроверки

1. Можно ли реализовать бинарный семафор на основе считающего семафора? (Да/Нет)

2. Верно ли, что в любой момент времени поток может находиться в очереди ожидания только одного семафора? (Да/Нет)

3. Верно ли, что операция V считающего семафора всегда увеличивает значение этого семафора на единицу? (Да/Нет)

Ответы на вопросы

1. Да. Для этого достаточно задать в качестве начального значения переменной S считающего семафора единицу.

2. Да. При помещении в очередь ожидания семафора поток блокируется, будучи не в состоянии выполнить программный код, из-за которого он мог бы оказаться в очереди ожидания другого семафора.

3. Нет. Если один и более потоков находятся в состоянии ожидания, операция V разбудит один из них, не увеличивая значения счетчика, так как разбуженному потоку будет выделен один освободившийся ресурс.

§ 4. Мониторы

Монитор (monitor) — конструкция параллельного программирования, которая содержит как данные, так и процедуры, необходимые для управления взаимоисключением при распределении общего ресурса или пула идентичных ресурсов.

Монитор

- Потоки, обращающиеся к монитору, не знают какие данные находятся внутри монитора и не имеют к ним доступа
- В каждый момент времени в мониторе может находиться только один поток

Переменная–условие (condition–variable) — переменная, которой соответствует очередь потоков, ожидающих входа в монитор, в случае, если распределяемый ресурс занят (см. рис. 2).

Переменная–условие

- Если потоку необходимо дождаться переменной–условия в тот момент, когда он находится внутри монитора, он выходит из монитора и попадает в очередь ожидания переменной–условия
- Потоки пребывают в этой очереди до тех пор, пока не получат оповещения от других потоков

`wait(conditionVariable)` — процедура монитора, которую поток использует в случае, если ресурс занят; выдав команду ожидания поток выходит из монитора и попадает в очередь (см. рис. 2).

`signal(conditionVariable)` — процедура монитора, используя которую поток оповещает другие потоки о том, что ресурс свободен и выходит из монитора; первый поток, ожидающий в очереди, получив сигнал, может выйти из очереди и войти в монитор (см. рис. 2).

```
1  // Resource allocator monitor
2
3  // monitor initialization (performed only once)
4  boolean inUse = false; // simple state variable
5  Condition available; // condition variable
6
7  // request resource
8  monitorEntry void getResource()
9  {
10    if ( inUse ) // is resource in use?
11    {
12       wait( available ); // wait until available is signaled
13    } // end if
14
15    inUse = true; // indicate resource is now in use
16
17 } // end getResource
18
19 // return resource
20 monitorEntry void returnResource()
21 {
22    inUse = false; // indicate resource is not in use
23    signal( available ); // signal a waiting thread to proceed
24
25 } // end returnResource
```

Рис. 2. Простейший монитор на псевдокоде. Здесь `getResource` — аналог операции ожидания P, `returnResource` — аналог операции оповещения V

Вопросы для самопроверки

1. Можно ли реализовать двоичный семафор с помощью монитора? (Да/Нет)

2. Верно ли, что каждый монитор имеет ровно одну переменную–условие? (Да/Нет)

3. Верно ли, что для переменных–условий мониторов используются очереди с приоритетами? (Да/Нет)

Ответы на вопросы

1. Да. Пример простейшего монитора показывает, как это можно сделать (см. рис. 2).

2. Нет. Монитор может иметь отдельные переменные–условия для каждой отдельной ситуации, которая может привести к вызову команды ожидания в мониторе.

3. Нет. Поток с низким приоритетом может оказаться в ситуации бесконечного откладывания из-за множества высокоприоритетных потоков, вызывающих функцию ожидания монитора при входе в приоритетную очередь.

Глава 6

ПЛАНИРОВАНИЕ РАБОТЫ ПРОЦЕССОРА

§ 1. Уровни планирования

Планирование на верхнем уровне (high-level scheduling) — определение заданий, которые могут быть допущены в систему активного участия в соревновании за системные ресурсы (см. рис. 1).

Планирование на верхнем уровне

- Используется в крупных мэйнфреймах, предназначенных для пакетной обработки данных
- Определяет степень многозадачности

Степень многозадачности (degree of multiprogramming) — общее количество процессов в системе в конкретный момент времени.

Планирование на промежуточном уровне (intermediate-level scheduling) — на этом уровне планирования определяется, какие процессы будут допущены к планировщику на нижнем уровне (см. рис. 1).

Планирование на промежуточном уровне

- В большинстве современных систем планировщик верхнего уровня отсутствует, и запуск заданий осуществляется планировщиком промежуточного уровня
- В роли планировщика промежуточного уровня выступает менеджер памяти

Планирование на нижнем уровне (low-level scheduling) — определение процесса, который должен получить управление процессором (см. рис. 1). Осуществляется планировщиком процессов.

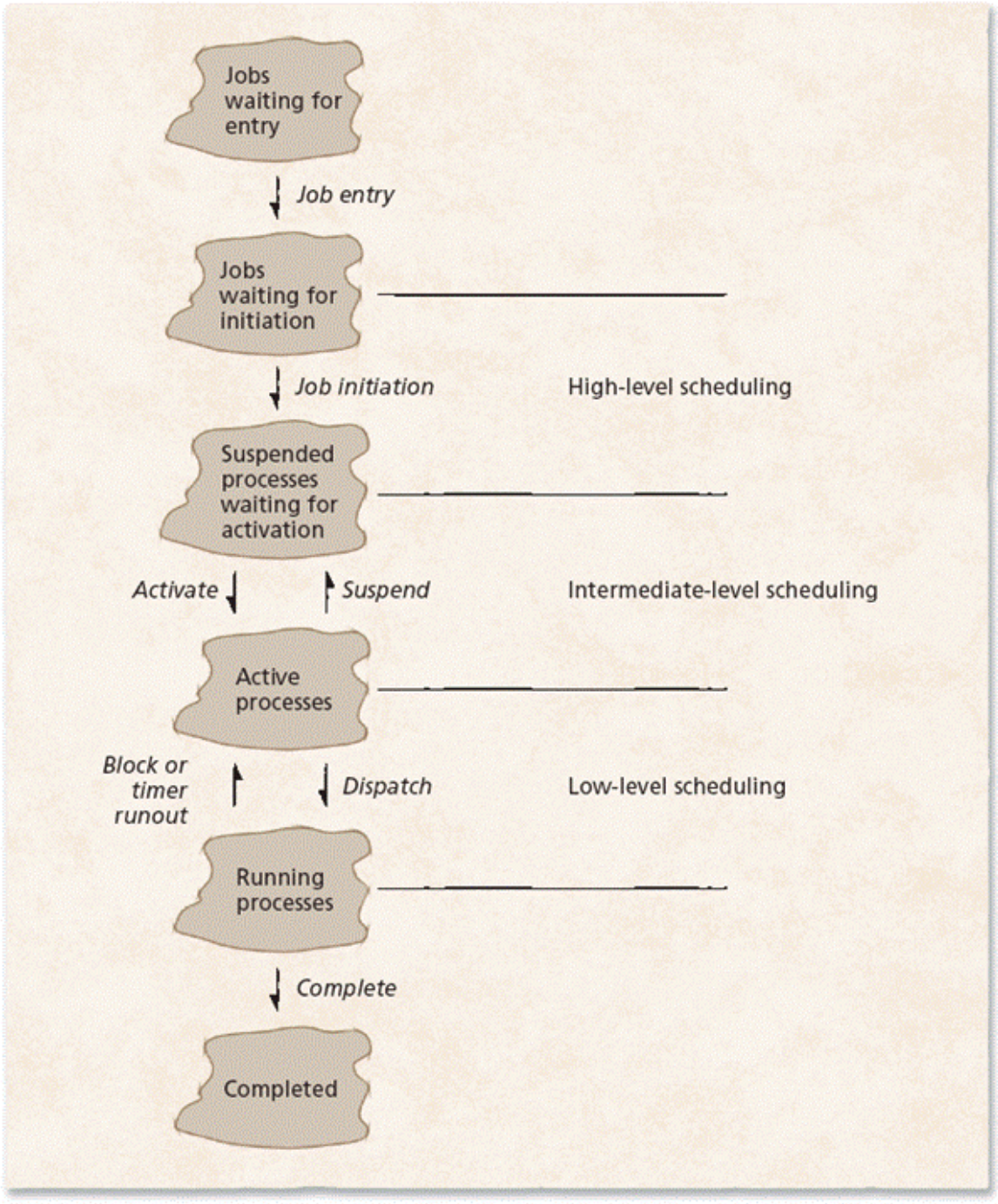


Рис. 1. Уровни планирования работы процессора. Здесь jobs waiting for entry — задания, ожидающие вода; jobs waiting for invitation — задания, ожидающие запуска; suspended processes waiting for activation — приостановленные процессы, ожидающие активации; active processes — активные процессы; running processes — выполняющиеся процессы; completed — завершенные процессы.

Вопросы для самопроверки

1. Может ли менеджер памяти запрещать процессам доступ к планировщику процессов? (Да/Нет)

2. Должен ли планировщик процессов оставаться резидентным в основной памяти? (Да/Нет)

Ответы на вопросы

1. Да. Планировщик промежуточного уровня может запрещать процессам доступ к планировщику нижнего уровня, если система перегружена.

2. Да. Планировщик нижнего уровня должен оставаться резидентным в основной памяти, поскольку код планировщика нужно выполнять достаточно часто, чтобы оперативно реагировать для уменьшения накладных расходов, связанных с планированием выполнения процессов.

§ 2. Планирование с приостановкой процессов

Активный процесс (active process) — это процесс, находящийся в оперативной памяти и имеющий возможность вести борьбу за процессорное время. Он может находиться в одном из активных состояний (выполняется, готов, блокирован).

Приостановленный процесс (suspended process) — это процесс, находящийся на диске, на некоторое время выбывший из борьбы за процессорное время, но не прекративший своего существования.

Приостановка процессов (suspend) — операция, осуществляемая менеджером памяти. Он выгружает процессы на диск, если система перегружена либо их в памяти слишком много и все они в ней не помещаются (см. рис. 2).

Активация процессов (activate) — операция, осуществляемая менеджером памяти. Он периодически просматривает процессы, находящиеся на диске, чтобы решить, какой из них переместить в память (см. рис. 2).

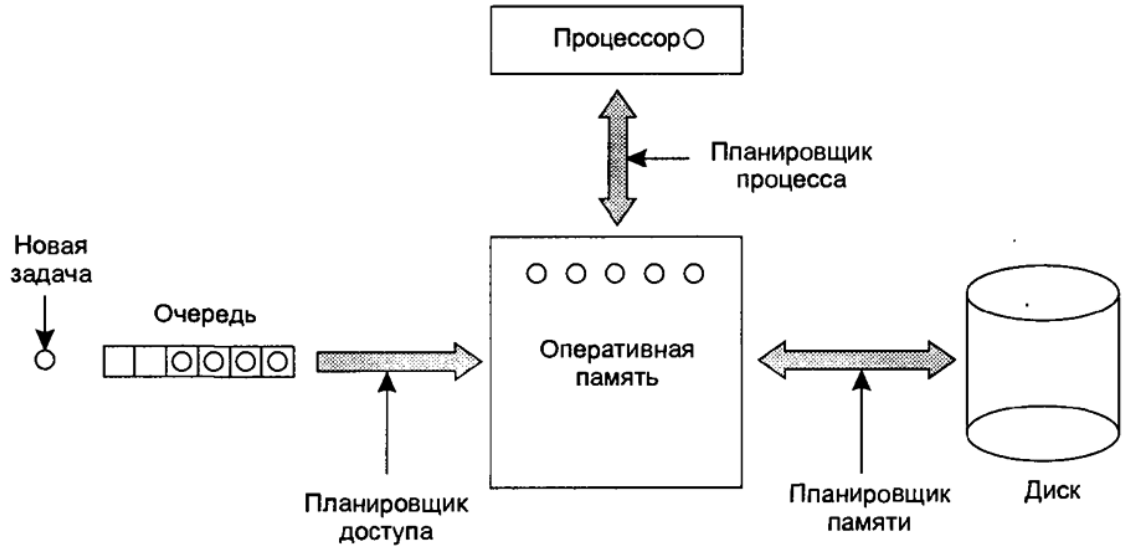


Рис. 2. Схема планирования работы процессора с приостановкой процессов.

Критерии активации процессов:

- сколько времени прошло с тех пор, как процесс был выгружен на диск;
- сколько времени процесс уже использовал процессор;
- какова важность процесса.
- ...

Вопросы для самопроверки

1. Может ли планировщик доступа переместить процесс из оперативной памяти в очередь задач? (Да/Нет)

2. Верно ли, что планировщик процессов вызывается чаще, чем менеджер памяти? (Да/Нет)

Ответы на вопросы

1. Нет. Планировщик доступа осуществляет планирование на верхнем уровне и отвечает за допуск задач к активному соревнованию за процессорное время. Он не может выгружать процессы обратно в очередь задач.

2. Да. Планировщик процессов отвечает за большинство переходов между активными состояниями процессов, что осуществляется чаще, чем приостановка и активация процессов.

§ 3. Планирование с приоритетным вытеснением

Планирование без приоритетного вытеснения (nonpreemptive scheduling) — политика планирования, которая не позволяет системе отбирать процессор у процесса до тех пор, пока сам процесс не отдаст его добровольно либо не закончит работу. Используется, например, в системах пакетной обработки данных.

Планирование с приоритетным вытеснением (preemptive scheduling) — политика планирования, позволяющая отбирать процессор у процесса, и основывающаяся на приоритетах процессов. Используется, например, в системах реального времени и интерактивных системах разделения времени.

Приоритет (priority) — мера важности процесса (потока), используемая для определения порядка и продолжительности его выполнения.

Статические приоритеты (static priorities) — не изменяющиеся во времени приоритеты. Они присваиваются процессам до начала их выполнения и остаются постоянными.

Динамические приоритеты (dynamic priorities) — изменяющиеся во время выполнения процессов приоритеты. Их значение может меняться в зависимости от изменения ситуации.

Вопросы для самопроверки

1. Может ли планирование без приоритетного вытеснения иметь преимущество перед планированием с вытеснением? (Да/Нет)

2. Может ли процесс, вошедший в бесконечный цикл выполнения, монопольно захватить систему с приоритетным вытеснением? (Да/Нет)

3. Может ли динамический планировщик отдать предпочтение процессу с низким приоритетом, запросившим недоиспользованный ресурс? (Да/Нет)

Ответы на вопросы

1. Да. Планирование без приоритетного вытеснения обеспечивает предсказуемое время обработки заданий, что необходимо для систем пакетной обработки данных.

2. Да. Если этот процесс имеет самый высокий приоритет. Операционная система может справляться с такими ситуациями, ограничивая максимальное время, в течение которого конкретный процесс может использовать процессор.

3. Да. Высока вероятность того, что этот ресурс окажется свободным и процесс, запросивший его сможет его использовать и завершить свою работу еще до того, как высокоприоритетный процесс дождется освобождения занятого ресурса.

§ 4. Цели планирования

- Обеспечить максимальную пропускную способность системы (число процессов в единицу времени)
- Гарантировать максимальному числу пользователей, работающих в интерактивном режиме приемлемые времена ответа
- Обеспечить максимальное использование ресурсов системы
- Исключить бесконечное откладывание (когда процесс ожидает появления события, которое может никогда не произойти)
- Учитывать приоритеты (оказывать предпочтение процессам с более высокими приоритетами)
- Минимизировать накладные расходы (потеря ресурсов системы на цели планирования)
- Обеспечить предсказуемость (гарантировать, что выполнение процесса займет конкретное время независимо от загрузки системы)

- Обеспечить равноправие (одинаково справедливое отношение ко всем процессам в системе)
- Обеспечить масштабируемость (плавное снижение производительности при увеличении загрузки системы)

Вопросы для самопроверки

1. Противоречит ли цель обеспечить равноправие процессов необходимости учитывать приоритеты? (Да/Нет)

2. Противоречит ли цель обеспечить предсказуемость необходимости учитывать приоритеты? (Да/Нет)

3. Всегда ли издержки планирования представляют собой бесполезную трату ресурсов? (Да/Нет)

Ответы на вопросы

1. Да. Трудно обеспечить одинаково справедливое отношение ко всем процессам в системе и одновременно оказывать предпочтение процессам с более высокими приоритетами.

2. Да. Трудно гарантировать, что выполнение процесса займет конкретное время независимо от загрузки системы, если в систему все время будут поступать новые более высокоприоритетные процессы.

3. Нет. Потеря ресурсов системы на цели планирования при эффективном планировании может с лихвой компенсироваться обеспечением максимального использования процессами этих ресурсов.

§ 5. Типы процессов

Процесс, интенсивно использующий ввод/вывод (I/O-bound) — процесс, который обычно кратковременно использует процессор перед генерацией запроса ввода/вывода.

Процесс, интенсивно использующий процессор (processor-bound) — процесс, расходующий в ходе выполнения весь отведенный ему квант времени. Подобные процессы обычно интенсивно проводят вычисления и генерируют мало запросов ввода/вывода.

Пакетный процесс (batch process) — процесс, который может работать без вмешательства пользователя.

Интерактивный процесс (interactive process) — процесс, которому во время работы необходимо вести диалог с пользователем.

Процесс, выполняемый в режиме реального времени (real-time process) — процесс, для которого не приемлемо замедление обслуживания во времени.

Пример. Приложения мультимедиа.

Вопросы для самопроверки

1. Пакетные процессы всегда относятся к разновидности процессов, интенсивно использующих процессор? (Да/Нет)

2. Интерактивные процессы обычно относятся к разновидности процессов, интенсивно использующих ввод/вывод? (Да/Нет)

3. Известно ли обычно планировщику, сколько времени процессу осталось до завершения? (Да/Нет)

Ответы на вопросы

1. Нет. Пакетные процессы не взаимодействуют с пользователем, и поэтому их обычно относят к процессам, интенсивно использующим процессор. Если же подобный процесс часто обращается к диску, то его относят к интенсивно использующим ввод/вывод.

2. Да. Интерактивные процессы часто вынуждены ожидать ввода данных от пользователя, поэтому в первую очередь их можно отнести к процессам, интенсивно использующим ввод/вывод.

3. Нет. Планировщику редко бывает точно известно, сколько времени необходимо каждому процессу до завершения. Процессы, если им разрешить это, могут неправильно указывать ожидаемое время завершения, чтобы получить лучшие условия обслуживания планировщиком.

§ 6. Базовые алгоритмы планирования

Планирование по принципу FIFO (first in — first out, первым пришел — первым ушел) — механизм планирования, согласно которому процессор поступает в распоряжение процессов в порядке их попадания в очередь готовых процессов (см. рис. 3). Принцип FIFO не

поддерживает приоритетное вытеснение — процесс продолжает свое выполнение до полного завершения.

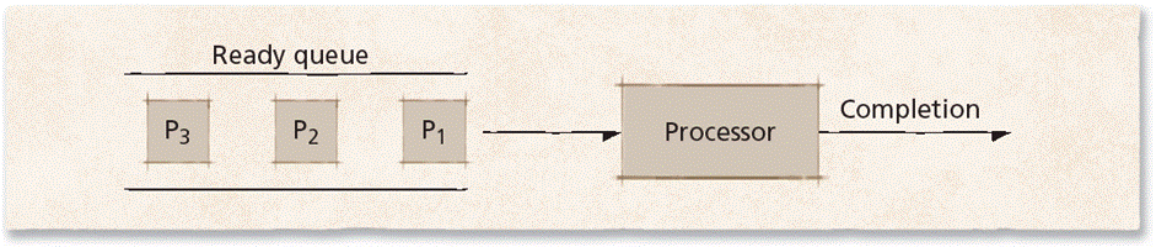


Рис. 3. Схема планирования по принципу FIFO. Здесь ready queue — очередь готовности, completion — завершение

Планирование по принципу FIFO

- Задает простейший план выполнения процессов
- Менее важные процессы могут заставить ждать своей очереди более важные для системы процессы
- В современных системах редко используется самостоятельно в качестве основной дисциплины обслуживания
- Можно встретить в качестве составляющей во многих современных алгоритмах планирования (например, в алгоритме планирования на основе приоритетов, где процессы с одинаковым приоритетом обслуживаются по принципу FIFO)

Циклическое планирование (round-robin scheduling, RR) — политика планирования, согласно которой каждый процесс в состоянии готовности может выполняться в течение не более, чем одного кванта в каждом цикле. После того как по одному разу будет запущен каждый процесс из очереди, планировщик начнет новый цикл, запустив на выполнение первый процесс из очереди (см. рис. 4).

Циклическое планирование

- Эффективно для работы в интерактивной среде

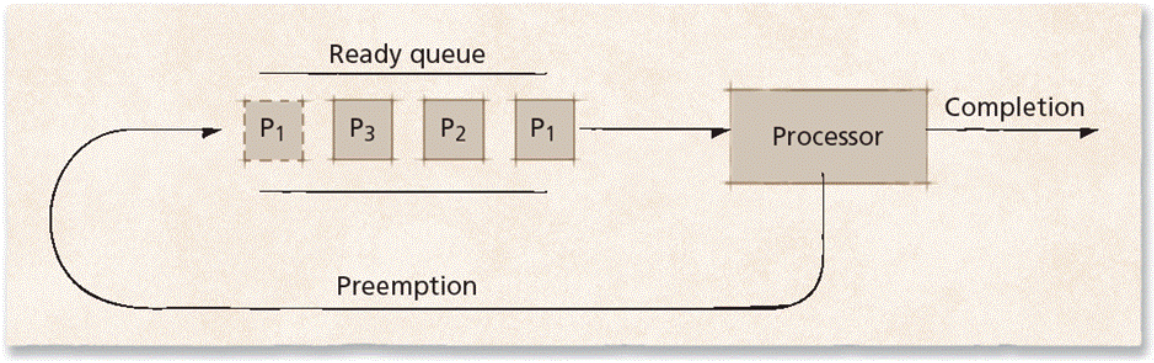


Рис. 4. Схема циклического планирования. Здесь preemption — приоритетное вытеснение

- В современных системах редко используется самостоятельно в качестве основной дисциплины обслуживания
- Можно встретить как компонент во многих современных алгоритмах планирования (например, в системах реального времени)

Вопросы для самопроверки

1. Может ли в системе, использующей принцип FIFO, возникнуть ситуация бесконечного откладывания, если все процессы должны обязательно завершить свою работу? (Да/Нет)

2. Правда ли, что в современных системах принцип планирования FIFO используется редко? (Да/Нет)

3. Может ли дисциплина планирования RR учитывать приоритеты процессов? (Да/Нет)

4. Верно ли, что алгоритм RR является менее приемлемым для обслуживания интерактивных пользователей, чем принцип FIFO? (Да/Нет)

Ответы на вопросы

1. Нет. Бесконечное откладывание не должно возникнуть, поскольку все пребывающие процессы будут помещены в конец очереди, не мешая выполняться ждущим процессам.

2. Нет. Его можно встретить во многих современных алгоритмах планирования в качестве составной части более сложных алгоритмов.

3. Да. Для этого надо изменять величину кванта времени и процессам разных приоритетов выделять кванты разных размеров.

4. Нет. Планирование по принципу FIFO заставляет ждать завершения работы выполняющегося процесса, что может оказаться неприемлемым для интерактивных процессов.

§ 7. Величина кванта времени

Квант (quantum) — временной интервал, в течение которого процесс может использовать процессор до момента приоритетного вытеснения (preemption).

Оптимальная величина кванта

- Меняется от системы к системе, от процесса к процессу
- Зависит от нагрузок
- Для процессов интенсивно использующих ввод/вывод равна времени, необходимому каждому процессу, чтобы сгенерировать запрос ввода/вывода, а затем отдать процессор следующему процессу
- Для интерактивных процессов слишком малый квант приведет к частому переключению процессов и небольшой эффективности, слишком большой квант может привести к медленному реагированию на короткие интерактивные нагрузки

Величина квантов в Linux

- По умолчанию для каждого процесса равна 100 мс (1 мс $= 10^{-3}$ с)
- Может меняться от 10 до 200 мс в зависимости от поведения и приоритета процесса

Величина квантов в Windows XP

- По умолчанию для каждого процесса равна в большинстве случаев 20 мс
- Зависит от архитектуры системы
- Может быть разной для разных процессов

Вопросы для самопроверки

1. Верно ли, что оптимальная величина кванта времени равна 20 мс? (Да/Нет)

2. Можно ли заранее определить оптимальную величину кванта для процессов, интенсивно использующих ввод/вывод? (Да/Нет)

Ответы на вопросы

1. Нет. Она может меняться в зависимости от поведения и приоритета процесса.

2. Нет. В общем случае невозможно спрогнозировать ход выполнения программы, то есть система не может точно сказать, когда процесс сгенерирует запрос ввода/вывода.

§ 8. Многоуровневые очереди с обратной связью

Многоуровневая очередь с обратной связью (multilevel feedback queue) — структура, обеспечивающая группирование процессов с одинаковым приоритетом в одну очередь, для обслуживания которой применяется принцип FIFO, или алгоритм RR (см. рис. 5).

Многоуровневая очередь с обратной связью

- Процессы, интенсивно использующие процессор, помещаются в очереди более низкого приоритета, поскольку пакетные процессы обычно не требуют малых времен ответа
- Процессы, интенсивно использующие ввод/вывод, которые используют процессор кратковременно перед работой с устройствами ввода/вывода, остаются в высоко приоритетных очередях
- В роли подобных процессов обычно выступают интерактивные процессы, которые нуждаются в малых временах ответа

Базовый алгоритм (см. рис. 5)

- Новый процесс входит в сеть очередей с конца верхней очереди
- Процесс перемещается по очереди, реализующей принцип FIFO, пока не получит в свое распоряжение процессор

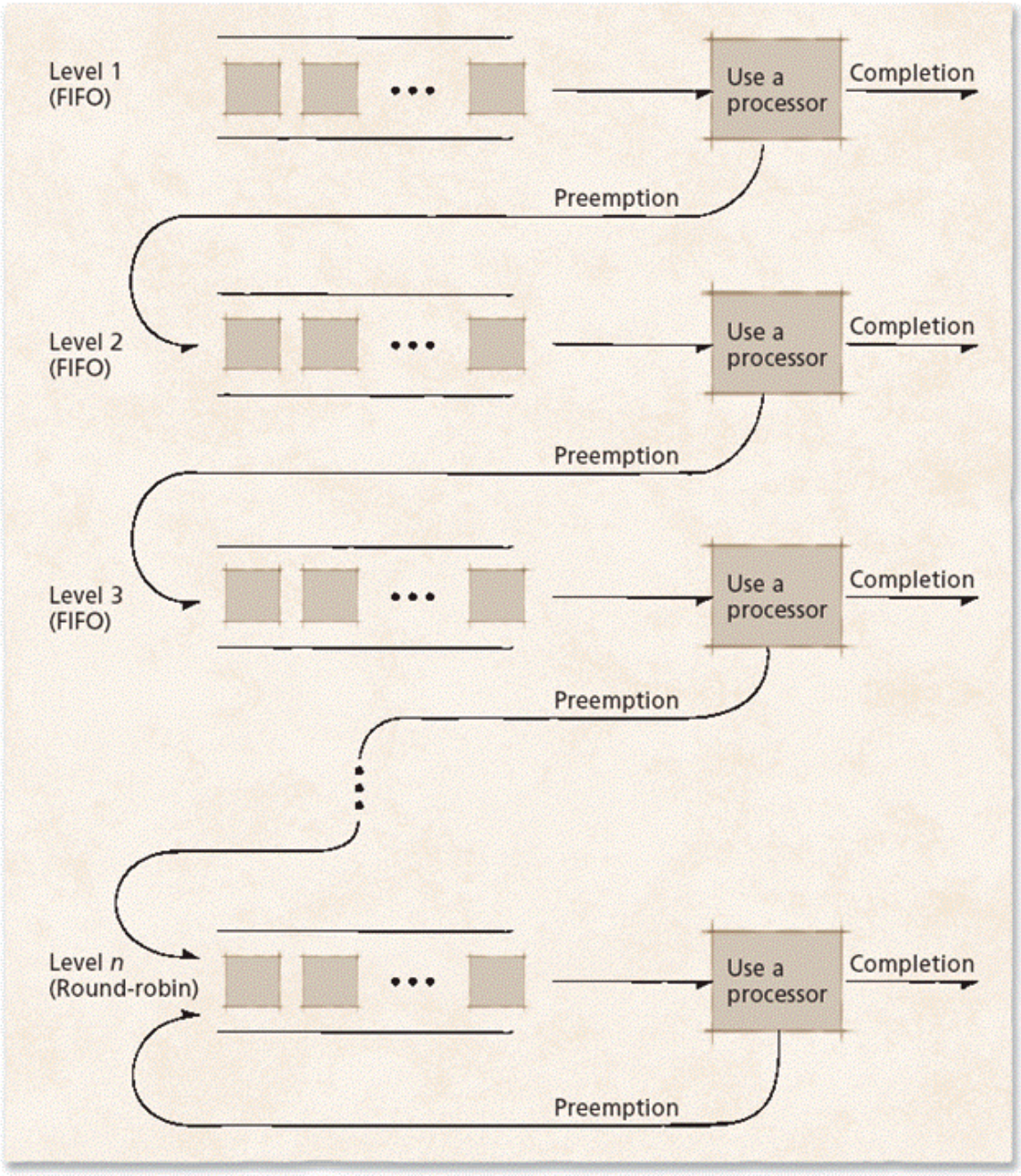


Рис. 5. Многоуровневая очередь с обратной связью

- Если процесс завершает свою работу до конца выделенного ему кванта времени, он уходит на завершение
- Если до конца выделенного процессу кванта времени он генерирует запрос ввода/вывода, процесс блокируется и выходит из сети очередей
- Если выделенный квант времени истекает до того, как процесс добровольно освободит процессор, этот процесс помещается в конец следующей очереди более низкого уровня с меньшим приоритетом и большим квантом времени
- Если данный процесс продолжит использовать полный квант времени, предоставляемый на каждом уровне, он и дальше будет переходить в конец очереди следующего нижележащего уровня
- В системе предусматривается очередь самого нижнего уровня, которая реализует принцип RR, в которой данный процесс циркулирует до тех пор, пока не закончит свою работу

Вопросы для самопроверки

1. Верно ли, что верхняя очередь имеет самый высокий приоритет? (Да/Нет)

2. Верно ли, что в очередях более низкого уровня процессам выделяются большие кванты времени? (Да/Нет)

3. Верно ли, что в самой нижней очереди реализован принцип циклического обслуживания процессов? (Да/Нет)

Ответы на вопросы

1. Да. Процесс первоначально входит в сеть очередей начиная с верхней очереди, имеющей самый высокий приоритет.

2. Да. Чем ниже уровень очереди, тем большие кванты времени выделяются процессам. Интерактивные процессы циркулируют в верхних очередях, интенсивно использующие процессор — в нижних.

3. Да. Это сделано для того, чтобы процессы, дошедшие до самой нижней очереди, в конце концов смогли закончить свою работу.

§ 9. Обслуживание процессов разных типов

Обслуживание интенсивно использующих ввод/вывод процессов

- Таким процессам отдается предпочтение, поскольку они входят в сеть очередей с высоким приоритетом и быстро получают процессор в свое распоряжение
- Квант времени для первой очереди выбирается достаточно большим, с тем чтобы подавляющее большинство процессов, связанных с вводом/выводом (в том числе интерактивных), успели выдать запрос ввода/вывода еще до истечения кванта
- Когда подобный процесс выдает запрос ввода/вывода, он блокируется и выходит из сети очередей, получив приоритетное обслуживание
- Повторный вход процесса происходит тогда, когда он снова переходит в состояние готовности
- После блокирования процесс входит в ту же очередь, из которой он вышел

Обслуживание процессов, интенсивно использующих процессор

- Процесс поступает в очередь сети с самым высоким приоритетом
- На этом этапе очередь не знает, к какой категории относится данный процесс — и цель сети выяснить это как можно скорее
- Процесс получает в свое распоряжение процессор весьма быстро, однако после истечения выделенного ему кванта времени процесс переходит в очередь следующего нижележащего уровня
- Теперь этот процесс будет иметь более низкий приоритет, так что поступающие процессы будут первыми получать в свое распоряжение процессор

- Интерактивные процессы будут продолжать получать хорошие времена ответа, даже если многие процессы интенсивно использующие процессор, постоянно теряют приоритет, переходя на очереди нижележащих уровней
- В конце концов, процесс, интенсивно использующий процессор, получит его в свое распоряжение
- Ему будет выделен квант времени большей величины, чем в очередях высшего приоритета, но он снова использует этот квант полностью
- Затем он будет перемещен в очередь следующего нижележащего уровня
- Таким образом, данный процесс будет продолжать переходить в очереди с более низкими приоритетами
- Он будет дольше ждать в промежутках между выделяемыми квантами времени
- Этот процесс будет каждый раз полностью использовать свой квант, когда будет получать в свое распоряжение процессор
- В конце концов, этот вычислительный процесс окажется в очереди самого низкого уровня с циклическим обслуживанием
- В этой очереди он будет циркулировать пока не завершится

Модификации базового алгоритма

Проблема: как во время запуска разделять процессы на категории в соответствии с их потребностями в процессорном времени?

Решение:

- когда процесс выходит из сети очередей, он может быть помечен признаком очереди самого низкого уровня, в которой он побывал;

- когда этот процесс в последствии вновь войдет в сеть очередей, он будет направлен прямо в ту очередь, в которой он в последний раз завершил работу.

Проблема: как реагировать на изменение характера процесса, например, на то, что процесс по преимуществу интенсивно использующий процессор, начинает преимущественно использовать ввод/вывод?

Решение:

- эту проблему можно решить, если в метке, которой сопровождается процесс, указать также степень использования им выделенных квантов времени во время последнего пребывания в сети очередей;

- если процесс полностью использует выделенный ему квант времени, система поместит его в очередь более низкого уровня;

- если процесс сгенерирует запрос ввода/вывода до того, как закончится квант, он может быть помещен в очередь более высокого уровня.

Проблема: как учитывать время ожидания обслуживания процесса?

Решение: планировщик может использовать механизм старения процессов, помещая процесс в очередь более высокого уровня после того, как процесс провел определенное время, ожидая обслуживания.

Проблема: как еще более улучшить обслуживание процессов, интенсивно использующих ввод/вывод по сравнению с процессами, интенсивно использующими процессор?

Решение:

- процесс может несколько раз циркулировать в каждой очереди, прежде чем перейти в очередь следующего нижележащего уровня;

- количество подобных циклов может увеличиваться по мере перехода процесса на нижележащие уровни.

Вопросы для самопроверки

1. После блокирования процесс входит в ту же очередь, из которой он вышел, согласно базовому алгоритму многоуровневой очереди? (Да/Нет)

2. Верно ли, что процесс, максимально интенсивно использующий процессор в конце концов будет обслуживаться в самой нижней очереди? (Да/Нет)

3. Верно ли, что многоуровневая очередь с обратной связью является примером адаптивного механизма? (Да/Нет)

4. Верно ли, что процесс всегда входит в сеть очередей с конца верхней очереди? (Да/Нет)

Ответы на вопросы

1. Да. Согласно базовому алгоритму многоуровневой очереди, когда процесс переходит в состояние готовности после блокирования, он вновь входит в сеть очередей, причем, в ту же очередь из которой он вышел.

2. Да. Процесс, использующий процессор максимальным образом и не выдающий запросов ввода/вывода, опустится на самый нижний уровень.

3. Да. Адаптивный механизм — это механизм, позволяющий реагировать на изменение поведения контролируемой им системы. Многоуровневая очередь с обратной связью является показательным примером адаптивного механизма.

4. Нет. Процесс может быть направлен прямо в ту очередь, в которой он в последний раз завершил работу.

§ 10. Оптимальное число очередей и уровней приоритета

Проблема: как обеспечить максимальную пропускную способность системы (число процессов в единицу времени) для процессов, интенсивно использующих процессор?

Решение:

- при увеличении числа уровней процессы, интенсивно использующие процессор, вынуждены дольше ждать, что приводит к снижению пропускной способности;
- необходимо уменьшить число уровней.

Проблема: как обеспечить максимальное использование процессорного времени и устройств ввода/вывода?

Решение:

- необходимо увеличить число очередей;
- с увеличением числа уровней многим процессам удается завершить работу и сгенерировать запрос ввода/вывода прежде, чем истечет выделенный им квант времени;
- это позволяет эффективно использовать процессорное время и устройства ввода/вывода.

Планировщик Linux

- Различает 40 различных уровней приоритета: от –20 до 19
- В соответствии с конвенцией UNIX наименьшее значение означает наибольший приоритет
- –20 — самый высокий приоритет, который может иметь процесс (поток)
- Каждому процессу (потоку) при запуске присваивается приоритет, который может быть динамически изменен
- Приоритет может быть изменен, но не более чем на 5 единиц в любую сторону в зависимости от поведения процесса (потока)

Планировщик Windows XP

- Различает 32 уровня приоритета от 0 до 31

- Потоки реального времени (т.е. такие потоки, которые должны обеспечить быструю реакцию на запросы пользователей) занимают верхние 16 уровней (от 16 до 32)
- Их приоритеты являются статическими и не меняются во время выполнения
- Нижние 16 уровней занимают динамические потоки, их приоритеты могут меняться в зависимости от поведения
- Однако динамический приоритет потока не может опуститься ниже определенного для него базового уровня либо подняться до диапазона приоритетов реального времени

Вопросы для самопроверки

1. Для обеспечения максимального использования вычислительных ресурсов необходимо уменьшать число очередей? (Да/Нет)

2. Для ускорения обслуживания процессов, интенсивно использующих процессор, необходимо уменьшать число очередей? (Да/Нет)

3. Планировщики Linux и Windows XP различают одинаковое число уровней приоритета? (Да/Нет)

Ответы на вопросы

1. Нет. Необходимо увеличить число очередей. Чем больше очередей, тем точнее учитываются потребности процессов в ресурсах.

2. Да. Необходимо уменьшить число уровней. При увеличении числа уровней процессы, интенсивно использующие процессор, вынуждены дольше ждать, что приводит к снижению пропускной способности.

3. Нет. Планировщики Linux различает 40 различных уровней приоритета, а Windows XP — 32.

§ 11. Планирование потоков Java

- Любой Java-апплет является многопоточным
- Приоритет любого потока Java может находиться в пределах от Thread.MIN_PRIORITY = 1 до Thread.MAX_PRIORITY = 10

- По умолчанию каждому потоку присваивается приоритет Thread.NORM_PRIORITY = 5
- Каждый новый поток получает приоритет, равный приоритету породившего его потока
- Если потоки реализуются в пространстве пользователя, Java-библиотеки времени выполнения используют квантование времени

Квантование времени (timeslicing) — задание для потоков плана выполнения, согласно которому каждый поток может выполняться в течение не более чем одного кванта до момента приоритетного вытеснения.

Поток уровня пользователя (user-level thread) — модель потоков, согласно которой все потоки процесса связываются с одним контекстом выполнения.

Потоки уровня пользователя

- При задании плана выполнения многопоточных процессов, реализованных в виде потоков пользовательского уровня, операционной системе ничего не известно о количестве потоков процесса
- Операционная система оперирует ими как единым блоком, требуя наличия библиотеки уровня пользователя для планирования выполнения потоков, входящих в состав данного процесса

Поток уровня ядра (kernel-level thread) — поток, создаваемый самой операционной системой.

Потоки уровня ядра

- Если система поддерживает потоки уровня ядра, она может задавать план выполнения каждого потока независимо от других потоков этого процесса
- В зависимости от платформы, потоки Java могут быть реализованы

  - в пространстве пользователя (некоторые ранние версии UNIX и OS Solaris);
  - в пространстве ядра (Linux, Windows XP).

Планирование потоков Java

- Планировщик потоков Java гарантирует, что поток с самым высоким приоритетом в виртуальной машине Java будет выполняться всегда
- Если в системе существует несколько потоков с одинаковым приоритетом, то для их обслуживания будет использована дисциплина RR

План выполнения потоков Java по приоритетам (см. рис. 6)

- Потоки A и B выполняются согласно алгоритму RR до своего окончания
- До самого конца выполняется поток C
- ...

Метод yield класса Thread

- Позволяет текущему потоку добровольно освобождать процессор, чтобы позволить выполняться потоку с таким же приоритетом
- Необходимость в функции yield существует только для систем, в которых не поддерживается квантование времени
- В таких системах поток обычно выполняется до завершения, прежде чем другой поток с таким же приоритетом получит возможность продолжить выполнение
- Поскольку приложения Java нацелены на переносимость между платформами и поскольку программисты не всегда знают, что конкретная платформа поддерживает квантование времени, программисты применяют метод yield, чтобы гарантировать корректную работу своего приложения

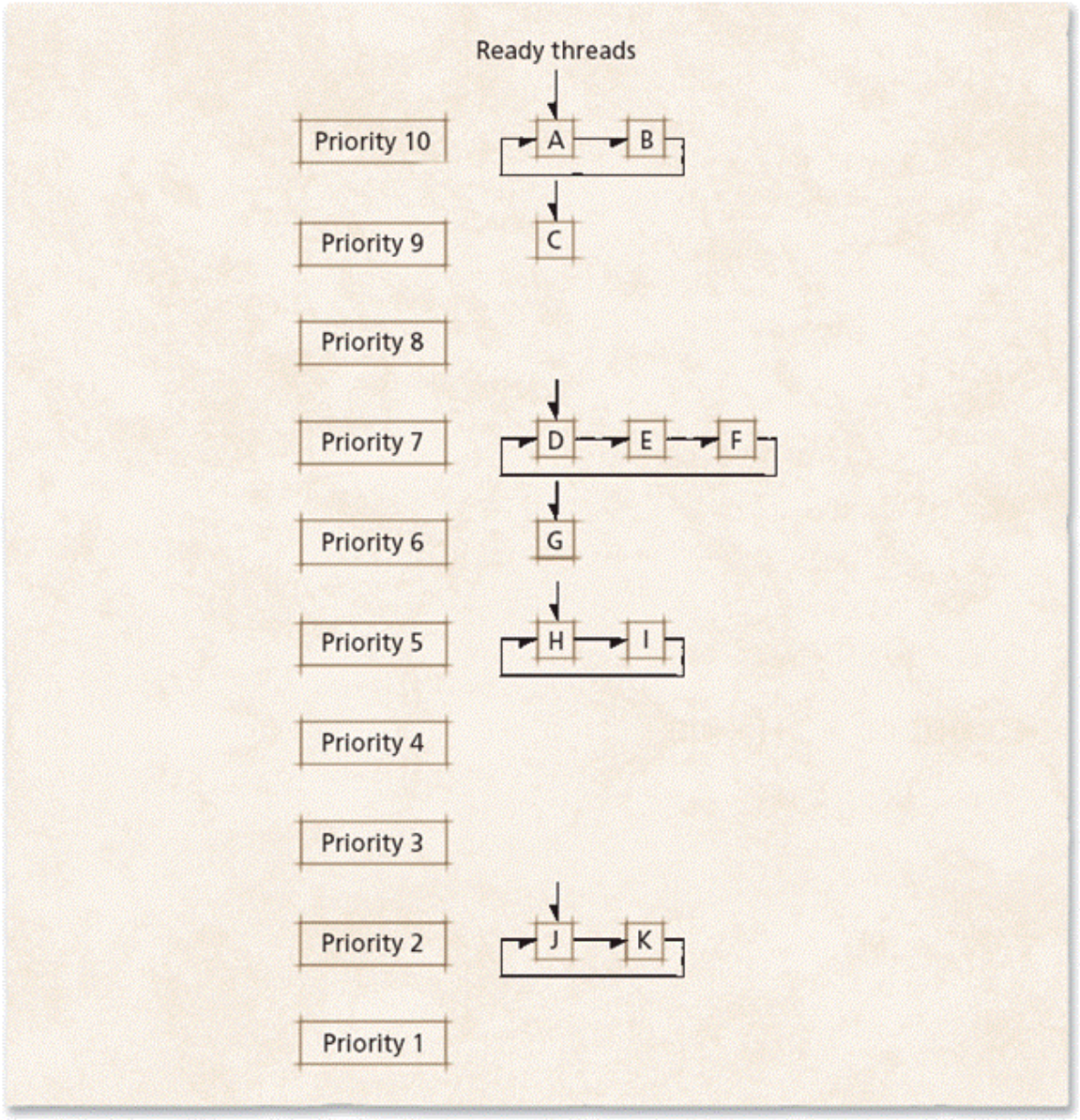


Рис. 6. План выполнения потоков Java по приоритетам

Вопросы для самопроверки

1. Могут ли прибывающие потоки более высокого приоритета привести к бесконечному откладыванию потоков с более низким приоритетом? (Да/Нет)

2. Поток Java с более низким приоритетом никогда не сможет начать работу, пока поток более высокого приоритета находится в состоянии готовности? (Да/Нет)

Ответы на вопросы

1. Да. Эта ситуация может произойти с потоками Java, если в операционной системе не будет предусмотрена соответствующая защита.

2. Да. Механизм планирования Java будет выполнять поток с самым высоким приоритетом, находящийся в состоянии готовности.

Часть III

Реальная и виртуальная память

Глава 7

ОПЕРАТИВНАЯ ПАМЯТЬ

§ 1. Стратегии управления памятью

- Стратегии загрузки
- Стратегии размещения
- Стратегии замены

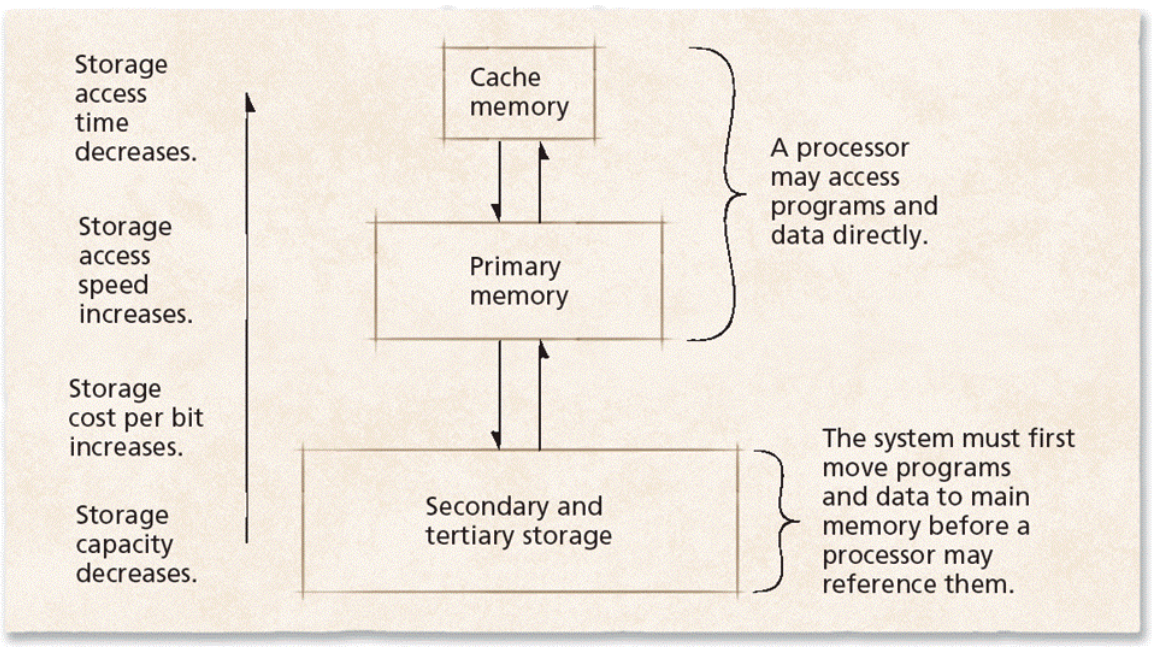


Рис. 1. Иерархическая организация памяти

Стратегия загрузки (fetch strategy) — подход, с помощью которого система определяет, когда загружать следующий фрагмент программы или данных со вторичного устройства хранения в оперативную память.

Стратегия размещения (placement strategy) — стратегия определяющая, где в оперативной памяти будут размещаться загружаемые программы и данные.

Стратегия замены (replacement strategy) — стратегия, определяющая, какие фрагменты программ и данных в оперативной памяти заменяются новыми.

Вопросы для самопроверки

1. Увеличивает ли производительность программ, содержащих циклы, наличие кэш-памяти? (Да/Нет)

2. Дешевизна оперативной памяти сделала ненужной использование сложных систем управления памятью? (Да/Нет)

3. Эффективное использование памяти важнее для стратегии управления ею, чем минимальные накладные расходы? (Да/Нет)

Ответы на вопросы

1. Да. Программа, содержащая циклы, раз за разом выполняет одни и те же последовательности инструкций. Если эти инструкции помещаются в кэш, процессор сможет обращаться к ним намного быстрее, чем если бы они находились в оперативной памяти.

2. Нет. Несмотря на дешевизну и большую емкость оперативной памяти все время появляются новые задачи, поглощающие всю доступную память.

3. Нет. Ответ зависит от назначения системы и относительной ценности памяти и накладных расходов. В общем случае разработчик операционной системы должен стараться выдержать баланс в этом вопросе.

§ 2. Выделение непрерывных блоков в однопользовательских системах

Однопользовательская система с выделением непрерывных блоков (single-user contiguous memory allocation system) — система, в которой программы размещаются в непрерывной области памяти и выполняются по одной за раз (см. рис. 2). Использовалась в ранних ЭВМ.

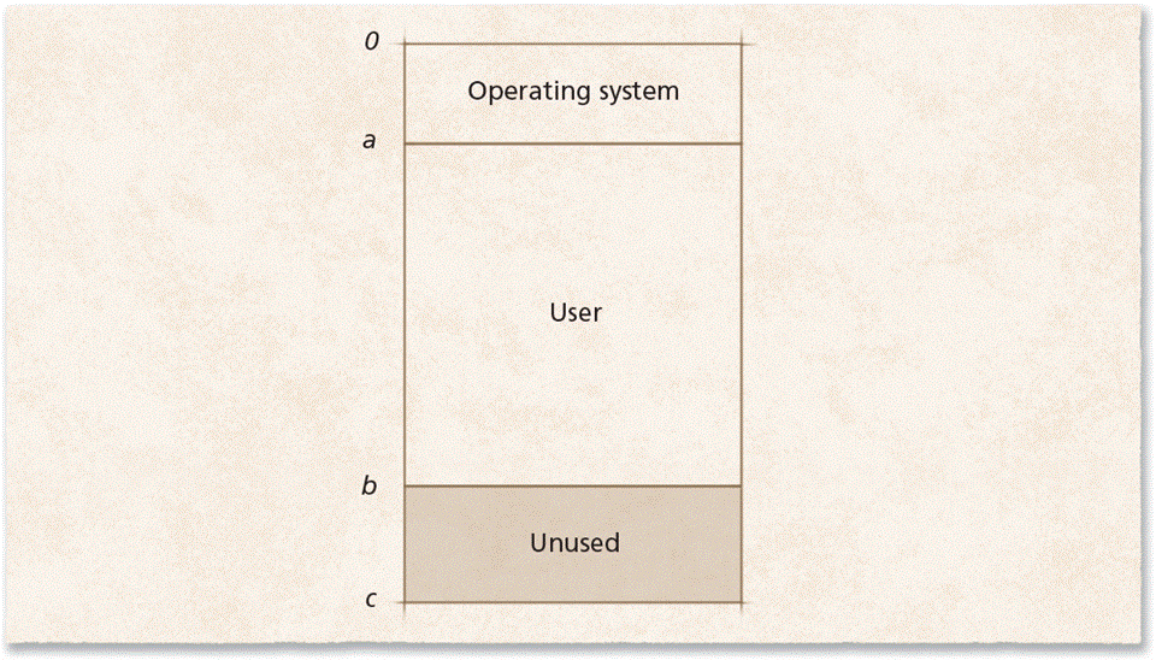


Рис. 2. Однопользовательская система с выделением непрерывных блоков

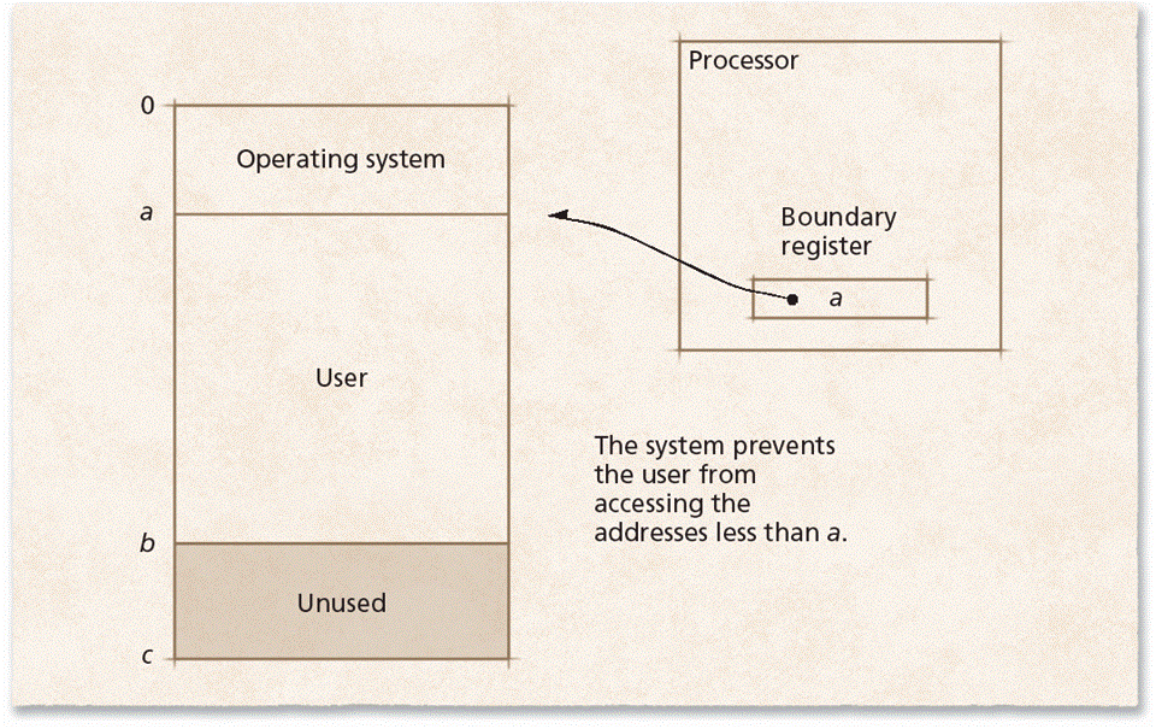


Рис. 3. Использование регистра границы

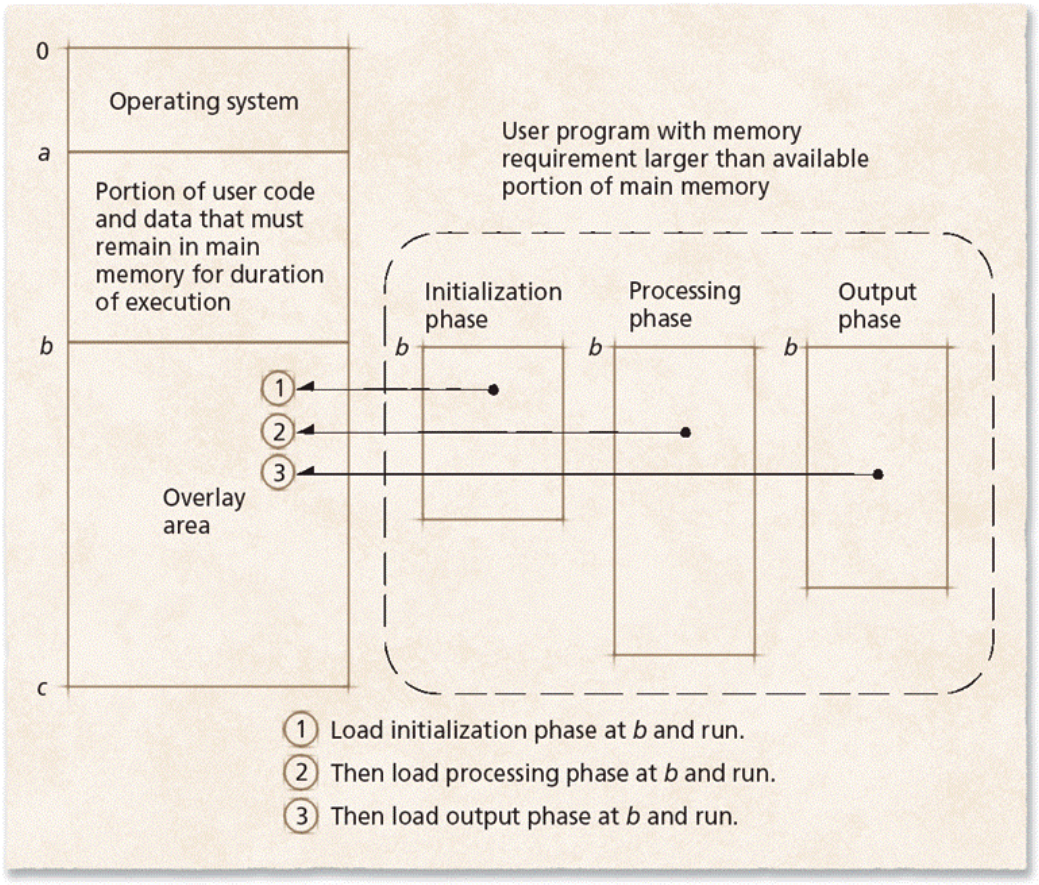


Рис. 4. Структура оверлеев. Программа пользователя делится на три части: инициализации, обработки и вывода данных. Части программы последовательно загружаются в область оверлея b и выполняются (этапы 1–3). По адресу a находится часть кода и данных программы, которая должна оставаться в оперативной памяти все время выполнения этой программы.

Регистр границы (boundary register) — в однопользовательских системах — регистр, использующийся для защиты памяти посредством разграничения области памяти ядра и пользовательской области памяти (см. рис. 3).

Оверлей (overlay) — подход, позволяющий выполняться программам, большим по объему, чем доступная в системе оперативная память. Программы делятся на части (их называют оверлеями), которым не обязательно находиться в памяти одновременно (см. рис. 4).

Вопросы для самопроверки

1. Позволяли ли оверлеи писать программы, большие, чем объем оперативной памяти? (Да/Нет)

2. Достаточно ли одного регистра границы для защиты в многопользовательских системах? (Да/Нет)

Ответы на вопросы

1. Да. Программы при этом делились на части, которые последовательно загружались в память. Управление оверлеями усложняло программы, увеличивало их размер и стоимость разработки.

2. Нет. Потому что один регистр сможет защитить операционную систему от повреждения пользовательскими процессами, но не защитит пользовательские процессы от повреждения другими процессами.

§ 3. Мультипрограммные системы с фиксированным распределением памяти

Мультипрограммный режим с фиксированным распределением памяти (fixed-partition multiprogramming) — организация оперативной памяти, в которой вся доступная память делится на несколько разделов фиксированного размера. В каждом из этих разделов размещается по одной задаче. Этот режим использовался в первых мультипрограммных системах.

Раздел (partition) — в оперативной памяти — часть памяти, выделенная процессу в мультипрограммной системе. Программы помещаются в разделы, чтобы операционная система могла защитить себя от пользовательских процессов и процессы — от других процессов.

Мультипрограммные системы с фиксированным распределением памяти

- С абсолютной трансляцией и загрузкой
- С перемещаемой трансляцией и загрузкой

Абсолютная загрузка (absolute loading) — технология загрузки, согласно которой загрузчик размещает программы в памяти по

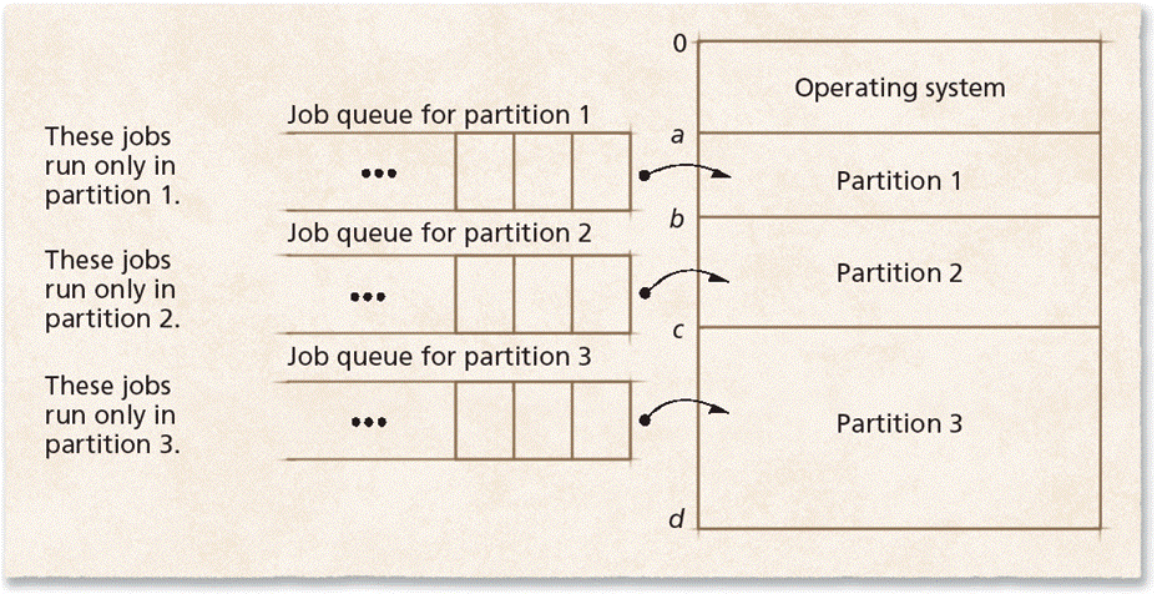


Рис. 5. Мультипрограммная система с фиксированным распределением памяти, абсолютной трансляцией и загрузкой

определенному программистом или транслятором абсолютному адресу (см. рис. 5). Для этой технологии характерны потери памяти (см. рис. 6).

Перемещаемая загрузка (relocatable loading) — технология загрузки, при которой относительные адреса в загрузочном модуле (определяемые на основе их расположения по отношению к началу модуля) трансформируются в абсолютные адреса, основанные на расположении запрашиваемого раздела памяти (см. рис. 7).

Базовый регистр (base register, нижняя граница, low boundary) — регистр, содержащий наименьший адрес в памяти, к которому может обращаться процесс (см. рис. 8). Использовался для защиты памяти в мультипрограммных системах с фиксированным распределением памяти.

Регистр предела (limit register, верхняя граница, high boundary) — регистр, в котором в мультипрограммных системах с фиксированным распределением памяти хранится адрес конца области памяти, выделенной процессу (см. рис. 8).

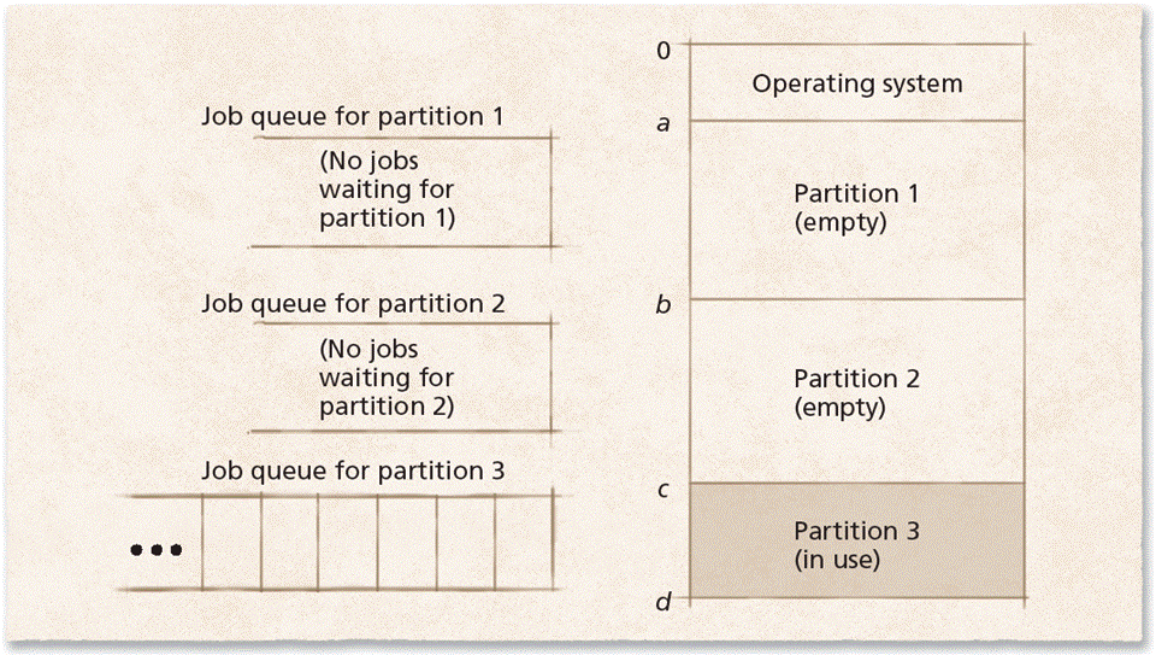


Рис. 6. Потери памяти в мультипрограммной системе с фиксированным распределением памяти, абсолютной трансляцией и загрузкой

Фрагментация (fragmentation) — в оперативной памяти — невозможность использовать некоторые области свободной оперативной памяти.

Внутренняя фрагментация (internal fragmentation) — явление в мультипрограммных системах с фиксированным распределением памяти, когда размер кода и данных процесса меньше, чем размер раздела, в котором этот процесс размещается (см. рис. 9).

Вопросы для самопроверки

1. Абсолютная трансляция и загрузка эффективнее перемещаемой? (Да/Нет)

2. Большие разделы оперативной памяти лучше маленьких? (Да/Нет)

Ответы на вопросы

1. Нет. До появления перемещаемых трансляторов и загрузчиков программистам приходилось вручную указывать раздел, в котором программы должны были выполняться. При этом могли напрасно расходоваться память и процессорное время.

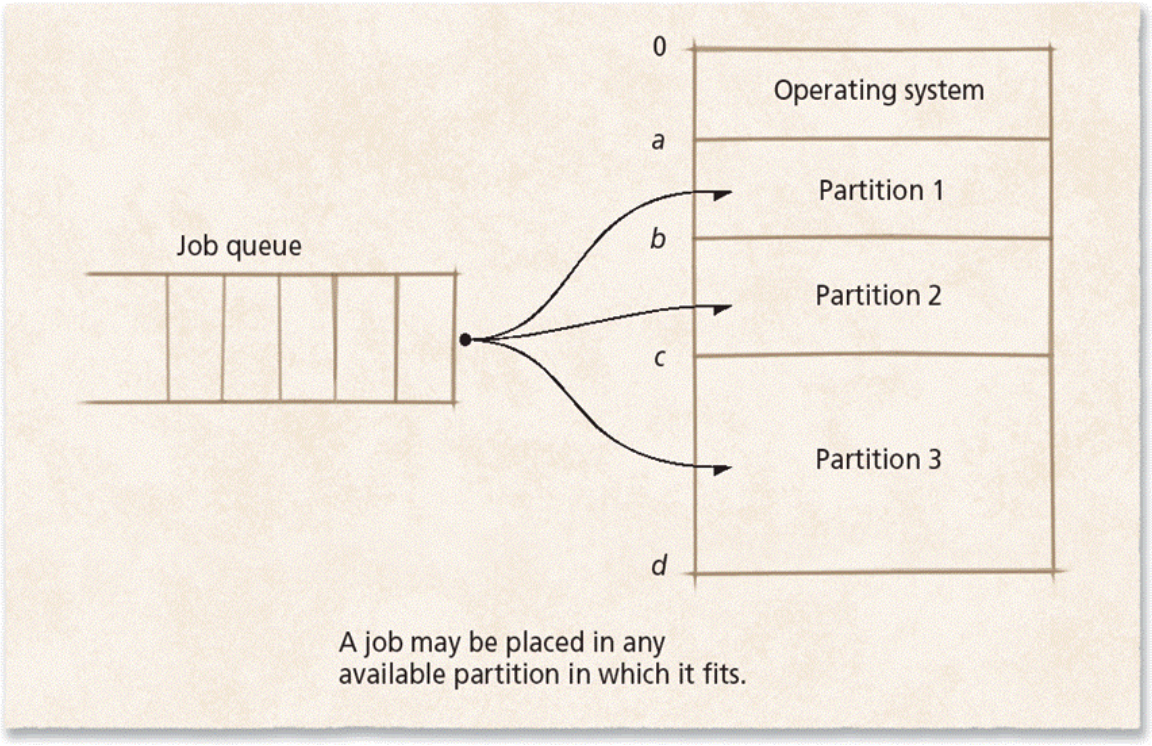


Рис. 7. Мультипрограммная система с фиксированным распределением памяти, перемещаемой трансляцией и загрузкой

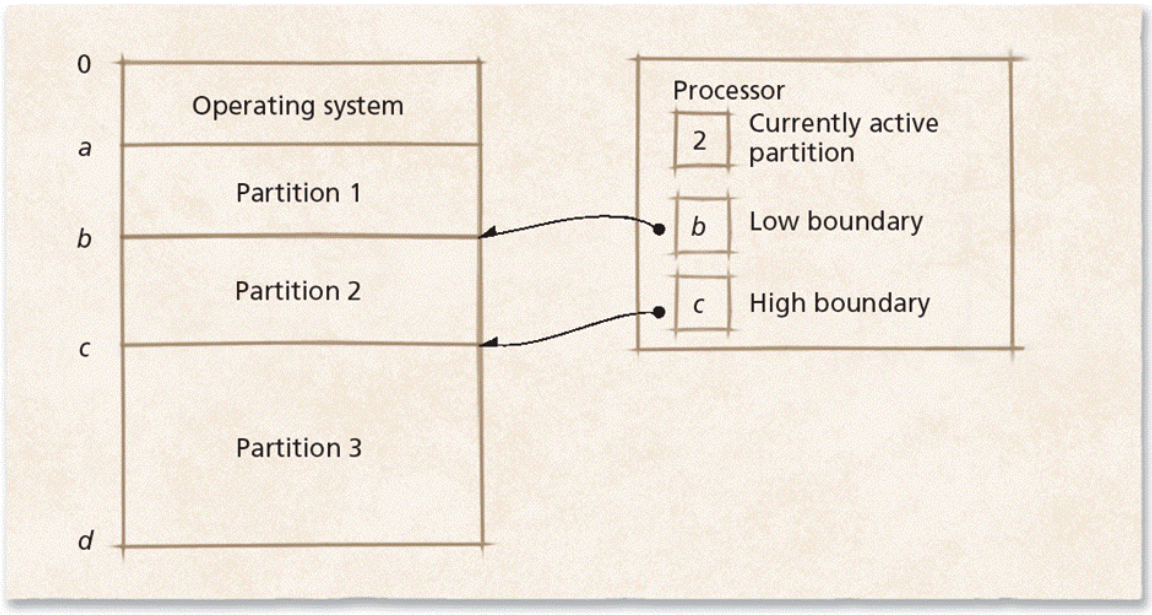


Рис. 8. Защита памяти в мультипрограммных системах с выделением непрерывных областей памяти

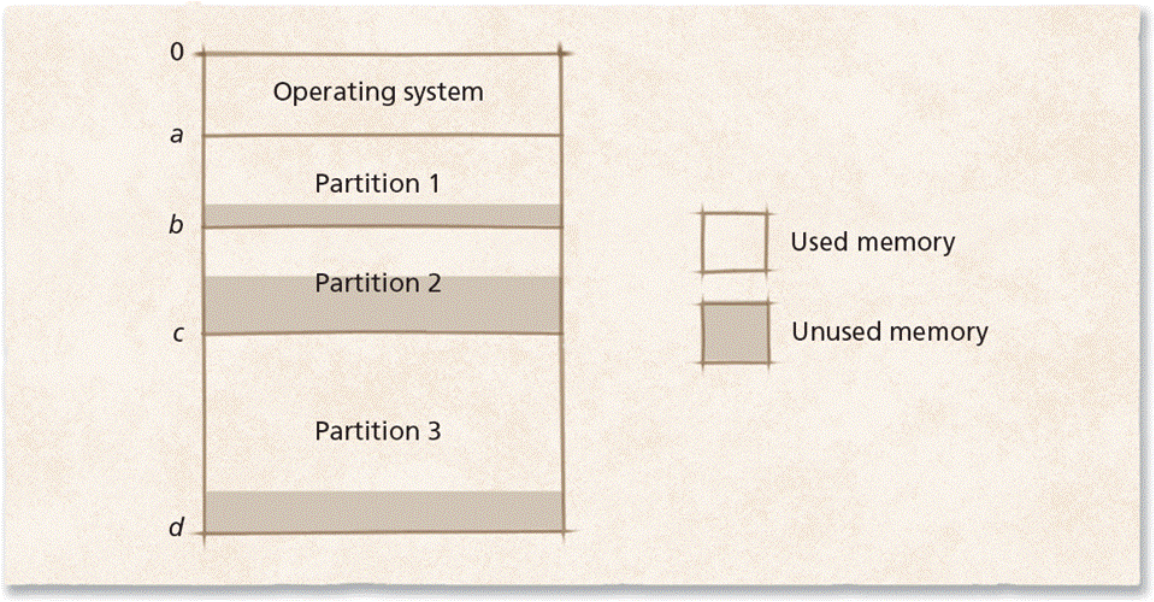


Рис. 9. Внутренняя фрагментация в мультипрограммных системах с фиксированным распределением памяти

2. Нет. В больших разделах могут размещаться большие программы, но если в них размещаются маленькие программы, то появляется сильная внутренняя фрагментация. В маленьких разделах внутренняя фрагментация меньше, и они позволяют достичь большей мультипрограммности, однако они ограничивают размер программ.

§ 4. Мультипрограммные системы с изменяемым распределением памяти

Мультипрограммный режим с изменяемым распределением памяти (variable-partition multiprogramming) — организация оперативной памяти, в которой каждому процессу выделяется ровно столько памяти, сколько ему нужно для работы (см. рис. 10).

Внешняя фрагментация (external fragmentation) — явление в мультипрограммных системах с изменяемым распределением памяти, при котором в адресном пространстве есть неиспользуемые области, слишком маленькие, чтобы вместить процесс.

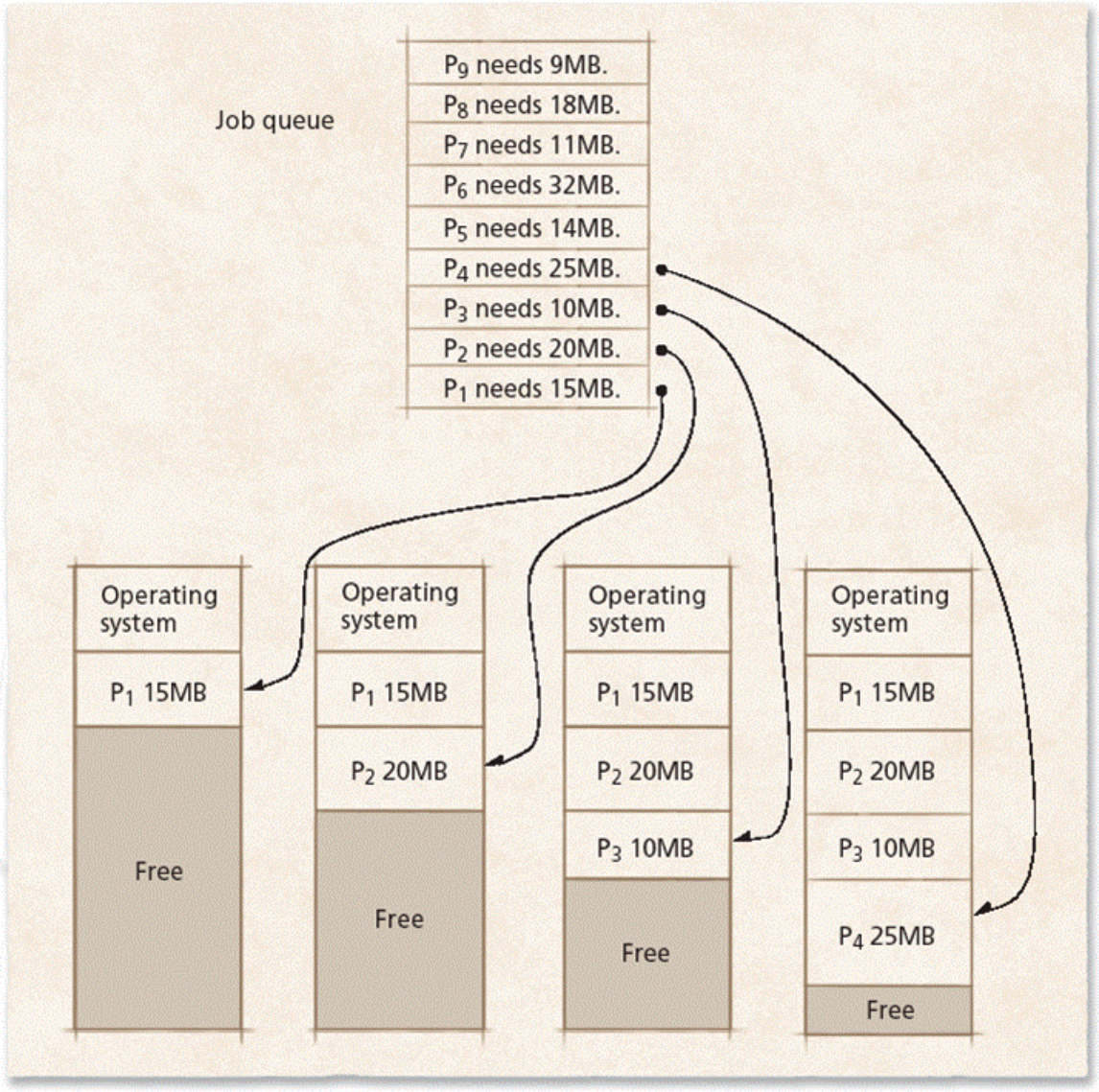


Рис. 10. Первичное выделение разделов в мультипрограммной системе с изменяемым распределением памяти

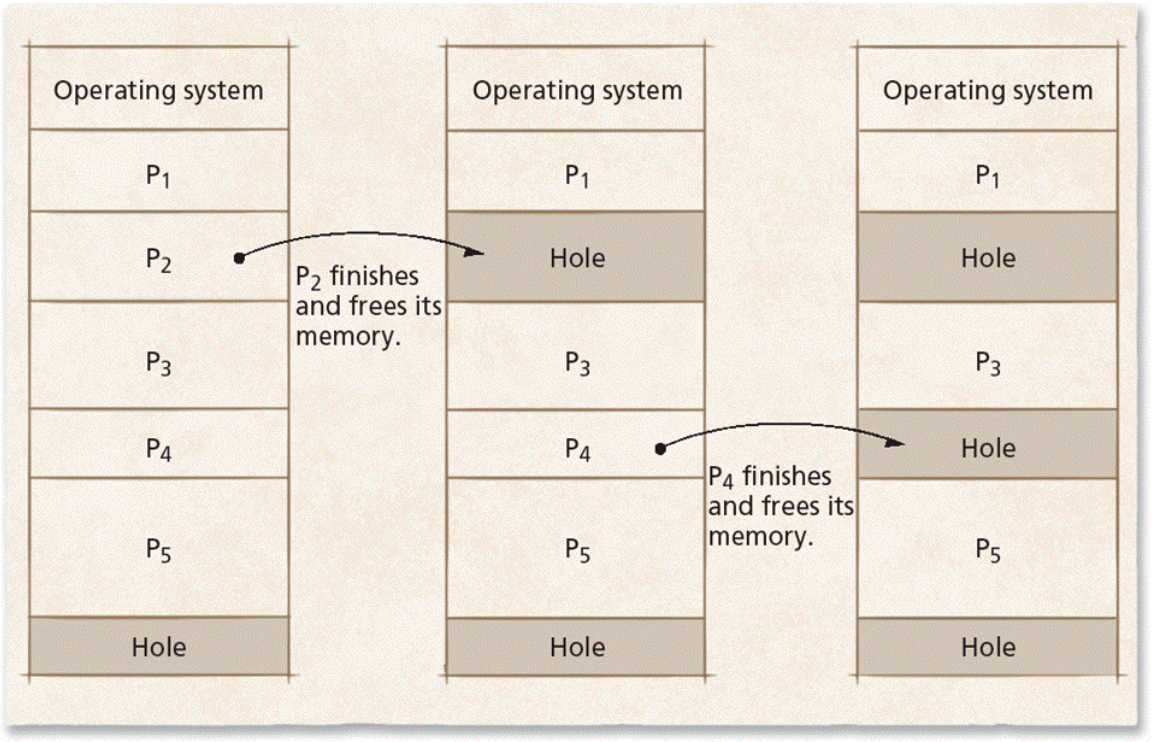


Рис. 11. Дыры в оперативной памяти в мультипрограммной системе с изменяемым распределением памяти

Дыра (hole) — свободная область в памяти в мультипрограммных системах с изменяемым распределением памяти (см. рис. 11).

Сращение дыр в памяти (coalescing the memory holes) — механизм слияния смежных свободных областей памяти в мультипрограммной системе с изменяемым распределением памяти. Он позволяет создать свободные области максимально возможного размера для размещения процессов (см. рис. 12).

Уплотнение памяти (memory compaction) — перемещение всех разделов в мультипрограммной системе с изменяемым распределением памяти к одному краю адресного пространства, чтобы создать свободную область максимально возможного размера (см. рис. 13).

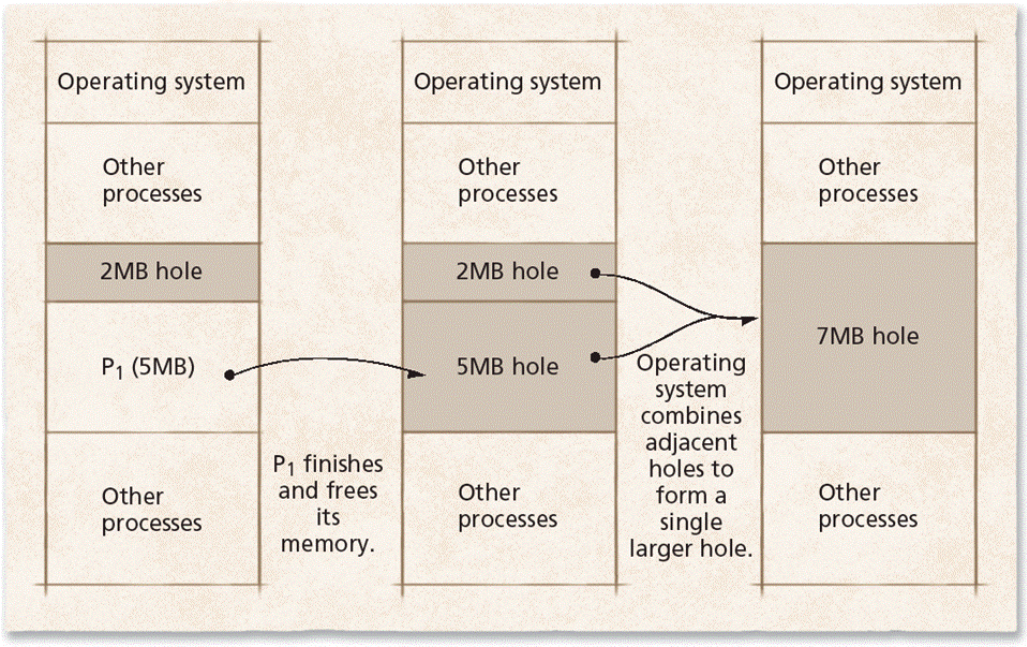


Рис. 12. Сращение дыр в памяти в мультипрограммной системе с изменяемым распределением памяти

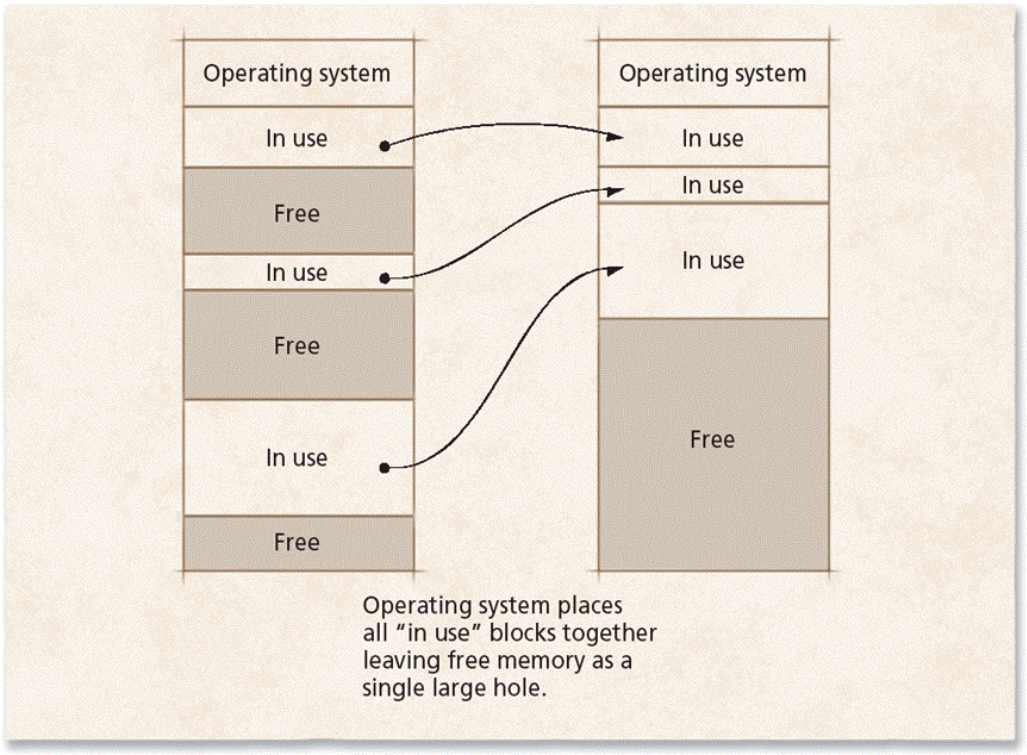


Рис. 13. Уплотнение памяти в мультипрограммной системе с изменяемым распределением памяти

Список свободных блоков (free memory list) — структура данных в операционной системе, содержащая данные о свободных блоках оперативной памяти.

Вопросы для самопроверки

1. Уплотнение памяти не применимо для систем реального времени? (Да/Нет)

2. Может ли в системе с изменяемым распределением памяти возникнуть внутренняя фрагментация? (Да/Нет)

Ответы на вопросы

1. Да. На время выполнения уплотнения выполнение всех процессов должно приостанавливаться. Это может привести к катастрофическим последствиям для систем реального времени.

2. Нет. Внутренняя фрагментация присутствует в средах с фиксированными разделами, если процессу выделяется больше пространства, чем нужно. В средах с изменяемыми разделами появляется внешняя фрагментация, когда не используются промежутки между ними.

§ 5. Стратегии размещения в памяти

- В первых подходящих участках
- В наиболее подходящих участках
- В наименее подходящих участках

Стратегия размещения в первых подходящих участках памяти (first-fit memory placement strategy) — стратегия, размещающая загружаемую задачу в первом найденном достаточном по размеру свободном участке памяти (см. рис. 14).

Стратегия размещения в наиболее подходящих участках памяти (best-fit memory placement strategy) — стратегия, размещающая загружаемую задачу в наименьшем достаточном по размеру участке памяти (см. рис. 15).

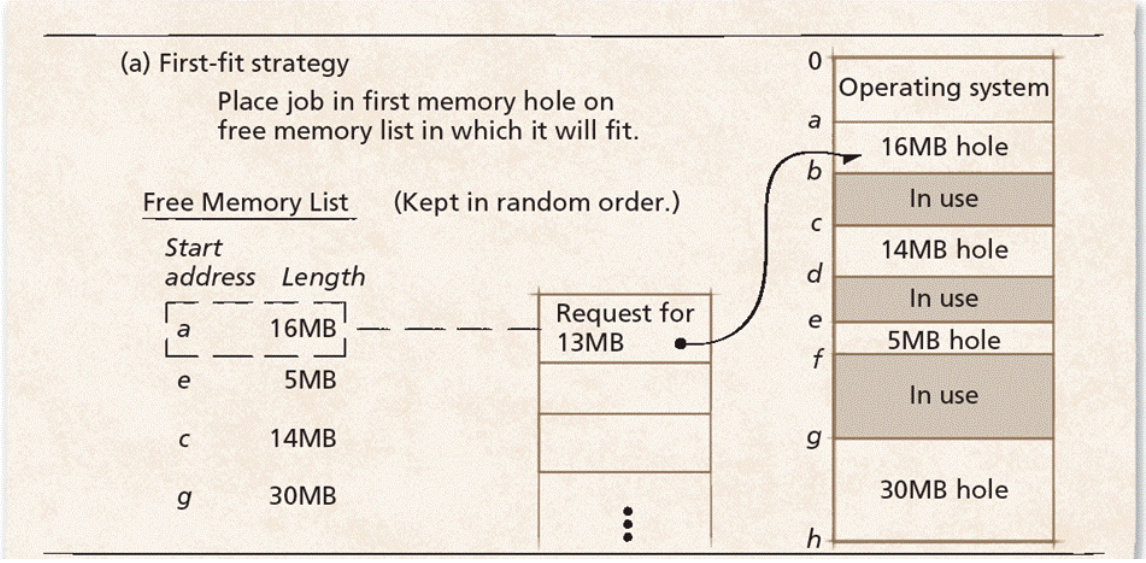


Рис. 14. Стратегия размещения в первых подходящих участках памяти. Список свободных блоков не упорядочен

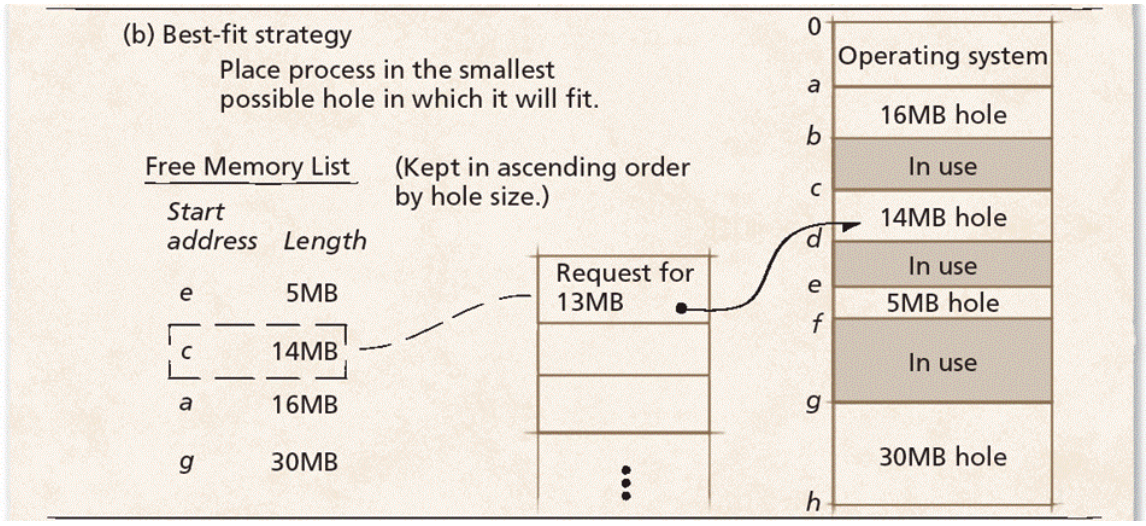


Рис. 15. Стратегия размещения в наиболее подходящих участках памяти. Список свободных блоков отсортирован в порядке возрастания объемов дыр

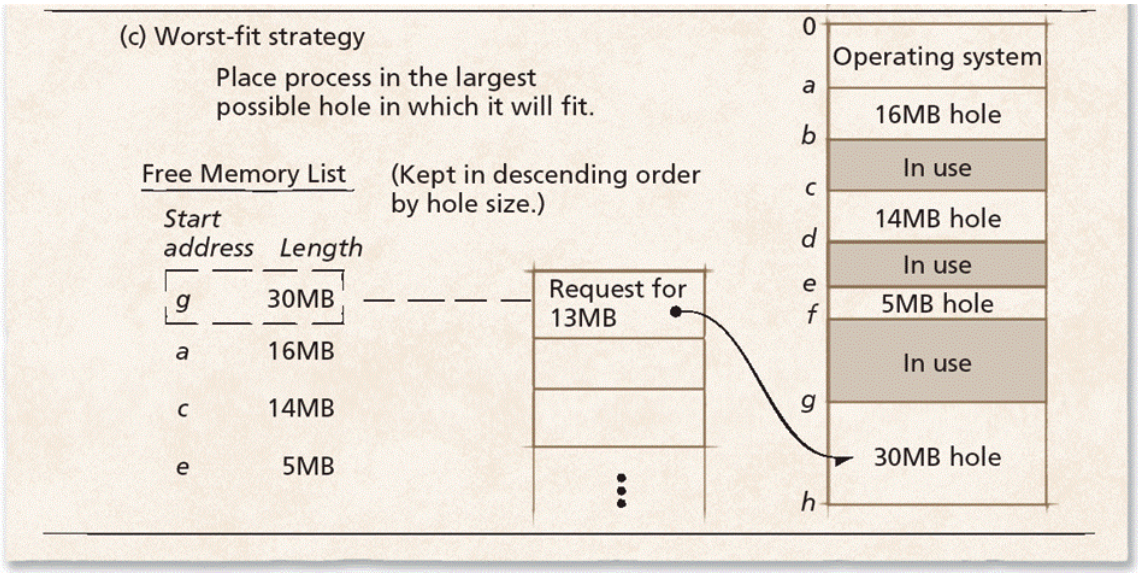


Рис. 16. Стратегия размещения в наименее подходящих участках памяти. Список свободных блоков отсортирован в порядке уменьшения объемов дыр

Стратегия размещения в наименее подходящих участках памяти (worst-fit memory placement strategy) — стратегия, размещающая загружаемую задачу в наибольшем по размеру участке памяти, достаточном для размещения этой задачи (см. рис. 16).

Вопросы для самопроверки

1. Стратегия размещения в первых подходящих участках памяти требует меньше всего накладных расходов? (Да/Нет)

2. Стратегия размещения в наиболее подходящих участках памяти самая эффективная? (Да/Нет)

Ответы на вопросы

1. Да. Не нужно каким-то образом упорядочивать список свободных блоков, соответственно накладные расходы для использования этой стратегии невелики.

2. Нет. Эта стратегия оставляет в памяти много мелких не пригодных к использованию дыр. Если же задача помещена в самую большую дыру, то оставшаяся дыра будет тоже велика и сможет вместить еще одну довольно большую программу.

Глава 8

ВИРТУАЛЬНАЯ ПАМЯТЬ

§ 1. Определение виртуальной памяти

Виртуальная память (virtual memory) — концепция, позволяющая решить проблему ограниченной емкости оперативной памяти за счет предоставления каждому процессу виртуального адресного пространства (возможно, большего объема, чем объем оперативной памяти машины) для хранения данных и исполняемых инструкций.

Виртуальный адрес (virtual address) — адрес, по которому процесс обращается к системе виртуальной памяти.

Виртуальное адресное пространство (virtual address space) — множество виртуальных адресов, по которым может обращаться процесс (см. рис. 1).

Физический адрес (physical address или real address) — адрес ячейки в оперативной памяти.

Физическое адресное пространство (physical address space) — диапазон физических адресов, соответствующий объему оперативной памяти данного компьютера. Физическое адресное пространство может быть меньше, чем виртуальное адресное пространство.

Динамическая трансляция адресов (Dynamic Address Translation, DAT) — механизм, преобразующий виртуальные адреса в физические во время выполнения программы (см. рис. 2). Чтобы не замедлять выполнение, трансляция должна выполняться очень быстро.

Устройство управления памятью (Memory Management Unit, MMU) — специализированное аппаратное устройство, выполняющее, в частности, трансляцию виртуальных адресов в физические.

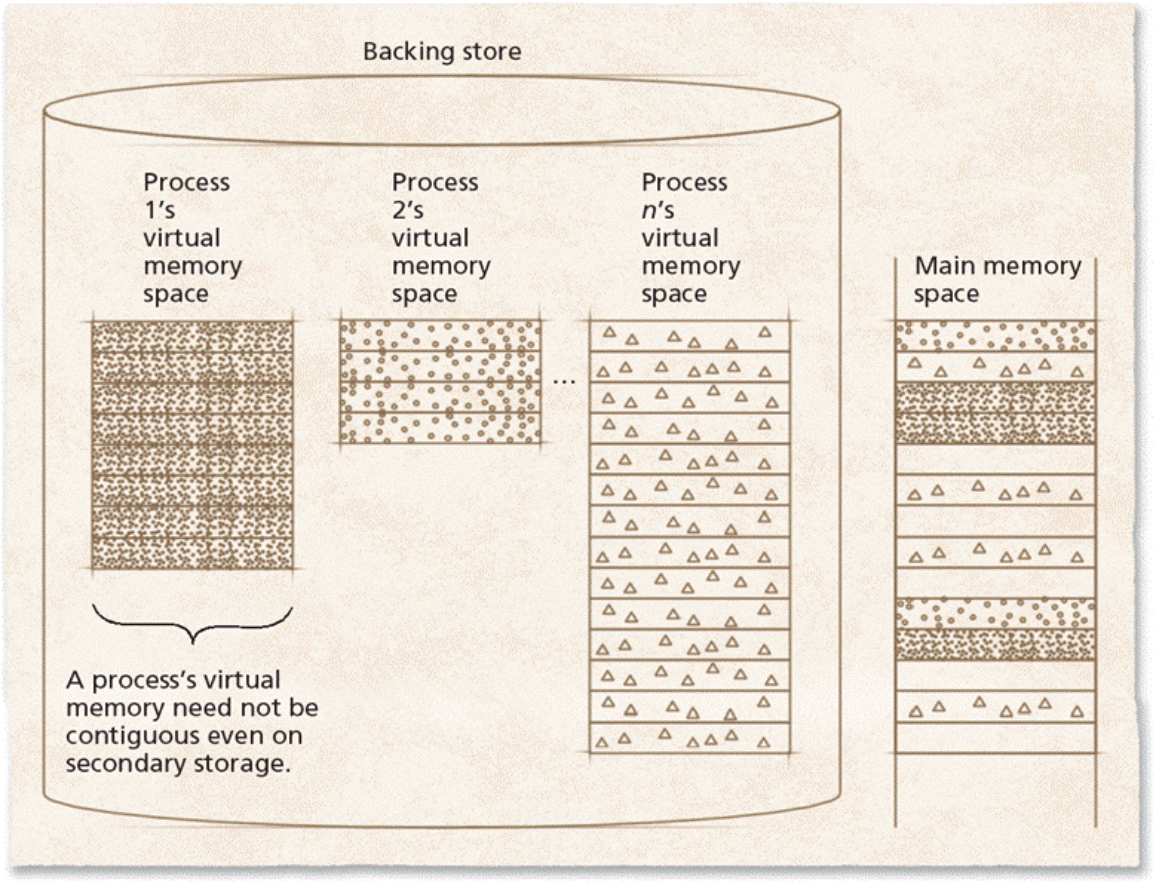


Рис. 1. Фрагменты адресных пространств в оперативной памяти и на вторичном устройстве хранения

Искусственная непрерывность (artificial contiguity) — подход, используемый в системах виртуальной памяти, создающий для программ иллюзию хранения их инструкций и данных в непрерывных областях, хотя в реальности фрагменты кода и данных рассеяны по оперативной памяти (см. рис. 3). Искусственная непрерывность упрощает программирование.

Вопросы для самопроверки

1. Существуют ли программы, которые неэффективно загружать в память целиком перед исполнением? (Да/Нет)

2. Эффективно ли решение проблемы ограниченности оперативной памяти наращиванием ее объема? (Да/Нет)

3. Совпадают ли виртуальное адресное пространство процесса и физическое адресное пространство системы? (Да/Нет)

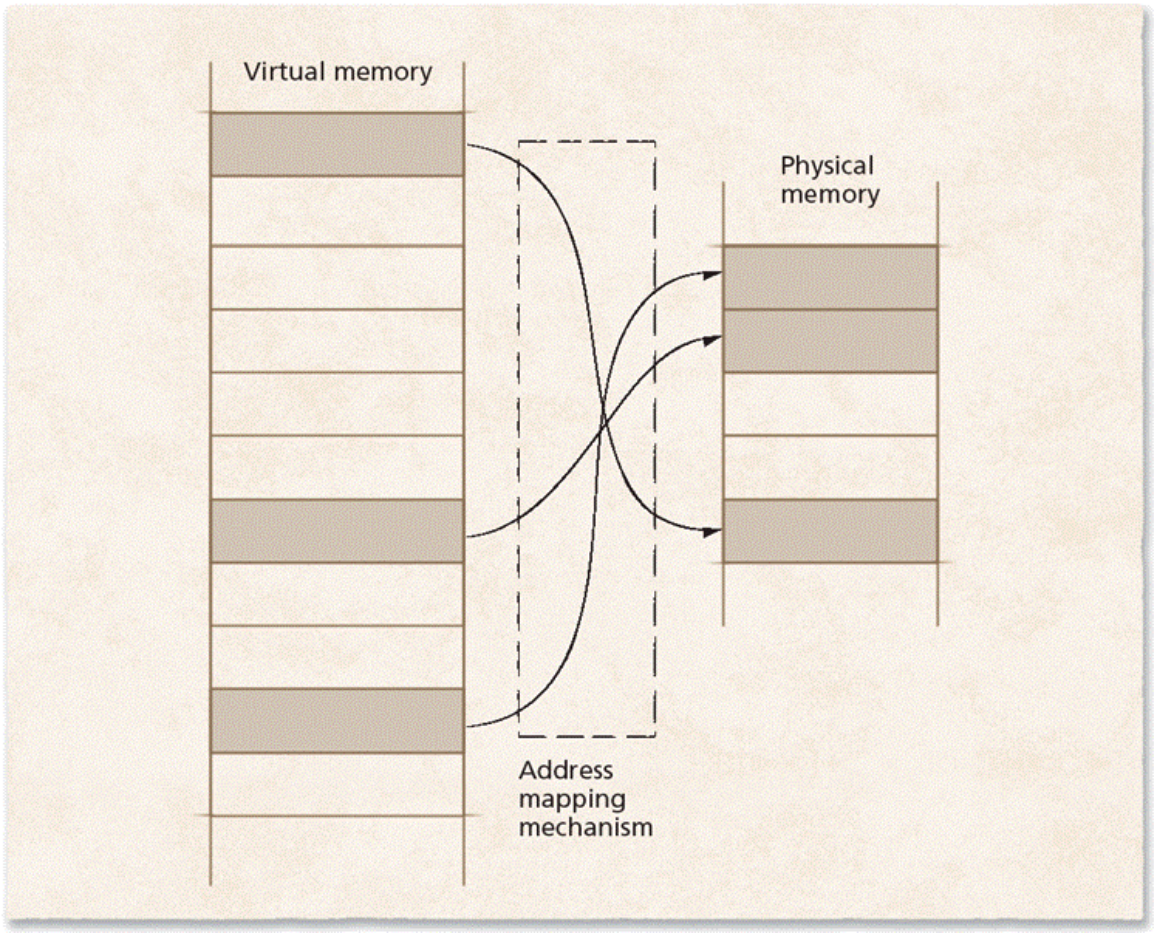


Рис. 2. Преобразование виртуальных адресов в физические

4. Искусственная непрерывность решает проблему ограниченности оперативной памяти? (Да/Нет)

Ответы на вопросы

1. Да. Во многих программах есть функции обработки ошибок, которые редко используются, если используются вообще. Загрузка этих функций в память уменьшает объем памяти, доступный процессам.

2. Нет. Покупка дополнительной оперативной памяти не всегда экономически оправдана. Лучше создать иллюзию наличия в системе объема памяти, заведомо большего, чем может понадобиться программам.

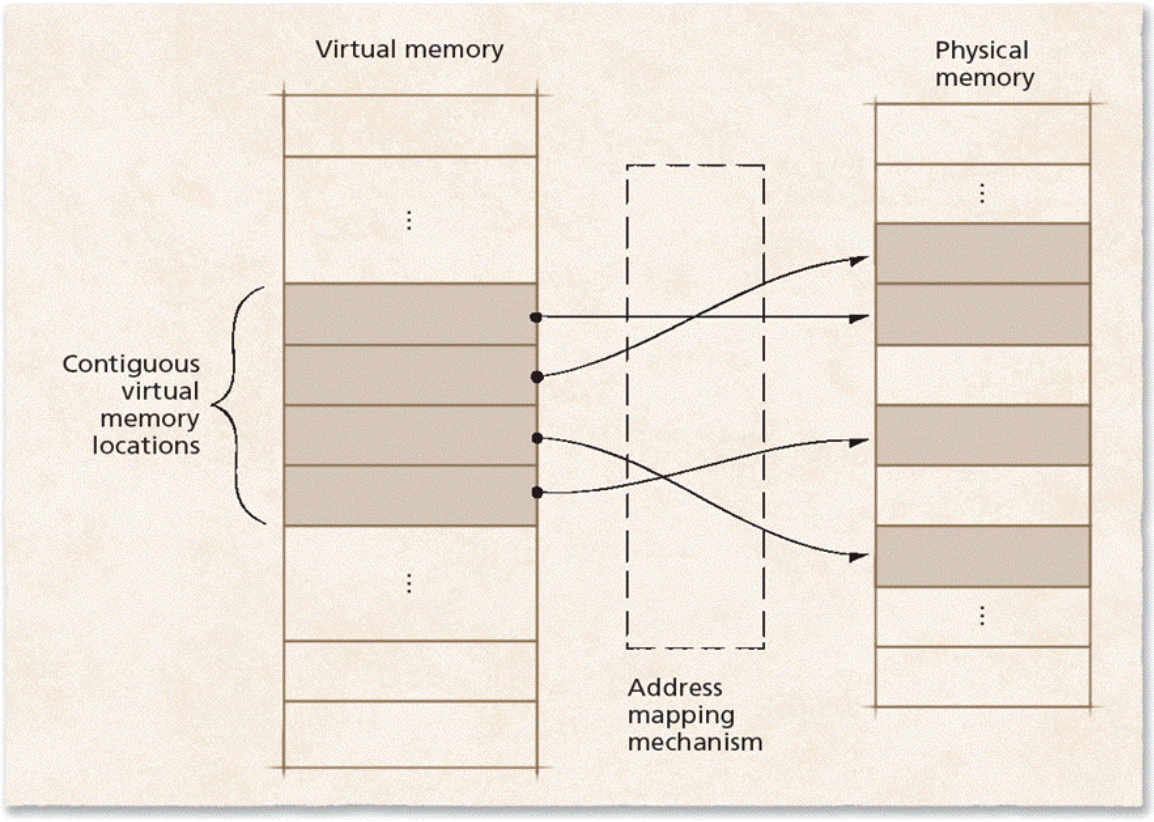


Рис. 3. Искусственная непрерывность

3. Нет. Виртуальное адресное пространство процесса — это множество адресов, по которым процесс может обращаться к памяти, работая под управлением системы виртуальной памяти. Процессы не используют физических адресов, соответствующих реальным ячейкам физической памяти.

4. Нет. Искусственная непрерывность упрощает программирование, позволяя процессу работать с его памятью как с непрерывной областью, даже если данные и код рассеяны по оперативной памяти.

§ 2. Размещение блоков

Блок (block) — область пространства памяти (реального или виртуального), представляющая собой диапазон смежных адресов.

Смещение (displacement или offset) — разность между адресом элемента данных и адресом начала его блока.

Рис. 4. Формат виртуального адреса в системе с блочным размещением

Таблица размещения блоков (block map table) — таблица, содержащая записи о размещении виртуальных блоков процесса в блоках реальной памяти (см. рис. 5).

Таблица размещения блоков

- Находится в оперативной памяти, а ее записи обычно загружаются в кэш-память перед использованием
- Если бы размер блока совпадал с размером ячейки оперативной памяти, то таблица размещения блоков занимала бы больше места, чем доступно в оперативной памяти
- Блоки одного размера называются страницами, разного — сегментами

Регистр адреса таблицы размещения блоков (block map table origin register) — регистр, в котором хранится адрес, по которому в оперативной памяти находится таблица размещения блоков процесса. Этот быстродействующий регистр обеспечивает быструю трансляцию виртуальных адресов (см. рис. 5).

Вопросы для самопроверки

1. Значение регистра адреса таблицы размещения блоков изменяется при переключении контекста? (Да/Нет)

2. Большие блоки памяти лучше маленьких? (Да/Нет)

3. Должны ли все блоки памяти быть одинакового размера? (Да/Нет)

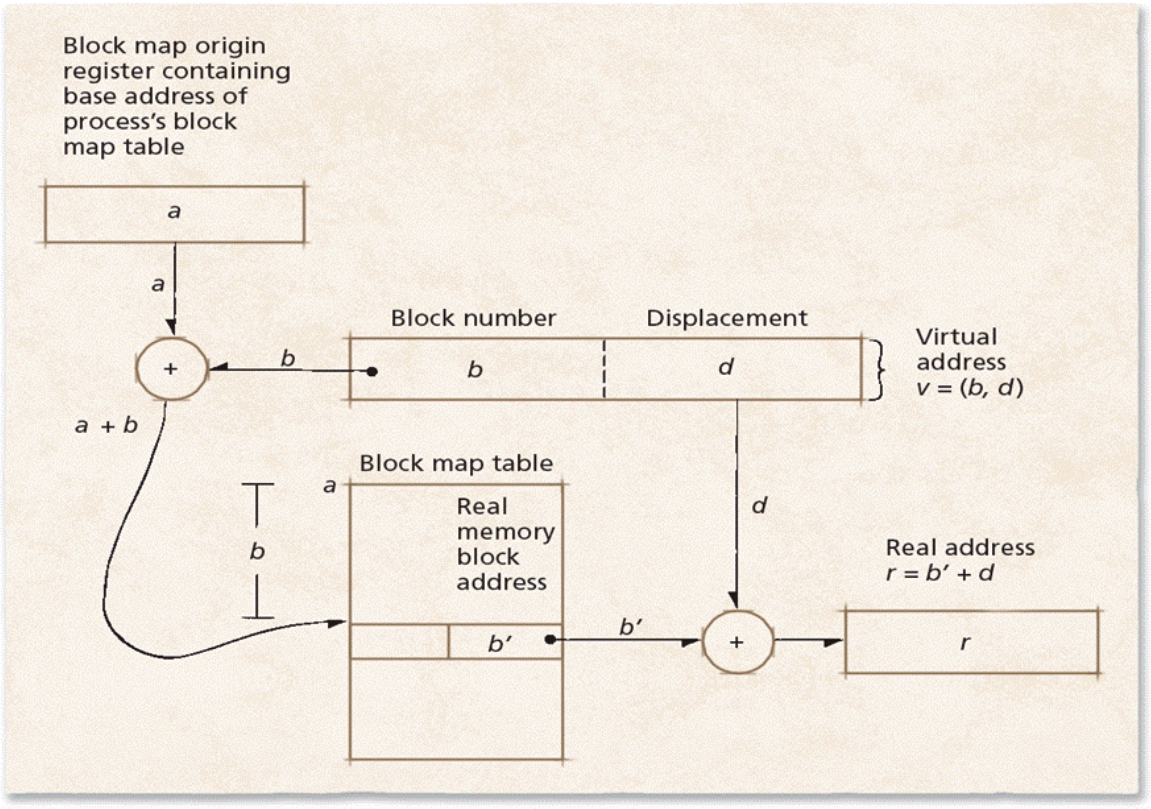


Рис. 5. Преобразование виртуальных адресов в физические в системе с блочным размещением

Ответы на вопросы

1. Да. Каждый процесс имеет свою таблицу размещения блоков. При переключении контекста система определяет реальный адрес, соответствующий адресу таблицы размещения блоков нового процесса в оперативной памяти.

2. Нет. Чем больше средний размер блока, тем меньше нужно хранить информации о размещении блоков. Однако использование больших блоков может привести к значительной внутренней фрагментации, кроме того, такие блоки требуют много времени на перемещение между оперативной памятью и вторичными устройствами хранения.

3. Нет. Если размер всех блоков одинаков, то блоки называют страницами. Если блоки могут быть произвольных размеров, они называются сегментами. В некоторых системах оба подхода совмещаются и сегменты состоят из страниц.

§ 3. Страничные системы

Страница (page) — определенного размера участок виртуального адресного пространства процесса, который управляется как единое целое. В страницах содержатся порции данных и/или кода процесса.

Рис. 6. Формат виртуального адреса в страничной системе

Кадр страницы (page frame) — блок оперативной памяти, в котором может размещаться страница виртуальной памяти. Страницу можно поместить в любой доступный кадр (см. рис. 7).

Страничные системы (paging systems) — системы виртуальной памяти, в которых она делится на фиксированного размера непрерывные блоки (см. рис. 8). В применении к виртуальному адресному пространству эти блоки называются страницами. В применении к реальному адресному пространству блоки называются кадрами страниц.

Страничные системы

- Страницы хранятся на вторичных устройствах хранения и при необходимости загружаются в кадры страниц в оперативной памяти
- Страничная организация упрощает решения о распределении памяти и не обладает внешней фрагментацией
- В страничных системах имеет место внутренняя фрагментация

Страничная таблица (page table или page map table) — таблица, в которой хранятся записи о номерах кадров, в которых размещаются страницы (см. рис. 9). В страничной таблице индексом является

номер виртуальной страницы, и такая таблица содержит по одной записи для каждой страницы процесса.

Запись в страничной таблице (Page Table Entry, PTE) — запись, в которой хранится номер кадра страницы, соответствующего странице виртуальной памяти. Кроме того, в этой записи хранится информация о том, находится ли эта страница в данный момент в памяти и разрешения на доступ к странице (см. рис. 10).

Вопросы для самопроверки

1. Требует ли механизм отображения страниц, чтобы p' и s хранились в отдельных ячейках PTE? (Да/Нет)

2. Страница и кадр страницы — это одно и то же? (Да/Нет)

3. Записи в страничной таблице должны быть одинаковыми по размеру? (Да/Нет)

4. Для трансляции адресов в страничных системах нужны специальные аппаратные устройства? (Да/Нет)

Ответы на вопросы

1. Нет. Чтобы уменьшить объем памяти, занимаемый PTE, многие системы используют только одно поле, значение которого воспринимается либо как номер кадра, либо как адрес на вторичном устройстве хранения в зависимости от значения бита резидентности.

2. Нет. Хотя страницы и кадры страниц одинаковы по размеру. Страница — это постоянный по размеру блок виртуального адресного пространства процесса. Кадр страницы — это постоянный по размеру блок оперативной памяти.

3. Да. Для реализации виртуальной памяти важно, чтобы трансляция адресов выполнялась как можно быстрее. Если записи в таблице одинаковы по размеру, процедура писка нужной записи проста, и трансляция адресов будет выполняться быстро.

4. Да. Для хранения базового адреса страничной таблицы нужен быстродействующий регистр в процессоре.

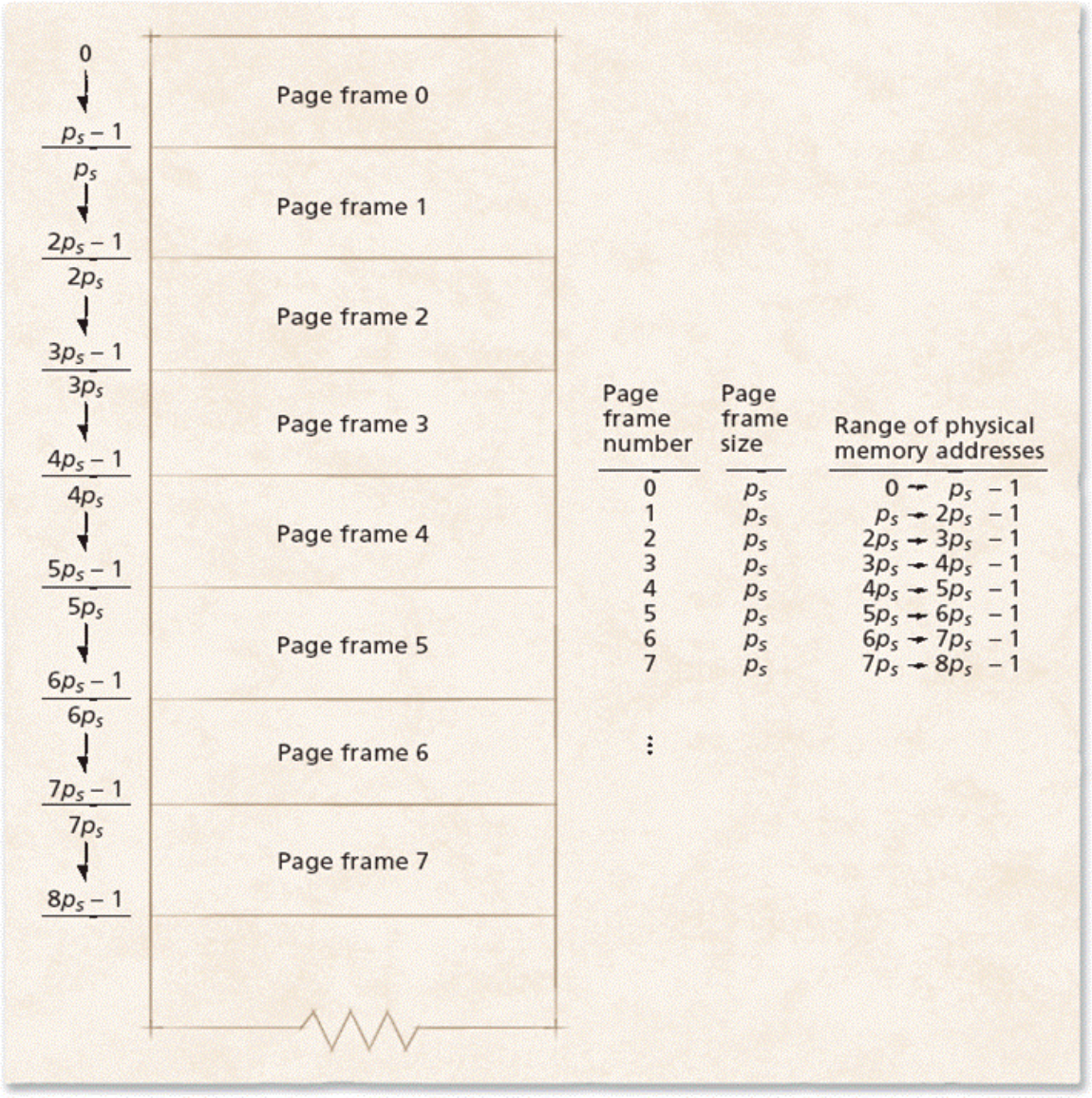


Рис. 7. Оперативная память, разделенная на кадры страниц

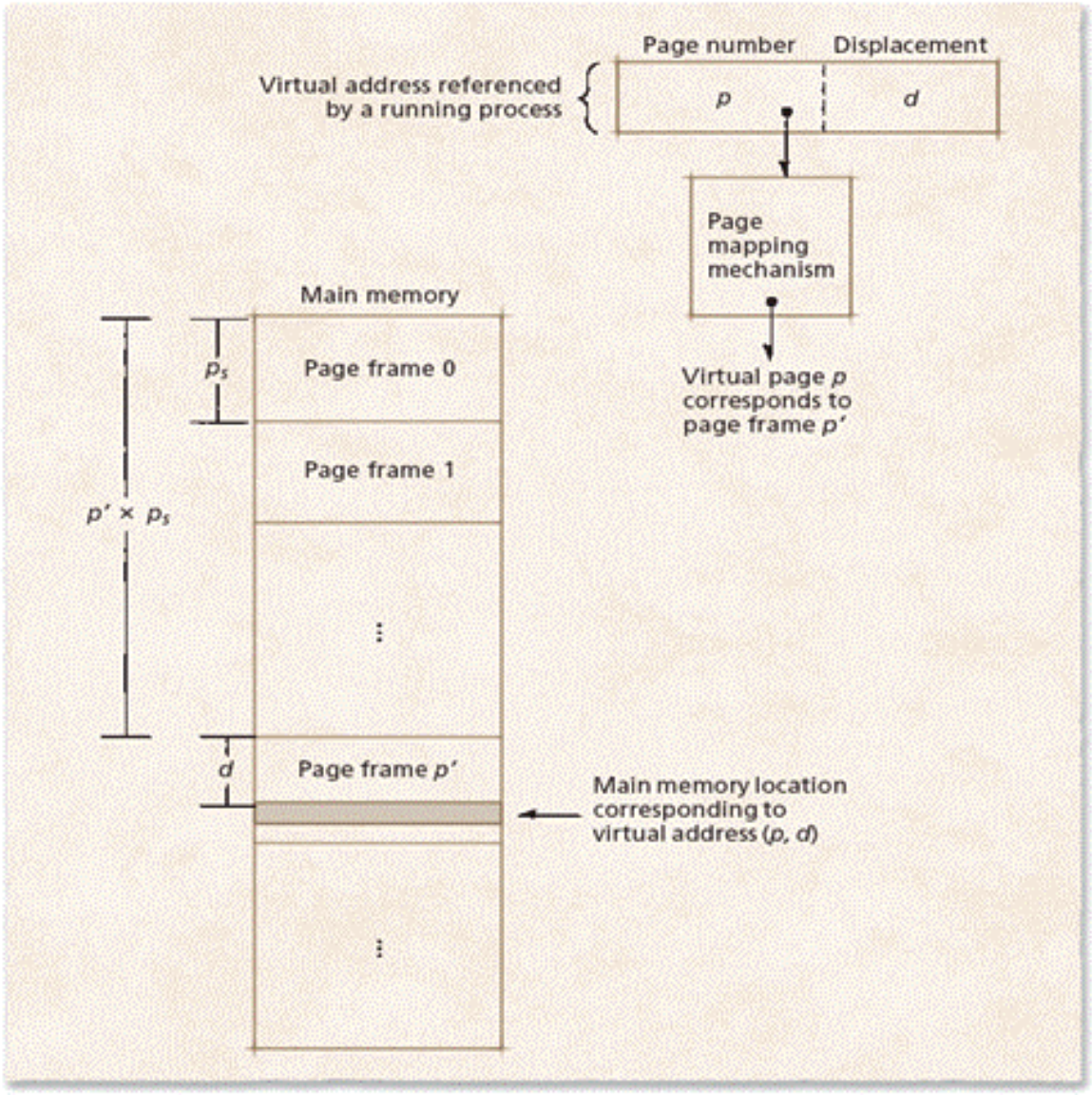


Рис. 8. Соответствие между адресами виртуальной и физической памяти в страничной системе

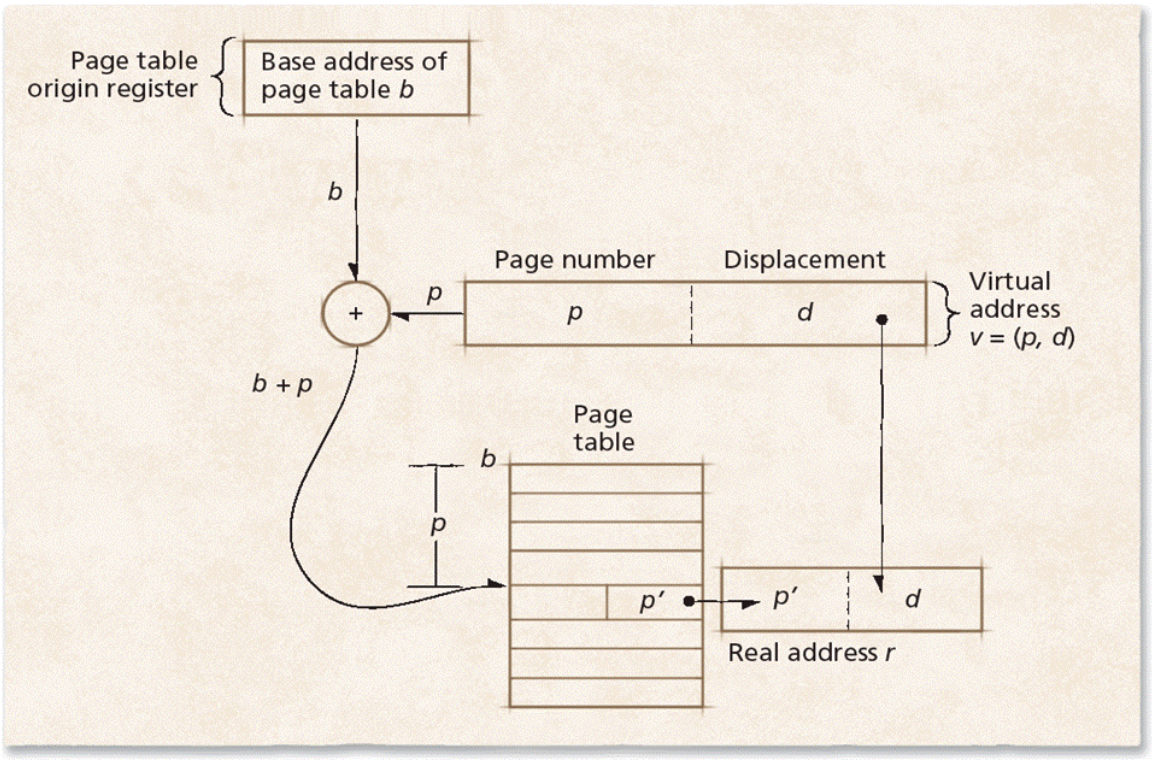


Рис. 9. Трансляция адресов в страничных системах

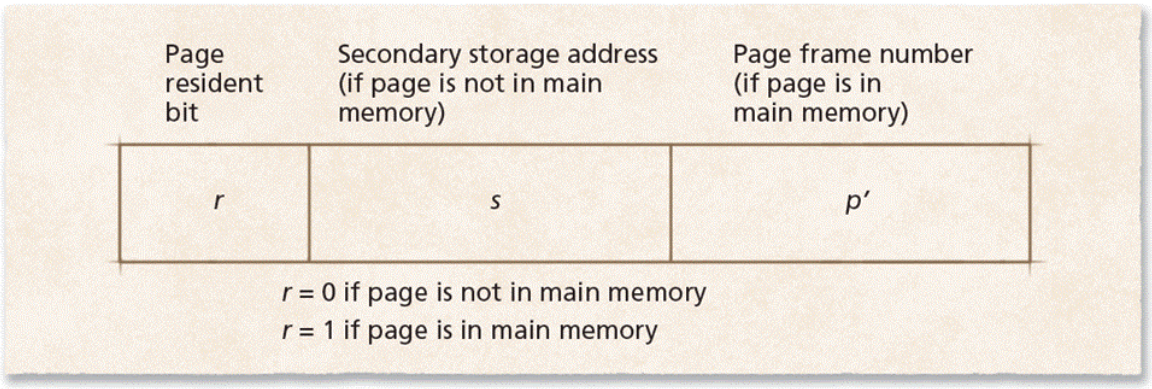


Рис. 10. Запись в страничной таблице

§ 4. Сегментация

Сегмент (segment) — переменного размера непрерывная область виртуального адресного пространства, управляемая как единое целое. Размер сегмента обычно соответствует размеру части программы — например, процедуры или массива, что позволяет системе защищать такие части, задавая четкие правила доступа к ним.

Сегменты

- Сегменту данных обычно назначаются права доступа только для чтения или для чтения и записи, но не для исполнения
- Для сегмента, содержащего исполняемые инструкции, обычно назначается доступ для чтения и исполнения, но не для записи
- Использование сегментов обычно приводит к внешней фрагментации оперативной памяти, но не к внутренней (см. рис. 11)

Системы виртуальной памяти с сегментацией

- В оперативной памяти хранятся только те сегменты, которые нужны программе для работы в данный момент
- Остальные сегменты могут находиться на вторичном устройстве хранения
- Если процесс обращается к памяти в сегменте, который находится на вторичном устройстве, система виртуальной памяти загружает его в основную память
- Сегменты переносятся из внешней памяти как неделимые единицы
- Все ячейки одного сегмента размещаются в непрерывной последовательности адресов в оперативной памяти
- Стратегии размещения сегментов аналогичны используемым в мультипрограммных системах с изменяемым распределением памяти

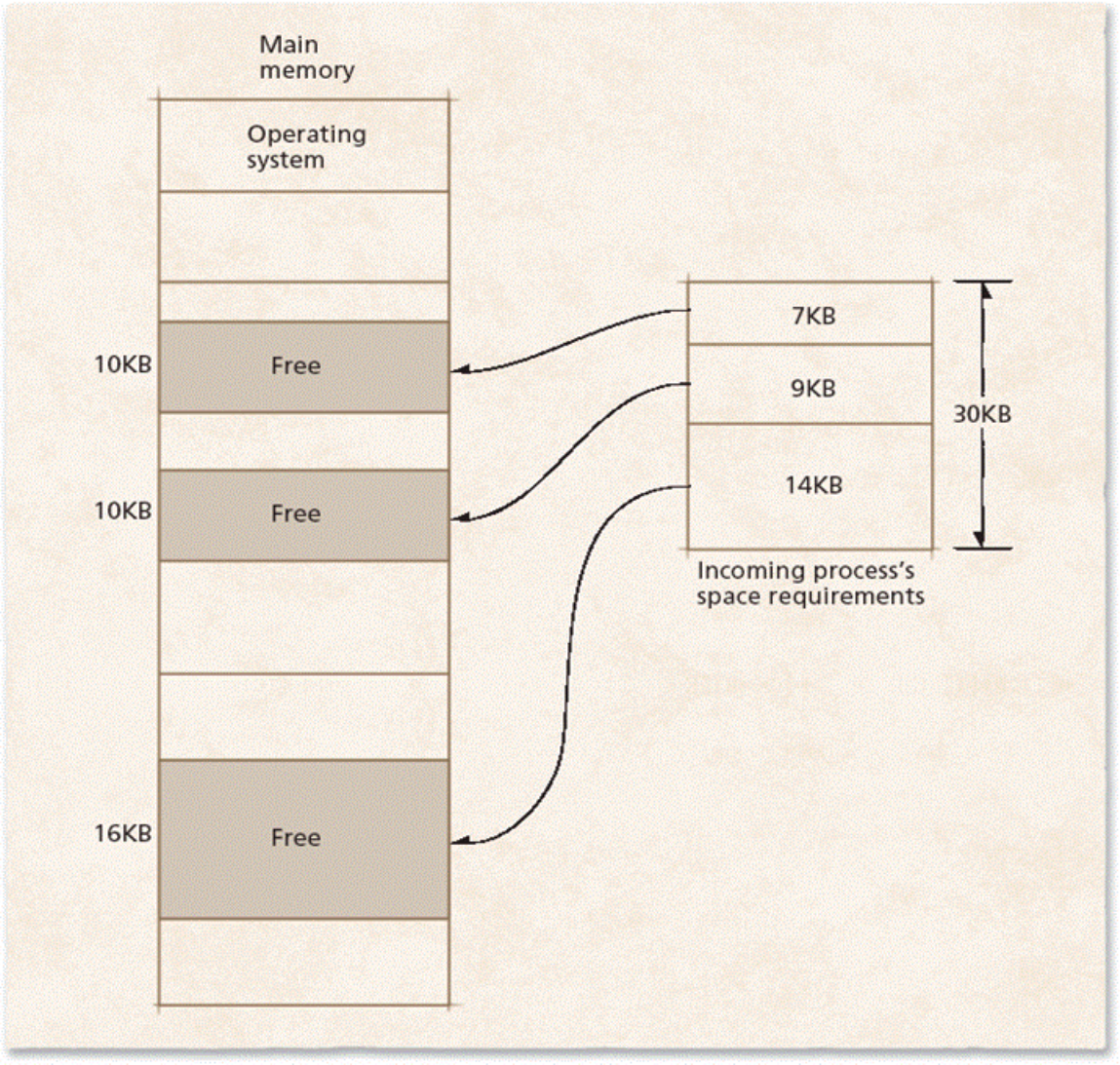


Рис. 11. Фрагментированное выделение памяти в системе реальной памяти с сегментацией

Рис. 12. Формат виртуального адреса в сегментной системе

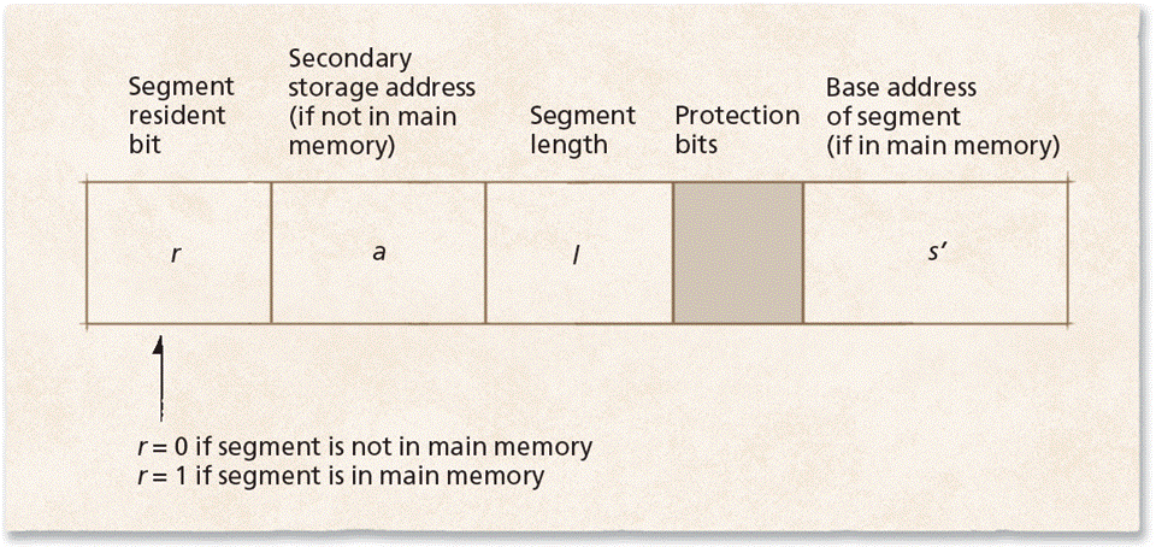


Рис. 13. Запись в сегментной таблице

Биты защиты (protection bits) — биты в записи в сегментной странице, значения которых ограничивают права доступа к сегменту (см. рис. 13).

Вопросы для самопроверки

1. В сегментных системах виртуальной памяти отсутствует фрагментация? (Да/Нет)

2. Сегментация отличается от мультипрограммности с изменяемым распределением памяти? (Да/Нет)

3. Требует ли механизм трансляции адресов в сегментной системе, чтобы значения s' и a хранились в отдельных ячейках записи в сегментной таблице? (Да/Нет)

Ответы на вопросы

1. Нет. В сегментных системах виртуальной памяти может возникнуть внешняя фрагментация, точно так же, как в мультипрограммных системах с изменяемым распределением памяти.

2. Да. В отличие от программ в мультипрограммных системах с изменяемым распределением памяти, программы в сегментных системах виртуальной памяти могут быть большего размера, чем оперативная память и находиться в нескольких областях памяти.

3. Нет. Чтобы уменьшить объем памяти, занимаемой сегментной

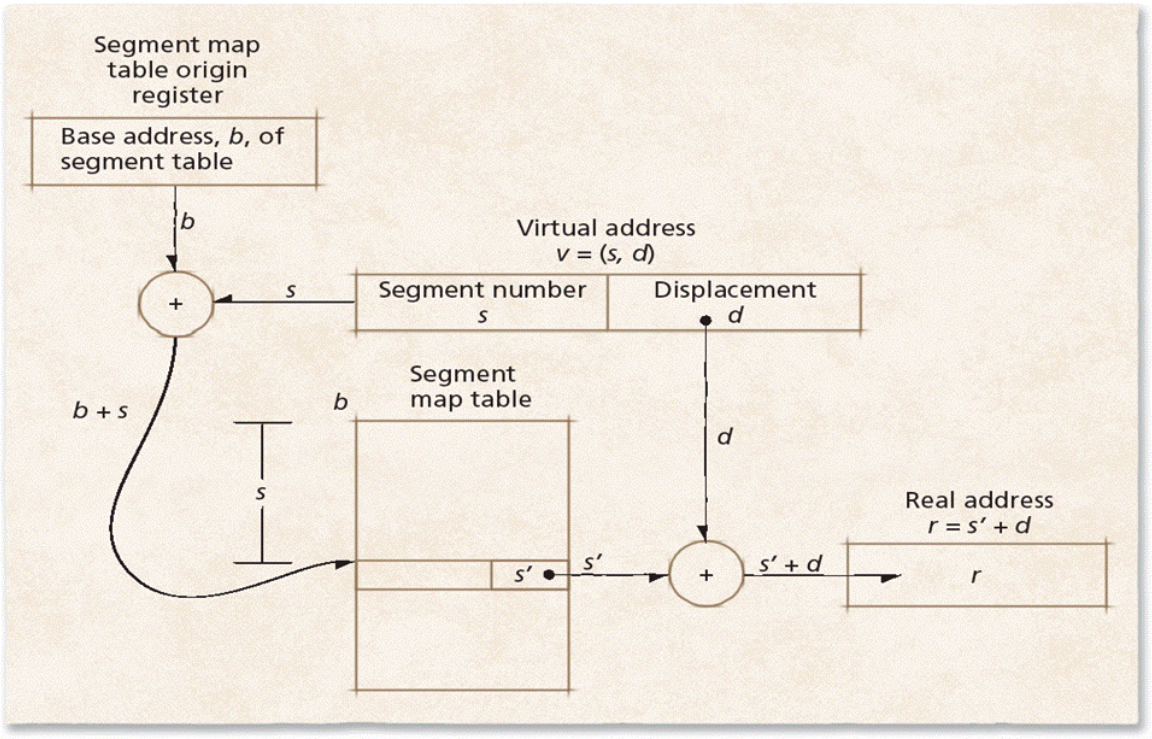


Рис. 14. Трансляция виртуального адреса в сегментной системе

таблицей (а она обычно хранится в кэше) эти параметры хранятся в одной ячейке содержимое которой интерпретируется в зависимости от значения бита резидентности.

§ 5. Контроль доступа в сегментных системах

Право доступа (access right) — право, определяющее, к каким ресурсам может обращаться программа. В памяти права доступа определяют, к каким сегментам может обращаться процесс и каким образом. При этом два процесса могут обращаться к одному сегменту (см. рис. 15).

Права доступа (см. рис. 16)

- Для чтения (read) — процесс может считывать данные из сегмента
- Для записи (write) — процесс может изменять содержимое сегмента

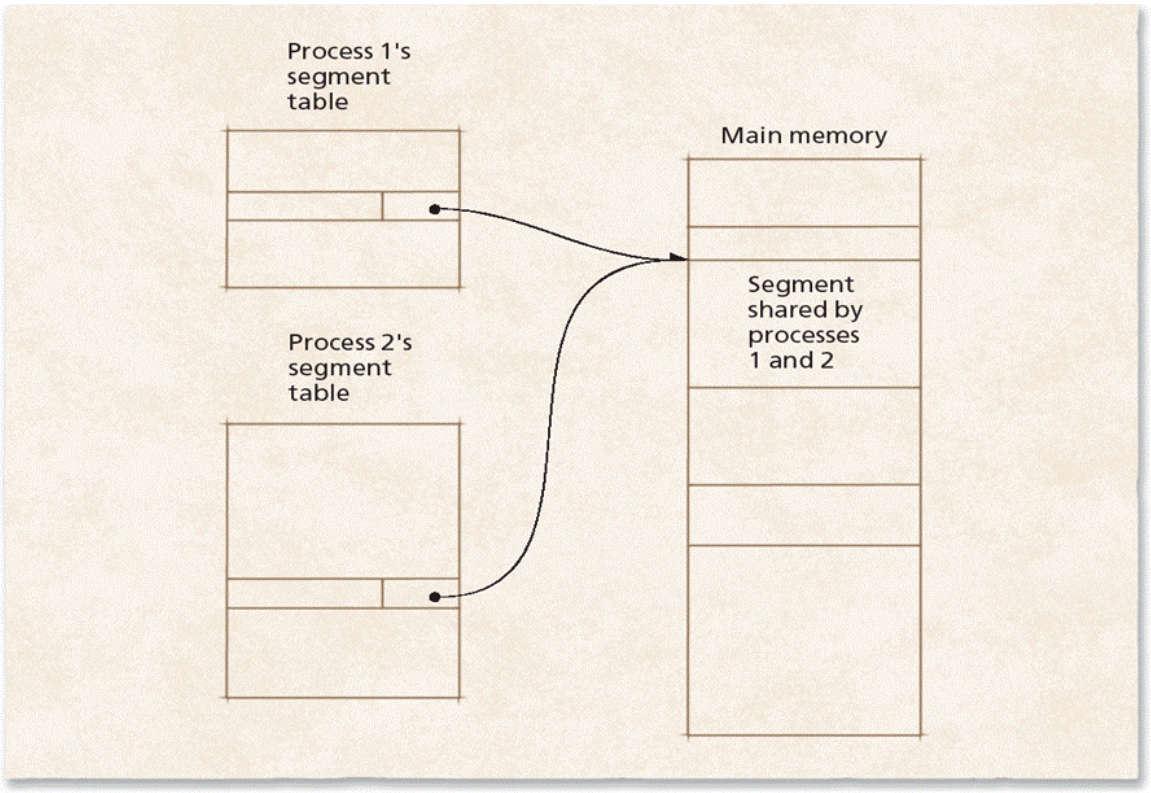


Рис. 15. Совместное использование ресурсов в сегментной системе

Type of access	Abbreviation	Description
Read	R	This segment may be read.
Write	W	This segment may be modified.
Execute	E	This segment may be executed.
Append	A	This segment may have information added to its end.

Рис. 16. Права доступа

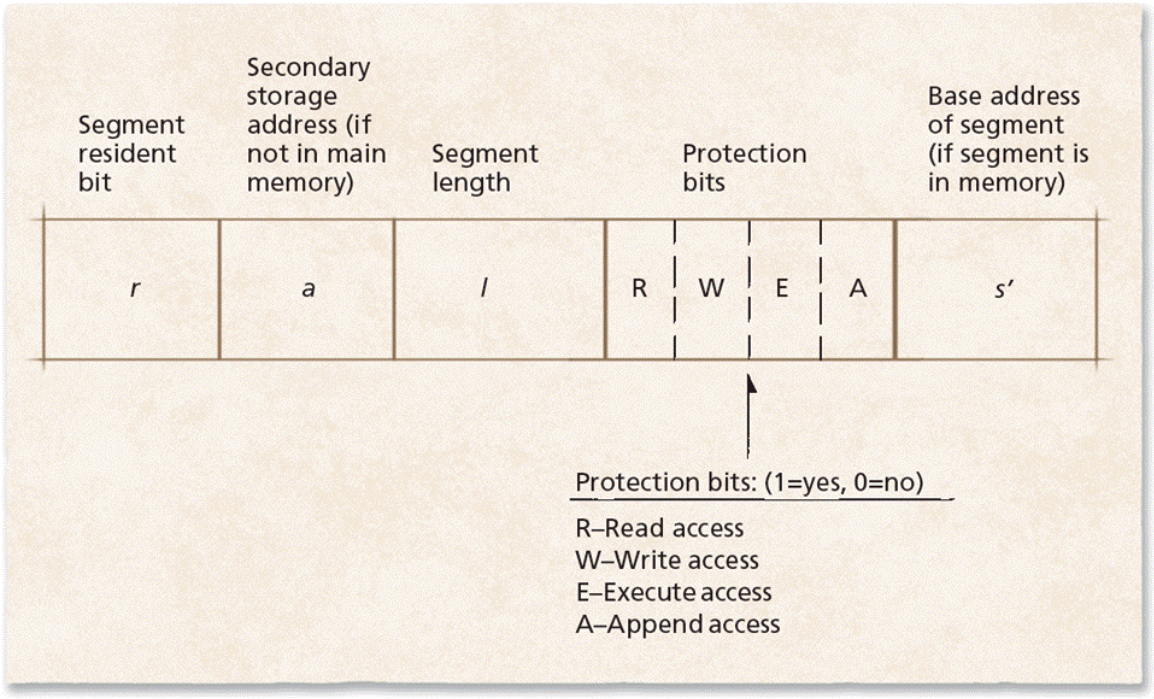


Рис. 17. Запись в сегментной таблице с битами защиты

- Для исполнения (execute) — процесс может исполнять инструкции, хранящиеся в сегменте
- Для дополнения (append) — процесс может дописывать в конец сегмента новые данные, но не изменять записанные ранее

Режим контроля доступа (access control mode) — набор прав (на чтение, на запись, на выполнение, на дополнение), которые определяют, каким образом можно обращаться к сегменту памяти. Может быть реализован с помощью записей в сегментной таблице с битами защиты (см. рис. 17).

Вопросы для самопроверки

1. Может ли один сегмент памяти использоваться несколькими процессами? (Да/Нет)

2. Существует 16 полезных режимов контроля прав доступа процессов к сегментам памяти? (Да/Нет)

Ответы на вопросы

1. Да. Один сегмент памяти может совместно использоваться несколькими процессами. Для этого сегментные таблицы этих процессов могут ссылаться на один и тот же сегмент.

2. Нет. По разному сочетая четыре права доступа можно создать 16 разных режимов контроля прав доступа. Некоторые из этих режимов весьма интересны, а другие — просто бессмысленны.

§ 6. Сегментно-страничные системы

- Используются в большинстве современных систем
- Сочетают преимущества сегментных и страничных систем
- Сегменты состоят из множества страниц
- Не обязательно всем страницам сегмента находиться в оперативной памяти одновременно

Рис. 18. Формат виртуального адреса в сегментно-страничной системе. Здесь segment number — номер сегмента, page number — номер страницы, displacement — смещение от начала страницы

Локальность (locality) — эмпирическая закономерность, связывающая близкие во времени или в пространстве события.

Пространственная и временная локальность

- В применении к последовательностям обращений к памяти пространственная локальность означает, что процесс, обращавшийся ранее по некоторому адресу, вероятно, будет обращаться и по близким к нему адресам

- Временная локальность означает, что процесс, обращавшийся недавно по какому-то адресу, вероятно, скоро обратится к нему снова

Буфер быстрой трансляции (Translation Lookaside Buffer, TLB) — схема быстродействующей ассоциативной памяти, в которой хранится небольшое количество записей о страницах виртуальной памяти и соответствующим им кадрам оперативной памяти (см. рис. 19). Обычно в TLB хранятся данные о станицах, к которым недавно были обращения, что повышает производительность процессов, обладающих локальностью.

Ассоциативная память (associative memory) — память, в которой поиск данных осуществляется по их значению, а не по адресу. Все записи в ассоциативной памяти просматриваются одновременно. Быстродействующая ассоциативная память помогает реализовать высокоскоростные механизмы динамической трансляции адресов.

Вопросы для самопроверки

1. Порождает ли циклическое обращение к массиву пространственную локальность? (Да/Нет)

2. Порождает ли циклическое обращение к массиву временную локальность? (Да/Нет)

3. Для сегментно-страничных систем необходимы специальные аппаратные устройства? (Да/Нет)

4. Сегментно-страничным системам присуща внешняя фрагментация? (Да/Нет)

5. Сегментно-страничным системам присуща внутренняя фрагментация? (Да/Нет)

Ответы на вопросы

1. Да. Циклическое обращение к массиву порождает пространственную локальность, поскольку элементы массива обычно расположены в непрерывной области виртуальной памяти.

2. Да. Временная локальность появляется потому, что элементы массива обычно намного меньше по размеру, чем страница. Поэтому обращения к двум соседним элементам массива, происходя за корот-

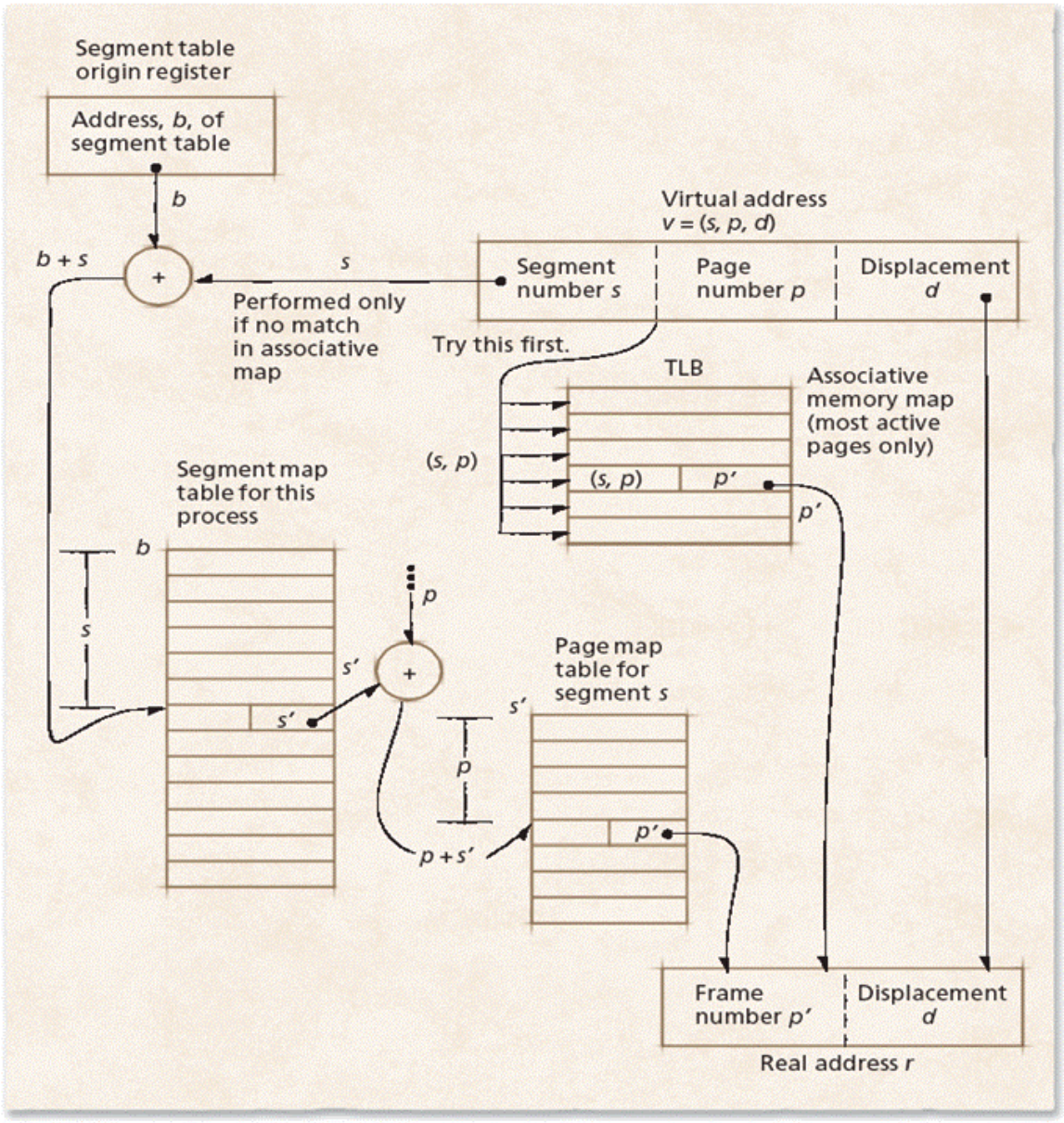


Рис. 19. Трансляция виртуальных адресов в сегментно-страничной системе

кий промежуток времени, обычно направляются к одной и той же странице.

3. Да. Для таких систем нужны быстродействующие регистры для хранения базовых адресов сегментной таблицы и соответствующей страничной таблицы, а также схема ассоциативной памяти (TLB).

4. Нет. В сегментно-страничных системах нет внешней фрагментации, поскольку, в конечном счете, вся память делится на кадры равного размера, в которых может помещаться любая страница.

5. Да. Это явление возникает тогда, когда объем сегмента меньше суммарного размера страниц, в которых он размещается. Чем меньше размер страниц, тем меньше памяти теряется на внутреннюю фрагментацию.

Глава 9

УПРАВЛЕНИЕ ВИРТУАЛЬНОЙ ПАМЯТЬЮ

§ 1. Подкачка по требованию

Подкачка по требованию (demand paging) — метод подкачки, при котором страницы по одной загружаются в оперативную память, только когда процесс явно обращается к ним.

Подкачка по требованию

- В начале выполнения процесса система загружает в оперативную память одну страницу, содержащую первую инструкцию процесса
- При обращении к новой странице процесс должен ожидать ее загрузки
- Чем больше страниц процесса уже загружено в оперативную память, тем эта задержка заметнее, так как тем большее пространство в оперативной памяти занимает простаивающий процесс

Пространственно-временной показатель (space-time product) — значение, равное произведению времени выполнения процесса (т.е. его нахождения в оперативной памяти) и объема оперативной памяти, занятого этим процессом (см. рис. 1). В идеале стратегии распределения памяти должны уменьшать этот показатель, чтобы повысить степень мультипрограммности.

Вопросы для самопроверки

1. Подкачка по требованию может увеличить степень мультипрограммности системы? (Да/Нет)

2. Подкачка по требованию может уменьшить степень мультипрограммности системы? (Да/Нет)

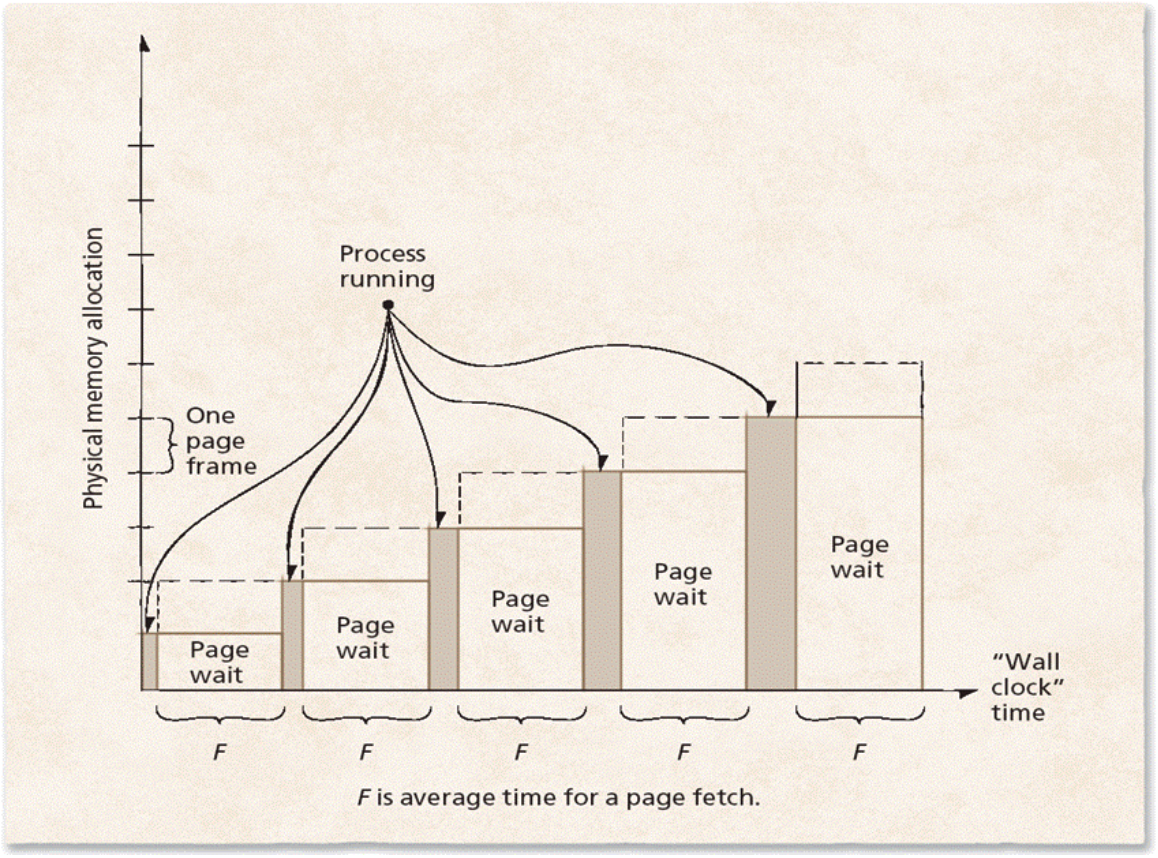


Рис. 1. Пространственно-временной показатель при подкачке по требованию. Заштрихованные области обозначают показатель при выполнении продуктивных операций, не заштрихованные — показатель ожидания загрузки страниц

Ответы на вопросы

1. Да. Подкачка по требованию может увеличить степень мультипрограммности, поскольку система будет загружать в память только страницы, действительно требующиеся процессам. Поэтому в физической памяти сможет разместиться большее количество процессов.

2. Да. Пока операционная система извлекает страницы со вторичного устройства хранения, память, занятая процессом, не используется, и степень мультипрограммности может уменьшиться.

§ 2. Предварительная подкачка

Предварительная подкачка (anticipatory paging, prepaging или prefetching) — метод, загружающий в оперативную память страницы

процесса, к которым вероятны обращения в ближайшем будущем. Если система правильно выберет страницы для загрузки, общее время работы процесса может уменьшиться.

Факторы, определяющие успех стратегий предварительной загрузки:

- объем памяти, доступный для размещения предварительно загружаемых страниц;
- количество страниц предварительно загружаемых за один заход;
- алгоритм выбора предварительно загружаемых страниц (с использованием временной или пространственной локальности).

Предварительная подкачка

- Если процесс обращается к странице, которой нет в оперативной памяти, система загружает ее и несколько страниц, расположенных поблизости от нее в виртуальной памяти
- Необходимо компактное размещение на вторичных устройствах страниц, смежных в виртуальном адресном пространстве процесса
- Разница между временем загрузки нескольких страниц, расположенных рядом на диске, и временем загрузки одной страницы невелика
- Предварительная подкачка может использоваться без заметного увеличения задержек по сравнению с подкачкой по требованию
- Linux загружает зразу 8 смежных страниц

Вопросы для самопроверки

1. Может ли стратегия предварительной подкачки Linux оказаться неэффективной? (Да/Нет)

2. С учетом локальности подкачка по требованию эффективнее предварительной подкачки? (Да/Нет)

Ответы на вопросы

1. Да. Если процесс обращается к страницам в случайной последовательности, то Linux, вероятно, будет загружать в память страницы не нужные процессы и засорять ее.

2. Нет. Локальность приводит к большей эффективности предварительной подкачки, поскольку операционная система сможет с высокой вероятностью предугадать, какие страницы процесс будет использовать.

§ 3. Стратегия замены страниц FIFO

Стратегия замены страниц (page-replacement strategy) — стратегия, определяющая, какие страницы убрать из оперативной памяти, чтобы освободить место для требуемых в данный момент. Стратегии замены страниц пытаются оптимизировать производительность за счет предугадывания дальнейшего использования страниц.

Стратегия замены страниц FIFO (FIFO page-replacement strategy) — стратегия замены страниц, заменяющая страницу, которая дольше всего находилась в памяти (см. рис. 2). Эта стратегия связана с низкими накладными расходами, но обычно неточно предсказывает будущие обращения к страницам.

Страничный промах (missing-page fault) — ситуация, возникающая при обращении к странице, которая в данный момент не загружена в оперативную память. Если происходит такой промах, операционная система должна загрузить требуемую страницу со вторичного устройства хранения.

Вопросы для самопроверки

1. Стратегия замены страниц FIFO обладает высокой производительностью? (Да/Нет)

2. Стратегия замены страниц FIFO редко используется на практике? (Да/Нет)

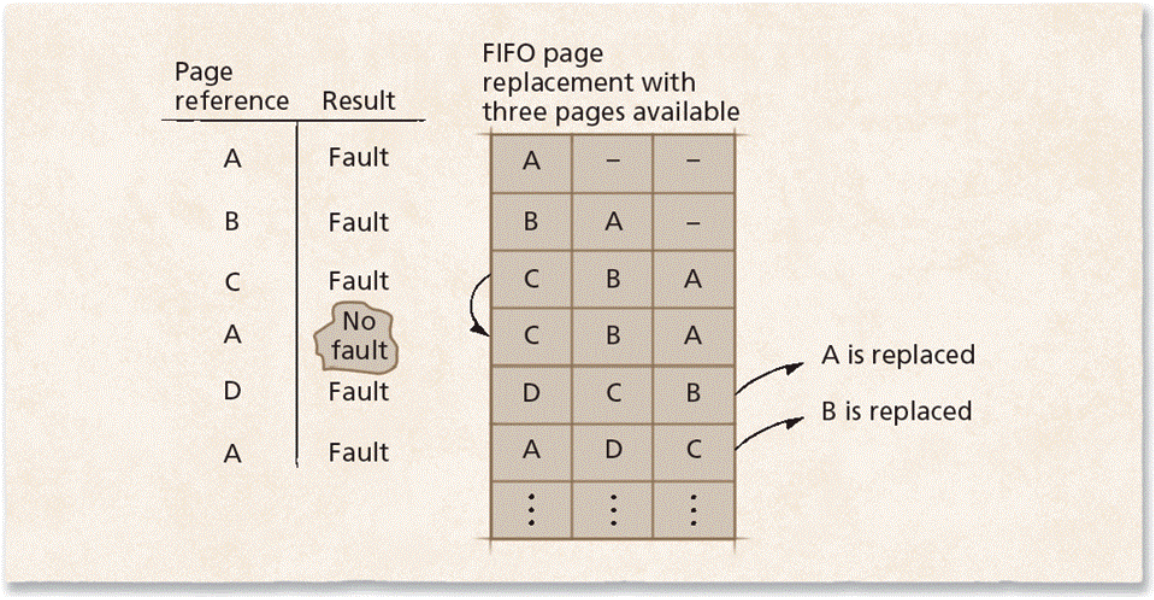


Рис. 2. Стратегия замены страниц FIFO

Ответы на вопросы

1. Нет. Стратегия FIFO заменяет страницы согласно их возрасту, который в отличие от локальности является плохой основой для предсказания будущих обращений к страницам.

2. Да. Но ее модификации служат основой различных практически используемых стратегий замены страниц.

§ 4. Стратегия замены дольше всего не использовавшихся страниц (LRU)

Стратегия замены дольше всего не использовавшихся страниц (Least Recently Used page replacement strategy, LRU) — стратегия замены страниц, заменяющая страницу, к которой дольше всего не обращался процесс (см. рис. 3). Эта стратегия обычно хорошо предсказывает будущие обращения к страницам, но она связана со значительными накладными расходами.

Стратегия замены страниц LRU

- Можно реализовать с помощью списка, содержащего по одной записи для каждого занятого кадра оперативной памяти

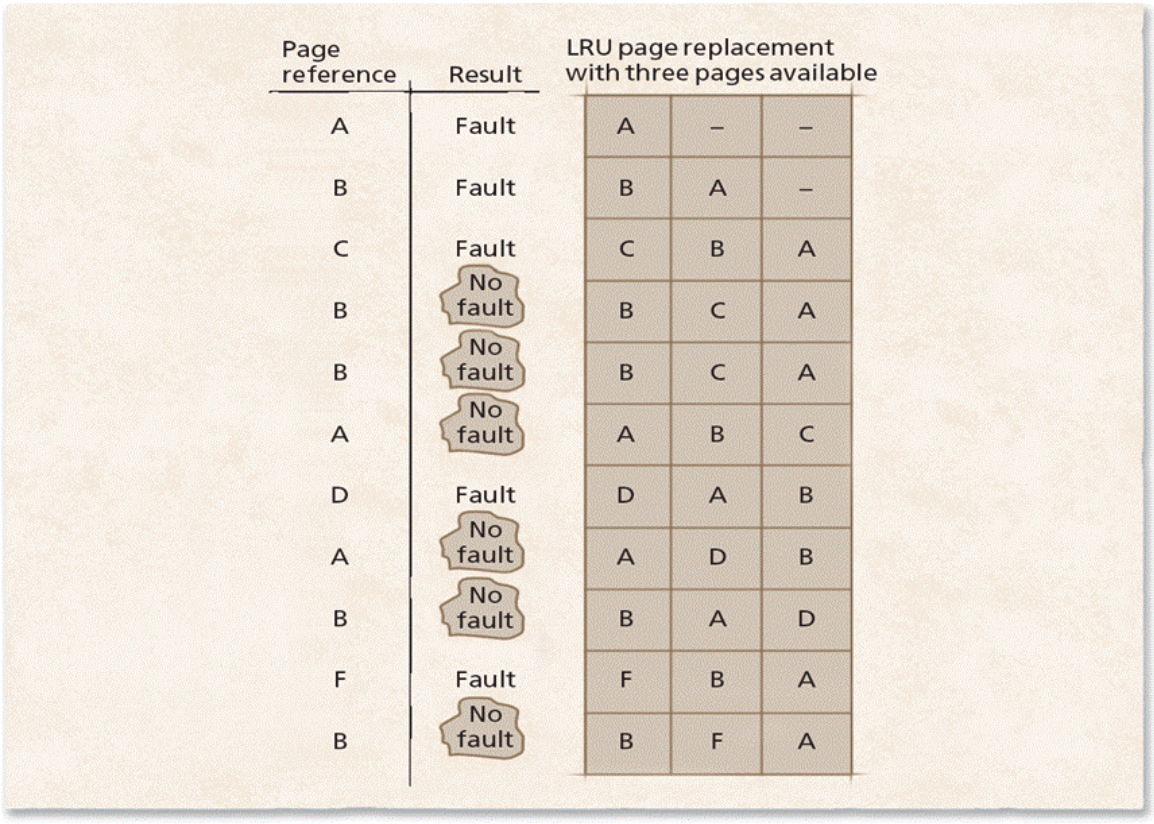


Рис. 3. Стратегия замены страниц LRU

- При каждом обращении к странице памяти LRU будет помещать запись об этой странице в голову списка
- Более старые записи будут отодвигаться к концу списка
- Когда нужно заменить страницу на новую, страница, подлежащая замене, выбирается из хвоста списка
- Система освобождает соответствующий кадр страницы и помещает в освободившийся кадр новую страницу
- Запись об этом кадре перемещается в голову списка, поскольку к странице в нем обращались последней

Вопросы для самопроверки

1. Стратегия LRU повышает скорость выполнения процессов с пространственной локальностью? (Да/Нет)

2. Стратегия LRU редко используется на практике? (Да/Нет)

Ответы на вопросы

1. Нет. Эта стратегия повышает скорость выполнения процессов с временной локальностью.

2. Да. Потому что она требует значительных накладных расходов на поддержание упорядоченного списка страниц и его переупорядочивание.

§ 5. Стратегия замены давно не используемых страниц (NUR)

Бит изменения (modified bit) — поле в записи страничной таблицы, значение которого указывает, изменялось ли содержимое страницы. Если этот бит установлен, то перед заменой этой страницы на новую ее содержимое нужно скопировать на вторичное запоминающее устройство.

Бит обращения (referenced bit) — поле в записи страничной таблицы, значение которого указывает были ли обращения к странице. Если этот бит установлен, это означает, что обращения были.

Стратегия замены давно не используемых страниц (Not Used Recently page replacement strategy, NUR) — аппроксимация стратегии замены страниц LRU, требующая меньших накладных расходов при использовании.

Стратегия замены страниц NUR (см. рис. 4)

- В зависимости от значений битов изменения и обращения все страницы делятся на четыре группы
- Страницы группы 1 — лучшие кандидаты на замену, а группы 4 — худшие
- При наличии страниц в группе с меньшим номером, страница для замены выбирается из этой группы случайным образом
- Группа 2 измененных страниц, к которым не было обращений возникает потому, что NUR периодически сбрасывает биты обращений для всех страниц (при интенсивной работе системы они все через некоторое время становятся равными 1)

Group	Referenced	Modified	Description
Group 1	0	0	Best choice to replace
Group 2	0	1	[Seems unrealistic]
Group 3	1	0	
Group 4	1	1	Worst choice to replace

Рис. 4. Стратегия замены страниц NUR: в зависимости от значений битов изменения и обращения все страницы делятся на четыре группы

Вопросы для самопроверки

1. Бит изменения повышает производительность стратегии NUR? (Да/Нет)

2. Может NUR заменить хуже всего подходящую для замены страницу? (Да/Нет)

3. Может ли оказаться, что содержимое страницы изменялось, а обращений к ней не было? (Да/Нет)

Ответы на вопросы

1. Да. Бит изменения позволяет операционной системе определить, какие страницы можно заменять без предварительного сброса их на диск. Выбор для замены страниц неизменного содержимого позволяет уменьшить число операций ввода/вывода при замене страниц.

2. Да. Если бит обращения страницы, к которой сейчас будут обращения, был сброшен непосредственно перед моментом, когда нужно заменить страницу.

3. Да. Такое может быть в стратегии NUR. На самом деле обращения к этой странице были, но биты обращений периодически сбрасываются в ноль.

§ 6. Замена страниц в Linux

Рабочий набор (working set) — набор часто используемых процессом страниц виртуальной памяти. Практические исследования показывают, что программы часто проявляют пространственную ло-

кальность, обращаясь на разных фазах выполнения к определенным рабочим наборам (см. рис. 5).

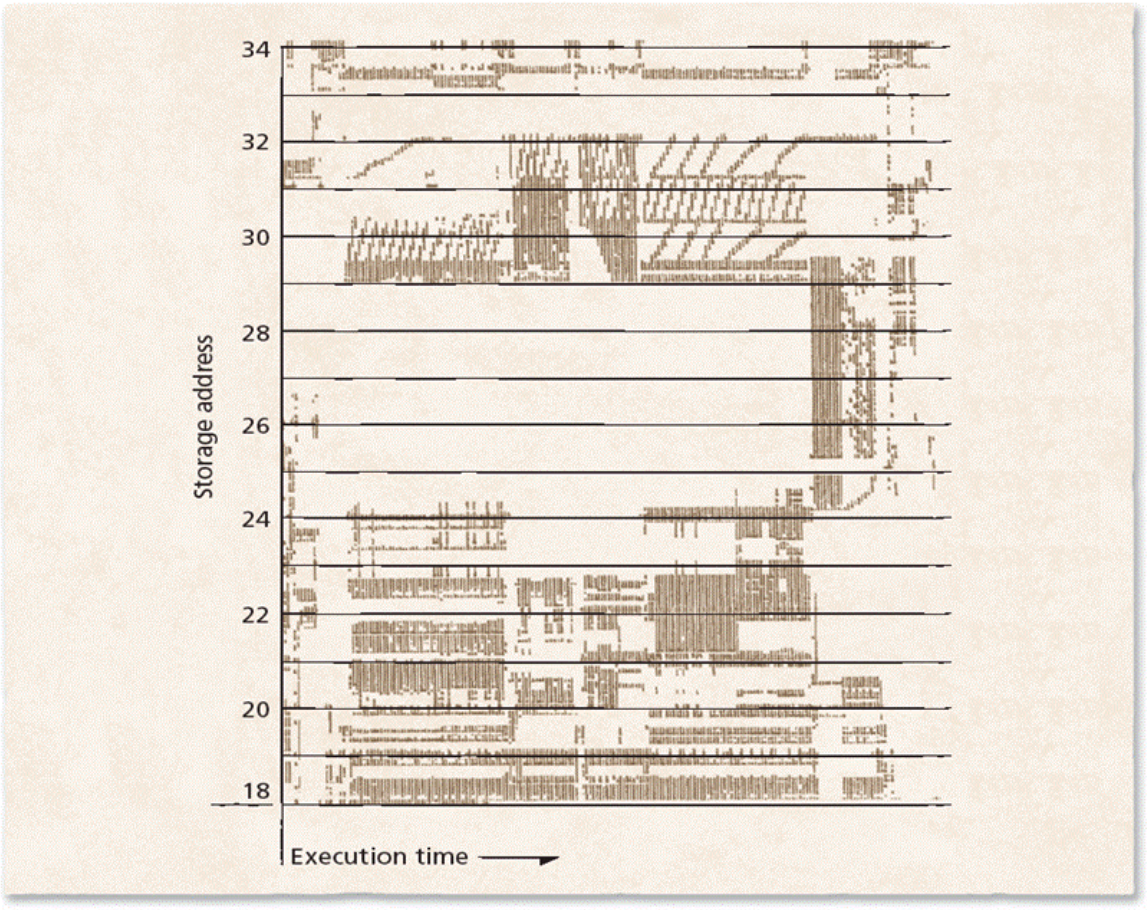


Рис. 5. Последовательность обращений к памяти, проявляющая локальность

Замена страниц в Linux

- Страницы делятся на активные и не активные
- Чтобы считаться активной страница должна недавно получить обращение
- Одна из задач системы управления памятью — хранить текущий рабочий набор в активных страницах
- Страницы, использовавшиеся позже всего, находятся поблизости от головы активного списка

- Страницы, дольше всего не использовавшиеся, находятся поблизости от хвоста неактивного списка

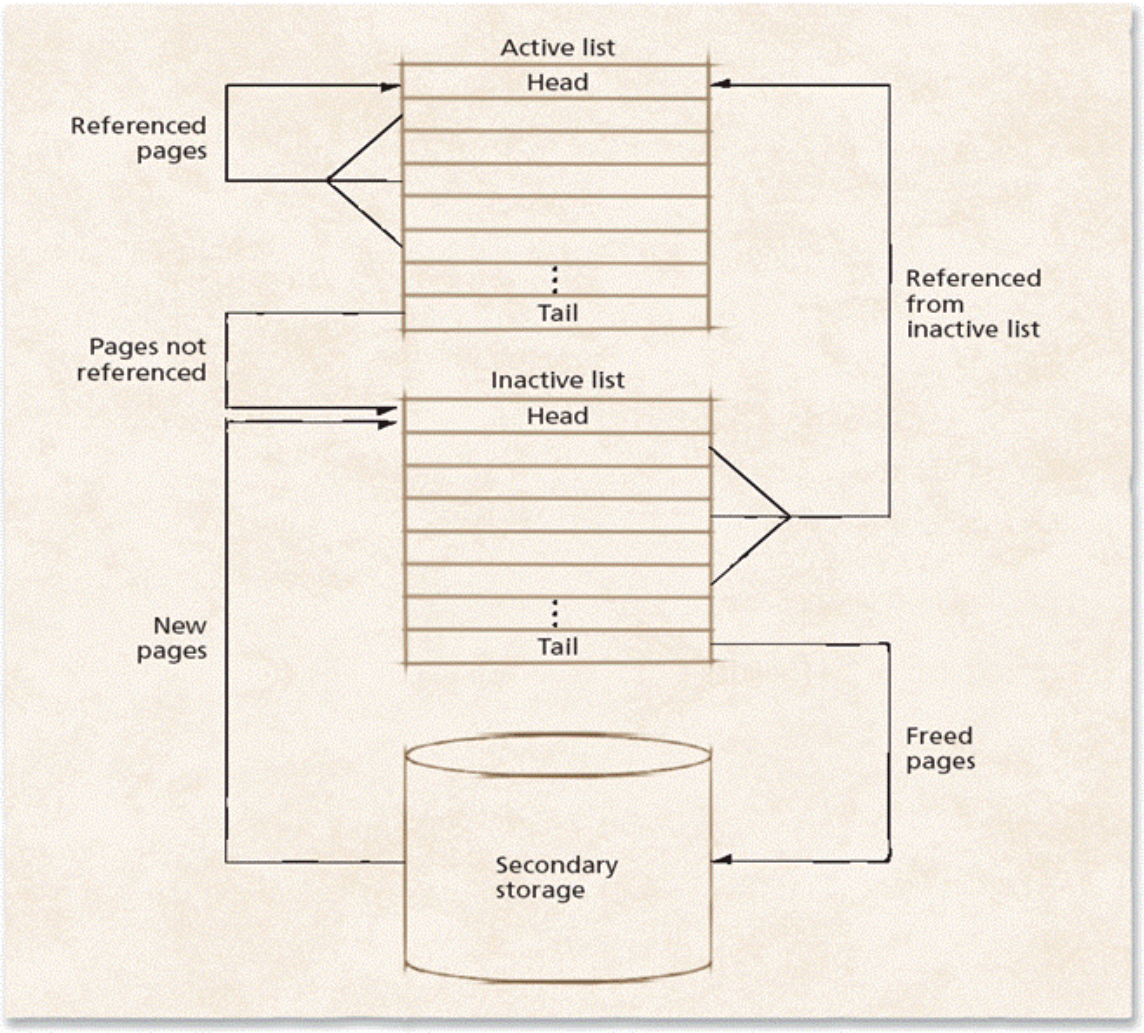


Рис. 6. Замена страниц в Linux

Алгоритм замены страниц в Linux (см. рис. 6):

1) когда страница впервые загружается в оперативную память, она помещается в неактивный список и ее бит обращения устанавливается в 1;

2) если страница неактивна и ее бит обращения уже установлен в 1, а к ней поступает обращение, то диспетчер переносит ее в голову активного списка и сбрасывает ее бит обращения;

3) страницы из активного списка, к которым поступают обращения, переносятся в голову этого списка, их бит обращения устанавливается в 1;

4) страницы из хвоста активного списка периодически переносятся в голову неактивного списка; пока страницы находятся в активном списке их нельзя заменить;

5) для замены выбираются страницы из хвоста неактивного списка.

Вопросы для самопроверки

1. Верно ли, что в активном списке Linux находятся страницы, которым было более чем одно обращение? (Да/Нет)

2. Можно ли заменить страницу из активного списка Linux? (Да/Нет)

3. Всегда ли при обращении страница из неактивного списка Linux переносится в активный? (Да/Нет)

Ответы на вопросы

1. Да. В активный список страницы попадают из неактивного, следовательно, как минимум после второго обращения к ним.

2. Нет. Для замены выбираются страницы из хвоста неактивного списка.

3. Нет. Если страница неактивна и ее бит обращения равен 0, а к ней поступает обращение, то диспетчер переносит ее в голову неактивного списка и устанавливает ее бит обращения в 1. Это гарантирует, что недавно использовавшиеся страницы не будут выбраны для замены.

§ 7. Размер страниц

Аргументы за большие страницы

- Чем больше страницы, тем их меньше
- Меньший размер имеют страничные таблицы
- Больше вероятность попадания записи о станице в буфер быстрой трансляции TLB

- Необходимо меньше операций ввода/вывода при замене страниц

Аргументы за маленькие страницы

- Страницы меньшего размера могут позволить процессу создать более компактный рабочий набор
- Чем меньше страницы, тем меньше внутренняя сегментация
- В сегментно-страничных системах каждый сегмент, в среднем, расходует половину одной страницы на внутреннюю фрагментацию (см. рис. 7)

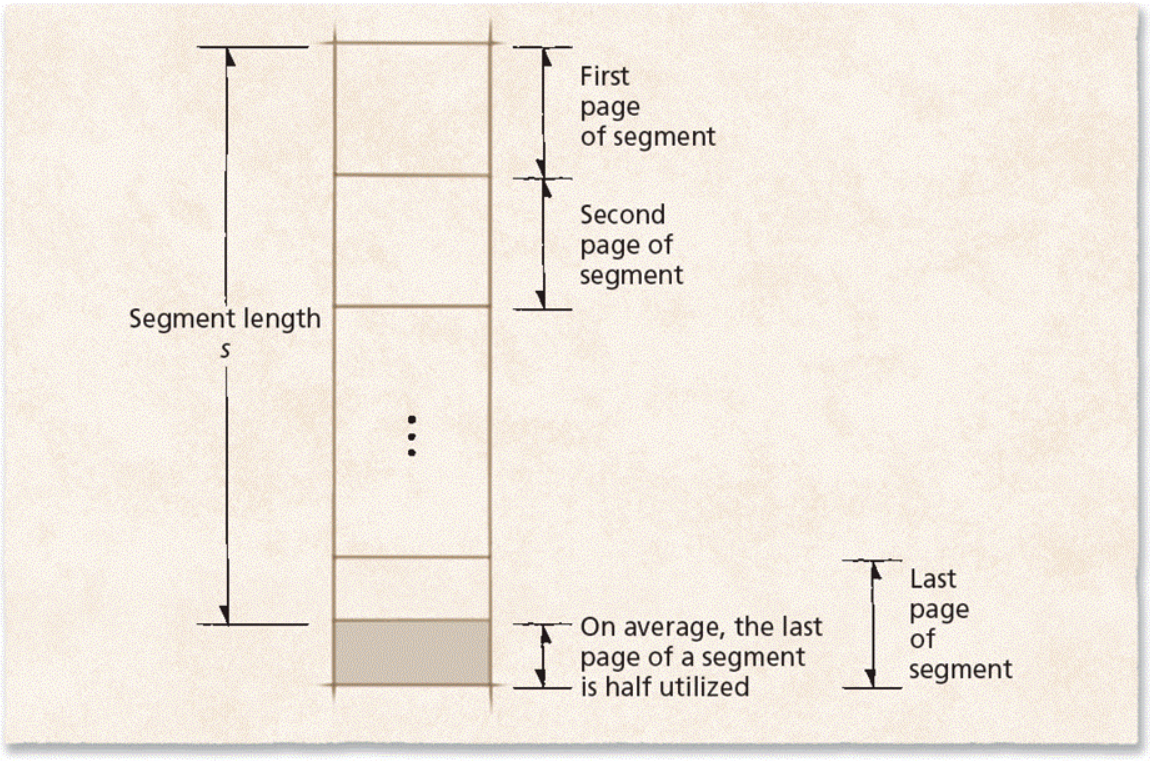


Рис. 7. Внутренняя фрагментация в сегментно-страничных системах

Вопросы для самопроверки

1. В современных системах используются страницы большого размера? (Да/Нет)

2. В современных системах используются страницы небольшого размера? (Да/Нет)

Manufacturer	Model	Page Size	Real address size
Honeywell	Multics	1KB	36 bits
IBM	370/168	4KB	32 bits
DEC	PDP-10 and PDP-20	512 bytes	36 bits
DEC	VAX 8800	512 bytes	32 bits
Intel	80386	4KB	32 bits
Intel / AMD	Pentium 4 / Athlon XP	4KB or 4MB	32- or 36 bits
Sun	UltraSparc II	8KB, 64KB, 512KB, 4MB	44 bits
AMD	Opteron / Athlon 64	4KB, 2MB and 4MB	32, 40, or 52 bits
Intel-HP	Itanium, Itanium 2	4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB	Between 32 and 63 bits
IBM	PowerPC 970	4KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB	32 or 64 bits

Рис. 8. Размеры страниц в разных процессорных архитектурах

Ответы на вопросы

1. Да. Память дешевеет, и появляется возможность использовать страницы большого размера.

2. Да. Поддержка страниц разных размеров позволяет эффективнее использовать память ценой дополнительных затрат на ее распределение. Например, необходимо иметь столько буферов быстрой трансляции TLB, сколько размеров страниц поддерживает система.

Часть IV

Файловые системы и базы данных

Глава 10

ФАЙЛЫ И ФАЙЛОВЫЕ СИСТЕМЫ

§ 1. Иерархия данных

Иерархия данных (data hierarchy) — классификация, группирующая различные последовательности битов для представления осмысленных значений.

Иерархия данных

- Последовательности битов, байты и слова содержат небольшие количества битов, интерпретируемые аппаратными устройствами и низкоуровневыми программами
- Поля, записи и файлы могут содержать множества битов, интерпретируемые операционными системами и пользовательскими приложениями

Сочетания битов (bit patterns) — нижний уровень иерархии данных. Битовые последовательности составляющие двоичные коды, используются для представления всех данных в компьютерных системах. В последовательности из n битов можно хранить 2^n различных сочетаний битов.

Байт (byte) — второй снизу уровень иерархии данных. Обычно байт состоит из 8 битов.

Слово (world) — последовательность битов, которую может одновременно обрабатывать процессор(ы) системы. В иерархии данных слова располагаются на уровень выше байтов.

Пример. Слово состоит из 4 байтов для 32-разрядного процессора и 8 байтов для 64-разрядного.

Символ (character) — в иерархии данных — последовательность битов фиксированной длины, обычно — 8, 16 или 32 бита.

Набор символов (character set) — таблица, содержащая определенное конечное множество символов. К популярным наборам символов относятся ASCII, EBCDIC и Unicode.

ASCII (American Standard Code for Information Interchange, Американский стандартный код обмена информацией) — набор символов, широко применяемый в персональных компьютерах и системах обмена данными, хранящий символы в 8-битовых байтах. В этом наборе может быть до 256 различных символов. Из-за этого ограничения в нем не поддерживаются международные наборы символов.

EBCDIC (Extended Binary-Coded Decimal Interchange Code, Расширенный двоично-десятичный код обмена информацией) — восьмибитовый набор символов, используемый для представления данных в больших компьютерах, особенно производства IBM.

Unicode — набор символов, поддерживающий международные кодировки и широко использующийся в Интернете и многоязычных приложениях. В Unicode есть 8-, 16- и 32-битные форматы представления символов.

Поле (field) — в иерархии данных — группа символов (например, имя человека, его адрес или номер телефона).

Запись (record) — в иерархии данных — группа полей (например, все поля, хранящие информацию о конкретном покупателе или студенте).

Файл (file) — именованный набор данных, который может обрабатываться как единое целое с помощью таких операций, как открытие, закрытие, чтение, запись, удаление, копирование и переименование. Отдельные элементы данных в файле могут подвергаться, например, операциям чтения, записи, обновления, вставки и удаления. Файлы могут состоять из одной или более записей.

Том (volume) — часть пространства накопителя, в которой может храниться множество файлов.

Вопросы для самопроверки

1. Большие наборы символов лучше маленьких? (Да/Нет)

2. Реализованы ли 64-битовые наборы символов? (Да/Нет)

Ответы на вопросы

1. Нет. Большие наборы символов, например, Unicode, позволяют хранить и передавать данные на множестве различных языков. Однако в больших наборах символов для представления одного символа используется большое количество битов, и объем хранения данных возрастает.

2. Нет. На сегодняшний день 64-битовые наборы символов еще не реализованы, поскольку они потребуют значительных объемов для хранения каждого символа, и позволяют представлять намного больше различных символов, чем может потребоваться в обозримом будущем.

§ 2. Файлы

Операции с файлами

- Открытие (open file) — подготовка файла к обращениям
- Закрытие (close file) — блокирование дальнейших обращений к файлу до нового открытия
- Создание (create file) — создание нового файла
- Уничтожение (destroy file) — удаление файла
- Копирование (copy file) — копирование содержимого файла в другой файл
- Переименование (rename file) — изменение имени файла
- Отображение (list file) — вывод содержимого файла на экран или печать

Операции с элементами данных, хранящимися в файлах

- Чтение (read data) — копирование данных из файла в память процесса

- Запись (write data) — копирование данных из памяти процесса в файл
- Обновление (update data) — изменение содержимого существующего элемента данных в файле
- Вставка (insert data) — добавление в файл нового элемента данных
- Удаление (delete data) — удаление элемента данных из файла

Свойства, характеризующие файлы

- Размер (size of file) — количество данных, хранящихся в файле
- Расположение (location of file) — место, где хранится файл (на накопителе или в логической структуре файлов системы)
- Режим доступа (accessibility of file) — ограничения на доступ к файлу
- Тип (type of file) — назначение файла (например, исполняемый файл содержит исполняемые инструкции для процесса, а для файла данных может быть указано приложение, предназначенное для работы с его содержимым)
- Изменчивость (volatility of file) — частота внесения изменений в данные, хранящиеся в файле
- Активность (activity of file) — процент записей в файле, к которым выполняются обращения в течение заданного периода времени

Физическая запись (physical record, физический блок, physical block) — единица данных, считываемая с накопителя, или записываемая на него.

Логическая запись (logical record, логический блок, logical block) — набор данных, воспринимаемый программами как единое целое.

Неблокированные записи (unblocked records) — записи, в которых одной физической записи соответствует одна логическая запись.

Сблокированные записи (blocked records) — записи, в которых в одной физической может содержаться несколько логических записей. В файле с фиксированной длинной логических записей все они имеют одинаковую длину. В файле с записями произвольной длины записи могут иметь любую длину вплоть до размера физического блока.

Вопросы для самопроверки

1. Может ли физическая запись содержать несколько логических записей файла? (Да/Нет)

2. При использовании записей переменной длины накладные расходы возрастают? (Да/Нет)

Ответы на вопросы

1. Да. В случае сблокированных записей в одной физической может содержаться несколько логических записей файла.

2. Да. Потому что система должна определять длину каждой записи. Например, можно помечать конец каждой записи специальным маркером или указывать длину каждой записи в специальном поле в начале записи.

§ 3. Файловые системы

Файловая система (file system) — часть операционной системы, занимающаяся организацией файлов и обеспечением доступа к ним. Файловые системы обеспечивают как логическую, так и физическую организацию файлов. Они также управляют свободным пространством накопителей, обеспечивают безопасность, поддерживают целостность данных и т.д.

Задачи файловых систем

- Управление файлами
- Управление вспомогательными устройствами хранения

- Обеспечение целостности файлов
- Организация методов доступа к данным

Управление файлами (file management) — задача файловой системы, в которую входит обеспечение возможностей хранения файлов, выполнения обращений к ним, совместного использования файлов и безопасности.

Управление вспомогательными устройствами хранения (auxiliary storage management) — задача файловой системы, сводящаяся к выделению пространства под файлы на вторичных устройствах хранения.

Обеспечение целостности файлов (file integrity management) — задача файловой системы, в которую входит гарантирование того, что хранимая в файлах информация не будет повреждена. Если целостность файлов гарантируется, то в файлах будет только та информация, которая должна быть.

Вопросы для самопроверки

1. Файловые системы работают только с данными на вторичных устройствах хранения? (Да/Нет)

2. Имеют ли сходство файловые системы и системы управления виртуальной памятью? (Да/Нет)

Ответы на вопросы

1. Нет. Файловые системы работают с файлами, представляющими собой именованные наборы данных, которые могут храниться на любом носителе, включая оперативную память.

2. Да. Файловые системы должны управлять выделением пространства на накопителях и контролировать доступ к накопителю. Во многих системах виртуальная память реализована в виде файла обмена.

§ 4. Директории

Директория (directory) — файл, хранящий ссылки на другие файлы. В записях в директориях обычно содержатся имена файлов, их типы, размеры и другие данные.

Пример содержимого записи в файле директории

Имя	Символьная строка, содержащая имя файла
Местоположение	Физический блок или логический адрес файла в файловой системе
Размер	Количество байтов, занимаемых файлом
Тип	Описание назначения файла
Время создания	Время создания файла

Плоская структура директорий (flat directory structure) — одноуровневая файловая система, содержащая только одну директорию.

Иерархически структурированная файловая система (hierarchically structured file system) — файловая система, в которой у каждой директории может быть много дочерних, но только одна родительская директория (см. рис. 1).

Корень (root) — начало иерархической структуры файловой системы.

Корневая директория (root directory) — директория, содержащая указатели на пользовательские директории.

Пользовательские директории (user directories) — директории, содержащие записи о пользовательских файлах; каждая запись указывает размещение файла на накопителе.

Рабочая директория(working directory) — директория, содержащая непосредственно доступные пользователю файлы.

Путь (pathname) — строка, обозначающая файл или директорию их логическими именами, разделяя директории символами-разделителями (например, “ / ” в UNIX). Абсолютный путь указывает размещение файла или директории начиная с корневой директории; относительный путь указывает их размещение начиная с текущей рабочей директории.

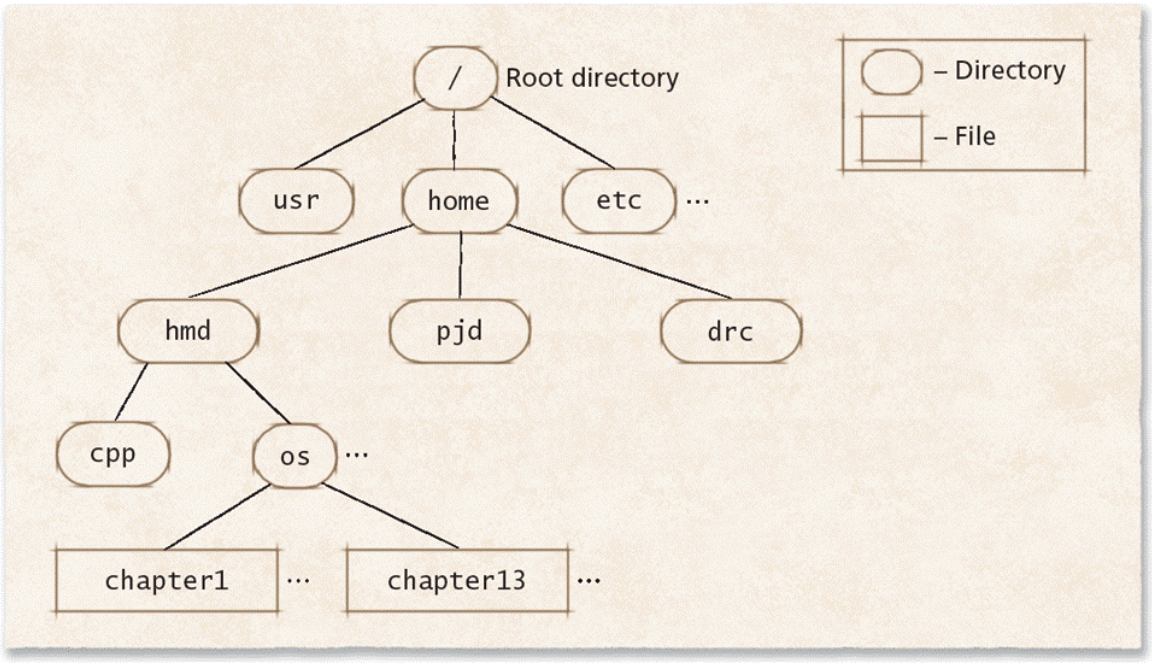


Рис. 1. Иерархическая файловая система

Мягкая ссылка (soft link) — файл, содержащий путь к другому файлу, на который он ссылается.

Пример. Ярлык в Windows.

Жесткая ссылка (hard link) — запись в директории, указывающая физическое размещение файла на накопителе (обычно номер начального блока).

Вопросы для самопроверки

1. Одноуровневые файловые системы подходят для большинства систем? (Да/Нет)

2. Директория — это файл? (Да/Нет)

3. Жесткая ссылка — это файл? (Да/Нет)

Ответы на вопросы

1. Нет. В большинстве систем нужно хранить множество файлов с совпадающими именами, что невозможно в одноуровневых файловых системах.

2. Да. Директория — это файл, хранящий ссылки на другие файлы.

3. Нет. Жесткая ссылка — это запись в директории, указывающая на размещение файла на устройстве хранения. В виде файлов реализуются мягкие ссылки.

§ 5. Метаданные

Метаданные (metadata) — недоступные непосредственно пользователям данные, с помощью которых файловая система управляет файлами. Например, данные о свободных блоках накопителей (чтобы гарантировать, что новые данные не будут записаны поверх записанных ранее), о времени последнего изменения файлов (для целей учета) и пр.

Суперблок (superblock) — блок, содержащий жизненно важные для обеспечения целостности файловой системы метаданные (например, количество блоков в файловой системе, список или битовый массив свободных блоков, идентификатор файловой системы, расположение корневой директории).

Форматирование накопителя (formatting of storage device) — подготовка накопителя для файловой системы, которая обычно включает проверку накопителя на наличие неработающих областей, стирание ранее хранившихся на нем данных, создание корневой директории. Многие файловые системы при форматировании также создают суперблок.

Таблица открытых файлов

- При открытии файла операционная система сначала находит информацию о нем, просматривая структуру директорий

- Чтобы избежать многократных просмотров система хранит в оперативной памяти таблицу, ведущую учет открытых файлов
- Таблица открытых файлов обычно содержит:
  - дескрипторы файлов;
  - блоки управления файлами.

Дескриптор файла (file descriptor) — неотрицательное целое число, являющееся индексом в таблице открытых файлов. Процесс обращается к дескриптору вместо имени файла, чтобы получить доступ к данным файла без необходимости перемещаться по структуре директорий.

Блок управления файлом (file control block) — метаданные, содержащие необходимую файловой системе информацию о файле.

Обычное содержимое блока управления файлом

- Символьное имя файла
- Данные о расположении файла на накопителе
- Организационная структура (например, файл последовательного доступа или произвольного доступа)
- Сведения о типе накопителя (например, жесткий диск или компакт-диск)
- Данные управления доступом (например, о том, какие пользователи могут обращаться к файлу и какие они могут выполнять операции)
- Данные о типе файла (например, файл данных, программа на языке C или исполняемый файл)
- Характер файла (постоянный или временный)
- Счетчики обращений к файлу (например, количество операций чтения)
- Дата и время создания файла

- ...

Вопросы для самопроверки

1. Желательно ли хранить избыточные копии суперблоков? (Да/Нет)

2. Могут ли пользователи непосредственно обращаться к метаданным? (Да/Нет)

Ответы на вопросы

1. Да. Если суперблок будет поврежден, файловая система будет не в состоянии обратиться к файлам. Поэтому большинство файловых систем хранит избыточные копии суперблоков на накопителях. Если суперблок будет поврежден, его можно заменить одной из копий.

2. Нет. Если доступ к метаданным не ограничивать, то случайное неправильное их использование может привести к потере целостности данных или их уничтожению.

§ 6. Монтирование

Монтирование (mount operation) — операция, объединяющая две отдельные файловые системы в одно пространство имен, делая их доступными из общей корневой директории. Применяется, например, для доступа к данным, хранящимся на переносном винчестере, DVD, или на другой рабочей станции в сети.

Пространство имен (namespace) — набор файлов, которые могут идентифицироваться файловой системой.

Точка монтирования (mount point) — заданная пользователем директория в иерархии родной файловой системы, в которую команда монтирования помещает корневую директорию монтируемой файловой системы.

Плоская структура монтирования

- Использовалась в ранних версиях файловых систем OS Windows

- Каждая монтируемая файловая система обозначалась буквой и размещалась на одном и том же уровне структуры директорий
- Например, файловая система, содержащая файлы операционной системы обозначалась C:, а следующая файловая система — D:

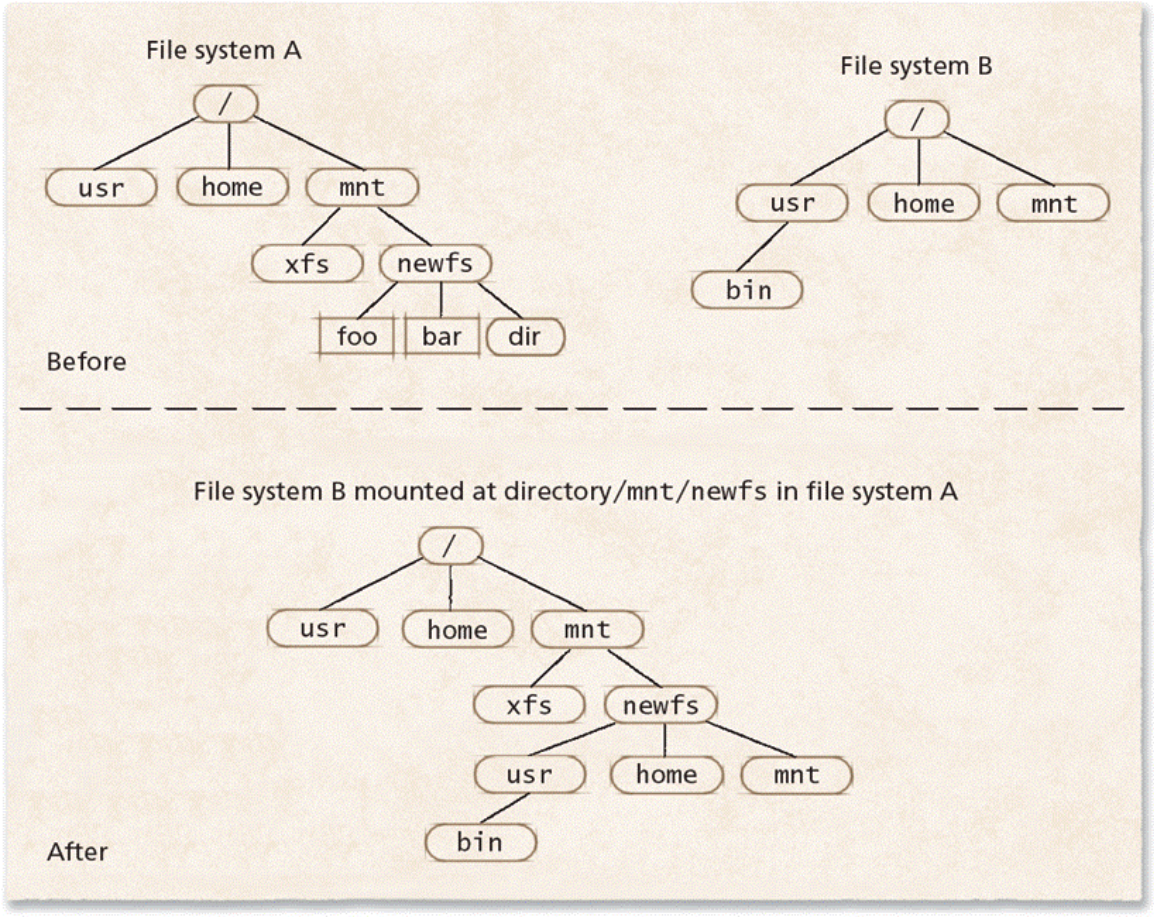


Рис. 2. Произвольное монтирование файловой системы

Произвольное монтирование (см. рис. 2)

- Используется в NTFS и файловых системах UNIX
- Точки монтирования можно размещать в файловой системе где угодно
- Содержимое системной директории родной файловой системы в

точке монтирования временно скрывается, пока другая файловая система монтирована в этой директории

Вопросы для самопроверки

1. Смонтированная файловая система и родная должны быть одного типа? (Да/Нет)

2. Может ли файловая система создать жесткие ссылки на файлы в смонтированной файловой системе? (Да/Нет)

Ответы на вопросы

1. Нет. Основное преимущество монтирования файловых систем состоит в том, что оно позволяет нескольким различным файловым системам быть доступными через единый интерфейс.

2. Нет. Жесткие ссылки указывают специфические для устройства номера блоков, соответствующие файловой системе, в которой они хранятся, и они не могут использоваться для указания физического размещения данных в других файловых системах.

Глава 11

РАЗМЕЩЕНИЕ ФАЙЛОВ

§ 1. Непрерывное размещение файлов

- Под файлы выделяются непрерывные участки адресов в пространствах устройств хранения
- Пользователь заранее указывает размер области, которую нужно выделить под файл
- Если непрерывную область заданного размера выделить невозможно, файл создан не будет

Преимущества непрерывного размещения файлов

- Следующие друг за другом логические записи обычно оказываются размещенными рядом и физически
- Непрерывное размещение файлов увеличивает скорость доступа к данным
- В директориях нужно хранить только информацию о месте, в котором файл начинается, и о длине файла

Недостатки непрерывного размещения файлов

- Этим системам присуща внешняя фрагментация после удаления файлов (см. рис. 1)
- Если размер файла превысит изначально заданный, то файл должен быть целиком перенесен в новый участок подходящего размера

Вопросы для самопроверки

1. Непрерывное размещение файлов не используется в современных системах? (Да/Нет)

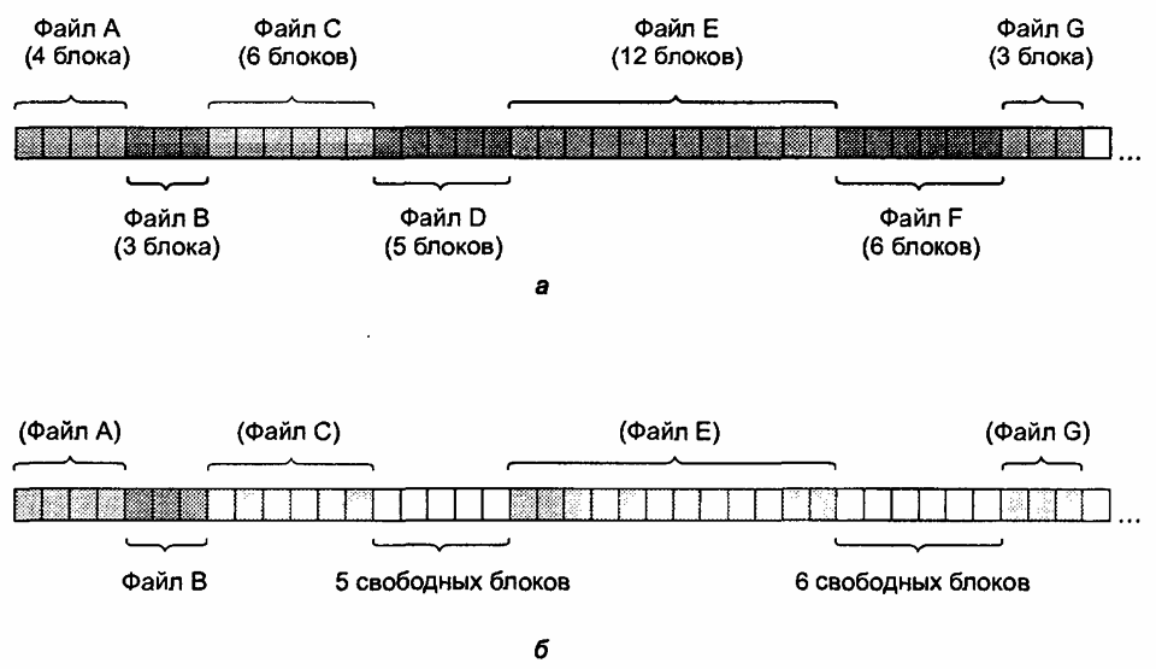


Рис. 1. а) семь непрерывных файлов на диске; б) состояние диска после удаления двух файлов

2. Подходит ли непрерывное размещение файлов для записи данных на магнитной ленте? (Да/Нет)

Ответы на вопросы

1. Нет. Непрерывное размещение файлов очень удобно и используется на однажды записываемых носителях, например, компакт-дисках и DVD, где размер файлов неизменен во времени.

2. Да. Для файлов, хранящихся на магнитной ленте, подходит непрерывное размещение файлов, поскольку магнитная лента — это носитель с последовательным доступом.

§ 2. Размещение файлов в виде связных списков

Дорожка (track) — кольцевая область на диске (см. рис. 2). Последовательности битов данных из файлов обычно размещаются в непрерывных областях на одной и той же дорожке, чтобы уменьшить потребность в операциях позиционирования головок.

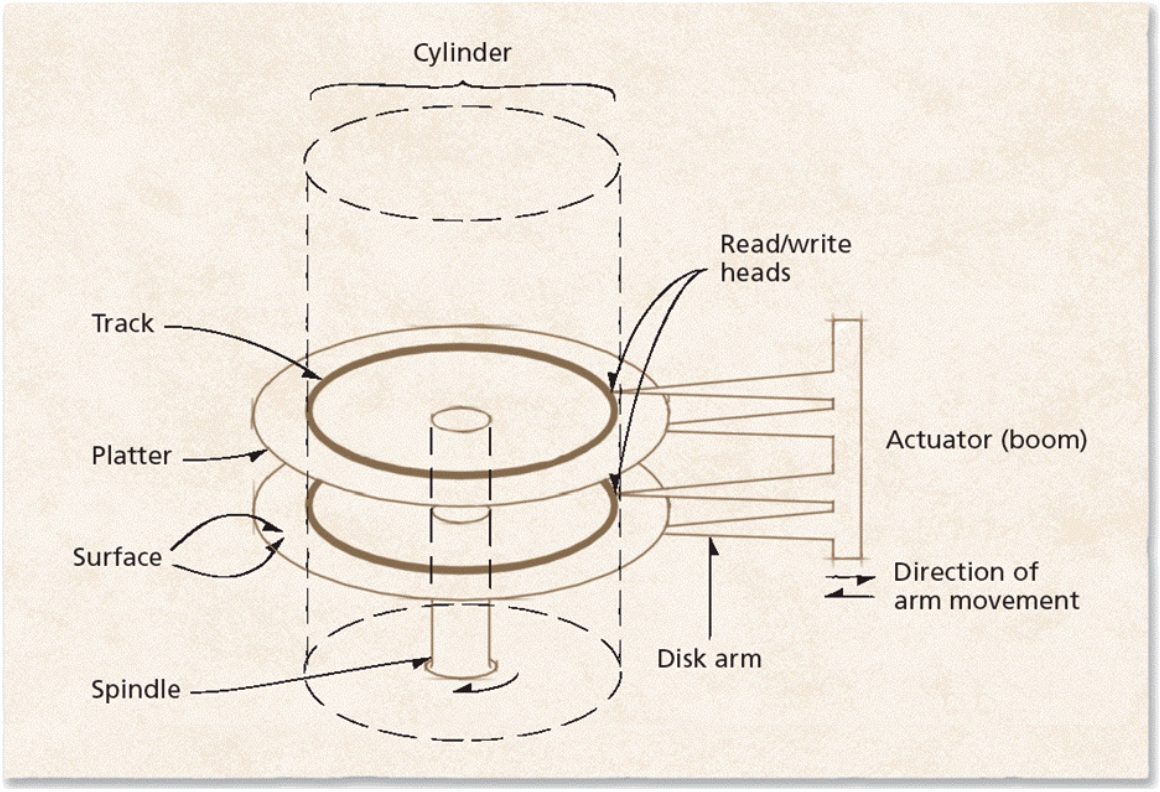


Рис. 2. Схема дискового накопителя с подвижными головками. Здесь дорожка выделена темным цветом

Сектор (sector) — наименьшая часть дорожки, к которой может обращаться запрос ввода/вывода (см. рис. 3). Размер сектора обычно равен 512 байт.

Блок (block) — элемент данных фиксированного размера, состоящий из непрерывной последовательности нескольких секторов. Обычно используются блоки размером от 1 до 8 килобайт (от 2 до 16 секторов).

Блочное размещение (block allocation) — прием, позволяющий файловым системам более эффективно распоряжаться пространством накопителей и уменьшать накладные расходы при просмотре файлов за счет размещения файлов в блоках.

Связный список (linked list) — метод размещения файлов, состоящих из цепочек блоков, при котором в каждом блоке одна из

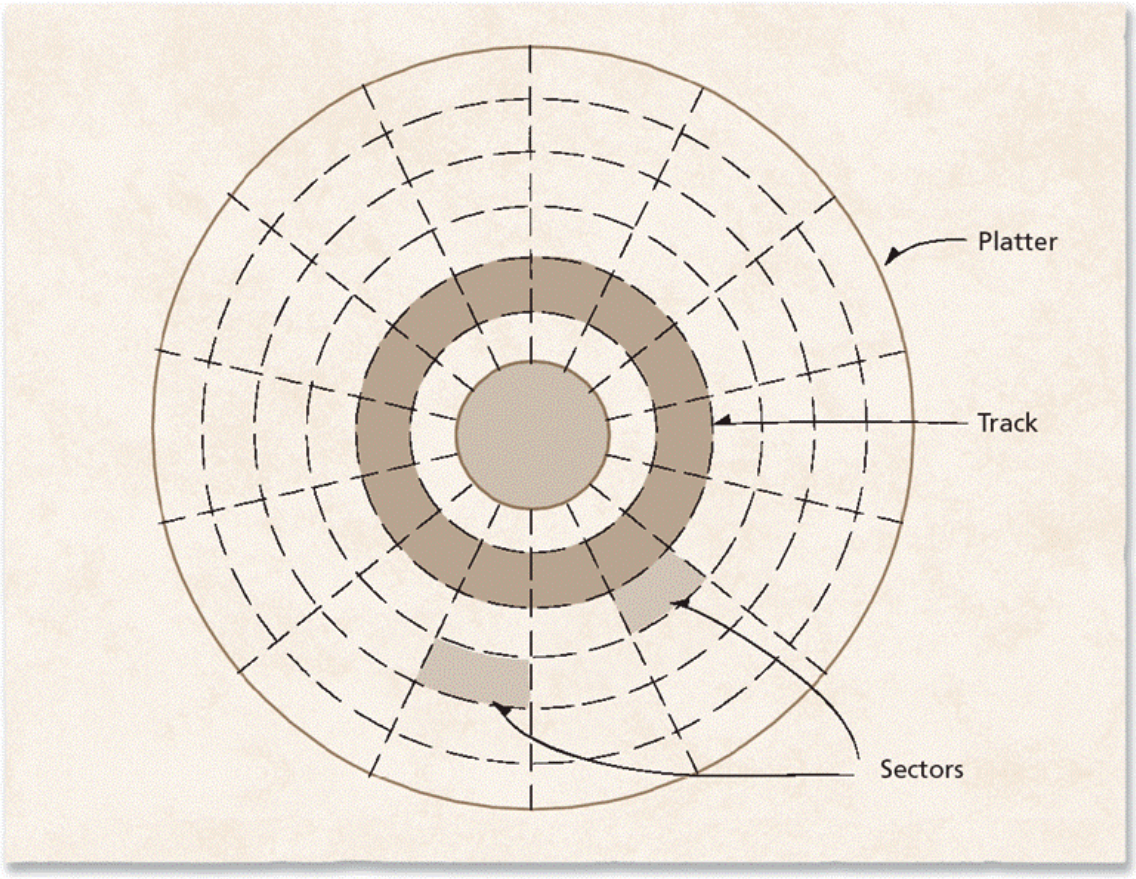


Рис. 3. Схематическое изображение структуры поверхности диска. Здесь дорожка и два сектора выделены темным цветом

записей зарезервирована под указатель на другой блок. Директория содержит указатель на первый блок в цепочке (см. рис. 4).

Вопросы для самопроверки

1. Размер сектора обычно равен 512 байт? (Да/Нет)

2. В худшем случае система со связным списком должна обратиться ко всем блокам файла при поиске данных? (Да/Нет)

3. Большие блоки в файлах лучше маленьких? (Да/Нет)

Ответы на вопросы

1. Да. Размер сектора обычно равен 512 байт.

2. Да. Если используется связный список, в худшем случае системе нужно обратиться к каждому блоку файла, чтобы найти нужные данные.

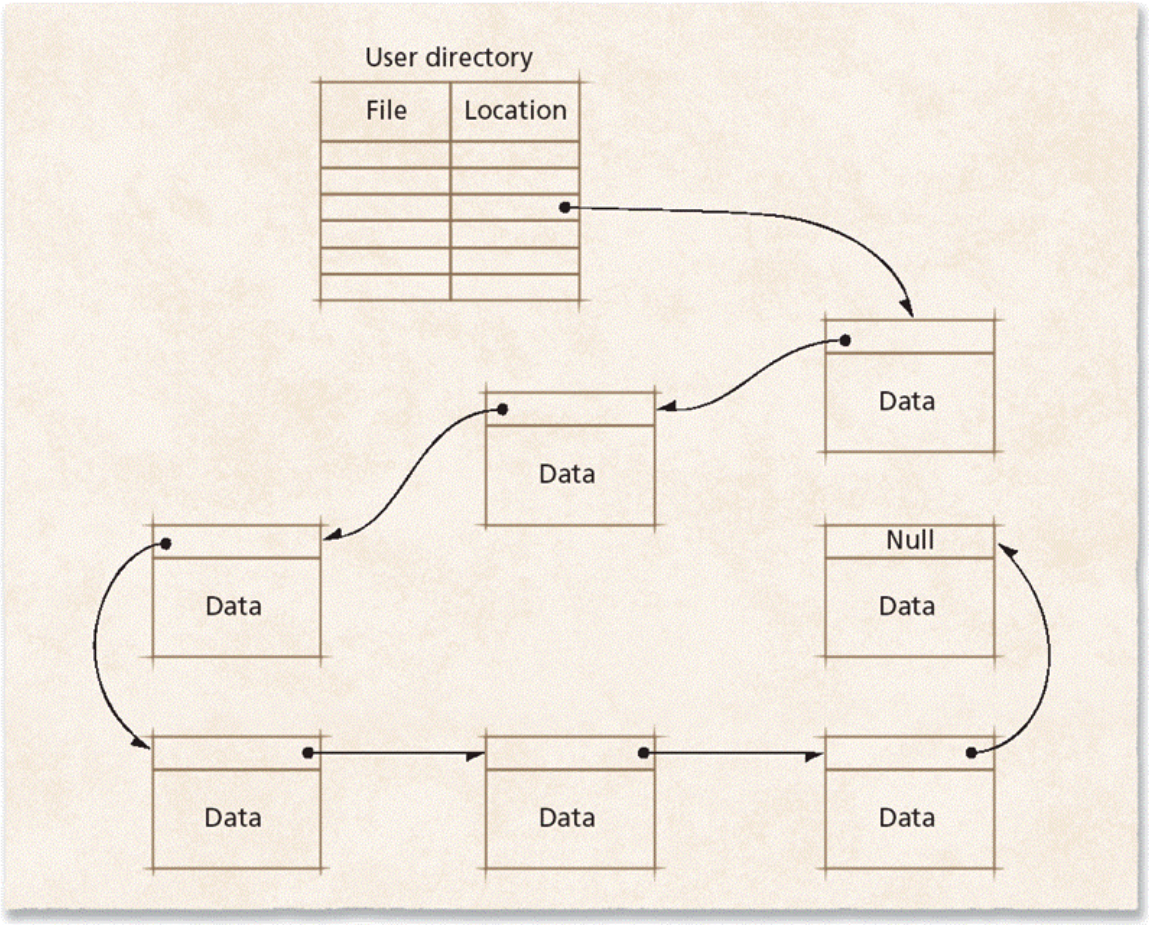


Рис. 4. Размещение файлов в виде связных списков

3. Нет. При большом размере блока для извлечения нужной записи требуется меньше операций ввода/вывода, но на внутреннюю фрагментацию расходуется более заметная часть пространства накопителя.

§ 3. Табличное фрагментированное размещение

Таблица выделения блоков (block allocation table) — таблица, использующаяся в файловых системах с табличным фрагментированным размещением файлов для ускорения доступа к данным (см. рис. 5). Обычно загружается в кэш. В этой таблице в качестве индекса используются номера блоков на накопителе, а записи хранят указа-

тели на блоки файлов. Записи в директориях указывают на первые блоки файлов.

FAT (File Allocation Table, таблица размещения файлов) — реализация файловой системы с табличным фрагментированным размещением, разработанная фирмой Microsoft. Известны версии FAT12, FAT16, FAT32, где цифры указывают на число бит в табличных записях.

Вопросы для самопроверки

1. Табличное фрагментированное размещение файлов эффективнее размещения в виде связных списков? (Да/Нет)

2. FAT не подходит для работы с современными дисковыми накопителями? (Да/Нет)

3. Верно ли, что FAT не используется в современных системах? (Да/Нет)

Ответы на вопросы

1. Да. Таблицы выделения блоков могут хранить информацию о размещении данных в непрерывном виде, и количество операций позиционирования, необходимых для доступа к данным, уменьшается.

2. Да. Современные дисковые накопители имеют большой объем и, следовательно, состоят из большого числа блоков. При этом увеличивается размер таблицы размещения файлов, за счет чего растет время доступа к файлам и затраты памяти на кэширование таблицы.

3. Нет. FAT эффективна и сейчас для носителей малой емкости, например, дискет.

§ 4. Индексированное фрагментированное размещение

Индексный блок (index block) — блок, содержащий список указателей на блоки с данными файла. Запись о файле в его директории содержит указатель на индексный блок этого файла (см. рис. 6). Индексные блоки используются, например, в Unix, Linux, NTFS Windows XP.

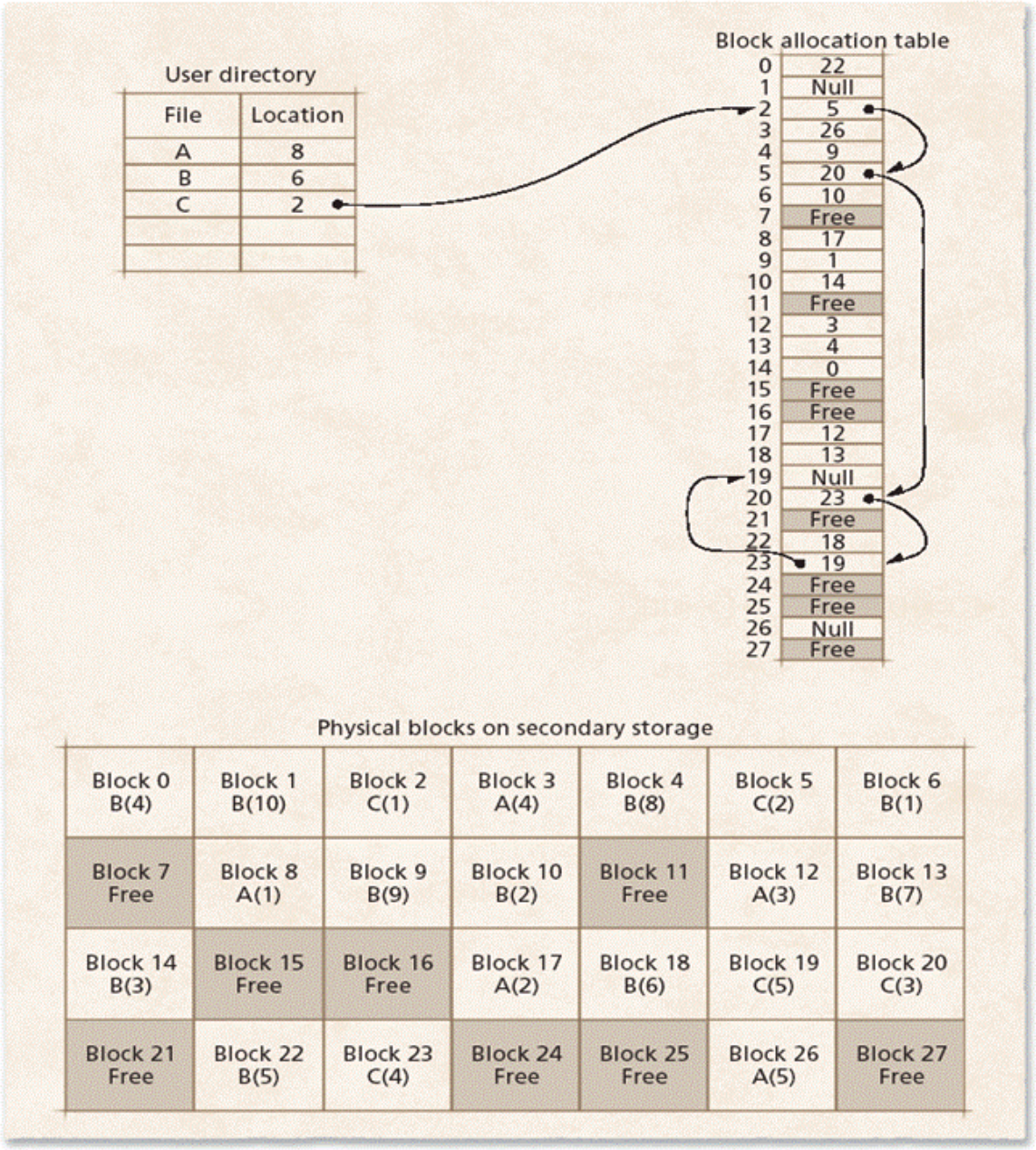


Рис. 5. Табличное фрагментированное размещение файлов

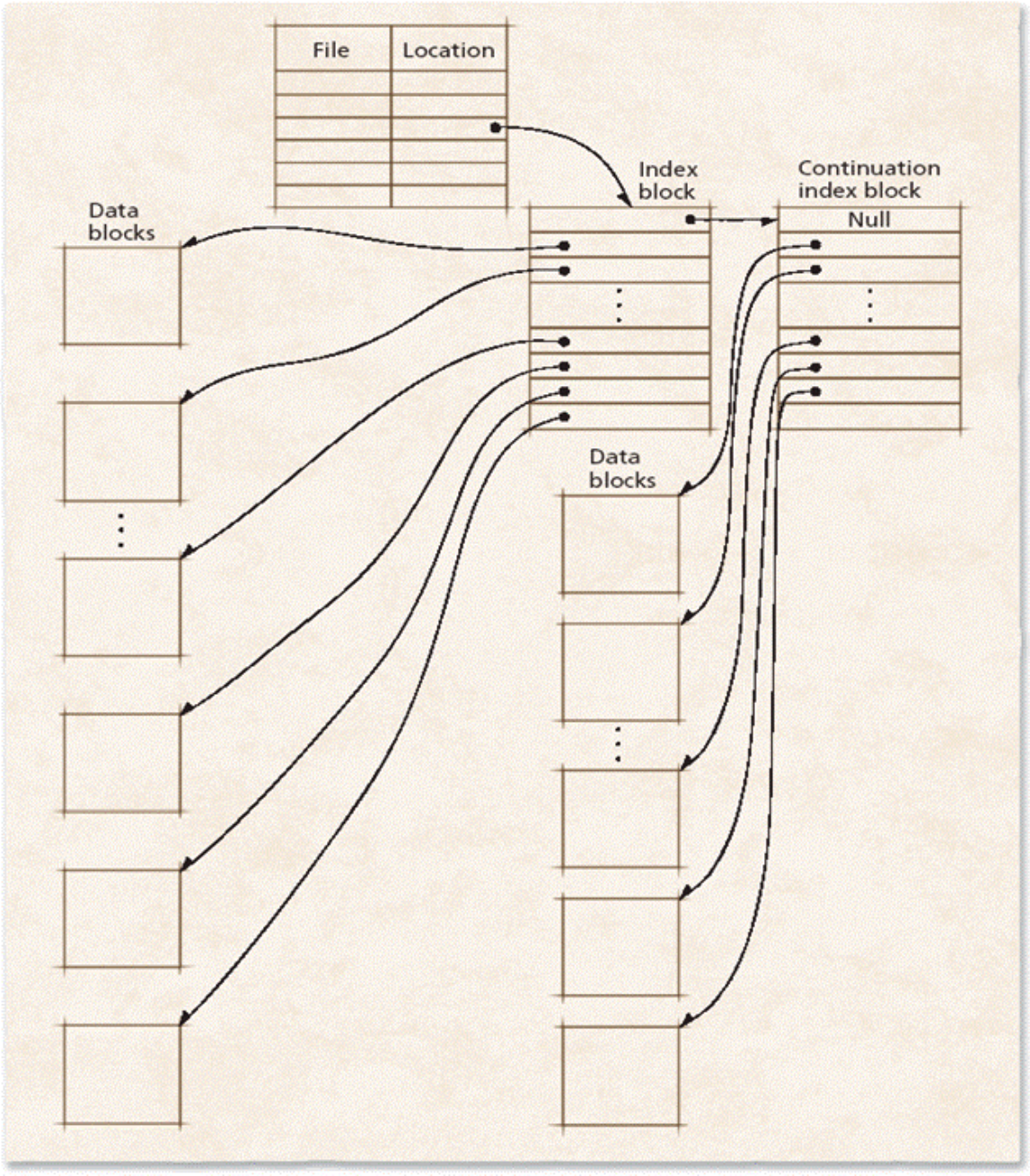


Рис. 6. Сцепление индексных блоков

Сцепление (chaining) — метод индексированного фрагментированного размещения, при котором в каждом индексном блоке последние несколько записей зарезервированы под указатели на другие индексные блоки, которые в свою очередь указывают на блоки данных (см. рис. 6). Сцепление позволяет индексным блокам указывать размещение больших файлов, распределяя указатели на блоки данных по нескольким индексным блокам.

Вопросы для самопроверки

1. Индексированное фрагментированное размещение для дисков большого объема эффективнее табличного? (Да/Нет)

2. Размещение индексных блоков рядом с блоками данных уменьшает время доступа? (Да/Нет)

Ответы на вопросы

1. Да. Индексированное фрагментированное размещение аналогично хранению отдельной таблицы выделения блоков для каждого файла. Такой подход может быть эффективнее, поскольку ссылки на блоки каждого файла хранятся в виде непрерывной последовательности в пределах индексного блока.

2. Да. При считывании индексного блока головка будет близко от блоков данных, на которые ссылается индексный блок, и потребность в позиционировании будет уменьшена или вообще исчезнет.

§ 5. Управление свободным пространством

Список свободных блоков (free list) — связный список блоков, содержащих адреса свободных блоков на накопителе (см. рис. 7).

Список свободных блоков

- Когда системе нужно выделить для файла новый блок, она находит адрес свободного блока в списке свободных блоков, записывает данные в этот блок и удаляет запись об этом блоке из списка
- Обычно файловая система выделяет блоки из начала списка и добавляет освободившиеся блоки в его конец

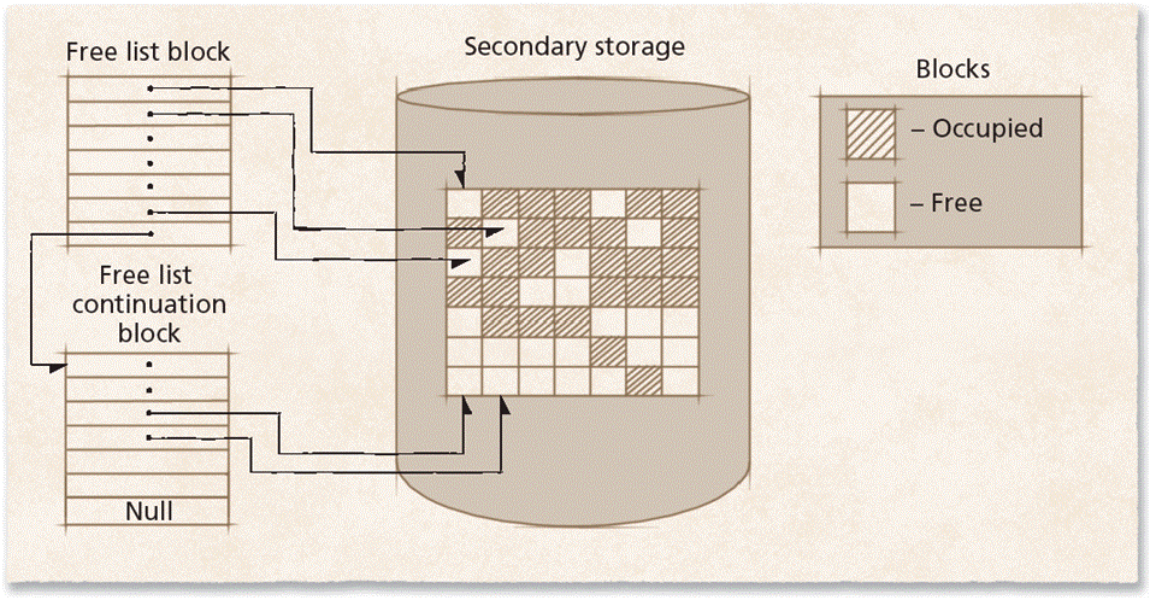


Рис. 7. Управление свободным пространством с помощью списка свободных блоков

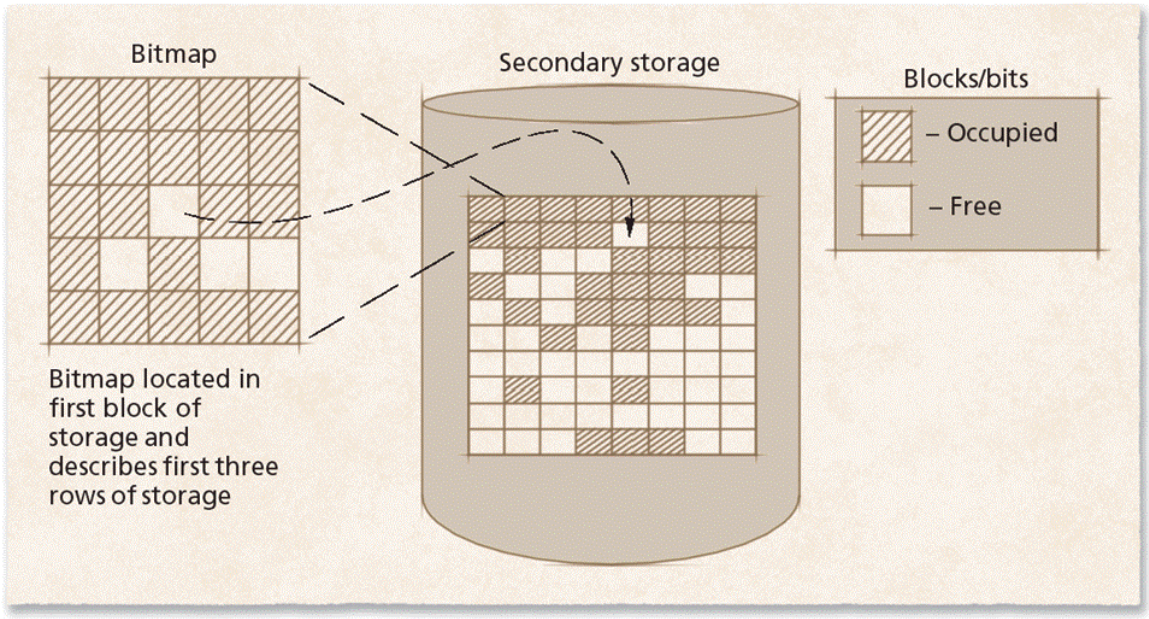


Рис. 8. Управление свободным пространством с помощью битового массива

- Указатели на начало и конец списка могут храниться в суперблоке файловой системы

Битовый массив (bitmap) — средство управления свободным пространством накопителя, в котором каждому блоку накопителя соответствует один бит, причем номер бита соответствует номеру блока (см. рис. 8). Битовые массивы позволяют выделять непрерывные блоки эффективнее, чем списки свободных блоков, но на поиск свободных участков с помощью битовых массивов может потребоваться заметное время.

Вопросы для самопроверки

1. Битовый массив эффективнее списка свободных блоков при выделении одного свободного блока? (Да/Нет)

2. Битовый массив эффективнее списка свободных блоков при поиске непрерывной последовательности блоков? (Да/Нет)

3. Битовый массив всегда занимает меньше места, чем список свободных блоков? (Да/Нет)

Ответы на вопросы

1. Нет. Список свободных блоков эффективнее при выделении одного свободного блока, поскольку для этого в нем нужно только выполнить переход по указателю в начало списка.

2. Да. При поиске непрерывной последовательности свободных блоков битовые массивы эффективнее — их можно просматривать в поисках соответствующих участков, а списки свободных блоков нужно сортировать, на что требуется заметное дополнительное время.

3. Нет. Часто битовый массив занимает меньше места, поскольку в нем каждому блоку соответствует один бит, а в списке — 32 или даже 64 бита. Однако, если на накопителе мало свободных блоков, то список будет занимать меньше места, чем битовый массив.

Глава 12

КОНТРОЛЬ ДОСТУПА К ФАЙЛАМ И ЗАЩИТА ДАННЫХ

§ 1. Контроль доступа к файлам

Матрица контроля доступа (access control matrix) — двумерный список всех пользователей и прав их доступа к файлам в системе (см. рис. 1).

User \ File	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0
3	0	1	0	1	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	1	1	0	0	0
7	1	0	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0	0	0
9	1	1	1	1	0	0	0	0	1	1
10	1	1	0	0	1	1	0	0	0	1

Рис. 1. Матрица контроля доступа

Матрица контроля доступа

- Обычно очень разрежена и имеет большой размер
- Для обозначения различных видов доступа (только для чтения, для записи и т.д.) нужно использовать специальные коды

Классы пользователей (user classes) — способ классификации, указывающий группы или отдельных пользователей, обладающих определенными правами доступа к файлам. Данные контроля доступа могут храниться в виде части блока управления файлом, занимая незначительное дисковое пространство.

Классы пользователей

- **Владелец** (owner) — пользователь, создавший файл
  - обладает неограниченным доступом к файлу;
  - может изменять права других пользователей на доступ к этому файлу.
- **Указанный пользователь** (specified user) — пользователь, которому владелец предоставил права на доступ к файлу
- **Группа** (group) или **проект** (project) — группа пользователей, работающая над конкретным проектом
  - члены группы могут обладать доступом к связанным с проектом файлам других членов группы
- **Общедоступный** (public) — файл, к которому могут обращаться все пользователи
  - можно читать или запускать, но не изменять

Вопросы для самопроверки

1. Матрицы контроля доступа пригодны для большинства систем? (Да/Нет)

2. Должна ли система обратиться к блоку управления файлом, если там хранятся данные контроля доступа, но чтение файла запрещено? (Да/Нет)

Ответы на вопросы

1. Нет. Они обычно велики по размеру и сильно разрежены, поэтому их хранение приводит к бессмысленным затратам пространства на накопителях и большому времени обращения при реализации контроля доступа.

2. Да. Если данные контроля доступа к файлу хранятся в блоке управления файлом, узнать о правах доступа к файлу можно только прочитав блок управления файлом.

§ 2. Резервное копирование и восстановление

Физическое резервное копирование (physical backup) — копирование каждого бита на накопителе. Попытки интерпретировать содержимое накопителя при физическом резервном копировании не предпринимаются.

Логическое резервное копирование (logical backup) — методика резервного копирования, при которой копируются данные файлов и информация о директориях файловой системы, часто в стандартном сжатом формате.

Инкрементное резервное копирование (incremental backup) — методика логического резервного копирования, при которой копируются только данные, изменившиеся со времени предыдущего резервного копирования.

Вопросы для самопроверки

1. Может ли физическая копия содержать часть файловой системы? (Да/Нет)

2. Поддерживает ли физическое резервное копирование инкрементное копирование данных? (Да/Нет)

Ответы на вопросы

1. Нет. Физическая резервная копия не учитывает логическую структуру файловой системы, поэтому она не может разделить содержащиеся в ней файлы.

2. Нет. Инкрементное резервное копирование — это методика логического резервного копирования.

§ 3. Журнальные файловые системы

Атомарная транзакция (atomic transaction) — группа операций, которая не влияет на состояние системы до тех пор, пока не завершаются все операции группы.

Пример. Перевод денег с одного банковского счета на другой.

Откат транзакции (rolling back a transaction) — возврат системы в состояние, в котором она пребывала до начала выполнения транзакции.

Журнальная файловая система (Log-structured File System, LFS) — файловая система, выполняющая все операции с файлами в виде транзакций, чтобы гарантировать целостность файловой системы и метаданных.

Примеры. NTFS Journal File System, файловые системы Linux и Unix.

Журнальная файловая система (см. рис. 2)

- Данные записываются в конец журнала (log), файл которого занимает все свободное пространство накопителя
- По журналу обычно распределяются метаданные файлов, что позволяет быстро находить запрашиваемые данные
- Файловая операция завершается только после записи метаданных файла
- Операция с директорией завершается только после записи суперблока

Вопросы для самопроверки

1. LFS записывает сначала метаданные файла, а затем его данные? (Да/Нет)

2. Повышает ли кэширование производительность LFS? (Да/Нет)

3. Свойственны ли LFS проблемы фрагментации? (Да/Нет)

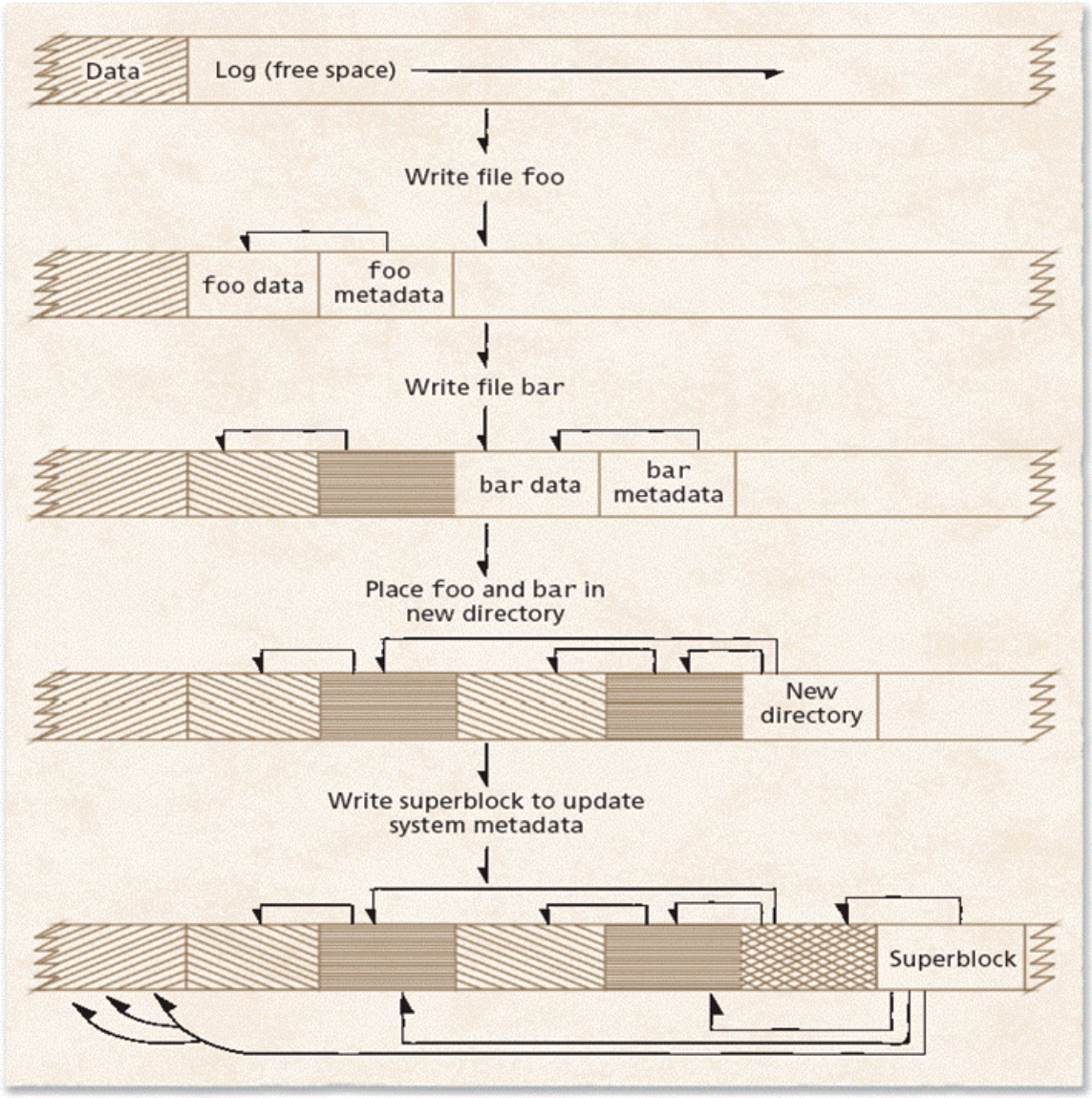


Рис. 2. Запись двух файлов в новую директорию в журнальной файловой системе

Ответы на вопросы

1. Нет. Если система запишет сначала метаданные файла, а затем данные и в системе возникнет отказ, то метаданные будут указывать на некорректные блоки (т.е. в блоках будут отсутствовать данные файла).

2. Да. Поскольку измененные директории и метаданные всегда записываются в конец журнала, LFS может потребоваться просмотреть весь журнал, чтобы найти конкретный файл. Дабы уменьшить серьезность этой проблемы, LFS кэширует информацию о размещении файлов в файловой системе.

3. Да. Когда в журнале больше не остается свободного места, LFS освобождает блоки для размещения новых данных, если другие блоки содержат измененные копии данных из этих блоков. При этом новые могут оказаться фрагментированными. Чтобы решить эту проблему, LFS может создавать непрерывные участки в журнале, группируя данные, что требует накладных расходов.

§ 4. Системы баз данных

База данных (database) — централизованно управляемое хранилище данных, представленных в стандартизованном формате (например, иерархические, реляционные, объектно-ориентированные базы данных).

Базы данных

- Данные можно просматривать в соответствии с определенными логическими взаимосвязями между ними
- Данные организуются по их содержанию, а не размещению, за счет чего можно уменьшить или исключить избыточность данных

Система базы данных (database system) — определенный набор данных, аппаратных устройств, на которых эти данные хранятся, и программ, управляющих доступом к данным (эти программы называются системой управления базы данных, СУБД).

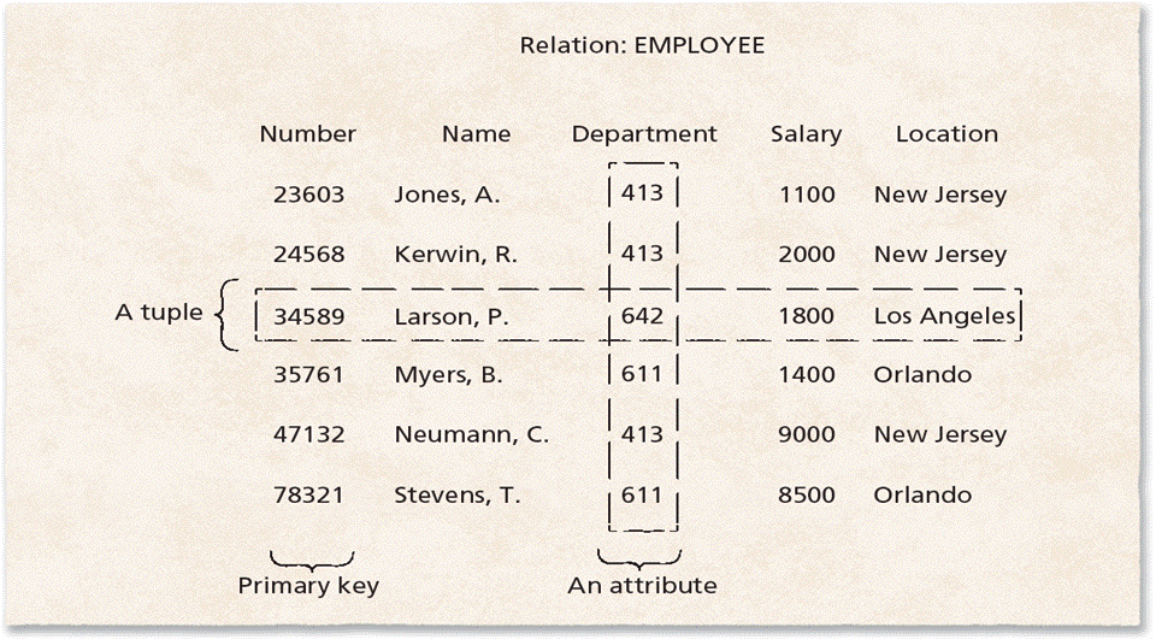


Рис. 3. Отношение в реляционной базе данных

Реляционная модель (relational model) — предложенная Кодом модель данных, лежащая в основе большинства современных систем баз данных. Реляционная база данных — это набор связанных отношений.

Отношение (relation) — набор строк в реляционной базе данных (см. рис. 3).

Строка (row) или кортеж (tuple) — элемент отношения, сочетание всех атрибутов (attribute) одного объекта (см. рис. 3).

Первичный ключ (primary key) — в реляционной базе данных — сочетание атрибутов, позволяющее однозначно идентифицировать строку (см. рис. 3).

Преимущества реляционной модели баз данных

- Табличное представление, используемое в реляционной модели, просто реализуется в системах баз данных
- Реляционную модель можно воспринимать как универсальную форму представления баз данных

- Контроль доступа к данным элементарен: важные данные просто разносятся по разным отношениям, доступ к которым контролируется

Операционные системы и системы баз данных

- Различные службы операционных систем поддерживают системы баз данных:
  - файловая система,
  - служба планирования,
  - менеджер процессов,
  - менеджер межпроцессного взаимодействия,
  - контроль целостности данных,
  - виртуальная память.
- Большинство этих служб не оптимизировано специально для нужд СУБД
- Для поддержки СУБД, располагающих собственными оптимизированными службами, лучше всего использовать минимизированные операционные системы
- Поддержка баз данных в современных системах получает все большее распространение
- Возможно в скором будущем системы баз данных заменят файловые системы

Вопросы для самопроверки

1. Базы данных уменьшают избыточность данных по сравнению с обычными файловыми системами? (Да/Нет)

2. Значения каждого атрибута должны быть уникальными для всех строк отношения? (Да/Нет)

3. Число операционных систем, прямо поддерживающих системы баз данных, будет расти? (Да/Нет)

Ответы на вопросы

1. Да. Системы баз данных организуют данные по их содержимому, и никакие две записи не содержат одинаковую информацию. Например в базе данных может храниться одна копия информации о покупателе, к которой множество приложений может обращаться с помощью запросов.

2. Нет. Одно и то же значение атрибута может содержаться во множестве строк отношения.

3. Да. Многие операционные системы будут использовать базы данных как основное средство хранения пользовательских данных.

Часть V

Многопроцессорные и распределенные системы

Глава 13

МНОГОПРОЦЕССОРНЫЕ СИСТЕМЫ

§ 1. Последовательные и параллельные архитектуры ЭВМ

Многопроцессорная система (multiprocessor system) — вычислительная система, использующая для обработки данных более одного процессора.

Примеры: двухпроцессорные персональные компьютеры, мощные серверы со множеством процессоров, суперкомпьютеры.

Преимущества

- Большие инженерные и исследовательские приложения, выполняемые на суперкомпьютерах, обеспечивают прирост производительности за счет параллельной обработки данных на нескольких процессорах
- Коммерческие и научные организации используют многопроцессорные системы для повышения производительности, предоставления задачам достаточных ресурсов и достижения высокой надежности

ОС многопроцессорных ЭВМ должны дополнительно гарантировать, что:

- все процессоры загружены работой;
- процессы равномерно распределены в системе;
- выполнение взаимосвязанных процессов синхронизировано;
- процессы работают с состоятельными копиями данных, хранящихся в общей памяти;
- обеспечивается взаимное исключение для выхода из тупиковых ситуаций.

Классификация многопроцессорных систем

- По структуре каналов обработки данных
- По схеме соединений процессоров
- По способу распределения ресурсов между процессорами
- По разделению функций операционных систем между процессорами
- По способу работы с памятью

Классификация последовательных и параллельных архитектур ЭВМ

- Архитектуры компьютеров можно классифицировать по структуре каналов обработки данных, основываясь на понятии потоков данных и команд:
  - SISD,
  - MISD,
  - SIMD,
  - MIMD.

Поток (stream) — битовая последовательность данных или инструкций, передаваемых процессору.

Замечание. Битовый поток (stream) не надо путать с потоком (thread) — логическим объектом, описывающим последовательность независимо выполняемых программных инструкций внутри процесса.

Архитектура с одним потоком команд и одним потоком данных (SISD — single instruction stream, single data stream) — архитектура компьютеров, в которой один процессор последовательно выполняет инструкцию за инструкцией из потока команд над элементами данных из одного потока данных; к этой архитектуре относятся традиционные одноядерные однопроцессорные компьютеры.

Архитектура с одним потоком данных и несколькими потоками команд (MISD — multiple instruction stream, single data

stream) — архитектура компьютеров, содержащих несколько вычислительных элементов, выполняющих независимые параллельные потоки операций над одним потоком данных; к этой архитектуре относятся многоядерные процессоры.

Архитектура с одним потоком команд и несколькими потоками данных (SIMD — single instruction stream, multiple data stream) — архитектура компьютеров, состоящих из нескольких процессорных элементов, одновременно выполняющих одни и те же инструкции над различными элементами данных; к этой архитектуре относятся векторные и матричные процессоры.

Векторный процессор (vector processor) — разновидность SIMD-компьютера, в котором используется один одноядерный процессор, одновременно выполняющий одну и ту же инструкцию над несколькими элементами данных. Высокая производительность достигается за счет использования длинного конвейера, состоящего из множества узлов процессора и высокой тактовой частоты.

Матричный процессор (array processors) — SIMD-система, состоящая из множества (до десятков тысяч) простых процессоров, каждый из которых выполняет такое же действие, как и остальные процессоры системы, но над своим элементом данных. Применяются, например, в научных расчетах для матричных преобразований.

Архитектура с множеством потоков команд и данных (MIMD — multiple instruction stream, multiple data stream) — архитектура компьютеров, состоящая из множества полноценных процессоров, каждый из которых выполняет свою последовательность инструкций над своим потоком данных; это полноценная параллельная архитектура настоящих многопроцессорных систем.

Вопросы для самопроверки

1. Верно ли, что несколько вычислительных потоков могут одновременно выполняться только в системах MISD и MIMD? (Да/Нет)

2. Присутствует ли параллелизм в SISD-системах? (Да/Нет)

3. Могут ли одновременно в одной системе использоваться MIMD и MISD архитектуры? (Да/Нет)

4. Архитектура ЭВМ Джона Фон Неймана является MIMD архитектурой? (Да/Нет)

Ответы на вопросы

1. Да. В системах SISD и SIMD несколько вычислительных потоков параллельно выполняться не могут. Однако потоки выполняемые в MISD-системах работают с одними и теми же данными и поэтому не являются независимыми. Параллелизм в полной мере свойственен только MIMD.

2. Да. Для повышения производительности процессоров в SISD-системах используется конвейерная обработка данных и суперскалярная архитектура. Это позволяет одновременно выполнять инструкции нескольких команд.

3. Да. Например, если вычислительная система состоит из множества многоядерных процессоров.

4. Нет. Архитектура ЭВМ Джона Фон Неймана является последовательной SISD архитектурой с одним потоком данных и одним потоком команд.

§ 2. Схемы соединений процессоров

Схема соединений (interconnection scheme) — определяет способ соединения компонентов многопроцессорной системы, например, процессоров и модулей памяти. Схемы соединений состоят из узлов и связей.

Узел (node) — компонент системы, например, процессор, модуль памяти или коммутатор, подключенный к сети. Во многих системах в узле могут содержаться один или несколько процессоров, связанный с ними кэш, модуль памяти и коммутатор. Иногда группу соединенных друг с другом узлов тоже называют узлом (суперузлом).

Коммутатор (switch) — узел, пересылающий сообщения между узлами-компонентами.

Связь (link) — соединение между двумя узлами многопроцессорной системы, по которому между узлами передается информация.

Параметры схем соединений

- Степень узла
- Ширина сечения
- Диаметр сети
- Стоимость схемы соединений

Степень узла (degree of node) — количество узлов, непосредственно соединенных с данным узлом. Разработчики пытаются свести к минимуму степень всех узлов, чтобы уменьшить стоимость и сложность коммуникационных интерфейсов узлов.

Ширина сечения (bisection width) — минимальное количество связей, которые нужно перерезать, чтобы разделить сеть на две несвязанные части. Чем больше ширина сечения сети, тем больше ее отказоустойчивость.

Диаметр сети (network diameter) — количество связей в кратчайшем пути между двумя самыми удаленными узлами. Чем меньше диаметр сети, тем выше ее производительность и меньше задержки при обмене информацией по каналам.

Стоимость схемы соединений (cost of interconnection scheme) — общее количество соединений в схеме. Проектировщики стараются минимизировать стоимость схемы соединений.

Типы схем соединений процессоров

- Общая шина
- Переключающая матрица
- Двумерная ячеистая сеть
- Гиперкуб
- Многоуровневая сеть

Общая шина (shared bus) — схема соединений компонентов в многопроцессорных системах, в которой все компоненты, включая процессоры и модули памяти, соединены одной шиной (см. рис. 1). В каждый момент времени от одного узла ко всем остальным узлам может быть отправлена только одна единица данных.

Пример. Dual-processor Intel Pentium.

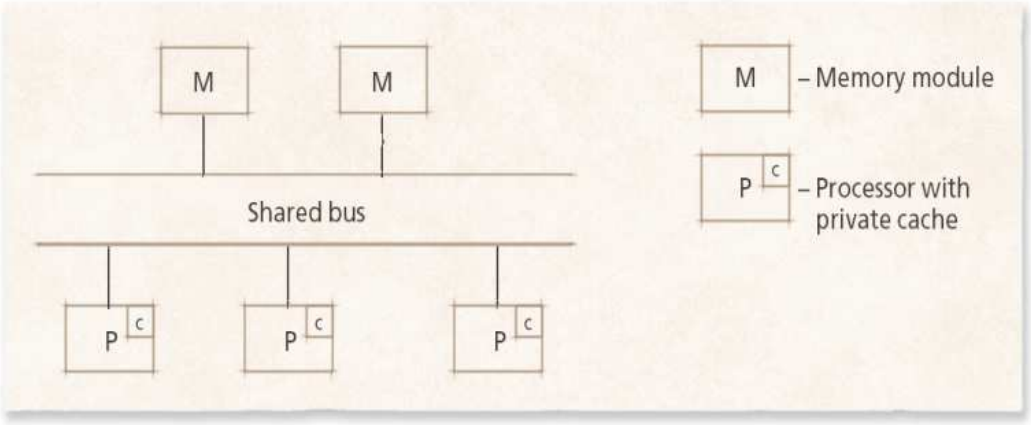


Рис. 1. Многопроцессорная система с общей шиной

Общая шина

- Простой и дешевый способ связи между несколькими процессорами (на практике обычно не более 32)
- Применяется для небольших систем и для создания суперузлов в крупных системах
- Много производительных процессоров могут быстро загрузить общую шину
- Если к шине сразу обратится несколько процессоров возникнет конфликт, чтобы их сократить у каждого процессора есть свой кэш

Переключающая матрица (crossbar-switch matrix) — схема соединения процессоров, обеспечивающая отдельный путь от каждого узла-отправителя к каждому узлу-получателю (см. рис. 2). Все узлы

одновременно могут отправить по одной единице данных, но в любой момент времени один узел может принимать лишь один поток данных.

Пример. Sun UltraSPARC-III.

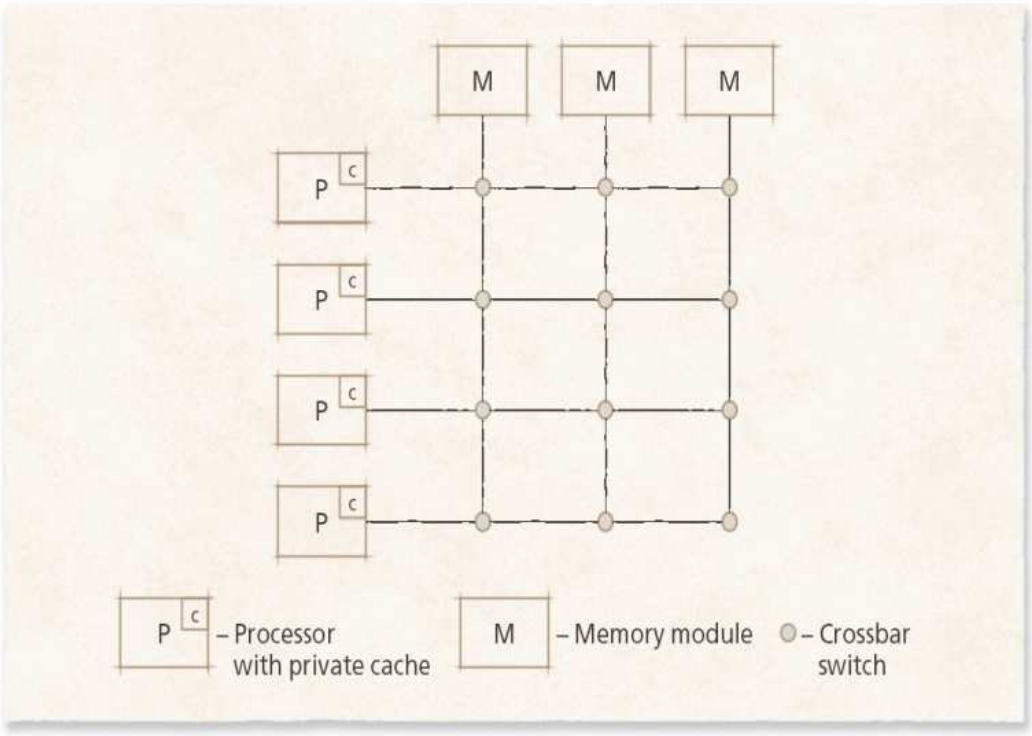


Рис. 2. Многопроцессорная система с переключающей матрицей

Переключающая матрица

- Диаметр сети практически равен 1, значит система обладает максимальной производительностью
- Чтобы разделить надвое такую сеть нужно перерезать половину связей, значит ширина сети равна (n x m)/2, где n — число процессоров, а m — число модулей памяти; поэтому система обладает высокой отказоустойчивостью
- Цена переключающих матриц велика и растет пропорционально n x m

- Аппаратное обеспечение дешевеет, и матрицы все чаще используются в больших системах

Двумерная ячеистая сеть (2-D mesh network) — схема соединения процессоров, в которой узлы организованы в прямоугольник размерности n x m (см. рис. 3). Каждый узел состоит из нескольких процессоров и модуля памяти и непосредственно соединен с соседними узлами слева, справа, снизу и сверху от него.

Пример. Intel Paragon.

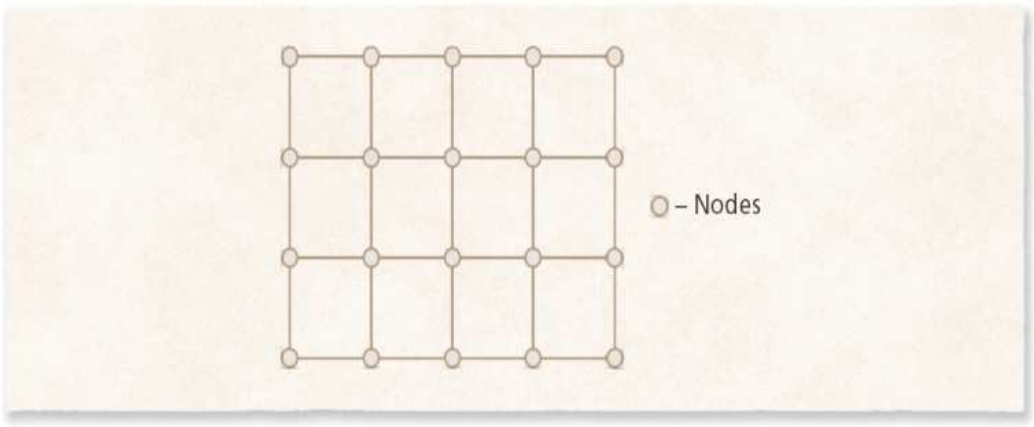


Рис. 3. Двумерная ячеистая сеть

Двумерная ячеистая сеть

- Степень каждого узла равна от 2 до 4, это относительно дешевая архитектура
- Ширина сечения равна min(m,n), значит отказоустойчивость невысока
- Диаметр сети очень велик, значит и производительность невысока
- Эта архитектура используется при построении систем, в которых обмен данными преимущественно осуществляется между соседними узлами

Гиперкуб (hypercube) — схема соединения процессоров, содержащая 2n узлов (см. рис. 4). Каждый узел в такой схеме связан с n

соседними узлами. Например, двумерный гиперкуб — это двумерная ячеистая сеть размерности 2 x 2.

Пример. Системы потоковой обработки видеоданных nCUBE используют гиперкубы до 13 измерений (8912 узлов).

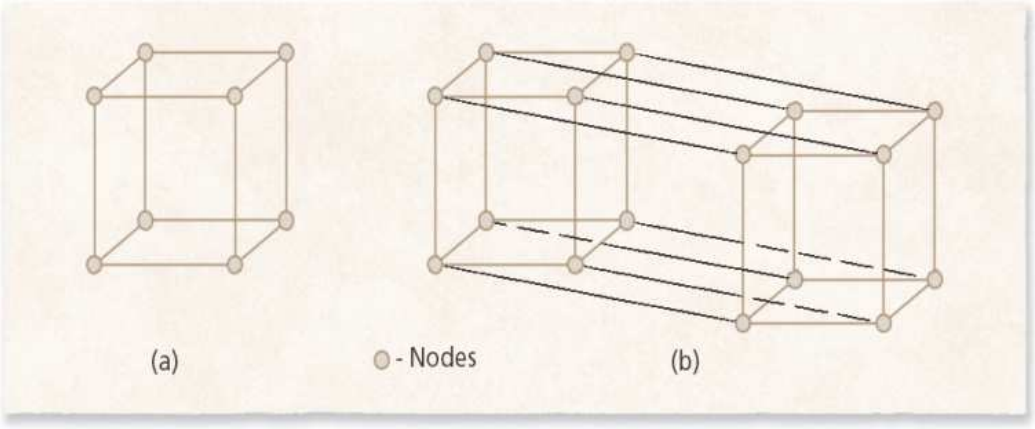


Рис. 4. 3 и 4-мерные гиперкубы

Гиперкуб

- Более производительная и отказоустойчивая, но и более дорогая схема, чем двумерная ячеистая сеть
- Удобная схема для небольшого количества процессоров, более дешевая, чем переключающая матрица

Многоуровневая сеть (multistage network) — схема соединения процессоров, в которой используются специальные узлы-коммутаторы для связи между отдельными процессорными узлами, снабженными локальной памятью (см. рис. 5).

Пример. Суперкомпьютеры IBM серии SP.

Многоуровневая сеть

- Попытка компромисса между ценой и производительностью
- Каждый процессор может связаться с каждым, не передавая данные между промежуточными процессорами

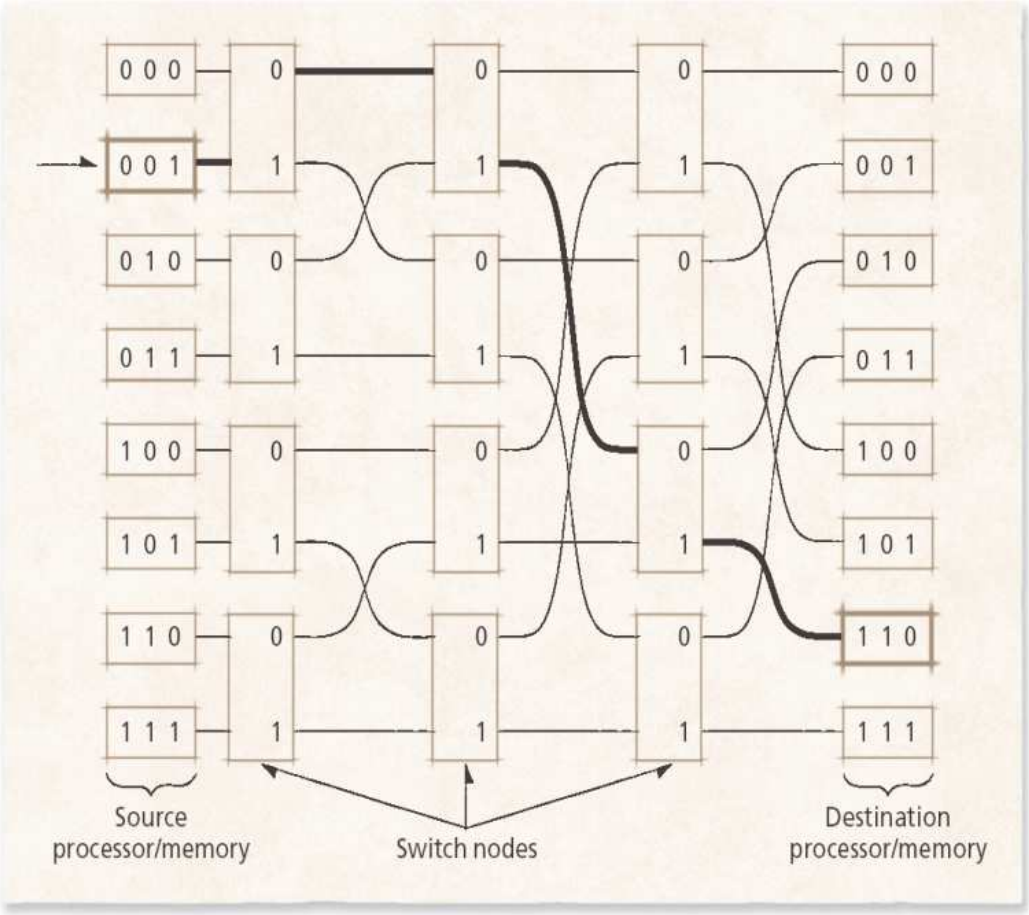


Рис. 5. Путь между процессорами в многоуровневой сети. Направление следующего шага определяется значением соответствующего разряда номера процессора-приемника начиная с младшего (правого) разряда

- Дешевле, чем переключающая матрица но ее производительность и отказоустойчивость ниже

Вопросы для самопроверки

1. Верно ли что диаметр сети, основанный на переключающей матрице, самый маленький? (Да/Нет)

2. Верно ли, что ширина сечения схемы с общей шиной минимальна? (Да/Нет)

3. Верно ли, что стоимость переключающей матрицы самая высокая? (Да/Нет)

4. Верно ли, что четырехмерный гиперкуб — это два трехмерных гиперкуба? (Да/Нет)

Ответы на вопросы

1. Нет. Диаметр схемы с общей шиной тоже равен 1.

2. Да. Достаточно перерезать только одну связь (шину), чтобы сеть перестала функционировать. Все остальные схемы гораздо более отказоустойчивы.

3. Да. Только эта схема обеспечивает связь каждого узла с каждым по отдельному каналу связи. Поэтому количество связей, т.е. стоимость этой схемы, максимальна.

4. Да. Четырехмерный гиперкуб — это два трехмерных гиперкуба, в которых связаны между собой соответствующие узлы. Это же справедливо и для гиперкубов других размерностей.

§ 3. Тесносвязанные и слабосвязанные системы

Классификация по способу распределения ресурсов между процессорами

- Тесносвязанные системы
- Слабосвязанные системы

Тесносвязанная система (tightly coupled system) — система, в которой каждому процессору доступно большинство ее ресурсов (см. рис. 6). В таких системах часто используется общая шина и процессоры обмениваются данными через общую физическую память.

Пример. Dual-processor Intel Pentium.

Слабосвязанная система (loosely coupled system) — система, в которой каждый процессор не может напрямую обратиться к любому ресурсу (см. рис. 7). Компоненты обычно соединяются специальными каналами связи, взаимодействие процессоров, у каждого из которых есть своя собственная память, осуществляется передачей сообщений.

Пример. Суперкомпьютер Earth Simulator.

Сравнительные характеристики

- Слабосвязанные системы более расширяемые и отказоустойчивые, но менее производительны, чем тесносвязанные

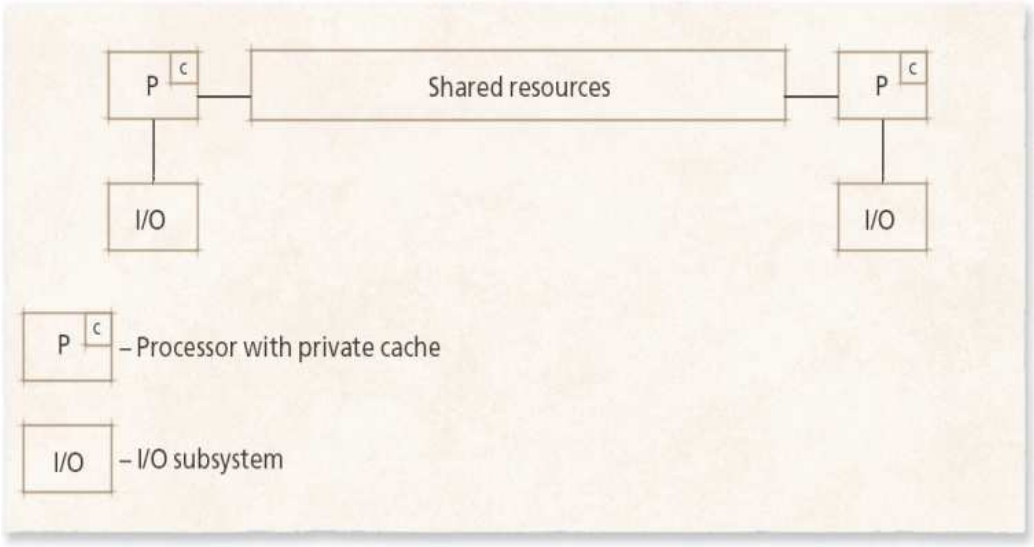


Рис. 6. Тесносвязанная система

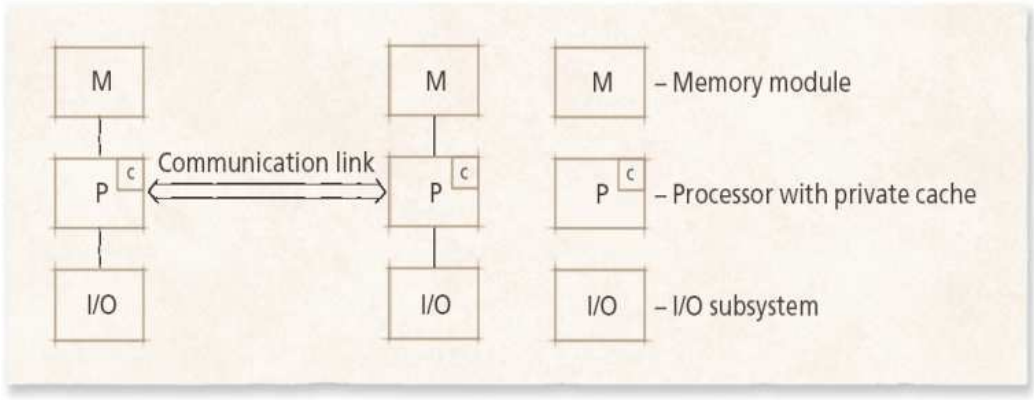


Рис. 7. Слабосвязанная система.

- Создание системного программного обеспечения систем для тесносвязанных систем проще, чем для слабосвязанных

Вопросы для самопроверки

1. Тесносвязанные системы эффективны для создания ЭВМ с большим числом процессоров? (Да/Нет)

2. Тесносвязанные системы редко используются на практике? (Да/Нет)

Ответы на вопросы

1. Нет. Большие системы обычно создаются слабосвязанными, чтобы уменьшить вероятность конфликтов при доступе к разделя-

емым ресурсам и вероятность выхода системы из строя при отказе одного компонента.

2. Нет. Маленькие системы обычно создаются тесносвязанными, поскольку в них невысока вероятность конфликтов при доступе к разделяемым ресурсам. Кроме того, многие крупные системы состоят их групп тесносвязанных компонентов, соединенных каналами связи в слабосвязанную систему.

§ 4. Многопроцессорные операционные системы

Схемы организации операционных систем для многопроцессорных ЭВМ

- Схема ведущий / ведомый
- Схема с раздельными ядрами
- Схема с симметричным обслуживанием всех процессоров

Многопроцессорная система схемы ведущий / ведомый (master / slave multiprocessor organization) — система, в которой на одном процессоре (ведущем, главном) выполняется операционная система, а остальные процессоры (ведомые, подчиненные) заняты выполнением только пользовательских задач (см. рис. 8).

Пример. Системы nCUBE.

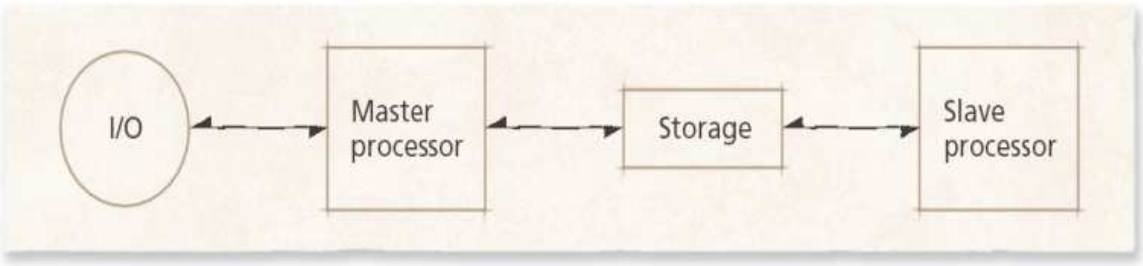


Рис. 8. Многопроцессорная система схемы ведущий / ведомый. Здесь Storage — устройство хранения

Особенности схемы ведущий / ведомый

- Асимметричность аппаратуры — когда процессу, выполняющемуся на ведомом процессоре, требуется услуга операционной системы, он генерирует прерывание и ожидает, пока ведущий процессор его обработает
- Вводом / выводом управляет только ведущий процессор, если выполняемые на ведомых процессорах задачи требуют интенсивного ввода / вывода, они будут часто обращаться к ведущему процессору
- При отказе ведущего процессора система становится полностью неработоспособной

Вопросы для самопроверки

1. Системы схемы ведущий / ведомый хорошо масштабируются до больших размеров? (Да/Нет)

2. Схема ведущий / ведомый хорошо подходит для интенсивных вычислений? (Да/Нет)

Ответы на вопросы

1. Нет. Операционная система выполняется только на одном процессоре. Между пользовательскими процессами, выполняющимися на других процессорах, возникает конкуренция при обращении к функциям ввода / вывода.

2. Да. Системы такой схемы лучше всего подходят для сред, в которых большая часть задач связана с интенсивными вычислениями. Эти задачи будут выполняться ведомыми процессорами без частых обращений к ведущему.

Многопроцессорная система с разделенными ядрами (separate kernels multiprocessor organization) — система, в которой на каждом процессоре выполняется отдельная операционная система, но процессы могут обращаться к некоторым глобальным данным (например, к списку всех работающих в системе процессов).

Пример. Отказоустойчивые системы Tandem.

Особенности схемы с разделенными ядрами

- Процесс, запущенный на определенном процессоре, выполняется на нем до своего завершения
- Каждый процессор работает с ресурсами, доступными только ему, например, файлами и устройствами ввода / вывода
- Полный отказ системы практически не возможен, система будет работать даже на одном последнем процессоре

Вопрос для самопроверки

1. Схема с разделенными ядрами более отказоустойчива, чем схема ведущий / ведомый? (Да/Нет)

Ответы на вопрос

1. Да. Схема с разделенными ядрами является слабосвязанной. У каждого процессора есть свои ресурсы, и он не взаимодействует с другими процессорами при выполнении своих задач. Если любой из процессоров выходит из строя, остальные продолжают работать. При отказе ведущего процессора схема ведущий / ведомый полностью отказывает.

Многопроцессорная система симметричной схемы (symmetrical multiprocessor organization) — система, в которой операционная система может выполняться одновременно на всех процессорах. Операционная система распоряжается набором одинаковых процессоров, каждый из которых может обращаться к любому устройству ввода / вывода.

Пример. BBN Butterfly.

Особенности симметричной схемы

- Самые мощные, но сложные системы
- Необходимо гарантировать взаимное исключение при работе с общими структурами данных
- Высокая конкуренция процессов при доступе к системным ресурсам

- При отказе одного из процессоров производительность снижается плавно, так как любой процесс может выполняться на любом из процессоров

Вопросы для самопроверки

1. Удвоение количества процессоров в многопроцессорной системе симметричной схемы увеличит ее производительность в два раза? (Да/Нет)

2. Симметричная схема лучше масштабируется, чем схема ведущий / ведомый? (Да/Нет)

3. Симметричная схема обеспечивает лучшую кооперацию между процессорами, чем схема с разделенными ядрами? (Да/Нет)

Ответы на вопросы

1. Нет. Добавление в систему новых процессоров приводит к росту конкуренции при обращении к ресурсам и увеличивает накладные расходы операционной системы, поэтому часть производительности добавленных процессоров теряется.

2. Да. Симметричная схема лучше масштабируется, чем схема ведущий / ведомый, поскольку операционная система может выполняться на всех процессорах.

3. Да. Симметричная схема позволяет обеспечить в полной мере взаимодействие процессов и эффективнее выполнять процессы в параллельном режиме.

§ 5. Архитектуры доступа к памяти

Архитектуры доступа к памяти в многопроцессорных системах

- Архитектура однородного доступа к памяти (UMA)
- Архитектура неоднородного доступа к памяти (NUMA)
- Архитектура без доступа к удаленной памяти (NORMA)

Однородный доступ к памяти (uniform memory access, UMA) — архитектура многопроцессорных систем, позволяющая всем процессорам обращаться к общей памяти; в общем случае время доступа к

памяти в такой системе постоянное, независимо от того, какой процессор запрашивает данные, за исключением случаев, когда запрашиваемые данные находятся в кэше процессора (см. рис. 9).

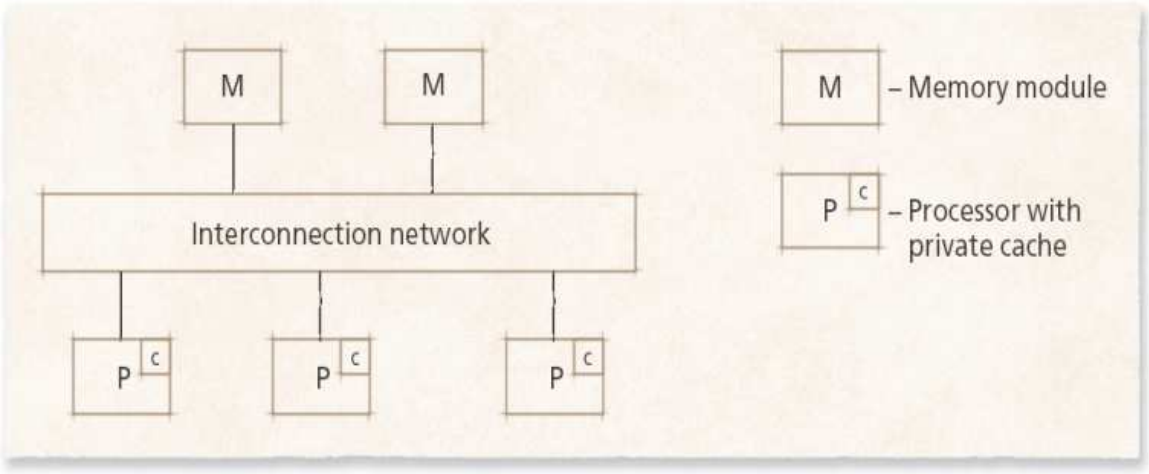


Рис. 9. Многопроцессорная система архитектуры UMA

Многопроцессорная система архитектуры UMA

- Также имеет название симметричной многопроцессорной системы, поскольку в ней любой процесс может выполняться на любом процессоре и все ресурсы (включая память, устройства ввода / вывода и процессы) доступны для всех процессоров
- Обычно используется в небольших системах (обычно — от двух до восьми процессоров)
- В качестве сети соединений используется общая шина или переключающая матрица

Вопросы для самопроверки

1. Ячеистые сети и гиперкубы подходят для соединений процессоров в системах архитектуры UMA? (Да/Нет)

2. Архитектура UMA не очень хорошо масштабируется? (Да/Нет)

Ответы на вопросы

1. Нет. В ячеистых сетях и гиперкубах процессоры и память размещаются в каждом узле, и обращения к локальной памяти осуществ-

ляются быстрее, чем к удаленной. Поэтому доступ к памяти в этих схемах соединений не будет однородным.

2. Да. В архитектуре UMA легко наступает насыщение шины, если одновременно большое число процессоров обращается к памяти, а переключающие матрицы весьма дороги даже при использовании в системах весьма скромного размера.

Неоднородный доступ к памяти (non-uniform memory access, NUMA) — архитектура многопроцессорных систем, в которой каждый узел состоит из одного или нескольких процессоров с кэшем и модуля памяти (см. рис. 10). Доступ к памяти, находящейся в том же узле, выполняется намного быстрее, чем к памяти, расположенной в других узлах.

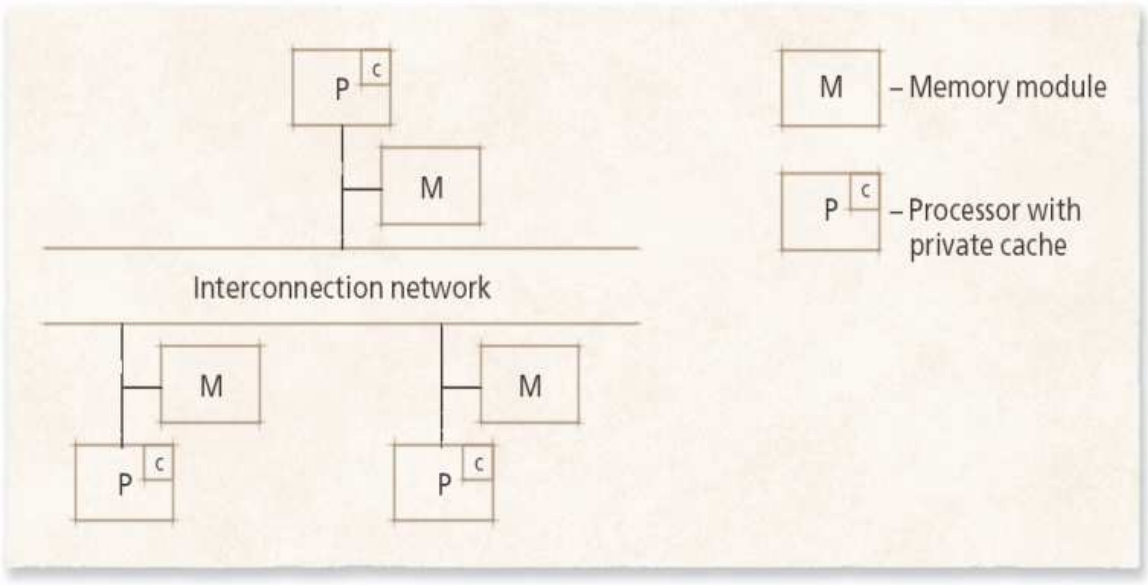


Рис. 10. Многопроцессорная система архитектуры NUMA

Многопроцессорная система архитектуры NUMA

- Более масштабируема, чем архитектура UMA, так как благодаря наличию локальной памяти процессы реже обращаются к шине
- Обычно системы, основанные на архитектуре NUMA, состоят из большего числа процессоров, чем системы архитектуры UMA

- В качестве сети соединений может быть использована любая из рассмотренных ранее схем соединений процессоров

Вопросы для самопроверки

1. Системы NUMA эффективнее, чем системы UMA при небольшом количестве процессоров? (Да/Нет)

2. Возрастет ли производительность системы NUMA, если операционная система обеспечит размещение процесса и памяти, с которой этот процесс работает в одном узле? (Да/Нет)

Ответы на вопросы

1. Нет. Поскольку системы UMA в отличие от систем NUMA обеспечивают одинаково быстрый доступ ко всей памяти для всех процессоров.

2. Да. Если данные, используемые процессом, находятся в удаленной памяти, то производительность системы снижается.

Архитектура без доступа к удаленной памяти (no remote memory access, NORMA) — архитектура многопроцессорных систем, в которой нет общей памяти, есть только локальная память компьютеров-узлов, в таких системах для обмена данными используется сетевой интерфейс либо общая виртуальная память (см. рис. 11).

Общая виртуальная память (shared virtual memory, SVM) — расширение концепции виртуальной памяти, используемое в многопроцессорных системах; SVM создает иллюзию присутствия в системе общей физической памяти.

Многопроцессорная система архитектуры NORMA

- Часто использование SVM не эффективно, так как процессорам приходится передавать целые страницы данных по медленным каналам связи
- Узлы в системах NORMA, не использующих SVM, для обмена данными применяют протоколы передачи сообщений
- Системы NORMA представляют собой распределенные системы, управляемые одной операционной системой

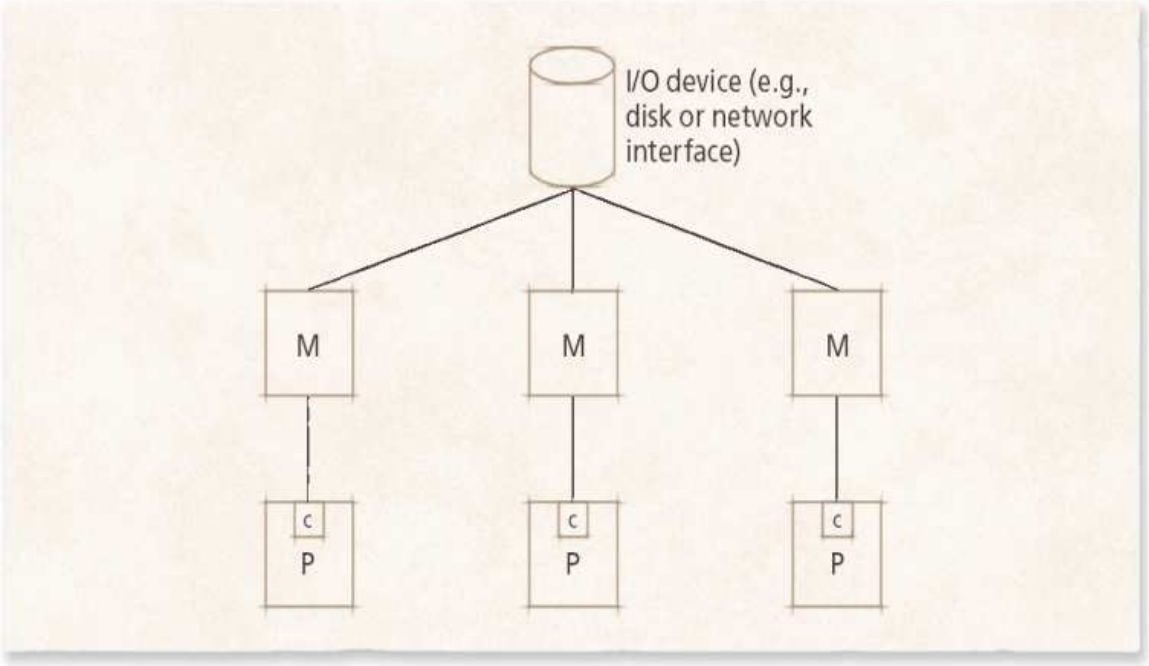


Рис. 11. Многопроцессорная система архитектуры NORMA

- Поисковый портал Google, службы которого выполняются с помощью 15000 серверов начального уровня, — пример системы NORMA

Вопросы для самопроверки

1. Архитектура NORMA плохо подходит для создания кластеров рабочих станций? (Да/Нет)

2. Есть ли среды, где архитектура NORMA будет бесполезна? (Да/Нет)

Ответы на вопросы

1. Нет. Пользователи в кластере рабочих станций обычно не обращаются к памяти, используемой другими пользователями. Обычно они используют общие ресурсы, например, файловую систему и процессоры. Архитектура NORMA обеспечивает доступ к этим ресурсам.

2. Да. Системы NORMA бесполезны в средах, в которых нужен доступ к общей памяти, особенно в системах с одним пользователем или с небольшим количеством процессоров. В таких средах лучше использовать архитектуры UMA или NUMA.

Глава 14

СЕТИ ЭВМ

§ 1. Топологии и типы сетей

Топология сети (network topology) — логическая схема сети, отражающая непосредственные связи узлов сети между собой. Наиболее распространенными топологиями являются: общая шина, кольцо, звезда, дерево, ячеистая и полносвязная топология, а также специальные (спонтанные) сети (см. рис. 1).

Узел сети (host) — объект, например, компьютер или сотовый телефон, обладающий доступом в сеть, принимающий и/или предоставляющий услуги по сети. Другое название — хост.

Канал связи (link) — среда передачи данных, используемая различными сетевыми службами для подключения друг к другу узлов сети.

Общая шина или линейная сеть (bus or linear) — топология сети, в которой все узлы подсоединены к одной шине данных. Поскольку в сетях с общей шиной отсутствуют промежуточные узлы для ретрансляции сообщений, с целью ограничения затухания сигнала длина линий связи должна быть ограничена. Отказ шины приводит к отказу всей сети.

Топология кольцо (rink network) — сеть, состоящая из узлов, каждый из которых соединен непосредственно с двумя соседними узлами. Такие сети могут иметь большую протяженность, так как в них каждый узел выступает в роли ретранслятора. Это позволяет компенсировать затухание сигнала, но приводит к задержке при его передаче. Отказ любого узла приводит к отказу сети.

Топология звезда (star network) — сеть, состоящая из центрального узла или концентратора (hub), к которому непосредственно подключены все остальные узлы сети. Концентратор отвечает за обмен

сообщениями между узлами сети. Размер сети ограничен затуханием сигнала, но задержки при его передаче меньше, чем в топологии кольцо. Отказ концентратора — отказ сети.

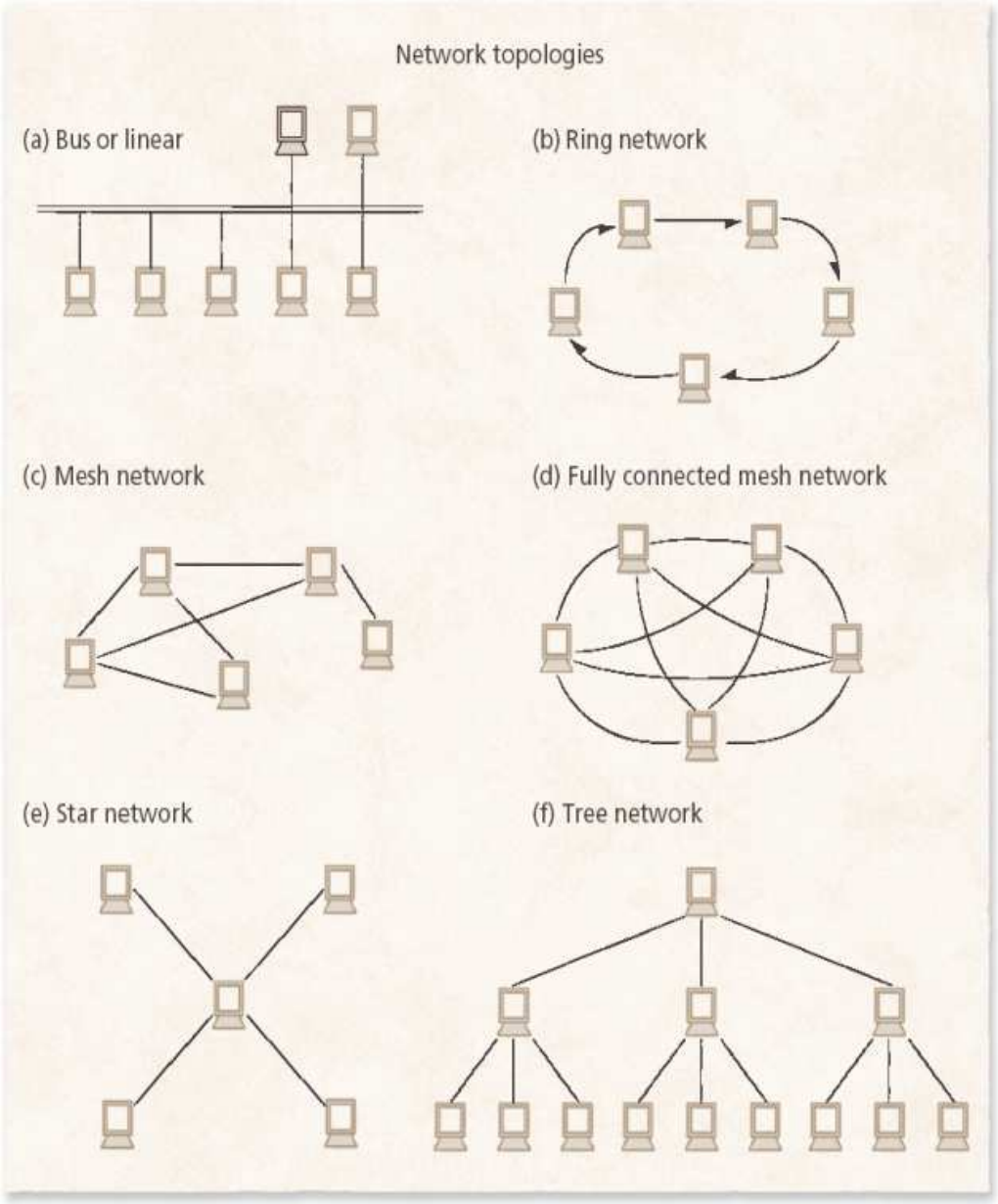


Рис. 1. Топологии сети

Топология дерево (tree network) — сеть с иерархической структурой, включающая в себя подмножество сетей с топологией звезда. В такой сети концентратор первой сети с топологией звезда является

корневым узлом дерева. Каждый узел сети, подключенный к корневому узлу, может выступать в роли концентратора для другой подсети, являясь корневым узлом вложенного поддерева.

Ячеистая топология (mesh network) — сеть, в которой, по крайней мере, два узла связаны между собой непосредственно более чем одним каналом. Более быстродействующие и отказоустойчивые, чем все остальные виды сетей за исключением полносвязных.

Полносвязная топология (fully-connected mesh network) — ячеистая сеть, в которой каждый узел сети непосредственно связан с любым другим узлом. Такая топология самая быстрая и отказоустойчивая, но может быть реализована лишь для самых малых сетей из-за высокой стоимости, связанной с необходимостью прокладки линий связи между каждой парой компьютеров.

Специальные сети (ad hoc network) — сети, характерной чертой которых является спонтанность; в любой момент времени к ним может быть подключено произвольное количество проводных либо беспроводных устройств. Топология сети может быстро меняться, что делает управление сетью с помощью центрального узла достаточно сложным.

Вопросы для самопроверки

1. Можно ли для управления АЭС использовать сеть топологии кольцо? (Да/Нет)

2. Для работы беспроводных устройств необходимы спонтанные сети? (Да/Нет)

Ответы на вопросы

1. Нет. В системах, предназначенных для решения жизненно важных задач, топология кольцо не используется из-за ее низкой отказоустойчивости. Жизненно важные сети нуждаются в нескольких уровнях избыточности для гарантии непрерывного функционирования.

2. Да. Беспроводные устройства предназначены для перемещения с места на место. Работая, они подключаются к различным сетям и отключаются от них. Именно спонтанные сети не требуют наличия фиксированной сетевой топологии.

Разновидности сетей

- Локальные (local area network, LAN)
- Глобальные (wide area network, WAN)

Локальные сети (local area network, LAN) — разновидность сетей, применяемых для соединения между собой различных ресурсов с помощью высокоскоростных каналов передачи данных, оптимизированных для использования в ограниченных пространствах, например, офисных зданиях либо университетских комплексах.

Глобальные сети (wide area network, WAN) — разновидность сетей, позволяющих объединять две и более локальные сети, которые могут находиться на большом расстоянии друг от друга. Обычно для глобальных сетей характерно использование ячеистой топологии и широкополосных соединений. Крупнейшей глобальной сетью является сеть Internet.

Вопросы для самопроверки

1. Имеют ли преимущества локальные сети перед глобальными? (Да/Нет)

2. Уровень ошибок в локальных сетях выше, чем в глобальных? (Да/Нет)

Ответы на вопросы

1. Да. Подсеть, например, университетского комплекса, позволяет напрямую объединить группу компьютеров при помощи высокоскоростных каналов передачи данных, обеспечив более высокую пропускную способность, гибкость управления и настройки.

2. Нет. Для глобальных сетей характерен более высокий уровень ошибок, поскольку в них имеет место взаимодействие многих, часто разнородных по своей топологии локальных сетей.

§ 2. Стек протоколов TCP/IP

Стек протоколов TCP/IP (TCP/IP protocol stack) — иерархическое разделение функций взаимодействия компьютеров на четыре

отдельных уровня абстракции. В стек протоколов TCP/IP входят: прикладной, транспортный, сетевой и канальный уровни.

Стек протоколов TCP/IP

- Прикладной уровень отвечает за реализацию протоколов для приложений, таких как веб-службы и веб-серверы, для их общения между собой
- Транспортный уровень отвечает за передачу информации с одного конца соединения на другой — от процесса, занятого отправкой данных, до процесса, занятого их приемом
- Сетевой уровень отвечает за ретрансляцию данных с одного узла сети на другой, пока данные не попадут адресату (маршрутизация)
- Канальный уровень занимается преобразованием информационных битов в электромагнитный сигнал и их передачей по физической среде передачи данных (сетевой кабель, радиосвязь :)

Кадр (frame) — фрагмент данных в канальном уровне. При передаче сообщений они разбиваются на кадры, в начало и конец которых каждый уровень добавляет свою управляющую информацию (например, адрес отправителя и получателя, тип и размер данных), которая позволяет общаться между собой протоколам одного уровня на разных узлах.

Передача сообщения

- После помещения сообщения в стек, оно разбивается на кадры и последовательно проходит через все уровни
- Каждый из уровней добавляет свою управляющую информацию в начало и конец каждого кадра
- Сформированный кадр передается на следующий уровень либо окончательно в физическую среду передачи данных

Прием сообщения

- Каждый последующий уровень принимает данные от нижележащего уровня либо из физической среды передачи данных
- Каждый уровень удаляет из принятого кадра управляющую информацию для данного уровня, записанную другим узлом
- Эти данные используются для проверки корректности сообщения и того факта, что его получателем является данный узел

Рис. 2. Передача и прием сообщения

Вопросы для самопроверки

1. Верно ли, что управляющая информация помещается в начало кадра? (Да/Нет)

2. Верно ли, что стек протоколов TCP/IP состоит из четырех уровней? (Да/Нет)

Ответы на вопросы

1. Нет. Управляющая информация помещается как в начало, так и в конец каждого кадра. Здесь могут содержаться адреса узлов сети, а также тип или размер передаваемых данных.

2. Да. Прикладной уровень отвечает за взаимодействие приложений, работающих на различных узлах сети; транспортный — за связь между концами соединения; сетевой — за отправку пакетов на другой узел сети по направлению к получателю; канальный — за передачу информации по физической среде передачи данных.

§ 3. Прикладной уровень

Прикладной уровень (application layer) позволяет приложениям на удаленных узлах сети взаимодействовать между собой. Протоколы этого уровня служат, например, для удаленного открытия файлов, отправки запросов к веб-страницам, передачи электронных сообщений либо вызова удаленных процедур.

Некоторые протоколы прикладного уровня

- Гипертекстовый протокол передачи данных (Hypertext Transfer Protocol, HTTP)
- Протокол передачи файлов (File Transfer Protocol, FTP)
- Простой протокол электронной почты (Simple Mail Transfer Protocol, SMTP)
- Служба доменных имен (Domain Name System, DNS)

Универсальный идентификатор ресурсов (Uniform Resource Identification, URI) — имя конкретного ресурса на удаленном узле сети. Многие протоколы прикладного уровня работают с удаленными ресурсами. Эти ресурсы распознаются при помощи URI.

Универсальный указатель ресурса (Uniform Resource Locator, URL) — URI, используемый в протоколах HTTP и FTP. Включает имя протокола, имя узла сети, номер порта и путь к ресурсу.

Пример. http://www.acm.org/dl/faq.html (для http стандартный номер порта 80, он явно не указывается)

Порт (port) определяет в системе специальный сокет, который будет принимать данные. Например, протокол HTTP по умолчанию использует порт 80, а FTP — 20 и 21.

Сокет (socket) — программная конструкция, являющаяся конечным элементом соединения. Процессы используют сокеты для отправки и приема сообщений по сети.

Name	Function
CDUP	Change from the current directory to the parent of the current directory.
CWD	Change the working directory.
PWD	Print the path of the working directory.
LIST	List the contents of the working directory.
DELE	Delete the specified file.
RETR	Retrieve the specified file.
STOR	Upload the specified file.
QUIT	Terminate the FTP session.

Рис. 3. Некоторые команды FTP

Гипертекстовый протокол передачи данных (Hypertext Transfer Protocol, HTTP) — протокол прикладного уровня, используемый для передачи документов HTML и данных других форматов (текст, изображение, аудио, видео, приложение). Работа протокола HTTP состоит из отправки запроса к ресурсу и получения ответа от удаленного узла сети.

Запрос HTTP (HTTP request) — запрос на предоставление ресурса. Включает оператор и URL ресурса. Оператор, описывает действия, которые должны быть применены к ресурсу. Удаленный компьютер обрабатывает запрос и отправляет отклик HTTP.

Отклик HTTP (HTTP response) — сообщение, возвращаемое в ответ на запрос HTTP. В заголовке содержит код, сообщающий клиенту о том, правильно ли был выполнен запрос. Если запрос был обработан корректно, вместе с заголовком будет возвращен запрошенный ресурс. В заголовке также определяется, какой тип имеет данный ресурс.

Протокол передачи файлов (File Transfer Protocol, FTP) — протокол прикладного уровня, позволяющий передавать файлы между двумя узлами сети с помощью команд и ответов на них (см. рис. 3). В протоколе FTP соединение устанавливается между двумя парами

портов: одна пара (обычно, 21) передает управляющую информацию для контроля сеанса, а по другой паре портов (обычно, 20) передаются сами данные.

Вопросы для самопроверки

1. Является ли номер порта частью идентификатора URL? (Да/Нет)

2. Может ли клиент отправить HTTP запрос к несуществующему ресурсу? (Да/Нет)

3. В протоколе FTP для взаимодействия узлов друг с другом обычно используется порт 20? (Да/Нет)

Ответы на вопросы

1. Да. Порт идентифицирует сокет компьютера, в который должно быть передано сообщение.

2. Да. Сервер вернет ответ HTTP с кодом ошибки в заголовке.

3. Нет. Для взаимодействия узлов в сети в протоколе FTP обычно используются два порта: 20 для передачи данных и 21 для контроля сеанса.

§ 4. Транспортный уровень

Транспортный уровень (transport layer) отвечает за сохранность данных во время доставки с одного конца соединения на другой. Он принимает данные от прикладного уровня, разбивает их на фрагменты меньшего размера, пригодные для транспортировки, добавляет в них управляющую информацию и передает дальше на сетевой уровень. Этот уровень реализуется с предварительной установкой соединения и без.

Протоколы транспортного уровня

- TCP — с предварительной установкой соединения
- UDP — без предварительной установки соединения

Протокол управления передачей (Transmission Control Protocol, TCP) — протокол с установлением соединения, гарантирующий сохранность фрагментов данных (называемых в TCP сегментами) на

пути от отправителя к получателю. Используется, например HTTP и FTP. Для создания соединения в TCP используется трехшаговое подтверждение.

Сегмент (segment) — фрагмент данных в протоколе TCP. Включает тело сообщения и заголовок TCP.

Трехшаговое подтверждение

- Одной из целей такого подтверждения является синхронизация порядковых номеров сегментов между узлами сети
- Порядковый номер следующего сегмента при отправке увеличивается на единицу, номер помещается в заголовок
- Система получатель использует эти номера для сортировки поступивших сегментов, если они поступили не в том порядке, в котором были отправлены
- Вначале отправитель отправляет сегмент синхронизации (synchronization segment, SYN) получателю, содержащий запрос на установку соединения и порядковый номер отправленного сегмента
- В ответ получатель возвращает сегмент синхронизации / подтверждения приема (synchronization / acknowledgement segment, SYN/ACK), подтверждающий установку соединения и содержащий порядковый номер принятого сегмента
- На третьем шаге отправитель отсылает сегмент подтверждения приема (ACK), означающий окончательную установку соединения между узлами сети
- В ответ на каждый принятый сегмент адресат возвращает в ответ сегмент подтверждения приема ACK, содержащий порядковый номер принятого сегмента
- Если для сегмента с определенным номером отправитель не получит подтверждения о приеме, он выполнит повторную отправку этого сегмента

- Данная технология гарантирует, что все сегменты будут получены адресатом, продублированные сегменты будут удалены, а сегменты, поступившие в неверном порядке, отсортированы

Протокол передачи дейтаграмм пользователя (User Datagram Protocol, UDP) относится к протоколам без предварительной установки соединения. Для протокола UDP характерен минимум накладных расходов. Однако нет гарантии, что фрагменты данных (называемые в UDP дейтаграммами) достигнут получателя и поступят в нужном порядке. Используется, например, в потоковых видео и аудио приложениях.

Вопросы для самопроверки

1. Отреагирует ли узел сети, отославший дейтаграмму UDP, при ее потере? (Да/Нет)

2. Протокол UDP по сравнению с TCP повышает нагрузку на сеть? (Да/Нет)

3. Существуют приложения устойчивые к потере фрагментов данных? (Да/Нет)

Ответы на вопросы

1. Нет. Узел никак не отреагирует. В протоколе UDP отсутствуют механизмы, с помощью которых узел-отправитель узнал бы о потере дейтаграммы. Поэтому повторная отправка дейтаграммы производиться не будет.

2. Нет. Протокол UDP по сравнению с TCP снижает нагрузку на сеть, поскольку не требует отправки пакетов установки соединения, подтверждения приема и повторной передачи данных.

3. Да. Например, приложения для воспроизведения потокового видео. Последствия при потере дейтаграмм сводятся к паузам при воспроизведении. Если же такие приложения будут работать с протоколом TCP, то паузы будут больше, поскольку воспроизведение будет продолжено только после прихода потерянного сегмента.

§ 5. Сетевой уровень

Сетевой уровень (network layer) принимает данные от транспортного уровня и отвечает за отправку полученных порций данных

(называемых на этом уровне дейтаграммами) на следующий узел сети, расположенный на пути к получателю. На какой узел отправляется дейтаграмма определяется в результате маршрутизации. Два наиболее распространенных протокола сетевого уровня — IP и IPv6.

Маршрутизация (routing) — определение оптимального пути (маршрута) между двумя узлами сети и отправка дейтаграмм по этому пути. Маршрутизацию осуществляют маршрутизаторы с помощью информации, распространяющейся в сети по протоколам маршрутизации.

Маршрутизатор (router) — коммутирующий компьютер, принимающий дейтаграммы с одной или нескольких линий и отправляющий их дальше по одной или нескольким линиям. Маршрутизаторы объединяются в большие сети ячеистой топологии. Каждая дейтаграмма передается от одного маршрутизатора к другому до тех пор, пока она не достигнет адресата.

Протокол маршрутизации (router protocol) — набор правил, определяющий распространение в сети информации о сетевой топологии и качестве линий связи, включая мощность сигнала, количество ошибок и помехи. Основываясь на этой информации каждый маршрутизатор вычисляет следующий маршрутизатор, на который будет направлена дейтаграмма.

Протокол маршрутной информации (Routing Information Protocol, RIP) — протокол маршрутизации прикладного уровня, применяемый в небольших сетях. Требует от каждого маршрутизатора сети передачи полной таблицы маршрутизации — иерархической матрицы, описывающей текущую топологию сети, — своим ближайшим соседям. Это процесс продолжается до тех пор, пока о текущей топологии не узнают все маршрутизаторы.

Вопрос для самопроверки

1. Отвечает ли сетевой уровень за доставку данных от отправителя к получателю? (Да/Нет)

Ответ на вопрос

1. Нет. Сетевой уровень определенного узла сети отвечает только за доставку дейтаграммы на следующий узел сети по направлению к получателю. За доставку данных от отправителя к получателю отвечает транспортный уровень.

Протокол Интернета (Internet Protocol, IP) — протокол сетевого уровня, используемый для задания адресов при передаче данных по сети. Узлы определяются при помощи 32-битовых чисел (IP-адресов). Имена узлов связаны с IP-адресами через службу имен доменов (Domain Name System, DNS). Прикладной уровень использует URL, а транспортный и сетевой — IP-адреса, получаемые от DNS.

Обращение браузера к http://www.acm.org/dl/faq.html:

1) браузер запрашивает у DNS IP-адрес www.acm.org;
2) DNS отвечает: 199.222.69.151;
3) браузер устанавливает TCP-соединение с портом 80 на хосте 199.222.69.151;
4) браузер запрашивает файл dl/faq.html;
5) сервер www.acm.org посылает файл dl/faq.html;
6) TCP-соединение разрывается;
7) браузер отображает на экране содержимое dl/faq.html.

Протокол Интернета версии 6 (Internet Protocol, IPv6) — новая версия IP, в которой используются 128-битовые адреса и три типа адресации: однонаправленная, широковещательная и альтернативная. Переход на IPv6 происходит постепенно. Часть маршрутизаторов пока не поддерживают IPv6. Они передают дейтаграммы IPv6 внутри дейтаграмм IP.

Адресация в IPv6

- Однонаправленная — дейтаграмма отправляется на один определенный узел

- Широковещательная — дейтаграмма отправляется целой группе узлов сети
- Альтернативная — дейтаграмма отправляется на ближайший из группы узлов сети

Вопросы для самопроверки

1. Заголовок дейтаграммы в протоколе IP содержит адрес отправителя и получателя? (Да/Нет)

2. В протоколе IP предусмотрено больше уникальных адресов, чем в IPv6? (Да/Нет)

3. Альтернативная адресация дает самую большую нагрузку на сеть? (Да/Нет)

Ответы на вопросы

1. Да. Каждая дейтаграмма в этом протоколе начинается с 40-байтового заголовка, содержащего кроме других полей IP-адреса отправителя и получателя.

2. Нет. В протоколе IP предусмотрено 2^{32} или около 4000000000 уникальных адресов. В протоколе IPv6 — 2^{128} или приблизительно $3,4 * 10^{38}$ адресов.

3. Нет. Наибольший трафик возникает при широковещательной адресации. Альтернативная адресация позволяет отправлять дейтаграммы одному узлу некоторой группы, тогда как широковещательная адресация подразумевает отправку дейтаграмм всем узлам этой группы.

§ 6. Канальный уровень

Канальный уровень (link layer) преобразует фрагменты данных (на этом уровне называемые кадрами (frame)) в сигнал (электромагнитный, оптический, ...) для передачи по среде передачи данных (медная жила, оптоволокно, ...). Для выявления поврежденных кадров обычно используется контрольная сумма бит кадра, которая добавляется в его конец. Могут применяться различные методы защиты данных и коррекции ошибок.

Некоторые протоколы канального уровня

- CSMA/CD (множественный доступ с контролем несущей и обнаружением столкновений) для сетей Ethernet
- Token Ring для сетей с топологией кольцо
- FDDI для передачи данных по волоконно-оптическим каналам
- CSMA/CA (множественный доступ с контролем несущей и предотвращением столкновений) для беспроводных сетей

Ethernet (IEEE 802.3) — локальные сети, поддерживающие различные скорости передачи данных, в зависимости от используемых линий связи. В них применяется протокол CSMA/CD.

Множественный доступ с контролем несущей и обнаружением столкновений (Carrier Sense Multiple Access with Collision Detection, CSMA/CD) — протокол, применяемый в сетях Ethernet, который предусматривает проверку трансиверами линий связи перед началом передачи данных, а также обработку конфликтов (случаев передачи данных несколькими трансиверами одновременно).

Трансивер (transceiver) — аппаратное устройство, соединяющее узел сети Ethernet со средой передачи данных. Перед началом передачи данных трансивер прослушивает линию связи. Если линия свободна, он начинает передавать кадр. Из–за задержки передачи сигнала, могут возникать конфликты, которые трансиверы обрабатывают на основе алгоритма экспоненциальной отсрочки.

Экспоненциальная отсрочка

- При конфликте трансивер продолжает передачу кадра, чтобы о конфликте стало известно остальным трансиверам
- Трансивер повторно пытается передать кадр через случайный интервал времени от 0 до T мкс., где T — некоторое число
- В случае повторных конфликтов максимальный интервал T увеличивается в два раза, пока кадр не будет передан

Топологии Ethernet (см. рис. 4)

- Классическая сеть Ethernet является линейной
- Коммутируемая Ethernet имеет топологию звезда; к центральному узлу (коммутатору) подключаются отдельные узлы или другие сети Ethernet
- На коммутаторе группы компьютеров могут обмениваться данными через выделенные порты, что снижает вероятность конфликтов

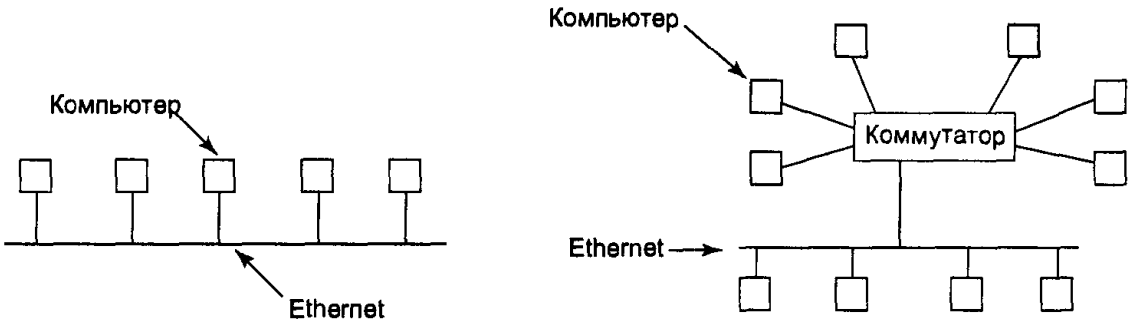


Рис. 4. Классическая и коммутируемая сети Ethernet

Вопросы для самопроверки

1. При передаче данных в сети Ethernet возможно возникновение конфликтов? (Да/Нет)

2. Сеть Ethernet имеет топологию кольцо? (Да/Нет)

3. Трансивер прекращает передачу кадра после обнаружения конфликта? (Да/Нет)

Ответы на вопросы

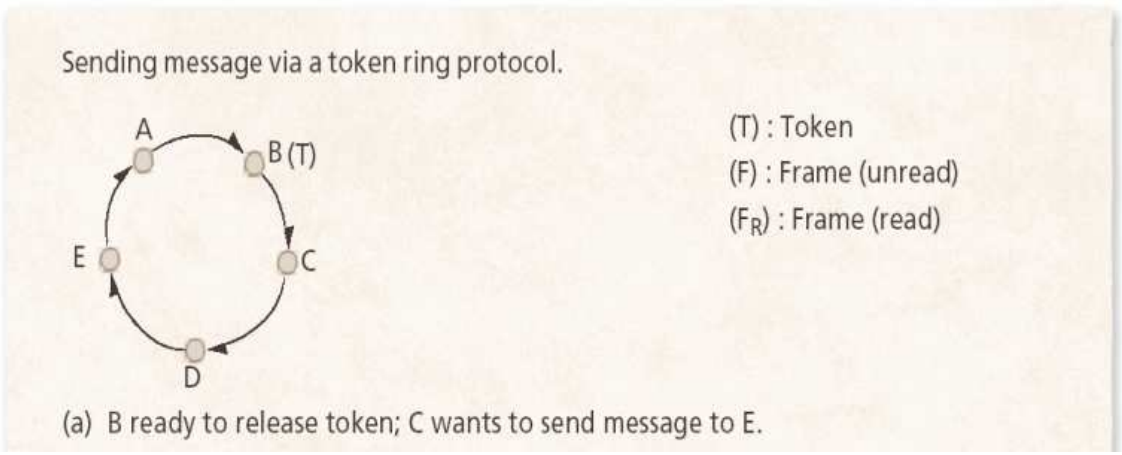
1. Да. Передача данных происходит не мгновенно, поэтому два трансивера могут решить, что линия связи свободна, и начать передачу данных одновременно.

2. Нет. Классическая сеть Ethernet является линейной. Коммутируемая Ethernet имеет топологию звезда.

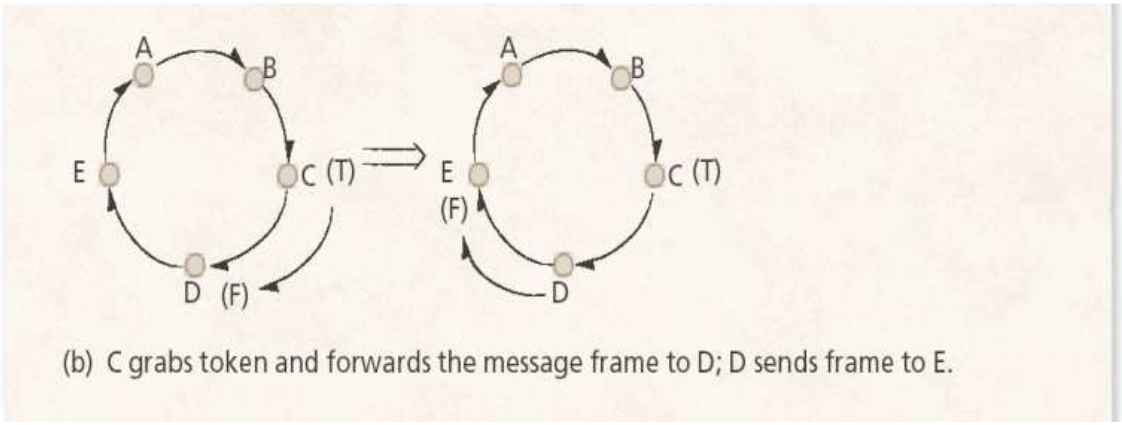
3. Нет. Трансивер продолжает передачу данных в течение определенного промежутка времени после обнаружения конфликта. Если

трансивер прекратит передачу немедленно, то другие трансиверы могут не распознать возникновение конфликта из-за задержек передачи данных по линиям связи.

Token Ring — протокол, в котором для передачи данных используется принцип циклической передачи токена (пустого кадра) по сети с топологией кольцо. Только один узел сети в определенный момент может владеть токеном, и только владелец токена может выполнить передачу данных.

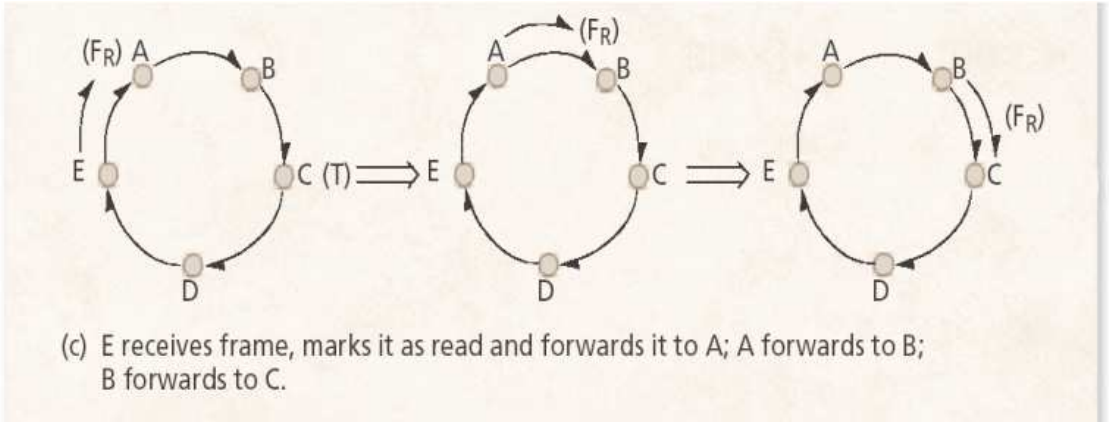


(a) Когда узел C хочет отправить сообщение узлу E, вначале ему следует дождаться приближения пустого кадра (токена).



(b) После получения токена, C записывает в него данные, адрес

получателя и отравляет кадр узлу D; он сравнивает адрес со своим и отправляет кадр дальше.



(c) E receives frame, marks it as read and forwards it to A; A forwards to B; B forwards to C.

(c) Узел E обнаруживает совпадение адресов, считывает данные, помечает кадр как прочитанный и отправляет его дальше; когда кадр достигает C, узел определяет, что сообщение достигло адресата E.

(d) C receives read frame, releases token to D.

(d) После получения прочитанного кадра, узел C удаляет из него данные и отправляет пустой кадр узлу D для дальнейшего использования токена.

Вопросы для самопроверки

1. Время на передачу сообщения в Token Ring непредсказуемо? (Да/Нет)

2. Могут ли все узлы в Token Ring оказаться заблокированными? (Да/Нет)

Ответы на вопросы

1. Нет. Если при передаче токена по сети проблем не возникает, время на передачу сообщения вполне предсказуемо, как и время необходимое на восстановление после различных ошибок.

2. Да. Если на узле, где в данный момент находится токен, произойдет сбой без возможности его устранения, то все узлы будут заблокированы. Поэтому в протоколе Token Ring предусматриваются механизмы защиты от потери токена.

Распределенный интерфейс передачи данных по волоконно-оптическим каналам (Fiber Distributed Data Interface, FDDI) — протокол, построенный на основе двух колец Token Ring, одно из которых является резервным. Ни одна станция не может начать передачу, пока она не получит токен от предыдущей станции. Во время передачи информации токен перестает циркулировать, заставляя все остальные узлы ждать своей очереди.

Вопрос для самопроверки

1. Могут ли в FDDI данные передаваться в противоположном направлении? (Да/Нет)

Ответ на вопрос

1. Да. Когда второе кольцо не требуется для резервных целей, FDDI использует его для передачи токенов и данных в противоположном направлении.

IEEE 802.11 — стандарт беспроводной связи. В соответствии с ним, узлы сети должны работать по протоколу CSMA/CA.

Множественный доступ с контролем несущей и предотвращением столкновений (Carrier Sense Multiple Access with Collision Avoidance, CSMA/CA) — протокол, применяемый в беспроводных сетях стандарта IEEE 802.11. В нем предусмотрена отправка сообщения запроса передачи (RTS) отправителем и возврат сообщения разрешения передачи (CTS) адресатом перед началом передачи данных.

Запрос передачи (Request to Sent, RTS) — широковещательное сообщение, отправляемое беспроводным устройством, которое свиде-

тельствует о желании начать передачу данных, а также содержит информацию о продолжительности передачи, адресах отправителя и получателя. Если среда передачи данных является доступной, адресат возвращает сообщение CTS.

Разрешение передачи (Clear to Sent, CTS) — широковещательное сообщение, говорящее о доступности среды передачи данных. После того как устройство, пожелавшее начать передачу данных, получит сообщение CTS, отправитель и получатель начинают передачу. После получения CTS все остальные станции, желающие начать передачу данных, ждут в течение времени, указанного в CTS.

Вопрос для самопроверки

1. Возникают ли в беспроводных сетях проблемы, не характерные для Ethernet? (Да/Нет)

Ответ на вопрос

1. Да. Беспроводные сети имеют спонтанный характер и могут разделяться физическими объектами, например, стенами зданий. Невозможно гарантировать, что каждое устройство в беспроводной сети будет знать о существовании всех других, поэтому конфликты не исключены.

Глава 15

РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ

§ 1. Связь в распределенных системах

Удаленный вызов процедур (Remote Procedure Call, RPC) — позволяет процессу, выполняющемуся на одном компьютере (клиенте), вызывать процедуры процесса, выполняющегося на другом компьютере (сервере). В RPC используется концепция заглушек.

Заглушка (stub) — специальный процесс, который готовит исходящие данные к передаче и преобразует входящие данные, чтобы те могли корректно интерпретироваться. Заглушки располагаются и на клиенте и на сервере.

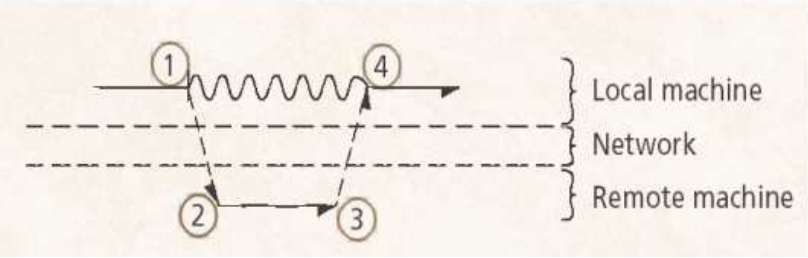


Рис. 1. Удаленный вызов процедуры

Удаленный вызов процедуры (см. рис. 1):

1) локальный процесс вызывает процедуру через клиентскую заглушку; процесс блокируется, заглушка упаковывает параметры и передает запрос на сервер;

2) серверная заглушка принимает запрос, распаковывает параметры и передает параметры соответствующему процессу на сервере;

3) серверный процесс завершает работу вызванной процедуры, заглушка на сервере упаковывает результаты и возвращает их клиенту;

4) клиентская заглушка распаковывает полученные результаты, уведомляет об их получении процесс, отправивший запрос и передает данные этому процессу.

Вопросы для самопроверки

1. Механизм RPC позволяет вызывать процедуры на сервере так же легко, как на локальной системе? (Да/Нет)

2. Поддерживает ли RPC глобальные переменные? (Да/Нет)

Ответы на вопросы

1. Да. Цель RPC — упрощение процесса написания распределенных приложений, при условии сохранения синтаксиса, используемого для вызова локальных процедур.

2. Нет. Процесс, осуществляющий удаленный вызов процедуры и соответствующая ему клиентская заглушка имеют различные адресные пространства данных и не могут иметь общие глобальные переменные.

Удаленный вызов методов (Remote Method Invocation, RMI) является аналогом RPC в языке Java и позволяет сценариям Java выполнять обращение к методам объекта на другом компьютере, используя тот же самый синтаксис, что и при вызове локальных методов.

RMI

- Заглушки — объекты Java, обеспечивающие интерфейс между клиентскими процессами и удаленными объектами
- Объекты передаются между удаленными процессами в упакованном виде в качестве параметров

Вопрос для самопроверки

1. Обладает ли преимуществом механизм RMI перед RPC? (Да/Нет)

Ответ на вопрос

1. Да. RMI позволяет клиентам отправлять объекты в качестве аргументов при удаленном вызове методов и принимать объекты в качестве возвращаемых значений от серверов.

Типовая архитектура брокера объектных запросов (Common Object Request Broker Architecture, CORBA) — стандартная технология построения распределенных объектных приложений, принятая в начале 90-х рабочей группой по развитию стандартов объектного программирования (Object Management Group, OMG).

Распределенная объектная модель программных компонентов (Distributed Component Object Model, DCOM) — архитектура распределенных объектов, разработанная независимо от других разработчиков в начале 90-х компанией Microsoft. Встроена в операционную систему Windows начиная с Windows 95.

CORBA и DCOM

- Подобно RMI поддерживают передачу объектов между процессами
- Независимы от языков программирования
- Приложения, написанные на разных языках, могут взаимодействовать через интерфейсы CORBA или DCOM

Вопросы для самопроверки

1. Может ли пользователь отдать предпочтение RMI вместо CORBA? (Да/Нет)

2. Есть ли общее между DCOM и CORBA? (Да/нет)

Ответы на вопросы

1. Да. Если пользователь имеет дело только с приложениями, написанными на языке Java, он ничего не выигрывает от языковой независимости CORBA, но должен дополнительно изучить интерфейсы CORBA.

2. Да. В обе эти технологии включена поддержка удаленных объектов, написанных для разных платформ с использованием различных языков программирования.

§ 2. Веб-службы

Веб-служба, веб-сервис (web service) — программная система, чьи общедоступные интерфейсы определены на языке XML. Описание этой программной системы может быть найдено другими программными системами, которые могут взаимодействовать с ней согласно этому описанию посредством сообщений, основанных на XML, и передаваемых с помощью интернет-протоколов.

Расширяемый язык разметки (eXtensible Markup Language, XML) — рекомендованный Консорциумом Всемирной паутины язык разметки, представляющий собой свод общих синтаксических правил. XML — это текстовый формат, предназначенный для хранения структурированных данных, для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки.

Веб-службы

- Обеспечивают взаимодействие программных систем независимо от платформы
- Основаны на базе открытых стандартов и протоколов. Благодаря использованию XML достигается простота разработки и отладки веб-служб
- Меньшая производительность и больший размер сетевого трафика по сравнению с технологиями RMI, CORBA, DCOM за счёт использования текстовых XML-сообщений

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Фактически представляет собой операционную систему внутри операционной системы. Основой платформы является виртуальная машина Common Language Runtime (CLR), способная выполнять как обычные настольные программы, так и веб-приложения. Отличительной особенностью .NET Framework является способность выполнять программы, написанные на разных языках программирования.

Архитектура .NET

- Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET понятный человеку низкоуровневый язык Common Intermediate Language (CIL)
- Затем компилятор производит перевод CIL-кода в объектный байт-код (в терминах .NET получается сборка, assembly), а уже байт-код исполняется виртуальной машиной CLR
- Встроенный в виртуальную машину JIT-компилятор <на лету> (just in time — компиляция на лету) преобразует промежуточный байт-код в машинные коды нужного процессора
- Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия
- Виртуальная машина CLR также сама заботится о базовой безопасности и управлении памятью, избавляя разработчика от части работы

Программная совместимость

- Одной из основных идей Microsoft .NET является совместимость программных частей, написанных на разных языках
- Служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi
- На C++ можно написать класс, наследованный от класса, написанного на Visual Basic .NET
- Исключение, созданное методом, написанным на C++, может быть перехвачено и обработано в Delphi

Вопрос для самопроверки

1. Может ли пользователь отдать предпочтение DCOM перед .NET? (Да/Нет)

Ответ на вопрос

1. Да. Если важна высокая производительность и ограничение сетевого трафика.

§ 3. Облачные вычисления

Облачные вычисления (cloud computing) — технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как интернет-сервис.

Рис. 2. Облачные вычисления

IEEE, 2008 год:

- Облачная обработка данных — это парадигма, в рамках которой информация постоянно хранится на серверах в Интернете и временно кэшируется на клиентской стороне, например, на персональных компьютерах, игровых приставках, ноутбуках, смартфонах и так далее.

Облачная обработка данных:

- программное обеспечение как услуга;
- инфраструктура как услуга;
- все как услуга.

Программное обеспечение как услуга (software as a service, SaaS) — бизнес-модель продажи и использования программного обеспечения, при которой поставщик разрабатывает веб-приложение и самостоятельно управляет им, предоставляя заказчикам доступ к программному обеспечению через Интернет. Основное преимущество модели SaaS для потребителя состоит в отсутствии затрат, связанных с установкой, обновлением и поддержкой работоспособности оборудования и работающего на нём программного обеспечения.

SaaS

- Приложение приспособлено для удаленного использования
- Одним приложением пользуется несколько клиентов (приложение коммунально)
- Оплата взимается либо в виде ежемесячной абонентской платы, либо на основе объема операций
- Техническая поддержка приложения включена в оплату
- Модернизация и обновление приложения происходит плавно и прозрачно для клиентов
- Заказчики платят не за владение программным обеспечением как таковым, а за его аренду (то есть за его использование через веб-интерфейс)
- Заказчик несет сравнительно небольшие периодические затраты, и ему не требуется инвестировать значительные средства в приобретение ПО и аппаратной платформы для его развертывания, а затем поддерживать его работоспособность
- Схема периодической оплаты предполагает, что если необходимость в программном обеспечении временно отсутствует, то заказчик может приостановить его использование и заморозить выплаты разработчику

Вопрос для самопроверки

1. Позволяет ли SaaS бороться с нелицензионным ПО? (Да/Нет)

Ответ на вопрос

1. Да. Модель SaaS позволяет эффективно бороться с нелицензионным использованием программного обеспечения, поскольку само программное обеспечение не попадает к конечным заказчикам.

Инфраструктура как услуга (Infrastructure as a Service, IaaS) — это предоставление компьютерной инфраструктуры (как правило в форме виртуализации) как услуги на основе концепции облачных вычислений.

Компоненты IaaS

- Аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование)
- Операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами)
- Связующее ПО (например, для управления системами)

IaaS

- Избавляет предприятия от необходимости поддержки сложных инфраструктур центров обработки данных, клиентских и сетевых инфраструктур
- Позволяет уменьшить связанные с этим капитальные затраты и текущие расходы
- Возможна и дополнительная экономия, если услуги предоставляются в рамках инфраструктуры совместного использования

Вопрос для самопроверки

1. Может ли IaaS приводить к возникновению неконтролируемых данных? (Да/Нет)

Ответ на вопрос

1. Да. Информация, оставленная пользователем, может храниться годами, либо использоваться без его ведома, либо он будет не в состоянии изменить какую-то её часть.

Литература

1. *Гордеев А.В.* Операционные системы: учебник для вузов / А.В. Гордеев. — 2-е изд. — СПб.: Питер, 2006. — 416 с.

2. *Дейтл Х.М.* Операционные системы / Х.М. Дейтл, П.Дж. Дейтл, Д.Р. Чофнес. — 3-е изд. — М.: ООО “Бином пресс”, 2006. — Т. 1. 1024 с., Т. 2. 704 с.

3. *Таненбаум Э.* Современные операционные системы / Э. Таненбаум. — 2-е изд. — СПб.: Питер, 2006. — 1038 с.