МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МИНИСТЕРСТВО ОБРАЗОВАНИЯ СТАВРОПОЛЬСКОГО КРАЯ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
СТАВРОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ

М.Р. Бибарсов, Г.Ш. Бибарсова, Ю.В. Кузьминов

ОПЕРАЦИОННЫЕ СИСТЕМЫ, СРЕДЫ И ОБОЛОЧКИ

Учебное пособие

Ставрополь
2010

УДК 681.3(075.8)
ББК 32.81я 73
Б 59

Печатается по решению
редакционно-издательского совета
ГОУ ВПО Ставропольского государственного
педагогического института

Рецензенты:
доктор педагогических наук, профессор ***А.В. Беляев*** (СГУ)
доктор технических наук, профессор ***П.А. Будко*** (СтГАУ)

БибарсовМ.Р., Бибарсова Г.Ш., Кузьминов Ю.В.
Б 59 **Операционные системы, среды и оболочки**: Учебное пособие. – Ставрополь: Изд-во СГПИ, 2010. – 120 с.

В учебном пособии сформулированы определение и основные функции операционных систем, сред и оболочек, представлены классификация, история развития, функциональные компоненты операционных систем и требования, предъявляемые к ним. Рассмотрены виды архитектур операционных систем, определены критерии эффективности вычислительной системы. Исследовано влияние выбора алгоритмов планирования процессов и потоков, алгоритмов распределения памяти, средств синхронизации, виртуализации памяти и кэширования данных на производительность операционных систем. Изложены понятие и задачи файловой системы, раскрыта логическая и физическая ее организация, приведены задачи подсистемы ввода-вывода. Проведен обзор современных операционных систем, сред и оболочек.

Пособие предназначено для студентов и может быть использовано аспирантами в системе повышения квалификации.

Содержание учебного пособия соответствует программе дисциплины «Операционные системы, среды и оболочки».

УДК 681.3(075.8)
ББК 32.81я 73

Содержание

ВВЕДЕНИЕ 5

1. ОБЩИЕ СВЕДЕНИЯ ОБ ОПЕРАЦИОННЫХ СИСТЕМАХ, СРЕДАХ И ОБОЛОЧКАХ

1.1. Основные функции операционных систем, сред и оболочек 7
1.2. История развития и поколения ОС 11
1.3. Классификация ОС 13
1.4. Функциональные компоненты ОС 14
1.5. Требования к современным ОС 20

2. УПРАВЛЕНИЕ ПРОЦЕССАМИ

2.1. Мультипрограммирование и распределение ресурсов 22
2.2. Понятие процессов и потоков 27
2.3. Алгоритмы планирования процессов и потоков 29
2.4. Синхронизация процессов 35

3. УПРАВЛЕНИЕ ПАМЯТЬЮ

3.1. Функции ОС по управлению памятью 45
3.2. Типы адресов 45
3.3. Виды алгоритмов распределения памяти 49
3.4. Виртуализация памяти. Классы виртуальной памяти 50
3.5. Кэширование данных 59

4. ВВОД-ВЫВОД И ФАЙЛОВАЯ СИСТЕМА

4.1. Файловая система ОС 65
4.2. Логическая организация файловой системы 67
4.3. Физическая организация файловой системы 76
4.4. Подсистема ввода-вывода 81

5. АРХИТЕКТУРА ОС

5.1. Архитектура на базе ядра в привилегированном режиме 86

5.2. Микроядерная архитектура 90
5.3. Переносимость ОС 92

6. ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ И ЭВОЛЮЦИЯ ИХ ФУНКЦИОНАЛЬНЫХ ХАРАКТЕРИСТИК

6.1. Операционные системы разных этапов разработки вычислительных машин 94
6.2. История развития и характеристики операционных систем UNIX 100
6.3. История развития и характеристики операционных систем семейства Windows 102

ЗАКЛЮЧЕНИЕ 108

ЛИТЕРАТУРА 112

СЛОВАРЬ ТЕРМИНОВ И ОПРЕДЕЛЕНИЙ 113

ВВЕДЕНИЕ

Операционные системы являются основой программного обеспечения вычислительных машин. ***Операционная система*** (ОС) – это комплекс управляющих и обрабатывающих программ, который, с одной стороны, выступает как интерфейс между пользователем и аппаратными компонентами вычислительных машин и вычислительных систем, а с другой стороны предназначен для эффективного управления вычислительными процессами, а также наиболее рационального распределения и использования вычислительных ресурсов.

Обеспечение пользователю определенного уровня удобств осуществляется посредством того, что ОС представляет для него так называемую «расширенную» (или виртуальную) машину, которая избавляет пользователя от необходимости работать напрямую с аппаратными компонентами и берет на себя выполнение большинства рутинных операций. Таким образом, абстрактная «расширенная» машина, с которой, благодаря ОС, имеет дело пользователь, гораздо проще и удобнее в обращении, чем реальная аппаратура, лежащая в основе этой абстрактной машины.

Идея о том, что ОС прежде всего система, обеспечивающая удобный интерфейс пользователям, соответствует рассмотрению ее сверху вниз. Другой взгляд, снизу вверх, дает представление об ОС как о некотором механизме, распределяющим и управляющим всеми компонентами и ресурсами ЭВМ с целью обеспечения максимальной эффективности их функционирования.

Таким образом, ОС выполняет функции управления вычислительными процессами в ЭВМ, распределяет ресурсы между различными вычислительными процессами и образует программную среду, в которой выполняются прикладные программы пользователей. Такая среда называется ***операционной средой***.

Благодаря наличию операционной системы пользователи-программисты при написании собственных программ могут вообще не знать многих деталей управления конкретными ресурсами ВМ и ВС, а должны только обращаться к ОС как к некоторой программной подсистеме с соответствующими вызовами и получать от нее необходимые функции и сервисы. Набор таких функций и правил обращения к ним как раз и образуют то базовое понятие, которое называют операционной средой. Таким образом можно сказать, что термин «операционная среда» означает, прежде всего, соответствующие интерфейсы, необходимые программам и пользователям для обращения к ОС с целью получить определенные сервисы.

Параллельное существование терминов **«операционная система»** и **«операционная среда»** вызвано тем, что операционная система в общем случае может поддерживать несколько операционных сред. Операционная среда в свою очередь может включать несколько разных пользовательских и программных интерфейсов. Например, ряд ОС предоставляют для пользователя **интерфейсы командной строки**, **интерфейсы-оболочки** (типа Norton Commander и т.п.), а также **графические интерфейсы**. Если же говорить о программных интерфейсах, то это те интерфейсы, к которым могут обращаться программисты для получения соответствующих функций и сервисов. Можно сказать, что операционная среда – это то системное программное окружение, в кото-

ром могут выполняться программы, созданные по правилам работы этой среды.

Операционная система в значительной степени определяет функциональные возможности, удобства пользования и эффективность работы ВМ и ВС. На сегодняшний день существует большое количество разных типов ОС, отличающихся областями применения, аппаратными платформами и методами реализации. Естественно, это обуславливает и значительные функциональные различия этих ОС. Поэтому при изучении операционных систем очень важно из всего многообразия выделить те функции, которые присущи всем операционным системам как классу программных продуктов. Именно общие понятия ОС, концепции их построения и функциональные возможности являются предметами рассмотрения данного учебного пособия. В заключительных разделах учебного пособия уделено внимание историческим сведениям о возникновении и развитии ОС, а также даны оценки свойств и характеристик современных ОС. Представлены примеры практического построения, состава и особенностей функционирования наиболее популярных и эффективных операционных систем.

Учебное пособие предназначено для студентов очной и заочной форм обучения специальности «Прикладная информатика», изучающих дисциплину «Операционные системы, среды и оболочки», и разработано с целью обеспечения обучающихся и преподавателей систематизированным учебным материалом по теоретическим основам операционных систем.

Рассмотрены основные понятия операционных систем, сред и оболочек, вопросы управления процессами и ресурсами, особенности построения операционных систем для многопроцессорных вычислительных машин и многомашинных вычислительных систем, общие концепции и принципы разработки операционных систем. Приведены краткие исторические сведения о возникновении и развитии операционных систем, а также даны оценки их свойств и характеристик. В качестве примеров практической реализации реально функционирующих современных операционных систем описаны наиболее распространенные и «знаковые» системы семейств UNIX и Windows.

1. ОБЩИЕ СВЕДЕНИЯ ОБ ОПЕРАЦИОННЫХ СИСТЕМАХ, СРЕДАХ И ОБОЛОЧКАХ

1.1. Основные функции операционных систем, сред и оболочек

Операционная система (***ОС***) – система программ, реализующая интерфейс между аппаратурой ЭВМ и пользователями. Согласно своему назначению ОС выполняет два вида взаимосвязанных функций и рассматривается в двух аспектах:

- управление распределением ресурсов вычислительной системы для обеспечения ее эффективной работы; ***ОС является менеджером ресурсов***;

- обеспечение пользователей набором средств для облегчения проектирования, программирования, отладки и сопровождения программ; ***ОС является виртуальной машиной, предоставляющей пользователю удобный интерфейс***.

Схематично роль ОС в организации вычислительного процесса представлена на рисунке 1.1.

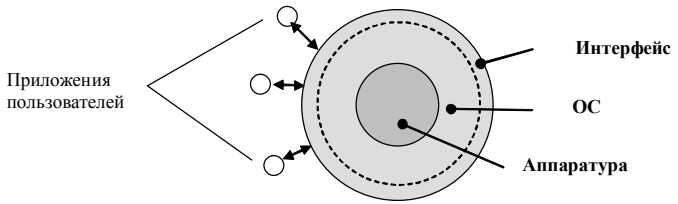


Рисунок 1.1 – ОС в организации вычислительного процесса

ОС как менеджер ресурсов должна обеспечивать:

· загрузку пользовательских программ в оперативную память;

· выполнение этих программ путем организуя работу процессора;

· работу с устройствами долговременной памяти, такими как магнитные диски, ленты, оптические диски и т.д. (как правило, ОС управляет свободным пространством на этих носителях и структурирует пользовательские данные.);

· стандартный доступ к различным устройствам ввода/вывода, таким как терминалы, модемы, печатающие устройства.

При этом в современных вычислительных системах реализуются следующие возможности:

· параллельное (или псевдопараллельное, если машина имеет только один процессор) исполнение нескольких задач;

· распределение ресурсов компьютера между задачами;

· организация взаимодействия задач друг с другом;

· взаимодействие пользовательских программ с нестандартными внешними устройствами;

· организация межмашинного взаимодействия и разделения ресурсов;

· защита системных ресурсов, данных и программ пользователя, исполняющихся процессов и самой себя от ошибочных и зловредных действий пользователей и их программ.

В итоге можно сказать, что функцией ОС как менеджера ресурсов является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы согласно выбранным критериям эффективности. Критерии рассматриваются далее (это, например, пропускная способность или реактивность системы).

Для решения общих задач управления ресурсами разные ОС используют различные алгоритмы, что в конечном счете и определяет их облик в целом, включая характеристики производительности, область применения и пользовательский интерфейс.

ОС как виртуальная машина должна предоставлять некий интерфейс, избавляющий пользователя от непосредственной работы с аппаратурой и организации вычислительного процесса (в частности, распределения ресурсов и их защиты). Здесь можно говорить о двух уровнях (или видах) интерфейсов:

· ***пользовательском***, предназначенном для работы с готовыми приложениями; это все интерфейсные средства в диапазоне от командной строки до развитых графических оболочек;

· ***программном, или интерфейсе прикладного программирования***, представляющем собой средства для обращения к возможностям ОС при создании собственных приложений (фактически это системные функции, доступные разработчику).

Операционная оболочка (operation shell) – комплекс программ, ориентированных на определенную операционную систему и предназначенный для облегчения диалога между пользователем и компьютером при выполнении определенных видов деятельности на компьютере.

Операционные оболочки дополняют и расширяют пользовательский интерфейс ОС за счет наглядного представления объектов (файлов, каталогов, дисков), использования систем меню и горячих клавиш.

Операционные оболочки предоставляют следующие услуги:

· работа с дисками (просмотр дерева каталогов, получение информации о состоянии диска, форматирование дисков);

· работа с файлами и каталогами (создание, просмотр содержимого, копирование, перенос, переименование, удаление, изменение атрибутов файлов и каталогов; редактирование текстовых файлов; создание архивов);

· дополнительные возможности (подключение к сети, создание пользовательских меню, подключение внешних редакторов и др.).

В соответствии со способом представления объектов оболочки можно разделить на два класса:

- ***графические,*** где используются визуальные средства представления (иконки, пиктограммы) и технология манипулирования объектами путем «перетаскивания»;

- ***неграфические (текстовые),*** где объекты представлены именами и обрабатываются посредством команд, систем меню и горячих клавиш.

Это разделение не является жестким, поскольку в большей или меньшей степени средства одного класса присутствуют и в другом.

Виды операционных оболочек и формы их использования определяются основным назначением операционной системы, кругом решаемых задач и уровнем профессионализма пользователя.

- Для современных ОС ***Windows*** (настольных ОС общего назначения) графические оболочки являются «родными», т.е. неотделимы от ОС, тогда как оболочки другого класса устанавливаются как отдельные приложения. Первые ориентированы на предоставление возможности работы с ОС пользователю с минимальным уровнем подготовки, прежде всего непрофессионалу, и не предполагают решения какого-либо специфического класса задач с помощью компьютера. Вторые используются как правило профессионалами, поскольку в подавляющем большинстве реальных применений повышают надежность и эффективность (скорость и качество) работы с данными.

Классическим и наиболее известным представителем неграфических оболочек является Norton Commander – оболочка для ОС MS DOS, принципы построения и функционирования которой легли в основу построения последующих оболочек. В настоящее время для ОС Windows в основном используются оболочки Far Manager и Total Commander, которые постоянно развиваются.

- ОС ***Unix*** как профессиональная серверная, напротив, исходно предполагает только интерфейс командной строки; использование оболочек диктуется желанием повысить удобство работы. ОС ***Linux***, базируясь на принципах Unix, но, претендуя на ту же роль, что и Windows, занимает некое промежуточное положение и *исходно предполагает* использование оболочек, устанавливаемых как компоненты системы, хотя они и представляют собой отдельные приложения. Тем не менее подход к работе с операционными оболочками в этих системах один.

Здесь из популярных текстовых оболочек можно назвать, например, Midnight Commander. Работа с графическими оболочками реализуется несколько иначе, чем в Windows. Подсистема графического интерфейса имеет два компонента. Первый представлен модулем X-server, входящим в ядро ОС. Второй компонент являет собой ряд приложений («менеджеров окон») под общим наименованием X-client; каждое из которых может взаимодействовать с X-serverом по протоколу TCP/IP. Совокупность «X-server + X-client» образует подсистему графического интерфейса, реализующую графическую оболочку. Вариант последней зависит от вида X-client; популярны, например, менеджеры KDE, Gnome, Afterstep и др.

- Наиболее совершенной в плане предоставления пользователю удобств посредством операционных оболочек является операционная система ***MacOS***, располагающая одновременно всеми видами оболочек. Так, система имеет встроенную поддержку графического интерфейса. Для удобства работы профессионалов в версии X имеется юниксоподобная консоль. Также имеется встроенная оболочка Finder, объединяющая в себе основные черты Norton-подобных оболочек и графический интерфейс и системы меню Windows.

Операционные среды. Под операционной средой (operating environment) понимается комплекс средств, обеспечивающих разработку и выполнение прикладных программ и представляющих собой набор функций и сервисов операционной системы и правил обращения к ним. Это понятие отражает аспект рассмотрения операционной системы как виртуальной машины. В общем случае операционная среда включает

операционную систему, программное обеспечение, интерфейсы прикладных программ, сетевые службы, базы данных, языки программирования и другие средства выполнения работы на компьютере – в зависимости от решаемых задач. Очевидно, что операционные оболочки являются компонентами операционной среды.

В такой трактовке примерами операционных сред могут служить следующие:

ОС Windows + Delphi + вспомогательные средства – операционная среда разработчика прикладных приложений;

ОС Windows + Adobe Photoshop + Adobe Illustrator + Macromedia Dreamweaver + Internet Explorer + вспомогательные средства – операционная среда WEB-разработчика;

ОС FreeBSD + WEB-сервер Apache + сервер СУБД MySQL + интерпретатор PHP + программы защиты + вспомогательные средства – операционная среда для создания приложений, работающих на стороне сервера.

Однако использование термина «операционная среда» объясняется прежде всего тем, что *одна операционная система* может поддерживать *несколько операционных сред* путем эмуляции функций *других операционных систем*. Такая поддержка на разных этапах развития ОС в зависимости от целей и класса ОС может быть более или менее целесообразной.

Так, для ОС Unix существует ряд приложений, например, WINE, которые позволяют в некоторой степени эмулировать интерфейс прикладного программирования WIN32API и, таким образом, позволяют запускать некоторые приложения, разработанные для ОС Windows. Однако практическая значимость такой деятельности невелика.

На профессиональных компьютерах и под ОС Unix, и под ОС Windows достаточно успешно эмулируются мобильные ОС, что дает возможность разработки и эксплуатации соответствующих приложений в отсутствие самой мобильной платформы. С усложнением мобильных платформ такая разработка будет все более и более усложняться, и целесообразность ее может снизиться.

Для ОС Windows в период с 1995 года и практически до настоящего времени имела место ситуация, когда 32-разрядные ОС путем эмуляции предшествующих, 16-разрядных, позволяли создать соответствующие операционные среды и, таким образом, работать с приложениями, написанными для старших ОС.

Такая эмуляция более старого программного обеспечения обеспечивает совместимость более ранних версий ОС с более поздними. Например, Windows 95/98 позволяли запускать программы для MS DOS.

Для Windows 2000/XP эта возможность тоже есть, но играет весьма незначительную роль, а с выходом обновления и дополнения Service Pack 2 к Windows XP при установке последнего она вообще убирается. Это объясняется появлением летом 2005 г. 64-разрядной ОС WINXP64E, в связи, с чем все ныне функционирующее программное обеспечение переводится в статус ***legacy*** (Legacy – термин, используемый для обозначения программного обеспечения, которое объявляется устаревшим, но допустимым к эксплуатации без соответствующих на то гарантий), а предшествующее ***legacy*** должно прекратить свое существование.

1.2. История развития и поколения ОС

Отдельные этапы развития операционных систем определяются как развитием элементной базы компьютеров, так и появлением новых идей в использовании последних и эволюцией информационных технологий в целом. Поэтому выделять эти этапы и говорить о поколениях ОС можно в большой степени условно, вплоть до того, что в настоящее время такая классификация скорее является данью традиции (аналогичная ситуация имеет место в классификации компьютеров по поколениям, утратившей актуальность еще раньше).

Первоначально поколения ОС определялись в соответствии с элементной базой компьютеров, на которых ОС ставилась, но очень скоро к этому классификационному признаку добавились характеристики организации вычислительного процесса, программного обеспечения, области использования, и классификация практически сразу потеряла четкость и приобрела черты исторического экскурса. Ниже эта история кратко приводится так, как она сложилась.

Первый период (1945 -1955).

Реально отсчет эры ЭВМ начинается с 40-х годов, когда были созданы ***первые ламповые вычислительные устройства*** (1946-48гг.). В то время одна и та же группа людей участвовала и в проектировании, и в эксплуатации, и в программировании вычислительной машины. Это была скорее научно-исследовательская работа в области вычислительной техники, а не использование компьютеров в качестве инструмента решения каких-либо практических задач из других прикладных областей. ***Программирование осуществлялось исключительно на машинном языке***. ***Об операционных системах не было и речи***, все задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Не было никакого другого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

Второй период (1955 – 1965).

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

В эти годы ***появились первые алгоритмические языки***, а следовательно, и первые системные программы – компиляторы. Стоимость процессорного времени возросла, что потребовало уменьшения непроизводительных затрат времени между запусками программ. Появились ***первые системы пакетной обработки***, которые просто автоматизировали запуск одной программ за другой и тем самым увеличивали коэффициент загрузки процессора. Системы пакетной обработки явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан

формализованный ***язык управления заданиями***, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило в виде колоды перфокарт, получила название пакета заданий.

Третий период (1965 – 1980).

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

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

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

Однако, несмотря на необозримые размеры и множество проблем, ***OS/360*** и другие ей подобные ***операционные системы машин третьего поколения*** действительно удовлетворяли большинству требований потребителей.

Важнейшим достижением ОС данного поколения явилась ***реализация мультипрограммирования***. Мультипрограммирование – это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ. Пока одна программа выполняет операцию ввода-вывода, процессор не простаивает, как это происходило при последовательном выполнении программ (однопрограммный режим), а выполняет другую программу (многопрограммный режим). При этом каждая программа загружается в свой участок оперативной памяти, называемый разделом.

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

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

Четвертый период (1980 – настоящее время).

Следующий период в эволюции операционных систем связан с ***появлением больших интегральных схем (БИС)***. В эти годы произошло резкое возрастание степени интеграции и удешевление микросхем. Компьютер стал доступен отдельному человеку, и наступила ***эра персональных компьютеров***. Если миникомпьютер дал возможность иметь собственную вычислительную машину отделу предприятия или университету, то персональный компьютер сделал это возможным для отдельного человека.

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

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

В середине 80-х стали бурно развиваться сети персональных компьютеров, работающие под управлением ***сетевых*** или ***распределенных*** ОС. К началу 90-х практически все ОС стали сетевыми, способными поддерживать работу с разнородными клиентами и серверами. Появились специализированные сетевые ОС, предназначенные исключительно для выполнения коммуникационных задач (например, система IOS компании Cisco Systems, работающая в маршрутизаторах).

1.3. Классификация ОС

Для построения классификации ОС прежде всего необходимо выбрать основание классификации. Таких оснований множество, но наиболее существенными можно считать следующие:

- область использования ОС;
- типы аппаратной платформы;
- методы проектирования;
- реализация внутренних алгоритмов управления ресурсами.

Классификация по области использования:

- ***настольные ОС (Desktop Operating System***) – ОС, ориентированные на работу отдельного пользователя в различных предметных областях (разработка программ, работа с документами и т.п.); основными чертами настольных ОС являются универсальность и ориентированность на пользователя; представители – MacOS, Windows;

- ***серверные ОС***, использующиеся в серверах сетей как центральное звено, а также в качестве элементов систем управления; основной чертой серверных ОС является надежность; представители – семейство UNIX, Windows NT;

- ***специализированные ОС***, ориентированные на решение узких классов задач с жестким набором требований (высокопроизводительные вычисления, управление в реальном времени); системы такого рода практически неразрывно связаны с аппаратной платформой; представители – QNX, редуцированные и специализированные версии UNIX, системы собственной разработки;

- ***мобильные ОС*** – вариант развития настольных ОС на аппаратной платформе КПК; основные черты – удобство использования и компактность; представители – PalmOS, Windows CE.

Безусловно, данная классификация не является абсолютно жесткой, т.е. одна и та же система может исполнять различные функции. Примером тому служит использование Linux с графической оболочкой в качестве настольной ОС или Windows NT в качестве серверной. Однако каждая ОС «сильна» только в своем классе.

Несложно заметить, что каждый класс ОС из приведенной классификации работает на своей ***аппаратной платформе***, так что эта классификация в той или иной мере является и классификацией по типу этой платформы. Можно, однако, попытаться провести более строгую классификацию такого рода, выделив, в частности, в отдельные классы:

- ОС для платформы x86, однопроцессорные варианты;
- ОС для платформы x86, многопроцессорные варианты;
- ОС для RISC платформ;
- ОС для мобильных устройств;
- встраиваемые ОС (ОС таких устройств, как принтеры, ЦФК и т.п.).

По внутренним алгоритмам управления ресурсами можно создать несколько бинарных классификаций:

- многозадачные /однозадачные ОС
- многопользовательские /однопользовательские ОС и т.п.

Последняя классификация будет уточняться по мере рассмотрения механизмов управления ресурсами.

1.4. Функциональные компоненты ОС

Рассмотрим понятие ресурса. Под *ресурсом* понимается любой объект, который может быть использован вычислительным процессом (распределен в процессе вычислений).

Основные ресурсы:

- аппаратные – процессоры, память, внешние устройства;
- информационные – данные и программы.

Программы ОС группируются согласно выполняемым функциям и называются *подсистемами ОС*. Все подсистемы разделяются на два больших класса по следующим признакам:

- по типам локальных ресурсов, которыми управляет ОС; соответствующие подсистемы – ***подсистемы управления ресурсами***;

- по специфические задачи, применимым ко всем ресурсам; соответствующие подсистемы – ***подсистемы, общие для всех ресурсов***.

Основные подсистемы управления ресурсами – это подсистемы:

· управления процессами;

· управления памятью;

· управления файлами и внешними устройствами.

Общие для всех ресурсов – это подсистемы:

· прикладного программного и пользовательского интерфейсов;

· защиты данных и администрирования.

Охарактеризуем их и далее перейдем к их рассмотрению в рамках отдельных тем.

Управление процессами.

Эта подсистема – важнейшая часть ОС. *Процесс* кратко можно определить как «*программу в стадии выполнения*». Реально это некоторый исполняемый код, содержащий обращения к функциям операционной системы и через их посредство получающий доступ к ресурсам. Таким образом, процесс можно также определить как некоторую *заявку на потребление системных ресурсов*.

Подсистема управления процессами планирует выполнение процессов и выполняет следующие функции:

- распределяет процессорное время между несколькими одновременно существующими в системе процессами;
- занимается созданием, переключением состояния и уничтожением процессов;
- обеспечивает процессы необходимыми системными ресурсами;
- поддерживает синхронизацию процессов;
- обеспечивает взаимодействие процессов.

Управление памятью.

Функциями подсистемы управления памятью являются:

- отслеживание свободной и занятой памяти;
- выделение памяти процессам и ее освобождение при завершении процесса;
- защита памяти процесса;
- вытеснение процессов из оперативной памяти на диск при ее нехватке и возвращение в оперативную память при освобождении места в ней (механизм ***виртуальной памяти***);
- настройка адресов программы на конкретную область физической памяти.

Управление файлами и внешними устройствами.

Управление файлами и внешними устройствами осуществляется совместной работой двух подсистем – *файловой системы* и *подсистемы ввода-вывода*.

Файловая система (ФС), экранирует сложности взаимодействия с реальной аппаратурой при работе с данными. ФС виртуализирует для пользователя набор данных на внешнем накопителе в виде *файла* – последовательности байтов, имеющей символьное имя. Файлы группируются в каталоги. Пользователь может с помощью ОС выпол-

нять над каталогами и файлами такие действия как создание, изменение, удаление, вывод содержимого, поиск по имени.

Файловая система выполняет преобразование символьных имен файлов в физические адреса данных на диске, организует совместный доступ к файлам, защищает их от несанкционированного доступа.

Подсистема ввода-вывода, или подсистема управления внешними устройствами, осуществляет передачу данных между дисками и оперативной памятью по запросам файловой системы. Эта подсистема, располагая набором *драйверов* различных устройств, обеспечивает также интерфейс между компьютером и устройствами, подключенными к нему.

Таким образом, ОС поддерживает высокоуровневый унифицированный интерфейс для написания прикладных программ. Со времени появления ОС Unix этот интерфейс в большинстве систем строится на *концепции файлового доступа*: обмен с внешним устройством выглядит как обмен с файлом. В качестве файла может выступать как реальный файл на диске, так и алфавитно-цифровой терминал, принтер или сетевой адаптер. Реальная аппаратура подменяется удобными для пользователя и программиста абстракциями.

Интерфейс прикладного программирования и пользовательский интерфейс.

Под *интерфейсом прикладного программирования* понимаются средства, предоставляемые операционной системой для обращения к ее возможностям при написании приложений. Традиционно термин «прикладное программирование» использовался как противоположный «системному программированию». Относительное различие системного и прикладного программирования заключается в том, что путем создания программ первое предполагает расширение возможностей операционной системы, а второе – решение задач из конкретной проблемной области. В данном случае это различие несущественно.

В программах обращения к ОС используются по крайней мере в следующих случаях:

- для выполнения действий с особым статусом, которым обладает только ОС (например, для управления аппаратными средствами компьютера); обычно это необходимо для более эффективного использования аппаратных ресурсов;

- для упрощения написания приложений посредством использования готовых отлаженных сервисных функций ОС, реализующих часто требующиеся универсальные действия.

Возможности ОС доступны программисту в виде набора функций, называющегося *API (Application Programming Interface, интерфейс прикладного программирования).* При этом для разработчика приложений все особенности конкретной ОС представлены особенностями ее API, поэтому операционные системы с различной внутренней организацией, но с одинаковым набором функций API представляются на этом уровне как одна ОС. Это упрощает стандартизацию ОС и обеспечивает переносимость приложений в рамках ОС одного стандарта. Например, следование общим стандартам API Unix позволяет говорить о некоторой обобщенной ОС Unix при существенных различиях внутренней организации версий этой ОС от разных производителей.

Приложения обращаются к функциям API с помощью *системных вызовов*. Способ организации системных вызовов зависит от структурной организации ОС, связанной с аппаратной платформой, и от языка программирования.

Так, в ***MS-DOS*** обращение к системным функциям осуществляется из программы на языке ассемблера путем вызова *программного прерывания.* Предварительно в регистры заносятся код системной функции и адрес размещения входных данных либо сами данные. Выходные данные возвращаются функцией в зависимости от ее назначения на устройство вывода либо через регистр. Термин API применительно к средствам вызова системных функций MS-DOS не использовался (он введен позже) и может быть применен только задним числом.

ОС Windows поддерживает объектно-ориентированный стиль программирования. Объектами являются окна. Поэтому работа с системными функциями Windows может осуществляться по тем же правилам, что и с пользовательскими функциями. Например, программу с использованием системных функций Windows можно реализовать в среде Delphi или на C++.

Функции WinAPI находятся в системных загружаемых библиотеках, таких как system32.dll, kernel32.dll, user32.dll, gdi32.dll и др. (каталог g:\WINDOWS\system32). Эти библиотеки используются самой ОС, поэтому они всегда находятся в памяти. Каждое приложение должно самостоятельно подключать библиотеки, содержащие необходимые ему функции.

Описания функций WinAPI на языке Си можно посмотреть через меню Пуск/Программы/Borland Delphi 7/Help/MS SDK Help Files/Win32 Developer's References.

В ***ОС Unix*** вызов системных функций аналогичен вызову пользовательских; необходимые описания функций организованы в системных заголовочных файлах.

- Под *пользовательским интерфейсом* понимается совокупность средств, предоставляемая человеку, работающему в интерактивном режиме за терминалом. Это может быть конечный пользователь, администратор ОС или тот же программист, но взаимодействующий с программой «извне». От такого пользователя функции API скрыты за оболочкой текстового (алфавитно-цифрового) или графического интерфейса.

Современные ОС поддерживают функции пользовательского интерфейса двух типов:

- командный язык для работы с терминалом; команды вводятся по одной с терминала либо их последовательность считывается из ***командного файла*** и выполняются ***командным интерпретатором*** (разновидности Unix в их исходном варианте);

- графический пользовательский интерфейс (GUI); выполнению нужного действия соответствует выбор с помощью мыши пункта меню или пиктограммы, либо иные операции с визуальными элементами на экране.

К графическому интерфейсу можно отнести и нестандартные устройства ввода, образующих «физический» интерфейс (рули, панели различных устройств типа сканера и т.п.) в ОС Windows и разновидностях Unix, располагающих GUI.

· Защита данных и администрирование

Безопасность данных обеспечивается:

- средствами отказоустойчивости ОС (защита от сбоев и отказов аппаратуры и ошибок программного обеспечения);

- средствами защиты от несанкционированного доступа (защита от ошибочного или злонамеренного поведения пользователей системы.

Функции защиты тесно связаны с функциями администрирования, так как именно администратор определяет права и возможности пользователей, отслеживает события, от которых зависит безопасность системы, и поддерживает отказоустойчивость (например, посредством утилит регулярно выполняя операции резервного копирования).

Рассмотрим функциональные компоненты сетевой ОС

Сетевые и распределенные ОС.

Компьютерная сеть – это набор компьютеров, связанных коммуникационной системой и снабженных соответствующим программным обеспечением, позволяющим пользователям сети получать доступ к ресурсам этого набора.

При организации сетевой работы операционная система играет роль интерфейса, экранирующего от пользователя все детали низкоуровневых программно-аппаратных средств сети. В зависимости от того, какой виртуальный образ реальной аппаратуры компьютерной сети создает ОС, различают *сетевые* и *распределенные* ОС.

Сетевая ОС предоставляет пользователю некую виртуальную систему, не полностью скрывающую распределенную природу реального прототипа. Пользователь сетевой ОС всегда знает, что он имеет дело с сетевыми ресурсами и что для доступа к ним нужно выполнить некоторые операции; должен знать, где хранятся его файлы, и использовать явные команды для их перемещения, а также знать, на какой машине выполняется его задание.

В идеальном случае ОС должна предоставлять пользователю сетевые ресурсы так, как если бы они были ресурсами единой централизованной виртуальной машины (ресурсы должны быть максимально *прозрачными*). Это – магистральное направление развития ОС. Такая операционная система носит название *распределенная ОС*.

Распределенная ОС существует как единая операционная система в рамках вычислительной системы и заставляет набор сетевых машин работать как виртуальный унипроцессор. Каждый компьютер сети выполняет часть функций этой единой ОС. Пользователь в таком случае, вообще говоря, не имеет сведений о том, на какой машине выполняется его работа.

В настоящее время практически все сетевые ОС далеки от идеала истинной распределенности.

Уточним термин «сетевая ОС». На разных компьютерах сети могут работать разные ОС, функционирующие независимо в том смысле, что каждая из них принимает независимые решения о создании и завершении *своих собственных* процессов и управлении *локальными* ресурсами. Но в любом случае эти операционные системы должны включать средства для работы в сети:

- взаимно согласованный набор коммуникационных протоколов для
- организации взаимодействия процессов, выполняющихся на разных компьютерах,
- разделения ресурсов этих компьютеров между пользователями сети;
- подсистемы, организующие работу по этим протоколам.

В итоге ОС получает возможность предоставления своих ресурсов в общее пользование и/или потребления ресурсов других компьютеров.

Под сетевой ОС будем понимать операционную систему отдельного компьютера, включающую средства для работы в сети.

ОС Windows, начиная с NT, различные варианты ОС Unix (HP-UX компании Hewlett-Packard, Solaris компании Sun, FreeBSD и др.), различные варианты ОС Linux, ОС MacOS, ОС NetWare компании Novell являются сетевыми.

Основные функциональные компоненты сетевой ОС показаны на рисунке 1.2.

Средства управления локальными ресурсами реализуют все функции ОС автономного компьютера, описанные выше.

Сетевые средства подразделяются на три компонента:

- ***серверная часть ОС*** – средства предоставления локальных ресурсов и услуг в общее пользование;

- ***клиентская часть ОС*** – средства запроса доступа к удаленным (т.е. принадлежащим другим компьютерам сети) ресурсам и услугам;

- ***транспортные, или коммуникационные средства ОС*** – средства, совместно с коммуникационной системой обеспечивающие обмен сообщениями в сети.

Рисунок 1.2 – Основные функциональные компоненты сетевой ОС

Правила взаимодействия компьютеров при передаче сообщений по сети фиксируются в коммуникационных протоколах (Ethernet, Token Ring, IP, IPX и пр.).

Упрощенная схема работы сетевых ОС иллюстрируется на рисунке 1.3. на примере взаимодействия двух компьютеров. Суть взаимодействия: пусть приложение, работающее на первом компьютере, использует файлы, размещенные на диске второго компьютера.

Для компьютера 1 дисковое пространство диска 2 является запрашиваемым удаленным ресурсом, следовательно, запрос на этот ресурс формируется *клиентской*

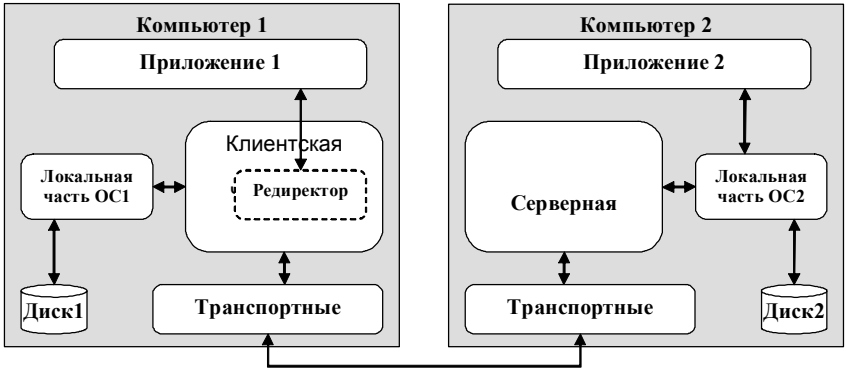


Рисунок 1.3 – Упрощенная схема работы сетевых ОС

частью ОС1. ОС2 предоставляет ресурс, следовательно, запрос будет обрабатываться *серверной частью* ОС2.

На рисунке в клиентской части ОС1 выделен компонент, названный *редиректором* (от *redirect* – перенаправлять). Это – программный модуль, предназначенный для распознавания запросов к удаленным и локальным файлам и перенаправления первых к удаленной машине. В таком случае приложения на клиентской машине не должны заботиться о том, с какими файлами они работают – удаленными или локальными.

Если функции перенаправления присутствуют в клиентской части сетевой ОС, но не оформлены в виде отдельного модуля, то редиректором часто называют всю клиентскую часть.

1.5. Требования к современным ОС

Суть требований к функциональности ОС состоит в управлении ресурсами и обеспечении интерфейса пользователя и прикладных программ.

Помимо этого, к операционным системам предъявляется целый ряд важных эксплуатационных требований.

· *Расширяемость* – возможность внесения изменений без нарушения целостности системы. Расширяемость достигается за счет модульной структуры ОС: программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс.

· *Переносимость*. В идеале код ОС должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа на аппаратную платформу другого типа. Поскольку переносимые ОС имеют несколько вариантов реализации для разных платформ, это свойство называют также *многоплатформенностью*.

· *Совместимость*. Если ОС имеет средства для выполнения прикладных программ, написанных для других операционных систем, то она обладает совместимостью с эти-

ми ОС. Различают: совместимость на уровне двоичных кодов (исполняемых программ); на уровне исходных текстов; поддержку пользовательских интерфейсов других ОС.

· *Надежность и отказоустойчивость*. Система должна быть защищена от внутренних и внешних ошибок, сбоев и отказов. Ее действия должны быть предсказуемы, а приложения не должны иметь возможности наносить вред ОС.

Эти свойства обеспечиваются архитектурными решениями, положенными в основу ОС, качеством их реализации (отлаженностью кода) и программной поддержкой аппаратных средств обеспечения отказоустойчивости (например, источников бесперебойного питания).

· *Безопасность*. Заключается в защите данных и других ресурсов от несанкционированного доступа. Обеспечивается средствами *аутентификации* (определения легальности пользователя), *авторизации* (предоставления дифференцированных прав доступа к ресурсам), *аудита* (фиксации «подозрительных» с точки зрения безопасности событий).

· ***Производительность*** – настолько хорошее быстродействие и время реакции, насколько это позволяет аппаратная платформа. Определяется архитектурой ОС, многообразием функций, качеством кода, возможностью использования высокопроизводительной аппаратной платформы.

2. УПРАВЛЕНИЕ ПРОЦЕССАМИ

2.1. Мультипрограммирование и распределение ресурсов

Главные сложности при решении задач управления ресурсами возникают в *мультипрограммных ОС*, где за ресурсы конкурирует сразу несколько приложений.

Мультипрограммирование (*многозадачность, multitasking*) – способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняется сразу несколько программ. Эти программы одновременно используют и другие ресурсы компьютера. Мультипрограммирование призвано повысить эффективность использования вычислительной системы согласно выбранным критериям. Однако требования к современным вычислительным системам столь многозначны, что любая совокупность критериев изменяется (размывается, переосмысливается либо вообще уходит в историю), следуя за изменениями в сфере информационных технологий. Поэтому приведем сложившийся набор критериев с последующим их анализом переходом к реально наличествующей ситуации.

В зависимости от реализуемого критерия ОС подразделяются на типы, каждый из которых имеет свою специфику реализации и область применения.

Ниже приведены наиболее характерные критерии эффективности вычислительных систем и соответствующие типы ОС.

Критерий эффективности	Смысл критерия	Тип ОС
Пропускная способность	Количество задач, выполняемых системой в единицу времени	Системы пакетной обработки
Удобство работы пользователей	Возможность интерактивной работы одновременно с несколькими приложениями на одной машине	Системы разделения времени
Реактивность системы	Способность выдерживать заранее заданные интервалы времени между запуском программ и получением результата (реакция за время не более заданного)	Системы реального времени

Некоторые ОС могут поддерживать одновременно несколько режимов: часть задач может выполняться в режиме пакетной обработки, часть – в режиме реального времени или в режиме разделения времени.

Пакетные ОС.

Цель: повышение пропускной способности вычислительной системы

Концепция мультипрограммирования: минимизация простоев процессора за счет его переключения на выполнение другой задачи.

Тип решаемых задач: в основном вычислительного характера, не требующие быстрого получения результатов.

Схема функционирования

Формируется пакет заданий, каждое из которых содержит требования к системным ресурсам. Из пакета формируется *мультипрограммная смесь*, т.е. множество задач для одновременного выполнения. Для этого выбираются задачи, предъявляющие разные требования к ресурсам, так, чтобы обеспечивалась сбалансированная нагрузка всех устройств вычислительной машины (например, в смеси должны присутствовать задачи вычислительного характера задачи с интенсивным вводом-выводом). Выбор очередного задания для выполнения зависит от складывающейся в системе ситуации – это должно быть «выгодное» с точки зрения загрузки задание.

Следствие: невозможно гарантировать выполнение конкретного задания в течение заданного периода времени.

Рассмотрим схему работы мультипрограммной пакетной ОС на примере совмещения во времени операций ввода-вывода и вычислений.

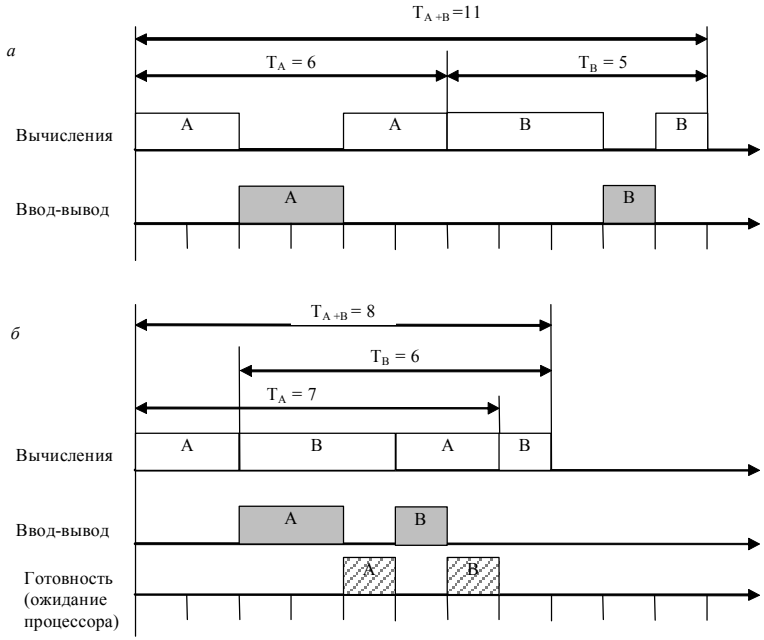


Рисунок 2.1 – Схема выполнения двух задач: в однопрограммной системе (а) и в мультипрограммной системе (б)

Из рисунка 2.1. видно, что в однопрограммном режиме каждая из задач выполняется быстрее, чем в мультипрограммном, зато в последнем меньше общее время выполнения задач.

В целом общее время выполнения смеси задач в рассматриваемой системе меньше, чем суммарное время их последовательного выполнения. Однако выполнение от-

дельной задачи может занять больше времени, чем при монопольном выделении ей процессора. Это происходит за счет того, что задача может быть готова к выполнению (как задача А по истечении 4-х единиц времени на рис. 2.1. б), но процессор занят выполнением другой задачи.

Переключение процессора с выполнения одной задачи на выполнение другой происходит *по инициативе активной задачи* (например, при отказе от процессора из-за необходимости выполнения ввода-вывода). Поэтому существует высокая вероятность того, что одна задача может надолго занять процессор. Во многом в силу этого пакетные ОС не предназначены для работы в интерактивном режиме. Суть взаимодействия пользователя с машиной сводится к сдаче задания оператору и получении результата после выполнения всего пакета.

Такой режим повышает эффективность функционирования аппаратуры, но снижает эффективность работы пользователя.

Системы разделения времени.

Цель: предоставление пользователю (пользователям) возможности интерактивной работы одновременно с несколькими приложениями.

Концепция мультипрограммирования: разделение времени, или циклическое поочередное выделение кванта процессорного времени каждому приложению.

Тип решаемых задач: задачи, ориентированные на решение в интерактивном режиме (периодически требующие участия пользователя).

Общая схема функционирования представлена на рис. 2.2.

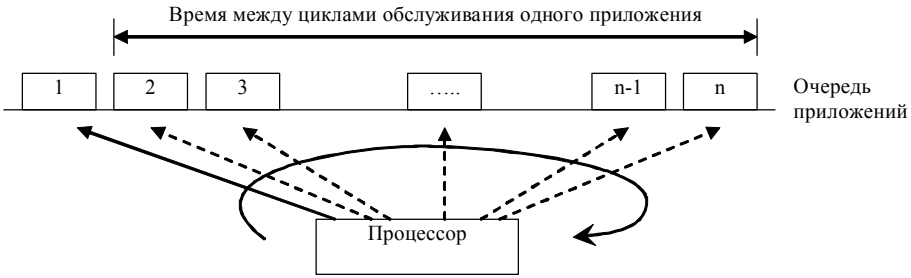


Рисунок 2.2 – Схема функционирования ОС в режиме разделения времени

Переключение процессора на выполнение очередной задачи осуществляется принудительно операционной системой, но к выполнению принимается каждая запущенная пользователем задача в порядке запуска.

В случае нескольких пользователей каждому из них предоставляется свой терминал. Так как ни одна задача не занимает процессор надолго, время ожидания ответа системы в интерактивном режиме оказывается приемлемым, а если кванты времени малы, то у всех пользователей складывается впечатление монопольной работы на машине.

Пропускная способность системы меньше, чем при пакетной обработке, за счет «навязанной извне» очередности задач и затрат на переключение процессора. Тем не менее аппаратура загружается лучше, чем в однопрограммном режиме, так как процессор не простаивает во время ожидания сообщения пользователя, а может обрабатывать другие приложения.

Системы реального времени.

Цель: управление техническими объектами (станком, спутником, ядерным реактором и т.п.) или процессами (доменным процессом и т.п.).

Концепция мультипрограммирования: выбор программы для выполнения по прерыванию (исходя из текущего состояния объекта) или согласно расписанию плановых работ.

Тип решаемых задач: задачи управления в реальном режиме времени.

Общая схема функционирования

Мультипрограммная смесь представляет собой фиксированный набор заранее составленных программ. Выбор программы для выполнения осуществляется согласно складывающейся ситуации управления (см. выше).

Главный критерий функционирования – быстрый ответ системы – часто определяет требования к классу процессора, обязанного опрашивать множество источников прерывания и быстро переключаться с задачи на задачу.

В таких системах не только не стремятся максимально загружать все устройства, но, напротив, предусматривают некоторый запас резервной мощности на случай пиковой нагрузки (например, аварии на атомной станции, когда ряд датчиков сработает одновременно).

В чистом виде ни одна из приведенных классификаций не описывает современных ОС, но в то же время каждая из концепций мультипрограммирования в той или иной степени присутствует.

Исходя из того, что требует и что получает пользователь от современной ОС, можно сделать следующие утверждения.

Прежде всего, качественно изменились критерии эффективности использования вычислительной системы.

Для пользователей настольных ОС этим критерием в первую очередь является необходимость решения имеющегося класса задач максимально удобным для пользователя образом. Вопросы производительности в смысле оптимального использования аппаратных ресурсов рассматриваются в последнюю очередь. Специфика решаемых задач во многом определяет дисциплину их выполнения. Большинство задач является не заданиями на вычисления с окончательным результатом, а интерактивными процессами обработки данных (текстовый редактор; мультимедийные средства; графические редакторы; игры; работа в сети и т.д.). В общем случае загрузка процессора невелика (порядка 1-2%) не из-за плохого планирования, а из-за отсутствия задач, требующих процессорного времени. Совмещение задач преследует цель не повышения производительности, а одновременного выполнения нескольких действий (за исключением ограниченных классов задач).

Для серверных ОС критерием эффективности является выполнение известного числа задач (которое зависит от посещаемости сервера) с минимальной нагрузкой на ресурсы

– дисковую подсистему и процессор. Поскольку сервер не столько обрабатывает данные, сколько передает их, ввод-вывод играет важнейшую роль и именно он является объектом оптимизации, но сама специфика задач способствует этому. Спектр задач невелик, причем они повторяются (предоставление услуг Web-сервера, почтовой службы, файл-сервера и т.п.). Спланировать порядок выполнения заявок нельзя, но зато можно оптимизировать выполнение одинаковых задач, сохраняя их промежуточные результаты (хранить в памяти, а не читать каждый раз с диска Web-страницы и т.п.).

Можно сказать, что от пакетных ОС частично унаследовано совмещение ввода-вывода и процессорной обработки и выбор задач на выполнение самой системой согласно определяемым ею же приоритетам. Режим в чистом виде принадлежит истории.

Режим разделения времени присутствует безусловно во всех настольных и серверных ОС в усовершенствованном виде – выделение квантов времени задаче с учетом ее приоритета.

Суть режима реального времени остается той же. Кроме того, это понятие можно применить и к настольным и серверным ОС. Время реакции системы, в зависимости от приложения, может быть весьма критичным, например, при просмотре фильмов, прослушивании музыки, ожидании ответа сервера. Безусловно, это время зависит и от аппаратуры.

Схема выполнения задач, изображенная на рис. 2.1, является абстракцией, иллюстрирующей принцип разделения операций между устройствами. Здесь прежде всего существенно, что устройства ввода-вывода настолько медленны, что на время их работы можно занять процессор, а именно на загрузку процессора ориентирована система. Кроме того, не отображено процессорное время, затрачиваемое на организацию ввода вывода, ввиду его несопоставимости с длительностью ввода-вывода.

Для современных ОС эта схема может выглядеть примерно следующим образом (для наглядности процессы A и B конкретизированы).

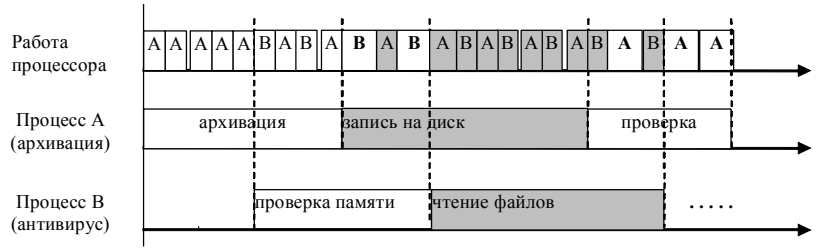


Рисунок 2.3 – Примерная схема мультипрограммной обработки двух задач с квантованием времени

Здесь для процессов A и B на временной оси отмечены моменты начала и конца выполнения различных операций, включая время ожидания задачей очередного кванта. Более широкие прямоугольники соответствуют большим квантам времени, а величина кванта определяется динамически изменяемым приоритетом задачи. Светлые прямоугольники соответствуют операциям, занимающим преимущественно процессор, а темные – операциям, занимающим преимущественно дисковую подсистему. Рисунок в представленном виде соответствует системе со старым, очень медленным накопите-

лем, не имеющим режима DMA (direct memory access, совокупность аппаратных возможностей, обеспечивающих работу с диском без участия процессора). В таком случае загрузка процессора может доходить до 70%, например, при работе архиватора.

При использовании режима DMA ввод-вывод практически не загружает процессор (его загрузка составляет 3 – 5%), и затемненные прямоугольники-кванты должны быть по меньшей мере в десятки раз уже.

2.2. Понятие процессов и потоков

Управление процессами состоит в их создании и уничтожении; приостановлении и возобновлении; изменении приоритета; переключении состояний.

При управлении процессами ОС использует два основных типа информационных структур: *дескриптор процесса* и *контекст процесса*.

Дескриптор процесса *(это термин, используемый в Unix, но ставший общеупотребительным. Аналогичная структура в Windows называется объект-процессом)* содержит информацию о процессе, которая необходима ядру ОС в течение всего жизненного цикла процесса независимо от его состояния.

Дескриптор содержит:

- идентификатор процесса;
- информацию о состоянии процесса;
- данные о степени привилегированности процесса;
- местоположение кодового сегмента;
- данные о родственных процессах;
- данные о событиях, которые ожидает процесс и др.

Дескрипторы отдельных процессов объединены в таблицу (очередь) процессов, на основе которой ОС осуществляет планирование и синхронизацию процессов. Память для таблицы отводится динамически в области ядра ОС.

Контекст процесса содержит информацию, необходимую для возобновления выполнения процесса после прерывания и поэтому сохраняемую перед прерыванием. Это:

- состояние аппаратуры компьютера:
- значение счетчика команд; содержимое регистров общего назначения;
- режим работы процессора;
- флаги;
- маски прерываний и др.;
- параметры операционной системы:
- указатели на открытые файлы;
- информация о незавершенных операциях ввода-вывода;
- коды ошибок выполняемых процессом системных вызовов и др.

Контекст, как и дескриптор, доступен только программам ядра, но хранится не в области ядра, а непосредственно примыкает к *образу* (совокупности кода и данных) процесса и может быть перемещен вместе с ним (из ОП на диск и наоборот).

Программный код начинает выполняться, когда для него создан процесс.

Создать процесс – это означает:

- создать информационные структуры, описывающие процесс, т.е. его дескриптор и контекст;
- включить дескриптор нового процесса в очередь готовых процессов;
- загрузить кодовый сегмент процесса в Оперативную память или область свопинга.

В многопоточных системах (с возможностью распараллеливания процессов) для каждого процесса должен создаться как минимум один поток выполнения. При этом ОС генерирует *описатель (дескриптор) потока.*

Момент выборки процесса на выполнение осуществляется в соответствии с принятой в ОС дисциплиной обслуживания.

Процесс может порождать процессы – потомки, в результате чего организуется иерархическая структура процессов. Отношения между потомками и родителями строятся по-разному в различных ОС.

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

· *выполнение* – активное состояние, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

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

· *готовность* – также пассивное состояние; но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В течение своей жизни каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, принятым в данной операционной системе.

Типичный граф состояний потока приведен на рисунке 2.4.

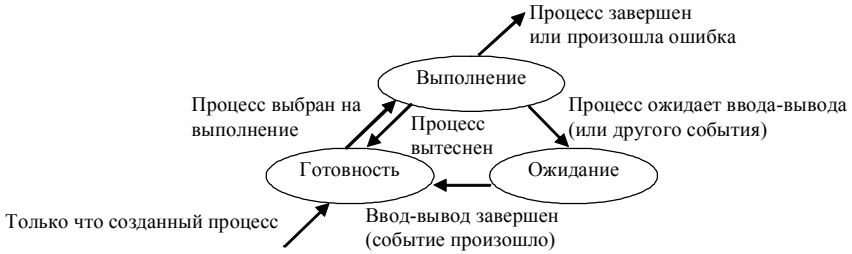


Рисунок 2.4 – Граф состояний процессов в многозадачной системе

Жизненный цикл процесса начинается с состояния *готовность*, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние *выполнение* и находится в нем до тех пор, пока либо он сам не освободит

процессор, перейдя в состояние *ожидания* какого-нибудь события, либо будет насильно вытеснен из процессора (например, вследствие истечения отведенного данному процессу кванта процессорного времени). В последнем случае процесс возвращается в состояние *готовность*. В это же состояние процесс переходит из состояния *ожидания* после того, как ожидаемое событие произойдет.

В состоянии *выполнение* в однопроцессорной системе может находиться только один процесс, а в каждом из состояний *ожидание* и *готовность* – несколько процессов. Эти процессы образуют очереди ожидающих и готовых процессов соответственно.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в *списки*. Таким образом, каждый элемент списка содержит по крайней мере один указатель на другой элемент, соседствующий с ним в очереди. Такая организация очередей позволяет легко переупорядочивать, включать и исключать процессы, переводить их из одного состояния в другое (удалять из одной и ставить в другую очередь). Рисунок 2.5 иллюстрирует размещение процессов в очереди.

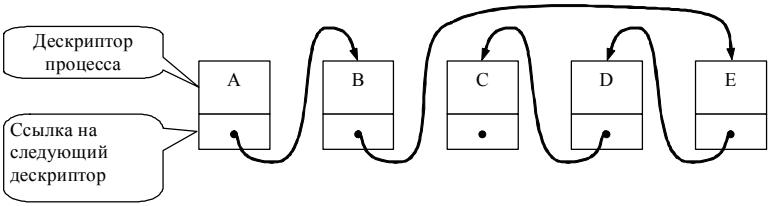


Рисунок 2.5. – Размещение процессов в очереди

Предположим, что на рисунке изображена очередь готовых к выполнению процессов. Тогда запланированный порядок выполнения выглядит так: A, B, E, D, C.

2.3. Алгоритмы планирования процессов и потоков

Планирование процессов включает в себя решение следующих задач:

- определение момента времени для смены выполняемого процесса;
- выбор процесса на выполнение из очереди готовых процессов.

Различные алгоритмы планирования могут преследовать различные цели и обеспечивать разное качество мультипрограммирования. Например, алгоритм должен гарантировать, что ни один процесс не будет занимать процессор дольше определенного времени; другой обеспечивает максимально быстрое выполнение «коротких» задач; третий обеспечивает преимущественное право на процессорное время интерактивным приложениям. Именно особенности планирования процессов в наибольшей степени определяют специфику ОС.

В большинстве ОС универсального назначения планирование осуществляется *динамически* (on-line), то есть решения принимаются во время работы системы на основе анализа текущей ситуации. ОС не имеет никакой предварительной информации о задачах, которые появляются в случайные моменты времени.

Статический тип планирования используется в специализированных системах, где набор одновременно выполняемых задач определен заранее (например, в системах реального времени). Здесь решение о планировании принимается заранее (off-line).

Диспетчеризация заключается в реализации найденного в результате планирования решения, т.е. в переключении процессора с одного потока на другой, и сводится к следующему:

- сохранение контекста текущего процесса;
- загрузка контекста нового процесса;
- запуск нового процесса.

В отличие от планирования, осуществляемого программными средствами ОС, диспетчеризация реализуется совместно с аппаратными средствами процессора.

Примечание. В различных ОС компоненты, занимающиеся планированием, могут называться по-разному: scheduler – распорядитель, или планировщик, – в Unix; dispatcher – в Windows.

Вытесняющие и невытесняющие алгоритмы планирования

С самых общих позиций – по принципу освобождения процессора активным процессом – существует два основных типа процедур планирования процессов: *вытесняющие* и *невытесняющие*.

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

При невытесняющем программировании механизм планирования распределен между ОС и прикладными программами, что создает проблемы как для пользователей, так и для разработчиков приложений, хотя и может быть преимуществом при решении некоторого фиксированного набора задач (примером эффективного использования невытесняющего планирования являются файл-серверы NetWare 3.x и 4.x, в которых благодаря такому планированию достигнута высокая скорость выполнения файловых операций).

Вытесняющая многозадачность (preemptive multitasking) – способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого принимается операционной системой, а не самой активной задачей.

При вытесняющем мультипрограммировании функции планирования процессов целиком сосредоточены в операционной системе.

Почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (Unix, Windows NT/2000, OS/2, VAX/VMS), реализованы вытесняющие алгоритмы планирования процессов, в которых механизм планирования задач целиком сосредоточен в операционной системе. Программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. Операционная система определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст.

Алгоритмы, основанные на квантовании (классификация по принципу смены активного процесса во времени)

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, *если исчерпан квант процессорного времени*, отведенный данному процессу (или процесс перешел в состояние ожидания, или произошла ошибка, или процесс завершился и покинул систему).

Процесс, который исчерпал свой квант, переводится в состояние готовности и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Это – концепция разделения времени. Ниже изображен граф состояний процесса, соответствующий описанному алгоритму. Видно, что это частный случай графа на рисунке 2.4.

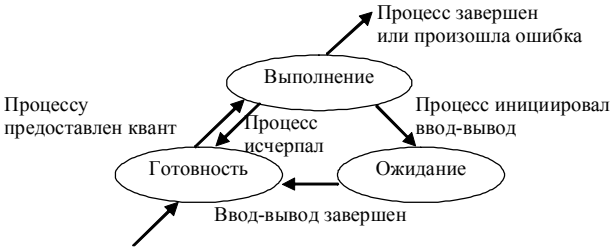


Рисунок 2.6 – Граф состояний процесс в системе с квантованием

Выделяемые кванты могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса.

Случай, когда всем процессам предоставляются кванты одинаково длины, представлен на рисунке 2.7.

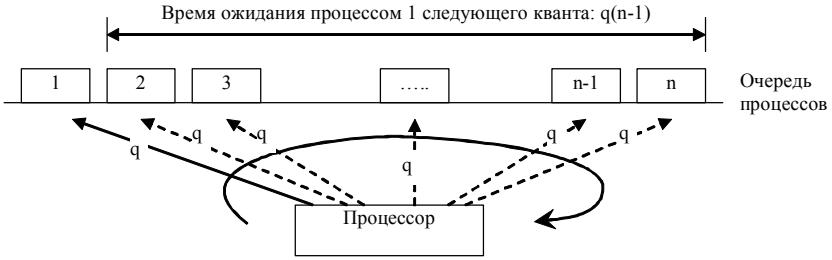


Рисунок 2.7 – Режим разделения времени.

Приведем некоторые весьма грубые оценки.

Пусть в системе имеется n процессов, каждый из которых требует для своего выполнения примерно B единиц времени при монопольном использовании системы. Пусть q – длина кванта.

Тогда процесс проводит в ожидании следующего кванта время, равное q(n-1). Чем больше процессов в системе, тем больше эта величина и меньше возможность вести интерактивную работу с пользователями. Однако если величина кванта очень невелика, то время ожидания все равно будет приемлемым для пользователей. Типичное значение кванта в системах разделения времени составляет десятки миллисекунд.

Если квант короткий (т.е. в общем случае B >> q и процесс многократно участвует в цикле обработки), то суммарное время, которое процесс проводит в ожидании процессора, прямо пропорционально B.

Действительно, необходимое для процесса количество циклов выполнения равно B/q, и тогда общее время ожидания равно (q(n-1))*(B/q), или в итоге – B(n-1).

При достаточно большом кванте алгоритм квантования вырождается в алгоритм последовательной обработки, при котором время ожидания задачи в очереди не зависит от её длительности.

· Варианты алгоритмов квантования с изменяющимся квантом.

- Пусть первоначально каждому процессу назначается достаточно большой квант, а величина каждого следующего кванта уменьшается до некоторой заранее заданной величины.

В таком случае преимущество получают короткие задачи, а длительные вычисления будут проводиться в фоновом режиме.

- Пусть каждый следующий квант, выделяемый очередному процессу, больше предыдущего. Такой подход позволяет уменьшить накладные расходы на переключение задач в том случае, когда сразу несколько задач выполняют длительные вычисления.

- Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить компенсацию в виде привилегий при последующем обслуживании. Для этого планировщик создает две очереди готовых процессов (рисунок 2.8).

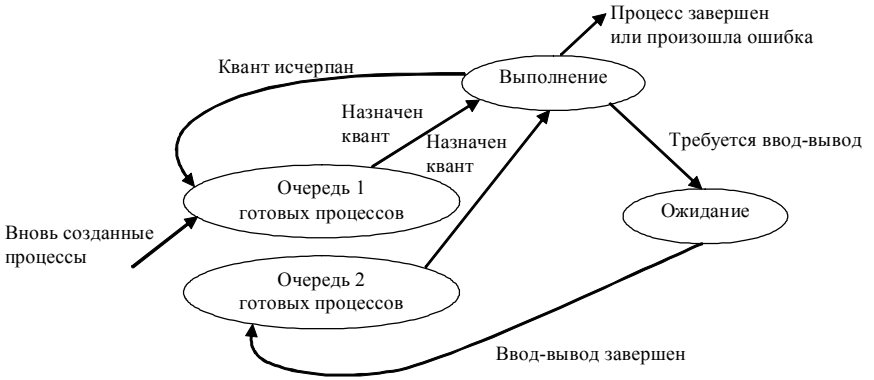


Рисунок 2.8 – Квантование с предпочтением процессов, интенсивно обращающихся к вводу-выводу

Очередь 1 образована процессами, которые пришли в состояние готовности в результате исчерпания кванта времени, а очередь 2 – потоками, у которых завершилась операция ввода-вывода. При выборе процесса для выполнения прежде всего просматривается вторая очередь, и только если она пуста, квант выделяется потоку из первой очереди.

· Очереди готовых процессов также могут быть организованы по-разному: по правилу «первый пришел – первый обслужился» (FIFO) или по правилу «последний пришел – первый обслужился» (LIFO).

· В любом случае в алгоритмах, основанных на квантовании, не используется никакой предварительной информации о задачах. Дифференциация обслуживания базируется только на истории существования процесса в системе.

Алгоритмы, основанные на приоритетах (классификация по принципу выбора процесса на выполнение из очереди)

· *Приоритет* – это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени. Чем выше приоритет процесса, тем значительнее его привилегии и тем меньше времени он будет проводить в очередях.

Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что большее число обозначает больший приоритет, в других – наоборот (большее число означает меньший приоритет).

Приоритет может назначаться директивно администратором системы, например, в зависимости от важности работы, либо вычисляться самой ОС по определенным правилам.

В зависимости от возможности изменения приоритета в течение жизни потока различаются *динамические* и *фиксированные* приоритеты. В системах с динамическими приоритетами изменения приоритета могут происходить по инициативе процесса, обращающегося с вызовом к операционной системе; по инициативе пользователя, выполняющего соответствующую команду; по инициативе ОС в зависимости от ситуации, складывающейся в системе.

· Существует две разновидности алгоритмов приоритетного планирования: *обслуживание с относительными* приоритетами и обслуживание с *абсолютными* приоритетами.

В обоих случаях выбор процесса на выполнение из очереди осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По-разному решается проблема определения момента смены активного процесса.

В системах с *относительными приоритетами* активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ожидания (или же произойдет ошибка, или процесс завершится) (рисунок 2.9, а).

В системах с *абсолютными приоритетами* выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности (рисунок 2.9, б).

В многопоточных ОС приоритет потока непосредственно связан с приоритетом процесса, в рамках которого выполняется данный поток. Приоритет процесса назначается операционной системой при создании процесса. ОС учитывает статус процесса (системный или прикладной), статус пользователя, запустившего

процесс; наличие явного указания пользователя на присвоение процессу определенного уровня приоритета.

Значение приоритета включается в дескриптор процесса и используется при назначении приоритета потокам этого процесса. Поток может быть инициирован по команде пользователя или в результате выполнения системного вызова другим потоком. В последнем случае ОС принимает во внимание значение параметров системного вызова.

Смешанные алгоритмы планирования (квантование с приоритетами)

Во многих операционных системах алгоритмы планирования построены с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок смены процессов и выбора процесса из очереди готовых определяется приоритетами процессов.

Рисунок 2.9 – Графы состояний процессов в системах с относительными и абсолютными приоритетами

На смешанных алгоритмах основано планирование в системах Windows NT и Unix System V Release 4. И в одной, и в другой системе реализована дисциплина вытесняющей многозадачности, основанная на использовании абсолютных приоритетов и квантования. Эти системы используем как основу для рассмотрения смешанных алгоритмов. Обобщенный граф состояния процессов (потоков) в этом случае имеет вид, представленный на рис. 2.10.

Общими для упомянутых ОС являются следующие моменты.

· Диапазон приоритетов подразделяется на несколько классов.

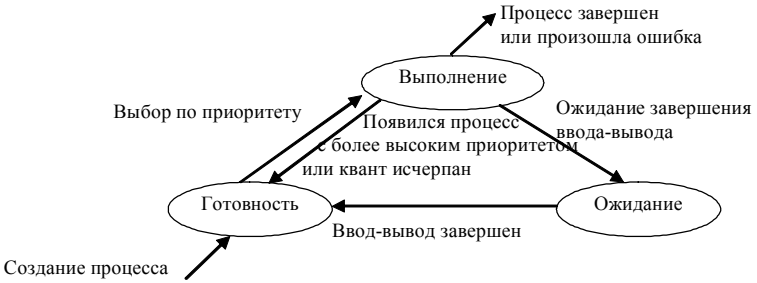


Рисунок 2.10 – Граф состояний процессов (потоков) в системах с планированием на основе абсолютных приоритетов и квантования

· Часть диапазона отведена для процессов (потоков) с переменными приоритетами, часть – для процессов (потоков) с фиксированными приоритетами. К последним относятся процессы реального времени, наиболее критичные ко времени и имеющие самые высокие приоритеты.

· При планировании процессов (потоков) с динамическими приоритетами приоритет процесса (потока), полностью исчерпавшего свой очередной квант времени, снижается; напротив, если квант полностью не исчерпан, приоритет повышается.

2.4. Синхронизация процессов

Потребность в синхронизации процессов возникает только в мультипрограммных ОС и связана с совместным использованием аппаратных и информационных ресурсов вычислительной системы. Выполнение процессов в таких ОС в общем случае имеет асинхронный характер, т.е. процессы выполняются независимо в том плане, что практически невозможно с полной определенностью сказать, на каком этапе выполнения будет находиться определенный процесс в определенный момент времени.

Суть синхронизации процессов состоит в согласовании их скоростей путем приостановки процесса до наступления некоторого события и последующей его активизации при наступлении этого события.

Синхронизация лежит в основе любого *взаимодействия процессов*, которое может быть связано:

- с обменом данными (процесс-получатель должен обращаться за данными только после их записи процессом-отправителем);
- с разделением ресурсов (например, если активному процессу требуется доступ к последовательному порту, занятому другим процессом, то активный процесс должен быть приостановлен до освобождения ресурса);
- с синхронизацией процесса с внешними событиями (например, с нажатием комбинации клавиш).

Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций при взаимодействии процессов. Пренебрежение вопросами синхронизации может привести к неправильной работе процессов или даже к краху системы. Примерами таких ситуаций являются *гонки* и *тупики*.

Гонками называются ситуации, когда в отсутствие синхронизации два (или более) процесса обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей процессов.

Тупики – это взаимные блокировки процессов, могущие возникать вследствие недостаточно корректного решения задачи синхронизации и состоящие в том, что ряд процессов удерживает ресурсы, запрашиваемые другими процессами, и в то же время запрашивает ресурсы, удерживаемые другими.

Гонки рассмотрим на примере приложения, ведущего базу данных о клиентах некоторого предприятия (рис. 2.12). Сведения о каждом клиенте представляют собой запись, содержащую поле описания заказа клиента и поле оплаты заказа. Таким образом, запись изменяется в двух случаях: когда клиент делает заказ и когда он его оплачивает.

Пусть приложение оформлено как единый процесс, имеющий два потока, A и B. Пусть поток A заносит в базу данные о заказах, а поток B – данные об оплате. Оба потока совместно работают над общим файлом базы данных по следующему общему алгоритму.

1. Считать из файла базы данных в буфер запись о клиенте.
2. Изменить запись (поток A заносит данные о заказе, поток B – об оплате).
3. Вернуть измененную запись в файл.

Предположим, что клиент, которому в базе уже соответствует запись, сделал заказ и сразу оплатил его, т.е. данные о заказе и об оплате должны поступить в базу практически одновременно. Далее возможен следующий вариант развития событий.

Пусть в некоторый момент поток A обновляет данные о заказе в записи о клиенте, выполняет шаги A1 и A2, но выполнить шаг A3 (занести содержимое буфера в запись базы) не успевает вследствие завершения кванта времени.

Потоку B требуется внести сведения об оплате заказа этого же клиента. Предположим, что, когда подходит очередь потока B, он успевает сделать шаги B1 и B2, а затем прерывается. При этом в его буфере оказывается запись, где данные о заказе относятся к старому заказу, а данные об оплате – к новому.

Далее поток A получает управление, выполняет шаг A3 – запись в базу содержимого своего буфера – и завершается. Вслед за ним то же проделывает поток B.

Смена содержимого буферов и базы приведена на рис. 2.12. В итоге данные о новом заказе оказались потеряны.

Приведенный вариант не является неизбежным. В силу нерегулярности возникающих ситуаций при взаимодействии процессов (в данном случае потоков) возможно и другие варианты развития событий (см. рис. 2.13). Все определяется взаимными скоростями потоков и моментами их прерывания.

Критическая секция

Важным понятием синхронизации процессов является понятие *критической секции* – части программы, в которой осуществляется доступ к разделяемым данным.

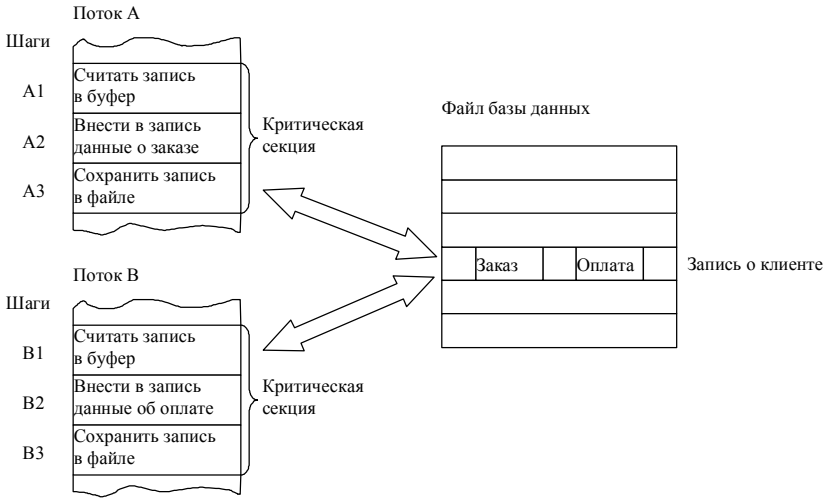


Рисунок 2.11 – Возникновение гонок при доступе к разделяемым данным

Шаги	Буфер А	Буфер В	Запись о клиенте
			Заказ old, оплата old
A1	Заказ old, оплата old		
A2	Заказ new, оплата old		Заказ old, оплата old
B1		Заказ old, оплата old	
B2		Заказ old, оплата new	Заказ old, оплата old
A3	Заказ new, оплата old		Заказ new, оплата old
B3		Заказ old, оплата new	Заказ old, оплата new

Рисунок 2.12 – Пошаговое выполнение и результаты потоков в ситуации гонок

Эти данные, соответственно, называются *критическими данными*; при несогласованном их изменении могут возникнуть нежелательные эффекты. В приведенном выше примере гонок такими данными являются записи файла базы данных. Критические секции каждого из потоков отмечены на рис. 2.11. В общем случае в разных потоках критическая секция состоит из разных последовательностей команд.

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

Средства синхронизации процессов и потоков

Для синхронизации процессов, порождаемых прикладными программами, программист может использовать как собственные средства и приемы, так и средства опера-

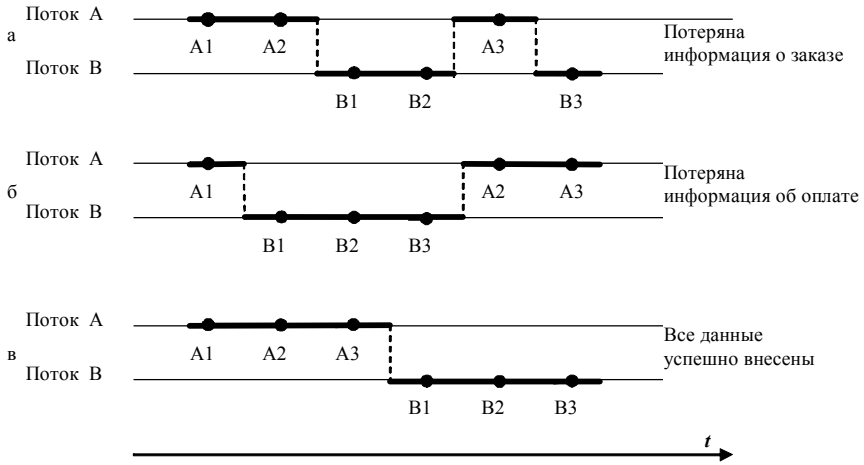


Рисунок 2.13 – Влияние относительных скоростей потоков на результат решения задачи

ционной системы, предоставляемые в форме системных вызовов. Последние являются во многих случаях более эффективными или единственно возможными. Например, потоки разных процессов могут взаимодействовать только через посредство операционной системы. Рассмотрим ряд средств синхронизации, начиная с простейших.

Простейший способ обеспечить взаимное исключение – позволить процессу, находящемуся в критической секции, ***запрещать все системные вызовы***.

Недостатки метода. Опасно доверять управление системой пользовательскому процессу; он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что системные вызовы никогда не будут разрешены.

Использование блокирующих переменных. С каждым разделяемым ресурсом D связывается двоичная переменная F(D), которая принимает следующие значения:

$$F(D) = \begin{cases} 1, \text{ если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом);} \\ 0, \text{ если ресурс занят.} \end{cases}$$

На рис. 2.14 показан фрагмент алгоритма процесса, использующего блокирующую переменную.

Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D. Если он занят, (F(D) = 0), то проверка циклически повторяется. Если же ресурс свободен (F(D) = 1), то значение переменной F(D) устанавливается в 0 и процесс входит в критическую секцию. После того как процесс выполнит все действия с разделяемым ресурсом D, значение переменной F(D) снова устанавливается равным 1.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. При этом процессы могут быть прерваны операци-

онной системой в любой момент и в любом месте, в том числе в критической секции. Ограничение на прерывания единственное: нельзя прерывать процесс между выполнением операций проверки и установки блокирующей переменной, т.е. *операция проверки и установки блокирующей переменной должна быть неделимой*. Поясним это.

Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную в 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. В итоге первый процесс полагает ресурс свободным, тогда как второй его занял. При возврате управления первому процессу он, считая ресурс свободным, установил признак занятости и начал выполнять свою критическую секцию. Таким образом, был нарушен принцип взаимного исключения, что потенциально может привести к нежелательным последствиям. Во избежание таких ситуаций в системе команд машины желательно иметь единую операцию “проверка-установка логической переменной” (например, это возможно реализовать посредством команд BTC, BTR и BTS процессоров x86) или же реализовывать системными средствами соответствующие программные *примитивы* (базовые функции ОС), которые бы запрещали прерывания на протяжении всей операции проверки и установки.

Недостатки метода. В течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выпол-

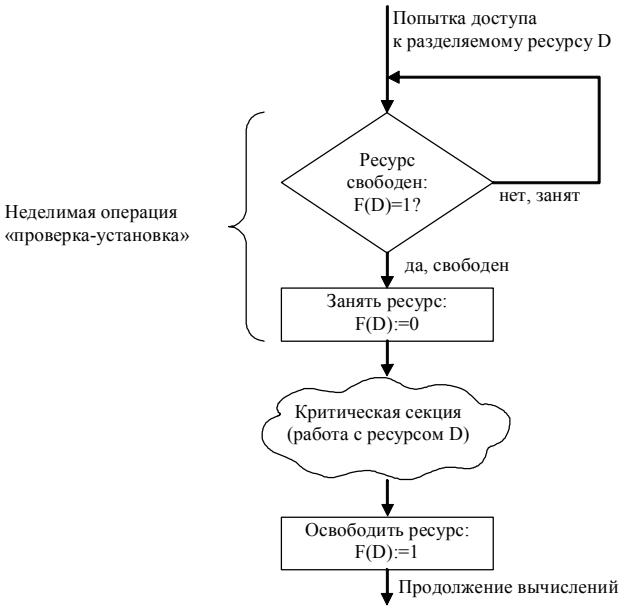


Рисунок 2.14 – Реализация критических секций с использованием блокирующих переменных

нять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время.

Специальные ***системные вызовы для работы с критическими секциями*** позволяют устранить такие ситуации простоя. В разных операционных системах соответствующие функции реализуется по-разному, но действия их и использование аналогичны. Если ресурс занят, то нуждающийся в нем процесс не выполняет циклический опрос, а вызывает системную функцию, переводящую его (процесс) в состояние ожидания освобождения ресурса. Процесс, который использует ресурс, после выхода из критической секции выполняет системную функцию, переводящую первый процесс, ожидающий ресурса, в состояние готовности.

На рисунке 2.15 показана реализация взаимного исключения при синхронизации потоков с помощью таких функций в ОС Windows NT.

Каждый из потоков, претендующих на доступ к разделяемому ресурсу D, должен содержать два системных вызова – для входа в критическую секцию с занятием ресурса, освобожденного другим потоком, и выхода из нее с освобождением ресурса для другого потока.

Поток, претендующий на доступ к критическим данным, для входа в критическую секцию выполняет системный вызов EnterCriticalSection. В рамках этого вызова выполняется

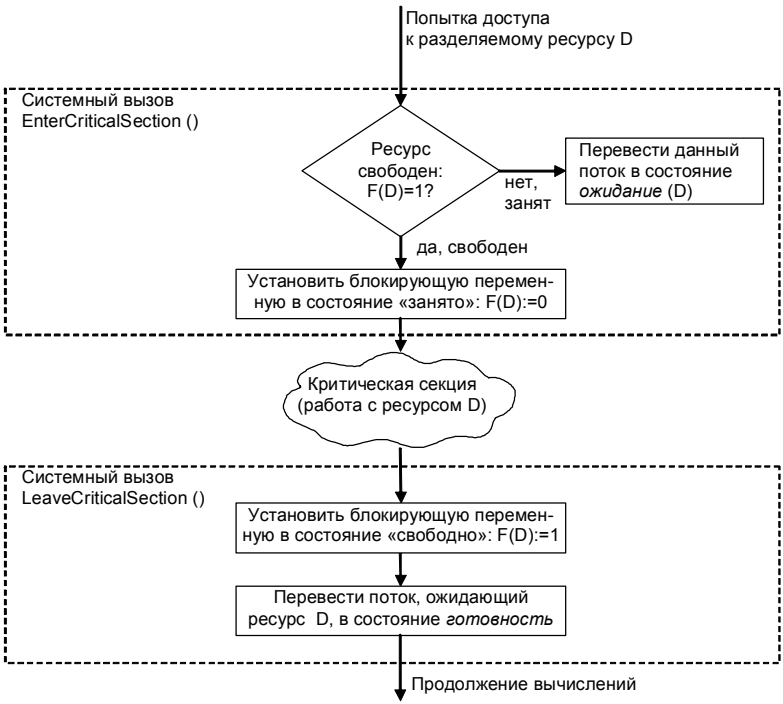


Рисунок 2.15 – Реализация взаимного исключения с использованием системных функций входа в критическую секцию и выхода из нее

проверка блокирующей переменной. В случае занятости ресурса поток переводится в состояние ожидания и делается отметка о том, что он должен быть активизирован по освобождении ресурса (поток ставится в очередь ожидающих освобождения ресурса). Если ресурс свободен, он занимается (F(D):=0), делается отметка о его принадлежности данному потоку и поток продолжает работу.

Поток, который использует ресурс, после выхода из критической секции должен выполнить системный вызов LeaveCriticalSection. В результате отмечается, что ресурс свободен (F(D):=1), и первый поток из очереди ожидающих ресурс переводится в состояние готовности.

Специфика метода. Если объем работы в критической секции небольшой и вероятность в скором доступе к ресурсу велика, то экономнее окажется метод блокирующих переменных (за счет затрат на вызов функций).

Семафоры Дейкстры – обобщение метода блокирующих переменных.

Вводятся два новых примитива. В абстрактной форме эти примитивы, традиционно обозначаемые P и V, оперируют над целыми неотрицательными переменными, называемыми *семафорами*. Пусть S – такой семафор. Операции определяются следующим образом.

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

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

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

Рассмотрим использование семафоров на классическом примере взаимодействия двух выполняющихся в режиме мультипрограммирования процессов, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула (рис. 2.16). Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. Процесс-писатель должен приостанавливаться, когда все буферы оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, процесс-читатель приостанавливается, когда все буферы пусты, и активизируется при появлении хотя бы одной записи.

Введем два семафора: e – число пустых и f – число заполненных буферов. До начала работы e=N, f=0.

Предположим, что запись в буфер и считывание из буфера являются критическими секциями. Введем двоичный семафор b, который будем использовать для обеспе-

чения взаимного исключения (т.е. этот семафор в данном качестве будет служить блокирующей переменной). Оба процесса после проверки доступности буферов должны выполнить проверку доступности критической секции.

Проблема тупиков.

Приведенный пример позволяет проиллюстрировать проблему синхронизации, называемую *тупиками, дедлоками* (deadlocks) или *клинчами* (clinch) и состоящую во *взаимных блокировках* процессов (ряд процессов удерживает ресурсы, запрашиваемые другими процессами, и в то же время запрашивает ресурсы, удерживаемые другими).

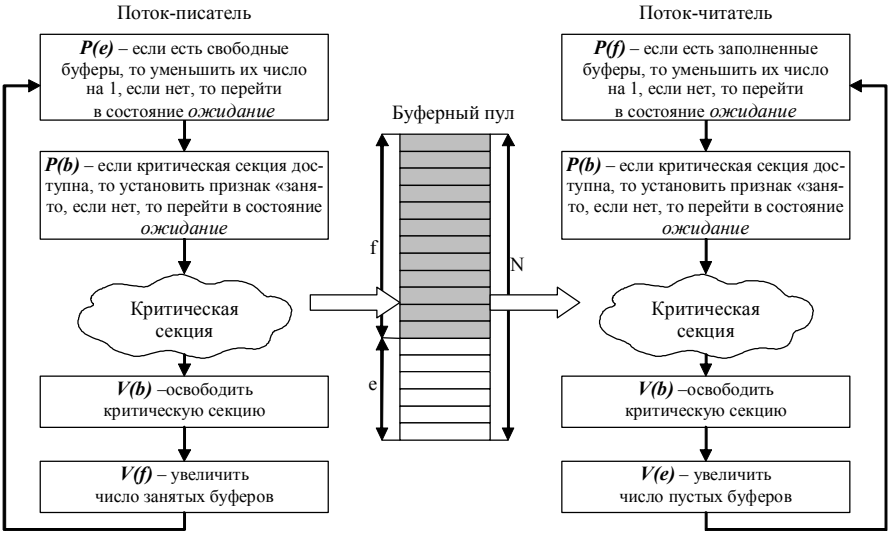


Рисунок 2.16 – Использование семафоров для синхронизации потоков

Если переставить местами операции ***P(e)*** и ***P(b)*** в программе-писателе, то при некотором стечении обстоятельств рассматриваемые два процесса могут заблокировать друг друга.

Пусть процесс-писатель первым войдет в критическую секцию и обнаружит отсутствие свободных буферов. Картина примет следующий вид.

Писатель ждет освобождения буфера, а читатель не может этого сделать, так как эти действия выполняются в ставшей недоступной критической секции.

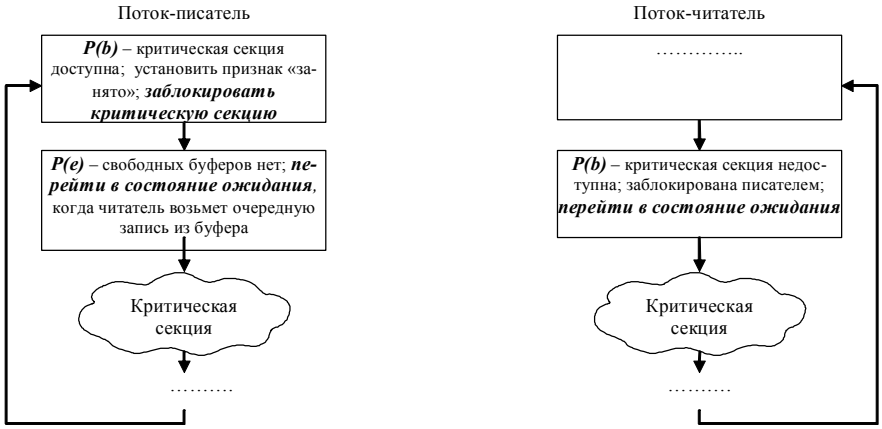
В рассмотренном примере тупик был образован двумя процессами, но взаимно блокировать друг друга могут и большее число процессов.

Решение проблемы тупиков требует решения следующих задач:

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

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

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



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

Из всего вышесказанного ясно, что использовать семафоры нужно очень осторожно, так как одна незначительная ошибка может привести к останову системы (или по крайней мере нескольких процессов).

Для того, чтобы облегчить написание корректных программ, было предложено высокоуровневое средство синхронизации, называемое монитором. ***Монитор*** – это набор процедур, переменных и структур данных. Процессы могут вызывать процедуры монитора, но не имеют доступа к внутренним данным монитора. Мониторы имеют важное свойство, которое делает их полезными для достижения взаимного исключения: только один процесс может быть активным по отношению к монитору. Компилятор обрабатывает вызовы процедур монитора особым образом. Обычно, когда процесс вы-

зывает процедуру монитора, то первые несколько инструкций этой процедуры проверяют, не активен ли какой-либо другой процесс по отношению к этому монитору. Если да, то вызывающий процесс приостанавливается, пока другой процесс не освободит монитор. Таким образом, исключение входа нескольких процессов в монитор реализуется не программистом, а компилятором, что делает ошибки менее вероятными.

Универсальные объекты синхронизации

Мьютексы. Обычно используются для управления доступом к данным.

Мьютекс – синхронизирующий объект как для процессов, так и для потоков. В каждый момент времени только один процесс (поток) имеет право обладания этим объектом. Суть его использования такова.

Пусть два (или более) процесса (потока) имеют необходимость в доступе к некоторому разделяемому ресурсу. Для организации взаимного исключения один из этих процессов должен посредством функции CreateMutex создать мьютекс с некоторым именем, предназначенный для пользования этим ресурсом. Остальные процессы должны знать это имя для обращения к мьютексу.

Существует набор функций, позволяющий запросить право владения этим объектом. Одни функции этого набора прерывают работу вызвавшего их потока до момента освобождения мьютекса, другие проверяют возможность получения права обладания и при наличии такой возможности сразу получают это право, иначе возвращают управление вызвавшему их процессу.. Первым занимает мьютекс либо его создатель, либо процесс, первым вызвавший одну из функций ожидания. По завершении работы с разделяемым ресурсом мьютекс освобождается.

Фактически мьютекс – это реализация монитора в семействе WinNT.

События. Обычно используются для оповещения процессов о завершении некоторых действий. В каждой ОС реализованы свои механизмы передачи сообщений и обработки событий.

Взаимодействие процессов и потоков включает в себя, помимо синхронизации, также и передачу данных между ними. Для передачи данных обычно используются средства ОС, специально ориентированные на межпроцессные взаимодействия.

В ОС семейства WinNT для передачи данных чаще всего употребляются сообщения (при этом «полезная информация» переносится как самим сообщением, так и его параметрами – wparam и lparam). Специально зарезервированное сообщение wm_copydata позволяет передавать большие объемы данных, используя механизм отображаемых в память файлов (memory-mapped files). Этот механизм можно использовать и напрямую, оперируя функциями winAPI.

В UNIX для передачи данных используются конвейеры (при последовательном запуске программ), каналы, сигналы (эквивалент сообщений windows), а также средства, базирующиеся на протоколе TCP/IP. Последние позволяют организовать в том числе и межмашинное взаимодействие.

3. УПРАВЛЕНИЕ ПАМЯТЬЮ

3.1. Функции ОС по управлению памятью

Если не оговорено иное, под *памятью* (*memory*) понимается оперативная память компьютера, в отличие от *внешней памяти* (*storage*).

Процессор может выполнять только инструкции, находящиеся в оперативной памяти. Память распределяется как между модулями прикладных программ, так и между модулями самой операционной системы.

Функции ОС по управлению памятью в мультипрограммной системе:

· отслеживание свободной и занятой памяти;

· выделение памяти процессам и ее освобождение при завершении процесса;

· вытеснение процессов из оперативной памяти на диск при нехватке оперативной памяти и возвращение в оперативную память при освобождении места в ней (механизм *виртуальной памяти*);

· настройка адресов программы на конкретную область физической памяти;

· динамическое выделение памяти процессам (выделение памяти по запросу приложения во время его выполнения); выделяются свободные участки, расположенные произвольным образом, что приводит к фрагментации памяти;

· дефрагментация освобожденной динамической памяти;

· выделение памяти для создания служебных структур ОС (дескрипторы процессов и потоков, таблицы распределения ресурсов, буферы, синхронизирующие объекты и т.д.;

· защита памяти – выполняемый процесс не должен записывать или читать данные из памяти, назначенной другому процессу.

3.2. Типы адресов

Для идентификации переменных и команд на разных этапах обработки программы операционной системой используются символьные имена, преобразуемые в виртуальные адреса и в итоге – в физические адреса (рис. 3.1).

Виртуальное адресное пространство

Виртуальные адреса для различных программ назначаются транслятором независимо. Диапазон виртуальных адресов определяется программно-аппаратным обеспечением компьютера, в частности, разрядностью его схем адресации. Совокупность всех возможных адресов из этого диапазона называется *виртуальным адресным пространством*.

Так, 32-разрядный процессор семейства x86 дает возможность адресовать до 2^{32} байтов, т.е. до 4 Гбайт памяти с диапазоном виртуальных адресов от 00000000h до FFFFFFFFh.

Реальные процессы используют только часть доступного виртуального пространства (на 1-2 порядка меньше максимума).

Совпадение виртуальных адресов переменных и команд различных программ не приводит к конфликтам, так как в случае, когда эти переменные или команды одно-

временно присутствуют в памяти, операционная система отображает совпадающие виртуальные адреса на разные физические (если эти переменные или команды не должны разделяться соответствующими процессами).

Образ процесса – термин, обозначающий содержимое назначенного процессу виртуального адресного пространства, т.е. коды команд и данные (исходные, промежуточные и результаты).

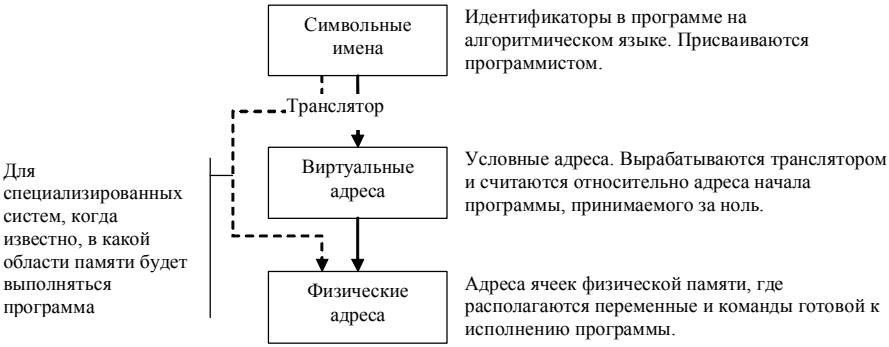


Рисунок 3.1 – Типы адресов

Способы структуризации виртуального адресного пространства в ОС

Структура адреса, или модель адресации определяется в совокупности компилятором, операционной системой и аппаратным обеспечением. Компилятор должен обеспечить простоту работы с адресом, но с минимальным разрывом между программистом и ОС. Поэтому в языке программирования отображается та модель, которая используется в ОС. Эта модель, в свою очередь, определяется заложенной в ОС идеей адресации с учетом необходимости реализации этой идеи на конкретной аппаратной платформе. Таким образом, ОС «сверху» должна обеспечить достаточно простую модель адресации для компилятора, а «снизу» уметь преобразовать эту модель в модель, навязанную аппаратурой.

Рассмотрим две наиболее характерных модели структуризации адресного пространства – плоскую и двухуровневую модель «сегмент-смещение». Эти модели представлены на рис. 3.2.

- *Плоская (flat)*структура. Виртуальное адресное пространство представлено в виде непрерывной линейной последовательности адресов. Линейный виртуальный адрес – число, представляющее собой смещение относительно начала виртуального адресного пространства (обычно это нулевое значение).

- *Сегментированная структура.* Виртуальное адресное пространство представляется разделенным на сегменты, а адрес любого объекта в памяти определяется номером сегмента и смещением относительно начала этого сегмента, т.е. парой *сегмент-смещение*.

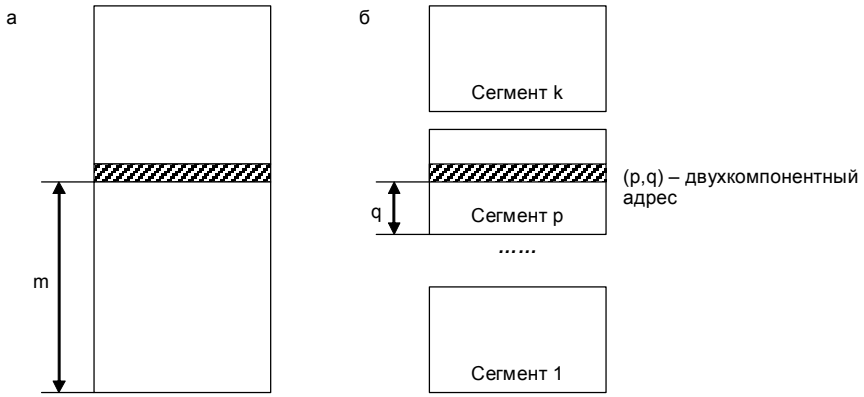


Рисунок 3.2 – Типы виртуальных адресных пространств: плоское (а), сегментированное (б)

Более конкретно способы структуризации виртуального адресного пространства рассмотрены в п. 4 темы в связи с механизмами виртуальной памяти.

Важно отметить следующее.

Использование и реализация универсального принципа сегментирования структуры адресного пространства в разные периоды развития вычислительной техники были принципиально различными и менялись по крайней мере трижды.

В ранних ОС на сегменты фиксированного размера делилась физическая память, о чем пользователь должен был знать и что при необходимости учитывал в программе. Необходимость структуризации адреса диктовалась архитектурой процессора и памяти. Так, модель памяти «сегмент-смещение» была реализована в 32-разрядной архитектуре IBM-360 (объем памяти оказывался меньше потенциально адресуемого, но тем не менее была реализована модель «сегмент-смещение») и в 16-разрядной архитектуре x-86 (по причине сугубо аппаратного свойства: процессор использовал 20-разрядную шину адреса, располагая 16-разрядными регистрами, и для формирования адреса использовалось два регистра).

Для программиста способы структуризации адресного пространства в таком случае могут отображаться в структуре адреса при работе с динамической памятью, т.е. при работе с указателями (выделение памяти, изменение значения указателя, определении содержимого памяти). Если в программе операции с указателями не используются, то по ее тексту определить эту схему в общем случае нельзя. Ниже приведены соответствующие примеры.

В языках сред turbo– и Borland–pascal имеются следующие операции с адресами, отображающие эту, неактуальную сейчас систему адресации:

- получение сегмента и смещения адреса объекта X:

function seg (var X):word; function ofs (var X):word;

- получение адреса из значений сегмента и смещения:

function ptr(<сегмент>,<смещение>:word):pointer;

Изменять значение указателя в языке нельзя. Поэтому для изменения указателя необходимо его преобразование, допускающее увеличение, изменение и затем обратное преобразование. Здесь возможны два пути.

- Следующий системе адресации MS DOS. Значение указателя раскладывают на сегмент и смещение, изменяют смещение, а затем объединяют сегмент и смещение.

- Учитывающий, что в современных операционных системах (начиная с Windows 3.11 с надстройкой Win32s) используются плоские (сплошные) адреса. Указатель преобразуется к числовому типу longint, увеличивается и преобразуется обратно.

Язык Си также содержит соответствующие операции, например:

- определение сегмента и смещения некоторого адреса (приведены примеры операторов):

seg_val=FP_SEG(p); off_val=FP_OFF(p);

- определение содержимого байта по адресу, определяемому сегментом и смещением:

b=peekb (seg, ofs).

Изменение указателя предусмотрено в языке. Так, согласно синтаксису языка, в фрагменте

```
nt* p;
p =p+1;
```

значение указателя p увеличивается на длину типа int. Схема адресации скрыта компилятором и по тексту операции неопределима.

С совершенствованием аппаратных элементов и ростом сложности программного обеспечения наблюдается тенденция к предоставлению и программисту, и процессу максимально удобной модели адресации. На данный момент таковой является плоская модель.

В современных ОС понятие «сегмент» приобрело иной смысл и используется как средство виртуализации памяти (см. ниже). Сегмент – это некоторая функционально осмысленная часть кода или данных, которая может помещаться в оперативную память или выгружаться из нее как единое целое. Элементы сегмента адресуются внутри него смещением относительно начала. При этом механизм структуризации адресного пространства прозрачен и для пользователя, и для процесса и осуществляется аппаратно-программными средствами ОС.

Для программы адресное пространство представляется плоским.

· ***Подходы к преобразованию виртуальных адресов в физические***

- Загрузка совместно с заменой виртуальных адресов физическими. Замена адресов выполняется один раз. Программа *перемещающий загрузчик*, имея начальный адрес загрузки (т.е. адрес оперативной памяти, начиная с которого будет размещена программа) и код в относительных (виртуальных) адресах, выполняет загрузку с одновременным увеличением виртуальных адресов на величину начального адреса загрузки.

- Динамическое преобразование виртуальных адресов. Программа загружается в память в виртуальных адресах. Начальный адрес загрузки ОС фиксирует в специальном регистре. Преобразование виртуальных адресов в физические (также путем прибавления начального адреса загрузки) производится во время выполнения программы при обращении к памяти. Таким образом, некоторый виртуальный адрес пересчитывается в физический столько раз, сколько обращений по нему производится.

Этот способ более гибок, так как позволяет перемещать программный код процесса во время выполнения, но менее экономичен из-за многократных преобразований одних и тех же адресов.

Понятие виртуальной памяти

Сегодня для компьютеров универсального назначения типична ситуация, когда объем виртуального адресного пространства превышает доступный объем оперативной памяти. Это достигается за счет отображения виртуального адресного пространства на физическую память посредством использования механизма виртуальной памяти.

Виртуальная память – картина памяти, формируемая операционной системой для процесса (вспомним, что одна из функций ОС – предоставление виртуальной машины; естественно предположить, что память такой машины тоже должна быть виртуальной). Деятельность ОС по созданию такой картины правомерно назвать *виртуализацией памяти*.

Например, для процессов (потоков) в Windows NT память представляется плоской (линейной) и имеет объем 4 Гб.

Реально ОС имеет в своем распоряжении некоторый объем физической оперативной памяти в виде установленных модулей (этот объем может варьироваться до 4 Гб) плюс объем, который ей разрешено использовать на диске (от 2 Мб, сверху ограничивается администратором). Эта память распределяется между всеми процессами, включая системные, отдельными фрагментами (например, страницами, см. далее). Страницы отдельного процесса располагаются частью в оперативной памяти, частью на диске в порядке, устанавливаемом ОС и в общем случае отличном от их последовательности в самом процессе (его виртуальном адресном пространстве). Эффект увеличения объема памяти достигается за счет вытеснения неактивных страниц на диск.

Таким образом, 4Гб оперативной памяти, с которой работает процесс, – фикция, создаваемая для него операционной системой.

Поскольку виртуальная память – *механизм управления* памятью, а не предоставляемое ее пространство, корректнее говорить о памяти, предоставляемой процессу посредством этого механизма. Ее объем складывается из доступного объема оперативной памяти и объема разрешенной к использованию дисковой памяти. Тогда справедливо утверждение: объем памяти, предоставляемой процессу механизмом виртуальной памяти, потенциально позволяет адресовать все виртуальное адресное пространство данного процесса. Реально на взаимодействие процессов накладывается целый ряд различных ограничений, в силу которых процессы должны вести себя корректно друг по отношению к другу, и ни один процесс не должен претендовать на всю доступную память. На сегодня «правила хорошего тона» предписывают использовать не более 200 – 500 Мб памяти, самостоятельно организуя программным путем обмен с диском в случае наличия более громоздких структур данных (как, например, это делает Adobe Photoshop).

3.3. Виды алгоритмов распределения памяти

Исторически выделяются два наиболее общих подхода к распределению памяти, в рамках каждого из которых реализуется ряд алгоритмов:

· ***распределение памяти без использования внешней памяти***:
· фиксированными разделами;
· динамическими разделами;
· перемещаемыми разделами;
· ***распределение памяти с использованием внешней памяти***:
· страничное распределение;
· сегментное распределение;
· сегментно-страничное распределение.

Алгоритмы первого класса предполагают, что размер виртуального адресного пространства каждого процесса меньше объема оперативной памяти. Эти алгоритмы использовались в ранних мультипрограммных ОС (OS/360, ранние версии OS/2) в 60-70 годах и в силу неактуальности здесь опущены.

Алгоритмы второго класса реализуют механизм виртуальной памяти и подлежат рассмотрению.

3.4. Виртуализация памяти. Классы виртуальной памяти

· Виртуализация оперативной памяти осуществляется совокупностью аппаратных средств процессора и программных средств ОС и включает решение следующих задач:

· размещение данных (образов процессов или их частей) в запоминающих устройствах разного типа: частично – в оперативной памяти, частично – на диске;

· выбор образов процессов или их частей для перемещения из оперативной памяти на диск и обратно;

· перемещение данных между памятью и диском;

· преобразование виртуальных адресов в физические.

Решение этих задач осуществляется автоматически, без участия программиста, и не отображаются в логике работы приложений.

· Виртуализация памяти может быть осуществлена на основе двух подходов – *свопинга* и механизма *виртуальной памяти*.

Свопинг (swapping). Между оперативной памятью и диском перемещаются образы процессов Более простой в реализации способ, чем виртуальная память. Однако обладает избыточностью при подкачке или выгрузке: часто для активизации процесса или освобождения памяти не требуется перемещение всего образа процесса. Избыточность приводит к замедлению работы системы и неэффективному использованию памяти. Кроме того, невозможно загрузить для выполнения процесс, виртуальное адресное пространство которого превышает имеющуюся в наличии свободную память.

Как основной механизм управления памятью в современных ОС почти не используется. В некоторых ОС, например, версиях Unix, основанных на коде SVR4, свопинг применяется как дополнительный к виртуальной памяти, включающийся только при серьезных перегрузках системы.

Виртуальная память (virtual memory). Между оперативной памятью и диском перемещаются части (сегменты, страницы – см. ниже) образов процессов.

В зависимости от способа структуризации виртуального адресного пространства, определяющего преобразование виртуальных адресов в физические, выделяется три класса виртуальной памяти.

· ***Страничное распределение***. Единицей перемещения между памятью и диском является *страница* – часть виртуального адресного пространства фиксированного и небольшого объема.

· ***Сегментное распределение***. Единицей перемещения между памятью и диском является *сегмент* – часть виртуального адресного пространства произвольного объема, содержащая осмысленную с некоторой точки зрения совокупность данных (подпрограмму, массив и т.д.).

· ***Сегментно-страничное распределение***. Объединяет элементы предыдущих классов. Виртуальное адресное пространство структурируется иерархически: делится на сегменты, а затем *сегменты делятся на страницы*. Единицей перемещения между памятью и диском является *страница*.

Для временного хранения вытесненных на диск сегментов и страниц отводится либо специальная область, либо специальный файл, обычно называемые *страничным файлом* (page file, paging file) или, по традиции, *файлом свопинга.*

Текущий размер страничного файла влияет на возможности работы ОС следующим образом: чем больше файл, тем больше одновременно работающих приложений, но тем медленнее их работа из-за многократной перекачки перемещаемых элементов на диск и обратно.

Размер страничного файла в современных ОС является настраиваемым параметром, который выбирается администратором системы для достижения компромисса между числом одновременно выполняемых приложений и быстродействием системы. Этот размер устанавливается в панели управления, пункт «система», вкладка «дополнительно» – «параметры быстродействия».

Страничное распределение

· Общая схема

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами (virtual pages)*. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

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

Для каждого процесса ОС создает *таблицу страниц* – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Схема страничного распределения памяти приведена на рис. 3.3.

При создании процесса его виртуальные страницы загружаются в оперативную память; в случае нехватки последней часть виртуальных страниц может быть вытеснена на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах.

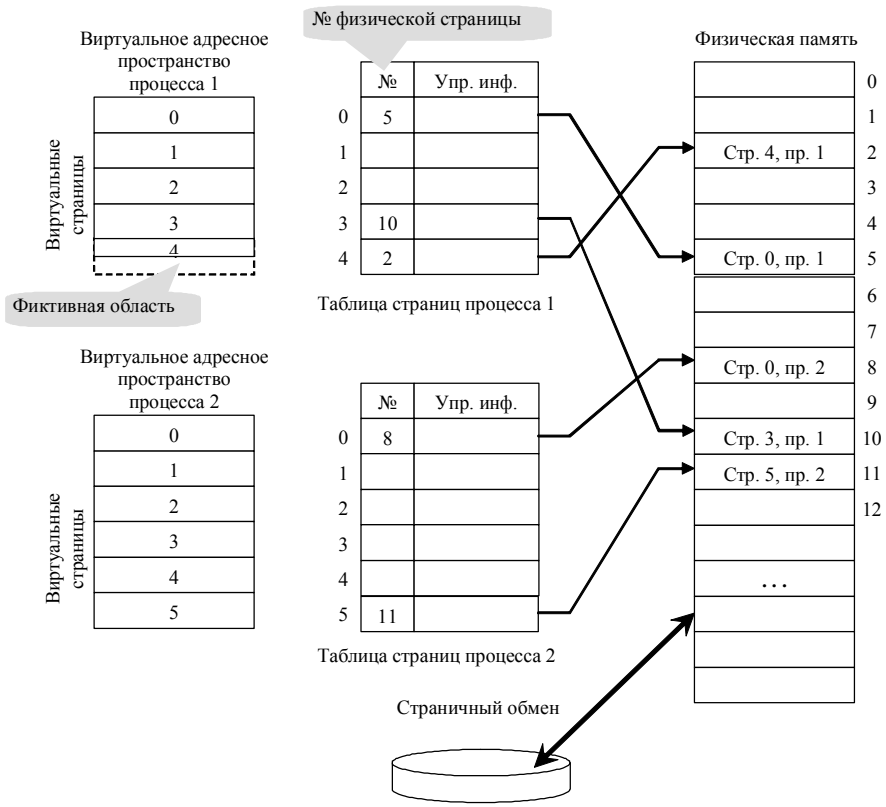


Рисунок 3.3 – Схема страничного распределения памяти

Запись таблицы страниц, называемая *дескриптором страницы*, включает следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия виртуальной страницы в оперативной памяти (1 в случае присутствия, 0 – в противном случае);
- признак модификации страницы; устанавливается в 1, когда производится запись по адресу, относящемуся к данной странице, и сбрасывается в 0, когда страница вытесняется из памяти;
- признак обращения к странице (бит доступа); устанавливается в 1 при каждом обращении по адресу, относящемуся к данной странице; используется для подсчета числа обращений за определенный период времени;
- признак невыгружаемости (выгрузка некоторых страниц может быть запрещена);
- информация о положении каждой вытесненной страницы в страничном файле;
- другие данные, формируемые и используемые механизмом виртуальной памяти.

Признаки присутствия, модификации и обращения в большинстве моделей современных процессоров устанавливаются аппаратно схемами процессора при операциях с памятью.

Таблицы страниц размещаются в оперативной памяти, а адрес каждой из них включается в контекст соответствующего процесса. При активизации процесса система загружает адрес его таблицы страниц в специальный регистр процессора.

· ***Действия при обращении к памяти***

При каждом обращении к памяти (т.е. по некоторому виртуальному адресу) выполняются следующие действия.

· Поиск в виртуальном адресном пространстве процесса номера виртуальной страницы, содержащей нужный адрес.

· Определение соответствующего этому номеру элемента таблицы страниц.

· Анализ признака присутствия.

· Если страница находится в оперативной памяти, то виртуальный адрес заменяется физическим.

· В противном случае происходит *страничное прерывание*. Процесс переводится в состояние ожидания. Программа обработки прерывания находит в страничном файле требуемую виртуальную страницу и загружает ее в свободную физическую страницу при наличии таковой. Если свободных физических страниц нет, то предварительно одна из имеющихся в оперативной памяти страниц выгружается.

· Для выгружаемой страницы обнуляется бит присутствия и анализируется признак модификации. Если страница модифицировалась во время пребывания в оперативной памяти, то новая ее версия переписывается в страничный файл. Физическая страница объявляется свободной.

· ***Преобразование виртуального адреса в физический***

И виртуальный, и физический адрес при страничном распределении представляются парой *(<номер страницы>, <смещение в пределах страницы>)*.

Пусть (p,sv) – виртуальный адрес некоторого объекта, (n,sf) – его физический адрес, где p, n – номера виртуальной и физической страниц, sv, sf – смещения в пределах страниц для виртуальной и физической страницы соответственно.

Задача подсистемы виртуальной памяти состоит в отображении (p,sv) в (n,sf). Решение этой задачи базируется на двух свойствах страничной организации.

1) Объем страницы выбирается равным степени двойки.

Пусть объем страницы равен 2^k, а общий объем памяти равен 2^v.

Тогда любой плоский адрес представляется v-разрядным двоичным числом от 0 до 2^v -1, а его значение определяется выражением: (<номер страницы>) * 2^k + <смещение> (страницы и адреса нумеруются с нуля).

Таким образом, последние k разрядов двоичной записи адреса представляют собой смещение, а первые (v-k) – номер страницы. Начальный адрес страницы можно получить, приписав к номеру k нулей.

Например, при объеме памяти в 4 Гб = 2^{32} и объеме страницы в 4 Кб = 2^{12} число страниц будет равно 2^{32-12}, двоичный номер каждой страницы представляется первыми 20 разрядами, а смещение – последними 12 разрядами.

Пусть адрес равен 12 F1 A0 01_{16}. Определим номер его страницы и смещение:
12 F1 A0 01_{16} = 0001 0010 1111 0001 1010 0000 0000 0001_2.

2) Последовательность адресов внутри виртуальной страницы отображается без изменения, т. е. смещения в виртуальном и физическом адресах равны: sv = sf.

Схема преобразования виртуального адреса в физический приведена на рис. 3.4. Помимо введенных выше, использованы следующие обозначения:

a – адрес нужного дескриптора в таблице страниц,

AT – начальный адрес таблицы страниц выполняющегося процесса,

L – длина отдельной записи в таблице страниц (системная константа).

Тогда a = AT + (p x L).

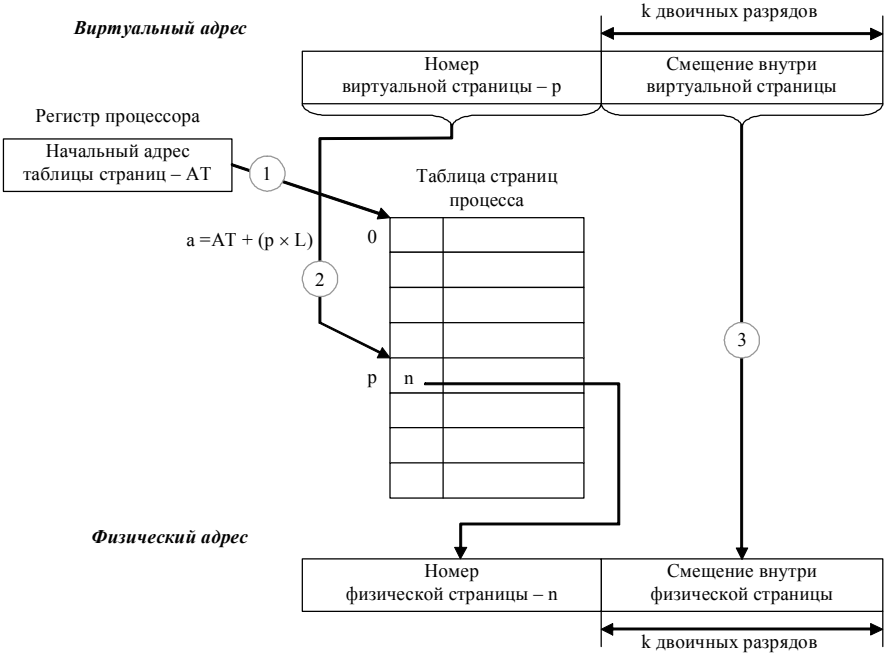


Рисунок 3.4 – Схема преобразования виртуального адреса в физический при страничной организации

На уровне аппаратных схем процессора выполняется следующее.

· Из специального регистра извлекается адрес таблицы страниц.

· На основании этого адреса, номера виртуальной страницы и длины записи таблицы страниц определяется адрес нужного дескриптора.

· Из этого дескриптора извлекается номер соответствующей физической страницы.

· К полученному номеру присоединяется (простым приписыванием, *конкатенацией*) смещение.

· ***Факторы, влияющие на производительность системы***

Частота страничных прерываний и размер страницы. При часто возникающих страничных прерываниях система может тратить большую часть времени впустую, на свопинг страниц. Чтобы уменьшить частоту страничных прерываний, следовало бы увеличивать размер страницы. Кроме того, увеличение размера страницы уменьшает размер таблицы страниц, а значит уменьшает затраты памяти. С другой стороны, если страница велика, значит велика и фиктивная область в последней виртуальной странице каждой программы. В среднем на каждой программе теряется половина объема страницы, что в сумме при большой странице может составить существенную величину.

Время доступа к таблице страниц. Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в "быстрых" запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

Критерий выбора страницы на выгрузку. Поскольку точно предсказать ход вычислительного процесса невозможно, используются эмпирические критерии, часто основывающиеся на предположении об инерционности вычислительного процесса. Наиболее популярные из таких критериев:

· вытесняется страница, к которой в последнее время было меньше всего обращений,

· вытесняется первая попавшаяся страница,

· вытесняется дольше всего не использовавшаяся страница.

Сегментное распределение

· Общая схема

Виртуальное адресное пространство процесса делится на *сегменты* – логические, осмысленные с точки зрения обработки фрагменты (программный код, данные, стек процесса – с точки зрения ОС, подпрограмма, массив данных – с точки зрения программиста).

Разбиение виртуального адресного пространства на сегменты дает следующие преимущества по сравнению со страничной организацией:

· возможность задания дифференцированных прав доступа процесса к его сегментам (для одних – только чтение, для других – чтение и запись и т.д.);

· возможность организации совместного использования фрагментов программ разными процессами (например, использование одной и той же подпрограммы).

В общем случае деление виртуального адресного пространства на сегменты осуществляется по умолчанию в соответствии с принятыми в системе соглашениями или компилятором на основе указаний программиста. Так, основными типами сегментов на уровне ОС являются сегмент данных, сегмент кода, системные сегменты.

Виртуальное адресное пространство процесса представляет собой набор виртуальных сегментов. Максимальный размер сегмента определяется разрядностью виртуального адреса (4 Гб при 32-разрядной организации). Каждый сегмент располагает своим независимым виртуальным адресным пространством с адресами от

нулевого до максимально возможного. Общего для сегментов линейного виртуального адреса не существует. В каждом сегменте виртуальные адреса задаются парой «номер сегмента – смещение внутри сегмента». Схема сегментного распределения памяти приведена на рис. 3.5.

Сегментное распределение памяти имеет много общего со страничным. На этапе создания процесса во время загрузки его образа в оперативную память система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается:

- начальный физический адрес сегмента в оперативной памяти;
- размер сегмента;
- права доступа к сегменту;
- признаки модификации, присутствия и обращения к данному сегменту за последний интервал времени, а также некоторая другая информация.

Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

Для каждого загружаемого сегмента ОС подыскивает непрерывный участок свободной памяти достаточного размера. В оперативную память помещается только часть сегментов; остальные размещаются на диске.

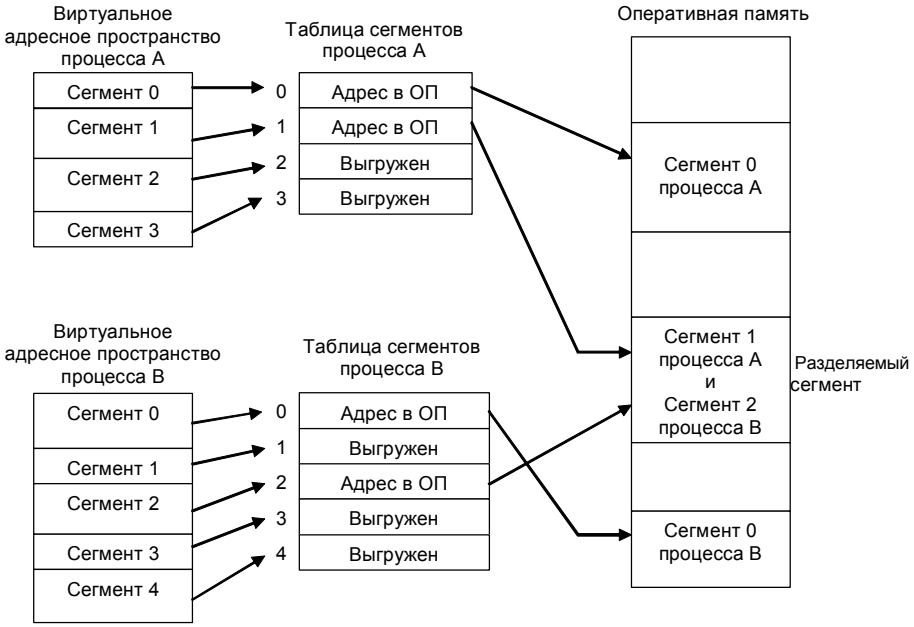


Рисунок 3.5 – Схема сегментного распределения памяти

·***Действия при обращении к памяти*** при сегментной организации, аналогичны действиям при страничной организации (при необходимости – прерывание, вытеснение сегмента и т.д.). Используются те же критерии вытеснения. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.

· Преобразование виртуального адреса в физический

Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g – номер сегмента, а s – смещение в сегменте.

В отличие от страницы, сегмент:

- имеет произвольный размер;

- располагается в физической памяти начиная с любого адреса.

Поэтому преобразование виртуального адреса в физический менее эффективно, чем при страничной организации:

- нельзя получить адрес начала сегмента по его номеру, и в таблице сегментов необходимо задавать полный адрес его начала (при страничной организации достаточно номера);

- физический адрес формируется путем сложения (вместо конкатенации) сегмента и смещения.

Схема преобразования виртуального адреса в физический приведена на рис. 3.6.

· Недостатки метода

Избыточность. Единица перемещения между памятью и диском – сегмент целиком, чего во многих случаях не требуется.

Фрагментация памяти. При многократной загрузке – выгрузке образуются небольшие участки свободной памяти, не вмещающие ни одного сегмента, но в сумме имеющие значительный объем.

Сегментно-страничное распределение

·*Общая схема*

Виртуальное адресное пространство процесса разделено на *сегменты*. Это позволяет определять разные права доступа к разным частям кодов и данных программы.

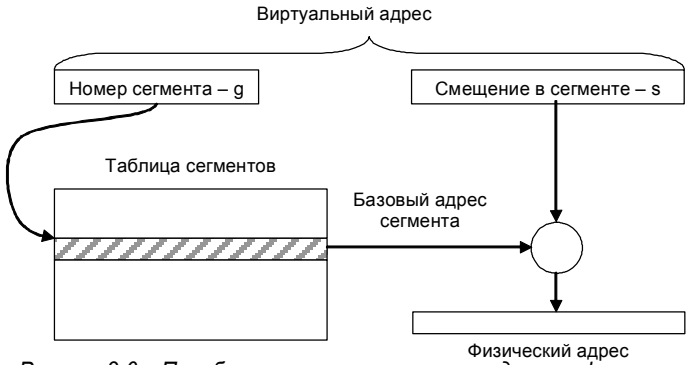


Рисунок 3.6 – Преобразования виртуального адреса в физический при сегментном распределении

Перемещение данных между памятью и диском осуществляется *страницами*, что позволяет минимизировать фрагментацию оперативной памяти. Для этого виртуальное адресное пространство и физическая память делятся на страницы равного размера.

Сегментация. В большинстве современных реализаций сегментно-страничного распределения все виртуальные сегменты образуют одно непрерывное виртуальное адресное пространство (в отличие от описанного в п. 4.3 чисто сегментного распределения).

Адрес в виртуальном адресном пространстве при сегментно-страничном распределении задается парой «номер сегмента – смещение относительно начала сегмента». Это позволяет проверить принадлежность адреса некоторому сегменту и соответствующие права доступа.

Для каждого процесса создается таблица сегментов, содержащая дескрипторы сегментов. В отличие от дескриптора сегмента при сегментном распределении, содержащего физический адрес сегмента, в данном случае в дескриптор заносится *начальный (базовый) линейный виртуальный адрес сегмента* в пространстве виртуальных адресов.

Соответствующая схема сегментации приведена на рис. 3.7.

Пара «базовый виртуальный адрес сегмента» – «смещение относительно начала сегмента» однозначно преобразуется в линейный виртуальный адрес, который далее преобразуется в физический адрес уже страничным механизмом.

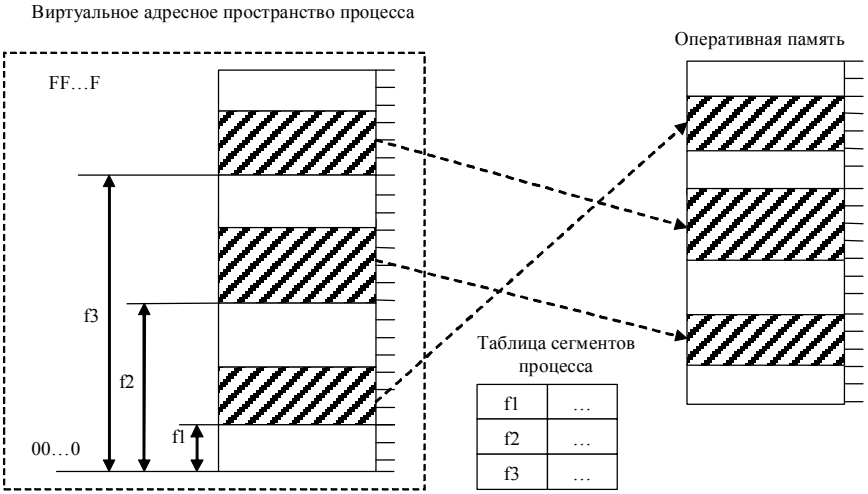


Рисунок 3.7 – Сегментация с непрерывным виртуальным адресным пространством

Страничный механизм. Деление общего линейного виртуального адресного пространства процесса и физической памяти и на страницы и действия при обращении к памяти осуществляются так же, как при страничной организации памяти.

Базовые адреса таблицы сегментов и таблицы страниц являются частью контекста процесса; при активизации процесса загружаются в специальные регистры и используются при преобразовании адресов.

· ***Преобразование виртуального адреса в физический*** осуществляется в два этапа механизмом сегментации и страничным механизмом (рис. 3.8).

На первом этапе вычисляется адрес поля дескриптора сегмента и анализируются права доступа к сегменту. Если доступ разрешен, то виртуальный адрес в виде пары «сегмент – смещение» преобразуется в линейный виртуальный адрес. Поскольку разбиение на страницы и размер страниц выбираются так же, как при страничном распределении, этот адрес одновременно представлен в виде «номер страницы – смещение внутри страницы».

На втором этапе преобразование адреса происходит так же, как при страничной организации.

Возможен и другой вариант комбинирования сегментного и страничного механизмов, который здесь не рассматривается: виртуальное адресное пространство процесса делится на сегменты, а каждый сегмент – на страницы. Виртуальный адрес в таком случае выражается тройкой «сегмент – страница – смещение в странице».

3.5. Кэширование данных

· Память вычислительной машины, представленная совокупностью запоминающих устройств (ЗУ) различных видов, может быть иерархизирована по следующим основным признакам:

· время доступа к данным;

· объем;

· стоимость хранения данных в расчете на один бит.

Конкретные значения этих характеристик изменяются очень быстро, поэтому в данном случае важны не столько их абсолютные значения, сколько соотношение для разных типов запоминающих устройств. Иерархия ЗУ приведена на рис. 3.9.

Закономерность такова: чем больше быстродействие, тем больше стоимость хранения данных в расчете на один бит и меньше объем устройства. Кэш-память представляет некоторое компромиссное решение этой проблемы.

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

Механизм кэш-памяти прозрачен для пользователя: все перемещения данных делаются автоматически системными средствами.

Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств – “быстрое” ЗУ. Оно стоит дороже

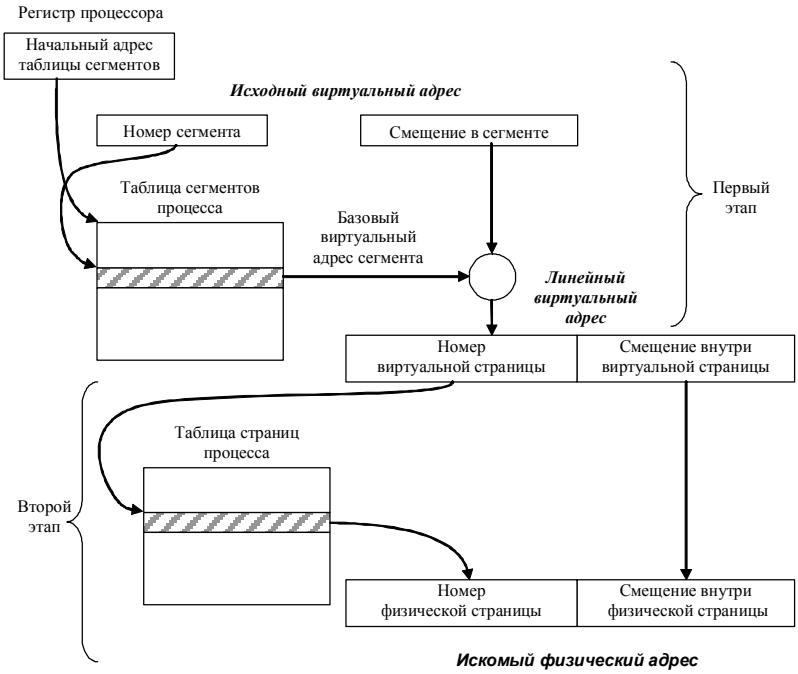


Рисунок 3.8 – Схема преобразования виртуального адреса в физический при сегментно-страничной организации

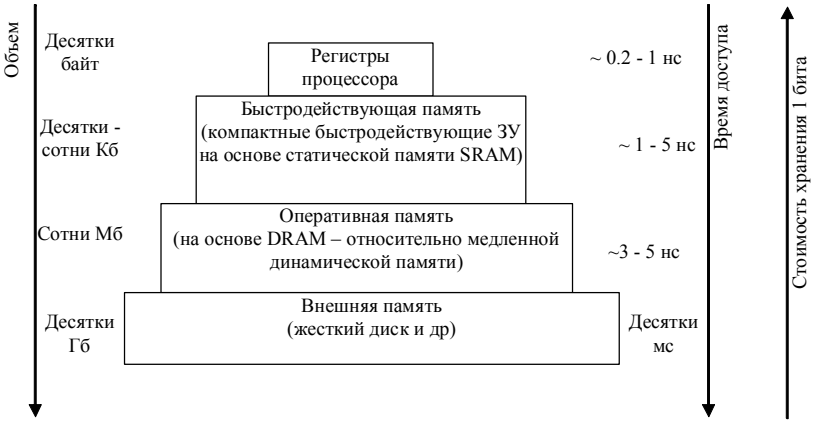


Рисунок 3.9 – Иерархия запоминающих устройств

и, как правило, имеет сравнительно небольшой объем. Медленное ЗУ из этой пары назовем *основной памятью*, быстрое представлено кэш-памятью.

· Кэширование – универсальный метод, пригодный для ускорения доступа к оперативной памяти, к диску и другим видам ЗУ. Так, в качестве составляющих пары «основная память – кэш-память» могут выступать: оперативная память – быстродействующая статическая память; система ввода-вывода – буферы в оперативной памяти (или специальная кэш-память).

Функционирование кэш-памяти

Рассмотрим одну из возможных схем кэширования.

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

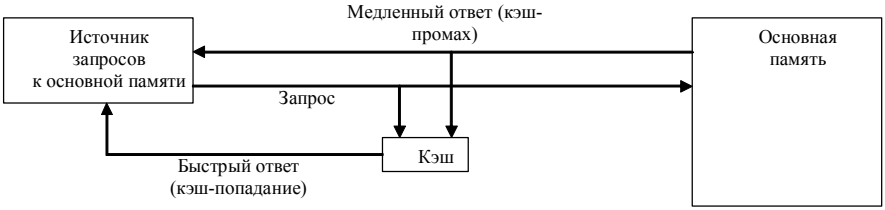
- значение элемента данных;
- адрес, который этот элемент данных имеет в основной памяти;
- управляющую информацию для реализации алгоритма замещения, обычно – признак модификации и признак обращения к данным за некоторый последний период времени.

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти с целью определения, не находятся ли там нужные данные. Поиск нужных данных осуществляется по содержимому – взятому из запроса значению поля адреса в оперативной памяти. Далее возможно одно из двух:

· произошло кэш-попадание – данные обнаружены в кэш-памяти; они считываются из кэш-памяти и результат передается источнику запроса;

· произошел кэш-промах (cache-miss) – нужных данных нет; они считываются из основной памяти, передаются источнику запроса и одновременно копируются в кэш-память.

Схема функционирования кэш-памяти приведена на рис. 3.10.



Структура кэш-памяти

Адрес данных в основной памяти	Данные	Управляющая информация

Рисунок 3.10 – Схема функционирования кэш-памяти

Покажем, что среднее время доступа к данным зависит от вероятности попадания в кэш.

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

$t = t1((1 – p) + t2p$.

Видно, что среднее время доступа изменяется от среднего времени доступа в основное ЗУ (при p=0) до среднего времени доступа непосредственно в кэш-память (при p=1).

Таким образом, использование кэш-памяти имеет смысл только при высокой вероятности кэш-попадания. Эта вероятность, в свою очередь, зависит от многих различных факторов. Тем не менее в реальных системах вероятность попадания в кэш очень высока и составляет более 0.9. Такое высокое значение вероятности нахождения данных в кэш-памяти связано с наличием у данных объективных свойств: *пространственной* и *временной локальности*.

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

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

На практике в кэш-память считывается не один элемент данных, к которому произошло обращение, а целый блок данных, что увеличивает вероятность попадания в кэш при последующих обращениях.

Проблемы кэширования

· Вытеснение данных их кэша в основную память

В процессе работы содержимое кэш-памяти постоянно обновляется, а значит, периодически данные должны из нее вытесняться. Вытеснение предполагает объявление соответствующей области кэша свободной (сброс бита действительности) и, если вытесняемые данные за время нахождения в кэше были изменены, копирование данных в основную память.

Методы выбора данных для вытеснения зависят от способа отображения основной памяти на кэш и базируются на предположениях о свойствах данных. Реально как правило учитывается интенсивность обращения к данным. В некоторых алгоритмах замещения предусматривается первоочередная выгрузка модифицированных данных.

Из-за непредсказуемости вычислительного процесса ни один алгоритм замещения данных не может быть максимально быстрым и одновременно гарантировать максимально возможную вероятность кэш-попаданий. Поэтому разработчики ограничиваются рациональными решениями, по крайней мере, не сильно замедляющими работу кэша.

· Согласование данных кэша и основной памяти при записи в последнюю

Проблема заключается в том, что при записи данных в основную память их копия в КЭШе (если она там есть) становится недостоверной. Для решения этой проблемы типичны два подхода.

Сквозная запись (write through). Если данные по запрашиваемому адресу отсутствуют в кэше, то запись выполняется только в основную память, в противном случае – одновременно в кэш и основную память.

Обратная запись (write back). Если данные по запрашиваемому адресу отсутствуют в кэше, то запись выполняется только в основную память, в противном случае –только в кэш-память. Во втором случае устанавливается признак модификации, указывающий на необходимость записи данных в основную память при вытеснении их из кэша.

Способы отображения основной памяти на кэш

Основными являются две схемы отображения: случайное и детерминированное отображение.

· Случайное отображение

Элемент основной памяти вместе с его адресом размещается в любом месте кэш-памяти. Адрес выступает в качестве признака, по которому производится поиск, или *тега* (*tag*). Схема поиска представляет собой ассоциативный поиск, при котором сравнение значения тега выполняется *параллельно* со всеми записями КЭШа. Такая схема реализуется аппаратно и приводит к удорожанию памяти, поэтому ассоциативная кэш-память используется в случаях, когда для обеспечения высокого процента попадания достаточно небольшого объема памяти.

Выбор данных на выгрузку происходит только в случае, когда вся кэш-память заполнена. Приемы выбор – те же, что при замещении страниц.

· Детерминированное отображение

Любой элемент основной памяти всегда отображается в одно и то же место кэш-памяти. Кэш-память разделяется на нумерованные строки. Между номерами этих строк и адресами основной памяти устанавливается соответствие «один ко многим»: одному номеру строки соответствует множество адресов основной памяти. Это множество может, например, характеризоваться младшими разрядами адресов. Для установления однозначного соответствия необходима дополнительная проверка, для чего каждая строка кэш-памяти дополняется тэгом, содержащим старшую часть адреса.

Стоимость КЭШа относительно низкая.

Алгоритмы замещения на основе детерминированного отображения существенно отличаются от алгоритмов, используемых при случайном отображении.

· Смешанная стратегия

Сочетает описанные подходы и используется во многих современных процессорах.

Произвольный адрес основной памяти отображается на некоторую группу адресов КЭШа. Группы пронумерованы. Отображение на группу прямое. Внутри группы отображение случайное.

Поиск в кэше осуществляется сначала по номеру группы, полученному из адреса в запросе, а затем – ассоциативно – внутри группы.

Алгоритм замещения может учитывать интенсивность обращения к данным в КЭШе и тем самым повысить вероятность попадания в будущем.

Двухуровневое кэширование

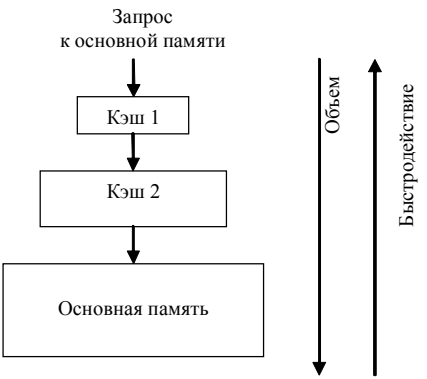


Рисунок 3.11 – Схема двухуровневого кэширования.

Такая схема используется во многих вычислительных системах.

При выполнении запроса сначала осуществляется поиск в кэше 1-го уровня. Если произошел промах, то поиск продолжается в кэше 2-го уровня, при промахе и здесь – в основной памяти.

При работе такой иерархически организованной памяти необходимо обеспечить многоуровневое копирование и непротиворечивость данных на всех уровнях.

Двухуровневое кэширование само по себе не является средством повышения производительности в том плане, что кэш, имеющий объем, равный хотя бы объему кэша второго уровня, но работающий со скоростью кэша первого уровня, дал бы более существенный прирост производительности. Однако зачастую это просто невозможно реализовать технически (либо такая реализация будет чрезмерно дорогой), тогда многоуровневое (процессор AMD k6-3 использовал три уровня) кэширование является компромиссом между скоростью и размером кэша.

4. ВВОД-ВЫВОД И ФАЙЛОВАЯ СИСТЕМА

4.1. Файловая система ОС

Файл. С одной стороны, это формально не определяемое понятие. С точки зрения смысла файл – множество данных, объединенных некоторой логической связью, т.е. одна и та же совокупность данных может рассматриваться как один или несколько файлов (исходные данные к задаче; учебные материалы и т.п.).

С другой стороны, понятие «файл» относится к объекту, вполне однозначно описываемому следующими признаками:

- файл объединяет множество данных;
- обладает именем;
- с целью долговременного и надежного хранения информации располагается на внешнем устройстве;
- предполагает многократное использование информации с разрывом во времени;
- предполагает совместное использование информации несколькими приложениями или пользователями, одновременно (разделяемый ресурс) или с разрывом во времени.

Последнее описание представляется вполне однозначным и при работе с файлами уточняется в зависимости от аспекта рассмотрения.

Файловая система как часть операционной системы – это подсистема, включающая:

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

Файловой системой называют также совокупность всех файлов на диске.

Использование одного и того же термина в двух смыслах как правило не приводит к недоразумениям, так как из контекста всегда ясно, о каком аспекте определения файловой системы идет речь – о средствах ОС или совокупности файлов.

Принятое выше определение файла представляет файл с точки зрения пользователя. Детали действительного расположения данных на внешнем устройстве и работы с ними на низком (физическом) уровне файловая система берет на себя, экранируя все сложности этого уровня и предоставляя пользователю удобную логическую модель и набор соответствующих команд.

Общая задача файловой системы состоит в предоставление пользователю логической модели для работы с файлами и отображении этой модели на физическую организацию внешнего устройства.

Логическая модель обеспечивает удобный для пользователя интерфейс и скрывает физическую организацию работы с внешними устройствами. Этот интерфейс обеспечивает следующие логические операции и средства:

· именование файлов;

· поддержка различных типов файлов;

· задание атрибутов файлов;

· организация хранения множества файлов;

· поддержка логической организации файлов;

· предоставление программного интерфейса для работы с различными файлами (в виде совокупности системных функций, например, WinAPI).

Отображение логической модели на физическую организацию внешнего устройства, или, что то же, реализация этой модели на физическом уровне, предполагает следующее:

· отображение (трансляция) имен файлов в адреса внешней памяти (сектора диска, адреса флэш-накопителя и т.п.);

· размещение данных на устройстве;

· обеспечение доступа к данным;

· буферизация обмена;

· организация совместного использования файлов (блокировка; предотвращение гонок и тупиков; согласование копий и др.);

· защита файлов одного пользователя от несанкционированного доступа другого;

· восстановление файлов в случае возникновения ошибок различного рода;

Таблица 4.1

Уровни работы с файлами

		Суть понятия «файл»	Средства работы, операции	Кто реализует
Логический уровень программы	Программа	Совокупность данных, с точки зрения программиста объединенных некоторой логической связью и размещенных на внешнем устройстве. По структуре – последовательность осмысленных единиц данных.	Описание представления файла в программе; открытие, закрытие, чтение, запись осмысленными единицами; множество операций, воспроизводящих операции ОС на уровне языка	Программист
Физический уровень программы. Физич. уровень		Именованная область данных на внешнем устройстве.	Связывание логического представления файла с реальным (физическим) файлом	Компилятор, загрузчик
Логический уровень ФС	Операционная система	Последовательность байтов, возможно, структурированная с точки зрения ОС	Средства именования и задания свойств файлов; организация множества файлов; обеспечение доступа к файлам; обработка; обеспечение целостности.	ФС ОС
			Организация хранения и обработки файла на устройстве	Средства управления внешней памятью
	АпАппаратура («железо»)	Область, последовательность байтов на внешнем устройстве	Чтение или запись в заданном диапазоне физических адресов	Контроллеры

· обеспечение устойчивости файловой системы к сбоям питания и программно-аппаратным ошибкам;

· обеспечение работы с файлами в сети.

Соответственно двум составляющим описанной задачи правомерно говорить о ***логической*** и ***физической*** организации файловой системы.

4.2. Логическая организация файловой системы

Имена файлов

Правила формирования имен файлов определяются операционной системой и учитываются ограничения на используемые символы, длину и структуру имени. Здесь речь идет только о *простых* именах.

Например, файловые системы ОС Unix различают символы верхнего и нижнего регистров (большие и малые буквы), тогда как для ОС семейства WIndows регистр не учитывается.

Ограничения на длину и структуру имени также различны и до недавнего времени были весьма сильными.

Так, в файловой системе FAT16 (Microsoft) структура имени подчиняется схеме «8.3» (8 символов – собственно имя, 3 символа – расширение имени). В ОС UNIX System V имя не может содержать более 14 символов.

Понятие расширения введено в MS DOS; в ОС UNIX оно отсутствует. Тенденция развития ФС такова, что ограничение на структуру имени снимается и в ОС семейства Microsoft: в имени файла может присутствовать более одной точки; однако в этих ОС последние 1 – 4 символа после точки сохраняют свою роль, так как распознаются приложениями. С другой стороны, в ОС UNIX также используются соглашения об именовании файлов, согласно которым тип файла в имени отделяется точкой.

Современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, Windows NT в файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

В некоторых системах одному и тому же файлу не может быть дано несколько разных имен (например, в ОС семейства Microsoft). В других такое ограничение отсутствует (например, в ОС UNIX). В последнем случае файл не имеет имени как такового. С каждым файлом ФС связывает *индексный дескриптор*, хранящий *метаданные*, содержащие все характеристики файла. Имя файла является указателем на его метаданные и, как следствие, на одни и те же метаданные может указывать несколько указателей. Действия по заданию альтернативных имен (псевдонимов) осуществляются командами ОС.

Типы файлов

Обычные файлы содержат информацию произвольного характера, которую заносит в них пользователь или программа, системная или пользовательская. Содержание такого файла определяется приложением, которое с ним работает. Системное

или стандартное приложение создает и распознает файлы своего собственного формата (текстовый редактор Word – файлы .doc, графическая программа Photoshop – .psd, .tif и т.п.); пользовательское приложение интерпретирует содержимое файла в соответствии с задачей и способом ее решения.

Обычные файлы в свою очередь подразделяются на файлы *во внешнем* и *внутреннем* представлении. Файлы первого типа условно можно назвать *текстовыми*. Они состоят из строк символов, представленных в ASCII-коде, и интерпретируются пользователем как текст в обычном понимании. Это могут быть документы, исходные тексты программ, исходные данные к программам и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Файлы второго типа условно можно назвать *двоичными*. Эти файлы создаются программным путем; их структура определяется программой – создателем (объектный код программы, исполняемый код, архивный файл и т.п.; простейший случай – файл чисел во внутреннем представлении, созданный пользовательской программой). Все операционные системы должны уметь распознавать хотя бы один тип файлов – их собственные исполняемые файлы.

Каталог – системный файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами). Двойственный смысл, вкладываемый в понятие каталог, практически не приводит к недоразумениям, так как в одном случае речь идет о схеме хранения файлов, а в другом – о хранении информации, описывающей ту схему.

Организация каталогов и их содержимое различаются в разных ОС, в частности, в MS DOS, Windows и Unix.

Так, каталог Unix – таблица, каждая запись которой соответствует некоторому файлу и содержит имя файла и указатель на дополнительную информацию – *метаданные*, хранящиеся в *индексных дескрипторах* (*inode*). Указатель представлен номером inode.

Каталог MS DOS хранит имена файлов и целый ряд атрибутов – дату и времена создания, последнего доступа и изменения; текущий размер файла; признаки “только для чтения”, “архивный файл”, “скрытый файл”, “системный файл”; номер начального кластера файла.

Специальные файлы – это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем преобразуются ОС в команды управления соответствующим устройством.

Символьная связь (Unix) – особый тип файла, позволяющий косвенно адресовать другой файл.

Символьная связь создается командным путем с целью использования файла некоторого каталога в другом каталоге. В итоге с точки зрения пользователя в его каталоге присутствует нужный файл, реально же там присутствует символьная связь.

Именованный канал, ***файл, отображаемый на память*** – типы файлов, используемые для связи между процессами. Различные ОС используют различные механизмы связи.

Атрибуты файлов

Атрибут – это информация, описывающая некоторое свойство файла, или некоторая характеристика файла. В разных файловых системах могут использоваться разные атрибуты и их наборы, хотя реально для современных файловых систем как смысл атрибутов, так и их совокупности имеют тенденцию к сближению.

Ниже перечислены возможные атрибуты, для некоторых из которых указываются ФС, где они используются:

- тип файла;
- владелец файла (NTFS, Unix);
- создатель файла;
- пароль для доступа к файлу (NTFS, Unix);
- информация о возможности доступа (права доступа) (NTFS, Unix);
- времена создания; последнего доступа и последнего изменения (везде);
- текущий размер файла (везде);
- максимальный размер файла;
- признак “только для чтения”;
- признак “скрытый файл”; FAT, NTFS;
- признак “системный файл”; носят информативно-рекомендательный характер
- признак “архивный файл”;
- признак “двоичный/символьный” (Unix);
- признак “временный” (удалить после завершения процесса) (Unix);
- признак блокировки (NTFS, Unix);
- длина записи;
- указатель на ключевое поле в записи;
- длина ключа.

Средства доступа к атрибутам предоставляются файловой системой. Обычно читать можно любые значения, а изменять – только некоторые. Например, при наличии соответствующих полномочий пользователь может изменить права доступа к файлу, но дату создания менять нельзя.

Дерево каталогов

Во всех широко используемых файловых системах различных ОС на уровне пользователя файлы организуются в *дерево каталогов* (одно или несколько). В UNIX пользователь, желая иметь в некотором каталоге файл из другого каталога, может соответствующей командой установить так называемую символьную связь, т.е. ссылку на существующий файл. В таком случае, если помнить об этой ссылке, дерево преобразуется в сеть. Учитывая, однако, что, установив эту связь, пользователь далее считает нужный файл размещенным в его каталоге и работает с деревом, все же правомернее на логическом уровне считать общую структуру каталогов древовидной (как, кстати, она и представляется в литературе)

В ОС Windows дисковое пространство разбивается пользователем на несколько логических дисководов, каждый из которых содержит свое дерево каталогов, не связанное с деревьями других дисководов.

В ОС UNIX все доступное пользователям файловое пространство объединено в единое дерево каталогов.

Каждый файл регистрируется только в одном каталоге. Каждый каталог имеет имя и может быть зарегистрирован в другом каталоге. Если каталог X зарегистрирован в каталоге Y, то X – *подкаталог* Y, а Y – *надкаталог* X.

Каталог высшего уровня – *главный*, или *корневой*. Он один, не имеет имени и обозначается в Windows как <имя дисковода>:\ , а в UNIX – как / (прямой слэш), без указания накопителя.

Текущий каталог – тот, с которым в текущий момент работает пользователь.

Простое, полное и относительное имя файла

Простое имя идентифицирует файл в пределах каталога, в котором файл непосредственно размещен.

Путь к файлу – последовательность имен каталогов, разделенных знаком слэш (прямым или обратным, в зависимости от ОС), начиная с имени текущего каталога и до каталога, в котором находится файл.

Полное имя файла в дереве каталогов – путь от корня к файлу с приписанным через слэш простым именем файла. Если деревья расположены на разных логических дисках, то в полное имя в общем случае включается имя дисковода; в противном случае подразумевается текущий дисковод.

Полное имя однозначно идентифицирует файл.

Если в некоторой команде указано полное имя файла, то он доступен из любого каталога.

Если путь не указан, то подразумевается текущий каталог. При таком указании имени файлы одного каталога недоступны из другого.

Относительное имя файла – путь от текущего каталога к файлу с приписанным через слэш простым именем файла.

Пример дерева каталогов для Windows приведен на рис. 4.1.

Здесь, например, пути от корневого каталога к файлам l.txt и s.doc, не включая корневой каталог:

P\E и C\I\J соответственно.

Пусть изображенная на рисунке система каталогов размещена на диске A:. Тогда полные имена файлов l.txt и s.doc запишутся так:

A:\P\E\ l.txt и A:\C\I\J\ s.doc соответственно.

Монтирование

Реально компьютер оснащен большим количеством дисковых накопителей даже без учета разбиения, например, винчестера на логические диски. Объективно в большинстве случаев файлы одного пользователя находятся на различных устройствах, поэтому размещение на каждом из устройств автономной файловой системы (как в ОС Windows) – один из естественных способов организации файлов.

Файловая система ОС UNIX организует единое дерево каталогов путем монтирования, или присоединения произвольной файловой системы к общему дереву каталогов в любой его точке.

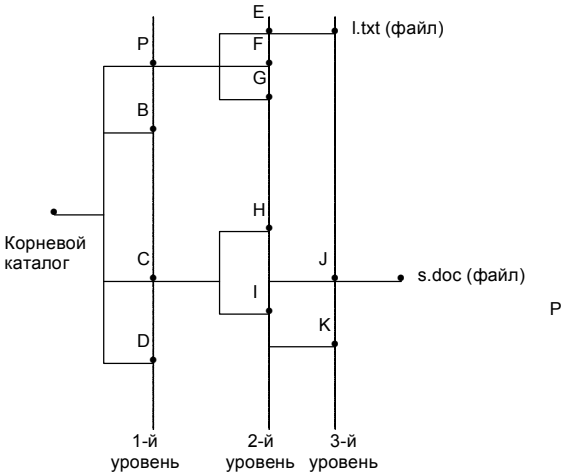


Рисунок 4.1 – Пример дерева каталогов

Одно из устройств является системным. На нем находятся загрузчик и все основные файлы системы. Корневой каталог этого устройства считается корневым каталогом (root) системы. Пусть соответствующее дерево каталогов – файловая система 1 (ФС1).

Пусть имеется отдельная файловая система 2 (ФС2).

Монтирование ФС2 к некоторому каталогу ФС1 состоит в присоединении ФС2 к этому каталогу как корню. ФС2 станет поддеревом единого дерева ФС1. Соответствующая иллюстрация приведена на рис. 4.2.

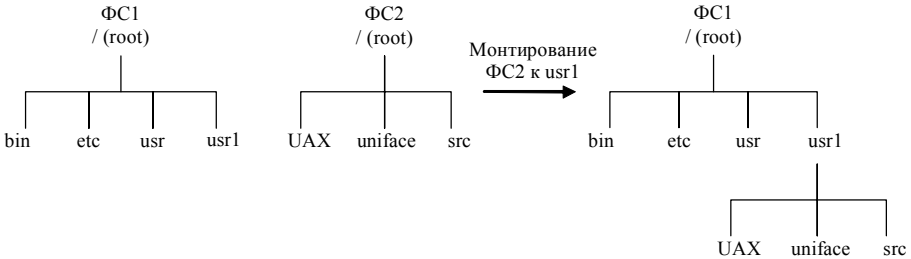


Рисунок 4.2 – Иллюстрация операции монтирования файловых систем в ОС Unix

Логическая организация файлов

·Два подхода к логической организации файлов

Очевидно, что на самом нижнем уровне представления вся информация, касающаяся файла (и собственно данные, и управляющая информация), – не более чем совокупность байтов. Интерпретация (осмысление) этой совокупности на том или ином уровне (уровне устройства; ОС; приложения; пользователя) представляет собой уже соответствующий логический уровень.

Пользователем данные файла интерпретируются в соответствии с содержанием решаемой задачи. Эта интерпретация в той или иной степени отображается в программе – описании данных и операциях с ними, в частности, файловых операциях. Далее в зависимости от используемых операций уже операционная система понимает файл как имеющий (с ее точки зрения) некоторую структуру или как простую последовательность байтов. Хотя понятие «смысл данных» для каждого уровня свое, эти уровни интерпретации тесно связаны тем, какие осмысленные единицы выделил в данных файла пользователь и как он отобразил их в программе.

Все системные программы предназначены для решения своих специфических задач, поэтому в каждую из них заложено «понимание» своего формата данных (компилятор генерирует объектный модуль определенного формата как выходные данные; для редактора связей это формат входных данных; файловая система должна различать разные типы файлов и т.д.). Поэтому вопрос о структуре файлов относится прежде всего к обычным пользовательским файлам с произвольным с точки зрения ФС содержанием.

В первых ОС (OS 360, ОС малых машин) структурирование данных в файле поддерживалась файловой системой. Это означает, что программист на соответствующем языке ОС описывал желаемую структуру, а ОС создавала файл с соответствующей организацией (так, OS 360 поддерживала несколько типов файлов с достаточно сложной структурой). Развитием этого подхода стали системы управления базами данных.

В настоящее время наиболее популярна модель файла как неструктурированной (неинтерпретируемой) последовательности байтов. Эта модель является более гибкой и эффективной для работы ОС, например, с точки зрения обмена с внешними устройствами, с точки зрения разделения файла между приложениями и т.д.

· ***Первый подход*** предполагает, что единица данных для обмена с внешним устройством также осмысленна и определяется программистом. Такая единица называется *логической записью*, и информация о том, что является такой записью, явно присутствует в программе или сообщается ОС на соответствующем языке. ОС должна обеспечивать доступ к отдельной записи как неделимой единице.

Способ доступа к записям файла определяет порядок их обработки (считывания – записи). Возможны два способа доступа:

· *последовательный* – доступной для обработки является запись, непосредственно следующая за обработанной; так, если была обработана 3-я запись, то доступной является только 4-я; чтобы получить доступ к 5-й, надо обработать (хотя бы пропустить) 4-ю;

· *прямой* – каждая запись имеет некоторый *ключ*; доступной для обработки является запись с заданным ключом, вне зависимости от того, какая запись была доступна перед этим; так, если ключом является номер записи и была обработана 3-я запись, то получить доступ к 5-й можно, указав ее номер в соответствующих операторах.

Способ доступа к записям и способ структурирования файла взаимосвязаны. Реально операционными системами поддерживается (или поддерживалось) небольшое число схем структурирования, три из которых приведены на рис. 4.3 – 4.5.

Возможен и последовательный, и прямой доступ по номеру записи, так как начальный адрес любой записи легко вычисляется исходя из ее номера и длины.

Для каждой записи задается ее длина. Возможен только последовательный доступ, так как вычислить адрес некоторой записи по ее номеру нельзя.

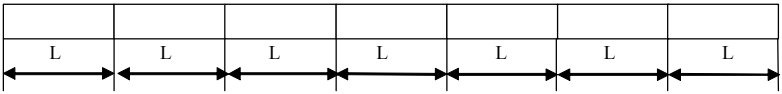


Рисунок 4.3 – Последовательная организация с записями фиксированной длины

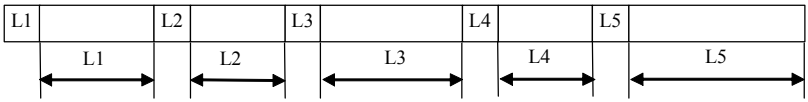


Рисунок 4.4 – Последовательная организация с записями переменной длины

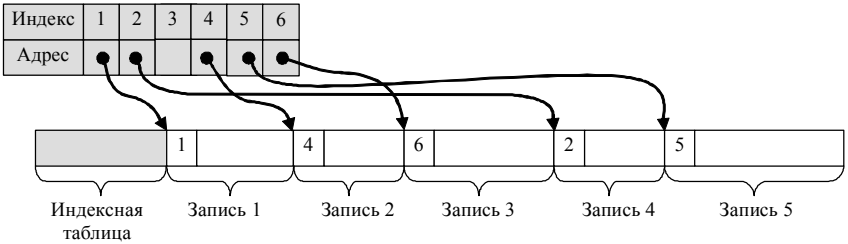


Рисунок 4.5 – Прямая организация. Индексированный файл

Записи имеют одно или более ключевых, или индексных, полей и могут быть расположены в произвольном порядке. Обращение происходит к записи с заданным ключом. Для быстрого поиска строится *индексная таблица*, содержащая упорядоченный список ключевых полей записей с сопоставленными им адресами записей в файле. Доступ осуществляется в два этапа: сначала по имени файла определяется адрес индексной таблицы, затем – собственно доступ.

Расширением этого способа организации является индексно-последовательная организация, когда адрес индексной таблицы указывает на группу записей, просматриваемых последовательно.

На уровне программирования способ доступа определяется операциями языка программирования.

· ***Второй подход*** предполагает, что приложение полностью берет на себя интерпретацию содержимого файла, а обмен между оперативной и внешней памятью осуществляется последовательностями байтов заданной длины, начиная с заданной позиции. Такие неинтерпретированные последовательности обычно называются *блоками* (паскаль, Си). Структура блока никак не отображается в структуре файла.

Этот подход выигрывает не только с точки зрения эффективности работы ОС, но и с точки зрения программирования, если при проектировании ввода-вывода учесть некоторые моменты.

- *В программе* характеристики блока обязаны присутствовать, и в первую очередь надо определить, что именно будет выступать в качестве блока. Для корректной работы с файлом структура его содержимого должна быть известна, и в общем случае данные осмысленны. Поэтому в качестве блока как единицы обмена реально выбирается осмысленная совокупность данных, т.е. то, что в первом подходе представляло собой логическую запись.

- Практически в операциях прямого доступа в качестве ключа рассматривается *номер записи*. Тогда оптимальной оказывается схема рис. 4.3. Такая схема реальна для файлов *во внутреннем представлении* (т.е. в представлении согласно типу, описанному в программе). В этом случае все блоки будут иметь один и тот же смысл и в то же время одну и ту же длину.

- Для чтения данных во внешнем представлении, записанных в виде обычного текста, в программе должны быть использованы операторы, понимающие такой текст как последовательность, включающую числа, символы и строки. В паскале это означает описание файла как текстового (тип text) и соответствующий способ считывания его компонентов; в Си используется *форматированный ввод* (потоковый или файловый, с форматированием по умолчанию, либо явным заданием форматов).

Файловые операции

· Множество операций с файлами, несмотря на их внешние различия, включает следующие группы основных операций:

· открытие файла;
· закрытие файла;
· создание файла (может быть совмещено с открытием);
· назначение файла (может быть совмещено с открытием);
· чтение из файла;
· запись в файл;
· прямой доступ к файлу.

Эти операции представлены на уровне операционной системы в виде системных функций. Так, соответствующие функции в ОС Windows предоставляются пользовательским программным интерфейсом WinAPI; ОС Unix предоставляет два основных интерфейса для ввода-вывода: так называемый низкоуровневый, функции которого непосредственно взаимодействуют с ядром системы, и стандартную библиотеку ввода-вывода.

И в том, и в другом случае должна быть написана программа, содержащая вызовы функций, на языке, компилятор с которого понимает системные вызовы. Фактически ведущим системным языком является Си (в том числе для Windows). К функциям WinAPI можно обратиться также из программы на object-паскале и иных языках.

При работе с файлами в прикладной программе также используются перечисленные группы операций, но уже как средства, включенные в язык программирования. Это операции более общие, не предусматривающие системных тонкостей.

Классы файловых операций ОС и аналогичных операций в языке программирования внешне могут как походить один на другой, так и кардинально отличаться. Например, пользовательская программа обработки файлов на Си может обращаться к тем же функциям, что и системная. Напротив, обработка файлов, описанная в программе средствами языка Паскаль, кардинально отличается от аналогичных действий, описанных системными средствами – как по внешнему виду, так и по сути: используются разные наборы функций.

Тем не менее каждый класс можно распознать по именам функций, которые включают корневую часть термина, определяющего суть функции: open, close, creat, assign, read, get, write, put, seek. Полные описания соответствующих функций относятся к документации операционной системы и требуются при непосредственной работе на системном уровне в конкретной ОС. Здесь мы рассмотрим только суть перечисленных классов системных операций и приведем имена некоторых функций.

· Открытие файла

Служит для получения доступа на чтение или запись к файлу. Если файл существует, он открывается и процессу возвращается указатель на файл, использующийся далее в операциях чтения-записи. Если файл н существует, он может быть создан.

Unix: open, fopen (функции с префиксом f относятся к стандартной библиотеке ввода-вывода).

Windows: CreateFile, mmioOpen.

· Закрытие файла

Разрывает связь между открытым файлом и указателем на файл.

Unix: close, fclose.

Windows: Функция CloseHandle (используется в том числе и для закрытия файлов).

· Создание файла (может быть совмещено с открытием)

Unix: creat.

Windows: CreateFile.

· Чтение из файла

Производит чтение указанного количества байтов из файла, начиная с позиции файлового указателя.

Unix: read, fread.

Windows: ReadFile.

· *Запись в файл*

Производит запись указанного количества байтов из файла, начиная с позиции файлового указателя.

Unix: write, fwrite.

Windows: WriteFile.

· Прямой доступ к файлу

Устанавливает файловый указатель на определенное место в файле, смещая его на заданное число байтов от текущей позиции, начала или конца файла.

Unix: lseek, fseek.

Windows: mmioSeek.

Напоминание. Во-первых, приведены только основные классы *обязательных* операций (функций); во-вторых, каждый из классов содержит не одну функцию; в-третьих, как классы, так и множества функций, в них входящие, различаются для различных ОС.

4.3. Физическая организация файловой системы

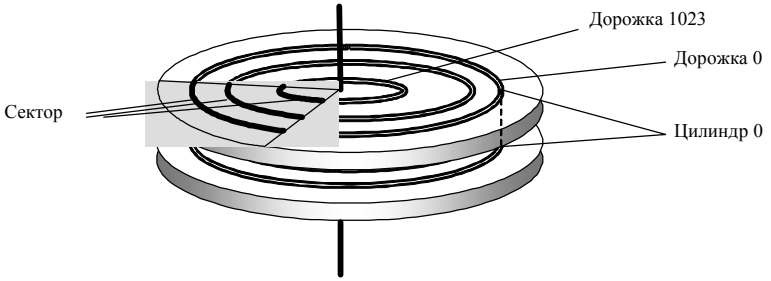


Рисунок 4.6 – Схема устройства жесткого диска

Жесткий диск в общем случае состоит из *пакета пластин*.

На каждой стороне каждой пластины размещены *дорожки* (*tracks*), на которых хранятся данные. Нумерация дорожек начинается с 0 от внешнего края диска.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин называется *цилиндром*.

Каждая дорожка разбивается на *секторы*, или *блоки* фиксированного размера, кратного двум (как правило, 512 байтов). Все дорожки имеют одинаковое число секторов, поэтому плотность записи тем выше, чем ближе дорожка к центру.

Дорожки и секторы создаются при *физическом*, или *низкоуровневом* форматировании диска, предшествующем его использованию. Низкоуровневый формат не зависит от типа операционной системы, которая будет этот диск использовать.

Сектор – наименьшая ***физическая адресуемая единица*** обмена данными диска с оперативной памятью. Доступ к секторам осуществляется на уровне контроллеров дисководов. Адрес сектора включает три составляющих: номер цилиндра, номер поверхности и номер сектора.

Операционная система использует ***логическую адресуемую единицу*** дискового пространства – ***кластер*** (***claster***). При создании файла память на диске ему выделяется кластерами.

Разметку диска под конкретный тип файловой системы выполняют процедуры *высокоуровневого*, или *логического* форматирования. При этом определяется размер кластера и на диск записывается информация о доступном и неиспользуемом пространстве, о поврежденных секторах, о границах областей, отведенных под файлы, а также загрузчик операционной системы.

Перед форматированием диска под определенную файловую систему он может быть разбит на *разделы*, или *логические устройства*, с которыми пользователь работает как с отдельными физическими дисками. При общем понятии раздела сами разделы специфичны для разных ОС. В одном разделе может быть установлена только одна файловая система. В разных разделах одного диска могут быть установлены разные файловые системы.

Физическая организация и адресация файла

· ***Основные критерии эффективности*** физической организации файла:

· скорость доступа к данным;

· объем адресной информации файла;

· степень фрагментированности дискового пространства;

· возможность увеличения размера файла.

· Варианты физической организации

Основные варианты приведены на рис. 4.7 с кратким описанием и общей оценкой.

- *Непрерывное размещение.* Схема очевидна из рисунка.

- *Связанный список кластеров*. В начале каждого кластера файла содержится указатель на следующий кластер. Расположение файла может быть задано номером первого кластера.

- *Связанный список индексов*. Может рассматриваться как усовершенствование предыдущего варианта. Применяется в файловой системе FAT для ОС семейства Microsoft.

С каждым кластером диска связывается индекс (номер). Индексы располагаются в отдельной таблице – FAT (File Allocation Table), занимающей один кластер. Когда память свободна, все индексы имеют нулевое значение.

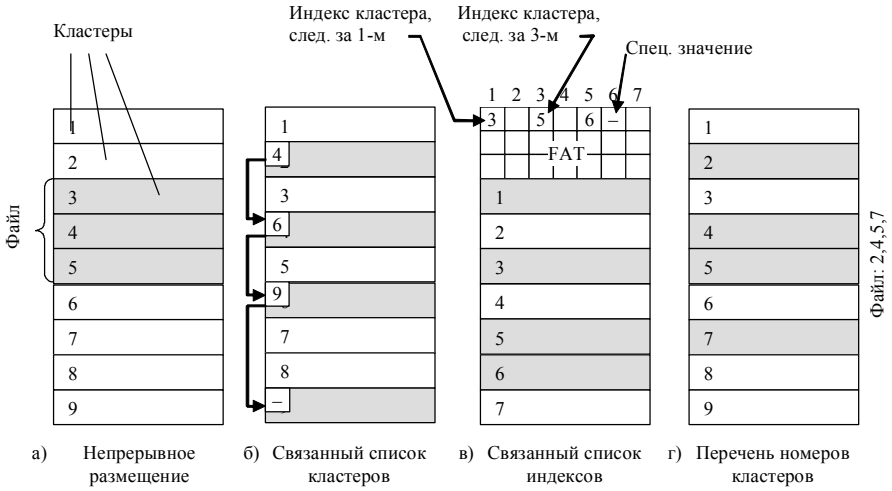


Рисунок 4.7 – Физическая организация файла

Файлу выделяется память в виде связанного списка кластеров. Индекс первого кластера файла запоминается в записи каталога, где хранится файл (см. п. 2.1 темы). Если некоторый кластер не последний в файле, то соответствующий ему индекс содержит номер следующего кластера, в противном случае – специальный признак конца файла. При присоединении очередного кластера к файлу индекс последнего кластера файла меняется на индекс присоединяемого кластера, а присоединяемый становится последним.

Сохраняются достоинства предыдущего способа плюс преимущества доступа. Номер любого кластера файла просто определяется по таблице индексов.

- *Перечисление номеров кластеров*, занимаемых файлом. Этот перечень служит адресом файла.

· Сравнительная оценка вариантов

Знаками + и – помечены достоинства и недостатки вариантов соответственно.

Критерий/ организация	Непрерывное размещение	Связанный список кластеров	Связанный список индексов	Перечисление номеров кластеров
Скорость доступа	высокая: нет затрат на поиск и считывание кластеров файла (+)	невысока: доступ к кластерам последовательный (–)	высокая: доступ близок к прямому (+)	высокая: прямой доступ к кластеру (+)
Объем адресной информации	минимален: номер первого кластера и объем файла (+)	минимален: номер первого кластера (+)	минимален: номер первого кластера и FAT (+)	длина адреса зависит от размера файла (–)
Степень фрагментации диска	высокая (–)	на уровне кластеров фрагментация отсутствует (+)	на уровне кластеров фрагментация отсутствует (+)	на уровне кластеров фрагментация отсутствует (+)
Возможность увеличения размера файла	проблемы: необходимость выделения сплошного участка (–)	число кластеров легко наращивается (+)	число кластеров легко наращивается (+)	увеличение файла увеличивает длину адреса (–)

· Модификация последнего способа, используемая в файловых системах s5 и ufs ОС Unix

Для сокращения объема адресной информации прямая адресация сочетается с косвенной (передачей адреса по адресу). Схема организации файлов приведена на рис. 4.8.

Для хранения адреса файла выделено 15 полей по 4 байта, отводимых на номер кластера. В первых 12 полях (с 0 по 11-е) размещаются номера первых 12 кластеров файла. 13-е поле содержит номер кластера, где могут быть расположены *номера* следующих кластеров файла. 14 поле содержит номер кластера, в котором размещены номера кластеров, хранящих номера кластеров уже непосредственно с данными.

Пусть кластер имеет размер 8 Кб (8192 б). Тогда первыми 12 полями можно адресовать 8192*12=98304 байтов.

Кластер, на который указывает 13-е поле, содержит 8192/4 = 2048 номеров кластеров следующего уровня, т.е. позволяет адресовать еще 8192*2048 байтов. Аналогично можно рассчитать объем памяти, адресуемой через посредство двойной и тройной косвенной адресации.

В итоге максимальный размер файла составляет $8192*(12+2048+2048^2+2048^3)$ байтов. При этом объем адресной информации равен 15 элементов по 4 байта плюс $(1+(1+2048)+(1+2048+2048^2))$ = 4 198 403 кластера для косвенной части адреса, что составляет около 0,05 % объема адресуемых данных, т.е. очень небольшой процент.

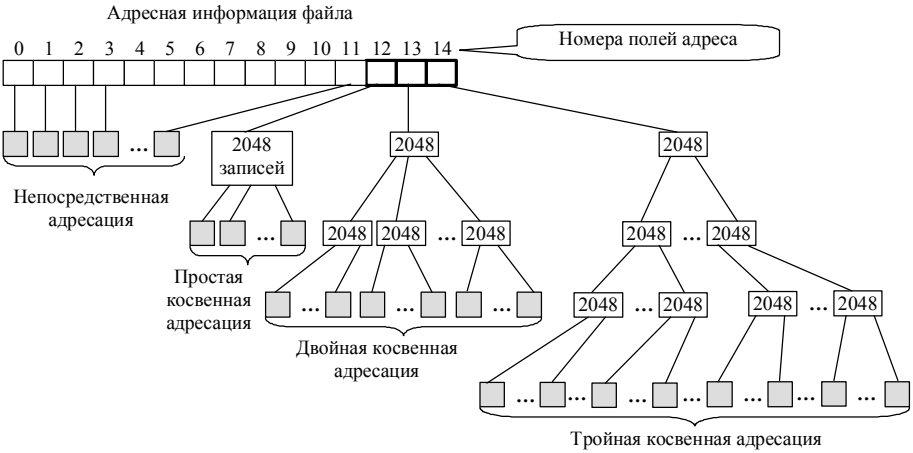


Рисунок 4.8 – Схема адресации файловой системы ufs

· Действия при обращении из программы к данным на диске

- FAT

Последовательное обращение (для определенности – чтение). Физический номер первого кластера определяется по записи о файле в каталоге. Первым читается этот кластер. В ячейке FAT, соответствующей этому кластеру, записан номер следующего кластера файла и т.д.

При чтении из текущего кластера позиция указателя файла смещается на число считанных байтов. Когда данные кластера исчерпаны, по содержимому соответствующей ячейки FAT определяется, какой кластер читать далее. Этот кластер становится текущим и т.д.

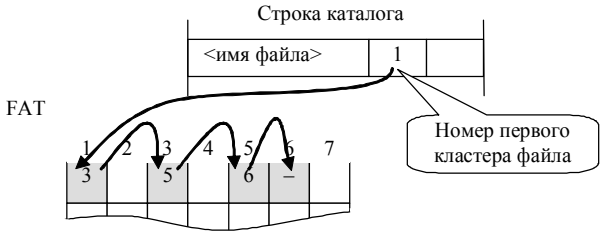
Прямой доступ. Непосредственно в операции чтения указывается позиция первого считываемого байта файла (исходя из логической модели, где файл представляется сплошной последовательностью байтов). По номеру этого байта определяется порядковый номер кластера в файле. Пусть этот номер – n.

Чтобы определить физический номер соответствующего кластера, надо последовательно просмотреть FAT по ссылкам, начиная с ячейки, содержащей номер первого кластера файла. (n–1)-й по счету элемент списка кластеров содержит физический номер искомого кластера, а n-й – следующего (n = 1, 2, …). Например, чтобы определить физический номер третьего кластера файла, необходимо знать содержимое второго элемента списка кластеров этого файла. Иллюстрация, соответствующая рис. 4.7, в, приведена ниже.

- ufs

Последовательное обращение. Первым читается кластер, физический номер которого записан в нулевом (первом по счету) поле адреса. Номер следующего кластера файла записан в первом поле адреса и т.д.

При чтении из текущего кластера позиция указателя файла смещается на число считанных байтов. По исчерпании данных текущего кластера считыванию подлежит кластер, номер которого указан в следующем поле адреса. Этот кластер становится текущим и т.д.



Прямой доступ. Непосредственно в операции чтения указывается позиция первого считываемого байта файла. По номеру этого байта определяется порядковый номер кластера в файле. Пусть этот номер – n. При n<=12 физический номер кластера расположен в n-м поле адреса; при n>12 для определения этого номера используется косвенная адресация согласно рис. 4.8.

Например, физический номер третьего кластера файла содержится в поле адреса с номером 2. Адресная информация для файла, изображенного на рис. 4.7, г, приведена ниже. Номера кластеров представлены в шестнадцатеричной системе счисления (4-байтовое поле содержит 8 шестнадцатеричных цифр).

0	1	2	3	4	…	12	13	14
00000002	00000004	00000005	00000007	00000000	…	00000000	00000000	00000000

Некоторые файловые системы

· Файловые системы ОС семейства Microsoft

- FAT

Общая организация файлов – связанный список индексов.

Цифры в названиях конкретных версий FAT означают разрядность индексных указателей (элементов таблицы FAT).

Размер кластеров выбирается из диапазона от 512 байтов до 64 кБ.

FAT12 используется для форматирования небольших дисков (до 16 мБ), практически – дискет. Поддерживает только короткие имена файлов по схеме «8.3».Эта ФС соответствует первым версиям ОС MS DOS.

FAT16 используется для дисков до 2Гб (небольших винчестеров). Максимальный размер раздела под FAT16 ограничен 4 гБ (65 536 кластеров по 64 кБ). Соответствует ОС MS DOS и Windows.

FAT32 – файловая система, адекватная современным дисковым накопителям. Использует класстеры разного объема: 4 кБ для дисков до 8 гБ; 8, 16, 32 кБ – для дисков большего объема. Максимальный размер раздела практически не ограничен – 2^{32}

кластеров по 32 кБ. Поддерживает короткие и длинные имена файлов. Соответствует всем ОС Windows.

- NTFS

Общая организация файлов основана на методе перечисления номеров кластеров файла. Для сокращения адресной информации адресуются не отдельные кластеры, а непрерывные области смежных кластеров диска – *отрезки* (*run*), или *экстенты* (*extent*). Экстент описывается парой чисел « начальный номер кластера – количество кластеров в экстенте». ФС NTFS используется в ОС Windows NT/2000/ XP.

· Файловые системы ОС Unix

ФС современных версий Unix имеют весьма сложную архитектуру, различную для различных версий. Тем не менее все они используют базовые идеи, заложенные в них ведущими разработчиками – компаниями AT&T (Unix System V) и Berkley Software Distribution (BSD Unix).

- s5, ufs

Общая организация файлов основана на методе перечисления номеров кластеров файла, сочетаемом с косвенной адресацией кластеров (см. п. 3.2). Это базовые файловые системы для Unix System V.

- FFS

Внесены существенные улучшения в архитектуру предшествующих ФС, повышающие производительность и надежность системы. FFS обладает полной функциональностью s5 и ufs, но оптимизированы расположение ФС на диске, дисковые структуры данных и алгоритмы размещения свободных блоков. Это базовая ФС BSD Unix.

· Поддержка нескольких файловых систем

ОС Windows предусматривает установку в разных разделах диска разных файловых систем, совместимых с Windows.

В ОС Unix на основе System V Release 4 имеется специальный слой программного обеспечения – *виртуальная файловая система* (*Virtual File System, VFS*), позволяющая установку в разных разделах диска разных файловых систем, включая «не родные», например, FAT32 и NTFS.

4.4. Подсистема ввода-вывода

Основные задачи управления вводом-выводом

Управление всеми устройствами ввода-вывода компьютера является одной из главных функций ОС. ОС должна:

· передавать устройствам команды;

· перехватывать прерывания и обрабатывать ошибки;

· обеспечивать интерфейс между устройствами и остальной частью системы.

При этом для обеспечения модифицируемости и расширяемости системы интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

Физическая организация устройств ввода-вывода

· Типы устройств

Устройства ввода-вывода делятся на два типа: *блок-ориентированные* устройства и *байт-ориентированные* устройства.

Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство – диск.

Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры. Однако некоторые внешние устройства не относятся ни к одному классу, например, часы, которые, с одной стороны, не адресуемы, а с другой стороны, не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

· Физическая структура внешнего устройства

Внешнее устройство обычно состоит из *механического* и *электронного* компонента.

Механический компонент представляет собственно устройство.

Электронный компонент называется *контроллером* устройства, или *адаптером*, и управляет механическим компонентом.

Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать как совместимые контроллеры, так и совместимые устройства.

· Взаимодействие ОС с внешним устройством

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

ОС выполняет ввод-вывод, записывая команды в регистры контроллера. Например, контроллер гибкого диска IBM PC принимает 15 команд, таких как READ, WRITE, SEEK, FORMAT и т.д. Когда команда принята, процессор оставляет контроллер и занимается другой работой. При завершении команды контроллер организует прерывание для того, чтобы передать управление операционной системе, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

Обобщенная структура подсистемы ввода-вывода

· Основные проблемы организации программного обеспечения ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование

особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

- Независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска.

- Обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Нижний уровень должен сообщать об ошибке верхнему, только если он не может справиться с ошибкой.

Например, если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске.

- Использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода-вывода выполняется асинхронно – процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Напротив, пользовательские программы намного легче писать, если операции ввода-вывода синхронны – после команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. Компромисс заключается в том, что ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

- Разделяемость устройств. Одни устройства являются разделяемыми, а другие – выделенными (неразделяемыми). Диски – это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры – это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

· Структура подсистемы ввода-вывода

Для решения поставленных проблем программное обеспечение ввода-вывода разделяется на четыре слоя (рис.4.9):

· обработка прерываний;

· драйверы устройств;

· независимый от устройств слой операционной системы;

· пользовательский слой программного обеспечения.

Обработка прерываний. Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело. Наилучший способ состоит в разрешении процессу, инициировавшему операцию ввода-вывода, блокировать себя до завершения операции и наступления прерывания, свидетельствующего о завершении этой операции. При наступлении прерывания процедура обработки прерывания выполняет разблокирование процесса, инициировавшего операцию ввода-вывода. Эффект от прерывания будет состоять в том, что ранее заблокированный процесс теперь продолжит свое выполнение.

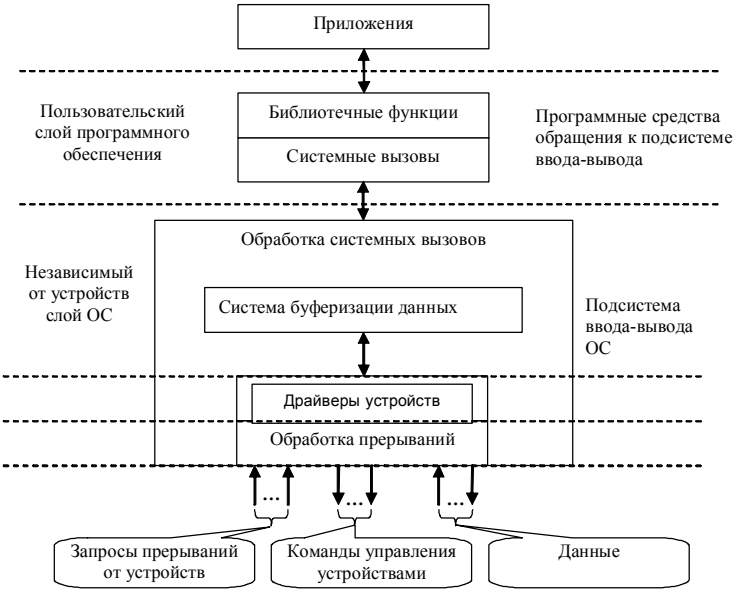


Рисунок 4.9 – Многоуровневая организация подсистемы ввода-вывода

Драйверы устройств. Весь зависимый от устройства код помещается в драйвер устройства. Каждый драйвер управляет устройствами одного типа. Только драйвер устройства знает о конкретных особенностях какого-либо устройства. Например, только драйвер диска имеет дело с дорожками, секторами, цилиндрами, временем установления головки и другими факторами, обеспечивающими правильную работу диска.

Драйвер устройства принимает запрос от устройств программного слоя и решает, как его выполнить. Типичным запросом является чтение n блоков данных. Если драйвер был свободен во время поступления запроса, то запрос начинает выполняться немедленно, иначе он ставится в очередь запросов к драйверу.

Первый шаг в реализации запроса ввода-вывода состоит в преобразовании его из абстрактной формы в конкретную, т.е. в последовательность операций контроллера. Например, для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром.

После передачи команды контроллеру драйвер должен решить, блокировать ли себя до окончания заданной операции или нет. Если операция занимает значительное время (например, при печати некоторого блока данных), то драйвер блокируется до завершения операции и обработчик прерывания не разблокирует его. Если команда ввода-вывода выполняется быстро (например, прокрутка экрана), то драйвер ожидает ее завершения без блокирования.

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

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

Остановимся на двух характерных функциях этого перечня – обеспечении независимого размера блока и распределении памяти на блок-ориентированных устройствах.

Верхние слои программного обеспечения используют понятие логического блока как единицы обмена с внешним устройством независимо от его физических характеристик, например, размера сектора. Логические и физические блоки – единицы разных уровней обмена; логический блок может состоять из нескольких физических или наоборот. Данный слой обеспечивает единый размер блока, например, за счет объединения нескольких различных блоков в единый логический блок.

При создании файла или заполнении его новыми данными необходимо выделить ему новые блоки. Для этого ОС должна вести список или битовую карту свободных блоков диска. Алгоритм поиска свободного блока может быть разработан как независимый от устройства и реализоваться программным слоем, находящимся выше слоя драйверов.

Пользовательский слой программного обеспечения

Системные процедуры. В общем случае часть программного обеспечения ввода-вывода входит в состав ОС, часть содержится в стандартных библиотеках, связываемых с пользовательскими программами.

Подсистема спулинга (*spooling*). Спулинг – это способ работы с выделенными (не разделяемыми между процессами) устройствами в мультипрограммной системе. Типичное устройство, требующее спулинга – принтер.

Если просто позволить каждому пользовательскому процессу открыть специальный файл, связанный с принтером, то принтер может быть монополизирован процессом на произвольное время, что недопустимо. Поэтому создается специальный процесс – *монитор*, получающий исключительные права на использование устройства, и специальный каталог – *каталог спулинга*.

5. АРХИТЕКТУРА ОС

5.1. Архитектура на базе ядра в привилегированном режиме

Концепция архитектуры

Наиболее общим подходом к структуризации операционной системы является разделение всех ее модулей на две группы: *ядро* и *вспомогательные модули*. Ядро выполняет все основные функции ОС и работает в особом – *привилегированном* – режиме.

Приложения выполняются независимо, каждое – в своем собственном адресном пространстве.

Преимущество такой архитектуры заключается в легкой расширяемости ОС: для добавления новой высокоуровневой функции достаточно разработать новое приложение, не касаясь ядра. В противовес этому, внесение изменений в функции ядра может оказаться достаточно сложным – вплоть до полной его перекомпиляции.

Архитектура ОС, основанная на привилегированном ядре и приложениях пользователя, считается классической. Она используется, в частности, в большинстве вариантов ОС Unix и с определенными модификациями – в ОС Windows NT.

Ядро и вспомогательные модули ОС

· ***Ядро*** включает модули, выполняющие основные функции ОС:

управление процессами;

управление памятью;

управление вводом-выводом и файловая система;

интерфейс прикладного программирования API (Application Program Interface) для поддержки обращений к ядру из приложений.

Для обеспечения высокой скорости работы ОС модули ядра (все или большая часть), являются *резидентными*, т.е. постоянно находятся в оперативной памяти.

· ***Вспомогательные модули*** *по выполняемым функциям* обычно подразделяются на следующие группы:

утилиты – программы, решающие отдельные задачи управления и сопровождения компьютерной системы (сжатие дисков, их проверка, дефрагментация; архивирование, сбор статистики и т.д.);

системные обрабатывающие программы (компиляторы, редакторы связей, загрузчики, отладчики, текстовые или графические редакторы);

библиотеки процедур различного назначения для разработки приложений (математические функции, функции ввода-вывода и т.д.);

программы, предоставляющие дополнительные услуги (калькулятор, некоторые игры).

По способу оформления эти модули представляют собой либо приложения, т.е. самостоятельные программы (утилиты, системные программы и программы дополнительных услуг), либо процедуры библиотек, вызываемые из приложений.

Вспомогательные модули ОС загружаются в оперативную память только на время выполнения (*транзитные* модули).

· Решение о том, является ли какая-либо программа частью ОС или нет, принимает производитель ОС. Так, самостоятельное приложение, имеющее спрос, может быть включено в состав ОС (например, Веб-браузер Internet Explorer), или, наоборот, модуль ОС может превратиться в отдельное приложение.

Все модули (как вспомогательные, так и пользовательские приложения) обращаются к функциям ядра посредством системных вызовов (рис. 5.1).

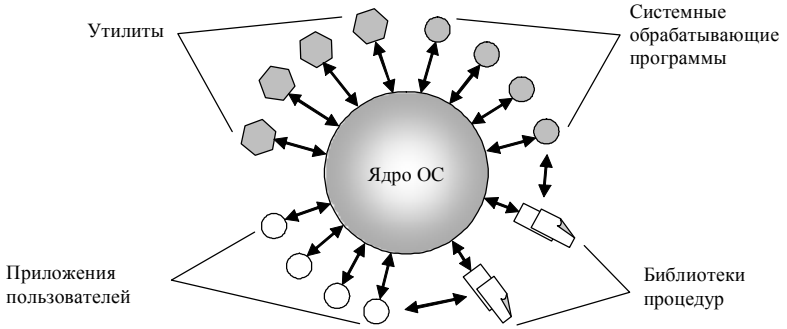


Рисунок 5.1 – Взаимодействие между ядром и другими модулями

Привилегированный режим ядра и пользовательский режим

Операционная система для осуществления своих управляющих функций должна иметь по отношению к приложениям определенные привилегии. Поэтому аппаратура компьютера поддерживает как минимум два режима:

пользовательский режим (*user mode*) – для работы приложений;

привилегированный режим, он же – *режим ядра* (*kernel mode*), или *режим супервизора* (*supervisor mode*) – для работы ОС или ее частей.

В привилегированном режиме чаще всего работает именно ядро как основная часть ОС. Понятия «ядро» и «привилегированный режим» тесно связаны, поэтому ядро также можно характеризовать как часть ОС, работающую в привилегированном режиме.

Привилегии обеспечиваются за счет запрета выполнения в пользовательском режиме некоторых критичных команд, связанных со следующими операциями:

- переключением процессора с задачи на задачу;
- управлением устройствами ввода-вывода;
- доступом к механизмам распределения и защиты памяти.

В пользовательском режиме безусловно запрещено выполнение инструкции перехода в привилегированный режим. Другие инструкции запрещается выполнять при определенных условиях, полностью контролируемых ОС. Например, ввод-вывод данных или доступ к памяти разрешены приложению, если соответствующие ресурсы выделены только этому приложению, и запрещены, если данные (соответственно память) являются общими для ОС и других приложений.

Если аппаратура (процессор) поддерживает хотя бы два уровня привилегий, то ОС может на этой основе создать программным способом сколь угодно развитую систему защиты и соответствующих прав доступа. Прямого соответствия между числом аппаратно реализуемых и программно реализуемых уровней привилегий нет. Так, на базе четырех уровней процессоров архитектуры x86 OS/2 строит трехуровневую, а Windows NT и Unix – двухуровневую систему привилегий.

Переключение процессора из пользовательского режима в привилегированный при системном вызове ядра, а затем обратное переключение повышает устойчивость ОС, но замедляет выполнение системных вызовов.

Многослойная структура ОС

· ***Многослойный подход*** – универсальный и эффективный способ декомпозиции сложных систем, базирующийся на следующих положениях.

Система представляется как иерархия слоев.

Функции нижележащего слоя являются *примитивами* для построения более сложных функций вышележащего слоя.

Взаимодействие слоев осуществляется через посредство функций *межслойного интерфейса*.

Отдельный модуль может либо выполнить свою работу самостоятельно, либо обратиться к другому модулю своего слоя, либо обратиться к нижележащему слою через межслойный интерфейс.

При таком подходе разработка системы осуществляется сверху вниз, от целей системы к их реализации. Сначала определяются функции слоев и межслойные интерфейсы, задающие общую структуру системы, а затем разрабатываются модули внутри слоев. Этот подход годится и для анализа сложных систем.

· Многослойная структура ОС

Вычислительную систему, работающую под управлением ОС на базе ядра, можно рассматривать как систему из трех иерархически упорядоченных слоев (рис. 5.2).

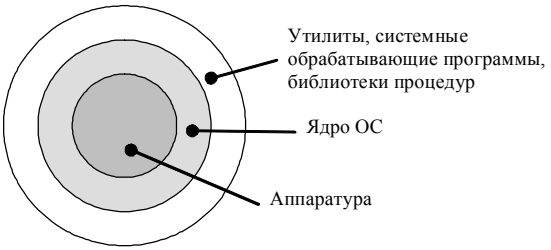


Рисунок 5.2 – Трехслойная структура вычислительной системы

При такой организации ОС приложения могут взаимодействовать с аппаратурой только через слой ядра.

· Многослойная структура ядра

Многослойный подход применим и к структуре ядра как сложного многофункционального комплекса. Обычно выделяют слои, приведенные на рис. 5.3, однако это разбиение достаточно условно.

Рисунок 5.3 – Многослойная структура ядра ОС

Средства аппаратной поддержки ОС – аппаратные средства, прямо участвующие в организации вычислительных процессов: средства поддержки привилегированного режима, система прерываний, переключение контекстов процессов, трансляция адресов, защита памяти и т.п.

Машино-зависимые модули – программные модули, в которых отображается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои от особенностей аппаратуры, т.е. позволяет делать модули вышележащих слоев машинно-независимыми (пригодными для всех типов платформ, поддерживаемых данной ОС). Примером может служить слой HAL (Hardware Abstraction Layer) в Windows NT/2000. На уровне HAL работа с устройством определенного типа (накопитель, видеоплата, мышь и т.п.) всегда описывается при помощи одного и того же заранее определенного набора функций. В случае, если устройство имеет иной набор функций (например, устаревший 3d-ускоритель может не поддерживать многих современных функций), драйвер обязан эмулировать стандартные функции с тем, чтобы ОС могла не заботиться о том, какое конкретно устройство установлено.

Базовые механизмы ядра. Модули этого слоя не принимают решений о распределении ресурсов, а только отрабатывают принятые на более высоком уровне решения. Выполняются наиболее примитивные операции ядра: программное переключение контекстов процессов, перемещение страниц между памятью и диском, диспетчеризация прерываний и т.п.

Менеджеры ресурсов. Модули этого уровня реализуют управление основными ресурсами системы. Группировка модулей в менеджеры обычно осуществляется по

функциям основных подсистем ОС: выделяются менеджеры процессов, ввода-вывода и файловой системы (могут быть объединены), оперативной памяти.

Интерфейс системных вызовов. Взаимодействует непосредственно с приложениями и систем-ными утилитами, образуя прикладной программный интерфейс ОС (API).

5.2. Микроядерная архитектура

Концепция архитектуры

В привилегированном режиме работает только небольшая часть ОС – микроядро, защищенное от остальных частей ОС приложений.

В состав функций микроядра включаются те функции ОС, которые трудно или невозможно выполнить в пространстве пользователя. В соответствии с рис. 5.3 это функции слоя базовых механизмов обычного ядра и ниже. Остальные, высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме. Соотношение классической и микроядерной архитектур приведено на рис. 5.4.

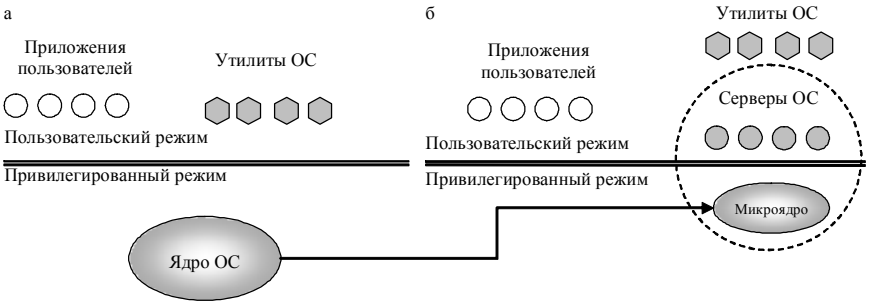


Рисунок 5.4 – Перенос функций ядра в пользовательское пространство: а – классическая архитектура, б – микроядерная архитектура

Однозначного решения о переносе в пользовательский режим тех или иных системных функций не существует. В общем случае как пользовательские приложения оформляются многие менеджеры ресурсов.

По определению, основным назначением такого приложения является обслуживание запросов других приложений (создание процесса, выделение памяти, проверка прав доступа и т.д.). Поэтому менеджеры ресурсов, вынесенные в пользовательский режим, называются *серверами* ОС. Одной из главных задач микроядра является поддержка взаимодействия серверов. Механизм обращения к функциям ОС с микроядерной архитектурой изображен на рис. 5.5.

Клиент (прикладная программа либо другой компонент ОС) посылает соответствующему серверу сообщение-запрос на выполнение некоторой функции. Непосредствен-

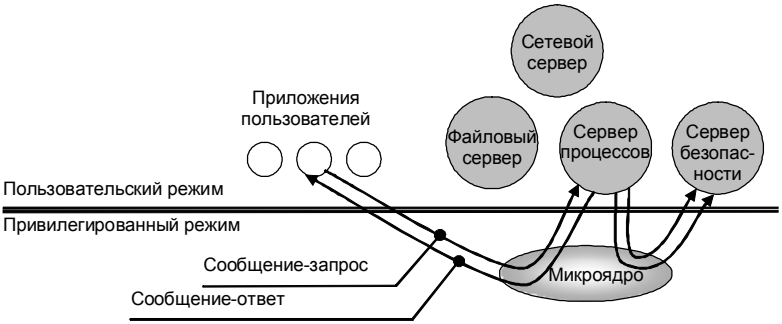


Рисунок 5.5 – Реализация системного вызова в ОС с микроядерной архитектурой

ная передача этого сообщения серверу невозможна, так как каждое приложение работает в своем адресном пространстве. В качестве посредника выступает микроядро, выполняющееся в привилегированном режиме и имеющее доступ к адресным пространствам всех приложений. Микроядро передает сообщение нужному серверу, сервер выполняет запрошенную операцию и результат, снова через посредство микроядра, возвращается клиенту с помощью другого сообщения.

Такая схема обработки запроса соответствует модели клиент-сервер, где микроядро выполняет роль транспортных средств.

Микроядерная архитектура используется, в частности, в некоторых вариантах ОС Unix и частично – в ОС Windows NT.

Преимущества и недостатки микроядерной архитектуры

ОС, основанные на концепции микроядра, в высокой степени удовлетворяют большинству требований, предъявляемых к современным ОС: обладают переносимостью, расширяемостью, надежностью, подходят для поддержки распределенных вычислений.

Основным недостатком микроядерной архитектуры является снижение производительности по сравнению с классическим вариантом. Так, при классической организации выполнение системного вызова требует двух переключений режимов «привилегированный – пользовательский», а при микроядерной – четырех (см. рис. 5.5). При обращении к часто используемым функциям работа приложений существенно замедляется. По этой причине микроядерный подход не получил широкого распространения.

Основная проблема при использовании микроядерного подхода – что включать в микроядро, а что – выносить в пользовательское пространство. В результате ОС с такой архитектурой образуют некоторый спектр, на одном краю которого находятся системы с минимально возможным микроядром, состоящим только из средств передачи сообщений, а на другом – системы, в которых микроядро выполняет достаточно большой объем функций. Примером ОС второго типа является Windows NT, где с

целью повышения производительности разработчики отклонились от микроядерной архитектуры и часто используемые функции графического интерфейса перенесли в ядро (начиная с версии 4.0) .

5.3. Переносимость ОС

Проблемы переносимости

Свойство *переносимости* (или *мобильности*) ОС состоит в возможности переноса ее кода с процессора (в общем случае аппаратной платформы) одного типа на процессор (в общем случае аппаратную платформу) другого типа. Очевидно, что чем больше платформ поддерживает ОС, тем меньше проблем создает она конечному пользователю.

В действительности понятие переносимости уточняется следующим образом: возможен ли вообще перенос кода ОС с одной платформы на другую, и если возможен, то насколько сложно это сделать. Иначе говоря, речь скорее идет о *степени* переносимости.

Проблема переносимости тесно взаимосвязана со структуризацией компонентов ОС и обеспечением максимально возможного разделения аппаратно-зависимых и аппаратно-независимых компонентов и уменьшения числа первых, т.е. с архитектурой ОС.

Объем машинно-зависимых компонентов ОС зависит от различий в аппаратных платформах, для которых разрабатывается ОС. Чем сложнее преодолимы различия (система команд процессора, разрядность процессора, количество процессоров, наличие или отсутствие аппаратной поддержки виртуальной памяти и др.), тем больше объем таких компонентов и вероятность переписывания ОС практически заново.

Для уменьшения количества машинно-зависимых модулей производители ОС обычно ограничивают универсальность машинно-независимых модулей (например, в Windows NT число типов процессоров ограничено четырьмя и для однопроцессорных и многопроцессорных платформ существуют различные коды ядра).

В итоге реально говорить не о переносимости вообще, а о переносимости в рамках нескольких аппаратных платформ.

· Принципы обеспечения переносимости

- Большая часть кода ОС должна быть написана на языке, трансляторы с которого имеются на всех машинах, куда предполагается переносить систему. Такими языками являются стандартизованные языки высокого уровня. Большинство современных переносимых ОС написано на языке Си.

Переносимость кода на ассемблере ограничена типами процессоров с одной и той же системой команд. В общем случае ассемблер используется для непереносимых частей системы, взаимодействующих с аппаратурой (обработчик прерываний) или обязанных обеспечивать максимальную скорость работы (арифметика повышенной точности).

- Объем машинно-зависимых частей кода, взаимодействующих с аппаратурой, должен быть минимизирован. Для управления аппаратурой должен быть написан набор аппаратно-зависимых функций. Тогда при переносе ОС должны быть изменены только эти функции и данные (характеристики аппаратных средств), которыми они манипулируют.

- Аппаратно-зависимый код должен быть сосредоточен (локализован) в нескольких модулях, а не распределен по всей системе.

В идеале слой машинно-зависимых компонентов должен полностью экранировать остальную часть ОС от аппаратной платформы, имитируя некую виртуальную аппаратуру, и все вышележащие слои могут быть написаны для управления этой виртуальной аппаратурой. Схема, иллюстрирующая идею переносимости, приведена на рис. 5.6.

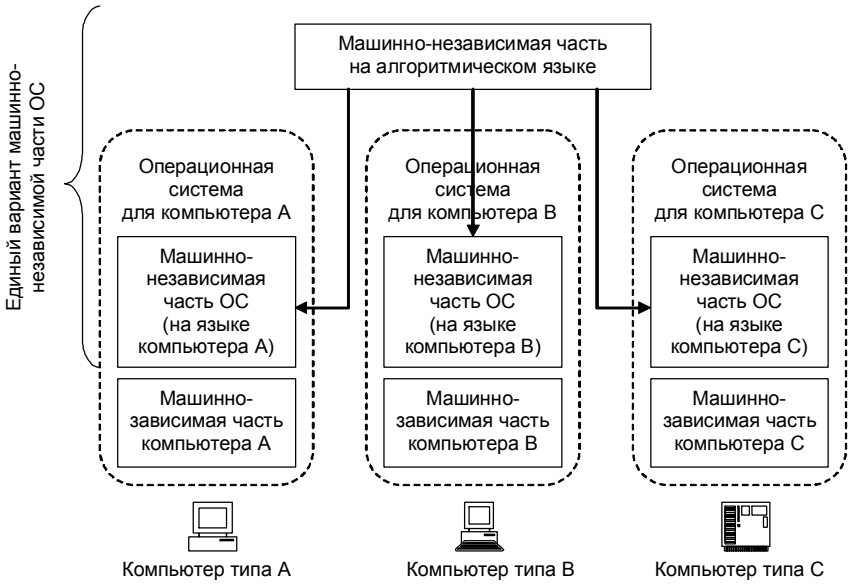


Рисунок 5.6 – Иллюстрация переносимости ОС

6. ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ И ЭВОЛЮЦИЯ ИХ ФУНКЦИОНАЛЬНЫХ ХАРАКТЕРИСТИК

6.1. Операционные системы разных этапов разработки вычислительных машин

Зарождение прообразов операционных систем в современном их толковании относят к периоду разработки в середине 1950-х годов вычислительных машин на полупроводниковой элементной базе (так называемого второго поколения ВМ) и появления первых систем **пакетной обработки информации**. Системы пакетной обработки (которые просто автоматизировали запуск одной программ за другой и тем самым увеличивали коэффициент загрузки процессора) явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило, в виде набора перфокарт, получила название *пакета заданий.*

Следующим важным этапом развития ВМ стал переход в 1960-х годах от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам, что привело к разработке более мощных вычислительных машин третьего поколения. Этот этап характеризуется созданием семейств программно-совместимых машин. Первым таким семейством была серия машин IBM/360 корпорации IBM (International Business Machines), построенных на интегральных микросхемах. Это семейство значительно превосходило ВМ второго поколения по удельному критерию производительность/стоимость. Вскоре идея программно-совместимых машин стала общепризнанной. Программная совместимость требовала и совместимости операционных систем. Такие ОС системы должны были работать как на больших, так и на малых ВМ, с различным количеством разнообразной периферии, в коммерческой области и в области научных исследований. Разработанные в то время операционные системы, в которых делалась попытка удовлетворения указанных противоречивых требований, оказывались чрезвычайно сложными. Они состояли из многих миллионов ассемблерных строк, написанных тысячами программистов, и, как правило, содержали множество ошибок. Однако, несмотря на огромные размеры и множество проблем, операционная система OS/360 и другие ей подобные ОС машин третьего поколения действительно удовлетворяли большинству требований потребителей. Важнейшим достижением ОС этого поколения явилась реализация **мультипрограммного режима**, когда на одном процессоре попеременно выполняется несколько программ. Мультипрограмм-

ный режим работы процессора значительно сократил время его простоев, которые были присущи однопрограммному режиму (при котором, например, процессор мог быть не загружен работой в ряде операций одной выполняемой программы). При мультипрограммном режиме каждая из выполняемых программ загружалась в свой участок (раздел) оперативной памяти. Другим нововведением стал спулинг (описанный выше в разделе 2), определявшийся в то время как способ организации вычислительного процесса, в соответствии с которым задания считывались с перфокарт на диск в том темпе, в котором они появлялись в очереди к устройству ввода, а затем, когда очередное задание завершалось, новое задание с диска загружалось в освободившийся раздел.

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

Следующий период в эволюции операционных систем связан с появлением больших интегральных схем (БИС). В 1980-х годах произошло резкое возрастание степени интеграции и удешевление микросхем. ВМ стала доступна отдельному человеку, и наступила эра персональных ВМ (чаще называемых более устоявшимся термином –«персональные компьютеры»). С точки зрения архитектуры персональные ВМ практически ничем не отличались от класса миниВМ, например, самых распространенных в мире на тот период времени миниВМ типа PDP-11, но их цены были существенно ниже. Если миниВМ давали возможность иметь собственную ВМ отдельному подразделению предприятия или образовательному учреждению, то персональный компьютер сделал это возможным для отдельного человека. ВМ стали широко использоваться неспециалистами, что потребовало разработки так называемого «дружественного» программного обеспечения.

На рынке операционных систем стали доминировать системы двух классов: *многопользовательские многозадачные (мультипрограммные)* ОС клона UNIX и *однопользовательские однозадачные (однопрограммные)* ОС клона MS-DOS.

История развития операционных систем клона UNIX, эволюция их функциональных характеристик, а также описание современных версий UNIX представлены далее в подразделе 6.2.

Операционные системы корпорации Microsoft под названием MS-DOS (MicroSoft Disk Operating System – «дисковая операционная система от Microsoft») были разработана для персональных компьютеров, построенных на базе микропроцессоров Intel 8088, а затем 80286, 80386 и 80486. История операционной системы MS-DOS, получившей широчайшее распространение во всем мире, начинается со скромной системы 86-DOS, написанной в 1980 г. При разработке 86-DOS были учтены требования совместимости с весьма популярной в то время системой CP/M-80 (Control Program for Microcomputers – программа управления для микрокомпьютеров), пред-

назначенной для восьмиразрядных микрокомпьютеров на базе процессоров Intel 8080 и Zylog Z-80. В июле 1981 г. корпорация Microsoft приобрела права на систему 86-DOS, существенно переработала ее и выпустила на рынок под торговой маркой MS-DOS. Когда в 1981 г., когда появились первые персональные компьютеры корпорации IBM, система MS-DOS 1.0 и ее аналог PC-DOS 1.0 (разработка IBM для персональных компьютеров – Personal Computer, PC) быстро стали основными системами для этих машин. В то же время непрерывное развитие аппаратных средств компьютеров и накопление опыта работы с ними привели к необходимости столь же непрерывного совершенствования исходных систем MS-DOS и PC-DOS. В дальнейшем они развивались параллельно и их новые версии практически во всем соответствовали друг другу. Первое серьезное усовершенствование MS-DOS (версия 2.0) было выполнено в 1983 г. Фактически была выпущена новая операционная система, хотя разработчикам удалось обеспечить полную совместимость с MS-DOS 1.0. В систему MS-DOS 2.0 были включены следующие новшества:

– поддержка дискет с повышенной плотностью записи и, главное, появившихся к этому времени жестких дисков;

– иерархическая структура каталогов (пришедшая из системы UNIX) вместе с группой команд ее поддержки;

– утилита PRINT, обеспечивающая вывод на печать в фоновом режиме с возможностью одновременного выполнения любой программы;

– атрибуты файлов и их системная поддержка;

– устанавливаемые драйверы внешних устройств;

– динамическое выделение и освобождение памяти;

– расширение возможностей командных файлов;

– большая группа новых команд, утилит и драйверов устройств, а также ряди других новшеств.

Система MS-DOS 3.0 появилась в 1984 г., одновременно с выпуском компьютеров IBM PC/AT на базе процессоров 80286. Начиная с этой версии в MS-DOS входит поддержка расширенной памяти, жестких дисков увеличенного объема, разделяемых файлов. Начиная с версии 3.1 в MS-DOS включается поддержка сетевых структур. В 1988 г. разработана версия MS-DOS 4.0, в которую была включена поддержка разделов на жестких дисках, превышающих 32 Мбайт, средства эмуляции дополнительной памяти, а также ряд новых команд. Наиболее привлекательной чертой MS-DOS 5.0 явилась возможность организации на компьютерах с расширенной памятью специальных областей, куда можно загружать устанавливаемые драйверы, резидентные программы и большую часть самой DOS. Это позволило существенно увеличить объем памяти, отводимой прикладным программам (до 600 – 610 Кбайт). Операционная система MS-DOS 6.0, выпущенная в 1993 г., вобрала в себя все лучшие качества предыдущих версий. В систему был включен целый ряд полноэкранных инструментальных утилит, охватывающих широкий диапазон потребностей пользователей персональных компьютеров. Утилиты имели развитый

интерфейс пользователя, могли управляться как от клавиатуры, так и мышью, включали контекстные справочники и элементы обучающих систем. Важнейшим усовершенствованием, введенным в версию MS-DOS 6.0, явилась возможность задания в процессе начальной загрузки альтернативных конфигураций системы (методика использования расширенной и дополнительной памяти, состав загружаемых драйверов устройств, наличие и характеристики электронных дисков и пр.).

Управление компьютером при помощи команд DOS требует определенных знаний, большой аккуратности и внимания. Для того, чтобы сделать общение с компьютером более простым, были разработаны специальные *программы-оболочки*. ***Операционная оболочка*** – это такая программа, которая позволяет пользователю осуществлять действия по управлению ресурсами компьютера в рамках более развитого (более удобного и интуитивно понятного) интерфейса, чем командная строка. Начиная с версии 4.0 в MS-DOS входила собственная псевдографическая оболочка SHELL, однако наибольшую популярность среди оболочек DOS завоевал пакет программ Norton Commander фирмы Symantec. Использование операционной оболочки Norton Commander значительно упростило управление компьютером, позволило в наглядном виде получать информацию о его основных ресурсах (и их загруженности), осуществлять все основные процедуры управления компьютером (выбор диска и каталога; создание каталога; создание, просмотр и редактирование текстовых файлов; копирование, перемещение, удаление файлов и каталогов; поиск файлов и каталогов; работа с архивными файлами и т. п.).

Также необходимо отметить, что Norton Commander явился не единственным шагом фирмы Symantec по расширению функциональных возможностей служебного и системного программного обеспечения, работающего в среде DOS. Другой ее известный продукт – Norton Utilities – объединил в себе большое количество утилит, реализующих многие важные и полезные функции, которые затруднительно или даже невозможно осуществить с помощью штатных средств операционной системы.

Следующим шагом в развитии оболочек операционных систем стало появление в 1986 г. графической многооконной операционной оболочки Windows от корпорации Microsoft. В последующие годы она претерпела ряд модификаций и в 1991 г. вышла версия Windows 3.1, а несколько позже – сетевой вариант Windows 3.11 (Windows 3.11 For WorkGroups), очень быстро завоевавшие широкое признание пользователей. Windows 3.1 запускалась на выполнение как обычная программа MS-DOS и работала на базе MS-DOS, используя на нижнем уровне внутренние функции и процедуры этой операционной системы. Приципиальным условием для программных приложений, предназначенных для работы в среде Windows, являлось то, что они должны работать с внешними устройствами (монитором, принтером, плоттером и т. п.) не напрямую, а через универсальную систему команд. Управляющая система транслировала вызовы (обращения к тому или иному физическому устройству) и передавала их соответствующему Windows-драйверу данного устройства, который непосредственно отвечал за работу с ним с учетом конкретных осо-

бенностей его функционирования. Почти все драйверы устройств Windows 3.1 фактически выполняли функции базовой системы ввода-вывода и работали с устройствами напрямую.

Основой пользовательского интерфейса Windows послужил так называемый **графический интерфейс пользователя** *GUI* (Graphical User Interface), разработанном еще в 1960-е годы Дагом Энгельбартом (Doug Engelbart) в научно-исследовательском институте Стэнфорда (Stanford Research Institute) и состоящим из окон, значков, различных меню и мыши. Идеи GUI впервые были использованы в разработках вычислительных машин Xerox PARC, а затем успешно внедрены в качестве пользовательского интерфейса в персональных компьютерах Apple компании Macintosh.

Итак, пользовательский интерфейс Windows 3.1 являлся графическим (использовался графический режим работы видеомонитора) и представлял собой иерархически организованную систему окон и других графических объектов. Интерфейс Windows в отличие от интерфейса командной строки в DOS и псевдографического интерфейса оболочки Norton Commander реализовывал оперативное управление на основе выбора того или иного графически визуализированного элемента (кнопки, пиктограммы, списка и т. п.) с помощью манипулятора мышь (команды клавиатуры, как правило, имели вспомогательное или резервное значение). Основным отличием версии Windows 3.11 была интеграция в программный пакет сетевых драйверов, что позволяло использовать эту версию для работы компьютеров в сети.

Дальнейшим развитием семейства Microsoft Windows стала разработка полноценных операционных систем Windows 95 (Windows 4.0) и Windows NT, положившим начало двух ветвей ОС от Microsoft: Windows 95/98/ME и Windows NT/2000/XP/2003. Более подробная информация об этих системах представлена далее в подразделе 6.3.

Значительную роль в развитии операционных систем играет фирма Novell. Наибольшую известность Novell приобрела благодаря своим сетевым операционным системам семейства NetWare. Эти системы реализованы как системы с выделенными серверами. Основные усилия Novell были затрачены на создание высокоэффективной серверной части сетевой ОС, которая за счет специализации на выполнении функций файл-сервера обеспечивала бы максимально возможную для данного класса ВМ скорость удаленного доступа к файлам и повышенную безопасность данных. Для серверной части своих ОС Novell разработала специализированную операционную систему, оптимизированную на файловые операции и использующую все возможности, предоставляемые процессорами Intel x386 и выше. Для рабочих станций Novell выпустила собственные ОС со встроенными сетевыми функциями: Novell DOS с входящей в нее сетевой одноранговой компонентой Personal Ware, а также ОС UnixWare, являющейся реализацией UNIX System V со встроенными возможности работы в сетях NetWare.

Первая рабочая версия NetWare была выпущена фирмой Novell в 1983 г. В 1993 году фирма Novell выпустила ОС NetWare v.4.0, явившуюся во многих отношениях революционно новым продуктом. Эта система была разработана специально для построения вычислительных сетей «масштаба предприятия» с несколькими файл-серверами, большим

количеством сетевых ресурсов и пользователей. Одним из основных нововведений явилась служба каталогов NetWare Directory Services (NDS), хранящая в распределенной по нескольким серверам базе данных информацию обо всех разделяемых сетевых ресурсах и пользователях, что обеспечило возможность при одном логическом входе в систему получать прозрачный доступ ко всем ресурсам многосерверной сети.

NetWare – это *специализированная* ОС, которая с самого начала проектировалась для оптимизации сетевого сервиса и, в первую очередь, доступа к удаленным файлам. Кроме основной цели разработки семейства ОС NetWare – повышения производительности – ставились также задачи создания открытой, расширяемой и надежной операционной системы, обеспечивающей высокий уровень защиты информации. В 1983 году фирма Novell ввела в систему концепций локальной сети понятия имени пользователя, пароля и характеристики пользователя («профиля пользователя»). Характеристика пользователя содержит перечень ресурсов, к которым пользователь имеет доступ, и права, которыми он обладает при работе с этими ресурсами. Администратор сети может ограничить права пользователя по входу в сеть датой, временем и конкретными рабочими станциями. Средства обнаружения нарушений защиты и блокировки действий нарушителя извещают администратора сети о попытках несанкционированного доступа. Помимо поддержки многопроцессорного режима, в число приоритетных направлений развития NetWare входит обеспечение процессорной независимости.

Операционная система OS/2 v.2.0 корпорации IBM была *первой работающей 32-х разрядной* ОС для персональных компьютеров. Версия OS/2 Warp, предназначенная для клиентских машин сетей типа клиент-сервер и одноранговых сетей, появилась на рынке раньше Windows 95, позиционированной аналогичным образом. OS/2 поддерживала вытесняющую многозадачность, виртуальную память и виртуальную машину для выполнения DOS-приложений, а также средства объектной ориентации. OS/2 Warp имела хорошо продуманный объектно-ориентированный графический пользовательский интерфейс, а также впервые включала набор средств поддержки сети Internet.

С интенсивным развитием компьютерных сетей в 1990-е годы появились специализированные ОС, которые предназначены исключительно для выполнения *коммуникационных задач*. Примером такой системы является ОС IOS компании Cisco Systems, которая работает в маршрутизаторах и организует выполнение в мультипрограммном режиме набора программ, реализующих коммуникационные протоколы.

Следует отметить также некоторые ОС, ориентированные на конкретную аппаратную платформу компьютеров, например, MacOS для компьютеров семейства Macintosh, PalmOS и Windows CE (Consumer Electronics – бытовая электроника) для серии *сверхминиатюрных* так называемых *«карманных»* компьютеров или PDA (Personal Digital Assistant – персональный цифровой помощник).

Среди операционных систем реального времени заслуженной популярностью пользуются ОС VxWorks и QNX.

Высоким уровнем *отказоустойчивости* при работе с сетью Internet и интрасетями характеризуются операционные системы SOLARIS фирмы Sun Microsystems. Однако в полной мере высокий уровень отказоустойчивости, обеспечиваемый благодаря сильным сторонам архитектуры ядра системы, может быть реализован только для платформы процессоров SPARC, а аппаратная поддержка платформы Intel ограничена.

6.2. История развития и характеристики операционных систем UNIX

История операционной системы UNIX началась в 1969 году с совместного проекта Массачусетского технологического института, исследовательской лаборатории Bell Labs и корпорации General Electric – системы MULTICS (Multiplexed Information and Computing Service – информационно-вычислительная служба с мультиплексированием каналов передачи данных). Ее разработчики поставили своей задачей создание большой и сложной системы общего назначения с разделением времени. Лаборатория Bell Labs вскоре вышла из числа участников проекта, после чего один из ее сотрудников, Кен Томпсон (Ken Thompson), к которому затем присоединился Денис Ритчи (Dennis Ritchie), создал усеченный вариант системы MULTICS для миникомпьютера PDP-7. Эта система в шутку была названа UNICS (UNiplexed Information and Computing Service – примитивная информационная и вычислительная служба). В дальнейшем, сохранив то же произношение, система UNICS получила несколько «сокращенное» написание в виде UNIX. UNIX привлекла внимание других специалистов, группа разработчиков увеличилась, и система была перенесена на более современные вычислительные машины PDP-11 (доминирующие в нише миниВМ в 1970-е годы), причем ее переписали на языке программирования высокого уровня C, специально созданном для этой цели Денисом Ритчи. Это событие произошло в 1973 году, когда еще не было принято писать операционные системы на высокоуровневых языках, но время для этого уже пришло, поскольку проблема переносимости системного программного обеспечения между различными аппаратными платформами стояла довольно остро. UNIX обладала огромным преимуществом перед другими ОС – она была переносимой и могла быть установлена, по крайней мере потенциально, на ВМ любой архитектуры.

Поначалу UNIX была сравнительно маленькой, очень простой в использовании и понятной системой. Первая ее версия, однопользовательская, вскоре была расширена и стала поддерживать многопользовательский доступ. Разработчики стремились максимально упростить структуру операционной системы, пусть даже за счет снижения эффективности работы и удаления некоторых функций, которыми на то время обладало большинство подобных систем. UNIX быстро приобрела популярность не только в Bell Labs, но и за ее пределами. Корпорация AT&T (учредитель лаборатории Bell Labs) стала вкладывать средства в развитие UNIX,

в результате чего было выпущено несколько новых версий ОС, архитектура которых все более усложнялась. Конечным результатом усилий корпорации AT&T стал продукт под названием System V, развитие которого затем было продолжено уже другими компаниями.

Параллельно с указанной ветвью операционной системы UNIX специалисты Калифорнийского университета в Беркли разрабатывали еще одну, названную BSD (Berkeley Software Distribution – программное изделие Калифорнийского университета). Она тоже получила широкое распространение, а впоследствии на основе обеих ветвей был создан ряд новых версий ОС.

Классической UNIX принято считать так называемую седьмую версию ОС, которая является исходной точкой двух основных ветвей развития данной архитектуры: System V и BSD.

Третья самостоятельная ветвь развития UNIX началась с попытки вернуться к исходному свойству этой операционной системы – предельной простоте. Данная ветвь начинается с микроядерной системы MINIX, за которой последовала значительно более мощная система Linux, разработанная Линусом Торвальдсом (Linus Torvalds) и представленная им в 1991 г. в качестве первой официальной версии 0.02. В настоящее время Linux – это полноценная многозадачная многопользовательская операционная система семейства UNIX. Одним из наиболее удачных вариантов реализации Linux является дистрибутивный комплект Red Hat Linux. Главными особенностями, выделяющими его на фоне других версий Linux, являются, во-первых, наличие средств управления пакетами (Red Hat Package Manager, RPM), служащих для установки программ, проверки их целостности, обновления и удаления программного обеспечения, а во-вторых, наличие графической панели управления (Control panel), с помощью которой можно осуществлять контроль практически за всеми ресурсами ВМ.

Для совместимости различных версий операционной системы UNIX и производных от нее систем Международный институт инженеров по электротехнике и электронике IEEE разработал стандарт системы UNIX, называемый POSIX, который теперь поддерживают большинство версий UNIX. Стандарт POSIX определяет минимальный интерфейс системного вызова, необходимый для совместимости UNIX-систем. Некоторые другие операционные системы в настоящее время тоже поддерживают интерфейс POSIX.

На сегодняшний день существуют версии UNIX для многих ВМ – от персональных компьютеров до суперВМ. Независимо от версии, общими для UNIX чертами являются:

– многопользовательский режим со средствами защиты данных от несанкционированного доступа,

– реализация мультипрограммного режима разделения времени, основанного на использовании алгоритмов вытесняющей многозадачности,

– использование механизмов виртуальной памяти и свопинга,

– унификация операций ввода-вывода на основе расширенного использования понятия «файл»,

– иерархическая файловая система, образующая единое дерево каталогов независимо от количества физических устройств, используемых для размещения файлов,

– переносимость системы за счет написания ее основной части на языке С,

– разнообразные средства взаимодействия процессов, в том числе и через сеть,

– кэширование диска для уменьшения среднего времени доступа к файлам.

Хотя многие пользователи UNIX, особенно опытные программисты, предпочитают командный интерфейс графическому, почти все UNIX-системы поддерживают оконную систему, созданную в Массачусетсском технологическом институте. Она называется X Windows. Эта система оперирует основными функциями окна, позволяя пользователю создавать, удалять, перемещать окна и изменять их размеры с помощью мыши. Часто поверх системы X Windows может быть установлен полный графический интерфейс, например Motif, придающий системе UNIX внешний вид системы типа Microsoft Windows или системы компьютеров Macintosh.

6.3. История развития и характеристики операционных систем семейства Windows

Особое значение в истории и сегодняшнем дне операционных систем имеет семейство продуктов Windows корпорации Microsoft как наиболее популярных ОС для персональных компьютеров и сетей на их основе.

Как уже отмечалось выше, вплоть до версий Windows 3.1/3.11 это была не операционная система в полном смысле данного понятия, а лишь графическая оболочка, надстройка над ОС MS-DOS. Полноценной ОС Windows стала начиная с четвертой версии, получившей наименование Windows 95. Ключевыми позициями, отличающими Windows 95 от Windows 3.x, явились следующие:

– новое 32-разрядное ядро;

– усовершенствованный механизм многозадачности;

– улучшенная поддержка аппаратного обеспечения;

– новые и существенно обновленные приложения.

После Windows 95 на рынке появилась ее новая редакция Windows 95 OSR2 (OEM Service Release – «сервисный выпуск для производителей компьютеров»). В эту редакцию были включены следующие чрезвычайно полезные дополнения:

– новая файловая система FAT32;

– встроенный браузер-обозреватель Internet Explorer 3.0;

– поддержка трехмерной графики;

– расширенный состав драйверов;

– повышенная стабильность работы системы.

Следующей версией стала Windows 98. Главным достоинством новой системы явилось включение более мощного браузера Internet Explorer 4.0 и обеспечение поддержки новых аппаратных устройств. Дальнейшим совершенствованием стала разработка Windows 98 SE (Second Edition), в которой была расширена поддержка ап-

паратных устройств с новыми интерфейсами USB и FireWire, включена новая версия браузера Internet Explorer 5.0, а также исправлены некоторые ошибки, добавлены новые функции и в целом существенно повышена стабильность работы.

Последней ОС в ряду Windows 9.x стала Windows ME (Millennium Edition). В нее включены новая система восстановления ОС, встроенный редактор видео, новый Internet Explorer 5.5. В Windows ME значительно обновлен графический интерфейс и упрощена работа с драйверами.

Параллельно с развитием линии ОС Windows 9.x корпорация Microsoft в 1988 году начала разработку и непрерывно продолжает развивать линию принципиально отличающихся от Windows 9.x операционных систем «новой технологии» – Windows NT (NT – New Technology). Если при создании Windows 9.x разработчики стремились сочетать новые возможности с предельной простотой установки и конфигурирования, максимальной совместимостью с имеющимся программным и аппаратным обеспечением, то серия NT прежде всего предназначалась для больших сетей и должна была обеспечить максимальную надежность и безопасность.

Команду разработчиков новой операционной системы возглавил перешедший в Microsoft из Digital Equipment Corporation Дэйв Катлер (Dave Cutler), ранее уже участвовавший в разработке нескольких ОС. К тому времени уже был накоплен немалый опыт академических исследований в области операционных систем, в том числе на основе Mach и других микроядерных архитектур. Результатом трудов команды из почти 40 разработчиков стало появление в 1993 году операционной системы, получившей название Windows NT и поддерживающей процессоры Intel x86, MIPS и Digital Alpha. Годом позже вышла версия Windows NT 3.51 с повышенной производительностью и поддержкой процессора PowerPC. За ней в 1996 последовала Windows NT 4.0.

Windows NT является 32-х разрядной ОС с приоритетной многозадачностью. В качестве фундаментальных компонентов в состав этой ОС входят развитый сетевой сервис и средства обеспечения безопасности. Windows NT обеспечивает совместимость со многими другими операционными и файловыми системами, а также с разнородными сетями, поддерживает высокопроизводительные многопроцессорные вычислительные комплексы. Windows NT не является дальнейшим развитием ранее существовавших продуктов. Ее архитектура создавалась заново с учетом современных жестких требований совместимости, переносимости, масштабируемости, распределенной обработки данных, расширяемости, надежности и отказоустойчивости операционных систем.

Несмотря на сходный интерфейс, Windows NT 4.0. имеет массу отличий от современной ей Windows 95:

– принципиально другое ядро, которое загружается самостоятельно и не имеет в своей основе MS-DOS;

– собственная файловая система NTFS, обеспечивающая разграничение доступа на уровне файлов, протоколирование файловых операций, хранение сверхбольших объемов данных, сжатие дисков и прочее;

– иной механизм многозадачности, обеспечивающий лучшую изоляцию приложений друг от друга и от ядра системы (в частности, драйверы устройств здесь не допускаются к ядру ОС, поэтому принципы написания драйверов для семейств 9.x и NT сильно отличаются);

– существенно расширенные многопользовательские возможности, чему способствует наличие серьезных средств авторизации, защиты трафика и т.п;

– высокая степень масштабируемости.

На смену версии Windows NT 4.0. пришла ОС Windows 2000 Professional (в бета-версиях она называлась Windows NT 5.0.), с которой началась новая схема именования версий системы.

Windows 2000 Professional позволяет:

1. Облегчить использование ОС корпоративными клиентами, что достигается благодаря привычному, но более простому и «интеллектуальному» интерфейсу, упрощению настройки системы путем использованию новых «программ-мастеров», ориентацией на работу с мобильными компьютерами, наличию эффективных встроенных инструментов для работы с Интернет.

2. Сохранить традиционные достоинства систем Windows NT: защищенность информации, устойчивость работы, надежность и высокую производительность.

3. Перенести в систему лучшие качества Windows 98, такие как поддержка существующих приложений и драйверов, поддержка аппаратных устройств нового поколения, встроенная сетевая поддержка для подключения к системам Windows NT Server, Novell NetWare и UNIX.

4. Создать легко конфигурируемую настольную систему с автоматизированным процессом инсталляции, развитыми средствами удаленного администрирования, установки и удаления программ, встроенной диагностики процесса загрузки.

Серверной версией Windows 2000 Professional является система Windows 2000 Server. Эта серверная платформа способна функционировать как сервер файлов, печати, приложений или Web-сервер и по сравнению с Windows 2000 Professional включает множество дополнительных специальных функций.

В конце 2001 г. корпорация Microsoft выпустила новую версию ОС – Windows XP, продолжившую линию Windows NT (внутренняя «фирменная» нумерация новой версии – Windows NT 5.1) и по сути являющейся модифицированной ОС Windows 2000 Professional. В новой ОС радикально изменен графический интерфейс, существенно повышена стабильность системы, значительно усовершенствована процедура инсталляции, усилена аппаратная и программная совместимость.

Новая *серверная версия* системы Windows 2000 Server – ОС Windows Server 2003 (внутренняя «фирменная» нумерация этой версии – Windows NT 5.2). Windows Server 2003 обладает рядом преимуществ по сравнению с Windows 2000 Server. Это самая быстрая, самая надежная и наиболее защищенная из всех серверных ОС Windows, когда-либо выпущенных Microsoft.

Windows Server 2003 предоставляет интегрированную инфраструктуру, помогающую гарантировать безопасность информации. Надежность, готовность и масш-

табируемость этой ОС позволяют реализовать сетевую инфраструктуру, соответствующую потребностям наиболее взыскательных пользователей. Windows Server 2003 предоставляет инструменты для развертывания, управления и использования сетевой инфраструктуры предприятия, средства реализации административной политики, автоматизации задач и упрощения процесса модернизации. Новые средства ОС помогают снизить расходы на сопровождение, позволяя пользователям выполнять большее число задач самостоятельно. Windows Server 2003 помогает организовать инфраструктуру прикладных задач и обеспечивает лучшее взаимодействие между системами и клиентами. Для этого ОС предоставляет интегрированные Web-сервер и сервер мультимедийных потоков, которые позволяют легко, быстро и безопасно создавать динамичные Web-сайты для интрасети и Интернета. ОС также включает интегрированный сервер приложений, обеспечивающий легкость разработки, развертывания и управления Web-сервисами.

Высокая надежность Windows Server 2003 позволяет управлять расходами, снижая время ремонтных работ и регламентных простоев. Windows Server 2003 может гибко масштабироваться вверх и вширь в зависимости от текущих потребностей. Инструменты администрирования и настройки Windows Server 2003 упрощают развертывание и управление. Обеспечивается совместимость с существующими приложениями и продуктами независимых производителей.

Резюме

Прообразом современных операционных систем являются разработанные в середине 1950-х годов системы пакетной обработки, которые просто автоматизировали запуск одной программ за другой, а последовательность подлежащих выполнению программ при этом составляла так называемый пакет заданий. Системы пакетной обработки стали первыми системными программами, предназначенными для управления вычислительным процессом.

Следующим этапом эволюции ОС стала разработка в 1960-х годах универсальных ОС, которые были способны работать на разных типах ВМ, имеющих различный набор периферийных устройств и используемых в разных областях человеческой деятельности. Попытки удовлетворения сложных и часто противоречивых требований приводили к тому, что такие ОС были чрезвычайно громоздкими и имели низкую надежность при эксплуатации. Однако, несмотря на указанные проблемы, операционная система OS/360 и другие ей подобные ОС получили широкое признание потребителей. Важнейшим достижением ОС этого периода явилась реализация мультипрограммного режима и спулинга.

Следующий период в эволюции операционных систем связан с появлением в 1980-х годах ВМ на больших интегральных схем. ВМ стали доступны отдельным небольшим организациям и учреждениям, а впервые предложенные в то время «персональные компьютеры» были доступны уже отдельному человеку. ВМ все чаще объединялись в распределенные вычислительные сети. Такие ВМ и сети на их ос-

нове стали широко использоваться неспециалистами, что потребовало разработки «дружественного» программного поддержки сетевого взаимодействия. На рынке операционных систем в то время начали доминировать системы двух классов: многопользовательские многозадачные (мультипрограммные) ОС клона UNIX и однопользовательские однозадачные (однопрограммные) ОС клона MS-DOS.

ОС UNIX получила несколько ветвей развития исходной архитектуры. Это ОС System V (корпорации AT&T) и BSD (Калифорнийского университета в Беркли). Впоследствии на основе обеих ветвей был создан ряд новых версий ОС UNIX. Третья самостоятельная ветвь развития UNIX начиналась с микроядерной системы MINIX, за которой в 1991 году последовала значительно более мощная многозадачная многопользовательская ОС LINUX.

В настоящее время существуют версии ОС UNIX для многих типов ВМ. Независимо от версии, общими для UNIX чертами являются многопользовательский режим со средствами защиты данных от несанкционированного доступа, реализация мультипрограммного режима разделения времени с вытесняющей многозадачностью, использование механизмов виртуальной памяти и свопинга, унификация операций ввода-вывода на основе расширенного использования понятия «файл», иерархическая файловая система, хорошая переносимость системы и множество других свойств.

Операционные системы корпорации Microsoft под названием MS-DOS и их аналоги других корпораций были разработаны для персональных компьютеров клона IBM PC. Управление компьютером при помощи команд DOS, вводимых в режиме командной строки, требует определенных знаний, большой аккуратности и внимания. Для того, чтобы сделать общение с компьютером более простым, были предложены так называемые программы-оболочки, представляющие собой программные надстройки операционной системы, позволяющие пользователю осуществлять действия по управлению ресурсами компьютера в рамках более развитого и удобного, чем командная строка, псевдографического интерфейса,. Следующим историческим шагом в развитии оболочек операционных систем стало появление в 1986 г. графической многооконной операционной оболочки Windows от корпорации Microsoft, которая работала на базе MS-DOS, а основой пользовательского интерфейса Windows послужил *графический интерфейс пользователя GUI,* представляющий собой в данном случае иерархически организованную систему окон и других графических объектов. Дальнейшим развитием семейства Microsoft Windows стала разработка полноценных операционных систем Windows 95 (Windows 4.0) и Windows NT, положившим начало двух ветвей ОС от Microsoft: Windows 95/98/ME и Windows NT/2000/XP/2003.

Полноценной ОС Windows стала начиная с четвертой версии, получившей наименование Windows 95. Ключевыми позициями, отличающими Windows 95 от Windows 3.x, явились новое 32-разрядное ядро, усовершенствованный механизм многозадачности, улучшенная поддержка аппаратного обеспечения, новые и существенно обновленные приложения. Следующими версиями стали Windows 98, Windows 98 SE и Windows ME. Главными достоинствами новых систем явилось включение более мощ-

ных версий браузера Internet Explorer, расширение поддержки новых аппаратных устройств с новыми интерфейсами.

Параллельно с развитием линии ОС Windows 9.x корпорация Microsoft в 1988 году начала разработку и непрерывно продолжает развивать линию принципиально отличающихся от Windows 9.x операционных систем «новой технологии» – Windows NT (NT – New Technology). Windows NT является 32-х разрядной ОС с приоритетной многозадачностью. В качестве фундаментальных компонентов в состав этой ОС входят развитый сетевой сервис и средства обеспечения безопасности. Windows NT совместима со многими другими операционными и файловыми системами, а также с разнородными сетями, поддерживает работу высокопроизводительных многопроцессорных вычислительных комплексов.

На смену версии Windows NT 4.0. пришла существенно усовершенствованная и усиленная ОС Windows 2000 Professional. Серверной версией Windows 2000 Professional является система Windows 2000 Server, включающаяет множество дополнительных специальных функций.

В конце 2001 г. корпорация Microsoft выпустила новую версию ОС – Windows XP, продолжившую линию Windows NT и по сути являющейся модифицированной ОС Windows 2000 Professional. Новая серверная версия системы Windows Server 2003 обладает рядом преимуществ по сравнению с Windows 2000 Server. Это самая быстрая, самая надежная и наиболее защищенная из всех серверных ОС Windows, выпущенных Microsoft до настоящего времени.

Значительную роль в развитии ОС играет фирма Novell со своим сетевым операционными системами семейства NetWare.

Историческое значение имеет ОС OS/2 корпорации IBM, которая появилась на рынке раньше Windows 95 и была первой работающей 32-х разрядной ОС для персональных компьютеров, а кроме того поддерживала вытесняющую многозадачность, виртуальную память и имела средства работы в сети Internet.

Следует отметить также некоторые специализированные ОС. Например, IOS компании Cisco Systems предназначена исключительно для выполнения коммуникационных задач, MacOS ориентирована на конкретную аппаратную платформу компьютеров семейства Macintosh, PalmOS и Windows CE работают в сверхминиатюрных так называемых «карманных» компьютерах.

Отметим также ОС Solaris фирмы Sun Microsystems, которая благодаря высокому уровню отказоустойчивости предпочтительна для использования в интер- и интрасетях.

ЗАКЛЮЧЕНИЕ

Основными функциями операционных систем являются функции управления процессами и ресурсами вычислительных машин и систем. Процессы и ресурсы – это ключевые понятия операционных систем. За время своего существования процессы могут неоднократно изменять свое состояние, проходя через стадии создания, готовности, выполнения, ожидания и завершения своей работы. Операционная система организует планирование выполнения процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие процессов и решает проблемы их синхронизации.

При управлении таким важнейшим ресурсом, как память вычислительной машины, операционная система контролирует наличие свободной и занятой памяти, выделяет память для выполнения процессов и освобождает память после завершении процессов, реализует вытеснение процессов из оперативной памяти на дисковую память и возвращение их обратно в оперативную память, обеспечивает настройку адресов программы на конкретную область физической памяти.

Управляя устройствами ввода-вывода, операционная система передает устройствам соответствующие команды, перехватывает прерывания, обрабатывает ошибки, обеспечивает интерфейс между устройствами ввода-вывода и остальной частью машины. Для освобождения процессора от операций последовательного вывода данных из оперативной памяти или последовательного ввода в нее используется механизм прямого доступа внешних устройств к памяти.

Возможность управления файлами со стороны операционной системы на логическом уровне структур данных и операций обеспечивают различные типы файловых систем. Файловая система представляет собой набор спецификаций и соответствующее им программное обеспечение, которые отвечают за создание, уничтожение, организацию, чтение, запись, модификацию и перемещение файловой информации, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами.

В операционных системах для многопроцессорных вычислительных машин сложность планирования процессов существенно возрастает, так как требуется не только решение вопросов очередности запуска процессов на выполнение, но и выбор центрального процессора, на котором должен быть запущен тот или иной процесс.

В многомашинных вычислительных системах механизмы организации межпроцессной взаимосвязи принципиально отличаются от организации такой взаимосвязи в автономных вычислительных машинах. Базой для взаимодействия процессов в автономных машинах служит общая разделяемая память. Многомашинные вычислительные системы по определению не имеют общей разделяемой памяти, и

поэтому основой межпроцессной взаимосвязи в них служит обмен физическими сообщениями посредством коммуникационной среды.

Операционные системы многомашинных вычислительных систем распределенного типа – вычислительных сетей – обычно называют сетевыми. Фундаментальным понятием сетевых операционных систем, позволяющим определить и реализовать взаимодействие удаленных процессов, является понятие сетевого протокола.

Наиболее совершенным и перспективным классом операционных систем являются так называемые распределенные операционные системы. Распределенная система создает для пользователя полную иллюзию того, что он работает в обычной автономной системе. Эффективным способом построения распределенных операционных систем является установка специального промежуточного уровня программного обеспечения поверх сетевой операционной системы, который предоставляет приложениям определенную однородность взаимодействия. Среди различных типов промежуточного программного обеспечения следует выделить документное, файловое, объектное и координационное.

Основными принципами построения современных операционных систем являются принципы модульности, генерируемости, функциональной избыточности, виртуализации, совместимости с другими системами, открытости, легкой наращиваемости, мобильности, обеспечения надежной безопасности. Операционные системы прошли длительный путь развития и совершенствования своей архитектуры от монолитных систем до хорошо структурированных модульных систем, способных к развитию, расширению и легкому переносу на новые платформы. При модульном построении в системе выделяется некоторая часть важных программных модулей, которые для более эффективной организации вычислительного процесса должны постоянно находиться в оперативной памяти. Эту группу модулей называют ядром операционной системы. Другие системные программные модули (транзитные или диск-резидентные) загружаются в оперативную память только при необходимости, а в случае отсутствия свободного пространства могут быть замещены другими транзитными модулями. Для использования прикладными программами системных ресурсов операционной системы и реализуемых ею функций предназначен интерфейс прикладного программирования, который может быть реализован как на уровне операционной системы, так и на уровне системы программирования или на уровне внешней библиотеки процедур и функций.

Прообразом современных операционных систем являются разработанные в середине 1950-х годов системы пакетной обработки, которые были первыми системными программами, предназначенными для управления вычислительным процессом. Следующим этапом эволюции операционных систем стала разработка в 1960-х годах универсальных систем, которые были способны работать на разных типах вычислительных машин, имеющих различный набор периферийных устройств и используемых в разных областях человеческой деятельности. Существенным достижением систем этого периода явилась реализация многозадачного режима и спулинга.

Важнейшей вехой в истории и современном состоянии операциионных систем является семейство многопользовательских многозадачных систем UNIX. UNIX получила несколько ветвей развития исходной архитектуры. Двумя главными из таких ветвей стали System V (корпорации AT&T) и BSD (Калифорнийского университета в Беркли). Впоследствии на основе обеих этих ветвей был создан ряд новых версий ОС UNIX. Третья самостоятельная ветвь развития UNIX начиналась с относительно простой «учебной» системы MINIX, за которой в 1991 году последовала значительно более мощная многозадачная многопользовательская ОС LINUX. Операционные системы типа UNIX широко используется на вычислительных машинах различных классов от ноутбуков до суперкомпьютеров.

Для персональных компьютеров клона IBM PC были разработаны однопользовательские однозадачные операционные системы типа MS-DOS корпорации Microsoft и их аналоги других корпораций. Управление компьютером в среде MS-DOS осуществлялось в режиме командной строки. Для того, чтобы сделать общение с компьютером более простым, были предложены так называемые программы-оболочки, представляющие собой программные надстройки операционной системы, позволяющие пользователю осуществлять действия по управлению ресурсами компьютера в рамках более развитого и удобного, чем командная строка, псевдографического интерфейса. Следующим историческим шагом в развитии оболочек операционных систем стало появление в 1986 г. графической многооконной операционной оболочки Windows от корпорации Microsoft, которая работала на базе MS-DOS, а основой пользовательского интерфейса Windows послужил так называемый *графический интерфейс пользователя GUI,* представляющий собой иерархически организованную систему окон и других графических объектов. Дальнейшим развитием семейства Microsoft Windows стала разработка полноценных операционных систем Windows 95 (Windows 4.0) и Windows NT, положившим начало двух ветвей ОС от Microsoft: Windows 95/98/ME и Windows NT/2000/XP/2003.

Линия систем «новой технологии» Windows NT принципиально отличается от линии Windows 9.x. В качестве фундаментальных компонентов в состав семейства Windows NT входят развитый сетевой сервис, поддержка работы высокопроизводительных многопроцессорных вычислительных комплексов, средства обеспечения эффективной безопасности. На смену версии Windows NT 4.0. в 2000 году пришла существенно усовершенствованная и усиленная ОС Windows 2000 Professional. Серверной версией Windows 2000 Professional является система Windows 2000 Server, включающаяет множество дополнительных специальных функций. В конце 2001 г. корпорация Microsoft выпустила новую версию – Windows XP, продолжившую линию Windows NT. Новая серверная версия Windows Server 2003 обладает рядом преимуществ по сравнению с Windows 2000 Server.

Значительную роль в развитии ОС играет фирма Novell со своим сетевым операционными системами семейства NetWare. Историческое значение имеет система OS/2 корпорации IBM, которая появилась на рынке раньше Windows 95 и была первой 32-х разрядной операционной системой для персональных компьютеров.

Следует отметить также некоторые специализированные системы, например, предназначенные исключительно для выполнения коммуникационных задач или ориентированные на определенную аппаратную платформу компьютеров.

Любая из современных операционных систем имеет свои особенности построения и практической реализации, применяет те или иные способы и механизмы управления процессами и ресурсами, использует различные по степени универсальности и совместимости прикладные интерфейсы, обладает разным уровнем функциональности и может позиционироваться для определенных сфер применения. Для каждой из систем можно указать ее преимущества и недостатки, сильные и слабые стороны. Естественно, что предприятию или отдельному пользователю хотелось бы работать с оптимальной операционной системой, удовлетворяющей комплексу наиболее важных требований. «Идеальная» операционная система скорее всего должна иметь такую же степень интеграции и такую же поддержку, как Microsoft Windows 2000/XP/Server 2003, такую же превосходную отказоустойчивость, как Solaris 8 компании Sun Microsystems, такую же службу справочника, какой снабжена Novell NetWare 5.1, а также универсальность и гибкость, свойственные системе Linux.

Подобно другим программным продуктам информационных технологий, операционные системы постоянно совершенствуются. Основное внимание при разработке новых версий операционных систем уделяется базовым службам (файловые службы, службы печати, защиты, аутентификации, службы справочника), средствам управления, масштабируемости, применимости, надежности, службам Интернет, интрасетей и электронной коммерции. Та компания, которая в своей версии операционной системы лучше других обеспечивает эти качества, становится лидером продаж на рынке. Конкурентная борьба, как хорошо известно, является лучшим двигателем прогресса, в том числе и в области операционных систем.

Каждая из компаний – игроков на рынке операционных систем – имеет собственные планы дальнейшего развития своих продуктов.

Например, в ближайших (на момент подготовки данного учебного пособия) планах корпорации Microsoft выпуск новых версий Windows XP. К ним относятся Windows XP Media Center Edition (MCE) 2004, включающая несколько новых специализированных модулей и призванная превратить персональный компьютер в полнофункциональный развлекательный центр, а также Windows XP 64-bit Edition, являющаяся модификацией XP для компьютеров с 64-разрядными процессорами. Microsoft ведет разработку операционной системы нового поколения, именуемой как Windows Longhorn. По заявлениям Microsoft эта система станет революционной версией, основанной на внедрении новых технологий. Планируется выпуск ее 32- и 64-разрядных модификаций.

Можно надеяться, что не остановятся на достигнутом и другие компании – разработчики операционных систем, поэтому пользователей ждут еще более совершенные, функциональные, производительные и комфортные среды взаимодействия с вычислительной техникой.

ЛИТЕРАТУРА

1. *Андреев А. Г.* и др. Microsoft Windows 2000 Server и Professio-nal / Под общ. ред. *А.Н. Чекмарева и Д.Б. Вишнякова.* – СПб.: БХВ – Петербург, 2001. – 1056 с.: ил.

2. *Андреев А. Г.* и др. Microsoft Windows XP. Руководство администратора/ Под общ. ред. *А. Н. Чекмарева*. – СПб.: БХВ – Петербург, 2003. – 848 с.: ил.

3. *Бэкон Д., Харрис Т.* Операционные системы. – СПб.: Питер, 2004. – 800 с.: ил.

4. *Вишневский А. В*. Windows Server 2003. Для профессионалов. – СПб.: Питер, 2004. – 767 с.: ил.

5. *Гордеев А.В.* Операционные системы. – СПб.: Питер, 2005. – 418 с.: ил.

6. *Гордеев А. В., Молчанов А. Ю.* Системное программное обеспечение. – СПб.: Питер, 2001. – 736 с.: ил.

7. *Назаров С. В.* Администрирование локальных сетей Windows NT/2000/NET: Учеб. пособие. – М.: Финансы и статистика, 2003. – 478 с.: ил.

8. *Новиков Ю., Черепанов А.* Персональные компьютеры: аппаратура, системы, Интернет: Учебный курс. – СПб.: Питер, 2001.– 464 с.: ил.

9. *Олаф Кирх.* Linux: Руководство администратора сети. – СПб.: Питер, 2000. – 242 с.: ил.

10. *Олифер В.Г., Олифер Н. А.* Сетевые операционные системы. – СПб.: Питер, 2001. – 544 с.: ил.

11. Основы операционных систем: Курс лекций. / *В. Е. Карпов, К. А. Коньков.* – М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2004. – 632 с.: ил.

12. *Партыка Т. Л., Попов И. И.* Операционные системы, среды и оболочки. – М.: ФОРУМ – ИНФРА-М, 2005. – 400 с.: ил.

13. *Таненбаум Э.* Современные операционные системы. – СПб.: Питер, 2004. – 1040 с.: ил.

14. *Таненбаум Э., М. ван Стеен*. Распределенные системы. Принципы и парадигмы. – СПб.: Питер, 2003. – 877 с.: ил.

15. *Ханикат Дж.* Знакомство с Microsoft Windows Server 2003: Пер. с англ. – М.: Издательско-торговый дом «Русская Редакция», 2003. – 464 с.: ил.

16. *Чекмарев А. Н., Вишневский А. В., Кокорева О. И*. Microsoft Windows Server 2003. – СПб.: БХВ – Петербург, 2003. – 1184 с.: ил.

СЛОВАРЬ ТЕРМИНОВ И ОПРЕДЕЛЕНИЙ

Виртуальная память

совокупность программно-аппаратных средств, позволяющих пользователям писать программы, которые для своей реализации требуют объемы памяти, превосходящие реально существующие объемы оперативной памяти вычислительной машины.

Гонка

ситуация, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей выполнения процессов.

Критическая секция (критическая область)

часть программы, в которой осуществляется доступ к разделяемым данным.

Кэширование информации

способ организации совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования наиболее часто используемой информации из относительно более «медленного» запоминающего устройства в более «быстрое».

Логическая запись

наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством.

Логическая организация файла

представление файла в виде определенным образом организованных логических записей.

Многозадачность невытесняющая

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

Многозадачность вытесняющая

способ планирования процессов, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Модуль операционной системы

функционально законченный элемент системы, выполненный в соответствии с принятыми межмодульными интерфейсами.

Операционная оболочка

программа, которая позволяет пользователю осуществлять действия по управлению ресурсами компьютера в рамках более развитого, удобного и интуитивно понятного интерфейса, чем командная строка.

Операционная система

комплекс управляющих и обрабатывающих программ, который, с одной стороны, выступает как интерфейс между пользователем (с его задачами) и аппаратными компонентами вычислительных машин и вычислительных систем, а с другой стороны предназначен для эффективного управления вычислительными процессами, а также наиболее рационального распределения и использования вычислительных ресурсов машин и систем.

Операционная система сетевая

операционная система, позволяющая реализовать обмен сообщениями между отдельными компонентами, которые входят в состав вычислительной сети.

Операционная среда

программная среда, которую образует операционная система и в которой выполняются прикладные программы пользователей.

Планирование процессов

распределение процессов между имеющимися ресурсами.

Планировщик

программа, управляющая миграцией процессов между различными очередями при их прохождении через вычислительную машину.

Подсистема буферизации

буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом.

Подсистема управления процессами

часть операционной системы, которая планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Правила синхронизации
определяют порядок взаимосвязи процессов.

Прерывание
принудительная передача управления от выполняемой программы к операционной системе (а через нее – к соответствующей программе обработки прерывания), происходящая при возникновении определенного события, механизм, позволяющий координировать параллельное функционирование отдельных устройств вычислительной машины и реагировать на особые состояния, возникающие при работе процессора.

Приоритет
характеризует степень привилегированности процесса при использовании ресурсов вычислительной машины.

Процесс
последовательность операций при выполнении программы или ее части в совокупности с используемыми данными.

Реентерабельность
свойство программы, позволяющее одновременно выполнять эту программу нескольким процессам.

Ресурс
любой потребляемый (расходуемый) объект вычислительной машины или системы, который может быть выделен его потребителю – процессу – на определенный интервал времени.

Сетевой протокол
набор специализированных правил, описывающих и регламентирующих типы и форматы сообщений, с помощью которых могут взаимодействовать отдельные компоненты вычислительной сети.

Супервизор
программа, обеспечивающая оптимальное использование ресурсов вычислительной машины в режиме многозадачности.

Таблица управления процессом
содержит набор значений и параметров, которые характеризуют текущее состояние процесса и используются операционной системой для управления прохождением процесса через вычислительную машину.

Транзитные программные модули операционной системы

часть программных модулей операционной системы, которые загружаются в оперативную память только при необходимости, а в случае отсутствия свободного пространства могут быть замещены другими транзитными модулями.

Файл

набор данных, организованных в виде совокупности записей определенной структуры.

Файловая система

набор спецификаций и соответствующее им программное обеспечение, которые отвечают за создание, уничтожение, организацию, чтение, запись, модификацию и перемещение файловой информации, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами.

Физическая организация файла

описывает правила расположения файла на устройстве внешней памяти, в частности, на диске.

Ядро операционной системы

часть наиболее важных программных модулей операционной системы, которые постоянно находятся в оперативной памяти с целью эффективной организации вычислительного процесса.

Для заметок

Для заметок

Для заметок

Учебное издание

Марат Рашидович Бибарсов,
Гульнара Шихмуратовна Бибарсова,
Юрий Владимирович Кузьминов

ОПЕРАЦИОННЫЕ СИСТЕМЫ, СРЕДЫ И ОБОЛОЧКИ

Учебное пособие

Редактор Т.Б. Кузнецова,
Компьютерная верстка П.Г. Немашкалов

Подписано в печать 1.12.10
Формат 60x84 $^{1}/_{16}$ Усл.печ.л. 6,98 Уч.-изд.л. 5,25
Бумага офсетная Тираж 100 экз. Заказ 80

Отпечатано в ООО «РБК-Сервис».