Томский межвузовский центр
дистанционного образования

Ю.Б. Гриценко

ОПЕРАЦИОННЫЕ СРЕДЫ, СИСТЕМЫ И ОБОЛОЧКИ

Учебное пособие

ТОМСК – 2005

Корректор: Осипова Е.А.

Гриценко Ю.Б.
Операционные среды, системы и оболочки: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2005. – 281 с.

Рассмотрены вопросы организации и построения операционных сред и систем. Основное внимание уделено понятиям вычислительного процесса, управлению задачами и ресурсами операционных систем. Рассмотрены также примеры реальных операционных систем: ОС Windows (Microsoft), ОС OS/2 (IBM), QNX (QNX Software Systems Limited), Unix, Linux и их оболочек.

Предназначено для студентов специальности 010502 (351400) – «Прикладная информатика (в экономике)».

СОДЕРЖАНИЕ

Введение 7

Часть 1. СТРУКТУРА И ПРИНЦИПЫ РАБОТЫ ОПЕРАЦИОННЫХ СИСТЕМ 10

1. Основные понятия и концепции построения операционных сред и систем 10

1.1 Классификация программного обеспечения 10
1.2 Вычислительный процесс. Ресурс 16
1.3 Потоки (треды) 26
1.4 Прерывания 32
1.5 Классификация операционных систем 39
Вопросы для самопроверки 43

2. Управление задачами 44

2.1 Основные функции управления задачами 44
2.2 Планирование процессов и диспетчеризация задач 46
2.2.1 Дисциплины диспетчеризации 46
2.2.2 Алгоритмы диспетчеризации 57
2.2.3 Качество диспетчеризации 60
Вопросы для самопроверки 63

3. Управление памятью 64

3.1 Основные понятия 64
3.1.1 Фон-неймановская архитектура вычислительных машин 64
3.1.2 Биты, байты, слова, параграфы 65
3.1.3 Ячейки памяти, порты и регистры 67
3.1.4 Адресация ячеек памяти в реальном режиме 69
3.1.5 Подсистемы памяти и хранения данных 69
3.1.6 Стек 70
3.2 Распределение оперативной памяти 71
3.2.1 Распределение оперативной памяти в MS DOS 71
3.2.2 Распределение оперативной памяти в Microsoft Windows 85
3.3 Организация режима защиты 95
3.3.1 Переключение задач и виртуальные машины 95
3.3.2 Защищенный режим и виртуальная память 96

3.3.3 Организация и адресация памяти в защищенном режиме ... 100
3.3.4 Кэширование памяти ... 102
Вопросы для самопроверки ... 106
4. Управление внешней памятью и файловые системы ... 107
4.1 Характеристика устройств внешней памяти ... 107
4.1.1 Общие свойства устройств внешней памяти ... 107
4.1.2 Основные характеристики устройств внешней памяти ... 108
4.1.3 Характеристики накопителей на жестких магнитных дисках ... 112
4.2 Структура магнитного диска ... 117
4.2.1 Физическая структура ... 117
4.2.2 Логическая структура ... 118
4.3 Файловые системы ... 124
4.3.1 Функции файловой системы ОС ... 124
4.3.2 Файловая система FAT ... 127
4.3.3 Файловая система NTFS ... 133
4.3.4 Файловая система HPFS ... 142
4.3.5 Файловая система ОС UNIX ... 151
4.3.6 Файловые системы для CD-ROM ... 157
Вопросы для самопроверки ... 158
5. Архитектуры операционных систем и интерфейсы прикладного программирования ... 159
5.1 Основные принципы построения операционных систем ... 159
5.2 Микроядерные операционные системы ... 170
5.3 Монолитные операционные системы ... 173
5.4 Принципы построения интерфейсов операционных систем ... 174
5.4.1 Интерфейс прикладного программирования ... 177
5.4.2 Функции API на различных уровнях реализации ... 179
5.4.3 Платформенно-независимый интерфейс POSIX ... 186
Вопросы для самопроверки ... 188
Часть 2. РЕАЛЬНЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ ... 190
6. Операционные системы фирмы Microsoft ... 190
6.1 Операционная система MS DOS ... 190
6.1.1 История ОС MS DOS ... 190

6.1.2 Основные части MS DOS ... 191
6.1.3 Последовательность загрузки MS DOS ... 193
6.1.4 Файл конфигурации MSDOS CONFIG.SYS ... 194
6.1.5 Работа интерпретатора команд COMMAND.COM ... 195
6.1.6 Командный файл автозапуска AUTOEXEC.BAT ... 196
6.1.7 Командный язык MS DOS и файлы пакетной обработки ... 198
6.2 Операционная система Windows 95 ... 202
6.3 Операционная система Windows 98 ... 207
6.4 Windows Millennium Edition ... 209
6.5 Платформа Windows NT ... 212
6.5.1 Общие сведения ... 212
6.5.2 Windows NT Server 4.0 ... 212
6.5.3 Windows NT Workstation 4.0 ... 214
6.6 Платформа Windows 2000 ... 215
6.6.1 Windows 2000 Server ... 215
6.6.2 Windows 2000 Professional ... 217
6.7 Операционная система Windows XP ... 219
6.8 Windows 2003 Server ... 222
Вопросы для самопроверки ... 226

7. Семейство операционных систем OS/2 Warp ... 228
7.1 Общее представление OS/2 Warp ... 228
7.2 Особенности архитектуры OS/2 Warp ... 231
7.3 Особенности интерфейса OS/2 Warp ... 233
7.4 Серверная операционная система OS/2 Warp 4.5 ... 235
Вопросы для самопроверки ... 237

8. Операционные системы семейства UNIX ... 238
8.1 Общее представление семейства ОС UNIX ... 238
8.2 Основные понятия семейства ОС UNIX ... 239
8.3 Выполнение процессов в ОС UNIX ... 244
8.4 Межпроцессные коммуникации в UNIX ... 247
8.5 Операционная система Linux ... 251
Вопросы для самопроверки ... 253

9. Операционные системы реального времени. Операционная система QNX ... 255
9.1 Общее представление ОС реального времени QNX ... 255
9.2 Особенности архитектуры системы QNX ... 257

9.3 Основные механизмы QNX 262
Вопросы для самопроверки 265
Список сокращений 266
Литература 271
Контрольные работы 273

ВВЕДЕНИЕ

Современное общество живет в век информации. Умение качественно управлять информационными ресурсами одно из важнейших направлений деятельности человека. В настоящий момент идет бурное развитие автоматизированных систем управления. Развивается как аппаратное, так и программное обеспечение (ПО). Изучение дисциплины «Операционные среды, системы и оболочки» представляет собой основу для изучения всего класса ПО.

В содержание дисциплины входит изучение как теоретического материала: структур, методов и алгоритмов построения современных операционных сред и систем (ОС), так и возможностей функционирования современных реальных ОС. Базовыми категориями в освоении данного курса являются основные понятия и концепции: построения ОС (операционная среда, вычислительный процесс, ресурс, поток, прерывание), управление задачами (функции, стратегии планирования, дисциплины и алгоритмы диспетчеризации), управление внутренней и внешней памятью, архитектура ОС и интерфейс прикладного программирования.

Развитие принципов построения ОС тесно связано с развитием средств вычислительной техники. Современная архитектура IBM PC-совместимого компьютера представляет собой реализацию так называемой фон-неймановской архитектуры вычислительных машин. Эта архитектура была представлена Джоном фон Нейманом в 1945 году. Фон-неймановская архитектура – не единственный вариант построения ЭВМ, имеются и другие, которые не соответствуют указанным принципам (например, потоковые машины). Однако подавляющее большинство современных компьютеров основаны именно на указанных принципах, включая и сложные многопроцессорные комплексы, которые можно рассматривать как объединение фон-неймановских машин. Теория фон Неймана явилась основой для построения первых ОС. Значительная часть теорий построения ОС была разработана в 70–80-х годах прошлого века. Данная область знаний на протяжении 90-х годов развивалась слабо, и только в последние годы

производители системного ПО осознали потребность пересмотра функциональности ОС.

Учебное пособие по курсу «Операционные среды, системы и оболочки» представлено в двух частях. Часть первая состоит из пяти разделов, в которых отражены теоретические моменты построения операционных сред, систем и оболочек. Во второй части учебного пособия (разделы 6–9) изложены примеры реальных ОС и их оболочек.

В первом разделе вводится классификация ПО, дается определение понятиям операционной системы и операционной среды, рассматриваются понятия вычислительного процесса, ресурса, потока, механизма обработки прерываний и приводится классификация ОС.

Второй раздел содержит описание функций управления задачами в ОС, стратегий планирования задачами, дисциплин и алгоритмов диспетчеризации, методов оценки качества диспетчеризации.

Третий раздел включает: описание фон-неймановской архитектуры вычислительных машин; вопросы организации и распределения оперативной памяти; методы переключения задач и построения виртуальных машин; способы организации и переключения памяти в реальном и защищенном режимах; вопросы распределения оперативной памяти; описание процесса кэширования памяти.

Процесс управления внешней памятью и описание файловых систем приведен в четвертом разделе. В нем рассмотрены общие свойства управления внешней памятью, основные характеристики устройств внешней памяти, функции файловой системы, структура магнитного диска, описание файловых систем: FAT (File Allocation Table), FAT32, NTFS (New Technology File System), HPFS (High Performance File System), файловая система ОС UNIX, файловых систем для постоянного запоминающего устройства на основе компакт-диска CD-ROM (Compact Disk Read Only Memory) – CDFS (Compact Disk File System), UDF (Universal Disk Format).

Пятый раздел содержит описание основных принципов построения операционных систем и интерфейса прикладного программирования, концепции построения микроядерных и моно-

литных ОС, описание платформенно-независимого интерфейса для компьютерного окружения POSIX (Portable Operating System Interface for Computer Environments).

В шестом разделе приведено описание операционных систем фирмы Microsoft, начиная с прародительницы ОС класса Windows – ОС MS DOS (Microsoft Disk Operation System) и заканчивая последней на момент написания учебного пособия ОС Windows 2003 Server.

В седьмом разделе рассматриваются особенности построения архитектуры и интерфейса ОС фирмы IBM – OS/2 Warp, в частности возможности серверной операционной системы OS/2 Warp 4.5.

Общие представления и основные понятия ОС Unix приводятся в восьмом разделе. В ней более детально рассмотрены вопросы выполнения процессов и межпроцессорные коммуникации в среде Unix; функциональные возможности Unix-подобной ОС – Linux.

В девятом разделе дано общее представление об операционных системах реального времени, в частности об QNX (Queue Nicks), разработанной фирмой QNX SoftWare System. Рассмотрены особенности архитектуры и основные механизмы ОС QNX для организации распределенных вычислений.

Часть 1. СТРУКТУРА И ПРИНЦИПЫ РАБОТЫ ОПЕРАЦИОННЫХ СИСТЕМ

1. ОСНОВНЫЕ ПОНЯТИЯ И КОНЦЕПЦИИ ПОСТРОЕНИЯ ОПЕРАЦИОННЫХ СРЕД И СИСТЕМ

1.1 Классификация программного обеспечения

Программное обеспечение (ПО) – неотъемлемая составляющая любой ЭВМ, без которой невозможно получить необходимые результаты всевозможных вычислительных операций. При всем многообразии и сложности современных программных систем при их разработке в качестве базовой основы используются уже существующие фундаментальные концепции, имеющие много общего в части принципов построения и отличающиеся некоторыми особенностями реализации.

В работах специалистов по рассматриваемой тематике предлагается множество неоднозначных классификаций программного обеспечения, в частности, предлагается все программы, созданные для ЭВМ, разделить на следующие основные классы [1]:

- операционные системы и сервисные программы;
- инструментальные языки и системы программирования;
- прикладные системы.

Рассмотрим эти классы программ.

1. Под операционной системой обычно понимают комплекс управляющих и обрабатывающих программ, который, с одной стороны, выступает как интерфейс между аппаратной частью компьютера и пользователем с его задачами, а с другой – предназначен для наиболее эффективного использования ресурсов вычислительной системы и организации надежных вычислений [2]. Любой из компонентов прикладного ПО обязательно работает под управлением ОС.

Основные функции ОС состоят в следующем [2]:

- прием от пользователя или от оператора системы заданий или команд, сформулированных на соответствующем языке

в виде директив (команд) оператора или указаний (своеобразных команд) с помощью соответствующего манипулятора (например, с помощью мыши), и их обработка;

- прием и исполнение программных запросов на запуск, приостановку, остановку других программ;
- загрузка в оперативную память подлежащих исполнению программ;
- инициация программы – передача данной конкретной программе управления, в результате чего процессор приступает к ее выполнению;
- идентификация всех программ и данных;
- обеспечение режима мультипрограммирования, то есть выполнение двух или более программ на одном процессоре, создающее видимость их одновременного исполнения;
- организация и управление всеми операциями ввода/вывода;
- исполнение режима жестких ограничений на время ответа в режиме реального времени (характерно для соответствующих ОС);
- распределение памяти, а в большинстве современных систем и организация виртуальной памяти;
- планирование и диспетчеризация задач в соответствии с заданной стратегией и дисциплиной обслуживания;
- организация механизмов обмена сообщениями и данными между выполняющимися программами;
- защита одной программы от влияния другой и обеспечение сохранности данных;
- предоставление услуг в случае частичного сбоя системы;
- обеспечение работы систем программирования, с помощью которых пользователи создают свои программы;
- обеспечение работы систем управления базами данных (СУБД);
- обеспечение работы систем управления файлами (СУФ).

Операционная система, выполняя функции управления вычислительными процессами в вычислительной системе, распределяет ресурсы вычислительной системы между различными вычислительными процессами и образует программную среду, в

которой выполняются прикладные программы пользователей, называемую **операционной средой** [2].

Чтобы лучше понять необходимость введения понятия операционной среды, обратимся к истории развития вычислительных систем. Любая программа имеет дело с некоторыми исходными данными, которые она обрабатывает, порождая в конечном итоге некоторые выходные данные и результаты вычислений. Очевидно, что в абсолютном большинстве случаев исходные данные с внешних (периферийных) устройств попадают в оперативную память, с которой непосредственно работает процессор, выполняя вычисления по программе. Аналогично и результаты вычислений также должны быть выведены на внешние устройства. Следует заметить, что программирование операций ввода/вывода относится, пожалуй, к наиболее сложным и трудоемким задачам. Дело в том, что при создании таких программ без использования современных систем программирования нужно знать не только архитектуру процессора (его состав, назначение основных регистров, систему команд процессора, форматы данных и т.п.), но и архитектуру подсистемы ввода/вывода (соответствующие интерфейсы, протоколы обмена данными, алгоритм работы контроллера устройства ввода/вывода и т.д.).

В пятидесятые годы при разработке первых систем программирования прежде всего создавали программные модули для подсистемы ввода/вывода, а уже затем – для вычисления часто встречающихся математических операций и функций. Благодаря этому при создании прикладных программ программисты могли просто обращаться к соответствующим функциям ввода/вывода либо иным функциям и процедурам, что избавляло их от необходимости каждый раз создавать все программные компоненты «с нуля» и знать во всех подробностях особенности работы контроллеров ввода/вывода и соответствующих интерфейсов.

Совершенствование процесса создания программ, выполненных в двоичных машинных кодах, выразилось в разработке программных продуктов, способных предоставить необходимые вызовы к уже готовым библиотечным программным модулям (например, транслятор с алгоритмического языка более высокого уровня в отличие от первых ассемблеров, который мог вместо

оператора типа «Чтение» или «Запись» подставить необходимый вызов). Состав и количество библиотек систем программирования постоянно увеличивались. В конечном итоге возникла ситуация, когда при создании программ в двоичных машинных кодах программистам уже не требовалось знание множества особенностей управления конкретными ресурсами вычислительной системы, а необходимо было только корректное обращение к некоторой программной подсистеме с целью получения требуемых сервисов. Эта программная подсистема и есть **операционная система (ОС)**, а набор ее функций, сервисов и правил обращения к ним как раз и образуют то базовое понятие, которое называется операционной средой. Таким образом, можно сказать, что **операционная среда** – это набор соответствующих интерфейсов, необходимых программам и пользователям для обращения к ОС с целью получения определенных сервисов.

Из всех перечисленных функций операционных систем следует остановиться особо на обеспечении работы систем управления файлами, назначение которой состоит в организации удобного доступа к данным, организованным как файлы. Именно благодаря системе управления файлами вместо низкоуровневого доступа к данным с указанием конкретных физических адресов записи используется логический доступ с указанием имени файла и записи в нем. Особое внимание к этой функции обусловлено тем, что СУФ можно выделить в отдельную категорию ПО [2], поскольку имеются ОС, позволяющие работать с несколькими файловыми системами (с одной из нескольких, либо с несколькими одновременно), и в этом смысле они самостоятельны. Более того, существуют ОС, которые могут работать и без файловых систем, а значит, им необязательно иметь систему управления файлами. Любая СУФ не существует сама по себе – она предназначена для работы в конкретной ОС и с конкретной файловой системой.

Для удобства взаимодействия с ОС могут использоваться дополнительные **интерфейсные оболочки**. Их основное назначение – расширение возможностей по управлению ОС и изменение встроенных в систему возможностей под конкретные требования пользователя. В качестве классических примеров ин-

терфейсных оболочек и соответствующих операционных сред выполнения программ можно назвать различные варианты графического интерфейса X Windows в системах семейства UNIX, PM Shell или Object Desktop в OS/2 с графическим интерфейсом Presentation Manager; разнообразные варианты интерфейсов для семейства ОС Windows компании Microsoft, которые заменяют Explorer и обладают функциями графического интерфейса, таких ОС, как UNIX, OS/2 либо MAC OS. Следует отметить, что о семействе ОС компании Microsoft с общим интерфейсом, реализуемым программными модулями с названием Explorer (в файле system.ini, находящемся в каталоге Windows, имеется строка SHELL=EXPLORER.EXE), все же можно сказать, что заменяемой в этих системах является только интерфейсная оболочка, в то время как сама операционная среда остается неизменной (она интегрирована в ОС). Другими словами, операционная среда определяется программным интерфейсом API (Application Program Interface), включающим в себя управление процессами, памятью и вводом/выводом.

Существуют операционные системы, способные организовывать выполнение программ, созданных для других ОС. Например, в OS/2 наряду с выполнением собственных программ могут использоваться программы, предназначенные для выполнения в среде MS DOS и Windows3.x. Соответствующая операционная среда организуется в ОС в рамках отдельной виртуальной машины. Аналогично, в системе Linux можно создать условия для выполнения некоторых программ, написанных для Windows 95/98/Me. Определенными возможностями исполнения программ, созданных для иной операционной среды, обладают ОС на платформе Windows NT. Эта система позволяет выполнять некоторые программы, созданные для MS DOS, OS/2, Windows3.x.

К сервисным программам ОС относятся и ***эмуляторы***, позволяющие смоделировать в одной операционной системе какую-либо виртуальную машину или операционную систему. Так, известна система эмуляции WMWARE, которая позволяет запустить в среде Linux любую другую ОС, например Windows. Можно, наоборот, создать эмулятор, работающий в среде Windows, который позволит смоделировать компьютер, функ-

ционирующий под управлением любой ОС, в том числе и под Linux.

В составе ОС присутствуют ***сервисные программы* (*утилиты* ОС).** Это специальные системные программы, с помощью которых можно как обслуживать саму операционную систему, так и подготавливать для работы носители данных, выполнять перекодирование данных, осуществлять оптимизацию размещения данных на носителе и производить некоторые другие работы, связанные с обслуживанием вычислительной системы. В качестве утилит также можно рассматривать такие программы, как программы разбиения на разделы накопителя на магнитных дисках, форматирования, переноса основных системных файлов самой ОС. К утилитам относятся и небезызвестные комплексы программ от фирмы Symantec, носящие имя Питера Нортона (создателя этой фирмы и соавтора популярного набора утилит для первых IBM PC). Естественно, что утилиты могут работать только в соответствующей операционной среде.

2. Инструментальные языки и системы программирования представляются, прежде всего, такими компонентами, как транслятор с соответствующего языка, библиотеки подпрограмм, редакторы, компоновщики и отладчики. Не бывает самостоятельных, оторванных от ОС, систем программирования. Любая система программирования может работать только в соответствующей, специально для нее созданной ОС, однако при этом она располагает возможностями создания программного обеспечения, предназначенного для других ОС. Например, одна из популярных систем программирования на языке C/C++ от фирмы Watcom для OS/2 позволяет создавать программы непосредственно для OS/2, а также и для DOS и Windows. В том случае, когда создаваемые программы должны работать на принципиально иной аппаратной базе, используют так называемые кросс-системы. Так, для ПК на базе микропроцессоров семейства i80x86 имеется большое количество кросс-систем, позволяющих создавать программное обеспечение для различных микропроцессоров и микроконтроллеров.

3. Прикладными системами является ПО, ориентирванное на автоматизацию конкретных видов деятельности, например обучение определенным предметам, проектирование

электронных изделий или строительных сооружений, анализ электрокардиограмм, проведение финансовых расчетов и многое другое. Кроме того, прикладные системы могут также обеспечивать автоматизацию таких общих функций, присущих многим видам деятельности, как формирование и печать различных документов, хранение и выдача справок и т.п.

1.2 Вычислительный процесс. Ресурс

Понятие **«вычислительный процесс»** (или просто – «процесс») является одним из основных при рассмотрении операционных систем. По принципу выполнения различают последовательные процессы и параллельные. ***Последовательный процесс,*** иногда называемый «задачей», – это выполнение отдельной программы с ее данными на последовательном процессоре [3]. В концепции, которая получила наибольшее распространение в 70-е годы, под *задачей* (*task*) понимается совокупность связанных между собой и образующих единое целое программных модулей и данных, требующая ресурсов вычислительной системы для своей реализации. В последующие годы задачей стали называть единицу работы, для выполнения которой предоставляется центральный процессор. Вычислительный процесс может включать в себя несколько задач. Концептуально процессор рассматривается в двух аспектах: во-первых, он является носителем данных и, во-вторых, он (одновременно) выполняет операции, связанные с их обработкой.

В качестве примеров можно назвать следующие процессы (задачи): выполнение прикладных программ пользователей, утилит и других системных обрабатывающих программ. Процессами могут быть редактирование какого-либо текста, трансляция исходной программы, ее компоновка, исполнение. Причем трансляция какой-либо исходной программы является одним процессом, а трансляция следующей исходной программы – другим процессом, хотя транслятор, как объединение программных модулей, здесь выступает как одна и та же программа, но данные, которые он обрабатывает, являются разными.

Определение концепции процесса преследует цель выработать механизмы распределения и управления **ресурсами**. Поня-

тие ресурса вычислительного процесса при рассмотрении операционных систем является не менее важным. Термин «ресурс» обычно применяется по отношению к неоднократно используемым, относительно стабильным и «дефицитным» объектам, которые запрашиваются, используются и освобождаются процессами в период их активности. Другими словами, **ресурсом** является любой объект, который может распределяться внутри системы [2]. Ресурсы могут быть ***разделяемыми***, когда несколько процессов могут их использовать *одновременно* (в один и тот же момент времени) или *параллельно* (в течение некоторого интервала времени процессы используют ресурс попеременно), а могут быть и ***неделимыми*** (рис. 1.1).

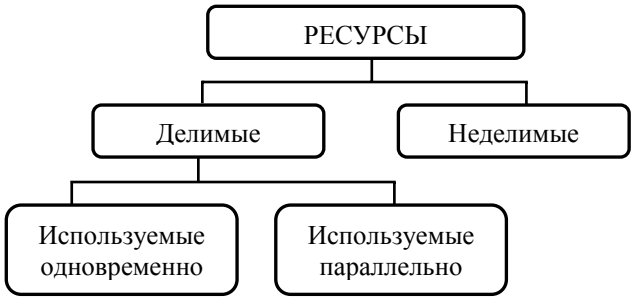

Рис. 1.1 – Классификация ресурсов

При разработке первых систем программирования под понятием «ресурсы» понимали процессорное время, память, каналы ввода/вывода и периферийные устройства [4]. Однако скоро понятие ресурса стало более универсальным и общим. Различного рода программные и информационные ресурсы также могут быть определены для системы как объекты, которые могут разделяться и распределяться, и доступ к которым необходимо соответствующим образом контролировать. В настоящее время понятие ресурса превратилось в абстрактную структуру с целым рядом атрибутов, характеризующих способы доступа к этой структуре и ее физическое представление в системе. Более того, к ресурсам стали относиться и такие объекты, как сообщения и синхросигналы, которыми обмениваются задачи.

В первых вычислительных системах любая программа могла выполняться только после полного завершения предыдущей. Поскольку такие вычислительные системы были построены в соответствии с принципами, изложенными в известной работе фон Неймана, все подсистемы и устройства компьютера функционировали исключительно под управлением центрального процессора. Центральный процессор осуществлял и выполнение вычислений, и управление операциями ввода/вывода данных. Соответственно, пока осуществлялся обмен данными между оперативной памятью и внешними устройствами, процессор не мог выполнять вычисления. Введение в состав вычислительной машины специальных контроллеров позволило совместить во времени (распараллелить) операции вывода полученных данных и последующие вычисления на центральном процессоре. Однако по-прежнему процессор продолжал часто и подолгу простаивать, дожидаясь завершения очередной операции ввода/вывода. Поэтому было предложено организовать так называемый **мультипрограммный (мультизадачный) режим работы вычислительной системы.** Суть его заключается в том, что пока одна программа (один вычислительный процесс или задача) ожидает завершения очередной операции ввода/вывода, другая программа (а точнее, другая задача) может быть поставлена на решение.

Мультипрограммирование – это режим обработки данных, при котором ресурсы вычислительной системы выделяются каждому процессу из группы процессов обработки данных, находящихся в вычислительной системе, на интервалы времени, длительность и очередность предоставления которых определяется управляющей программой этой системы с целью обеспечения одновременной работы в интерактивном режиме. При мультипрограммировании повышается пропускная способность системы, но время выполнения отдельного процесса никогда не превышает времени выполнения данного процесса в однопрограммном режиме. Всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождающегося ресурса.

ОС поддерживает режим мультипрограммирования и организует процесс эффективного использования ресурсов путем управления к ним очередью запросов, составляемых тем или

иным способом. Данный режим достигается поддерживанием в памяти более одного процесса, ожидающего процессор, и более одного процесса, готового использовать другие ресурсы, как только последние станут доступными. Общая схема выделения ресурсов такова. При необходимости использовать какой-либо ресурс (оперативную память, устройство ввода/вывода, массив данных и т.п.) задача обращается к супервизору операционной системы (центральному управляющему модулю, который может состоять из нескольких модулей, например супервизора ввода/вывода, супервизора прерываний, супервизора программ, диспетчера задач и т.д.) посредством специальных вызовов (команд, директив) и сообщает о своем требовании. При этом указывается вид ресурса и, если необходимо, его объем (например, количество адресуемых ячеек оперативной памяти, количество дорожек или секторов на системном диске, объем выводимых на устройство печати данных).

Директива обращения задачи к операционной системе передает ей управление, переводя процессор в ***привилегированный режим*** работы, если такой существует. Кроме привилегированного, вычислительные комплексы могут функционировать в ***пользовательском режиме***, а также ***режиме эмуляции*** какого-либо другого компьютера и т.д. Ресурс может быть выделен задаче, обратившейся к супервизору с соответствующим запросом, в следующих случаях [2]:

- ресурс свободен, и в системе нет запросов от задач более высокого приоритета к запрашиваемому ресурсу;
- текущий запрос и ранее выданные запросы допускают совместное использование ресурсов;
- ресурс используется задачей низшего приоритета и может быть временно отобран (разделяемый ресурс).

Получив запрос, операционная система удовлетворяет его и после выполнения запроса ОС возвращает управление задаче, выдавшей данный запрос, или, если ресурс занят, ставит задачу в очередь, переводя ее в состояние ожидания (блокируя). Очередь к ресурсу может быть организована несколькими способами, но чаще всего это осуществляется с помощью списка.

После окончания работы с ресурсом задача с помощью специального вызова супервизора посредством соответствующей

директивы сообщает операционной системе об отказе от ресурса, либо операционная система самостоятельно забирает ресурс, если управление возвращается супервизору после выполнения какой-либо системной функции. Супервизор операционной системы, получив управление по этому обращению, освобождает ресурс и проверяет, имеется ли очередь к освободившемуся ресурсу. При наличии очереди в соответствии с принятой дисциплиной обслуживания и в зависимости от приоритета заявки он выводит из состояния ожидания ждущую ресурс задачу и переводит ее в состояние готовности к выполнению. После этого управление либо передается данной задаче, либо возвращается той, которая только что освободила ресурс.

При выдаче запроса на ресурс в задаче должен быть определен способ владения ресурсами – монопольный или совместное использование с другими задачами. Например, с файлом можно работать монопольно, а можно и совместно с другими задачами.

Если в системе имеется некоторая совокупность ресурсов, то управлять их использованием можно на основе определенной стратегии. Стратегия подразумевает четкую формулировку целей, следуя которым можно добиться эффективного распределения ресурсов.

При организации управления ресурсами необходимо принять решение о том, что в данной ситуации выгоднее: *быстро обслуживать отдельные наиболее важные запросы, предоставлять всем процессам равные возможности либо обслуживать максимально возможное количество процессов и наиболее полно использовать ресурсы* [3].

Необходимо отличать системные управляющие процессы, представляющие работу супервизора операционной системы и занимающиеся распределением и управлением ресурсов, от других процессов: системных обрабатывающих, которые не входят в ядро операционной системы, и процессов пользователя. Отметим, что назначение **ядра ОС** состоит в распределении ресурсов между задачами (процессами) пользователей и системными процессами. Основные функции ядра ОС [5]:

- порождение процессов, уничтожение процессов (завершение) и реализация механизмов связи между процессами;

- обработка прерываний;
- реализация основных функций распределения ресурсов.

Для системных управляющих процессов в большинстве операционных систем ресурсы распределяются изначально и однозначно. Эти процессы управляют ресурсами системы, очередность использования которых складывается вследствие конкуренции между всеми остальными процессами. Поэтому исполнение системных управляющих программ не принято называть процессами. Термин задача можно употреблять только по отношению к процессам пользователей и к системным обрабатывающим процессам. Однако это справедливо не для всех операционных систем. Например, в так называемых микроядерных ОС (см. подраздел 5.2) большинство управляющих программных модулей самой ОС и даже драйверы имеют статус высокоприоритетных процессов, для выполнения которых необходимо выделить соответствующие ресурсы (в качестве примера можно привести ОС реального времени QNX фирмы Quantum Software Systems) [6]. Аналогично и в UNIX-системах выполнение системных программных модулей тоже имеет статус системных процессов, которые получают ресурсы для своего исполнения в первую очередь.

Анализ ОС общего назначения, а также, например, ОС реального времени, показывает, что процесс может находиться в одном из двух состояний: активном или пассивном. В активном состоянии процесс может участвовать в конкуренции за использование ресурсов вычислительной системы, а в пассивном – не участвует, хотя в системе и имеется информация об его существовании, что сопряжено с предоставлением ему оперативной и/или внешней памяти. Активный процесс может находиться в одном из следующих состояний [2]:

выполнение: затребованные процессом ресурсы выделены. В этом состоянии в каждый момент времени может находиться только один процесс, если речь идет об однопроцессорной вычислительной системе;

готовность к выполнению: ресурсы могут быть предоставлены, тогда процесс перейдет в состояние выполнения;

блокирование или ожидание: затребованные ресурсы не могут быть предоставлены, или не завершена операция ввода/вывода.

В большинстве операционных систем последнее состояние, в свою очередь, подразделяется на множество состояний ожидания, соответствующих определенному виду ресурса, из-за отсутствия которого процесс переходит в заблокированное состояние.

В обычных ОС, как правило, процесс инициализируется при запуске какой-нибудь программы. ОС организует (порождает или выделяет) для нового процесса соответствующий дескриптор процесса, и процесс (задача) начинает развиваться (выполняться). Поэтому пассивного состояния не существует. В ОС реального времени ситуация иная. Обычно при проектировании системы реального времени уже заранее известен состав программ (задач), которые должны будут выполняться. Известны и многие их параметры, которые необходимо учитывать при распределении ресурсов (например, объем памяти, приоритет, средняя длительность выполнения, открываемые файлы, используемые устройства и т.п.). Поэтому для них заранее заводят дескрипторы задач, с тем чтобы впоследствии не тратить драгоценное время на организацию дескриптора и поиск для него необходимых ресурсов. Таким образом, в ОС реального времени многие процессы (задачи) могут находиться в состоянии бездействия.

За время своего существования процесс может неоднократно совершать переходы из одного состояния в другое. Это обусловлено обращениями к операционной системе на запрос ресурсов и выполнение системных функций, которые предоставляет операционная система, взаимодействием с другими процессами, появлением сигналов прерывания от таймера, каналов и устройств ввода/вывода, а также других устройств. Возможные переходы процесса из одного состояния в другое отображены в виде графа состояний на рис. 1.2 [2]. Рассмотрим переходы из одного состояния в другое более подробно.

Процесс из состояния бездействия может перейти в состояние готовности в следующих случаях:

- по команде оператора (пользователя). Имеет место в тех диалоговых операционных системах, где программа может иметь статус задачи и при этом являться пассивной, а не быть просто исполняемым файлом и только на время исполнения получать статус задачи (как это происходит в большинстве современных ОС для ПК);
- при выборе из очереди планировщиком (характерно для операционных систем, работающих в пакетном режиме);
- по вызову из другой задачи (посредством обращения к супервизору один процесс может создать, инициировать, приостановить, остановить, уничтожить другой процесс);
- по прерыванию от внешнего инициативного[1] устройства (сигнал о свершении некоторого события может запускать соответствующую задачу);
- при наступлении запланированного времени запуска программы.

Последние два способа запуска задачи, при которых процесс из состояния бездействия переходит в состояние готовности, характерны для операционных систем реального времени.

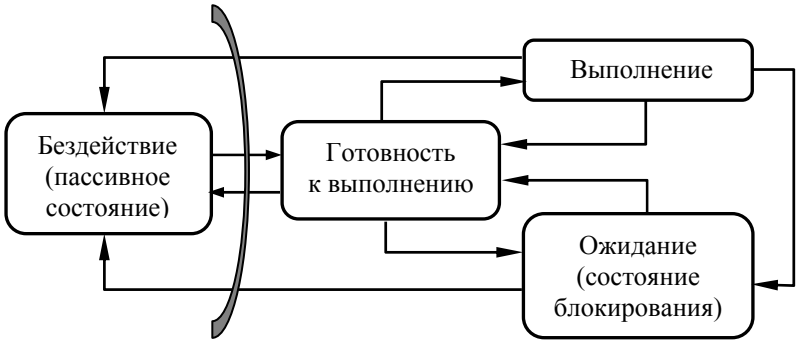

Рис. 1.2 – Граф состояния процесса

Процесс, который может исполняться, как только ему будет предоставлен процессор, а для диск-резидентных задач в некоторых системах – и оперативная память, находится в состоянии

[1] Устройство называется «инициативным», если по сигналу запроса на прерывание от него должна запускаться некоторая задача.

готовности. Считается, что такому процессу уже выделены все необходимые ресурсы за исключением процессора.

Из состояния выполнения процесс может выйти по одной из следующих причин [2]:

- процесс завершается, при этом он посредством обращения к супервизору передает управление ОС и сообщает о своем завершении. В результате этих действий супервизор либо переводит его в список бездействующих процессов (процесс переходит в пассивное состояние), либо уничтожает (уничтожается, естественно, не сама программа, а именно задача, которая соответствовала исполнению некоторой программы). В состояние бездействия процесс может быть переведен принудительно: по команде оператора (действие этой и других команд оператора реализуется системным процессом, который «транслирует» команду в запрос к супервизору с требованием перевести указанный процесс в состояние бездействия), или путем обращения к супервизору операционной системы из другой задачи с требованием остановить данный процесс;
- процесс переводится супервизором ОС в состояние готовности к исполнению в связи с появлением более приоритетной задачи или в связи с окончанием выделенного ему кванта времени;
- процесс блокируется (переводится в состояние ожидания) либо вследствие запроса операции ввода/вывода, которая должна быть выполнена прежде, чем он сможет продолжить исполнение, либо в силу невозможности предоставить ему ресурс, запрошенный в настоящий момент (причиной перевода в состояние ожидания может быть и отсутствие сегмента или страницы в случае организации механизмов виртуальной памяти), а также по команде оператора на приостановку задачи или по требованию через супервизор от другой задачи.

При наступлении соответствующего события (завершение операции ввода/вывода, освобождение затребованного ресурса, загрузка в оперативную память необходимой страницы виртуальной памяти и т.д.) процесс деблокируется и переводится в состояние готовности к исполнению. Таким образом, движущей силой, меняющей состояния процессов, являются события. Одним из основных видов событий являются прерывания.

Для того чтобы операционная система могла управлять процессами, она должна располагать всей необходимой для этого информацией. С этой целью на каждый процесс заводится специальная информационная структура, называемая **дескриптором процесса** (описателем задачи, блоком управления задачей). В общем случае дескриптор процесса содержит следующую информацию:

- идентификатор процесса PID (Process Identificator);
- тип (или класс) процесса, который определяет для супервизора некоторые правила предоставления ресурсов;
- приоритет процесса, в соответствии с которым супервизор предоставляет ресурсы. В рамках одного класса процессов в первую очередь обслуживаются более приоритетные процессы;
- переменную состояния, которая определяет, в каком состоянии находится процесс (готов к работе, в состоянии выполнения, ожидание устройства ввода/вывода и т.д.);
- защищенную область памяти (или адрес такой зоны), в которой хранятся текущие значения регистров процессора, если процесс прерывается, не закончив работы. Эта информация называется контекстом задачи;
- сведения о ресурсах, которыми процесс владеет и/или имеет право пользоваться (указатели на открытые файлы, информация о незавершенных операциях ввода/вывода и т.п.);
- место (или его адрес) для организации общения с другими процессами;
- параметры времени запуска (момент времени, когда процесс должен активизироваться, и периодичность этой процедуры);
- в случае отсутствия системы управления файлами – адрес задачи на диске в ее исходном состоянии и адрес на диске, куда она выгружается из оперативной памяти, если ее вытесняет другая (для диск-резидентных задач, которые постоянно находятся во внешней памяти на системном магнитном диске и загружаются в оперативную память только на время выполнения).

Описатели задач, как правило, постоянно располагаются в оперативной памяти с целью ускорения работы супервизора, который организует их в списки (очереди) и отображает изменение состояния процесса перемещением соответствующего описателя из одного списка в другой. Для каждого состояния (за

исключением состояния выполнения для однопроцессорной системы) операционная система ведет соответствующий список задач, находящихся в этом состоянии. Однако для состояния ожидания может быть не один список, а столько, сколько различных видов ресурсов могут вызывать состояние ожидания. Например, состояний ожидания завершения операции ввода/вывода может быть столько, сколько устройств ввода/вывода имеется в системе.

В некоторых операционных системах количество описателей определяется жестко и заранее (на этапе генерации варианта операционной системы или в конфигурационном файле, который используется при загрузке ОС), в других – по мере необходимости система может выделять участки памяти под новые описатели. Например, в OS/2 максимально возможное количество описателей задач определяется в конфигурационном файле CONFIG.SYS, а в Windows NT оно в явном виде не задается. Стоит заметить, что в упомянутом файле указывается количество не процессов, а именно задач, и под задачей в данном случае понимается как процесс, так и поток этого же процесса, называемый **потоком** или **тредом**.

Для аппаратной поддержки работы ОС с этими информационными структурами (дескрипторами задач) в процессорах могут быть реализованы соответствующие механизмы. Так, например, в микропроцессорах Intel 80x86, начиная с 80286, имеется специальный регистр TR (Task Register), указывающий местонахождение сегмента состояния задачи TSS (Task State Segment), в котором при переключении с задачи на задачу автоматически сохраняется содержимое регистров процессора [7]. Как правило, в современных ОС для этих микропроцессоров дескриптор задачи включает в себя TSS. Другими словами, дескриптор задачи больше по размеру, чем TSS, и включает в себя такие традиционные поля, как идентификатор задачи, ее имя, тип, приоритет и т.п.

1.3 Потоки (треды)

Понятие процесса было введено при реализации мультипрограммного режима работы вычислительной техники. В свое время различали термины «*мультизадачность*» и «*мультипро-*

граммирование». Для реализации «мультизадачности» в ее исходном толковании необходимо было тоже ввести соответствующую сущность. Такой сущностью и стали так называемые легковесные процессы, или, как их теперь преимущественно называют, – **потоки** или **треды (нити)**.

При рассмотрении процессов (process) имеется в виду, что операционная система поддерживает их обособленность: у каждого процесса имеется свое виртуальное адресное пространство; каждому процессу назначаются свои ресурсы – файлы, окна, семафоры и т.д. Такая обособленность нужна для того, чтобы защитить один процесс от другого, поскольку они, совместно используя все ресурсы вычислительной системы, конкурируют друг с другом. В общем случае процессы просто никак не связаны между собой и могут принадлежать даже разным пользователям, разделяющим одну вычислительную систему.

Однако желательно иметь еще и возможность задействовать внутренний параллелизм, который может быть в самих процессах. Внутренний параллелизм встречается достаточно часто, и его использование позволяет ускорить их решение. Например, некоторые операции, выполняемые приложением, могут требовать для своего исполнения достаточно продолжительное время использования центрального процессора. В этом случае при интерактивной работе с приложением пользователь вынужден долго ожидать завершения заказанной операции и не может управлять приложением до тех пор, пока операция не выполнится до самого конца. Такие ситуации встречаются достаточно часто, например, при обработке больших изображений в графических редакторах. Если же программные модули, исполняющие такие длительные операции, оформлять в виде самостоятельных «подпроцессов» – **легковесных или облегченных процессов** (**потоков** или задач), которые будут выполняться параллельно с другими «подпроцессами» (потоками, задачами), то у пользователя появляется возможность параллельно выполнять несколько операций в рамках одного приложения (процесса). Легковесными эти задачи называют потому, что операционная система не должна для них организовывать полноценную виртуальную машину. Эти задачи не имеют своих собственных ресурсов, они развиваются в том же виртуальном адресном про-

странстве, могут пользоваться теми же файлами, виртуальными устройствами и иными ресурсами, что и исполняемый процесс. Единственное, что им необходимо иметь, – это процессорный ресурс. В однопроцессорной системе треды (задачи) разделяют между собой процессорное время так же, как это делают обычные процессы, а в мультипроцессорной системе могут выполняться одновременно, если не встречают конкуренции из-за обращения к иным ресурсам.

Главное, что обеспечивает **многопоточность**, – это возможность параллельно выполнять несколько видов операций в одной прикладной программе. Параллельные вычисления (а следовательно, и более эффективное использование ресурсов центрального процессора, и меньшее суммарное время выполнения задач) теперь уже часто реализуются на уровне тредов, и программа, оформленная в виде нескольких тредов в рамках одного процесса, может быть выполнена быстрее за счет параллельного выполнения ее отдельных частей. Например, если электронная таблица или текстовый процессор были разработаны с учетом возможностей многопоточной обработки, то пользователь может запросить пересчет своего рабочего листа или слияние нескольких документов и одновременно продолжить заполнение таблицы или открыть для редактирования следующий документ.

Особенно эффективно можно использовать многопоточность для выполнения распределенных приложений; например, многопоточный сервер может параллельно выполнять запросы сразу нескольких клиентов. Как известно, в операционной системе OS/2, одной из первых среди ОС, используемых на ПК, была введена многопоточность. В середине девяностых годов для этой ОС было создано очень большое количество приложений, в которых использование механизмов многопоточной обработки реально приводило к существенно большей скорости выполнения вычислений.

Итак, сущность «***поток***» была введена для того, чтобы с помощью именно этих единиц распределять процессорное время между возможными работами. Сущность «***процесс***» предполагает, что при диспетчеризации нужно учитывать все ресурсы, закрепленные за ним. А при манипулировании ***тредами*** можно

менять только контекст задачи (образ ее текущего состояния) при переключении с одной задачи на другую в рамках одного процесса. Все остальные вычислительные ресурсы при этом не затрагиваются. Каждый процесс всегда состоит, по крайней мере, из одного потока, и только если имеется внутренний параллелизм, программист может «расщепить» один тред на несколько параллельных [2].

Каждый тред выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Треды, как и процессы, могут порождать треды-потомки, поскольку любой процесс состоит по крайней мере из одного треда. Подобно традиционным процессам (то есть процессам, состоящим из одного треда), каждый тред может находиться в одном из активных состояний. Пока один тред заблокирован (или просто находится в очереди готовых к исполнению задач), другой тред того же процесса может выполняться. Треды разделяют процессорное время так же, как это делают обычные процессы, в соответствии с различными вариантами диспетчеризации.

Так как все треды имеют одно и то же виртуальное адресное пространство своего процесса, они разделяют одни и те же глобальные переменные. Поскольку любой тред может иметь доступ к любому виртуальному адресу, один тред может использовать стек другого треда. Между потоками нет полной защиты, так как это, во-первых, невозможно, а во-вторых, не нужно. Все потоки одного процесса всегда решают общую задачу одного пользователя, и механизм потоков используется здесь для более быстрого решения задачи путем ее распараллеливания. При этом программисту очень важно получить в свое распоряжение удобные средства организации взаимодействия частей одной программы. Кроме разделения адресного пространства, все треды совместно используют также набор открытых файлов, общие устройства, выделенные процессу, имеют одни и те же наборы сигналов, семафоры и т.п. Собственными у тредов являются программный счетчик, стек, рабочие регистры процессора, потоки-потомки, состояние.

Поскольку треды, относящиеся к одному процессу, выполняются в одном и том же виртуальном адресном пространстве, между ними легко организовать тесное взаимодействие, в отли-

чие от процессов, для которых нужны специальные механизмы обмена сообщениями и данными. Более того, при создании многопоточного приложения можно заранее продумать работу множества тредов процесса, организовав их взаимодействие наиболее выгодным способом, а не участвовать в конкуренции за предоставление ресурсов, если этого можно избежать.

Для того чтобы можно было эффективно организовать параллельное выполнение процессов и тредов, в архитектуру современных процессоров включена возможность работы со специальной информационной структурой, описывающей тот или иной процесс (тред). Для этого уже на уровне архитектуры микропроцессора используется понятие «задача» (task). Оно как бы объединяет в себе обычный и «легковесный» процессы. Это понятие и поддерживаемая для него на уровне аппаратуры информационная структура позволяют в дальнейшем при разработке операционной системы построить соответствующие дескрипторы как для процесса, так и для треда. Отличаться эти дескрипторы будут, прежде всего, тем, что дескриптор треда может хранить только контекст приостановленного вычислительного процесса, тогда как дескриптор процесса (process) уже должен содержать поля, описывающие тем или иным способом ресурсы, выделенные этому процессу. Заметим, что ***контекст процесса*** включает в себя содержимое адресного пространства задачи, выделенного процессу, а также содержимое относящихся к процессу аппаратных регистров и структур данных ядра. С формальной точки зрения, контекст процесса объединяет в себе *пользовательский, регистровый и системный контексты*. Каждый тред может быть оформлен в виде самостоятельного сегмента, что приводит к тому, что простая (не многопоточная) программа будет иметь всего один сегмент кода в виртуальном адресном пространстве.

В завершение можно привести несколько советов по использованию потоков при создании приложений [8].

1. В случае использования однопроцессорной системы множество параллельных потоков зачастую не ускоряет работу приложения, поскольку в каждый отдельно взятый промежуток времени возможно выполнение только одного потока. Кроме того, чем больше у вас потоков, тем больше нагрузка на систему

вследствие переключения между ними. Если ваш проект имеет более двух постоянно работающих потоков, которые требуют частого ввода/вывода, то в этом случае применение режима мультизадачности не сделает программу быстрее.

2. Прежде всего необходимо определить цели создания потока. Поток, осуществляющий обработку, может мешать быстрому реагированию системы на запросы ввода/вывода. Потоки позволяют программе отзываться на просьбы пользователя и устройств, но при этом сильно загружают процессор. Потоки позволяют компьютеру одновременно обслуживать множество устройств, и созданный вами поток, отвечающий за обработку специфического устройства, в качестве минимума может потребовать столько времени, сколько системе необходимо для обработки запросов всех устройств.

3. Потокам можно назначить определенный приоритет для того, чтобы наименее значимые процессы выполнялись в фоновом режиме. Это путь честного разделения ресурсов процессора. Однако необходимо осознать тот факт, что процессор один на всех, а потоков много. Если в вашей программе главная процедура передает нечто для обработки в низкоприоритетный поток, то сама программа становится просто неуправляемой.

4. Потоки эффективно работают, когда они независимы. Но они начинают работать непродуктивно, если достаточно часто возникает необходимость в процессах синхронизации для доступа к общим ресурсам. Блокировка и критические секции отнюдь не увеличивают скорость работы системы, хотя без использования этих механизмов взаимодействующие вычисления организовывать нельзя.

5. Помните, что память виртуальна. Механизм виртуальной памяти следит за тем, какая часть виртуального адресного пространства должна находиться в оперативной памяти, а какая должна быть сброшена в файл подкачки. Потоки усложняют ситуацию, если они обращаются в одно и то же время к разным адресам виртуального адресного пространства приложения. Это значительно увеличивает нагрузку на систему, особенно при небольшом объеме кэш-памяти (см. раздел 3).

6. Всякий раз, когда какой-либо из потоков пытается воспользоваться общим ресурсом вычислительного процесса, кото-

рому он принадлежит, вы обязаны тем или иным образом легализовать и защитить свою деятельность. Хорошим средством для этого являются критические секции, семафоры и очереди сообщений. Если вы протестировали свое приложение и не обнаружили ошибок синхронизации, то это еще не значит, что их там нет. Пользователь может создать самые непредсказуемые ситуации. Это очень ответственный момент в разработке многопоточных приложений.

7. Не возлагайте на поток несколько функций. Сложные функциональные отношения затрудняют понимание общей структуры приложения, его алгоритм. Чем проще и однозначнее каждая из рассматриваемых ситуаций, тем больше вероятность исключения ошибок.

1.4 Прерывания

Прерывания представляют собой механизм, позволяющий координировать параллельное функционирование отдельных устройств вычислительной системы и реагировать на особые состояния, возникающие при работе процессора [2]. Таким образом, **прерывание** – это принудительная передача управления от выполняемой программы к системе, а через нее – к соответствующей программе обработки прерывания, происходящая при возникновении определенного события.

Идея прерываний была предложена в середине 50-х годов и можно без преувеличения сказать, что явилась достаточно весомым вкладом в развитие вычислительной техники. Основная цель введения прерываний – реализация асинхронного режима работы и распараллеливание работы отдельных устройств вычислительного комплекса.

Механизм прерываний реализуется аппаратно-программными средствами. Структуры систем прерывания в зависимости от аппаратной архитектуры могут быть самыми разными, но все они имеют одну общую особенность – прерывание непременно влечет за собой изменение порядка выполнения команд процессором. Механизм обработки прерываний независимо от архитектуры вычислительной системы включает следующие шаги [2]:

1) установление факта прерывания (прием сигнала на прерывание) и идентификация прерывания (в операционных системах иногда осуществляется повторно на шаге 4);

2) запоминание состояния прерванного процесса, определяемое, прежде всего, значением счетчика команд (адресом следующей команды, который, например, в процессоре i80x86 определяется регистрами CS и IP-указателем команды), содержимым регистров процессора и может включать также спецификацию режима (например, режим пользовательский или привилегированный) и другую информацию [7];

3) аппаратная передача управления подпрограмме обработки прерывания. В простейшем случае в счетчик команд заносится начальный адрес подпрограммы обработки прерываний, а в соответствующие регистры – информация из слова состояния. В более развитых процессорах, например в i80286 и последующих 32-битовых микропроцессорах, начиная с i80386, осуществляется достаточно сложная процедура определения начального адреса соответствующей подпрограммы обработки прерывания и не менее сложная процедура инициализации рабочих регистров процессора;

4) сохранение информации о прерванной программе, которую не удалось спасти на шаге 2 с помощью действий аппаратуры. В некоторых вычислительных системах предусматривается запоминание довольно большого объема информации о состоянии прерванного процесса;

5) обработка прерывания. Эта работа может быть выполнена той же подпрограммой, которой было передано управление на шаге 3, но в ОС чаще всего она реализуется путем последующего вызова соответствующей подпрограммы;

6) восстановление информации, относящейся к прерванному процессу (шаг, обратный шагу 4);

7) возврат в прерванную программу.

Шаги 1–3 реализуются аппаратно, а шаги 4–7 – программно.

При возникновении запроса на прерывание естественный ход вычислений нарушается и управление передается программе обработки возникшего прерывания (рис. 1.3). При этом средствами аппаратуры сохраняется (как правило, с помощью механизмов стековой памяти) адрес той команды, с которой следует

продолжить выполнение прерванной программы. После выполнения программы обработки прерывания управление возвращается прерванной ранее программе посредством занесения в указатель команд сохраненного адреса команды. Однако такая схема используется только в самых простых программных средах. В мультипрограммных операционных системах обработка прерываний происходит по более сложным схемам.

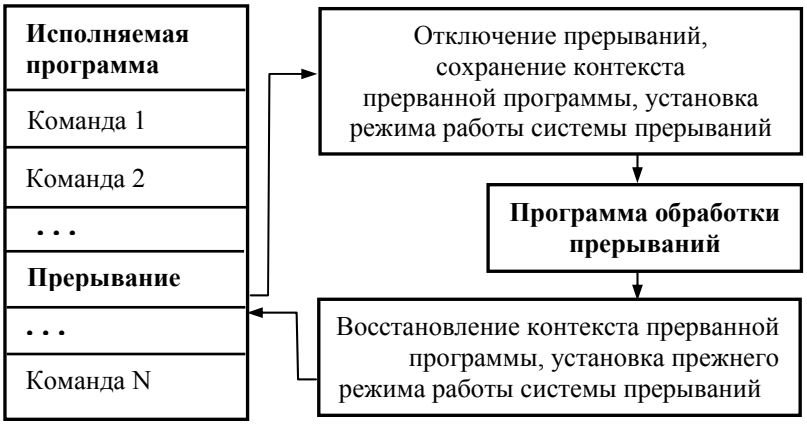

Рис. 1.3 – Обработка прерываний

Механизм прерываний состоит из трех функций:

1) распознавания или классифицирования прерываний;
2) передачи управления обработчику прерываний;
3) корректного возвращения к прерванной программе.

Переход от прерываемой программы к обработчику и обратно должен выполняться как можно быстрей. Одним из быстрых методов является использование таблицы, содержащей перечень всех допустимых для компьютера прерываний и адреса соответствующих обработчиков. Для корректного возвращения к прерванной программе перед передачей управления обработчику прерываний содержимое регистров процессора запоминается либо в памяти с прямым доступом, либо в системном стеке – system stack.

Прерывания, возникающие при работе вычислительной системы, можно разделить на два основных класса: внешние (их иногда называют асинхронными) и внутренние (синхронные).

Внешние прерывания вызываются асинхронными событиями, которые происходят вне прерываемого процесса, например:

- прерывания от таймера;
- прерывания от внешних устройств (по вводу/выводу);
- прерывания по нарушению питания;
- прерывания с пульта оператора вычислительной системы;
- прерывания от другого процессора или другой вычислительной системы.

Внутренние прерывания вызываются событиями, которые связаны с работой процессора и являются синхронными с его операциями. Примерами могут служить следующие запросы на прерывания:

- при нарушении адресации (в адресной части выполняемой команды указан запрещенный или несуществующий адрес, обращение к отсутствующему сегменту или странице при организации механизмов виртуальной памяти);
- при наличии в поле кода операции незадействованной двоичной комбинации;
- при делении на нуль;
- при переполнении или исчезновении порядка;
- при обнаружении ошибок четности, ошибок в работе различных устройств аппаратуры средствами контроля.

Могут возникать прерывания при обращении к супервизору ОС – в некоторых компьютерах часть команд может быть использована только ОС, но не пользователями. Соответственно в аппаратуре предусмотрены различные режимы работы, и пользовательские программы выполняются в режиме, в котором привилегированные команды не исполняются. При попытке использовать команду, запрещенную в данном режиме, происходит внутреннее прерывание и управление передается супервизору ОС. К привилегированным командам относятся и команды переключения режима работа центрального процессора.

Наконец, существуют собственно **программные прерывания**. Такие прерывания происходят по соответствующей команде прерывания, то есть по этой команде процессор осуществляет практически те же действия, что и при обычных внутренних прерываниях. Данный механизм был специально введен для того, чтобы переключение на системные программные модули

происходило не просто как переход в подпрограмму, а точно таким же образом, как и обычное прерывание. Этим обеспечивается автоматическое переключение процессора в привилегированный режим с возможностью исполнения любых команд.

Сигналы, вызывающие прерывания, формируются вне процессора или в самом процессоре; они могут возникать одновременно. Выбор одного из них для обработки осуществляется на основе приоритетов, приписанных каждому типу прерывания. Очевидно, что прерывания от схем контроля процессора должны обладать наивысшим приоритетом (если аппаратура работает неправильно, то не имеет смысла продолжать обработку информации). Учет приоритета может быть встроен в технические средства, а также определяться операционной системой, то есть кроме аппаратно реализованных приоритетов прерывания большинство вычислительных машин и комплексов допускают программно-аппаратное управление порядком обработки сигналов прерывания. Второй способ, дополняя первый, позволяет применять различные дисциплины обслуживания прерываний.

Наличие сигнала прерывания не обязательно должно вызывать прерывание исполняющейся программы. Процессор может обладать средствами защиты от прерываний: отключение системы прерываний, маскирование (запрет) отдельных сигналов прерывания. Программное управление этими средствами посредством специальных команд позволяет операционной системе регулировать обработку сигналов прерывания, заставляя процессор обрабатывать их сразу по поступлению, откладывать их обработку на некоторое время или полностью игнорировать. Обычно операция прерывания выполняется только после завершения выполнения текущей команды. Поскольку сигналы прерывания возникают в произвольные моменты времени, то на момент прерывания может существовать несколько сигналов прерывания, которые могут быть обработаны только последовательно. Чтобы обработать сигналы прерывания в разумном порядке, им, как уже отмечалось, присваиваются приоритеты. Сигнал с более высоким приоритетом обрабатывается в первую очередь, обработка остальных сигналов прерывания откладывается. Программное управление специальными регистрами мас-

ки (маскирование сигналов прерывания) позволяет реализовать различные дисциплины обслуживания:

- **с относительными приоритетами**, то есть обслуживание не прерывается даже при наличии запросов с более высокими приоритетами. После окончания обслуживания данного запроса обслуживается запрос с наивысшим приоритетом. Для организации такой дисциплины необходимо в программе обслуживания данного запроса наложить маски на все остальные сигналы прерывания или просто отключить систему прерываний;
- **с абсолютными приоритетами**, то есть всегда обслуживается прерывание с наивысшим приоритетом. Для реализации этого режима необходимо на время обработки прерывания замаскировать все запросы с более низким приоритетом. При этом возможно многоуровневое прерывание, то есть прерывание программ обработки прерываний. Число уровней прерывания в этом режиме изменяется и зависит от приоритета запроса;
- **по принципу стека**, то есть запросы с более низким приоритетом могут прерывать обработку прерывания с более высоким приоритетом. Для этого необходимо не накладывать маски ни на один сигнал прерывания и не выключать систему прерываний.

В мультипрограммной ОС обработка прерываний происходит по следующей схеме. Супервизор прерываний прежде всего сохраняет в дескрипторе текущей задачи рабочие регистры процессора, определяющие контекст прерываемого вычислительного процесса. Далее он определяет ту подпрограмму, которая должна выполнить действия, связанные с обслуживанием настоящего (текущего) запроса на прерывание. Наконец, перед тем как передать управление этой подпрограмме, супервизор прерываний устанавливает необходимый режим обработки прерывания. После выполнения подпрограммы обработки прерывания управление вновь передается супервизору, на этот раз уже на тот модуль, который занимается диспетчеризацией задач. И уже диспетчер задач, в свою очередь, в соответствии с принятым режимом распределения процессорного времени между выполняющимися процессами восстановит контекст той задачи, для которой будет принято решение о выделении процессора

(рис. 1.4). На рисунке показано, что непосредственного возврата в прерванную ранее программу прямо из самой подпрограммы обработки прерывания нет. Для прямого непосредственного возврата достаточно адрес возврата сохранить в стеке, что и делает аппаратура процессора. При этом стек легко обеспечивает возможность возврата в случае вложенных прерываний, поскольку он всегда реализует дисциплину «последним пришел – первым обслужился».

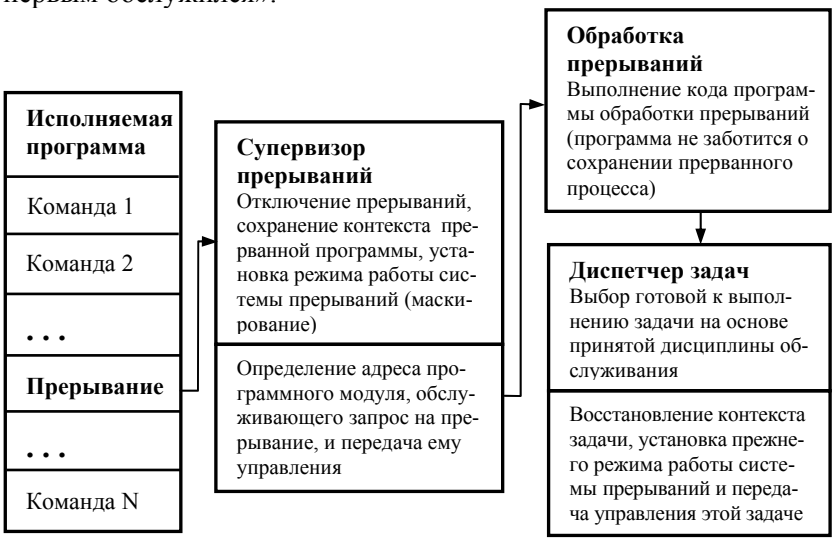

Рис. 1.4 – Обработка прерываний при участии супервизоров ОС

Однако если бы контекст процессов сохранялся просто в стеке, как это обычно реализуется аппаратурой, а не в описанных выше дескрипторах задач, то у нас не было бы возможности гибко подходить к выбору той задачи, которой нужно передать процессор после завершения работы подпрограммы обработки прерывания. Естественно, что это только общий принцип. В конкретных процессорах и в конкретных ОС могут существовать некоторые отступления от рассмотренной схемы и/или дополнения к ней. Например, в современных процессорах часто имеются специальные аппаратные возможности для сохранения контекста прерываемого процесса непосредственно в его деск-

рипторе, то есть дескриптор процесса (по крайней мере, его часть) становится структурой данных, которую поддерживает аппаратура.

1.5 Классификация операционных систем

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями используемых методов проектирования, типов аппаратных платформ, областей применения и многими другими свойствами.

В зависимости от особенностей используемого алгоритма управления процессором выделяют следующие типы операционных систем [1]: ***многозадачные и однозадачные, многопользовательские и однопользовательские; многопроцессорные и однопроцессорные системы; системы, поддерживающие и не поддерживающие распараллеливания вычислений в рамках одной задачи.***

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

1) однозадачные (MS DOS, MSX);
2) многозадачные (OS/2, UNIX, Windows).

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

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

Поддержка многопользовательского режима. По числу одновременно работающих пользователей выделяют: однопользовательские (MS DOS, Windows3.x, ранние версии OS/2); многопользовательские (UNIX, Windows на платформе NT). Главное отличие многопользовательских систем от однопользова-

тельских – наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует заметить, что не всякая многозадачная система является многопользовательской, и не всякая однопользовательская ОС является однозадачной.

Многопроцессорная обработка. Одним из важных свойств ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки. В настоящее время становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris фирмы Sun, Open Server компании Santa Crus Operations, OS/2 фирмы IBM, Windows NT фирмы Microsoft и NetWare фирмы Novell.

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

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

- системы пакетной обработки (ОС ЕС),
- системы разделения времени (UNIX, Windows),
- системы реального времени (QNX).

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки являлась максимальная пропускная способность, то есть решение максимального числа задач в единицу времени. Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач.

Для одновременного выполнения выбираются задачи, предъявляющие отличающиеся требования к ресурсам, так чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины; так, например, в мультипрограммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в таких ОС невозможно гарантировать выполнение того или иного задания в течение определенного периода времени. В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит только в случае, если активная задача сама отказывается от процессора, например, из-за необходимости выполнить операцию ввода-вывода. Поэтому одна задача может надолго занять процессор, что делает невозможным выполнение интерактивных задач. Таким образом, взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а после выполнения всего пакета заданий получает результат. Очевидно, что такой порядок снижает эффективность работы пользователя. В настоящее время системы пакетной обработки практически не используются.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается

каждая запущенная пользователем задача, а не та, которая «выгодна» системе, и, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу. Критерием эффективности систем с разделением времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

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

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

По основному архитектурному принципу ОС разделяются на **микроядерные** и **монолитные.** В некоторой степени это разделение тоже условно, однако можно в качестве яркого примера микроядерной операционной системы привести систему реального времени QNX, тогда как в качестве примера моно-

литной ОС можно назвать Windows 9x или ОС Linux. Ядро ОС Windows нельзя изменить, пользователям недоступны его исходные коды и у них нет программы для сборки (компиляции) этого ядра. А вот в случае работы с ОС Linux пользователи могут сами собрать ядро, отвечающее их потребностям, включив в него программные модули и драйверы, которые целесообразно, по их мнению, включить именно в ядро.

Вопросы для самопроверки

1. Дайте объяснение понятиям операционной среды и операционной системы.
2. В чем отличие между понятиями процесса и задачи?
3. Изобразите диаграмму состояний процесса, поясните все возможные переходы из одного состояния в другое.
4. Объясните значения следующих терминов: task (задача), process (процесс), thread (поток, нить). Как они между собой соотносятся?
5. Для чего каждая задача получает соответствующий дескриптор? Какие поля, как правило, содержатся в дескрипторе процесса (задачи)?
6. Объясните понятие ресурса. Почему понятие ресурса является одним из фундаментальных при рассмотрении ОС? Какие виды и типы ресурсов вы знаете?
7. Как вы считаете: сколько и каких списков дескрипторов задач может быть в системе? От чего должно зависеть это число?
8. Перечислите дисциплины обслуживания прерываний; объясните, как можно реализовать каждую из этих дисциплин.
9. С какой целью в ОС вводится специальный системный модуль, иногда называемый супервизором прерываний?
10. Приведите классификацию ОС?

2. УПРАВЛЕНИЕ ЗАДАЧАМИ

2.1 Основные функции управления задачами

Система управления задачами обеспечивает прохождение их через компьютер. Операционная система выполняет следующие основные функции, связанные с управлением задачами:

- создание и удаление задач;
- планирование процессов и диспетчеризация задач;
- синхронизация задач, обеспечение их средствами коммуникации.

Создание и удаление задач осуществляется по соответствующим запросам от пользователей или от самих задач. Задача может породить новую задачу. При этом между процессами появляются «родственные» отношения. Порождающая задача называется «предком», «родителем», а порожденная – «потомком», «сыном» или «дочерней задачей». «Предок» может приостановить или удалить свою дочернюю задачу, тогда как «потомок» не может управлять «предком».

Основным подходом к организации того или иного метода управления процессами, обеспечивающего эффективную загрузку ресурсов или выполнение каких-либо иных целей, является **организация очередей процессов и ресурсов** [2]. Очевидно, что на распределение ресурсов влияют конкретные потребности тех задач, которые должны выполняться параллельно. Другими словами, можно столкнуться с ситуациями, когда невозможно эффективно распределить ресурсы, исключив возможность их простаивания. Например, всем выполняющимся процессам требуется некоторое устройство с последовательным доступом. Но поскольку оно не может распределяться между параллельно выполняющимися процессами, то процессы вынуждены будут очень долго ждать своей очереди. Таким образом, недоступность одного ресурса может привести к тому, что длительное время не будут использоваться и многие другие ресурсы.

Если же возьмем набор таких процессов, которые не будут конкурировать между собой за неразделяемые ресурсы при параллельном выполнении, то, скорее всего, процессы смогут выполниться быстрее (из-за отсутствия дополнительных ожида-

ний), да и имеющиеся в системе ресурсы будут использоваться более эффективно. Итак, возникает **задача планирования вычислительного процесса**, т.е. организация бесконфликтного выполнения множества процессов.

Задача планирования процессов возникла на этапе создания первых пакетных ОС при планировании пакетов задач, которые должны были выполняться на компьютере и оптимально использовать его ресурсы. В настоящее время актуальность этой задачи не так велика. На первый план уже очень давно вышли задачи динамического (или краткосрочного) планирования, то есть текущего наиболее эффективного распределения ресурсов, возникающего практически при каждом событии. Организация работ по выполнению задач динамического планирования получила название **диспетчеризации**.

Очевидно, что долгосрочное планирование осуществляется гораздо реже, чем задача текущего распределения ресурсов между уже выполняющимися процессами и потоками. Основное отличие между долгосрочным и краткосрочным планировщиками заключается в частоте запуска: краткосрочный планировщик, например, может запускаться каждые 30 или 100 мс; долгосрочный – один раз за несколько минут или чаще (здесь многое зависит от общей длительности решения заданий пользователей).

С помощью долгосрочного планировщика определяется, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых процессов в случае освобождения ресурсов памяти и выбираются процессы из входной очереди с целью создания неоднородной мультипрограммной смеси. Это означает, что в очереди готовых к выполнению процессов должны находиться (в разной пропорции) как процессы, ориентированные на ввод/вывод, так и процессы, ориентированные на преимущественную работу с центральным процессором.

Функция краткосрочного планировщика состоит в определении конкретных задач из находящихся в очереди, готовых к выполнению, которые должны быть переданы на исполнение. В большинстве современных операционных систем долгосрочный планировщик отсутствует.

2.2 Планирование процессов и диспетчеризация задач

2.2.1 Дисциплины диспетчеризации

В основе определения дисциплины диспетчеризации лежит выбор стратегии планирования. **Стратегия планирования** определяет, какие именно процессы могут быть направлены на выполнение для достижения целей, поставленных перед данными процессами. Известно большое количество различных стратегий выбора процесса, которому необходимо предоставить процессор. Среди них, прежде всего, можно назвать следующие стратегии [2]:

- обеспечения соответствия порядка окончания вычислений (вычислительных процессов) последовательности, в которой они были приняты к исполнению;
- оказание предпочтения более коротким процессам;
- предоставление всем пользователям (процессам пользователей) одинаковых услуг, в том числе и одинакового времени ожидания.

Долгосрочное планирование заключается в подборе таких вычислительных процессов, которые бы меньше всего конкурировали между собой за ресурсы вычислительной системы. Когда говорят о *стратегии планирования,* всегда имеют в виду *понятие процесса*, а не понятие задачи, поскольку процесс может состоять из нескольких потоков (задач).

Следует отметить, что при рассмотрении стратегий планирования, как правило, имеется в виду краткосрочное планирование, то есть диспетчеризация (обслуживание). Когда говорят о *диспетчеризации*, то всегда в явном или неявном виде имеют в виду *понятие задачи* (*потока*). Если ОС не поддерживает механизм тредов, то можно заменять понятие задачи понятием процесса.

Известно большое количество правил (дисциплин) диспетчеризации, в соответствии с которыми формируется список (очередь) готовых к выполнению задач. Различают два больших класса дисциплин диспетчеризации (дисциплин обслуживания) – бесприоритетные и приоритетные. При **бесприоритетном обслуживании**

выбор задач производится в некотором заранее установленном порядке без учета их относительной важности и времени обслуживания. При реализации **приоритетных дисциплин обслуживания** отдельным задачам предоставляется преимущественное право попасть в состояние исполнения. Классификация дисциплин диспетчеризации приведена на рис. 2.1.

Рис. 2.1 – Дисциплины диспетчеризации

Приоритеты имеют следующие свойства [2]:

- приоритет, присвоенный задаче, может являться величиной постоянной (статический приоритет);
- приоритет задачи может изменяться в процессе ее решения (динамический приоритет).

Диспетчеризация с динамическими приоритетами требует дополнительных расходов на вычисление значений приоритетов исполняющихся задач, поэтому во многих ОС реального времени используются методы диспетчеризации на основе статических (постоянных) приоритетов. Хотя следует заметить, что ди-

намические приоритеты позволяют реализовать гарантии обслуживания задач.

Рассмотрим кратко основные, наиболее часто используемые **дисциплины диспетчеризации с использованием статических приоритетов**.

Самой простой в реализации является **дисциплина FCFS** (first come – first served), согласно которой задачи обслуживаются в порядке очереди, то есть в порядке их появления. Те задачи, которые были заблокированы в процессе работы (попали в какое-либо из состояний ожидания, например, из-за операций ввода/вывода), после перехода в состояние готовности ставятся в эту очередь перед теми задачами, которые еще не выполнялись. Другими словами, образуются две очереди (рис. 2.2): одна очередь образуется из новых задач, а вторая очередь – из ранее выполнявшихся, но попавших в состояние ожидание. Такой подход позволяет реализовать стратегию обслуживания, формулируемую как «по возможности заканчивать вычисления в порядке их появления». Эта дисциплина обслуживания не требует внешнего вмешательства в ход вычислений и перераспределения процессорного времени. Существующие дисциплины диспетчеризации процессов могут быть разбиты на два класса: **вытесняющие** (preemptive) и **невытесняющие** (non-preemptive). В ранее разработанных пакетных ОС часто реализовывали параллельное выполнение заданий без принудительного перераспределения процессора между задачами. В большинстве современных ОС для мощных вычислительных систем, а также и в ОС для ПК, ориентированных на высокопроизводительное выполнение приложений (Windows NT, OS/2, Linux), реализована вытесняющая многозадачность. Можно сказать, что рассмотренная дисциплина относится к невытесняющим.

К достоинствам этой дисциплины диспетчеризации, прежде всего, можно отнести простоту реализации и малые расходы системных ресурсов на формирование очереди задач. Однако эта дисциплина приводит к тому, что при увеличении загрузки вычислительной системы растет и среднее время ожидания обслуживания, причем короткие задания, требующие небольших затрат машинного времени, вынуждены ожидать столько же,

сколько и трудоемкие задания. Избежать этого недостатка позволяют дисциплины SJN и SRT.

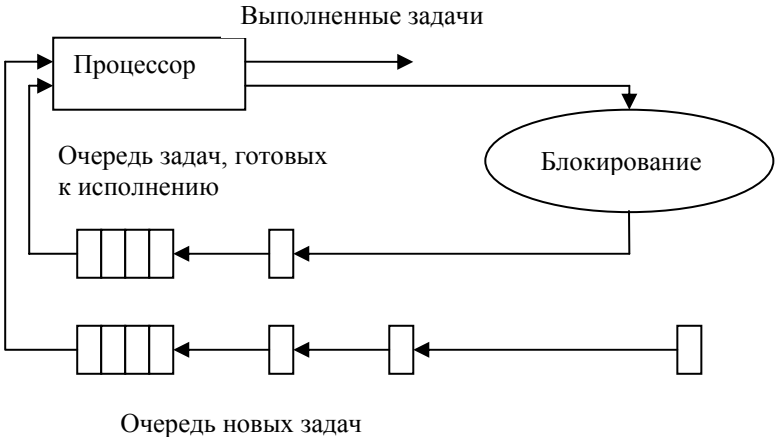

Рис. 2.2 – Дисциплины диспетчеризации FCFS

Дисциплина обслуживания SJN (Shortest Job Next) предполагает, что следующим будет выполняться кратчайшее задание. Для ее реализации необходимо, чтобы для каждого задания была известна оценка в потребностях машинного времени. Необходимость сообщать ОС характеристики задач, в которых описывались бы потребности в ресурсах вычислительной системы, привела к тому, что были разработаны соответствующие языковые средства. Одним из наиболее известных был, в частности, язык управления заданиями JCL (Job Control Language). Пользователи вынуждены были указывать предполагаемое время выполнения задания. Для того чтобы они не злоупотребляли возможностью указать заведомо меньшее время выполнения с целью получить результаты раньше других, ввели подсчет реальных потребностей. Диспетчер задач сравнивал заказанное время и время выполнения и в случае превышения указанной оценки в данном ресурсе ставил данное задание не в начало, а в конец очереди. В некоторых ОС в таких случаях использовалась система штрафов, при которой в случае превышения заказанно-

го машинного времени оплата вычислительных ресурсов осуществлялась уже по другим расценкам.

Дисциплина обслуживания SJN предполагает, что имеется только одна очередь заданий, готовых к выполнению. Задания, которые в процессе своего исполнения были временно заблокированы (например, ожидали завершения операций ввода/вывода), вновь попадают в конец очереди готовых к выполнению наравне с вновь поступающими. Это приводит к тому, что задания, которым требуется очень немного времени для своего завершения, вынуждены ожидать процессор наравне с длительными работами, что не всегда целесообразно.

Для устранения этого недостатка и была предложена **дисциплина SRT** (Shortest Remaining Time), в соответствии с которой следующим будет выполняться задание, требующее наименьшего времени для своего завершения.

Три вышеназванные дисциплины обслуживания могут использоваться для пакетных режимов обработки, когда для пользователя не является обязательным ожидание реакции системы: он просто сдает свое задание и через несколько часов получает результаты вычислений. Для интерактивных же вычислений желательно, прежде всего, обеспечить приемлемое время реакции системы и равенство в обслуживании, если система является мультитерминальной. Если же это однопользовательская система, но с возможностью мультипрограммной обработки, то желательно, чтобы те программы, с которыми пользователь непосредственно работает в данный момент времени, имели лучшее время реакции, нежели фоновые задания. При этом возникает необходимость выполнять некоторые приложения без непосредственного участия пользователя (например, программа получения электронной почты, использующая модем и коммутируемые линии для передачи данных). Тем не менее фоновые задания гарантированно должны получать необходимую им долю процессорного времени. Для решения подобных проблем используется дисциплина обслуживания, называемая RR (Round Robin, круговая, карусельная), и приоритетные методы обслуживания.

Дисциплина обслуживания RR предполагает, что каждая задача получает процессорное время порциями (***квантами времени)***. После окончания кванта времени q задача снимается с процессора, и процессорное время передается следующей зада-

че. Снятая задача ставится в конец очереди задач, готовых к выполнению (рис. 2.3). Для оптимальной работы системы необходимо правильно выбрать закон, по которому кванты времени выделяются задачам.

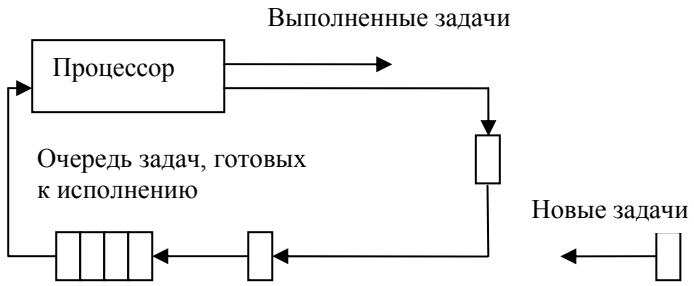

Рис. 2.3 – Дисциплина обслуживания RR

Величина кванта времени q выбирается как компромисс между приемлемым временем реакции системы на запросы пользователей и дополнительными расходами на частую смену контекста задач, с тем чтобы их простейшие запросы не вызывали длительного ожидания. Очевидно, что при прерываниях ОС вынуждена сохранять достаточно большой объем информации о текущем (прерываемом) процессе, ставить дескриптор снятой задачи в очередь, загружать контекст задачи, которая теперь будет выполняться, поскольку ее дескриптор был первым в очереди готовых к исполнению. Если величина q велика, то при увеличении очереди готовых к выполнению задач реакция системы будет ослаблена. Если же величина q мала, то относительная доля дополнительных расходов на переключения между исполняющимися задачами станет большой и это ухудшит производительность системы. В некоторых ОС есть возможность указывать в явном виде величину q либо диапазон ее возможных значений, поскольку система будет стараться выбирать оптимальное значение сама.

Дисциплина диспетчеризации RR – одна из самых распространенных дисциплин обслуживания. Однако бывают ситуации, когда ОС не поддерживает в явном виде дисциплину карусельной диспетчеризации. Например, в некоторых ОС реального времени используется диспетчер задач, работающий по принципам абсолютных приоритетов (процессор предоставля-

ется задаче с максимальным приоритетом, а при равенстве приоритетов он действует по принципу очередности) [9]. Другими словами, снять задачу с выполнения может только появление задачи с более высоким приоритетом. Поэтому если нужно организовать обслуживание задач таким образом, чтобы все они получали процессорное время равномерно и равноправно, то системный оператор может сам организовать эту дисциплину. Для этого достаточно всем пользовательским задачам присвоить одинаковые приоритеты и создать одну высокоприоритетную задачу, которая не предусматривает никаких действий по ее выполнению, но которая, тем не менее, будет по таймеру через указанные интервалы времени планироваться на выполнение. Эта задача снимет с выполнения текущее приложение, оно будет поставлено в конец очереди, и поскольку этой высокоприоритетной задаче на самом деле ничего делать не надо, то она тут же освободит процессор и из очереди готовности будет взята следующая задача. В своей простейшей реализации дисциплина карусельной диспетчеризации предполагает, что все задачи имеют одинаковый приоритет. Если же необходимо ввести механизм приоритетного обслуживания, то это, как правило, делается за счет организации нескольких очередей. Процессорное время будет предоставляться в первую очередь тем задачам, которые стоят в привилегированной очереди. Если она пуста, то диспетчер задач начнет просматривать остальные очереди. Именно по такому алгоритму действует диспетчер задач в операционных системах OS/2 и Windows NT.

Диспетчеризация задач с использованием динамических приоритетов

При выполнении программ, реализующих какие-либо задачи контроля и управления, что характерно, прежде всего, для систем реального времени, может случиться такая ситуация, когда одна или несколько задач не могут быть реализованы (решены) в течение длительного промежутка времени из-за возросшей нагрузки в вычислительной системе. Потери, связанные с невыполнением таких задач, могут оказаться больше, чем потери от невыполнения программ с более высоким приоритетом. При этом оказывается целесообразным временно изменить приори-

тет «аварийных» задач, для которых истекает отпущенное для них время обработки. После выполнения этих задач их приоритет восстанавливается. Поэтому почти в любой ОС реального времени имеются средства для изменения приоритета программ. Есть такие средства и во многих ОС, которые не относятся к классу ОС реального времени. Введение механизмов динамического изменения приоритетов позволяет реализовать более быструю реакцию системы на короткие запросы пользователей, что очень важно при интерактивной работе, но при этом гарантировать выполнение любых запросов.

Рассмотрим, например, как реализован механизм динамических приоритетов в ОС UNIX, которая, как известно, не относится к ОС реального времени. Приоритет процесса вычисляется следующим образом [10]. Во-первых, в вычислении участвуют значения двух полей дескриптора процесса – p_nice и p_cpu. Первое из них назначается пользователем явно или формируется по умолчанию с помощью системы программирования. Второе поле формируется диспетчером задач (планировщиком разделения времени) и называется системной составляющей или текущим приоритетом. Другими словами, каждый процесс имеет два атрибута приоритета, с учетом которого и распределяется между исполняющимися задачами процессорное время: **текущий приоритет**, на основании которого происходит планирование, и заказанный **относительный приоритет (**nice number или nice).

Схема нумерации (числовых значений) текущих приоритетов различна для различных версий UNIX. Например, более высокому значению текущего приоритета может соответствовать более низкий фактический приоритет планирования. Разделение между приоритетами режима ядра и задачи также зависит от версии ОС. Рассмотрим частный случай, когда текущий приоритет процесса варьируется в диапазоне от 0 (низкий приоритет) до 127 (наивысший приоритет). Процессы, выполняющиеся в режиме задачи, имеют более низкий приоритет, чем в режиме ядра. Для режима задачи приоритет меняется в диапазоне 0–65, для режима ядра – 66 –95 (системный диапазон). Процессы, приоритеты которых лежат в диапазоне 96 –127, являются процессами с фиксированным приоритетом, не изменяемым операци-

онной системой, и предназначены для поддержки приложений реального времени.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение **приоритета сна**, выбираемое ядром из диапазона системных приоритетов и связанное с событием[2], вызвавшим это состояние. Когда процесс пробуждается, ядро устанавливает значение текущего приоритета процесса равным приоритету сна. Поскольку приоритет такого процесса находится в системном диапазоне и выше, чем приоритет режима задачи, вероятность предоставления процессу вычислительных ресурсов весьма велика. Такой подход позволяет, в частности, быстро завершить системный вызов, выполнение которого, в свою очередь, может блокировать некоторые системные ресурсы.

После завершения системного вызова перед возвращением в режим задачи ядро восстанавливает приоритет режима задачи, сохраненный перед выполнением системного вызова. Это может привести к понижению приоритета, что, в свою очередь, вызовет переключение контекста.

Текущий приоритет процесса в режиме задачи p_priuser зависит от значения nice number и степени использования вычислительных ресурсов p_cpu:

$$p\_priuser = a*p\_nice - b*p\_cpu.$$

Задача планировщика разделения времени состоит в справедливом распределении вычислительного ресурса между конкурирующими процессами. Для принятия решения о выборе следующего запускаемого процесса планировщику необходима информация об использовании процессора. Эта составляющая приоритета уменьшается обработчиком прерываний таймера по каждому «тику» таймера. Таким образом, пока процесс выполняется в режиме задачи, его текущий приоритет линейно уменьшается.

Каждую секунду ядро пересчитывает текущие приоритеты готовых к запуску процессов, приоритеты которых меньше не-

---

[2] Здесь можно сказать, что процесс построения операционных систем использует событийное программирование, то есть ориентированное на события.

которого порогового значения (в нашем примере эта величина равна 65), последовательно увеличивая их. Это осуществляется за счет того, что ядро последовательно уменьшает отрицательную компоненту времени использования процессора. Как результат, эти действия приводят к перемещению процессов в более приоритетные очереди и повышают вероятность их последующего запуска.

Возможно использование следующей формулы: p_cpu = p_cpu/2.

Это правило выявляет недостаток данной дициплины диспетчеризации – нивелирование приоритетов при повышении загрузки системы. Происходит это потому, что в таком случае каждый процесс получает незначительный объем вычислительных ресурсов и, следовательно, имеет малую составляющую p_cpu, которая еще более уменьшается вследствие пересчета величины p_cpu. В результате степень использования процессора перестает оказывать заметное влияние на приоритет, и низкоприоритетные процессы (то есть процессы с высоким значением nice number) практически «отлучаются» от вычислительных ресурсов системы. В некоторых версиях ОС UNIX для пересчета значения p_cpu используется другая формула: p_cpu = =p_cpu*(2*load)/(2*load+l).

Здесь параметр load равен среднему числу процессов, находившихся в очереди на выполнение за последнюю секунду, и характеризует среднюю загрузку системы за этот период времени. Такой алгоритм позволяет частично избавиться от недостатка планирования по формуле p_cpu = p_cpu/2, поскольку при значительной загрузке системы уменьшение p_cpu при пересчете будет происходить медленнее.

Описанные алгоритмы планирования позволяют учесть интересы низкоприоритетных процессов, так как в результате длительного ожидания очереди на запуск приоритет таких процессов увеличивается, соответственно увеличивается и вероятность запуска. Эти алгоритмы также обеспечивают более вероятный выбор планировщиком интерактивных процессов по отношению к вычислительным (фоновым). Такие задачи, как командный интерпретатор или редактор, большую часть времени проводят в ожидании ввода, имея, таким образом, высокий при-

оритет (приоритет сна). При наступлении ожидаемого события (например, пользователь осуществил ввод данных) им сразу же предоставляются вычислительные ресурсы. Фоновые процессы, потребляющие значительные ресурсы процессора, имеют высокую составляющую p_cpu и, как следствие, менее высокий приоритет.

Аналогичные механизмы имеют место и в таких ОС, как OS/2 или Windows NT [2]. Правда, алгоритмы изменения приоритета задач в этих системах иные. Например, в Windows NT каждый поток (тред) имеет базовый уровень приоритета, который лежит в диапазоне от двух уровней ниже базового приоритета процесса, его породившего. Базовый приоритет процесса определяет, сколь сильно могут различаться приоритеты потоков процесса и как они соотносятся с приоритетами потоков других процессов. Поток наследует этот базовый приоритет и может изменять его так, чтобы он стал немного больше или немного меньше. В результате формируется приоритет планирования, с которым поток и начинает исполняться. В процессе исполнения потока его приоритет может отклоняться от базового.

Для определения порядка выполнения потоков диспетчер использует систему приоритетов, направляя на выполнение потоки с высоким приоритетом раньше потоков с низкими приоритетами. Система прекращает исполнение или ***вытесняет*** (preempts) текущий поток, если становится готовой к выполнению другая задача (поток) с более высоким приоритетом. Каждому уровню приоритета соответствует определенная очередь. Windows NT поддерживает 32 уровня приоритетов. Потоки делятся на два класса: ***реального времени*** **и** ***переменного приоритета***. Потоки реального времени, имеющие приоритеты от 16 до 31 – это высокоприоритетные потоки, используемые программами с критическим временем выполнения, то есть требующие немедленного внимания системы (по терминологии Microsoft).

Диспетчер задач просматривает очереди, начиная с самой приоритетной. При этом, если очередь пустая, т.е. нет готовых к выполнению задач с таким приоритетом, осуществляется переход к следующей очереди. Следовательно, если есть задачи, требующие процессор немедленно, они будут обслужены в пер-

вую очередь. Для собственно системных модулей, функционирующих в статусе задач, зарезервирована очередь с номером 0.

Большинство потоков в системе относятся к классу переменного приоритета с уровнями приоритета (номером очереди) от 1 до 15. Эти очереди используются потоками с переменным приоритетом (variable priority), так как диспетчер задач корректирует их приоритеты по мере выполнения задач для оптимизации отклика системы. Диспетчер приостанавливает исполнение текущего потока после того, как тот израсходует свой квант времени. При этом если прерванный тред является потоком переменного приоритета, то диспетчер задач понижает его приоритет на единицу и перемещает в другую очередь. Таким образом, приоритет потока, выполняющего много вычислений, постепенно понижается (до значения его базового приоритета). С другой стороны, диспетчер повышает приоритет потока после освобождения задачи (потока) из состояния ожидания. Обычно добавка к приоритету потока определяется кодом исполнительной системы, находящимся вне ядра ОС, однако величина этой добавки зависит от типа события, которого ожидал заблокированный тред. Так, например, поток, ожидавший ввода очередного байта с клавиатуры, получает большую добавку к значению своего приоритета, чем процесс ввода/вывода, работавший с дисковым накопителем. Однако в любом случае значение приоритета не может достигнуть 16.

2.2.2 Алгоритмы диспетчеризации

Диспетчеризация без перераспределения процессорного времени, то есть **невытесняющая многозадачность** (non-preemptive multitasking) – это такой способ диспетчеризации процессов, при котором активный процесс выполняется до тех пор, пока он сам, что называется «по собственной инициативе», не отдаст управление диспетчеру задач для выбора из очереди другого, готового к выполнению процесса или треда. Дисциплины обслуживания FCFS, SJN, SRT относятся к невытесняющим.

Диспетчеризация с перераспределением процессорного времени между задачами, то есть **вытесняющая многозадач-**

ность (preemptive multitasking) – это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается диспетчером задач, а не самой активной задачей [2]. При вытесняющей многозадачности механизм диспетчеризации задач целиком сосредоточен в операционной системе, и программист может писать свое приложение, не заботясь о том, как оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения текущей задачи, сохраняет ее контекст в дескрипторе задачи, выбирает из очереди готовых задач следующую и запускает ее на выполнение, предварительно загрузив ее контекст. Дисциплина RR и многие другие, построенные на ее основе, относятся к вытесняющим.

При невытесняющей многозадачности механизм распределения процессорного времени распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление супервизору ОС с помощью соответствующего системного вызова. При этом естественно, что диспетчер задач, так же как и в случае вытесняющей мультизадачности, формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом порядка поступления задач или их приоритетов) следующую задачу на выполнение. Такой механизм создает некоторые проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что на некоторый произвольный период времени, определяемый процессом выполнения приложения, «теряется» управление системой, вследствие чего не обеспечивается приемлемое время реакции на запросы пользователей [11]. Таким образом, если приложение тратит слишком много времени на выполнение какой-либо работы (например, на форматирование диска), пользователь не может переключиться с этой задачи на другую (например, на текстовый или графический редактор с условием продолжения форматирования в фоновом режиме). Эта ситуация нежелательна, так как

пользователи заинтересованы в скорейшем завершении процесса выполнения задачи.

Поэтому разработчики приложений для невытесняющей операционной среды, возлагая на себя функции диспетчера задач, должны создавать приложения таким образом, чтобы они выполняли свои задачи небольшими частями. Так, упомянутая выше программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы отформатировать следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста.

Например, в ныне уже забытой операционной среде Windows 3.x активные приложения этой системы разделяли между собой процессорное время именно таким образом. И программисты сами должны были обеспечивать «дружественное» отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая управление ядру системы. Крайним проявлением «недружественности» приложения является его зависание, которое приводит к общему краху системы. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный механизм диспетчеризации, во-первых, обеспечивает все задачи процессорным временем, а во-вторых, дает возможность иметь надежные механизмы для мониторинга вычислений и позволяет снять зависшую задачу с выполнения. Однако распределение функций диспетчеризации между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, поскольку дает возможность разработчику приложений самому проектировать алгоритм распределения процессорного времени, наиболее подходящий для данного фиксированного набора задач [11]. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в «неудобные» для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их

монопольно, на протяжении периода выполнения задачи используемые ею данные не будут изменены другой задачей. Примером эффективного использования невытесняющей многозадачности является сетевая операционная система Novell NetWare, в которой в значительной степени благодаря этому достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.x.

2.2.3 Качество диспетчеризации

Одна из проблем, которая возникает при выборе подходящей дисциплины обслуживания, – это гарантия обслуживания. Дело в том, что при некоторых дисциплинах, например при использовании дисциплины абсолютных приоритетов, низкоприоритетные процессы оказываются обделенными многими ресурсами и, прежде всего, процессорным временем. Возникает реальная дискриминация низкоприоритетных задач, и ряд таких процессов, имеющих к тому же большие потребности в ресурсах, могут очень длительное время откладываться или, в конце концов, вообще могут быть не выполнены. Известны случаи, когда вследствие высокой загрузки вычислительной системы отдельные процессы так и не были выполнены, несмотря на то, что прошло несколько лет с момента их планирования. Поэтому вопрос гарантии обслуживания является очень актуальным.

Более жестким требованием к системе, чем просто гарантированное завершение процесса, является его гарантированное завершение к указанному моменту времени или за указанный интервал времени. Существуют различные дисциплины диспетчеризации, учитывающие жесткие временные ограничения, но не существует дисциплин, которые могли бы предоставить больше процессорного времени, чем может быть в принципе выделено.

Планирование с учетом жестких временных ограничений легко реализовать, организуя очередь готовых к выполнению процессов в порядке возрастания их временных ограничений. Основным недостатком такого простого упорядочения является то, что процесс (за счет других процессов) может быть обслу-

жен быстрее, чем это ему реально необходимо. Для того чтобы избежать такой ситуации, проще всего процессорное время выделять все-таки квантами. Гарантировать обслуживание можно следующими тремя способами [2]:

1) выделять минимальную долю процессорного времени некоторому классу процессов, если, по крайней мере, один из них готов к исполнению. Например, можно отводить 20 % от каждых 10 мс процессам реального времени, 40 % от каждых 2 с – интерактивным процессам и 10 % от каждых 5 мин – пакетным (фоновым) процессам;

2) выделять минимальную долю процессорного времени некоторому конкретному процессу, если он готов к выполнению;

3) выделять столько процессорного времени некоторому процессу, чтобы он мог выполнить свои вычисления к определенному сроку.

Для сравнения алгоритмов диспетчеризации обычно используются следующие показатели [2]:

- загрузка (использование) центрального процессора (CPU uti-lization). В большинстве персональных систем средняя загрузка процессора не превышает 2–3 %, доходя в моменты выполнения сложных вычислений и до 100 %. В реальных системах, где компьютеры выполняют очень много работы, например в серверах, загрузка процессора колеблется в пределах 15–40 % для легко загруженного процессора и до 90–100 % – для сильно загруженного процессора;
- пропускная способность (CPU throughput) процессора, измеряемая количеством процессов, выполняемых в единицу времени;
- время оборота (turnaround time). Для некоторых процессов важным критерием является полное время выполнения, то есть интервал от момента появления процесса во входной очереди до момента его завершения. Это время названо временем оборота. Оно включает время ожидания во входной очереди, время ожидания в очереди готовых процессов, время ожидания в очередях к оборудованию, время выполнения в процессоре и время ввода/вывода;

- время ожидания (waiting time) – суммарное время нахождения процесса в очереди готовых процессов;
- время отклика (response time). Для интерактивных программ важным показателем является время отклика или время, прошедшее от момента попадания процесса во входную очередь до момента первого обращения к терминалу.

Очевидно, что простейшая стратегия краткосрочного планировщика должна быть направлена на максимизацию средних значений загруженности и пропускной способности, времени ожидания и времени отклика. Производительность работы системы в целом во многом зависит от рационального планирования процессов.

Можно выделить следующие главные причины, приводящие к уменьшению производительности системы:

- возникновение дополнительных затрат времени на переключение процессора. Они определяются не только переключениями контекстов задач, но и затратами времени на перемещение страниц виртуальной памяти при переключении на процессы другого приложения, а также необходимостью обновления данных в кэше (коды и данные одной задачи, находящиеся в кэше, не нужны другой задаче и должны быть заменены);
- переключение на другой процесс в тот момент, когда текущий процесс выполняет критическую секцию, а другие процессы активно ожидают входа в свою критическую секцию. В этом случае потери будут особенно велики (хотя вероятность прерывания выполнения коротких критических секций мала).

В случае использования мультипроцессорных систем применяются следующие методы повышения производительности системы:

- совместное планирование, при котором все потоки одного приложения (неблокированные) одновременно выбираются для выполнения процессорами и одновременно снимаются с них (для сокращения переключений контекста);
- планирование, при котором находящиеся в критической секции задачи не прерываются, а активно ожидающие входа в критическую секцию – не выбираются до тех пор, пока вход в секцию не освободится;

- планирование с учетом так называемых «советов» программы во время ее выполнения. Например, в известной своими новациями ОС Mach имелись два класса таких советов (hints) [2]: указания разной степени категоричности о снятии текущего процесса с процессора, а также указания о процессе, который должен быть выбран взамен текущего.

Вопросы для самопроверки

1. Перечислите и поясните основные функции ОС, которые связаны с управлением задачами.
2. Какие стратегии диспетчеризации вы знаете?
3. Какие дисциплины диспетчеризации задач вы знаете? Опишите их.
4. В чем отличие алгоритмов диспетчеризации с вытесняющей и невытесняющей многозадачностью?
5. Сравните механизмы диспетчеризации задач в ОС Windows NT и UNIX. В чем заключаются основные различия?
6. Опишите оценки качества диспетчеризации.

3. УПРАВЛЕНИЕ ПАМЯТЬЮ

3.1 Основные понятия

3.1.1 Фон-неймановская архитектура вычислительных машин

Любой IBM PC-совместимый компьютер представляет собой реализацию так называемой фон-неймановской архитектуры вычислительных машин. Машина состоит из блока управления, арифметико-логического устройства (АЛУ), памяти и устройств ввода-вывода. В ней реализуются следующие принципы организации архитектуры:

- концепция хранимой программы: программы и данные хранятся в одной и той же памяти;
- последовательная передача управления. Выполняемые действия определяются блоком управления и АЛУ, которые вместе являются основой центрального процессора. Центральный процессор выбирает и исполняет команды из памяти последовательно, адрес очередной команды задается «счетчиком адреса» в блоке управления.

Фон-неймановская архитектура – это не единственный вариант построения ЭВМ, есть и другие, которые не соответствуют указанным принципам (например, потоковые машины). Однако подавляющее большинство современных компьютеров основаны именно на указанных принципах, включая и сложные многопроцессорные комплексы, которые можно рассматривать как объединение фон-неймановских машин.

Процессор имеет набор регистров, часть которых доступна для хранения операндов, выполнения действий над ними и формирования адреса инструкций и операндов в памяти. Другая часть регистров используется процессором для служебных (системных) целей, доступ к ним может быть ограничен (есть даже программно-невидимые регистры). Все компоненты компьютера представляются для процессора в виде наборов ячеек памяти или/и портов ввода-вывода, в которые процессор может производить запись и/или считывать содержимое.

3.1.2 Биты, байты, слова, параграфы

Компьютер работает в двоичной системе исчисления – минимальным информационным элементом является **бит**, который может принимать значение 0 или 1. Этим значениям соответствуют различимые физические состояния ячейки, чаще всего – уровень напряжения (низкий или высокий). Биты организуются в более крупные образования – ячейки памяти и регистры. Каждая ячейка памяти (регистр) имеет свой **адрес**, однозначно ее идентифицирующий в определенной системе координат. Минимальной адресуемой (пересылаемой между компонентами компьютера) единицей информации является **байт**, состоящий, как правило, из 8 бит[3]. Два байта со смежными адресами образуют **слово** (word) разрядностью 16 бит, два смежных слова образуют **двойное слово** (double word) – 32 бита, два смежных двойных слова образуют **учетверенное слово** (quad word) – 64 бита.

В двухбайтном слове принят ***LH-порядок следования байт:*** адрес слова указывает на младший байт L (Low), а старший байт H (High) размещается по адресу, на единицу большему. В двойном слове порядок будет аналогичным – адрес укажет на самый младший байт, после которого будут размещены следующие по старшинству. Этот порядок, естественный для процессоров Intel, применяется не во всех микропроцессорных семействах. Байт (8 бит) делится на пару тетрад (nible): старшую тетраду – биты (7:4) и младшую тетраду – биты (3:0).

В технической документации, электрических схемах и текстах программ могут применяться разные способы представления чисел:

---

[3] Существуют процессоры и компьютеры с разрядностью обрабатываемого слова, не кратной 8 (например, 5, 7, 9, ...), и их байты не восьмибитные, но в мире PC столкновение с ними маловероятно. Также в некоторых системах (обычно коммуникационных) совокупность восьми соседних бит данных называют октетом. Название «октет» обычно подразумевает, что эти 8 бит не имеют явного адреса, а характеризуются только своим местоположением в длинной цепочке бит.

двоичные (binary) ***числа*** – каждая цифра означает значение одного бита (0 или 1), старший бит всегда пишется слева, после числа ставится буква «b». Для удобства восприятия тетрады могут быть разделены пробелами. Например, 1010 0101b;

шестнадцатеричные (hexadecimal) ***числа*** – каждая тетрада представляется одним символом 0...9, A, B, ..., F. Обозначаться такое представление может по-разному. В данном пособии используется символ «h» после последней шестнадцатеричной цифры (например, A5h). В текстах программ это же число может обозначаться и как 0xA5, и как 0A5h, в зависимости от синтаксиса языка программирования. Незначащий ноль (0) добавляется слева от старшей шестнадцатеричной цифры, изображаемой буквой, чтобы различать числа и символические имена;

десятичные (decimal) ***числа*** – каждый байт (слово, двойное слово) представляется обычным числом, а признак десятичного представления (букву «d») обычно опускают. Байт из предыдущих примеров имеет десятичное значение 165. В отличие от двоичной и шестнадцатеричной формы записи по десятичной трудно в уме определить значение каждого бита, что иногда приходится делать;

восьмеричные (octal) ***числа*** – каждая тройка бит (разделение начинается с младшего) записывается в виде цифры 0–7, в конце ставится признак «o». То же самое число будет записано как 245o. Восьмеричная система неудобна тем, что байт нельзя разделить поровну, но зато все цифры – привычные.

В табл. 3.1 приведены разные представления одной тетрады (4 бит). Чтобы перевести любое 16-битное число в десятичное, нужно десятичный эквивалент старшей тетрады умножить на 16 и сложить с эквивалентом младшей тетрады. Приведем пример: A5h = 10 * l6 + 5 = 165.

Обратный перевод тоже несложен: десятичное число делится на 16, целая часть даст значение старшей тетрады, остаток – младшей.

Таблица 3.1 – Представление двоичных чисел в разных системах счисления

Двоичное	Шестнадцатеричное	Десятичное	Восьмеричное
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	10
1001	9	9	11
1010	A	10	12
1011	B	11	13
1100	C	12	14
1101	D	13	15
1110	E	14	16
1111	F	15	17

3.1.3 Ячейки памяти, порты и регистры

Кратко опишем разницу между ячейками памяти, портами и регистрами. **Ячейки памяти** служат лишь для хранения информации. Сначала информацию записывают в ячейку, а потом могут прочитать, а также записать иную информацию. **Порты ввода-вывода**, как правило, служат для преобразования двоичной информации в какие-либо физические сигналы и обратно. Например, порт данных параллельного интерфейса формирует электрические сигналы на разъеме, к которому обычно подключают принтер. Порт состояния того же интерфейса электрические сигналы, поступающие от принтера, отображает в виде набора бит, который может быть считан процессором. **Регистр** – довольно широкое понятие, которое зачастую используется как синоним порта.

Каждый байт (ячейка памяти, порт) имеет собственный уникальный **физический адрес** [12], устанавливаемый на системной шине процессором при инициировании обращения к данной ячейке или порту. В семействе x86 и PC-совместимых компьютерах пространства адресов ячееек памяти и портов ввода-вывода разделены. Это предусмотрено с обеих сторон: процессоры позволяют, а компьютеры используют данное разделение. Нынешние процессоры имеют разрядность физического адреса памяти 32 и даже 36 бит, что позволяет адресовать до 4 и 64 Гбайт соответственно. Пространство ввода-вывода использует только младшие 16 бит адреса, что позволяет адресовать до 65384 однобайтных регистров. Адреса «исторических» системных устройств PC не изменились с самого рождения – это дань совместимости, которая без разделения пространств вряд ли просуществовала бы столько лет. Пространства памяти и портов ввода-вывода неравнозначны не только по объему, но и по способам обращения. Способов адресации к ячейке памяти в x86 великое множество, в то время как для адресации ввода-вывода их только два. К памяти возможна (и широко используется) виртуальная адресация, при которой для программиста, программы и даже пользователя создается иллюзия гигантского размера оперативной памяти. К портам ввода-вывода обращаются только по реальным адресам, правда, и здесь возможна виртуализация, но уже чисто программными средствами операционной системы. И, наконец, самое существенное различие пространств памяти и портов ввода-вывода: процессор может считывать инструкции для исполнения только из пространства памяти; через порт ввода можно считать фрагмент программного кода, что и происходит, например, при считывании данных с диска, но для того чтобы этот код исполнить, его необходимо записать в память.

Регистры различных устройств могут быть приписаны как к пространству портов ввода-вывода, так и к пространству памяти [12]. Под портом устройства, как правило, подразумевают регистр, связанный с этим устройством и приписанный к пространству портов ввода-вывода. Точность вышеприведенной терминологии, конечно же, относительна. Так, к примеру, ячейки видеопамяти (тоже память!) служат в основном не для хранения информации, а для управления свечением элементов экрана.

3.1.4 Адресация ячеек памяти в реальном режиме

В «наследство» от процессоров 8086/88 достался своеобразный способ задания адреса ячейки памяти в виде указателя «seg:offset», состоящего из двух слов: **сегмента** (seg – segment) и **смещения** (offset). Такая запись предполагает вычисление полного адреса по формуле

$$Addr = 16 * seg + offset.$$

Такое представление 20-битного адреса двумя 16-битными числами в процессорах 8086/88 поддерживается и в реальном режиме всех последующих процессоров x86. Здесь сегмент указывает адрес **параграфа** – 16-байтной области памяти. Выравнивание адреса по границе параграфа означает, что он кратен 16 (4 младших бита нулевые). Нетрудно заметить, что один и тот же адрес можно задавать разными сочетаниями этих двух компонентов. Например, адрес начала области данных BIOS (Base Input Output System) 00400h представляют как 0000:0400, так и 0040:0000 (шестнадцатеричное представление подразумевается). Возможны и другие варианты, но их не используют.

Обозначение и порядок бит и байт шин адреса и данных, принятое в аппаратуре PC, пришло от процессоров Intel 8086/88 (и даже от 8080). Самый младший бит ***LSB*** (Least Significant Bit) имеет номер 0, старший бит ***MSB*** (Most Significant Bit) байта – 7, слова – 15, двойного слова – 31. На рисунках принято старший бит изображать слева, а младший – справа.

3.1.5 Подсистемы памяти и хранения данных

Память компьютера предназначена для кратковременного и долговременного хранения информации – кодов команд и данных. В памяти информация хранится в массиве ячеек. Минимальной адресуемой единицей является байт – каждый байт памяти имеет свой уникальный адрес. Память можно рассматривать как иерархическую систему, простирающуюся от кэш-памяти процессора до ленточных архивов.

Со времени появления больших по размерам компьютеров сложилось деление памяти на внутреннюю и внешнюю. Под внутренней подразумевалась память, расположенная внутри

процессорного «шкафа» (или плотно к нему примыкающая). Сюда входила и электронная, и магнитная память (на магнитных сердечниках). Внешняя память представляла собой отдельные устройства с подвижными носителями – накопителями на магнитных дисках (а ранее, барабанах) и ленте. Со временем все устройства компьютера удалось «поселить» в один небольшой корпус, и прежнюю классификацию памяти применительно к PC можно переформулировать следующим образом [12]:

внутренняя память – электронная (полупроводниковая) память, устанавливаемая на системной плате или на платах расширения;

внешняя память – память, реализованная в виде устройств с различными принципами хранения информации, чаще всего с подвижными носителями. В настоящее время сюда входят устройства магнитной (дисковой и ленточной) памяти, оптической и магнитооптической памяти; устройства внешней памяти могут размещаться как в системном блоке компьютера, так и в отдельных корпусах, достигающих иногда размеров небольшого шкафа.

Для процессора непосредственно доступной является внутренняя память, доступ к которой осуществляется по адресу, заданному программой. Для внутренней памяти характерен одномерный (линейный) адрес, который представляет собой одно двоичное число определенной разрядности. Внутренняя память подразделяется на *оперативную,* информация в которой может изменяться процессором в любой момент времени, и *постоянную,* информацию которой процессор может только считывать. Обращение к ячейкам оперативной памяти может происходить в любом порядке как по чтению, так и по записи, поэтому оперативную память называют памятью с произвольным доступом RAM (Random Access Memory) – в отличие от постоянной памяти ROM (Read Only Memory).

3.1.6 Стек

Стек представляет собой непрерывную область памяти, адресуемую регистрами ESP (указатель стека) и SS (селектор сегмента стека) [12, 13]. Особенность стека заключается в том, что

данные в него помещаются и извлекаются по принципу – «первым вошел – последним вышел». Данные помещаются в стек с помощью инструкции PUSH (заталкивание), а извлекаются по инструкции POP (вытаскивание).

Кроме явного доступа к стеку с помощью инструкций PUSH и POP, стек автоматически используется процессором при выполнении инструкций вызова, возврата, входа и выхода из процедур, а также при обработке прерываний. Стек используют для разных целей:

- организации прерываний, вызовов и возвратов;
- временного хранения данных, когда для них нет смысла выделять фиксированные места в памяти;
- передачи и возвращения параметров при вызовах процедур.

Перед использованием стек должен быть инициализирован так, чтобы регистры SS:ESP указывали на область реальной оперативной памяти (стек в ПЗУ, естественно, работать не может). Прикладные программы получают от операционной системы, как правило, готовый к употреблению стек. В защищенном режиме сегмент состояния задачи содержит четыре селектора сегментов стека для разных уровней привилегий, но в каждый момент используется, естественно, только один стек.

3.2 Распределение оперативной памяти

3.2.1 Распределение оперативной памяти в MS DOS

Логическая структура памяти персональных компьютеров обусловлена особенностями системы адресации процессоров семейства x86. Процессоры 8086/88, применявшиеся в первых моделях PC, имели доступное адресное пространство 1 Мбайт (20 бит шины адреса). Эти процессоры использовали сегментную модель памяти, унаследованную и следующими моделями в реальном режиме. Согласно этой модели исполнительный (линейный) адрес вычисляется по формуле, приведенной в п. 3.1.4. Таким образом обеспечивался доступ к адресному пространству Addr = 00000 – FFFFFh при помощи пары 16-битных регистров. Заметим, что при Seg = FFFFh и Offset = FFFFh данная формула

дает адрес 10FFEFh, но ввиду 20-битного ограничения на шину адреса эта комбинация в физической памяти указывает на 0FFEFh. Таким образом, адресное пространство как бы сворачивается в кольцо с небольшим «нахлестом». Начиная с процессора 80286, шина адреса была расширена до 24 бит, а впоследствии (в процессорах 386DX, 486 и выше) до 32 и даже 36 (P6[4]). В реальном режиме процессора, используемом в DOS, применяется та же сегментная модель памяти и формально доступен лишь 1 Мбайт памяти, что является недостаточным для большинства современных приложений. Однако выяснилось, что процессоры 80286 в реальном режиме эмулируют 8086 с ошибкой: та самая единица в бите A20, которая отбрасывалась в процессорах 8086/88, теперь попадает на шину адреса, и в результате максимально доступный линейный адрес в реальном режиме достиг 10FFEFh. Дополнительные байты оперативной памяти (64Кб – 16 б), адресуемой в реальном режиме, позволили освободить дефицитное пространство оперативной памяти для прикладных программ. В эту область (100000h – 10FFEFh), названную **высокой памятью HMA** (High Memory Area), стали помещать часть операционной системы и небольшие резидентные программы. Однако для обеспечения полной совместимости с процессором 8086/88 в схему PC ввели вентиль линии A20 шины адреса – *GateA20,* который либо пропускает сигнал от процессора, либо принудительно обнуляет линию A20 системной шины адреса. Более старшие биты такой «заботы» не требуют, поскольку переполнение при суммировании 16-битных компонентов адреса по данной схеме до них не распространяется. Управление этим вентилем подключили к свободному программно-управляемому выходному биту 1 контроллера клавиатуры 8042, ставшего стандартным элементом архитектуры PC, начиная с AT. Предполагалось, что этим вентилем часто пользоваться не придется. Однако жизнь внесла свои поправки, и оказалось, что переключение вентиля в многозадачных ОС, часто переключающих процессор между защищенным режимом (см. п. 3.4.2), реальным режимом и режимом V86, контроллером клавиатуры выполняется слишком медленно. Так появились альтернативные

[4] P6 – компьютеры, основанные на процессоре Pentium Pro и выше.

методы быстрого переключения вентиля, специфичные для различных реализаций системных плат (например, через порт 92h). Кроме того, иногда использовали и аппаратную логику быстрого декодирования команды на переключение бита, поступающую к контроллеру клавиатуры. Для определения способа переключения в утилиту CMOS Setup ввели соответствующие параметры, позволяющие выбрать между стандартным, но медленным способом и менее стандартизованным, но быстрым, в зависимости от используемого ПО.

32-разрядные процессоры позволяют организовать режим, иногда называемый «нереальным» или «большим реальным», в котором инструкции выполняются как в реальном, но доступны все 4 Гбайт памяти. Этот режим часто используется в игровых программах, целиком захватывающих все ресурсы компьютера, не заботясь о «правилах хорошего тона» по отношению к другим исполняемым программам.

Основную часть адресного пространства занимает оперативная память. Объем установленной памяти определяется тестом POST при начальном включении (перезагрузке) компьютера, начиная с младших адресов. Натолкнувшись на отсутствие памяти (ошибку), тест останавливается на достигнутом и сообщает системе объем реально работающей памяти.

Распределение памяти PC, непосредственно адресуемой процессором, представляется следующим образом [12, 14] (рис. 3.1).

00000h-9FFFFh – **стандартная (базовая) память** (Conventional (Base) Memory) объемом 640 Кбайт – доступная DOS и программам реального режима. В некоторых системах с видеоадаптером MDA верхняя граница сдвигается к AFFFFh (704 Кбайт). Иногда верхние 128 Кбайт стандартной памяти (область 80000h-9FFFFh) называют **Extended Conventional Memory**.

A0000h-FFFFFh – **верхняя память UMA** (Upper Memory Area) объемом 384 Кбайт, зарезервированная для системных нужд. В ней размещаются области буферной памяти адаптеров (например, видеопамять) и постоянная память (BIOS с расширениями). Эта область, обычно используемая не в полном объеме, ставит непреодолимый архитектурный барьер на пути непре-

рывной (нефрагментированной) памяти, о которой мечтают программисты.

Память выше 100000h – **дополнительная (расширенная) память Extended Memory,** непосредственно доступная только в защищенном (и в «большом реальном») режиме для компьютеров с процессорами 286 и выше. В ней выделяется область 100000h-10FFEFh – высокая память HMA – это единственная область расширенной памяти, доступная 286+ в реальном режиме при открытом вентиле *Gate A20.*

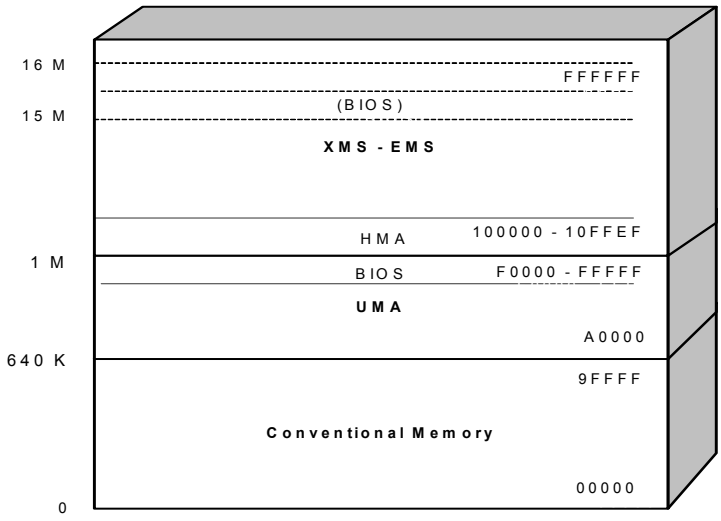

Рис. 3.1 – Распределение памяти ПК

Область памяти выше первого мегабайта в различных источниках называется по-разному. Ее современное английское название «**Extended Memory»** пересекается с названием одной из спецификаций ее использования «**Extended Memory Specification»**. Но название другой спецификации использования «**Expanded Memory Specification»** в прямом переводе на русский язык неотличимо от перевода предыдущего термина (и Extended, и Expanded переводятся как «расширенный»). Будем придерживаться терминологии, укрепившейся в литературе, и область всей физической памяти, расположенной в адресном пространстве выше 1 Мбайта, будем называть дополнительной

памятью. Ее объем у современных компьютеров указывается строкой Extended Memory xxxxx Kbyte в таблице, выводимой после прохождения теста POST, и в меню стандартной конфигурации CMOS Setup.

Вышеприведенное разделение памяти актуально только для приложений и ОС реального режима типа MS DOS. Для ОС защищенного режима (в том числе Windows 9x/NT/2000) доступна вся оперативная память, причем без каких-либо ухищрений вроде EMS и XMS, описанных ниже. Однако область UMA, сохраняемая ради совместимости, остается барьером на пути к единой однородной памяти.

Для компьютеров класса AT-286 с 24-битной шиной адреса верхняя граница оперативной памяти – FDFFFFh (максимальный размер 15,9 Мбайт). Область FE0000h-FFFFFFh содержит ПЗУ BIOS (ROM BIOS Area), обращение к этой области эквивалентно обращению к ROM BIOS по адресам 0E0000h-0FFFFFh. Для процессоров 386+ и 32-битной шины адреса теоретическая верхняя граница – 4 Гбайт, а для P6 – 64 Гбайт (36-битная шина адреса). В компьютерах с 32-разрядной шиной адреса образ BIOS дополнительно проецируется в адреса FFFE0000h-FFFFFFFFh, хотя для процессоров P6 это и необязательно. Однако иногда используется и проекция BIOS в область FE0000h-FFFFFFh, что не позволяет задействовать более 16 Мбайт ОЗУ, поскольку система воспринимает только найденную непрерывную область оперативной памяти. Если 32-разрядный компьютер имеет отображение области BIOS под границей 16 Мбайт, это отображение обычно можно запретить установкой соответствующего параметра CMOS Setup. Иногда для использования специфических адаптеров ISA, имеющих буфер с адресами в 16-м мегабайте памяти, предусматривают параметр Memory Hole At 15–16 Мб+. Его установка также не позволяет использовать оперативную память свыше 16 Мбайт.

Реально современные системные платы позволяют установить до 512–2048 Мбайт ОЗУ для мощных серверных платформ, и это не предел. Обращение по адресам, превышающим границу установленной оперативной памяти (или максимально возможного объема), транслируется на шину PCI, которая имеет 32-битную адресацию.

Компьютеры, использующие режим системного управления *SMM* (System Management Mode), имеющийся у большинства процессоров последних поколений, имеют еще одно адресное пространство памяти – *SMRAM.* Это адресное пространство «параллельно» пространству обычной памяти и при работе доступно процессору только в режиме обработки SMI[5]. Память SMRAM может представлять собой часть физической оперативной памяти DRAM (Dynamic Random Access Memory), а может быть реализована и специальной микросхемой энергонезависимой памяти, размер которой варьируется в диапазоне от 32 Кбайт (минимальные потребности SMM) до 4 Гбайт. SMRAM располагается, начиная с адреса SMIBASE (по умолчанию 30000h), и распределяется относительно адреса SMIBASE следующим образом:

- область сохранения контекста FE00h-FFFFh (3FE00h-3FFFFh) – начиная со старших адресов по направлению к младшим. По прерыванию SMI сохраняются почти все регистры процессора, но сохранение регистров FPU не производится;
- точка входа в обработчик (SMI Handler) – 8000h (38000h);
- свободная область – 0-7FFFh (30000h-37FFFh).

Память SMRAM должна быть схемотехнически защищена от доступа прикладных программ. Процессор генерирует специальный выходной сигнал SMIACT# во время обработки SMI, который и должен являться «ключом» доступа к этой памяти. Если SMRAM не является энергонезависимой, то системная логика должна обеспечить возможность ее инициализации (записи программного кода обработчика) процессором из обычного режима работы до разрешения появления сигнала SMI#.

3.2.1.1 Стандартная память – Conventional memory

При работе в среде операционных систем типа MS DOS стандартная память является самой дефицитной в PC. На ее не-

[5] SMI# (System Management Interrupt). В режим SMM процессор может войти только по сигналу на входе SMI#. Сигнал SMI# для процессора является немаскируемым прерыванием с наивысшим приоритетом.

большой объем (типовое значение 640 Кбайт) претендуют и BIOS, и ОС реального режима, а оставшаяся часть предназначена для прикладного ПО. Стандартная память распределяется следующим образом [12]:

00000h-003FFh – *Interrupt Vectors* – векторы прерываний (256 двойных слов);

00400h-004FFh – *BIOS Data Area* – область переменных BIOS;

00500h-00xxxh – *DOS Area* – область DOS;

00xxxh-9FFFFh – *User RAM* – память, предоставляемая пользователю (до 638 Кбайт); при использовании PS/2 Mouse область 9FC00h-9FFFFh используется как расширение BIOS Data Area, и размер User RAM уменьшается.

Управление пользовательской памятью осуществляется с использованием специализированных структур: таблицы таблиц – list of list (таблица 3.2) и упраляющих блоков памяти – memory control block (таблица 3.3). Данные структуры являются недокументированными.

Таблица 3.2 – Струтура таблицы таблиц

Смещение	Длина	Содержимое
–2	2	сегм. адр. 1 MCB – memory control block
0	4	указ. на 1 DPB – disk parametr blokout
+ 4	4	указ. на список таблиц открытых файлов
+ 8	4	указ. на первый драйвер DOS (CLOCK$)
…	…	…

Таблица 3.3 – Струтура блока управления памятью

Смещение	Длина	Содержимое
+0	1	'M'(4dH) – за этим блоком есть еще блоки 'Z'(5aH) – данный блок является последним
+1	2	Владелец; параграф владельца (для FreeMem); 0 = владеет собой
+3	2	Размер, число параграфов в этом блоке распределения
+5	0Bh	Зарезервировано
+10h	?	Блок памяти начинается здесь и имеет длину (Размер*10H) байт

ЗАМЕЧАНИЯ:

- блоки памяти всегда выровнены на границу параграфа («сегмент блока»);
- блоки M-типа: следующий блок находится по (сегмент блока + Размер):0000;
- блоки Z-типа: (сегмент блока + Размер):0000 = конец памяти (a000H=640K).

В любом MCB указан его владелец – сегментный адрес PSP программы владельца данного блока памяти. А в PSP есть ссылка на окружение данной программы, в котором можно найти имя программы – путь ее запуска.

Следует помнить, что сама программа (и PSP в том числе) и ее окружение сами располагаются в блоках памяти. Поэтому в MCB блока памяти самой программы в качестве хозяина указан собственный адрес самого себя.

Когда программа в реальном режиме начинает выполнение, DS:0000 и ES:0000 указывают на начало PSP этой программы. Информация PSP позволяет выделить имена файлов и опции из строки команд, узнать объем доступной памяти, определить окружение и т.д.

Использование окружения. Окружение не превышает 32Кб и начинается на границе параграфа. Смещение 2cH в PSP текущей программы содержит номер параграфа окружения.

Вы можете найти нужное 'имя' серией сравнений строк ASCIIZ (Строка ASCIIZ, используемая во многих функциях DOS и в языке C, представляет собой последовательность символов ASCII, заканчивающуюся байтом 00H), пока не дойдете до пустой строки (нулевой длины), что указывает конец окружения. Обычно 'имя' в каждой строке окружения задано прописными буквами, но это необязательно.

Одна типичная операция с окружением используется программами типа оболочки, которые запускают вторичную копию COMMAND.COM. Такие программы обычно ищут 'имя' «COMSPEC» и используют соответствующее 'значение' как полный путь интерпретатора команд DOS – программы, запускаемой через функцию DOS 4bH .

Некоторые программы требуют, чтобы оператор поместил информацию для приложения в окружение посредством команды SET. Приложение может использовать такую информацию при каждом выполнении. Например, текстовый процессор может отыскивать в окружении 'имя' «DICTIONARY» и использовать соответствующее 'значение' как имя файла со словарными данными.

Более подробную информацию о структурах памяти можно получить из электронного справочника TECH Help!

3.2.1.2 Верхняя память – UMA

Верхняя память имеет области различного назначения, которые могут быть заполнены буферной памятью адаптеров, постоянной памятью или оставаться незаполненными. Раньше эти «дыры» не использовали из-за сложности «фигурного выпиливания» адресуемого пространства. С появлением механизма страничной переадресации (у процессоров 386 и выше) их стали по возможности заполнять «островками» оперативной памяти, названными блоками верхней памяти ***UMB*** (Upper Memory Block). Эти области доступны DOS для размещения резидентных программ и драйверов через драйвер EMM386, который отображает в них доступную дополнительную память. Стандартное распределение верхней памяти выглядит следующим образом [12]:

A0000h-BFFFFh – *Video RAM* (128 Кбайт) – видеопамять (обычно используется не полностью);

C0000h-DFFFFh – *Adapter ROM, Adapter RAM* (128 Кбайт) – резерв для адаптеров, использующих собственные модули ROM BIOS или/и специальное ОЗУ, совместно используемое с системной шиной;

E0000h-EFFFFh – свободная область (64 Кбайт), иногда занятая под System BIOS;

F0000h-FFFFFh – *System BIOS* (64 Кбайт) – системная BIOS;

FD000h-FDFFFh – *ESCD* (Extended System Configuration Data) – область энергонезависимой памяти, используемая для конфигурирования устройств Plug and Play. Эта область имеется

только при наличии PnP BIOS, ее положение и размер жестко не заданы.

В области UMA практически всегда присутствует графический адаптер. В зависимости от модели он занимает следующие области:

- MDA RAM – B0000h-B0FFFh;
- CGA RAM – B8000h-BBFFFh;
- EGA ROM – C0000h-C3FFFh/C7FFFh;
- VGA ROM – C0000h-C7FFFh;
- EGA, VGA RAM – A0000h-BFFFFh, в зависимости от видеорежима используются следующие области:
    - Graphics – A0000h-AFFFFh;
    - Color Text – B8000h-BFFFFh;
    - Mono Text – B0000h-B7FFFh.

Распространенным потребителем UMA являются также расширения ROM BIOS, расположенные на платах дисковых контроллеров, и микросхемы удаленной загрузки (Boot ROM) на платах адаптеров ЛВС. Обычно они занимают область C8000h – CBFFFh/C9FFFh/C8FFFh (для дисковых контроллеров), но могут и перемещаться при конфигурировании адаптеров.

Размер области, занимаемой системной ROM BIOS, колеблется от 8 Кбайт у PC/XT до 128 Кбайт, однако разумное значение – 64 Кбайт. Большая область высвободилась с появлением микросхем ROM и флэш-памяти объемом 1 Мбит (128К*8), но при этом размер доступной UMA сократился. Тогда микросхемы того же и большего объема стали отображать только на область F0000h-FFFFFh (64 Кбайт), а иногда и меньшую. Это оказалось возможным, так как не все содержимое микросхемы ROM BIOS должно быть доступно одновременно. Таким способом удалось примирить интересы пользователей UMB с необходимостью расширения объема BIOS, связанной с усложнением технических средств.

Видеопамять графического адаптера является особой областью памяти, к которой во время непрерывного процесса регенерации экрана интенсивно обращаются и центральный процессор, и графический акселератор (если таковой имеется). Видеопамять традиционно является физически выделенной памятью сравнительно (по сравнению с ОЗУ) небольшого объема, и для

нее разными способами обеспечивают максимальную производительность – увеличивают разрядность до 128 бит, повышают частоту, применяют специализированные, в том числе и двухпортовые, микросхемы памяти. Это, конечно же, приводит к удорожанию компьютера. Для современных графических акселераторов требуется доступ к большому объему памяти, причем с высокой производительностью. Вместо предоставления локальной памяти адаптера была предложена ***архитектура унифицированной памяти UMA*** (Unified Memory Architecture). Здесь для видеопамяти и других нужд акселератора выделяется область в общем пространстве единой физической оперативной памяти, что снижает ее стоимость, но одновременно уменьшается и производительность как видеосистемы, так и основной памяти. Архитектура UMA применяется в чипсетах системной платы с интегрированной графикой для недорогих компьютеров. При этом может предоставляться возможность установки и дополнительного специализированного модуля видеопамяти, что позволяет отказаться от UMA, но требует дополнительных денежных средств. Если с графического адаптера AGP (Accelerated Graphic Port, ускоренный графический порт) убрать локальную память, этот высокопроизводительный адаптер вырождается в систему с UMA.

3.2.1.3 Отображаемая и расширенная память – спецификации EMS и XMS

Отображаемая память EMS (Expanded Memory Specification) – программная спецификация использования дополнительной памяти DOS-программами реального режима. Спецификация *LIM EMS* – продукт соглашения фирм Lotus, Intel, Microsoft на использование EMS. С помощью специальных аппаратных или программных средств любая область дополнительной памяти может быть отображена на небольшие страницы, расположенные в области UMA. В первоначальном варианте можно было использовать 4 страницы по 16 Кбайт, примыкающие друг к другу, начиная обычно с адреса D0000h (положение страниц можно менять в пределах свободных областей UMA). Обращение прикладных программ к памяти EMS осуще-

ствляется через диспетчер памяти, вызываемый по прерыванию Int 67h. Программа, нуждающаяся в дополнительной памяти, должна сначала запросить выделение области, указав ее размер в 16-килобайтных страницах. В ответ на этот запрос, если имеется свободная память, диспетчер сообщает программе номер дескриптора EMS (EMS handler), по которому программа в дальнейшем будет ссылаться на выделенную ей область при управлении отображением. Далее программа через диспетчер назначает отображение требуемой логической страницы из выделенной ей области дополнительной памяти на выбранную физическую страницу, расположенную в области UMA. После этого любые программные обращения процессора к физической странице, расположенной в пределах первого мегабайта, будут в действительности работать с логической страницей дополнительной памяти, расположенной выше первого мегабайта, причем без переключения в защищенный режим. Для работы с иной логической страницей требуется вызов диспетчера для переназначения отображения. В EMS 4.0, эмулируемой на процессорах 386+, появилась возможность увеличения числа доступных физических страниц и отображения дополнительной памяти не только на фиксированные области UMA, но и на любые области памяти.

Для поддержки EMS поначалу требовались специальные аппаратные средства. В компьютерах на процессорах 386 и выше появилась возможность программной эмуляции EMS, которую в MS DOS 5+ выполняет драйвер EMM386.EXE.

Система EMS в основном предназначена для хранения данных. Для исполняемого в данный момент программного кода она неудобна, поскольку требует программного переключения страниц через каждые 16 Кбайт. EMS использовалась фирмой Lotus для хранения больших электронных таблиц. Ее используют для создания виртуальных дисков, хранения очередей заданий для печати, а также и для хранения данных и даже программного кода некоторых резидентных программ в целях экономии стандартной памяти.

Расширенная память XMS (Extended Memory Specification) – иная программная спецификация использования дополнительной памяти DOS-программами, разработанная компаниями

Lotus, Intel, Microsoft и AST для компьютеров на процессорах 286 и выше. Эта спецификация позволяет программе получить в распоряжение одну или несколько областей дополнительной памяти, а также использовать область HMA. Распределением областей ведает драйвер HIMEM.SYS. Драйвер позволяет захватывать или освобождать область HMA (65520 байт, начиная с 100000h), а также управлять вентилем линии адреса A20. Функции XMS позволяют программе:

- определить размер максимального доступного блока памяти;
- захватить или освободить блок памяти;
- копировать данные из одного блока в другой, причем участники копирования могут быть блоками как стандартной, так и дополнительной памяти в любых сочетаниях;
- запереть блок памяти (запретить копирование) и отпереть его;
- изменить размер выделенного блока.

В ответ на запрос выделения области драйвер выдает номер дескриптора блока (16-битное число XMS handler), по которому выполняются дальнейшие манипуляции с этим блоком. Размер блока может достигать 64 Мбайт. Спецификация XMS позволяет программам реального режима устраивать «склады» данных в дополнительной памяти, которая им непосредственно недоступна, копируя в нее и из нее данные доступных областей первого мегабайта памяти. Доступ к драйверу XMS осуществляется через прерывание Int 2Fh. Заботу о переключении в защищенный режим и обратно для получения доступа к дополнительной памяти берет на себя диспетчер. По умолчанию HIMEM.SYS позволяет использовать до 32 дескрипторов блоков, но это число можно увеличить, задав параметр /NUMHANDLES=xx в строке загрузки драйвера HIMEM.SYS. Кроме работы с дополнительной памятью, спецификация XMS определяет две функции и для работы с блоками UMB – захватить блок требуемого размера (или определить максимально доступный блок) и освободить его.

Как видно, спецификации EMS и XMS отличаются по принципу действия [12]: в EMS для доступа к дополнительной памяти выполняется отображение памяти (страничная переад-

ресация), а в XMS – копирование блоков данных. На компьютерах с процессорами 386+ эти спецификации мирно сосуществуют при использовании драйвера HIMEM.SYS, поверх которого может быть загружен и драйвер EMM386.EXE, пользующийся памятью XMS для эмуляции EMS-памяти. Память, доступная EMS и XMS, может выделяться динамически из числа дополнительной. Ключ NOEMS в строке запуска EMM386 запрещает выделение памяти под использование по спецификации EMS.

3.2.1.4 Теневая память – Shadow ROM и Shadow RAM

В области верхней памяти UMA обычно располагаются устройства с медленной памятью [12]: системная BIOS (System ROM BIOS); расширения BIOS на графическом адаптере (Video ROM BIOS), на контроллерах дисков и интерфейсов (Adapter ROM); ПЗУ начальной загрузки на сетевой карте (Boot ROM); видеопамять (Video Memory Buffer). Они, как правило, реализованы на 8- или 16-битных микросхемах с довольно большим временем доступа. Обращение к полноразрядному системному ОЗУ выполняется гораздо быстрее. Для ускорения обращений к памяти этих устройств применяется **теневая память** (Shadow Memory), подменяющая память устройств системным ОЗУ. Теневая память появилась на развитых моделях AT-286, где она была реализована аппаратно. Процессоры класса 386+ позволяют ее реализовать программно, с помощью страничной переадресации. Затенение ОЗУ и ПЗУ устройств выполняется поразному.

При инициализации *теневого ПЗУ (Shadow ROM)* содержимое затеняемой области копируется в ОЗУ, и при дальнейшем чтении по этим адресам подставляется ОЗУ, а запись в эту область блокируется.

При использовании *теневого ОЗУ (Shadow RAM)* запись производится одновременно в физическую память затеняемой области и в системное ОЗУ, наложенное на эту область. При чтении затененной области обращение идет только к системной памяти, что происходит достаточно быстро. Особенно велик

эффект от затенения видеопамяти старых графических адаптеров, которая по чтению бывает доступна только во время обратного хода развертки, и процессору приходится долго ждать этого момента. Однако затенение областей разделяемой памяти, модифицируемых со стороны адаптеров, недопустимо: эти изменения не будут восприняты процессором. К разделяемой относятся буферная память сетевых адаптеров, видеопамять адаптеров с графическими сопроцессорами (акселераторами). Из этого следует, что затенение видеопамяти применимо только к примитивным графическим картам, устанавливаемым в слот ISA, и то не во всех режимах.

Обычно теневая память включается через CMOS Setup отдельными областями размером по 16 Кбайт или более крупными, и для каждой области указывают режим затенения (Shadow ROM или Shadow RAM). Возможно ее включение и драйверами ОС (например, драйвером EMM386). На современных системных платах затенение области системной BIOS выполняется всегда, на старых платах затенением этой области можно было управлять. Затенение BIOS видеоадаптера (Video BIOS Shadowing) для работы в среде Windows с «родными» драйверами графического адаптера может и не вызвать повышения производительности.

3.2.2 Распределение оперативной памяти в Microsoft Windows

3.2.2.1 Microsoft Windows 9x

С точки зрения базовой архитектуры операционные системы Windows 9x являются 32-разрядными, многопотоковыми ОС с вытесняющей многозадачностью. Основной пользовательский интерфейс этих ОС – графический. Для своей загрузки они используют операционную систему MS DOS 7.X, и если в файле MSDOS.SYS в секции [Options] прописано «BootGUI = 0», то процессор работает в обычном реальном режиме. Распределение памяти в MS DOS 7.X такое же, как и в предыдущих версиях DOS. Однако при загрузке GUI-интерфейса перед загрузкой ядра Windows 95/98 процессор переключается в защищенный режим

работы и начинает распределять память уже с помощью страничного механизма.

Использование так называемой **плоской модели памяти**, при которой все возможные сегменты, используемые программистом, совпадают друг с другом и имеют максимально возможный размер, определяемый системными соглашениями данной ОС, приводит к тому, что, с точки зрения программиста, память получается неструктурированной. За счет представления адреса как пары (P, i) память можно трактовать и как двумерную, то есть «плоскую», и как линейную, что существенно облегчает создание системного программного обеспечения и прикладных программ с помощью соответствующих систем программирования.

Таким образом, в системе фактически действует только страничный механизм преобразования виртуальных адресов в физические. Программы используют классическую «small» (малую) модель памяти. Каждая прикладная программа определяется 32-битными адресами, в которых сегмент кода имеет то же значение, что и сегменты данных. Единственный сегмент программы отображается непосредственно в область виртуального линейного адресного пространства, который, в свою очередь, состоит из 4 килобайтных страниц. Каждая страница может располагаться в любом месте оперативной памяти (естественно, в том месте, куда ее разместит диспетчер памяти, который сам находится в невыгружаемой области) или может быть перемещена на диск, если не запрещено использовать страничный файл.

Младшие адреса виртуального адресного пространства совместно используются всеми процессами. Это сделано для обеспечения совместимости с драйверами устройств реального режима, резидентными программами и некоторыми 16-разрядными программами Windows. Безусловно, это плохое решение с точки зрения надежности, поскольку оно приводит к тому, что любой процесс может непреднамеренно (или же, наоборот, специально) испортить компоненты, находящиеся в этих адресах.

В Windows 95/98 каждая 32-разрядная прикладная программа выполняется в своем собственном адресном пространст-

ве, но все они используют совместно один и тот же 32-разрядный системный код. Доступ к чужим адресным пространствам в принципе возможен. Другими словами, виртуальные адресные пространства не используют всех аппаратных средств защиты, заложенных в микропроцессор. В результате неправильно написанная 32-разрядная прикладная программа может привести к аварийному сбою всей системы. Все 16-битовые прикладные программы Windows разделяют общее адресное пространство, поэтому они так же уязвимы друг перед другом, как и в среде Windows 3.X.

Системный код Windows 95 размещается выше границы 2 Гбайт. В пространстве с отметками 2 и 3 Гбайт находятся системные библиотеки DLL (Dynamic Link Library – динамически загружаемый библиотечный модуль), используемый несколькими программами. Заметим, что в 32-битовых микропроцессорах семейства i80x86 имеются четыре уровня защиты, именуемые кольцами с номерами от 0 до 3. Кольцо с номером 0 является наиболее привилегированным, то есть максимально защищенным. Компоненты системы Windows 95, относящиеся к кольцу 0, отображаются на виртуальное адресное пространство между 3 и 4 гигабайтами. К этим компонентам относятся собственно ядро Windows, подсистема управления виртуальными машинами, модули файловой системы и виртуальные драйверы (VxD).

Область памяти между 2 и 4 гигабайтами адресного пространства каждой 32-разрядной прикладной программы совместно используется всеми 32-разрядными прикладными программами. Такая организация позволяет обслуживать вызовы API непосредственно в адресном пространстве прикладной программы и ограничивает размер рабочего множества, однако это снижает надежность. Ничто не может помешать программе, содержащей ошибку, произвести запись в адреса, принадлежащие системным DLL, и вызвать крах всей системы.

В области между 2 и 3 гигабайтами также находятся все запускаемые 16-разрядные прикладные программы Windows. С целью обеспечения совместимости эти программы выполняются в совместно используемом адресном пространстве, где они могут испортить друг друга так же, как и в Windows 3.x.

Адреса памяти ниже 4 Мбайт также отображаются в адресное пространство каждой прикладной программы и совместно используются всеми процессами, благодаря чему становится возможной совместимость с существующими драйверами реального режима, которым необходим доступ к этим адресам, но вследствие этого еще одна область памяти становится незащищенной от случайной записи. К нижним 64 килобайтам этого адресного пространства 32-разрядные прикладные программы обращаться не могут, что дает возможность перехватывать неверные указатели, но 16-разрядные программы, которые, возможно, содержат ошибки, могут записывать туда данные.

Вышеизложенную модель распределения памяти можно проиллюстрировать с помощью рис. 3.2.

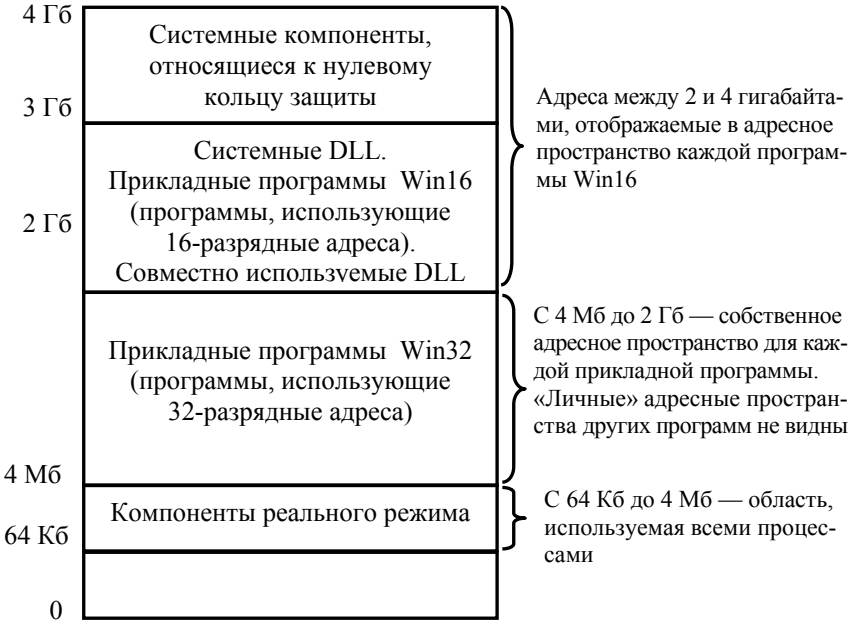

Рис. 3.2 – Модель памяти ОС Windows 9x

Минимально допустимый объем оперативной памяти ОС Windows 95 равен 4 Мбайт, однако при таком объеме пробуксовка столь велика, что практически работать нельзя. Странич-

ный файл, с помощью которого реализуется механизм виртуальной памяти, по умолчанию располагается в каталоге самой Windows и имеет переменный размер. Система отслеживает его длину, увеличивая или сокращая этот файл при необходимости. Вместе с фрагментацией файла подкачки (см. подраздел 3.3) это приводит к тому, что быстродействие системы снижается по сравнению с ситуацией, когда файл был бы фиксированного размера и располагался в смежных кластерах (т.е. был бы дефрагментирован). Сделать файл подкачки заданного размера можно либо через специально разработанную для этого подсистему: Панель управления → Система → Быстродействие → Файловая система; либо просто прописав в файле SYSTEM.INI в секции [386Enh] строчки с указанием диска и имени этого файла, например:

```
PagingDrive=C:
PagingFi1e=C:\PageFi1e.sys
MinPagingFi1eSize=65536
MaxPagingFileSize=262144
```

Две первых строки указывают имя страничного файла и его размещение, а две последних – начальный и предельный размер страничного файла (значения указываются в килобайтах). Для определения необходимого минимального размера этого файла можно рекомендовать запустить программу SysMon (системный монитор) и, выбрав в качестве наблюдаемых параметров размер файла подкачки и объем свободной памяти, оценить потребности в памяти, запуская те приложения, с которыми чаще всего приходится работать.

3.2.2.2 Microsoft Windows на платформе NT

В операционных системах Windows NT тоже используется плоская модель памяти. Заметим, что Windows NT 4,0 Server практически не отличается от Windows NT 4.0 Workstation; разница лишь в наличии у сервера некоторых дополнительных служб, дополнительных утилит для управления доменом и несколько иных значений в настройках системного реестра. Однако схема распределения возможного виртуального адресного

пространства в системах Windows NT разительно отличается от модели памяти Windows 95/98. Прежде всего, в отличие от Windows 95/98 в Win NT в гораздо большей степени используется ряд серьезных аппаратных средств защиты, имеющихся в микропроцессорах, а также применено принципиально другое логическое распределение адресного пространства.

Во-первых, все системные программные модули находятся в своих собственных виртуальных адресных пространствах, и доступ к ним со стороны прикладных программ невозможен. Ядро системы и несколько драйверов работают в нулевом кольце защиты в отдельном адресном пространстве.

Во-вторых, остальные программные модули самой операционной системы, которые выступают как серверные процессы по отношению к прикладным программам (клиентам), функционируют также в своем собственном системном виртуальном адресном пространстве, невидимом для прикладных процессов. Логическое распределение адресных пространств приведено на рис. 3.3.

Рис. 3.3 – Модель памяти ОС Windows NT

Прикладным программам выделяется 2 Гбайт локального (собственного) линейного (неструктурированного) адресного пространства от границы 64 Кбайт до 2 Гбайт (первые 64 Кбайт полностью недоступны). Прикладные программы изолированы друг от друга, хотя могут общаться через буфер обмена (clipboard) и механизмы: универсальные механизмы динамического обмена данными DDE (Dynamic Data Exchange); технологию документно-ориентированной архитектуры приложений OLE (Object Linking and Embedding).

В верхней части каждой 2-гигабайтной области прикладной программы размещен код системных DLL кольца 3, который выполняет перенаправление вызовов в совершенно изолированное адресное пространство, где содержится уже собственно системный код, выступающий как сервер-процесс (server process), который проверяет значения параметров, исполняет запрошенную функцию и пересылает результаты назад в адресное пространство прикладной программы. Хотя сервер-процесс сам по себе остается процессом прикладного уровня, он полностью защищен от вызывающей его прикладной программы и изолирован от нее. Между отметками 2 и 4 Гбайт расположены низкоуровневые системные компоненты Windows NT кольца 0, в том числе ядро, планировщик потоков и диспетчер виртуальной памяти. Системные страницы в этой области наделены привилегиями супервизора, которые задаются физическими схемами кольцевой защиты процессора. Это делает низкоуровневый системный код невидимым и недоступным для записи программ прикладного уровня, но приводит к падению производительности во время переходов между кольцами.

Для 16-разрядных прикладных Windows-программ ОС Windows NT реализует сеансы Windows on Windows (WOW). В отличие от Windows 95/98 ОС Windows NT дает возможность выполнять 16-разрядные программы Windows индивидуально в собственных пространствах памяти или совместно в разделяемом адресном пространстве. Почти во всех случаях 16- и 32-разрядные прикладные программы Windows могут свободно взаимодействовать, используя OLE, независимо от того, выполняются они в отдельной или общей памяти. Собственные прикладные программы и сеансы WOW выполняются в режиме вы-

тесняющей многозадачности, основанной на управлении отдельными потоками. Множественные 16-разрядные прикладные программы Windows в одном сеансе W0W выполняются в соответствии с кооперативной моделью многозадачности. Windows NT может также выполнять в многозадачном режиме несколько сеансов DOS. Поскольку Windows NT имеет полностью 32-разрядную архитектуру, не существует теоретических ограничений на ресурсы интерфейса графических устройств GDI (Graphics Device Interface) и USER.

При запуске приложения создается процесс со своей информационной структурой. В рамках процесса запускается задача. При необходимости этот тред (задача) может запустить множество других тредов (задач), которые будут выполняться параллельно в рамках одного процесса. Очевидно, что множество запущенных процессов также выполняются параллельно и каждый из процессов может представлять из себя мультизадачное приложение. Задачи (треды) в рамках одного процесса выполняются в едином виртуальном адресном пространстве, а процессы выполняются в различных виртуальных адресных пространствах. Отображение различных виртуальных адресных пространств исполняющихся процессов на физическую память реализует сама ОС; именно корректное выполнение этой задачи гарантирует изоляцию приложений от невмешательства процессов. Для обеспечения взаимодействия между выполняющимися приложениями и между приложениями и кодом самой операционной системы используются соответствующие механизмы защиты памяти, поддерживаемые аппаратурой микропроцессора.

Процессами выделения памяти, ее резервирования, освобождения и подкачки управляет диспетчер виртуальной памяти Windows NT VMM (Virtual Memory Manager). В своей работе этот компонент реализует сложную стратегию учета требований к коду и данным процесса для минимизации доступа к диску, поскольку реализация виртуальной памяти часто приводит к большому количеству дисковых операций.

Каждая виртуальная страница памяти, отображаемая на физическую страницу, переносится в так называемый ***страничный фрейм*** (page frame). Прежде чем код или данные можно будет переместить с диска в память, диспетчер виртуальной памяти

(модуль VMM) должен найти или создать свободный страничный фрейм или фрейм, заполненный нулями. Заметим, что заполнение страниц нулями представляет собой одно из требований стандарта на системы безопасности уровня C2, принятого правительством США. Страничные фреймы должны заполняться нулями для того, чтобы исключить возможность использования их предыдущего содержимого другими процессами. Чтобы фрейм можно было освободить, необходимо скопировать на диск изменения в его странице данных, и только после этого фрейм можно будет повторно использовать. Программы, как правило, не меняют страницы кода. Страницы кода, в которые программы не внесли изменений, можно удалить.

Диспетчер виртуальной памяти может быстро и относительно легко удовлетворить программные прерывания типа «ошибка страницы» (page fault). Что касается аппаратных прерываний типа «ошибка страницы», то они приводят к подкачке (paging), которая снижает производительность системы. Ранее говорилось о том, что в Windows NT, к большому сожалению, выбрана дисциплина FIFO для замещения страниц, а не более эффективные дисциплины LRU и LFU. Когда процесс использует код или данные, находящиеся в физической памяти, система резервирует место для этой страницы в файле подкачки Pagefile.sys на диске. Это делается с расчетом на тот случай, что данные потребуется выгрузить на диск. Файл Pagefile.sys представляет собой ***зарезервированный блок*** дискового пространства, который используется для выгрузки страниц, помеченных как «грязные», при необходимости освобождения физической памяти. Заметим, что этот файл может быть как непрерывным, так и фрагментированным; он может быть расположен на системном диске либо на любом другом и даже на нескольких дисках. Размер этого страничного файла ограничивает объем данных, которые могут храниться во внешней памяти при использовании механизмов виртуальной памяти. По умолчанию размер файла подкачки устанавливается равным объему физической памяти плюс 12 Мбайт, однако пользователь имеет возможность изменить его размер по своему усмотрению. Проблема нехватки виртуальной памяти часто может быть решена за счет увеличения размера файла подкачки. В системах Windows NT 4.0 объекты, созда-

ваемые и используемые приложениями и операционной системой, хранятся в так называемых ***пулах памяти*** (memory pools). Доступ к пулам может быть получен только в привилегированном режиме работы процессора, в котором работают компоненты операционной системы. Поэтому для того чтобы объекты, хранящиеся в пулах, стали видимы тредам приложений, треды должны переключиться в привилегированный режим. *Перемещаемый* или *нерезидентный пул* (paged pool) содержит объекты, которые могут быть при необходимости выгружены на диск. *Неперемещаемый* или *резидентный пул* (nonpaged pool) содержит объекты, которые должны постоянно находиться в памяти. В частности, к такого рода объектам относятся структуры данных, используемые процедурами обработки прерываний, а также структуры, используемые для предотвращения конфликтов в мультипроцессорных системах. Исходный размер пулов определяется объемом физической памяти, доступной Windows NT. Впоследствии размер пула устанавливается динамически и в зависимости от работающих в системе приложений и сервисов может изменяться в широком диапазоне.

Вся виртуальная память в Windows NT подразделяется на классы: зарезервированную (reserved), выделенную (committed) и доступную (available).

Зарезервированная память представляет собой набор непрерывных адресов, которые диспетчер виртуальной памяти (VMM) выделяет для процесса, но не учитывает в общей квоте памяти процесса до тех пор, пока она не будет фактически использована. Когда процессу требуется выполнить запись в память, ему выделяется нужный объем из зарезервированной памяти. Если процессу потребуется больший объем памяти, то дополнительная память может быть одновременно зарезервирована и использована, если в системе имеется доступная память.

Память выделена, если диспетчер VMM резервирует для нее место в файле Pagefile.sys на тот случай, когда потребуется выгрузить содержимое памяти на диск. Объем выделенной памяти процесса характеризует фактически потребляемый им объем памяти. Выделенная память ограничивается размером файла подкачки. Предельный объем выделенной памяти в системе (commit limit) определяется тем, какой объем памяти можно вы-

делить процессам без увеличения размеров файла подкачки. Если в системе имеется достаточный объем дискового пространства, то файл подкачки может быть увеличен и тем самым будет расширен предельный объем выделенной памяти.

Вся память, которая не является ни выделенной, ни зарезервированной, является **доступной**. К доступной относятся свободная память, обнуленная память (освобожденная и заполненная нулями), а также память, находящаяся в ***списке ожидания*** (standby list), которая была удалена из рабочего набора процесса, но может быть затребована вновь.

3.3 Организация режима защиты

3.3.1 Переключение задач и виртуальные машины

Пусть имеются два процесса (например, две прикладные программы), которые должны выполняться как бы одновременно (классический фон-неймановский процессор одновременно их выполнить не в состоянии). Можно запустить один процесс, а через некоторое время по аппаратному прерыванию (от таймера) сохранить в памяти образ его текущего состояния и запустить другой процесс. Через некоторое время по следующему прерыванию выполнить обратное переключение: сохранить состояние второго процесса в другом месте памяти, загрузить в регистры процессора образ состояния первого процесса и продолжить его выполнение. Такие ***переключения задач*** следует выполнять в течение времени исполнения обеих программ с частотой, создающей у пользователя иллюзию непрерывности и одновременности. Понятно, что ресурсы процессора (производительность) в этом случае делятся между задачами пропорционально выделяемым им квантам времени. Чтобы пользователя удовлетворяла такая производительность процессов, процессор должен обладать достаточной мощностью. Процессоры семейства x86, начиная со второго и, особенно, с третьего (386) поколения, имеют встроенные средства многозадачности (число задач почти не ограничено), работающие в защищенном режиме. Переключение задач производится по сигналу прерывания от таймера совершенно «прозрачно» для процессов, работающих псевдопа-

раллельно. В связи с чем программисту, разрабатывающему прикладную программу, в большинстве случаев не надо заботиться о многозадачной работе. В распоряжение его программы предоставляется **виртуальная машина** (тоже фон-неймановская), в которой управление передается последовательно этой программой, как будто она – единственный процесс. Конечно, поддержка виртуальных машин требует определенных усилий со стороны многозадачной операционной системы, которой приходится распределять не только процессорное время, но и память, устройства хранения, ввода-вывода и коммуникационные – все ресурсы реального компьютера. Это осуществляется с помощью специальных средств, введенных в процессоры x86 2–3-го поколений и постоянно развиваемых в следующих поколениях [12].

3.3.2 Защищенный режим и виртуальная память

Для того чтобы процессы не мешали друг другу (по недосмотру или умышленно), требуются меры принудительной защиты критических ресурсов. Современные операционные системы используют **защищенный режим** процессора, в котором эти меры реализуются на аппаратном уровне. Поскольку программа может взаимодействовать с подсистемами компьютера только через пространства памяти и портов ввода-вывода, а также аппаратные прерывания, то и защищать нужно эти три типа ресурсов. Самую сложную защиту имеет память. ОС выделяет каждому процессу области памяти – ***сегменты*** – различного назначения и с разными правами доступа. Из одних сегментов можно только читать данные, в другие возможна и запись. Для программного кода выделяются специальные сегменты, инструкции могут выбираться и исполняться только из них. По отношению к принципу хранимости программы это является искусственным ограничением для фон-неймановской машины, но целесообразность программного кода очевидна. Процессору «безразлично» содержимое ячейки памяти, на которую передалось управление, – он всегда пытается трактовать ее как код инструкции (или префикс). Если ошибочно управление передалось на область данных, то дальнейшее поведение процессора непредсказуемо – это так называемый «вылет». Защита не позволяет передать

управление на сегмент данных – сработает исключение защиты, которое обрабатывается ОС, и ошибочный процесс будет принудительно завершен. Таким образом, вероятность «вылета» уменьшается. Чтобы выдержать принцип хранимости программы, на время ее загрузки в память или программной модификации эту область объявляют сегментом данных, в который разрешена запись. Система защиты может полностью контролировать распределение памяти, генерируя исключения в случаях различных нарушений. Конечно же, эффективность защиты (устойчивость компьютера к ошибкам) в значительной мере определяется предусмотрительностью разработчиков операционной системы.

Чем сложнее программа и больше объем обрабатываемых ею данных, тем больше ее потребности в памяти. В первых процессорах семейства i80x86 память предоставлялась в виде сегментов с размером по 64 Кбайт, а суммарный объем программно-адресуемой памяти не превышал значения в 1 Мбайт. Потребности решаемых задач довольно быстро переросли эти ограничения, и в процессоры ввели средства организации виртуальной памяти.

Виртуальная память (Virtual Memory) представляет собой программно-аппаратное средство расширения пространства памяти, предоставляемой программе в качестве оперативной [12, 13]. Эта память физически реализуется в оперативной и дисковой памяти под управлением соответствующей операционной системы. Впервые она появилась в процессорах 80286, но удобный для употребления вид приняла только в 32-разрядных процессорах (80386 и выше). Во-первых, было снято ограничение на 64-Кбайтный размер сегмента – теперь любой сегмент может иметь почти произвольный размер до 4 Гбайт. Во-вторых, был введен механизм страничной переадресации памяти (paging). Теперь любая страница (область фиксированного размера) виртуальной логической памяти, адресуемой программой в пределах выделенных ей сегментов, может отображаться на любую область физической памяти (реально установленной оперативной памяти). Отображение поддерживается с помощью специальных таблиц страничной переадресации, в которых кроме связи адресов есть указание на присутствие страницы в фи-

зической памяти на данный момент времени. Теперь страница памяти, не нужная процессору в данный момент времени, может быть выгружена на устройство хранения (диск). На ее место можно загрузить нужную страницу. Заявку на загрузку нужной страницы делает сам процессор без каких-либо усилий выполняемой программы: если программе потребовалась ячейка виртуальной памяти из страницы, образа которой сейчас нет в физической памяти, вырабатывается специальное исключение. Обработчик этого исключения (это часть ОС) найдет свободную физическую страницу, выгрузив на диск ту, которая пока не нужна, «подкачает» на нее с диска требуемую информацию и вернет управление процессу, прерванному исключением. Этот процесс ничего «не заметит», кроме некоторой задержки выполнения инструкций. Таким образом, в распоряжение всех процессов, исполняемых на компьютере псевдопараллельно, предоставляется виртуальная оперативная память, размер которой ограничен суммой объема физической оперативной памяти и областью дисковой памяти, выделенной для подкачки страниц. Существуют следующие стратегии подкачки страниц: подкачка по запросу и подкачка с упреждением.

Подкачка страниц по запросу. Наиболее рациональной является загрузка в основную память страниц, необходимых для работы процесса, по его запросу. Не следует переписывать из внешней памяти в основную ни одной страницы до тех пор, пока к ней явно не обратится выполняющийся процесс. В пользу такой стратегии можно привести несколько аргументов [12]:

- на основании положений теории вычислимости, в частности по решению ***проблемы останова***, путь, выбираемый программой при своем выполнении, невозможно точно предсказать. Поэтому любая попытка заранее загрузить страницы в память в предвидении того, что они потребуются в работе (стратегия вталкивания с упреждением), может оказаться неудачной – будут загружены другие страницы;
- подкачка страниц по запросу гарантирует, что в основную память будут переписываться только те страницы, которые фактически необходимы для работы процессов;
- дополнительные затраты на то, чтобы определить, какие страницы следует передавать в основную память, минимальны.

Вталкивание с упреждением может потребовать значительных дополнительных затрат процессорного времени.

Подкачка страниц по запросу имеет свои проблемы. Процесс должен накапливать в памяти требуемые ему страницы по одной. При появлении ссылки на каждую новую страницу процессу приходится ждать, когда эта страница будет передана в основную память. В зависимости от того, сколько страниц данного процесса уже находятся в основной памяти, эти периоды ожидания будут обходиться все более дорого, поскольку ожидающие процессы будут занимать все больший объем памяти. Уменьшение затрат времени и объема информации за счет периодов ожидания процессом нужных ему страниц является важнейшей целью всех стратегий управления памятью.

Подкачка страниц с упреждением. Главный критерий эффективности управления ресурсами можно сформулировать следующим образом: интенсивность использования каждого ресурса должна определяться относительной ценностью этого ресурса. Так, в настоящее время стоимость аппаратуры резко снижается, и поэтому значительно снижается относительная ценность машинного времени на обработку информации по сравнению с временем, затрачиваемым человеком. Сейчас разработчики операционных систем активно ищут пути уменьшения количества времени, в течение которого пользователям приходится ждать получения результатов от компьютера. Одним из весьма перспективных в этом смысле является метод подкачки страниц с упреждением (с опережением).

При упреждающей подкачке операционная система пытается заблаговременно предсказать, какие страницы потребуются процессу, а затем, когда в основной памяти появляется свободное место, загружает в нее эти страницы. Пока процесс работает со своими текущими страницами, система запрашивает новые страницы, которые будут уже готовы к использованию, когда процесс к ним обратится. Если решения о выборе страниц для подкачки принимаются правильно, то удается значительно сократить общее время выполнения данного процесса.

Метод подкачки страниц с упреждением характеризуется следующими преимуществами [12]:

- если в большинстве случаев удается принимать правильные решения о выборе страниц для подкачки, то время выполнения процесса значительно уменьшается. Поэтому имеет смысл пытаться создавать механизмы упреждающей подкачки, даже если от них нельзя ожидать абсолютной точности;
- во многих случаях можно находить вполне правильные решения. Если их удается реализовать при относительно малых затратах, то выполнение данного процесса можно значительно ускорить, не замедляя при этом работы других активных процессов;
- поскольку аппаратура вычислительных машин все более дешевеет, последствия неоптимальных решений становятся менее серьезны. Сейчас мы можем позволить себе приобрести дополнительную основную память, обеспечивающую накопление дополнительных страниц, которые механизм упреждающей подкачки будет передавать в основную память.

3.3.3 Организация и адресация памяти в защищенном режиме

В процессорах x86 предусматривается разделение пространств памяти и ввода-вывода. **Пространство памяти** (Memory Space) предназначено для хранения кодов инструкций и данных, доступ к которым осуществляется на основе большого выбора способов адресации (24 режима). Память может логически организовываться в виде одного или множества сегментов переменной длины или фиксированной длины в реальном режиме. Кроме сегментации в защищенном режиме возможно разбиение (Paging) логической памяти на страницы размером 4 Кбайт, каждая из которых может отображаться на любую область физической памяти. Начиная с 5-го поколения, появилась возможность увеличения размера страницы до 4 Мбайт. Сегментация и разбиение на страницы могут применяться в любых сочетаниях. Сегментация является средством организации логической памяти на прикладном уровне. Разбиение на страницы применяется на системном уровне для управления физической памятью. Сегменты и страницы могут выгружаться из физической оперативной памяти на диск и по мере необходимости под-

качиваться с него обратно в физическую память. Таким образом реализуется виртуальная память.

Применительно к памяти различают три адресных пространства: логическое, линейное и физическое. Основным режимом работы 32-разрядных процессоров считается защищенный режим, в котором работают все механизмы преобразования адресных пространств (рис. 3.4) [12, 13].

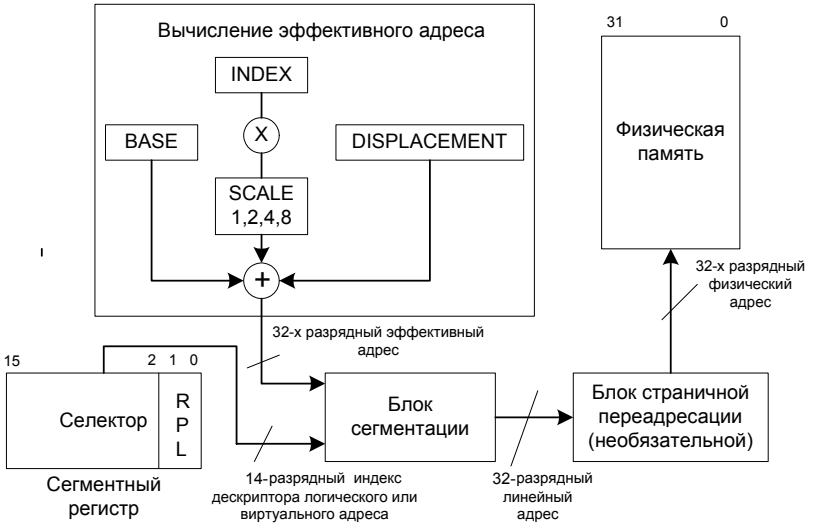

Рис. 3.4 – Формирование адреса памяти 32-разрядных процессоров в защищенном режиме

Логический адрес, также называемый виртуальным, состоит из селектора сегмента Seg (в реальном режиме – просто адреса сегмента) и эффективного адреса, называемого также смещением (Offset). Логический адрес обозначается в форме Seg:Offset. ***Селектор сегмента*** хранится в старших 14 битах сегментного регистра (CS, DS, ES, SS, FS или GS), участвующего в адресации конкретного элемента памяти. По значению селектора из специальных таблиц, хранящихся в памяти, извлекается начальный адрес сегмента. ***Эффективный адрес*** формируется суммированием компонентов base, index, displacement с

учетом масштаба scale. Поскольку каждая задача может иметь до 16 Кбайт селекторов (2^{14}), а смещение, ограниченное размером сегмента, может достигать 4 Гбайт, логическое адресное пространство для каждой задачи может достигать 64 Тбайт. Все это пространство виртуальной памяти в принципе доступно программисту при условии поддержки со стороны операционной системы.

Блок сегментации транслирует логическое адресное пространство в 32-битное пространство линейных адресов. **Линейный адрес** образуется сложением базового адреса сегмента с эффективным адресом. Базовый адрес сегмента в реальном режиме образуется умножением содержимого используемого сегментного регистра на 16 (как и в 8086). В защищенном режиме базовый адрес загружается из дескриптора, хранящегося в таблице, по селектору, загруженному в используемый сегментный регистр.

32-битный **физический адрес** памяти образуется после преобразования линейного адреса блоком страничной переадресации. Он выводится на внешнюю шину адреса процессора. В простейшем случае при отключенном блоке страничной переадресации физический адрес совпадает с линейным. Включенный блок страничной переадресации осуществляет трансляцию линейного адреса в физический страницами размером 4 Кбайт (для последних поколений процессоров возможны страницы размером 2– 4 Мбайт). Блок обеспечивает и расширение разрядности физического адреса процессоров шестого поколения до 36 бит. Блок переадресации может включаться только в защищенном режиме.

Для обращения к памяти процессор совместно с внешними схемами формирует шинные сигналы для операций записи и чтения. Шина адреса разрядностью 32/36 бит позволяет адресовать 4/64 Гбайт физической памяти, но в реальном режиме доступен только 1 Мбайт, начинающийся с младших адресов.

3.3.4 Кэширование памяти

Архитектура современных 32-разрядных процессоров включает ряд средств кэширования памяти: два уровня кэша инструкций и данных (L1 Cache и L2 Cache); буферы ассоциа-

тивной трансляции (TLB) блока страничной переадресации и буферы записи. Эти средства в разных вариациях (на кристалле, картридже процессора или системной плате) представлены в системах с процессорами 486, Pentium и P6. В процессоре 80386 (Intel) имелся только TLB, а кэш-память, устанавливаемая на системной плате, не имела поддержки со стороны процессора.

Все механизмы кэширования в основном прозрачны для прикладных программ и после разрешения кэширования пропускают через себя потоки инструкций и данных без требования явного программного управления. Однако знание особенностей механизмов кэширования помогает в оптимизации кода. Например, можно определить оптимальные размеры одновременно обрабатываемых структур данных, при которох кэш не «буксует» (cache thrashing). Процессоры разных моделей имеют различные характеристики отдельных элементов кэша, их определение для процессоров P6 осуществляется посредством вызова инструкции CPUID(2). Заметим, что не все модели процессоров способны кэшировать весь объем физически адресуемой памяти.

Кэш-память процессоров строится с учетом возможности обращений к памяти со стороны внешних абонентов – других процессоров или иных контроллеров шины. Процессоры имеют механизмы внешнего слежения за состоянием собственного кэша с соответствующими аппаратными интерфейсами. Для поддержания согласованности данных кэша и основной памяти процессор отрабатывает циклы слежения (Snoop Cycle или Inquire Cycle), инициированные внешней для него системой. В этих циклах, происходящих при обращении к памяти со стороны внешнего абонента, процессор определяет присутствие затребованной области в своем собственном кэше. Если область отображается в кэше, то действия процессора зависят от состояния соответствующей строки кэша и типа внешнего обращения: обращение по записи вызовет аннулирование данной строки; обращение по чтению к области, соответствующей модифицированной («грязной») строке, – выгрузку ее содержимого в основную память, прежде чем внешний абонент выполнит реальное считывание. В процессорах P6 обращение к «грязной» строке со стороны другого процессора может вызывать выгрузку ее содержимого непосредственно в обращающийся процессор, что

экономит время. Выгрузка этой строки в основную память будет произведена позже, согласно алгоритму обратной записи.

Кэш процессоров, начиная с Pentium, поддерживает протокол MESI, названный по определяемым им состояниям M (Modified), E (Exclusive), S (Shared), I (Invalid). Первичный кэш инструкций реализует протокол лишь в части «SI», поскольку он не допускает записи. Существуют различные состояния строк для каждого процессора [13]:

M-состояние – строка присутствует в кэше только этого процессора и модифицирована, то есть отличается от содержимого основной памяти; запись в эту строку не приведет к генерации внешнего по отношению к локальной шине цикла обращения;

E-состояние – строка присутствует в кэше только этого процессора, но не модифицирована (ее копия в основной памяти действительна); запись переведет ее в M-состояние, не вызывая внешнего цикла обращения оперативной памяти;

S-состояние – строка присутствует в кэше этого процессора и потенциально может присутствовать в кэшах других процессоров (копия в памяти действительна); запись в нее должна сопровождаться сквозной записью в основную память, что повлечет аннулирование соответствующих строк в других кэшах;

I-состояние – строка отсутствует в кэше, ее чтение может привести к генерации цикла заполнения строки; запись в нее будет сквозной и выйдет на внешнюю шину.

В процессорах шестого поколения в связи с их «беспорядочностью» и «спекулятивностью» обращения к памяти могут производиться с различными методами повышения эффективности. По возможностям кэширования память можно классифицировать следующим образом [13]:

некэшируемая память UC (Uncacheable). Все обращения процессора по чтению и записи выполняются строго в порядке, предписанном программным кодом, и выходят на системную шину. Никакие спекулятивные чтения и предварительные выборки не используются. Такой тип требуется для ввода-вывода, отображенного на память. Работа процессора в этом режиме с обычным ОЗУ приведет к значительному снижению производительности;

память с комбинируемой записью WC (Write Combining). Некэшируемая память, когерентность памяти не поддерживаются протоколом шины. Спекулятивное чтение допустимо, записи могут комбинироваться и откладываться до любого события, вызывающего сериализацию (инструкция CPUID – обращение к некэшируемой памяти, прерывание и др.). Такой тип применим, например, для видеопамяти графического адаптера (порядок записей не важен);

память со сквозной записью WT (Write-through). Кэшируемая память, все операции записи и отражаются в кэше, и выходят на системную шину. Чтения по возможности выполняются из кэша, кэш-промахи вызывают заполнение строк кэша. Спекулятивное чтение и комбинирование записей разрешено. Данный тип применим, например, для буферов кадров, а также для памяти, к которой могут обращаться устройства, подключенные к шине и не поддерживающие протоколов обеспечения когерентности;

память с обратной записью WB (Write-back). Кэшируемая память, все операции чтения и записи по возможности выполняются только с кэш-памятью. Запись на системную шину выходит только при необходимости освобождения строк или по требованию от других абонентов шины, что уменьшает необязательный трафик шины. Спекулятивное чтение и комбинирование записи разрешено. Этот тип самый производительный, но требует поддержки протокола обеспечения когерентности от всех абонентов шины, обращающихся к данной области памяти;

память с защищенной записью WP (Write protected). Кэшируемая память, операции чтения по возможности выполняются из кэша, промахи вызывают заполнение строк. Записи выходят на системную шину и вызывают аннулирование строк в кэшах всех остальных абонентов шины (процессоров).

Доступные методы кэширования зависят от возможностей процессора. Базовые методы (сквозная и обратная запись или отмена кэширования) управляются атрибутами системы управления страничной переадресации, более совершенные методы программируются только через регистры MTRR (Memory Type Range Registers – регистры, описывающие свойства областей

памяти) или PAT (Page Attribute Table – таблица атрибутов страниц памяти), имеющиеся в процессоре.

Вопросы для самопроверки

1. Назовите основные принципы фон-неймановской архитектуры вычислительных машин.
2. Что такое LH-порядок следования байт?
3. Расскажите, как переводить числа из одной системы исчисления в другую.
4. В чем отличие ячеек памяти, портов ввода-вывода и регистров?
5. Как строится адресация ячеек памяти в реальном режиме?
6. Опишите распределение оперативной памяти в MS DOS.
7. Опишите распределение оперативной памяти в Windows 9x.
8. Опишите распределение оперативной памяти в Windows NT.
9. Что такое Conventional memory?
10. Приведите структуру UMA.
11. Что такое HMA?
12. В чем отличие EMS и XMS спецификаций?
13. В чем отличие Shadow ROM и Shadow RAM?
14. Опишите принцип работы виртуальной памяти.
15. Как осуществляется переключение задач в ОС Windows и работа виртуальных машин?
16. Как строится адресация ячеек памяти в защищенном режиме?
17. Что такое стек?
18. Расскажите о механизме кэширования памяти.

4. УПРАВЛЕНИЕ ВНЕШНЕЙ ПАМЯТЬЮ И ФАЙЛОВЫЕ СИСТЕМЫ

4.1 Характеристика устройств внешней памяти

4.1.1 Общие свойства устройств внешней памяти

К внешней памяти компьютера относятся устройства хранения данных, позволяющие сохранять и накапливать информацию для последующего ее использования независимо от состояния компьютера (включен или выключен). В устройствах хранения данных могут быть реализованы различные физические принципы хранения информации – магнитный, оптический, электронный – в любых их сочетаниях.

В состав внешней памяти компьютера входят:

- накопители на жестких магнитных дисках;
- накопители на гибких магнитных дисках;
- накопители на магнито-оптических компакт-дисках;
- накопители на магнитной ленте (стримеры) и др.

Внешняя память принципиально отличается от внутренней (оперативной, постоянной и специальной) способом доступа процессора (исполняемой программы) к тому или другому виду памяти. Устройства внешней памяти оперируют блоками информации, но никак не байтами или словами, как, например, оперативная память. Блоки обычно имеют фиксированный размер, кратный степени числа 2. Блок может быть переписан из внутренней памяти во внешнюю или обратно только целиком, и для выполнения любой операции обмена с внешней памятью требуется специальная процедура (подпрограмма). Процедуры обмена с устройствами внешней памяти привязаны к типу устройства, его контроллеру и способу подключения устройства к системе (интерфейсу).

По методу доступа к информации устройства внешней памяти разделяются на устройства с прямым (или непосредственным) и последовательным доступом [12]. **Прямой доступ** (direct access) подразумевает возможность обращения к блокам по их адресам в произвольном порядке. Традиционными устройствами с прямым доступом являются дисковые накопители, и часто

в понятие «диск», или дисковое устройство» (disk device), вкладывают значение «устройство внешней памяти прямого доступа». Так, например, виртуальный диск в ОЗУ и электронный диск на флэш-памяти отнюдь не имеют круглых, а тем более вращающихся деталей. Традиционными устройствами **с последовательным доступом** являются накопители на магнитной ленте (tape device), они же стримеры. Здесь каждый блок информации тоже может иметь свой адрес, но для обращения к нему устройство хранения должно сначала найти некоторый маркер начала ленты (тома), после чего последовательным холостым чтением блока за блоком дойти до требуемого места и только тогда производить собственно операции обмена данными. Конечно, каждый раз возвращаться на начало ленты необязательно, однако необходимость последовательного сканирования блоков (вперед или назад) – неотъемлемое свойство устройств последовательного доступа. Несмотря на очевидный проигрыш во времени доступа к требуемым данным, ленточные устройства последовательного доступа в качестве внешней памяти находят применение для хранения очень больших массивов информации. В отличие от них устройства прямого доступа – диски самой различной природы – являются обязательной принадлежностью подавляющего большинства компьютеров.

4.1.2 Основные характеристики устройств внешней памяти

Главная характеристика устройств – **емкость хранения** (capaci-ty), измеряемая в килобайтах (Кбайт), мегабайтах (Мбайт), гигабайтах (Гбайт) и терабайтах (Тбайт), или в английской транскрипции – KB, MB, GB, TB соответственно. Здесь, как правило, приставки кило-, мега-, гига-, тера- имеют десятичные значения – 103, 106, 109 и 1012 соответственно. В других подсистемах компьютера, например при определении объема ОЗУ, ПЗУ и другой внутренней памяти, эти же приставки чаще применяют в двоичных значениях 210, 220, 230 и 240 соответственно, при этом 1 Кбайт равен 1024 байтам, 1 Мбайт – 1024 Кбайт, 1 Гбайт – 1024 Мбайт, 1 Тбайт – 1024 Гбайт [12]. Этими разночтениями объясняются различия значений емкости одного и

того же устройства, полученные из разных источников. «Двоичные» кило-, мега-, гига-, тера- более «увесисты», поэтому емкость устройства, отраженная в десятичных единицах, будет выглядеть внушительнее. Например, до недавнего времени один и тот же порог «беспроблемного» объема жесткого диска составлял 528 Мбайт (десятичных) или 504 Мбайт (двоичных).

Устройства внешней памяти могут иметь сменные или фиксированные носители информации. Применение сменных носителей (remo-vable media) позволяет хранить неограниченный объем информации, а если носитель и формат записи стандартизованы, то они позволяют еще и обмениваться информацией между компьютерами. Существуют устройства с автоматической сменой носителя – ленточные карусели, дисковые устройства (JukeBox). Эти достаточно дорогие устройства применяют в мощных файл-серверах. Для настольных машин имеются накопители CD-ROM с несколькими дисками (CD-changer), сменяемыми автоматически.

Важнейшими общими параметрами устройств являются время доступа, скорость передачи данных, удельная стоимость хранения информации, скорость записи и считывания.

Время доступа (access time) определяется как усредненный интервал времени от выдачи запроса на передачу блока данных до фактического начала передачи. Дисковые устройства имеют время доступа от единиц до сотен миллисекунд. Для электронных устройств внешней памяти время доступа определяется быстродействием используемых микросхем памяти и при чтении составляет доли микросекунд, причем запись может продолжаться значительно дольше, что объясняется природой энергонезависимой электронной памяти. Для устройств с подвижными носителями основной расход времени имеет место в процессе позиционирования головок (seek time – время поиска) и ожидания подхода к ним требуемого участка носителей (latency – скрытый период). Для дисковых и ленточных устройств принципы позиционирования различны, и различные составляющие процесса поиска будут подробнее рассмотрены в описании соответствующих устройств.

Скорость записи и считывания определяется как отношение объема записываемых или считываемых данных ко времени,

затрачиваемому на эту операцию. В затраты времени входит и время доступа, и время передачи данных. При этом оговаривается характер запросов (линейный или случайный), что сильно сказывается на величине скорости из-за влияния времени доступа. При определении скорости линейных запросов чтения-записи (linear transfer rate read/write) производится обращение к длинной цепочке блоков с последовательным нарастанием адреса. При определении скорости случайных запросов чтения-записи (random transfer rate read/write) соседние запросы разбросаны по всему носителю, что увеличивает время записи или считывания. Для современных многозадачных ОС характерно чередующееся выполнение нескольких потоков запросов, и в каждом потоке высока вероятность последовательного нарастания адреса.

Скорость передачи данных (Transfer Speed, Transfer Rate) определяется как производительность обмена данными после выполнения поиска данных. Однако в способе измерения этого параметра возможны разночтения, поскольку современные устройства имеют в своем составе буферную память[6] существенных размеров. Скорости обмена буферной памяти с собственно носителем (внутренняя скорость) и внешним интерфейсом могут существенно различаться. Если скорость работы внешнего интерфейса ограничивается быстродействием электронных схем и достижимой частотой передаваемых сигналов, то внутренняя скорость более жестко ограничивается возможностями электромеханических устройств (скоростью движения носителя и плотностью записи). При измерениях скорости передачи на небольших объемах пересылок проявится ограничение внешнего интерфейса буферной памяти, при средних объемах – ограничение внутренней скорости, а при больших объемах проявится еще и время поиска последующих блоков информации. Бывает, что в качестве скорости передачи данных указывают лишь максимальную скорость интерфейса, а о внутренней скорости можно судить по частоте вращения дисковых носителей и числу секторов на треке, но об этих понятиях будет сказано чуть позже.

---

[6] Буферной памятью называется объект памяти, которым владеют одновременно два объекта, не связанные между собой.

Определение **удельной стоимости хранения информации** для накопителей с фиксированными носителями пояснения не требует. В случае сменных носителей этот показатель интересен для собственно носителей, но не следует забывать и о цене самих приводов, которую тоже можно приводить к их емкости.

В табл. 4.1 приведены основные параметры распространенных устройств внешней памяти.

Таблица 4.1 – Основные параметры устройств внешней памяти

Тип и размер диска	Емкость диска, байт	Время доступа,	Скорость, Мбайт/с
FDD 3,5"	1,44 Мб	100*	0,055
HDD IDE	ЗО Гб	7,5–10	2–20+
HDD SCSI	ЗО Гб	7,5–10	2–40+
CD-ROM Ix	650 Мб	240–500	0,15
CD-ROM 48x	650 (700) Мб	75	7,2 (макс.)
CD-RW 4/4/32 (IDE)	650 (700) Мб	150	Чтение: 4,8 (макс.) Запись: 0,6
CD-RW 8/8/32 (SCSI)	650 (700) Мб	150	Чтение: 4,8 (макс.) Запись: 1,2
DVD-ROM 12x	4,7–17,08 бГ	200	130 (макс.)
MOD 3,5"	230 Мб	50	Чтение: 1,2–2 Запись: 0,6–2
MOD 3,5"	540/640 Мб	28	Чтение: до 3,67, Запись: до 1,2
MOD 3,5"	1,З Гб	28	Чтение: до 4,5, Запись: до 1,5
LS-120	120 Мб	70	0,1–0,5
Iomega Zip 100	100 Мб	29	1,4
Iomega Zip 250	250 Мб	29	2,4
Iomega Jaz	1 Гб	16	7,5
Iomega Jaz	2 Гб	16	7,5
SyQuest EZ135	135 Мб	21	1,4

Окончание табл. 4.1

Тип и размер диска	Емкость диска, байт	Время доступа,	Скорость, Мбайт/с
SyQuest EZFlyer	230 Мб	13,5	До 2,4
SyQuest SparQ	1 Гб	12	3,7–6,9
SyQuest Syjet	1,5 Гб	12	3,7–6,9
Compact Flash	32 Мб	0,001	Чтение: до 20

4.1.3 Характеристики накопителей на жестких магнитных дисках

Накопители на жестких магнитных дисках (НЖМД), они же HDD (Hard Disk Drive), являются главными устройствами дисковой памяти большинства компьютеров. По случайному совпадению цифр первые модели НЖМД назвали «винчестером» (просто игра слов), и это неофициальное название закрепилось в качестве синонима HDD и НЖМД [12]. Винчестер определяет мощность компьютера наряду с процессором и оперативной памятью. Мощность винчестера характеризуется большим объемом хранимой информации (десятки гигабайт), малым временем доступа (единицы миллисекунд), большой скоростью передачи данных (десятки мегабайт в секунду), высокой надежностью, умеренной стоимостью и рядом других полезных свойств. Прогресс в области производства винчестеров устойчив и стремителен: от 10-мегабайтного диска XT уже пришли к десяткам гигабайт, скорость передачи данных возросла на три порядка. Каждый год «модная» емкость винчестера ПК примерно удваивается, при этом цена устройства постепенно снижается.

Рассмотрим **общие параметры диска** [12].

Форматированная емкость (formatted capacity), измеряемая в гигабайтах (мегабайтах), представляет собой объем хранимой полезной информации (сумму полей данных всех доступных секторов). Неформатированная емкость (unf formatted capacity) представляет собой максимальное количество битов, записываемых на всех треках диска, включая и служебную информацию (заголовки секторов, контрольные коды полей данных). Соотношение форматированной и неформатированной

емкостей определяется форматом трека (размером сектора), но поскольку для рядового пользователя свободы в выборе формата нет, практический интерес представляет только форматированная емкость диска, которая указывается для стандартного размера сектора (512 байт). Напомним, что мегабайт и гигабайт здесь обычно означают 10^3 и 10^6 байт. Иногда указывается число доступных секторов.

Скорость вращения шпинделя (spindle speed), измеряемая в оборотах в минуту RPM (Revolutions Per Minute), позволяет косвенно судить о производительности (внутренней скорости). Для жестких дисков широкого применения значение 3600 об/мин было стандартным несколько лет назад; сейчас обычной считается 4500 и 5400 об/мин, а 7200 – более высокой скоростью. Там, где производительность особо критична, используют диски со скоростью 10000 и 15000 об/мин.

Интерфейс (interface) определяет способ подключения накопителя. Для накопителей со встроенным контроллером распространены интерфейсы ATA, а также IDE и SCSI, для устройств внешнего исполнения применяют шины USB, FireWire и Fibre Channel, а также подключение к LPT-порту.

К группе **параметров внутренней организации диска** относятся:

количество физических дисков (disks) или рабочих поверхностей (data surfaces), используемых для хранения данных. Современные накопители с небольшой высотой имеют малое (1–2) количество дисков для облегчения блока головок. Большее число дисков характерно для старых накопителей и современных накопителей большой емкости;

количество физических головок чтения-записи (read/write heads), естественно, совпадающее с числом рабочих поверхностей. Заметим, что число головок и соответственно рабочих поверхностей может быть и меньше удвоенного числа дисков – обычно в каждом семействе есть такого рода модели. Это делается для утилизации дисков, у которых одна из поверхностей оказывается с производственным браком, или исходя из других технологических соображений;

физическое количество цилиндров (cylinders), возросшее от нескольких сотен, характерных для первых винчестеров, до десятков тысяч;

размер сектора (Bytes Per Sector), составляющий обычно 512 байт. Количество зон и количество секторов на треке (Sectors Per Track) в крайних зонах;

расположение сервометок или сервоголовок (servo head), находящееся на выделенной поверхности (dedicated servo), рабочих поверхностях (embe-ded servo) или гибридное (hybrid servo);

метод кодирования-декодирования данных (recording method или data encoding sheme): MFM (FM почти и не применяли); RLL (ARLL); PRML, последний из них является наиболее прогрессивным.

Быстродействие и производительность диска характеризуются следующими параметрами:

временем перехода на соседний трек (track-to-track seek), измеряемым в миллисекундах, показывающим быстродействие системы позиционирования. Для современных жестких дисков характерно время перехода 0,5–2 мс, причем для записи оно несколько больше, чем для считывания (записывать лучше при более точном позиционировании);

средним временем поиска (average seek time), определяемым по набору обращений к случайным цилиндрам. Для большинства современных дисков оно составляет около 8–10 мс, в самых быстрых его удается снизить до 4–5 мс. Чем больше объем накопителя, тем сложнее достичь снижения времени поиска: большее число головок труднее быстро перемещать; большее число цилиндров или увеличивает длину перемещения головок, или повышает требования к точности позиционирования;

максимальным или полным временем поиска (maximum seek time, full seek time), определяемым для самых удаленных переходов между крайними цилиндрами. Оно примерно в два раза превышает среднее время поиска. Среднее ожидание сектора при одиночном обращении (average latency) обычно составляет половину времени полного оборота (для 3600 об/мин – 8 мс, 7200 – 4 мс, 15000 – 2 мс);

внутренней скоростью передачи данных (internal transfer rate) между носителем и буферной памятью контроллера, задающей физический предел производительности накопителя. Скорость выражается в разных величинах: если указывается в мегабитах в секунду (Mb/s), то сюда, кроме пользовательских данных, входят и биты служебных полей. У высокоскоростных винчестеров с частотой вращения 15000 об/мин этот параметр достигает 500 Мбит/с. При выражении скорости в мегабайтах в секунду (MB/s) подразумевают только байты пользовательских данных, и поэтому пересчет на мегабиты в секунду простым умножением на 8 (число битов в байте) неправомерен. У современных винчестеров с частотой вращения 5400 об/мин скорость составляет 8–15 Мбайт/с, 7200 об/мин – 15–35 Мбайт/с. Для каждой модели обычно указываются минимальное и максимальное значения скорости, соответствующие внутренним и внешним трекам;

внешней скоростью передачи данных (external transfer rate), измеряемой в килобайтах (мегабайтах) полезных данных в секунду, передаваемых по шине внешнего интерфейса, и зависящей от быстродействия электроники контролера, типа интерфейсной шины и режима обмена;

длительной производительностью (sustained throughput), определяемой при последовательном чтении большого количества секторов (например, при применении накопителей для мультимедийных приложений). На этот параметр влияют все составляющие: внутренняя и внешняя скорости, время позиционирования, задержка подхода сектора, количество ошибок позиционирования и чтения. Винчестеры с частотой вращения 5400 об/мин выдерживают потоки 8–25 Мбайт/с, с частотой 7200 об/мин – 10–30 Мбайт/с и 15000 об/мин – 35–45 Мбайт/с.

Группа параметров надежности устройств и достоверности хранения включает следующие показатели:

ожидаемое время до отказа MTBF (Mean Time Before Failure), измеряемое в сотнях тысяч часов, – среднестатистический показатель для данного устройства. Реально столько часов (100000 часов – это более 10 лет) испытания проводить, естественно, невозможно. На самом деле делается выборка из большой группы устройств, из которых за вполне обозримое время испытаний какая-то часть выйдет из строя;

наиболее значимый для пользователя параметр – ***гарантийный срок*** (limited warranty), в течение которого изготовитель (или поставщик) обеспечивает ремонт или замену отказавшего устройства. Примечательно, что даже при MTBF, равном 800000 часам (91 год), изготовитель дает гарантию всего на 3–5 лет;

вероятность неисправимых ошибок чтения (nonrecoverable read errors per bits read). Для современных винчестеров имеет порядок одной ошибки на 1014 считанных битов. Оценить, много это или мало, можно следующим образом. Пусть винчестер постоянно находится в работе и к нему непрерывно поступают обращения со средней производительностью чтения, которую приблизительно можно оценить в 1 Мбайт/с, что соответствует умеренной загруженности диска сервера. Простая арифметика показывает, что один раз в 115 дней будут возникать ошибки, не восстанавливаемые (но обнаруженные!) схемами контроллера. Вполне вероятно, что повторное считывание сектора пройдет без ошибок;

вероятность исправимых ошибок (recoverable read errors per bits read) имеет порядок единицы на 10 считанных битов. При отсутствии контроллера или неисправной схеме контроля этот поток ошибок сделал бы работу с таким накопителем просто невыносимой (ошибки будут появляться чаще, чем один раз в три часа);

вероятность ошибок поиска (seek errors per seek), характеризующая качество сервосистемы. Для современных винчестеров характерна вероятность одной ошибки на 108 операций поиска. Эти ошибки при малом их числе вполне безобидны, поскольку наличие номера цилиндра в заголовке каждого сектора не позволяет «промахнуться» при выполнении операций чтения или записи. Повторение операции поиска только слегка снижает среднее время доступа.

В отдельную группу параметров выделяется ***уровень акустического шума***, который характеризуется звуковой мощностью (sound power), излучаемой винчестером. На холостом ходу для винчестеров со скоростью вращения 5400 об/мин предпочтителен уровень до 30 дБ, при позиционировании желательно, чтобы он возрастал не более чем на 3–4 дБ. Для высокопроизво-

дительных винчестеров (7200 об/мин) желательно, чтобы уровень шума не превышал 35 дБ на холостом ходу; для винчестеров, предназначенных для работы в устройствах бытовой электроники, – 25 дБ.

4.2 Структура магнитного диска

4.2.1 Физическая структура

Загрузка собственно операционной системы и организация с ее помощью работы той или иной файловой системы осуществляется с магнитного диска. Были приняты специальные системные соглашения о структуре диска. Структура данных, содержащая информацию о логической организации диска, и простейшая программа, с помощью которой можно находить и загружать загрузочные программы той или иной ОС, – это первый (загрузочный) сектор магнитного диска.

Как известно, информация на магнитных дисках размещается и передается блоками. Каждый такой блок называется ***сектором*** (sec-tor), сектора расположены на концентрических дорожках поверхности диска. Каждая дорожка (track) образуется при вращении магнитного диска под зафиксированной в некотором предопределенном положении головкой чтения/записи. Накопитель на жестких магнитных дисках (НЖМД) содержит один или более дисков (в современных распространенных НЖМД часто – два или три). Однако обычно под термином «жесткий диск» понимают весь пакет магнитных дисков.

Группы дорожек (треков) одного радиуса, расположенных на поверхностях магнитных дисков, образуют так называемые ***цилиндры*** (cylinder). Современные жесткие диски могут иметь по несколько десятков тысяч цилиндров, в то время как на поверхности дискеты число дорожек (число цилиндров), как правило, составляет всего восемьдесят.

Каждый сектор состоит из **поля данных** и **поля служебной информации,** ограничивающей и идентифицирующей его. Размер сектора (точнее – емкость поля данных) устанавливается контроллером или драйвером. Пользовательский интерфейс DOS поддерживает единственный размер сектора – 512 байт [1]. BIOS

же непосредственно предоставляет возможности работы с секторами размером 128, 256, 512 или 1024 байт. Если осуществлять управление контроллером непосредственно, а не через программный интерфейс более высокого уровня (например, уровня DOS), то можно обрабатывать секторы и с другими размерами. Однако в большинстве современных ОС размер сектора выбирается равным 512 байтам.

Физический адрес сектора на диске определяется с помощью трех «координат», то есть представляется триадой [c-h-s], где c – номер цилиндра (дорожки на поверхности диска, cylinder), h – номер рабочей поверхности диска (магнитной головки, head), а s – номер сектора на дорожке. Номер цилиндра [c] лежит в диапазоне 0, …, c-1, где c – количество цилиндров. Номер рабочей поверхности диска [h] принадлежит диапазону 0, …, h-1, где h – число магнитных головок в накопителе. Номер сектора на дорожке [s] указывается в диапазоне 1, …, s, где s – количество секторов на дорожке. Например, триада [1-0-2] адресует сектор 2 на дорожке 0 (обычно верхняя рабочая поверхность) цилиндра 1.

Обмен информацией между ОЗУ и дисками физически осуществляется только секторами. Вся совокупность физических секторов на винчестере представляет его неформатированную емкость.

4.2.2 Логическая структура

Жесткий диск может быть разбит на несколько **разделов** (partition), которые могут использоваться либо одной ОС, либо различными ОС. Причем главным является то, что на каждом разделе может быть организована своя файловая система. Однако для организации даже одной-единственной файловой системы необходимо определить, по крайней мере, один раздел. Разделы диска могут быть двух типов – ***primary*** (обычно этот термин переводят как ***первичный)*** и ***extended*** (***расширенный***). Максимальное число primary-разделов равно четырем. При этом на диске обязательно должен быть, по крайней мере, один primary-раздел. Если primary-разделов несколько, то только один из них может быть активным. Именно загрузчику, расположенному в активном разделе, передается управление при включении ком-

пьютера и загрузке операционной системы. Остальные primary-разделы в этом случае считаются «невидимыми, скрытыми» (hidden).

Согласно спецификациям на одном жестком диске может быть только один ***extended-раздел,*** который, в свою очередь, может быть разделен на большое количество подразделов – ***логических дисков*** (***logical***). В этом смысле термин «первичный» следует признать не совсем удачным переводом слова primary; можно это слово перевести и как «простейший, примитивный». В этом случае становится понятным и логичным термин «extended».

Один из primary-разделов должен быть *активным,* именно с него должна загружаться программа загрузки операционной системы, или так называемый ***менеджер загрузки,*** назначение которого – загрузить программу загрузки ОС из какого-нибудь другого раздела и уже с ее помощью загружать операционную систему. Поскольку до загрузки ОС система управления файлами работать не может, то следует использовать для указания упомянутых загрузчиков исключительно абсолютные адреса в формате [c-h-s].

Операционная система назначает логическим дискам расширенных разделов имена (буквы), остающиеся после дисков первичных разделов. Так, если имеется один жесткий диск и у него есть первичные и вторичный разделы, причем последний разбит на два логических диска, мы увидим следующее [12]:

C: – первичный раздел;

D: – первый логический диск расширенного раздела;

E: – второй логический диск расширенного раздела.

Теперь если добавить второй жесткий диск (всего с одним первичным разделом, то картина изменится:

C: – первичный раздел первого диска (остался на месте);

D: – первичный раздел второго диска (новый);

E: – первый логический диск расширенного раздела первого диска (тот, что был D:);

F: – второй логический диск расширенного раздела первого диска (тот, что был E:).

Если бы у нового диска был расширенный раздел со своими логическими дисками, то они бы заняли следующие буквы (G:, H:, ...).

О механизме присвоения логических имен следует помнить, устанавливая программы на компьютер, которому эпизодически подключают дополнительные винчестеры. Незыблемое имя (C:) будет только у первичного раздела винчестера, подключенного к первому контроллеру ATA (если используется SCSI, то все немного сложнее).

По физическому адресу [0-0-1] на винчестере располагается ***главная загрузочная запись*** MBR (Master Boot Record,), содержащая ***внесистемный загрузчик*** NSB (Non-System Bootstrap) и ***таблицу разделов*** PT (Partition Table) [2]. Эта запись занимает ровно один сектор и размещается в памяти, начиная с адреса 0:7C00h, после чего управление передается коду, содержащемуся в первом секторе диска. Таким образом, в первом (стартовом) секторе физического жесткого диска находится не обычная запись boot record, как на дискете, а master boot record.

MBR является основным средством загрузки с жесткого диска, поддерживаемым BIOS. В MBR находятся три важных элемента:

1) программа начальной загрузки (внесистемный загрузчик). Именно она запускается BIOS после успешной загрузки в память первого сектора с MBR. Она не превышает 512 байт, и ее хватает только для загрузки следующей, чуть более сложной, программы – стартового сектора операционной системы, – и передачи ей управления;

2) таблица описания разделов диска, располагающаяся в MBR по смещению OxlBE и занимающая 64 байта;

3) сигнатура MBR. Последние два байта MBR должны содержать число AA55h. По наличию этой сигнатуры BIOS проверяет, что первый блок был загружен успешно. Сигнатура эта выбрана не случайно. Ее успешная проверка позволяет установить, что все линии передачи данных могут передавать и нули, и единицы.

Упрощенно структура MBR представлена в табл. 4.3.

Таблица 4.3 – Структура MBR

Смещение (Offset)	Размер (Sise), байт	Содержимое (Contents)
0	446	Программа анализа Partition Table и загрузки System Bootstrap с активного раздела жесткого диска
+ 1BEh	16	Partition 1 entry (Описатель раздела)
+ 1CEh	16	Partition 2 entry
+ 1DEh	16	Partition 3 entry
+1EEh	16	Partition 4 entry
+1FEh	2	Сигнатура (AA55h)

Таблица ***partition table*** описывает размещение и характеристики имеющихся на винчестере разделов. Можно сказать, что таблица разделов – одна из наиболее важных структур данных на жестком диске. Если эта таблица повреждена, то не только не будет загружаться операционная система (или одна из операционных систем, установленных на винчестере), но перестанут быть доступными и данные, расположенные на винчестере, особенно если жесткий диск был разбит на несколько разделов.

В табл. 4.3 показано, что в начале загрузочного сектора располагается программа анализа таблицы разделов и чтения первого сектора из активного раздела диска. Сама таблица partition table располагается в конце MBR, и для описания каждого раздела в этой таблице отводится по 16 байтов.

Первым байтом в элементе раздела идет флаг активности раздела boot indicator (0 – не активен, 128 (80H) – активен). Он определяет, является ли раздел системным загрузочным и есть ли необходимость производить загрузку операционной системы с него при старте компьютера. Активным может быть только один раздел. За флагом активности раздела следуют байт номера головки, с которой начинается раздел. За ним следует два байта, означающие соответственно номер сектора и номер цилиндра загрузочного сектора, где располагается первый сектор загрузчика операционной системы. Затем следует кодовый идентификатор System ID длиной в один байт, указывающий на при-

надлежность данного раздела к той или иной операционной системе и тип установленной на нем файловой системы.

За байтом кода операционной системы расположен байт номера головки конца раздела, за которым идут два байта – номер сектора и номер цилиндра последнего сектора данного раздела. Ниже представлен формат элемента таблицы разделов (табл. 4.4).

Таблица 4.4 – Формат элемента таблицы разделов

Название записи элемента Partition Table	Длина, байт
Флаг активности раздела	1
Номер головки начала раздела	1
Номер сектора и номер цилиндра загрузочного сектора раздела	2
Кодовый идентификатор операционной системы	1
Номер головки конца раздела	1
Номер сектора и цилиндра последнего сектора раздела	2
Младшее и старшее двухбайтовое слово относительного номера начального сектора	4
Младшее и старшее двухбайтовое слово размера раздела в секторах	4

В табл. 4.5 приведены наиболее известные идентификаторы.

Вслед за сектором MBR размещаются собственно сами разделы. В процессе начальной загрузки сектора MBR, содержащего таблицу partition table, работают программные модули BIOS. Начальная загрузка считается выполненной корректно только в том случае, когда таблица разделов содержит допустимую информацию.

В MS DOS в первичном разделе может быть сформирован только один логический диск, а в расширенном – любое их количество. Каждый логический диск «управляется» своим логическим приводом. Каждому логическому диску на винчестере

соответствует своя (относительная) ***логическая*** нумерация. Физическая же адресация жесткого диска сквозная.

Таблица 4.5 – Типы разделов жесткого диска

System ID, идентификатор	Тип раздела	System ID, идентификатор	Тип раздела
00	Empty («пустой» раздел)	41	PPC PreP Boot
01	FAT 12	42	SFS
02	XENIX root	4D	QNX 4.x
03	XENIX usr	4E	QNX 4.x 2nd part
04	FAT16 (<32 Мбайт)	4F	QNX 4.x 3nd part
05	Extended	50	OnTrack DM
06	FAT 16	51	OnTrack DM6 Aux
07	HPFS/NTFS	52	CP/M
08	AIX	53	OnTrack DM6
09	AIX bootable	54	OnTrack DM6
0A	OS/2 Boot Manager	55	EZ Drive
0B	Win95 FAT32	56	Golden Bou
0C	Win95 FAT32 LBA	5C	Priam Edisk
0E	Win95 FAT16 LBA	61	Speed Stor
0F	Win95 Extended	64	Novell Netware
10	OPUS	65	Novell Netware
11	Hidden FAT12	75	PC/IX
12	Compaq diagnost	80	Old Minix
14	Hidden FAT16 (<32 Мбайт)	82	Linux swap
16	Hidden FAT 16	83	Linux native
17	Hidden HPFS/NTFS	84	OS/2 hidden C:
18	AST Windows swap	85	Linux Extended
1B	Hidden Win95 Fat	86	NTFS volume set
1C	Hidden Win95 Fat	A5	BSD/386
1E	Hidden Win95 Fat	A6	Open BSD
24	NEC DOS	A7	Next Step
3C	Partition Magic	EB	Be OS
40	Venix 80286		

Первичный раздел DOS включает только ***системный логический диск*** без каких-либо дополнительных информационных структур.

Расширенный раздел DOS содержит вторичную запись SMBR (Secondary MBR), в состав которой вместо partition table входит таблица логического диска LDT (Logical Disk Table), ей аналогичная. Таблица LDT описывает размещение и характеристики раздела, содержащего единственный логический диск, а также может специфицировать следующую запись SMBR. Следовательно, если в расширенном разделе DOS создано К логических дисков, то он содержит К экземпляров SMBR, связанных в список. Каждый элемент этого списка описывает соответствующий логический диск и ссылается (кроме последнего) на следующий элемент списка.

Для работы с разделами жесткого диска в MS DOS, Windows 9x используйте утилиту fdisk.exe, а в Windows на платформе NT используйте «Control Panel», далее «Administrative Tools», «Computer Management». Существует также очень удобная, но не входящая в состав операционных систем, утилита Partition Magic (фирмы Power Quest), предназначенная для работы с разделами жестких дисков. Она выполняет такие функции с разделами, как форматирование под различные файловые системы, перемещение, изменение размеров, активизацию, скрытие и т.д.

4.3 Файловые системы

4.3.1 Функции файловой системы ОС

Под **файлом** обычно понимают набор данных, организованных в виде совокупности записей одинаковой структуры. Для управления этими данными создаются соответствующие системы управления файлами. Возможность иметь дело с логическим уровнем структуры данных и операций, выполняемых над ними в процессе их обработки, предоставляет файловая система. Таким образом, **файловая система** – это 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 управление ресурсами, которые используются файлами [2]. Именно файловая система определяет способ организации данных на диске или каком-нибудь ином носителе данных. В качестве примера можно привести файловую систему FAT (File Allocation Table), реализация для которой имеется в абсолютном большинстве ОС, работающих в современных ПК.

Как правило, все современные ОС имеют соответствующие системы управления файлами. **Система управления файлами** является основной подсистемой в абсолютном большинстве современных операционных систем, хотя в принципе можно обходиться и без нее. Во-первых, через систему управления файлами связываются все системные обрабатывающие программы. Во-вторых, с помощью этой системы решаются проблемы централизованного распределения дискового пространства и управления данными. В-третьих, благодаря использованию той или иной системы управления файлами пользователям предоставляются следующие возможности:

- создание, удаление, переименование (другие операции) именованных наборов данных (именованных файлов) посредством собственных программ или специальных управляющих программ, реализующих функции интерфейса пользователя с его данными;
- работа с недисковыми периферийными устройствами как с файлами;
- обмен данными между файлами, устройствами, между файлом и устройством (и наоборот);
- работа с файлами с помощью обращений к программным модулям системы управления файлами;
- защита файлов от несанкционированного доступа.

В некоторых ОС может быть несколько систем управления файлами, что обеспечивает им возможность работы с несколькими файловыми системами. Очевидно, что системы управления файлами, будучи компонентом ОС, не являются независимыми от этой ОС, поскольку они активно используют соответствующие вызовы прикладного программного интерфейса API (Application Program Interface). С другой стороны, системы

управления файлами сами дополняют API новыми вызовами. Можно сказать, что основное назначение файловой системы и соответствующей ей системы управления файлами – организация удобного доступа к данным, организованным как файлы; то есть вместо низкоуровневого доступа к данным с указанием конкретных физических адресов нужной нам записи используется логический доступ с указанием имени файла и записи в нем. Другими словами, термин «файловая система» определяет, прежде всего, принципы доступа к данным, организованным в файлы. Этот же термин часто используют и по отношению к конкретным файлам, расположенным на том или ином носителе данных. А термин «система управления файлами» следует употреблять по отношению к конкретной реализации файловой системы, т.е. СУФ – это комплекс программных модулей, обеспечивающих работу с файлами в конкретной операционной системе.

Следует еще раз заметить, что любая система управления файлами не существует сама по себе – она разработана для работы в конкретной ОС. В качестве примера можно сказать, что всем известная файловая система FAT имеет множество реализаций как система управления файлами. Так, система, получившая это название и разработанная для первых персональных компьютеров, называлась просто FAT (сейчас ее называют FAT-12). Ее разрабатывали для работы с дискетами, и некоторое время она использовалась при работе с жесткими дисками. Затем она была усовершенствована для работы с жесткими дисками большего объема, и эта новая реализация получила название FAT-16. Это название файловой системы мы используем и по отношению к системе управления файлами самой MS DOS. Реализацию же системы управления файлами для OS/2, которая использует основные принципы системы FAT, называют super-FAT, основное отличие которой состоит в возможности поддерживать для каждого файла расширенные атрибуты. Есть версия системы управления файлами с принципами FAT и для Windows 9x, для Windows NT и т.д. Другими словами, для работы с файлами, организованными в соответствии с некоторой файловой системой, для каждой ОС должна быть создана соответствую-

щая система управления файлами, которая будет работать только в конкретной операционной системе, для нее предназначенной; но при этом она позволит работать с файлами, созданными с помощью системы управления файлами другой ОС, работающей по тем же основным принципам файловой системы.

4.3.2 Файловая система FAT

4.3.2.1 FAT16

Файловая система FAT16 была разработана еще до создания MS DOS и в настоящее время поддерживается всеми операционными системами Microsoft для обеспечения совместимости. Ее название – таблица расположения файлов (File Allocation Table) – отлично отражает физическую организацию файловой системы, к основным характеристикам которой можно отнести максимальный размер поддерживаемого тома (жесткого диска или раздела на жестком диске), не превышающий 4095 Мбайт. В период эксплуатации MS DOS 4-гигабайтные жесткие диски казались несбыточной мечтой (роскошью были диски объемом 20–40 Мбайт), поэтому такой запас был вполне оправданным [15].

FAT может состоять из 12- или 16-разрядных элементов. Для дисков с объемом менее 384 Кб очень эффективны 12-разрядные элементы. Файловая система такого диска в полном объеме помещается в один сектор (512 байтов). В настоящее время FAT12 используется для работы с гибкими магнитными дисками.

Том, отформатированный для использования FAT16, разделяется на кластеры. Размер кластера по умолчанию зависит от размера тома и может колебаться от 512 байт до 64 Кбайт. В табл. 4.6 показана зависимость размера кластера от размера тома. Отметим, что размер кластера может отличаться от значения по умолчанию, но должен иметь одно из значений, указанных в табл. 4.6 [15].

Таблица 4.6 – Зависимость размера тома от размера кластера в FAT16

Размер тома, Мбайт	Число секторов в кластере	Размер кластера, Кбайт
0–32	1	0,5 (512 байт)
33–64	2	1
65–128	4	2
129–255	8	4
256–511	16	8
512–1023	32	16
1024–2047	64	32
2048–4095	128	64

Не рекомендуется применять файловую систему FAT16 на томах размером больше 511 Мбайт, так как для относительно небольших по объему файлов дисковое пространство будет использоваться крайне неэффективно: файл размером в 1 байт будет занимать 64 Кбайт. Независимо от размера кластера файловая система FAT16 не поддерживается для томов размером больше 4 Гбайт.

На рис. 4.1 показано, как организован том при использовании файловой системы FAT16.

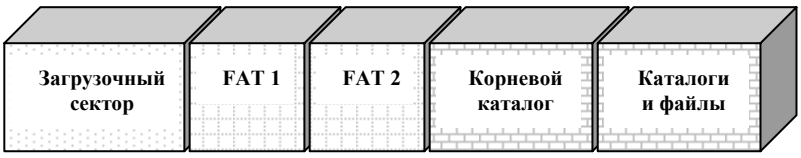

Рис. 4.1 – Структура тома при использовании FAT16

Первый сектор тома является загрузочным сектором. Далее за ним идут таблицы FAT1 и FAT2. Таблица FAT – это часть файловой системы FAT. Она содержит элементы, описывающие состояния кластеров в томе. FAT2 является копией FAT1. При использовании файловой системы FAT16 за второй копией таблицы FAT всегда располагается корневой каталог. Единственным различием между корневым каталогом и другими является

то, что корневой располагается в определенном месте и имеет фиксированное число вхождений. Каждый каталог и файл используют одно или более вхождений. Например, если число фиксированных вхождений для корневого каталога равно 512 и создано 100 подкаталогов, в корневом каталоге можно создать не более 412 файлов (512–100).

Для каждого файла и каталога в файловой системе хранится информация в соответствии со структурой, изображенной в табл. 4.7.

Таблица 4.7 – Структура элемента корневого каталога

Размер поля данных, байт	Содержание поля
11	Имя файла или каталога
1	Атрибуты файла
1	Резервное поле
3	Время создания
2	Дата создания
2	Дата последнего доступа
2	Зарезервированное
2	Время последней модификации
2	Дата последней модификации
2	Номер начального кластера в FAT
4	Размер файла

Каждый элемент каталога содержит номер начального кластера файла, описываемого данным элементом. Этот номер является указателем в FAT, где содержится информация об остальных кластерах файла, организованная в связный список.

В FAT16 кластеры могут иметь различное значение:

(0)000h свободный кластер,
(F)FF0h – (F)FF6h зарезервированный кластер,
(F)FF7h дефектный кластер,
(F)FF8h – (F)FFFh конец файла,
(0)002h – (F)FEFh номер следующего кластера файла.

Примечание: Старшая тетрада, заключенная в скобки, относится к 16-разрядным элементам. Например, дефектный кластер помечается FF7h в 12-разрядный FAT и FFF7h – в 16-разрядный FAT.

Расположение файлов по кластерам показано на рис. 4.2: в папке расположены три файла; первый из них – File1 – занимает три кластера (файл не фрагментирован, кластеры 2, 3 и 4 расположены последовательно); второй файл – File2 – фрагментирован и располагается в кластерах 5, 6 и 8; третий – File3 – занимает всего один кластер. Вхождение для каждого файла содержит адрес его начального кластера (2, 5 и 7 соответственно). Последний кластер каждого файла (4, 8 и 7) в качестве адреса следующего кластера содержит значение FFFF, указывающее на то, что это последний кластер для данного файла.

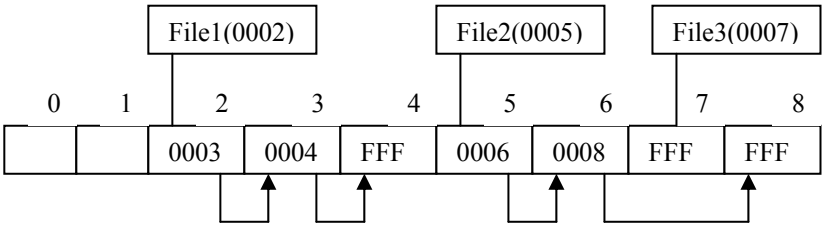

Рис. 4.2 – Пример расположения файлов по кластерам в FAT16

Так как все вхождения имеют одинаковый размер информационного блока, они различаются по байту атрибутов. Один из битов в данном байте может указывать, что это каталог, другой – что это метка тома. Для пользователей доступны четыре бита, позволяющие управлять атрибутами файла: архивный (archive), системный (system), скрытый (hidden), доступный только для чтения (read-only).

Среди ***преимуществ FAT16*** можно отметить следующие [15]:

- файловая система поддерживается операционными системами MS DOS, Windows 95, Windows 98, Windows NT, Windows 2000, а также некоторыми операционными системами UNIX;

- существует большое число программ, позволяющих исправлять ошибки в этой файловой системе и восстанавливать данные;
- при возникновении проблем с загрузкой с жесткого диска система может быть загружена с флоппи-диска;
- данная файловая система достаточно эффективна для томов объемом менее 256 Мбайт.

К ***основным недостаткам FAT16*** относятся следующие [15]:

- корневой каталог не может содержать более 512 элементов. Использование длинных имен файлов существенно сокращает число этих элементов;
- FAT16 поддерживает не более 65536 кластеров, а так как некоторые кластеры зарезервированы операционной системой, то число доступных кластеров составляет 65524. Каждый кластер имеет фиксированный размер для данного логического устройства. При достижении максимального числа кластеров с максимальным размером в 32 килобайта максимальный объем поддерживаемого тома ограничивается 4-гигабайтами под управлением Windows 2000. Для поддержания совместимости с MS DOS, Windows 95 и Windows 98 объем тома под FAT16 не должен превышать 2 Гбайт;
- не поддерживается резервная копия загрузочного сектора;
- в FAT16 не поддерживается встроенная защита файлов и их сжатие;
- на дисках большого объема теряется много места за счет того, что используется максимальный размер кластера. Место под файл выделяется исходя из размера не файла, а кластера.

4.3.2.2 FAT32

В версии Microsoft Windows 95 OEM Service Release 2 (OSR2) в Windows появилась поддержка 32-битной FAT. Для систем на базе Windows NT эта файловая система впервые стала поддерживаться в Microsoft Windows 2000. Если FAT16 может поддерживать тома объемом до 4 Гбайт, то FAT32 способна обслуживать тома объемом до 4 Тбайт. Размер кластера в FAT32 может изменяться от 1 (512 байт) до 64 секторов (32 Кбайт).

Для хранения значений кластеров FAT32 требуется 4 байта (32 бит, а не 16, как в FAT16). Это означает, в частности, что некоторые файловые утилиты, рассчитанные на FAT16, не могут работать с FAT32.

Основным отличием FAT32 от FAT16 является изменение размера логического раздела диска. При этом если при использовании FAT16 с 2-гигабайтными дисками требовался кластер размером в 32 Кбайт, то в FAT32 кластер размером в 4 Кбайт подходит для дисков объемом от 512 Мбайт до 8 Гбайт (табл. 4.8) [15]. Это соответственно означает более эффективное использование дискового пространства – чем меньше кластер, тем меньше места требуется для хранения файла и, как следствие, диск реже становится фрагментированным.

Таблица 4.8 – Зависимость размера тома от размера кластера в FAT32

Размер раздела, Гбайт	Размер кластера по умолчанию, Кбайт
Менее 8	4
От 8 до 16	8
От 16 до 32	16
32 и более	32

При применении FAT32 максимальный размер файла может достигать 4 Гбайт минус 2 байта. Если при использовании FAT16 максимальное число вхождений в корневой каталог ограничивалось 512, то FAT32 позволяет увеличить это число до 65 535.

FAT32 накладывает ограничения на минимальный размер тома – не менее 65527 кластеров. При этом размер кластера не может быть таким, при котором бы FAT занимала более 16 Мбайт – 64 Кбайт/4 или 4 млн. кластеров.

При использовании длинных имен файлов данные, необходимые для доступа из FAT16 и FAT32, не перекрываются. При создании файла с длинным именем Windows создает соответ-

ствующее имя в формате 8.3 и одно или более вхождений в каталог для хранения длинного имени (по 13 символов из длинного имени файла на каждое вхождение). Каждое последующее вхождение хранит соответствующую часть имени файла в формате Unicode. Такие вхождения имеют атрибуты «идентификатор тома», «только чтение», «системный» и «скрытый». Атрибут вхождения «скрытый» характеризует набор, игнорируемый MS DOS, в которой доступ к файлу осуществляется по его «псевдониму» в формате 8.3 (восемь символов – имя файла, три – расширение файла).

Среди ***преимуществ FAT32*** можно отметить следующие [15]:

- выделение дискового пространства выполняется более эффективно, особенно для дисков большого объема;
- корневой каталог в FAT32 представляет собой обычную цепочку кластеров и может находиться в любом месте диска.
- за счет использования кластеров меньшего размера (4 Кбайт на дисках объемом до 8 Гбайт) занятое дисковое пространство обычно на 10–15 % меньше, чем под FAT16;
- FAT32 является более надежной файловой системой. В частности, она поддерживает возможность перемещения корневого каталога и использования резервной копии FAT. Кроме того, загрузочная запись содержит ряд критичных для файловой системы данных.

Основные недостатки FAT32 [15]:

- размер тома при использовании FAT32 под Windows 2000 ограничен 32 Гбайт;
- тома FAT32 недоступны из многих операционных систем, которые поддерживают FAT;
- не поддерживается резервная копия загрузочного сектора;
- в FAT32 не поддерживается встроенная защита файлов и их сжатие.

4.3.3 Файловая система NTFS

В названии файловой системы NTFS (New Technology File System) содержатся слова «новая технология». Действительно, NTFS характеризуется рядом значительных усовершенствова-

ний и изменений, существенно отличающих ее от других файловых систем. С точки зрения пользователей файлы по-прежнему хранятся в каталогах, часто называемых «папками» или ***фолдерами*** в среде Windows. Однако в NTFS, в отличие от FAT, работа на дисках большого объема происходит намного эффективнее: имеются средства для ограничения в доступе к файлам и каталогам; введены механизмы, существенно повышающие надежность файловой системы; сняты многие ограничения на максимальное количество дисковых секторов и/или кластеров.

При проектировании системы NTFS особое внимание было обращено на следующие характеристики:

надежность. Высокопроизводительные компьютеры и системы совместного пользования (серверы) должны обладать повышенной надежностью, которая является ключевым элементом структуры и поведения NTFS. Одним из способов увеличения надежности является введение механизма транзакций, при котором осуществляется ***журналирование*** файловых операций;

расширенную функциональность. NTFS проектировалась с учетом возможного расширения. В ней были реализованы многие дополнительные возможности: усовершенствованная отказоустойчивость; эмуляция других файловых систем; мощная модель безопасности; параллельная обработка потоков данных; создание файловых атрибутов, определяемых пользователем;

поддержку платформенно-независимого системного интерфейса для компьютерного окружения POSIX (Portable Operating System Interface for Computer Environments). Поскольку правительство США требовало, чтобы все закупаемые им системы хотя бы в минимальной степени соответствовали стандарту POSIX, такая возможность была предусмотрена и в NTFS. К числу базовых средств файловой системы POSIX относится необязательное использование имен файлов с учетом регистра, хранение времени последнего обращения к файлу и механизм так называемых «жестких ссылок» (альтернативных имен, позволяющих ссылаться на один и тот же файл по двум и более именам);

гибкость. Модель распределения дискового пространства в NTFS отличается чрезвычайной гибкостью. Размер кластера может изменяться от 512 байт до 64 Кбайт; он представляет собой число, кратное внутреннему кванту распределения дискового пространства. NTFS также поддерживает длинные имена файлов, набор символов Unicode и альтернативные имена формата 8.3 для совместимости с FAT.

Как и при использовании FAT, основной информационной единицей в NTFS является кластер. В табл. 4.9 показаны размеры кластеров по умолчанию для томов различной емкости [15].

Таблица 4.9 – Зависимость размера тома от размера кластера в NTFS

Размер тома, Мбайт	Число секторов в кластере	Размер кластера, Кбайт
512 и менее	1	0,5 (512 байт)
513–1024 (1Гбайт)	2	1
1025–2048 (2Гбайт)	4	2
Более 2049	8	4

Теоретически NTFS поддерживает тома с числом кластеров до 2^{32}, но тем не менее помимо отсутствия жестких дисков такого объема существуют и другие ограничения на максимальный размер тома. Одним из таких ограничений является таблица разделов. Индустриальные стандарты ограничивают размер таблицы разделов 2^{32} секторами. Другим ограничением является размер сектора, который обычно равен 512 байтам. Поскольку размер сектора может измениться в будущем, текущий размер дает ограничение на размер одного тома – 2 Тбайт ($2^{32} \times 512$ байт = 2^{41}). Таким образом, размер тома в 2 Тбайт является практическим пределом для физических и логических томов NTFS.

Управление доступом к файлам и каталогам. При использовании томов NTFS можно устанавливать права доступа к файлам и каталогам. Эти права доступа указывают, какие поль-

зователи и группы имеют доступ к ним и какой уровень доступа допустим. Такие права доступа распространяются как на пользователей, работающих за компьютером, на котором располагаются файлы, так и на пользователей, обращающихся к файлам через сеть, когда файл располагается в каталоге, открытом для удаленного доступа. Под NTFS можно также устанавливать разрешения на удаленный доступ, объединяемые с разрешениями на доступ к файлам и каталогам. Помимо этого файловые атрибуты (только чтение, скрытый, системный) также ограничивают доступ к файлу. Под управлением FAT16 и FAT32 тоже можно устанавливать атрибуты файлов, но они не обеспечивают права доступа к файлам. В версии NTFS, используемой в Windows 2000, появился новый тип разрешения на доступ – наследуемые разрешения.

Сжатие файлов и каталогов. В Windows 2000 поддерживается сжатие файлов и каталогов, расположенных на NTFS-томах. Сжатые файлы доступны для чтения и записи любыми Windows-приложениями. Для этого нет необходимости в их предварительной распаковке. Используемый алгоритм сжатия схож с тем, который используется в Double-Space (MS DOS 6.0) и DriveSpace (MS DOS 6.22), но имеет одно существенное отличие – под управлением MS DOS выполняется сжатие целого первичного раздела или логического устройства, тогда как под NTFS можно упаковывать отдельные файлы и каталоги.

Алгоритм сжатия в NTFS разработан с учетом поддержки кластеров размером до 4 Кбайт. Если величина кластера больше 4 Кбайт, функции сжатия NTFS становятся недоступными.

Самовосстановление NTFS. Файловая система NTFS обладает способностью самовосстановления и может поддерживать свою целостность за счет использования протокола выполняемых действий и ряда других механизмов. NTFS рассматривает каждую операцию, модифицирующую системные файлы на NTFS-томах, как транзакцию и сохраняет информацию о такой транзакции в протоколе. Начатая транзакция может быть либо полностью завершена (commit), либо откатывается (rollback). В последнем случае NTFS-том возвращается в состояние, предшествующее началу транзакции. Для того чтобы управлять транзакциями, перед тем как осуществить запись на диск, NTFS запи-

сывает все операции, входящие в транзакцию, в файл протокола. После того как транзакция завершена, все операции выполняются. Таким образом, под управлением NTFS не может быть незавершенных операций. В случае дисковых сбоев незавершенные операции просто отменяются. Под управлением NTFS также выполняются операции, позволяющие «на лету» определять дефектные кластеры и отводить новые кластеры для файловых операций. Этот механизм называется cluster remapping.

При формировании файловой системы NTFS программа форматирования создает файл MFT (Master File Table) и другие области для хранения метаданных. Метаданные используются NTFS для реализации файловой структуры. Первые 16 записей в MFT зарезервированы самой NTFS. Местоположение файлов метаданных $Mft и SMftMirr записано в загрузочном секторе диска. Если первая запись в MFT повреждена, NTFS считывает вторую запись для нахождения копии первой. Полная копия загрузочного сектора располагается в конце тома. Основные метаданные, хранимые в MFT, перечислены в табл. 4.10 [15].

Таблица 4.10 – Основные метаданные MFT

Системные файлы	Имя файла	Запись MFT	Содержание записи MFT
Master file table	$Mft	0	Одна базовая файловая запись для каждого файла или каталога на томе NTFS
Master file table2	$MftMirr	1	Копия первых четырех записей MFT. Гарантирует доступ к MFT в случае, если первый сектор поврежден
Log file	$LogFile	2	Список действий, необходимый для восстановления NTFS
Volume	$Volume	3	Информация о томе – метка и номер версии
Attribute definitions	$AttrDef	4	Таблица имен атрибутов и описание

Окончание табл. 4.10

Системные файлы	Имя файла	Запись MFT	Содержание записи MFT
Root file name index	$	5	Корневой каталог
Cluster bitmap	$Bitmap	6	Информация о занятых кластерах
Boot sector	$Boot	7	Код загрузки для загрузочных томов
Bad cluster file	$BadClus	8	Информация о дефектных кластерах
Security file	$Secure	9	Уникальные дескрипторы для всех файлов
Upcase table	$Upcase	10	Информация для преобразования символов нижнего регистра в соответствующие Unicode-символы верхнего регистра
NTFS extension file	$Extend	11	Информация для различных служб ОС: службы квот, службы пересчета и идентификаторы объектов
		12-15	Зарезервированные записи для будущих версий

Остальные записи MFT содержат записи для каждого файла и каталога, расположенных на данном томе.

Обычно один файл использует одну запись в MFT, но если у файла большой набор атрибутов или он становится слишком фрагментированным, то для хранения информации о нем могут потребоваться дополнительные записи. В этом случае первая запись о файле, называемая базовой записью, хранит местоположение других записей. Данные о файлах и каталогах небольшого размера (до 1500 байт) полностью содержатся в первой записи.

Каждый занятый сектор на NTFS-томе принадлежит тому или иному файлу. Даже метаданные файловой системы являются частью файла. NTFS рассматривает каждый файл (или каталог) как набор файловых атрибутов. Такие элементы, как имя файла,

информация о его защите и даже данные в нем, являются атрибутами файла. Каждый атрибут идентифицируется кодом определенного типа и именем атрибута.

Если атрибуты файла вмещаются в файловую запись, они называются резидентными атрибутами. Такими атрибутами всегда являются имя файла и дата его создания. В тех случаях, когда информация о файле слишком велика, чтобы вместиться в одну MFT-запись, некоторые атрибуты файла становятся нерезидентными. Резидентные атрибуты хранятся в одном или более кластерах и представляют собой поток альтернативных данных для текущего тома. Для описания местонахождения резидентных и нерезидентных атрибутов NTFS создает атрибут Attribute List.

Возможности файловой системы NTFS по ограничению доступа к файлам и каталогам. Благодаря наличию механизма расширенных атрибутов в NTFS реализованы ограничения в доступе к файлам и каталогам. Эти дополнительные атрибуты, использованные для ограничения в доступе к файловым объектам, назвали атрибутами безопасности. При каждом обращении к такому объекту сравнивается специальный список дискреционных прав доступа, приписанный ему, со специальным системным идентификатором, несущим информацию об имени пользователя, осуществляющего текущий запрос к файлу или каталогу. Если имеется в списке необходимое разрешение, то действие выполняется, в противном случае система сообщает об отказе.

Файловая система NTFS имеет так называемые индивидуальные разрешения, которые могут быть приписаны любому файлу и/или каталогу: **Read** *(прочитать),* **Write** *(записать),* **eXecute** *(выполнить),* **Delete** *(удалить),* **Change Permissions** *(изменить разрешения)* и **Take Ow-nership** *(стать владельцем).* Соответствующие этим разрешениям действия можно выполнять только в случаях, когда для данного пользователя или группы, к которой он принадлежит, имеется одноименное разрешение. Другими словами, если для некоторого файла указано, что все пользователи могут его читать и исполнять, то только эти действия и можно с ним сделать, если при этом не указано, что для какой-нибудь другой группы пользователей (отдельного

пользователя) имеются другие разрешения. Комбинации индивидуальных разрешений и определяют действия, которые могут быть выполнены с файлом или каталогом.

Изначально всему диску, а значит, и файлам, которые на нем создаются, присвоены все индивидуальные разрешения для группы Everyone (все). Это означает, что любой пользователь, имея полный набор индивидуальных разрешений на файлы и каталоги, может изменять их по своему усмотрению, т.е. ограничивать других пользователей в правах доступа к тому или иному объекту. Если изменить разрешения на каталог, то новые файлы, создаваемые в нем, будут получать и соответствующие разрешения: они будут наследовать разрешения своего родительского каталога.

Каталоги обычно обладают теми же разрешениями, что и находящиеся в них файлы и папки, хотя у каждого файла могут быть свои разрешения. Разрешения, которые имеются у файла, имеют приоритет над разрешениями, которые установлены на каталог, в котором находится этот файл. Например, если вы создаете каталог внутри другого каталога, для которого администраторы обладают правом полного доступа, а пользователи – правом чтения, то новый каталог унаследует эти права. То же относится и к файлам, копируемым из другого каталога или перемещаемым из другого раздела NTFS.

Если каталог или файл перемещается в другой каталог того же раздела NTFS, то атрибуты безопасности не наследуются от нового каталога. Дело в том, что при перемещении файлов в границах одного раздела NTFS изменяется только указатель местонахождения объекта, а все остальные атрибуты (включая атрибуты безопасности) остаются без изменений.

Существует три важных правила, которые помогут определить состояние прав доступа при перемещении или копировании объектов NTFS:

1) при перемещении файлов в границах раздела NTFS сохраняются исходные права доступа;

2) при выполнении других операций (создании или копировании файлов, а так же их перемещении между разделами NTFS) наследуются права доступа родительского каталога;

3) при перемещении файлов из раздела NTFS в раздел FAT все права NTFS теряются.

Основные отличия FAT и NTFS

Если говорить о дополнительных расходах на хранение служебной информации, то можно отметить, что FAT отличается от NTFS большей компактностью и меньшей сложностью. В большинстве томов FAT на хранение таблицы размещения, содержащей информацию обо всех файлах тома, расходуется менее 1 Мбайт. Столь низкие дополнительные расходы позволяют форматировать в FAT жесткие диски малого объема и флоппи-диски. В NTFS служебные данные занимают больше места, чем в FAT. Так, каждый элемент каталога занимает 2 Кбайт. Однако это имеет и свои преимущества, так как содержимое файлов объемом 1500 байт и менее может полностью храниться в элементе каталога.

Система NTFS не может использоваться для форматирования флоппи-дисков. Не стоит пользоваться ею для форматирования разделов объемом менее 50–100 Мбайт. Относительно высокие дополнительные расходы приводят к тому, что для малых разделов служебные данные могут занимать до 25 % объема носителя.

Следующий критерий сравнения – размер файлов. Разделы FAT имеют объем до 2 Гбайт, FAT32 – до 4 Тбайт. Тем не менее из-за особенностей своего внутреннего строения разделы FAT лучше всего работают для разделов объемом 200 Мбайт и менее. В NTFS в настоящее время из-за аппаратных и других системных причин размер файлов ограничивается 2 терабайтами.

Разделы FAT могут использоваться практически во всех операционных системах. За редкими исключениями, с разделами NTFS можно работать напрямую только из Windows NT, хотя и имеются для ряда ОС соответствующие реализации систем управления файлами для чтения файлов из томов NTFS. Так, например, утилита (драйвер) NTFSDOS позволяет читать данные NTFS на компьютере, загруженном в режиме MS DOS. Однако полноценных реализаций для работы с NTFS вне системы Windows NT пока нет.

Разделы FAT не обеспечивают локальной безопасности, в то время как разделы NTFS обеспечивают локальную безопасность как файлов, так и каталогов. Для разделов FAT могут устанавливаться общие права, связанные с общим доступом к каталогам в сети. Однако такая защита не помешает пользователю с локальным входом получить доступ к файлам своего компьютера. В вопросе организации системы безопасности NTFS оказываются более предпочтительным вариантом. Разделы NTFS могут запрещать или ограничивать доступ как удаленных, так и локальных пользователей. Следовательно, к защищенным файлам смогут обратиться лишь те пользователи, которым были предоставлены соответствующие права.

Напомним, что Windows NT содержит специальную утилиту CONVERT.EXE, которая преобразует тома FAT в эквивалентные тома NTFS, однако для обратного преобразования из NTFS в FAT подобных утилит не существует. Чтобы выполнить обратное преобразование, необходимо создать раздел FAT, скопировать в него файлы из раздела NTFS и затем удалить оригиналы. Важно при этом не забывать и о том, что при копировании файлов из NTFS в FAT теряются все атрибуты безопасности NTFS. Напомним, что в FAT не предусмотрены средства для определения и последующего хранения этих атрибутов.

В последнее время появилось еще одно очень важное обстоятельство, связанное с тем, что объемы дисковых механизмов намного превысили максимально допустимый размер 8,4 Гбайт, приемлемый для FAT. Этот предел объясняется максимально возможными значениями в адресе сектора, для которого отводится всего 3 байта. Поэтому в подавляющем большинстве случаев при работе в среде Windows-систем используют либо FAT32, либо NTFS. Последняя, безусловно, лучше, но она не поддерживается в широко распространенных ОС Windows 98 и ныне все более часто встречающейся Windows Millennium Edition.

4.3.4 Файловая система HPFS

HPFS (High Performance File System) – высокопроизводительная файловая система – впервые появилась в OS/2 1.2 и

LAN Manager [2]. HPFS была разработана совместными усилиями лучших специалистов компании IBM и Microsoft на основе опыта IBM по созданию файловых систем MVS, VM/CMS и виртуального метода доступа. Архитектура HPFS начала создаваться как файловая система, которая сможет использовать преимущества многозадачного режима и обеспечит в будущем более эффективную и надежную работу с файлами на дисках большого объема.

HPFS стала первой файловой системой для ПК, в которой была реализована поддержка длинных имен. HPFS, как FAT и многие другие файловые системы, обладает структурой каталогов, но в ней также предусмотрены автоматическая сортировка каталогов и специальные расширенные атрибуты, упрощающие реализацию безопасности файлового уровня и создание множественных имен. HPFS поддерживает те же самые атрибуты, что и файловая система FAT, а также и новую форму file-associated, то есть информацию, называемую **расширенными атрибутами**. Каждый расширенный атрибут концептуально подобен переменной окружения. Но самым главным отличием систем FAT и HPFS являются базовые принципы хранения информации о местоположении файлов.

Принципы размещения файлов на диске, положенные в основу HPFS, увеличивают как производительность файловой системы, так и ее надежность и отказоустойчивость. Для достижения этих целей предложено несколько способов: размещение каталогов в середине дискового пространства; использование методов бинарных сбалансированных деревьев для ускорения поиска информации о файле; рассредоточение информации о местоположении записей файлов по всему диску, при том, что записи каждого конкретного файла размещаются по возможности в смежных секторах и поблизости от данных об их местоположении. Действительно, система HPFS стремится прежде всего к тому, чтобы расположить файл в смежных кластерах, или, если такой возможности нет, разместить его на диске таким образом, чтобы ***экстенты*** (фрагменты) файла физически были как можно ближе друг к другу. Такой подход существенно уменьшает ***время позиционирования*** головок записи/чтения жесткого диска и ***время ожидания*** (rotational latency), т.е. время задержки

между установкой головки чтения/записи на нужную дорожку диска и началом чтения данных с диска. Можно сказать, что файловая система HPFS обладает по сравнению с FAT следующими основными преимуществами:

- высокой производительностью;
- надежностью;
- возможностью работы с расширенными атрибутами, что позволяет управлять доступом к файлам и каталогам;
- возможностью эффективного использования дискового пространства.

Все эти преимущества обусловлены структурой диска HPFS. В начале диска расположено несколько управляющих блоков. Все остальное дисковое пространство в HPFS разбито на части: полосы, ленты из смежных секторов, band). Каждая такая группа данных занимает на диске пространство в 8 Мбайт и имеет свою собственную битовую карту распределения секторов, показывающую, какие секторы данной полосы заняты, а какие – свободны. Каждому сектору ленты данных соответствует один бит в ее битовой карте. Если бит имеет значение 1, то соответствующий сектор занят, а если 0 – свободен.

Битовые карты двух полос располагаются на диске рядом, так же располагаются и сами полосы, то есть последовательность полос и карт выглядит следующим образом: битовая карта, битовая карта, лента с данными, лента с данными, битовая карта, битовая карта и т.д. Такое расположение лент позволяет непрерывно разместить на жестком диске файл размером до 16 Мбайт и в то же время не удалять от самих файлов информацию об их местонахождении.

Дисковое пространство в HPFS выделяется не кластерами, как в FAT, а ***блоками***. В современной реализации размер блока взят равным одному сектору, но в принципе он мог бы быть и иного размера. По сути дела, блок – это и есть кластер. Размещение файлов в таких небольших блоках позволяет более эффективно использовать пространство диска, так как непроизводительные потери свободного места составляют в среднем всего 256 байт на каждый файл. Вспомните, что чем больше размер кластера, тем больше места на диске расходуется напрасно. Например, кластер на отформатированном под FAT диске объемом

от 512 до 1024 Мбайт имеет размер 16 Кбайт. Следовательно, непродуктивные потери свободного пространства на таком разделе в среднем составляют 8 Кбайт (8192 байт) на один файл, в то время как на разделе HPFS эти потери всегда будут составлять всего 256 байт на файл. Таким образом, на каждый файл экономится почти 8 Кбайт.

Помимо лент с записями файлов и битовых карт **в томе** с HPFS имеются еще три информационные структуры. Это так называемый загрузочный блок (boot block), дополнительный блок (super block) и запасной (резервный) блок (spare block). Загрузочный блок (boot block) располагается в секторах с 0 по 15; он содержит имя тома, его серийный номер, блок параметров BIOS и программу начальной загрузки. Программа начальной загрузки находит файл OS2LDR, считывает его в память и передает управление этой программе загрузки ОС, которая, в свою очередь, загружает с диска в память ядро OS/2 – OS2KRNL, и уже OS2KRNL с помощью сведений из файла CONFIG.SYS загружает в память все остальные необходимые программные модули и блоки данных.

В блоке (super block) содержится указатель на список битовых карт (bitmap block list). В этом списке перечислены все блоки на диске, в которых расположены битовые карты, используемые для обнаружения свободных секторов. Также в дополнительном блоке хранится указатель на список дефектных блоков (bad block list), указатель на группу каталогов (directory band), указатель на файловый узел (F-node) корневого каталога, а также дата последней проверки раздела программой CHKDSK. В списке дефектных блоков перечислены все поврежденные секторы (блоки) диска. Когда система обнаруживает поврежденный блок, он вносится в этот список и для хранения информации больше не используется. Кроме этого, в структуре super block содержится информация о размере полосы. В текущей реализации HPFS размер полосы т равен 8 Мбайт. Блок super block размещается в секторе с номером 16 логического диска, на котором установлена файловая система HPFS.

Резервный блок (spare block) содержит указатель на карту аварийного замещения (hotfix map или hotfix-areas), указатель на список свободных запасных блоков (directory emergency free

block list), используемых для операций на почти переполненном диске, и ряд системных флагов и дескрипторов. Этот блок размещается в 17 секторе диска. Резервный блок обеспечивает высокую отказоустойчивость файловой системы HPFS и восстановление поврежденных данных на диске.

Файлы и каталоги в HPFS базируются на фундаментальном объекте, называемом F-Node [2]. Эта структура характерна для HPFS и аналога в файловой системе FAT не имеет. Каждый файл и каталог диска имеет свой файловый узел F-Node. Каждый объект F-Node занимает один сектор и всегда располагается поблизости от своего файла или каталога (обычно непосредственно перед файлом или каталогом). Объект F-Node содержит информацию о длине и первых 15 символах имени файла, специальную служебную информацию, статистику по доступу к файлу, расширенные атрибуты файла и список прав доступа (или только часть этого списка в случае большого размера), ассоциативную информацию о расположении и подчинении файла и т.д. Структура распределения в F-node может принимать несколько форм в зависимости от размера каталога или файлов. HPFS просматривает файл как совокупность одного или более секторов. Из прикладной программы это не видно; файл появляется как непрерывный поток байтов. Если расширенные атрибуты слишком велики для файлового узла, то в него записывается указатель на них.

В HPFS структура каталога представляет собой сбалансированное дерево с записями, расположенными в алфавитном порядке. Каждая запись, входящая в состав B-Tree дерева, содержит: атрибуты файла; указатель на соответствующий файловый узел; информацию о времени и дате создания файла, времени и дате последнего обновления и обращения, длине данных, содержащих расширенные атрибуты; счетчик обращений к файлу; информацию о длине имени файла и само имя и другую информацию.

Файловая система HPFS при поиске файла в каталоге просматривает только необходимые ветви двоичного дерева (B-Tree). Такой метод во много раз эффективнее, чем последовательное чтение всех записей в каталоге, что имеет место в системе FAT. Для того чтобы найти искомый файл в каталоге (точ-

нее, указатель на его информационную структуру F-node), организованном на принципах сбалансированных двоичных деревьев, большинство записей вообще читать не нужно. В результате для поиска информации о файле необходимо выполнить существенно меньшее количество операций чтения диска.

Действительно, если, например, каталог содержит 4096 файлов, то файловая система FAT потребует чтения в среднем 64 секторов для поиска нужного файла внутри такого каталога, в то время как HPFS осуществит чтение всего только 2–4 секторов (в среднем) и найдет искомый файл. Несложные расчеты позволяют увидеть явные преимущества HPFS над FAT. Так, например, при использовании 40 входов на блок блоки каталога дерева с двумя уровнями могут содержать 1640 входов, а блоки каталога дерева с тремя уровнями – уже 65640 входов. Другими словами, некоторый файл может быть найден в типичном каталоге из 65640 файлов максимум за три обращения. Это намного лучше файловой системы FAT, где для нахождения файла нужно прочитать в худшем случае более 4000 секторов.

Размер каждого из блоков, в терминах которых выделяются каталоги в текущей реализации HPFS, равен 2 Кбайт. Размер записи, описывающей файл, зависит от размера имени файла. Если имя занимает 13 байтов (для формата 8.3), то блок из 2 Кбайт вмещает до 40 описателей файлов. Блоки связаны друг с другом посредством списковой структуры, как и описатели экстентов, для облегчения последовательного обхода.

При переименовании файлов может возникнуть так называемая перебалансировка дерева. Создание файла, переименование или стирание может приводить к каскадированию блоков каталогов. Фактически, переименование может потерпеть неудачу из-за недостатка дискового пространства, даже если файл непосредственно в размерах не увеличился. Во избежание такой ситуации HPFS поддерживает небольшой пул свободных блоков, которые могут использоваться при «аварии». Эта операция может потребовать выделения дополнительных блоков на заполненном диске. Указатель на пул свободных блоков сохраняется в SpareBlock.

Важное значение для повышения скорости работы с файлами имеет уменьшение их фрагментации. В HPFS считается, что

файл является фрагментированным, если он содержит больше одного экстента. Снижение фрагментации файлов сокращает время позиционирования и время ожидания за счет уменьшения количества перемещений головок, необходимых для доступа к данным файла. Алгоритмы работы файловой системы HPFS работают таким образом, чтобы по возможности размещать файлы в последовательных смежных секторах диска, что обеспечивает максимально быстрый доступ к данным впоследствии. В системе FAT, наоборот, запись следующей порции данных в первый же свободный кластер неизбежно приводит к фрагментации файлов. В системе HPFS также, если это предоставляется возможным, данные записываются в смежные секторы диска, а не в первый попавшийся. Это позволяет несколько снизить число перемещений головок чтения/записи от дорожки к дорожке. При этом когда данные дописываются в существующий файл, HPFS сразу же резервирует как минимум 4 Кбайт непрерывного пространства на диске. Если же часть этого пространства не потребовалась, то после закрытия файла она высвобождается для дальнейшего использования. Файловая система HPFS равномерно размещает непрерывные файлы по всему диску для того, чтобы впоследствии без фрагментации обеспечить их возможное увеличение. Если же файл не может быть увеличен без нарушения его непрерывности, HPFS вновь резервирует 4 Кбайт смежных блоков как можно ближе к основной части файла с целью сокращения времени позиционирования головок чтения/записи и времени ожидания соответствующего сектора.

Очевидно, что степень фрагментации файлов на диске зависит как от числа и размеров расположенных на нем файлов и размеров самого диска, так и от характера и интенсивности самих дисковых операций. Незначительная фрагментация файлов практически не сказывается на быстродействии операций с файлами. Файлы, состоящие из двух-трех экстентов, практически не снижают производительность HPFS, так как эта файловая система следит за тем, чтобы области данных, принадлежащие одному и тому же файлу, располагались как можно ближе друг к другу. Файл из трех экстентов имеет только два нарушения непрерывности, и, следовательно, для его чтения потребуется всего лишь два небольших перемещения головки диска. Програм-

мы (утилиты) дефрагментации, имеющиеся для этой файловой системы, по умолчанию считают наличие двух-трех экстентов у файла нормой. Например, программа HPFSOPT из набора утилит Gamma-Tech по умолчанию не дефрагментирует файлы, состоящие из трех и менее экстентов, а файлы, которые имеют большее количество экстентов, по-возможности приводятся к 2 или 3 экстентам. Файлы объемом в несколько десятков мегабайт всегда будут фрагментированы, ибо максимально возможный размер экстента равен 8 Мбайт. Практика показывает, что в среднем на диске имеется не более 2 процентов файлов, имеющих три и более экстентов. Даже общее количество фрагментированных файлов, как правило, не превышает 3 %. Такая ничтожная фрагментация оказывает пренебрежимо малое влияние на общую производительность системы. Кратко рассмотрим вопрос надежности хранения данных в HPFS. Любая файловая система должна обладать средствами исправления ошибок, возникающих при записи информации на диск. Система HPFS для этого использует ***механизм аварийного замещения*** (hotfix) [2].

Если файловая система HPFS сталкивается с проблемой в процессе записи данных на диск, то на экран выводится соответствующее сообщение об ошибке. Затем HPFS сохраняет информацию, которая должна была быть записана в дефектный сектор, в одном из запасных секторов, заранее зарезервированных на этот случай. Список свободных запасных блоков хранится в резервном блоке HPFS. При обнаружении ошибки во время записи данных в нормальный блок HPFS выбирает один из свободных запасных блоков и сохраняет эти данные в нем. Затем файловая система обновляет карту аварийного замещения в резервном блоке. Эта карта представляет собой просто пары двойных слов, каждое из которых является 32-битным номером сектора. Первый номер указывает на дефектный сектор, а второй – на тот сектор среди имеющихся запасных секторов, который был выбран для его замены. После замены дефектного сектора запасным карта аварийного замещения записывается на диск, и на экране появляется всплывающее окно, информирующее пользователя о произошедшей ошибке записи на диск. Каждый раз, когда система выполняет запись или чтение сектора диска, она просматривает карту аварийного замещения и подменяет все номера де-

фектных секторов номерами запасных секторов с соответствующими данными. Следует заметить, что это преобразование номеров существенно не влияет на производительность системы, так как оно выполняется только при физическом обращении к диску, но не при чтении данных из дискового кэша. Очистка карты аварийного замещения автоматически выполняется программой CHKDSK при проверке диска HPFS. Для каждого замещенного блока (сектора) эта программа выделяет новый сектор для файла, которому принадлежат данные, в наиболее подходящем месте жесткого диска. Затем программа перемещает данные из запасного блока в этот сектор и обновляет информацию о положении файла, что может потребовать новой балансировки дерева блоков размещения. После этого CHKDSK вносит поврежденный сектор в список дефектных блоков, который хранится в дополнительном блоке HPFS, и возвращает освобожденный сектор в список свободных запасных секторов резервного блока. Затем удаляет запись из карты аварийного замещения и записывает отредактированную карту на диск.

Все основные файловые объекты в HPFS, в том числе файловые узлы, блоки размещения и блоки каталогов, имеют уникальные 32-битные идентификаторы и указатели на свои родительские и дочерние блоки. Файловые узлы, кроме того, содержат сокращенное имя своего файла или каталога. Избыточность и взаимосвязь файловых структур HPFS позволяют программе CHKDSK полностью восстанавливать файловую структуру диска, последовательно анализируя все файловые узлы, блоки размещения и блоки каталогов. Руководствуясь собранной информацией, CHKDSK реконструирует файлы и каталоги, а затем заново создает битовые карты свободных секторов диска. Запуск программы CHKDSK следует осуществлять с соответствующими ключами. Так, например, один из вариантов работы этой программы позволяет найти и восстановить удаленные файлы. HPFS относится к так называемым монтируемым файловым системам. Это означает, что она не встроена в операционную систему, а добавляется к ней при необходимости.

4.3.5 Файловая система ОС UNIX

Файловая система UNIX имеет иерархическую структуру каталогов и файлов, включая корневой каталог. Файловая система располагается на устройстве, которое обычно является магнитным диском того или иного типа. Если диск достаточно велик, он может быть разбит на несколько логических дисков; тогда на каждом логическом диске может быть размещена отдельная файловая система. Каждая файловая система, прежде чем стать доступной, должна быть смонтирована.

Рассмотрим одну из ранних реализаций файловой системы UNIX, основные идеи которой сохранились до сих пор. Каждая файловая система имеет четыре основные части:

1) загрузочный блок – это первый блок диска (блок 0), зарезервированный для системной загрузочной программы;

2) суперблок – это первый блок собственно файловой системы (блок 1), содержащий основные данные о файловой системе и ее размещении на диске, в том числе о списках свободных i-узлов и блоков;

3) i-узлы – это последовательность блоков, следующих за суперблоком. I-узел содержит ссылки на блоки. Имеется ровно один i-узел для каждого каталога или файла в файловой системе;

4) блоки – это блоки, которые занимают оставшееся пространство диска и содержат либо действительные данные каталогов и файлов (блоки данных), либо ссылки на блоки (косвенные блоки).

Суперблок содержит следующие данные:

- размер дискового пространства, доступного файловой системе (в блоках);
- число блоков, зарегистрированных для i-узлов;
- имя файловой системы;
- имя тома;
- время последнего изменения;
- время последнего копирования (back up);
- ссылку на список свободных блоков;
- ссылку на список свободных i-узлов.

Структура файловой системы UNIX представлена на рис. 4.3.

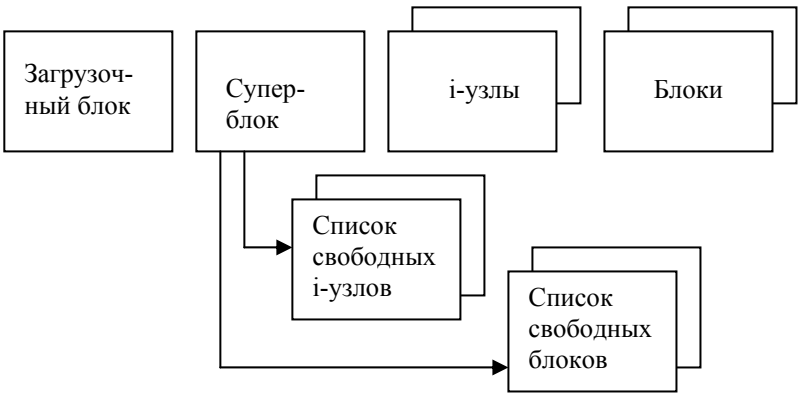

Рис. 4.3 – Структура файловой системы UNIX

Каждый файл и каждый каталог в файловой системе представлен i-узлом, содержащим указатели на блоки, составляющие файл.

В i-узле содержится также информация о правах доступа к файлу, число ссылок на файл из каталогов и другие данные.

Структура i-узлов и блоков файла для UNIX показана на рис. 4.4. Каждый i-узел содержит 13 указателей.

Первые 10 указателей непосредственно ссылаются на блоки данных файла. Поскольку блок содержит 512 байтов, то этого достаточно для обработки файлов до 5120 байтов (512×10).

Если длина файла больше 5120 байт, используется 11-й указатель i-узла, который ссылается на косвенный блок из 128 ссылок на блоки данных. Использование косвенного блока позволяет увеличить длину файла до величины 70656 байт ($512 \times (10 + 128)$).

Если и этого недостаточно, то используется 12-й указатель i-узла, ссылающийся на дважды косвенный блок, содержащий 128 ссылок на косвенные блоки. Тогда максимальный размер файла увеличивается до величины 8459264 байта ($512 \times (10+128^2)$). Наконец, использование последнего 13-го указателя на трижды косвенный блок из 128 ссылок на дважды косвенные блоки дает предельную длину в файловой системе – 1082201088 байтов ($512 \times (10+128+128^2+128^3)$). Другие версии системы UNIX могут отличаться количеством ссылок в i-узле, косвенных блоках и размером блока данных.

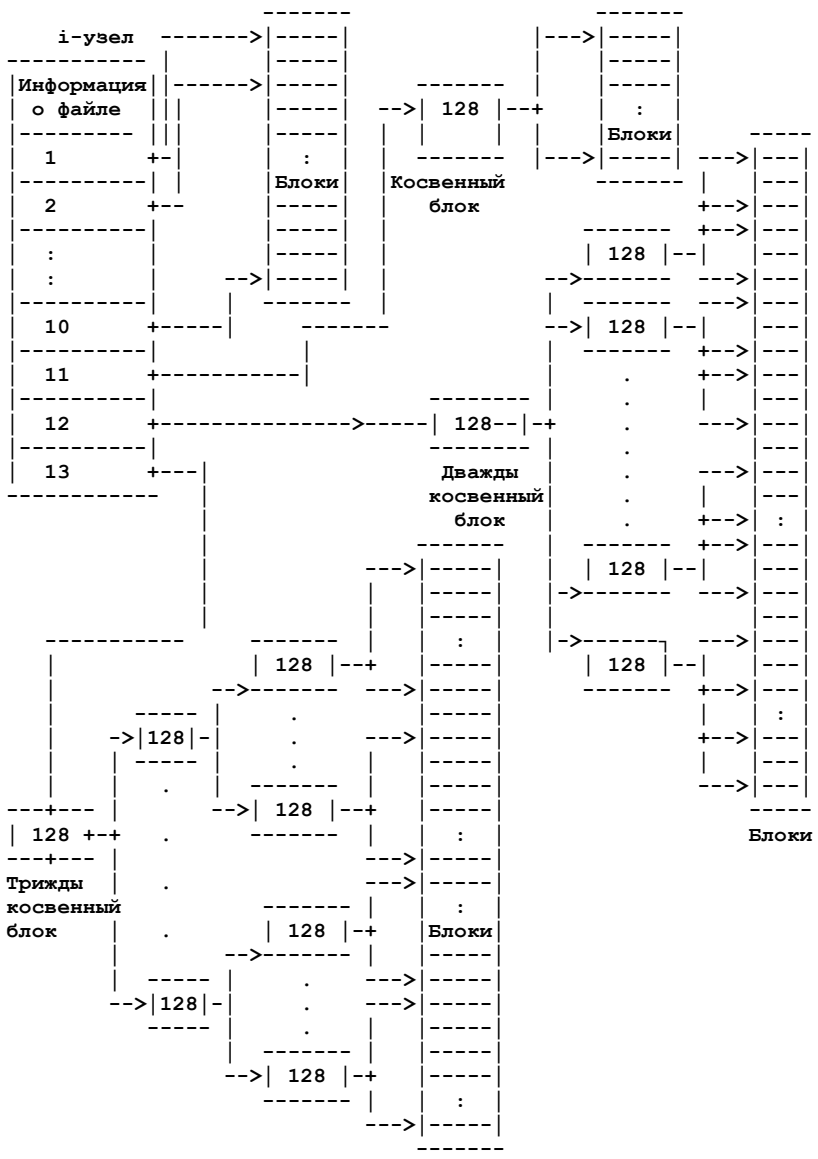

Рис. 4.4 – Структура i-узлов и блоков файла для UNIX

Когда система загружается, имеется только одна из файловых систем, называемая корневой. В ней находятся все важнейшие каталоги (/dev,/etc, /bin и т.п.). Все остальные файловые системы должны быть созданы и смонтированы.

Команда mkfs создает новую файловую систему. Она расположена в каталоге /etc и имеет два параметра:

```
/etc/mkfs <имя> <размер>
```

Первый параметр является именем специального файла и указывает устройство, на котором создается файловая система. Второй параметр – размер пространства файловой системы в блоках – используется для определения по некоторым правилам числа блоков после того, как размещены i-узлы.

Пример создания файловой системы на флоппи-диске:

```
/etc/mkfs /dev/flo 2000
isize = 230
```

Ответное сообщение указывает число блоков, выделенных для размещения i-узлов.

Монтирование файловой системы UNIX для какой-либо ОС осуществляется посредством команды mount. Эта команда подключает корневой каталог монтируемой файловой системы в один из каталогов корневой файловой системы. Команда расположена в каталоге /etc и имеет два параметра:

```
/etc/mount <устройство> <каталог>
```

Первый параметр является именем спецфайла для монтируемого логического устройства, содержащего подключаемую файловую систему. Второй – имя уже существующего каталога, под которым монтируется файловая система.

Чтобы выяснить, какие файловые системы смонтированы в данный момент, надо выполнить команду mount без параметров:

```
mount
/dev/fl0 on /floppy0
```

Ответом является сообщение об этих системах (в данном случае об одной системе). Оно формируется на основе данных о монтаже файловых систем, хранимых в файле /etc/mnttab.

Права доступа корневого каталога монтируемой файловой системы и каталога, под которым производится монтаж, должны быть одинаковыми во избежание ошибок операционной системы.

Если файловая система на съемном устройстве больше не используется, ее можно демонтировать командой umount, расположенной в каталоге /etc и имеющей один параметр:

```
umount <устройство>
```

Например, демонтаж файловой системы на гибком диске из предыдущего примера выполняется командой:

```
umount /dev/fl0
```

Результатом демонтажа является разрыв связи между корневым каталогом демонтируемой файловой системы и каталогом корневой файловой системы, в котором производился монтаж. При выполнении команды демонтажа текущий каталог должен быть вне демонтируемой файловой системы, иначе будет выдано сообщение о том, что устройство занято (`umount : device busy`), и команда не будет выполнена.

Структура файловой системы, описанная выше в терминах i-узлов, блоков, косвенных блоков и суперблока, может быть нарушена. В этом случае возникает необходимость в ее восстановлении. При разрушении информации в трижды косвенном блоке могут появиться следующие проблемы:

- некоторый блок может оказаться вне системы, т.е. перестать быть частью файла, и отсутствовать в списке свободных блоков;
- могут появиться дубли i-узлов, описывающие один и тот же файл дважды;
- возникновение ситуации, когда некоторый блок является частью файла и одновременно присутствует в списке свободных блоков;
- существование некоторых файлов, которые не включены ни в один каталог.

Эти проблемы могут быть ликвидированы, поскольку структура файловой системы обладает некоторой избыточностью (наличием дополнительной информации), позволяющей восстанавливать отдельные поломки. Вот некоторые виды избыточности:

- блок данных, являющийся каталогом, содержит имена файлов и номера i-узлов; где-то имеется i-узел, соответствую-

щий этому каталогу, и этот i-узел должен быть каталогом, а не обычным файлом;

- блок, включенный в список свободных блоков, теоретически не может быть частью какого-либо файла; для проверки этого достаточно сканировать все i-узлы для просмотра всех блоков, занятых файлами, и сканировать список свободных блоков;
- аналогично, блок, принадлежащий файлу, должен принадлежать только одному файлу; это легко проверить.

Эти и другие виды избыточности использует программа проверки файловой системы, запускаемая командой fsck (file system check). В различных реализациях существуют разные команды проверки целостности файловой системы: icheck, dcheck, ncheck. Однако все они в большей или меньшей степени перекрываются командой fsck.

Команда fsck выполняется в несколько фаз, на которых производится следующая работа:

- проверка целостности i-узлов (счетчик связи, тип и формат i-узла);
- проверка каталогов, указывающих на i-узлы, содержащие ошибки;
- проверка каталогов, на которые нет ссылок;
- проверка счетчиков связей в каталогах и файлах;
- проверка неверных блоков и дублированных блоков в списке свободных блоков; неиспользуемых блоков, которые должны быть включены, но не являющихся таковыми, в список свободных блоков; счетчика общего числа свободных блоков.

Команда по умолчанию всегда проверяет корневую файловую систему: все другие файловые системы проверяются, если их имена занесены в файл /etc/checklist.

После выполнения fsck, связанного с «починкой» файловой системы, может появиться сообщение

```
***** BOOT UNIX (NO SYNC!) *****,
```

требующее перезагрузки системы без выполнения команды sync.

Если этого не сделать, работа по восстановлению списка свободных блоков будет утрачена, так как копии управляющих таблиц и буфера в оперативной памяти остались старыми. Для

их обновления требуется перезагрузка без выгрузки буферов на диск командой sync.

Необходимым условием правильной работы fsck является также наличие пустого каталога /lost+found в корневом каталоге.

Если при выполнении fsck будут найдены каталоги, на которые никто не ссылается в проверяемой файловой системе, они будут подключены в каталог /lost+found для дальнейшего изучения их принадлежности.

4.3.6 Файловые системы для CD-ROM

4.3.6.1 Файловая система CDFS

В Windows 2000 обеспечивается поддержка файловой системы CDFS (Compact Disk File System), отвечающей стандарту ISO'9660, описывающему расположение информации на CD-ROM. Поддерживаются длинные имена файлов в соответствии с ISO'9660 Level 2 [15].

При создании CD-ROM под управлением Windows 2000 следует иметь в виду следующее:

- все имена каталогов и файлов должны содержать менее 32 символов;
- все имена каталогов и файлов должны состоять только из символов верхнего регистра;
- глубина каталогов не должна превышать 8 уровней от корня;
- использование расширений имен файлов не обязательно.

4.3.6.2 Файловая система UDF

Поддержка файловой системы UDF (Universal Disk Format) является одним из новшеств в Windows 2000. UDF – это файловая система, отвечающая стандарту ISO'13346 и используемая для обмена данными с накопителями CD-ROM и DVD. В настоящее время поддерживаются только диски версий UDF 1.02 и 1.50 [15].

Вопросы для самопроверки

1. На какие классы делятся устройства внешней памяти по методу доступа?
2. Назовите общие параметры устройств внешней памяти.
3. Назовите общие параметры жестких дисков.
4. Опишите структуру магнитного диска (разбиение дисков на разделы). Сколько разделов может быть на магнитном диске? Каково назначение разделов магнитного диска?
5. В какой последовательности ОС дает имена разделам на жестком диске, если в ПК подключают два жестких диска с различными типами разделов?
6. Как в общем случае осуществляется загрузка ОС после включения компьютера?
7. Приведите основные характеристики FAT16? Что такое кластер? От каких параметров зависит его размер?
8. Приведите основные характеристики FAT32.
9. Приведите основные характеристики NTFS.
10. Приведите основные характеристики HPFS. За счет чего в файловой системе HPFS обеспечена высокая производительность?
11. Приведите основные характеристики файловой системы ОС UNIX?
12. Где используются файловые системы CDFS и UDF?

5. АРХИТЕКТУРЫ ОПЕРАЦИОННЫХ СИСТЕМ И ИНТЕРФЕЙСЫ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

5.1 Основные принципы построения операционных систем

Существует множество принципов построения операционных систем. Рассмотрим основные из них [2].

5.1.1.1 Принцип модульности

Под **модулем** в общем случае понимают функционально законченный элемент системы, выполненный в соответствии с принятыми межмодульными интерфейсами. По своему определению модуль предполагает возможность легкой замены его на другой при наличии заданных интерфейсов. Способы обособления составных частей ОС в отдельные модули могут существенно различаться, но чаще всего разделение происходит именно по функциональному признаку. В значительной степени разделение системы на модули определяется используемым методом проектирования ОС (снизу вверх или наоборот).

Особо важное значение при построении ОС имеют ***привилегированные, повторно входимые и реентерабельные***[7] ***модули***, так как они позволяют более эффективно использовать ресурсы вычислительной системы. В некоторых системах реентерабельность программы получают автоматически, благодаря неизменяемости кодовых частей программ при исполнении вследствие особенностей системы команд машины, а также автоматическому распределению регистров, автоматическому отделению кодовых частей программ от данных и помещению последних в системную область памяти. Естественно, что для этого необходима соответствующая аппаратная поддержка или специальные системные модули.

---

[7] Реентерабельность – (буквально, повторновходимость; специальный термин для обозначения работоспособности программы) – свойство программы корректно выполняться при рекурсивном (возвращаемом) вызове из прерывания.

В основе технологических и эксплуатационных свойств системы заложен принцип модульности, наибольший эффект от использования которого достижим в случае одновременного распространения данного принципа на операционную систему, прикладные программы и аппаратуру.

5.1.1.2 Принцип функциональной избирательности

В ОС выделяется некоторая часть важных модулей, которые должны постоянно находиться в оперативной памяти для более эффективной организации вычислительного процесса. Эту часть в ОС называют ядром, так как это – основа системы. При формировании состава ядра требуется учитывать два противоречивых требования. С одной стороны, в состав ядра должны войти наиболее часто используемые системные модули, с другой – количество модулей должно быть таковым, чтобы объем памяти, занимаемый ядром, не был слишком большим. В состав ядра, как правило, входят модули по управлению системой прерываний, средства по переводу программ из состояния счета в состояние ожидания, готовности и обратно, средства по распределению таких основных ресурсов, как оперативная память и процессор. Помимо программных модулей, входящих в состав ядра и постоянно располагающихся в оперативной памяти, может быть много других системных программных модулей, которые получают название ***транзитных***. Транзитные программные модули загружаются в оперативную память только при необходимости и в случае отсутствия свободного пространства могут быть замещены другими транзитными модулями. В качестве синонима к термину «транзитный» можно использовать термин «диск-резидентный».

5.1.1.3 Принцип генерируемости ОС

Суть принципа состоит в организации (выборе) такого способа исходного представления центральной системной управляющей программы ОС (ядра и постоянно находящихся в оперативной памяти основных компонентов), который позволял бы

настраивать эту системную супервизорную часть исходя из конкретной конфигурации конкретного вычислительного комплекса и круга решаемых задач. Эта процедура проводится редко перед достаточно протяженным периодом эксплуатации ОС. Процесс генерации осуществляется с помощью специальной программы-генератора и соответствующего входного языка для этой программы, позволяющего описывать программные возможности системы и конфигурацию машины. В результате генерации получается полная версия ОС. Сгенерированная версия ОС представляет собой совокупность системных наборов модулей и данных.

Упомянутый раньше принцип модульности положительно проявляется при генерации ОС. Он существенно упрощает настройку ОС на требуемую конфигурацию вычислительной системы. В настоящее время при использовании персональных компьютеров с принципом генерируемости ОС можно столкнуться разве что только при работе с Linux. В этой UNIX-системе имеется возможность не только использовать какое-либо готовое ядро ОС, но и самому сгенерировать (скомпилировать) такое ядро, которое будет оптимальным для данного конкретного персонального компьютера и решаемых на нем задач. Кроме генерации ядра в Linux имеется возможность указать и набор подгружаемых драйверов и служб, то есть часть функций может реализовываться модулями, непосредственно входящими в ядро системы, а часть – модулями, имеющими статус подгружаемых, транзитных.

В остальных современных распространенных ОС для персональных компьютеров конфигурирование ОС под соответствующий состав оборудования осуществляется на этапе инсталляции, а потом состав драйверов и изменение некоторых параметров ОС может быть осуществлено посредством редактирования конфигурационного файла.

5.1.1.4 Принцип функциональной избыточности

Этот принцип учитывает возможность проведения одной и той же работы различными средствами. В состав ОС может входить несколько типов мониторов (модулей супервизора, управ-

ляющих тем или другим видом ресурса), различные средства организации коммуникаций между вычислительными процессами. Наличие нескольких типов мониторов, нескольких систем управления файлами позволяет пользователям быстро и наиболее адекватно адаптировать ОС к определенной конфигурации вычислительной системы, обеспечивать максимально эффективную загрузку технических средств при решении конкретного класса задач, получать максимальную производительность при решении заданного класса задач.

5.1.1.5 Принцип виртуализации

Построение виртуальных ресурсов, их распределение и использование в настоящее время применяется практически в любой ОС. Этот принцип позволяет представить структуру системы в виде определенного набора планировщиков процессов и распределителей ресурсов (мониторов) и использовать единую централизованную схему распределения ресурсов.

Наиболее естественным и законченным проявлением концепции виртуальности является понятие **виртуальной машины**. По сути, любая операционная система, являясь средством распределения ресурсов и организуя по определенным правилам управление процессами, скрывает от пользователя и его приложений реальные аппаратные и иные ресурсы, заменяя их некоторой абстракцией. В результате пользователи видят и используют виртуальную машину как некое устройство, способное воспринимать их программы, написанные на определенном языке программирования, выполнять их и выдавать результаты. При таком языковом представлении пользователя совершенно не интересует реальная конфигурация вычислительной системы, способы эффективного использования ее компонентов и подсистем. Он мыслит и работает с машиной в терминах используемого им языка и тех ресурсов, которые ему предоставляются в рамках виртуальной машины.

Виртуальная машина, предоставляемая пользователю, воспроизводит архитектуру реальной машины, но архитектурные элементы в таком представлении выступают с новыми или улучшенными характеристиками, как правило, упрощающими

работу с системой. Характеристики могут быть произвольными, но чаще всего пользователи желают иметь собственную «идеальную» по архитектурным характеристикам машину в следующем составе:

- единообразная по логике работы виртуальная память практически неограниченного объема. Среднее время доступа соизмеримо со значением этого параметра для оперативной памяти. Организация работы с информацией в виртуальной памяти производится в терминах обработки данных – в терминах работы с сегментами данных на уровне выбранного пользователем языка программирования;
- произвольное количество виртуальных процессоров, способных работать параллельно и взаимодействовать во время работы. Способы управления процессорами, в том числе синхронизация и информационные взаимодействия, реализованы и доступны пользователям на уровне используемого языка в терминах управления процессами;
- произвольное количество внешних виртуальных устройств, способных работать с памятью виртуальной машины параллельно или последовательно, асинхронно или синхронно по отношению к работе того или иного виртуального процессора, инициирующего работу этих устройств. Информация, передаваемая или хранимая на виртуальных устройствах, не ограничена допустимыми размерами. Доступ к такой информации осуществляется на основе либо последовательного, либо прямого способа доступа в терминах соответствующей системы управления файлами. Предусмотрено расширение информационных структур данных, хранимых на виртуальных устройствах.

Степень приближения к «идеальной» виртуальной машине может быть большей или меньшей в каждом конкретном случае. Чем больше виртуальная машина, реализуемая средствами ОС на базе конкретной аппаратуры, приближена к «идеальной» по характеристикам машине и, следовательно, чем больше ее архитектурно-логические характеристики отличны от реально существующих, тем больше степень виртуальности у полученной пользователем машины.

Одним из аспектов виртуализации является организация возможности выполнения в данной ОС приложений, которые

разрабатывались для других ОС. Другими словами, речь идет об организации нескольких операционных сред. Реализация этого принципа позволяет данной ОС иметь очень сильное преимущество перед аналогичными ОС, не имеющими такой возможности. Примером реализации принципа виртуализации может служить ***VDM-машина*** (Virtual DOS Machine) – защищенная подсистема, предоставляющая полную среду MS DOS и консоль для выполнения MS DOS-приложений. Одновременно может выполняться практически произвольное число VDM-сессий. Такие VDM-машины имеются и в системах Microsoft Windows, и в OS/2.

5.1.1.6 Принцип независимости программ от внешних устройств

Этот принцип реализуется сейчас в подавляющем большинстве ОС общего применения. Мы уже говорили о нем, рассматривая принципы организации ввода/вывода. Пожалуй, впервые наиболее последовательно данный принцип был реализован в ОС UNIX. Реализован он и в большинстве современных ОС для ПК. Напомним, этот принцип заключается в том, что связь программ с конкретными устройствами производится не на уровне трансляции программы, а в период планирования ее исполнения. В результате перекомпиляция при работе программы с новым устройством, на котором располагаются данные, не требуется.

Этот принцип позволяет осуществлять операции управления внешними устройствами независимо от их конкретных физических характеристик. Например, программе, содержащей операции обработки последовательного набора данных, безразлично, на каком носителе эти данные будут располагаться. Смена носителя и данных, размещаемых на них при неизменности структурных характеристик данных, не принесет каких-либо изменений в программу, если в системе реализован принцип независимости.

5.1.1.7 Принцип совместимости

Одним из аспектов совместимости является способность ОС выполнять программы, написанные для других ОС или для более ранних версий данной операционной системы, а также для другой аппаратной платформы. Необходимо разделять вопросы двоичной совместимости и совместимости на уровне исходных текстов приложений. Двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее на выполнение на другой ОС. Для этого необходимы совместимость на уровне команд процессора, и совместимость на уровне системных вызовов, и даже на уровне библиотечных вызовов, если они являются динамически связываемыми. Совместимость на уровне исходных текстов требует наличия соответствующего транслятора в составе системного программного обеспечения, а также совместимости на уровне библиотек и системных вызовов. При этом необходима перекомпиляция имеющихся исходных текстов в новый выполняемый модуль.

Гораздо сложнее достичь двоичной совместимости между процессорами, основанными на разных архитектурах. Для того чтобы один компьютер выполнял программы другого (например, программу для ПК типа IBM PC желательно выполнить на ПК типа Macintosh фирмы Apple), этот компьютер должен работать с машинными командами, которые ему изначально непонятны. В таком случае процессор типа 680x0 (или PowerPC) должен исполнять двоичный код, предназначенный для процессора i80x86. Процессор 80x86 имеет свои собственные дешифратор команд, регистры и внутреннюю архитектуру. Процессор 680x0 не понимает двоичный код 80x86, поэтому он должен выбрать каждую команду, декодировать ее, чтобы определить, для чего она предназначена, а затем выполнить эквивалентную подпрограмму, написанную для 680x0. Так как к тому же у 680x0 нет в точности таких же регистров, флагов и внутреннего арифметико-логического устройства, как в 80x86, он должен имитировать все эти элементы с использованием своих регистров или памяти. Также он должен тщательно воспроизводить результаты каждой команды, а это требует специально написанных подпрограмм для 680x0, гарантирующих, что состояние эмулируе-

мых регистров и флагов после выполнения каждой команды будет в точности таким же, как и на реальном 80x86. Выходом в таких случаях является использование так называемых прикладных сред или эмуляторов. Учитывая, что основную часть программы, как правило, составляют вызовы библиотечных функций, прикладная среда имитирует библиотечные функции целиком, используя заранее написанную библиотеку функций аналогичного назначения, а остальные команды эмулирует каждую по отдельности.

Одним из средств обеспечения совместимости программных и пользовательских интерфейсов является соответствие стандартам POSIX, использование которого позволяет создавать программы в стиле UNIX, легко переносимых впоследствии из одной системы в другую.

5.1.1.8 Принцип открытости и наращиваемости

Открытая операционная система доступна для анализа как пользователям, так и системным специалистам, обслуживающим вычислительную систему. Наращиваемая (модифицируемая, развиваемая) ОС позволяет не только использовать возможности генерации, но и вводить в ее состав новые модули, совершенствовать существующие и т.д. Другими словами, следует обеспечить возможность легкого внесения дополнений и изменений в необходимых случаях без нарушения целостности системы. Прекрасные возможности для расширения предоставляет подход к структурированию ОС по типу клиент-сервер с использованием микроядерной технологии. В соответствии с этим подходом ОС строится как совокупность привилегированной управляющей программы и набора непривилегированных услуг (сервееров). Основная часть ОС остается неизменной, и в то же время могут быть добавлены новые серверы или улучшены старые. Этот принцип иногда трактуют как **расширяемость системы**. К открытым ОС, прежде всего, следует отнести UNIX-системы и, естественно, ОС Linux.

5.1.1.9 Принцип мобильности (переносимости)

Операционная система относительно легко должна переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа, которая включает наряду с типом процессора и способ организации всей аппаратуры компьютера (архитектуру вычислительной системы), на аппаратную платформу другого типа. Заметим, что принцип переносимости очень близок принципу совместимости, хотя это и не одно и то же. Создание переносимой ОС аналогично написанию любого переносимого кода, при этом нужно следовать некоторым правилам. Во-первых, большая часть ОС должна быть выполнена на языке, имеющемся на всех системах, на которые планируется в дальнейшем ее переносить. Это, прежде всего, означает, что ОС должна быть написана на языке высокого уровня, предпочтительно стандартизованном, например на языке С. Программа, написанная на ассемблере, не является в общем случае переносимой. Во-вторых, важно минимизировать или, если возможно, исключить те части кода, которые непосредственно взаимодействуют с аппаратными средствами. Зависимость от аппаратуры может иметь много форм. Некоторые очевидные формы зависимости включают прямое манипулирование регистрами и другими аппаратными средствами. Наконец, если аппаратно-зависимый код не может быть полностью исключен, то он должен быть изолирован в нескольких хорошо локализуемых модулях. Аппаратно-зависимый код не должен быть распределен по всей системе. Например, можно спрятать аппаратно-зависимую структуру в программно задаваемые данные абстрактного типа. Другие модули системы будут работать с этими данными, а не с аппаратурой, используя набор некоторых функций. Когда ОС переносится, то изменяются только эти данные и функции, которые ими манипулируют.

Введение стандартов POSIX преследовало цель обеспечить переносимость создаваемого программного обеспечения.

5.1.1.10 Принцип обеспечения безопасности вычислений

Обеспечение безопасности при выполнении вычислений является желательным свойством для любой многопользовательской системы. Правила безопасности определяют такие свойства, как защиту ресурсов одного пользователя от других и установление квот по ресурсам для предотвращения захвата одним пользователем всех системных ресурсов, таких, например, как память.

Обеспечение защиты информации от несанкционированного доступа является обязательной функцией сетевых операционных систем. Во многих современных ОС гарантируется степень безопасности данных, соответствующая уровню С2 в системе стандартов США. Основы стандартов в области безопасности были заложены в документе «Критерии оценки надежных компьютерных систем». Этот документ, изданный Национальным центром компьютерной безопасности NCSC (National Computer Security Center) в США в 1983 году, часто называют «Оранжевой книгой».

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

Иерархия уровней безопасности, приведенная в «Оранжевой книге», помечает низший уровень безопасности как D, а высший – как А.

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

Основными свойствами, характерными для систем класса С, являются наличие подсистемы учета событий, связанных с безопасностью, и избирательный контроль доступа. Класс (уровень) С делится на 2 подуровня: уровень С1, обеспечивающий защиту данных от ошибок пользователей, но не от действий

злоумышленников, и более строгий уровень С2. На уровне С2 должны присутствовать:

- средства секретного входа, обеспечивающие идентификацию пользователей путем ввода уникального имени и пароля, перед тем, как им будет разрешен доступ к системе;
- избирательный контроль доступа, позволяющий владельцу ресурса определить, кто имеет доступ к ресурсу и что он может с ним делать. Владелец делает это путем предоставляемых прав доступа пользователю или группе пользователей;
- средства учета и наблюдения (auditing), обеспечивающие возможность обнаружения и фиксирования важных событий, связанных с безопасностью, или любых попыток создать, получить доступ или удалить системные ресурсы;
- защита памяти, заключающаяся в том, что память инициализируется перед повторным использованием.

Системы уровня В основаны на принципах пометки данных и распределения пользователей по категориям, реализующим мандатный контроль доступа. Каждому пользователю присваивается рейтинг защиты, и он может получать доступ к данным только в соответствии с этим рейтингом. Этот уровень в отличие от уровня С защищает систему от ошибочного поведения пользователя. Уровень А является самым высоким уровнем безопасности, он требует в дополнение ко всем требованиям уровня В выполнения формального, математически обоснованного доказательства соответствия системы требованиям безопасности. Различные коммерческие структуры (например, банки) особо выделяют необходимость учетной службы, аналогичной той, что предлагают государственные рекомендации С2. Любая деятельность, связанная с безопасностью, может быть отслежена и тем самым учтена. Это как раз то, чего требует стандарт для систем класса С2 и что обычно нужно банкам. Однако коммерческие пользователи, как правило, не хотят расплачиваться производительностью за повышенный уровень безопасности. А-уровень безопасности занимает своими управляющими механизмами до 90 % процессорного времени, что, безусловно, в большинстве случаев уже неприемлемо. Более безопасные системы не только снижают эффективность, но и существенно ограничивают число доступных прикладных пакетов, которые соответствующим образом

могут выполняться в подобной системе. Например, для ОС Solaris (версия UNIX) есть несколько тысяч приложений, а для ее аналога B-уровня – только около ста.

5.2 Микроядерные операционные системы

Основой модульных и переносимых расширений является **микроядро** – минимальная стержневая часть операционной системы [2]. Существует мнение, что большинство операционных систем следующих поколений будут обладать микроядрами. Имеется масса разных мнений по поводу организации служб операционной системы по отношению к микроядру: как проектировать драйверы устройств, чтобы добиться наибольшей эффективности, но сохранить функции драйверов максимально независимыми от аппаратуры; следует ли выполнять операции, не относящиеся к ядру, в пространстве ядра или в пространстве пользователя; стоит ли сохранять программы имеющихся подсистем (например, UNIX) или лучше отбросить все и начать с нуля. Основная идея, заложенная в технологию микроядра, будь то собственно ОС или ее графический интерфейс, заключается в том, чтобы конструировать необходимую среду верхнего уровня, из которой можно легко получить доступ ко всем функциональным возможностям уровня аппаратного обеспечения. При такой структуре ядро служит стартовой точкой для создания системы. Искусство разработки микроядра заключается в выборе базовых примитивов, которые должны в нем находиться для обеспечения необходимого и достаточного сервиса. В микроядре содержится и исполняется минимальное количество кода, необходимое для реализации основных системных вызовов. В число этих вызовов входят передача сообщений и организация другого общения между внешними по отношению к микроядру процессами, поддержка управления прерываниями, а также ряд некоторых других функций. Остальные функции, характерные для «обычных» (не микроядерных) ОС, обеспечиваются как модульные дополнения-процессы, взаимодействующие главным образом между собой и осуществляющие взаимодействие посредством передачи сообщений.

Микроядро является маленьким передающим сообщения модулем системного программного обеспечения, работающим в наиболее приоритетном состоянии компьютера и поддерживающим остальную часть операционной системы, рассматриваемую как набор серверных приложений. Интерес к микроядрам возрастал по мере того как системные разработчики реагировали на сложность современных реализаций операционных систем и поддерживался тем, что исследовательское сообщество успешно демонстрировало реализуемость концепции микроядра. Созданная в университете Карнеги Меллон технология микроядра Mach служит основой для многих ОС.

Исполняемые микроядром функции ограничены в целях сокращения его размеров и максимизации количества кода, работающего как прикладная программа.

Микроядро включает только те функции, которые требуются для определения набора абстрактных сред обработки прикладных программ и организации совместной работы приложений при обеспечении сервисов и взаимодействия клиентами и серверами. В результате микроядро обеспечивает только пять различных типов сервисов:

1) управление виртуальной памятью;
2) управление задания и потоки;
3) поддержку межпроцессных коммуникаций IPC (Inter-Process Communication);
4) управление поддержкой ввода/вывода и прерываниями;
5) поддержку сервисов набора хоста[8] и процессора.

Другие подсистемы и функции операционной системы, такие как системы управления файлами, поддержка внешних устройств и традиционные программные интерфейсы, размещаются в одном или более системных сервисах либо в задаче операционной системы. Эти программы работают как приложения микроядра. В соответствии с концепцией множественных потоков на одно задание микроядро создает прикладную среду, обеспечивающую использование режима мультипроцессора без наличия требования о применении какой-либо конкретной машины

[8] Host – главный компьютер; в настоящее время этим термином обозначают любой компьютер, имеющий IP-адрес.

как мультипроцессорной: на однопроцессорной машине различные потоки просто выполняются в разные времена. Вся поддержка, требуемая для мультипроцессорных машин, сконцентрирована в сравнительно малом и простом микроядре.

Благодаря своим размерам и способности поддерживать стандартные сервисы программирования и характеристики в виде прикладных программ микроядра организованы проще, чем ядра монолитных или модульных операционных систем. Посредством использования микроядра функция операционной системы разбивается на модульные части, которые могут быть сконфигурированы целым рядом способов, позволяющих строить большие системы добавлением частей к меньшим. Например, каждый аппаратно-независимый нейтральный сервис логически отделен и может быть сконфигурирован в широком диапазоне способов. Микроядра также облегчают поддержку мультипроцессоров созданием стандартной программной среды, которая может использовать множественные процессоры в случае их наличия, однако не требует их в случае отсутствия таковых. Специализированный код для мультипроцессоров ограничен самим микроядром. Более того, сети из общающихся между собой микроядер могут быть использованы для обеспечения операционной системной поддержки возникающего класса массивно параллельных машин. В некоторых случаях определенной сложностью использования микроядерного подхода на практике является замедление скорости выполнения системных вызовов при передаче сообщений через микроядро по сравнению с классическим подходом. С другой стороны, можно констатировать и обратное. Поскольку микроядра малы и имеют сравнительно мало требуемого к исполнению кода уровня ядра, они обеспечивают удобный способ поддержки характеристик реального времени, требующихся для мультимедиа, управления устройствами и высокоскоростных коммуникаций. Наконец, хорошо структурированные микроядра обеспечивают изолирующий слой для аппаратных различий, которые не маскируются применением языков программирования высокого уровня. Таким образом, они упрощают перенесение кода и увеличивают уровень его повторного использования.

Наиболее ярким представителем микроядерных ОС является ОС реального времени QNX (Queue Nicks – очередь точных моментов времени, или очередь зарубок). Микроядро QNX поддерживает только планирование и диспетчеризацию процессов, взаимодействие процессов, обработку прерываний и сетевые службы нижнего уровня. Микроядро обеспечивает всего лишь около двух десятков системных вызовов, но благодаря этому оно может быть целиком размещено во внутреннем кэше даже таких процессоров, как Intel 486. Как известно, разные версии этой ОС имели и различные объемы ядер – от 8 до 46 Кбайт.

Чтобы построить минимальную систему QNX, требуется добавить к микроядру менеджер процессов, который создает процессы, управляет процессами и памятью процессов. Чтобы ОС QNX была применима не только во встроенных и бездисковых системах, нужно добавить файловую систему и менеджер устройств. Эти менеджеры исполняются вне пространства ядра, так что ядро остается небольшим.

5.3 Монолитные операционные системы

Монолитные ОС являются прямой противоположностью микроядерным ОС. В монолитной ОС, несмотря на ее возможную сильную структуризацию, достаточно трудно удалить какой-либо уровень многоуровневой модульной структуры. Добавление новых функций и изменение существующих для монолитных ОС требует глубокого знания всей архитектуры ОС и чрезвычайно больших усилий. Поэтому более современный подход к проектированию ОС, который может быть условно назван «клиент-серверной технологией», позволяет в большем количестве и с меньшими трудозатратами реализовать перечисленные выше принципы проектирования ОС [2].

Модель клиент-сервер предполагает наличие программного компонента, являющегося потребителем какого-либо сервиса, – **клиента,** и программного продукта, служащего поставщиком этого сервиса, – **сервера**. Взаимодействие между клиентом и сервером стандартизируется, так что сервер может обслуживать клиентов, реализованных разными способами и, может быть, разными разработчиками. При этом главным требованием явля-

ется использование единообразного интерфейса. Инициатором обмена обычно является клиент, который посылает запрос на обслуживание серверу, находящемуся в состоянии ожидания запроса. Один и тот же программный компонент может быть клиентом по отношению к одному виду услуг и сервером для другого вида услуг. Модель клиент-сервер является скорее удобным концептуальным средством ясного представления функций того или иного программного элемента в какой-либо ситуации, нежели технологией. Эта модель успешно применяется не только при построении операционных систем, но и на всех уровнях программного обеспечения и имеет в некоторых случаях более узкий, специфический смысл, сохраняя, естественно, при этом все свои общие черты.

При поддержке монолитных ОС возникает ряд проблем, связанных с тем, что все функции макроядра работают в едином адресном пространстве. Во-первых, это опасность возникновения конфликта между различными частями ядра; во-вторых – сложность подключения к ядру новых драйверов. Преимущество микроядерной архитектуры перед монолитной заключается в том, что каждый компонент системы представляет собой самостоятельный процесс, запуск или остановка которого не отражается на работоспособности остальных процессов. Микроядерные ОС в настоящее время разрабатываются чаще монолитных. Однако следует заметить, что использование технологии клиент-сервер – это еще не гарантия того, что ОС станет микроядерной. В качестве подтверждения можно привести пример с ОС Windows NT, которая построена на идеологии клиент-сервер, но которую тем не менее трудно назвать микроядерной.

5.4 Принципы построения интерфейсов операционных систем

ОС всегда выступает как интерфейс между аппаратурой компьютера и пользователем с его задачами. Под интерфейсами операционных систем здесь и далее следует понимать специальные интерфейсы системного и прикладного программирования, предназначенные для выполнения следующих задач:

- управления процессами, которое включает в себя следующий набор основных функций:
  - запуск, приостанов и снятие задачи с выполнения;
  - назначение или изменение приоритета задачи;
  - организация взаимодействия задач между собой (механизмы сигналов, семафорные примитивы, очереди, конвейеры, почтовые ящики);
  - организации удаленного вызова подпрограмм RPC (Remote Procedure Call);
- управления памятью:
  - запрос на выделение блока памяти;
  - освобождение памяти;
  - изменение параметров блока памяти (например, память может быть заблокирована процессом либо предоставлена в общий доступ);
  - отображение файлов на память (имеется не во всех системах);
- управления вводом/выводом:
  - запрос на управление виртуальными устройствами. Напомним, что управление вводом/выводом является привилегированной функцией самой ОС, и никакая из пользовательских задач не должна иметь возможности непосредственно управлять устройствами;
  - файловые операции (запросы к системе управления файлами на создание, изменение и удаление данных, организованных в файлы).

Здесь мы перечислили основные наборы функций, которые выполняются ОС по соответствующим запросам от задач. Что касается пользовательского интерфейса операционной системы, то он реализуется с помощью специальных программных модулей, которые принимают его команды на соответствующем языке (возможно, с использованием графического интерфейса) и транслируют их в обычные вызовы в соответствии с основным интерфейсом системы. Обычно эти модули называют интерпретатором команд. Так, например, функции такого интерпретатора в MS DOS выполняет модуль COMMAND.COM. Получив от пользователя команду, такой модуль после лексического и синтаксического анализа либо сам выполняет действие, либо, что

случается чаще, обращается к другим модулям ОС, используя механизм API. Надо заметить, что в последние годы большую популярность получили графические интерфейсы GUI (Graphical User Interface), в которых задействованы соответствующие манипуляторы типа «мышь» или «трекбол». Указание курсором на объекты и щелчок (клик) или двойной щелчок по соответствующим клавишам приводит к каким-либо действиям – запуску программы, ассоциированной с указываемым объектом, выбору и/или активизации пунктов меню и т.д. Можно сказать, что такая интерфейсная подсистема транслирует команды пользователя в обращения к ОС.

Поясним также, что управление GUI – частный случай задачи управления вводом/выводом, не являющийся частью ядра операционной системы, хотя в ряде случаев разработчики ОС относят функции GUI к основному системному API. Следует отметить, что имеются два основных подхода к управлению задачами. Так, в одних системах порождаемая задача наследует все ресурсы задачи-родителя, тогда как в других системах существуют равноправные отношения, и при порождении нового процесса ресурсы для него запрашиваются у операционной системы.

Обращения к операционной системе в соответствии с имеющимся API может осуществляться как посредством вызова подпрограммы с передачей ей необходимых параметров, так и через механизм программных прерываний. Выбор метода реализации вызовов функций API должен определяться архитектурой платформы.

Так, например, в операционной системе MS DOS, которая разрабатывалась для однозадачного режима (поскольку процессор i8086 не поддерживал мультипрограммирование), использовался механизм программных прерываний. При этом основной набор функций API был доступен через точку входа обработчика int 21h. В более сложных системах имеется не одна точка входа, а множество – по количеству функций API. Так, в большинстве операционных систем используется метод вызова подпрограмм. В этом случае вызов (например, это функция **биб-**

лиотеки времени выполнения RTL[9] (Run Time Lubrary) сначала передается в модуль API, который и перенаправляет вызов соответствующим обработчикам программных прерываний, входящим в состав операционной системы. Использование механизма прерываний вызвано, главным образом, тем, что при этом процессор переводится в режим супервизора.

5.4.1 Интерфейс прикладного программирования

5.4.1.1 Общее представление об интерфейсе прикладного программирования

Прежде всего, необходимо однозначно разделить общий термин интерфейс прикладного программирования API (Application Program Interface) на следующие направления:

- API как интерфейс высокого уровня, принадлежащий к библиотекам RTL;
- API прикладных и системных программ, входящих в поставку операционной системы;
- прочие API.

Интерфейс прикладного программирования, как это и следует из названия, предназначен для использования прикладными программами системных ресурсов операционной системы и реализуемых ею функций. API описывает совокупность функций и процедур, принадлежащих ядру или надстройкам ОС. API представляет собой набор функций, предоставляемых системой программирования разработчику прикладной программы и ориентированных на организацию взаимодействия результирующей прикладной программы с целевой вычислительной системой. **Целевая вычислительная система** представляет собой совокупность программных и аппаратных средств, в окружении которых выполняется результирующая программа, порождаемая системой программирования на основании кода исходной про-

[9] *RTL* включает в себя те стандартные подпрограммы, которые система программирования подставляет на этапе компиляции. В общем случае RTL включает в себя не только модули из системы программирования, но и модули самой ОС.

граммы, созданного разработчиком, а также объектных модулей и библиотек, входящих в состав системы программирования.

В принципе API используется не только прикладными, но и многими системными программами как в составе ОС, так и в составе системы программирования. Функции API позволяют разработчику строить результирующую прикладную программу так, чтобы использовать средства целевой вычислительной системы для выполнения типовых операций. При этом разработчик программы избавлен от необходимости создавать исходный код для выполнения этих операций. Программный интерфейс API включает в себя не только сами функции, но и соглашения об их использовании, которые регламентируются операционной системой, архитектурой целевой вычислительной системы и системой программирования. Существует несколько вариантов реализации API:

- реализация на уровне ОС;
- реализация на уровне системы программирования;
- реализация на уровне внешней библиотеки процедур и функций.

Система программирования в каждом из этих вариантов предоставляет разработчику средства для подключения функций API к исходному коду программы и организации их вызовов. Объектный код функций API подключается к результирующей программе компоновщиком при необходимости.

Возможности API можно оценивать со следующих позиций [2]:

- эффективностью выполнения функций API, характеризуемой скоростью выполнения функций и объемом вычислительных ресурсов, требующихся для их выполнения;
- широтой предоставляемых возможностей;
- степенью зависимости прикладной программы от архитектуры целевой вычислительной системы.

В идеале хотелось бы иметь набор функций API, выполняющихся с наивысшей эффективностью, предоставляющих пользователю все возможности современных ОС и имеющих минимальную зависимость от архитектуры вычислительной системы, а еще лучше – лишенных такой зависимости.

Добиться наивысшей эффективности выполнения функций API практически трудно по тем же причинам, по которым невозможно добиться наивысшей эффективности выполнения для любой результирующей программы. Поэтому об эффективности API можно говорить только в сравнении его характеристик с другим API.

Что касается двух других показателей, то в принципе нет никаких технических ограничений на их реализацию. Однако существуют организационные проблемы и узкие корпоративные интересы, тормозящие создание такого рода библиотек.

5.4.2 Функции API на различных уровнях реализации

5.4.2.1 Реализация функций API на уровне ОС

При реализации функций API на уровне ОС за их выполнение ответственность несет ОС. Объектный код, выполняющий функции, либо непосредственно входит в состав ОС (или даже ядра ОС), либо поставляется в составе динамически загружаемых библиотек, разработанных для данной ОС. Система программирования ответственна только за организацию интерфейса для вызова этого кода [2].

В таком варианте результирующая программа обращается непосредственно к ОС. Поэтому достигается наибольшая эффективность выполнения функций API по сравнению со всеми другими вариантами реализации API.

Недостатком организации API по такой схеме является практически полное отсутствие переносимости не только кода результирующей программы, но и кода исходной программы. Программа, созданная для одной архитектуры вычислительной системы, не сможет исполняться на вычислительной системе другой архитектуры даже после того, как ее объектный код будет полностью перестроен. Чаще всего система программирования не сможет выполнить перестроение исходного кода для новой архитектуры вычислительной системы, поскольку многие функции API, ориентированные на определенную ОС, будут в новой архитектуре просто отсутствовать.

Таким образом, в данной схеме для переноса прикладной программы с одной целевой вычислительной системы на другую необходимо изменение исходного кода программы.

Перенос можно осуществить при условии унифицирования функций API в различных ОС. С учетом корпоративных интересов производителей ОС такое направление их развития представляется практически невозможным. В лучшем случае при приложении определенных организационных усилий удается добиться стандартизации смыслового (семантического) наполнения основных функций API, но не их программного интерфейса. Хорошо известным примером API такого рода может служить набор функций, предоставляемых пользователю со стороны ОС типа Microsoft Windows – WinAPI (Windows API). Надо сказать, что даже внутри этого корпоративного API существует определенная несогласованность, которая несколько ограничивает переносимость программ между различными ОС типа Windows. Другим примером такого API можно считать набор сервисных функций ОС типа MS DOS, реализованный в виде набора подпрограмм обслуживания программных прерываний.

5.4.2.2 Реализация функций API на уровне системы программирования

Если функции API реализуются на уровне системы программирования, они предоставляются пользователю в виде библиотеки функций соответствующего языка программирования. Обычно речь идет о библиотеке времени исполнения RTL. Система программирования предоставляет пользователю библиотеку соответствующего языка программирования и обеспечивает подключение к результирующей программе объектного кода, ответственного за выполнение этих функций [2].

Очевидно, что эффективность функций API в таком варианте будет несколько ниже, чем при непосредственном обращении к функциям ОС. Это происходит вследствие того, что для выполнения многих функций API библиотека RTL языка программирования должна все равно выполнять обращения к функциям ОС. Наличие всех необходимых вызовов и обращений к

функциям ОС в объектном коде RTL обеспечивает система программирования.

Однако переносимость исходного кода программы в таком варианте будет самой высокой, поскольку синтаксис и семантика всех функций будут строго регламентированы в стандарте соответствующего языка программирования. Они зависят от языка и не зависят от архитектуры целевой вычислительной системы. Поэтому для выполнения прикладной программы на новой архитектуре вычислительной системы достаточно заново построить код результирующей программы с помощью соответствующей системы программирования.

Единообразное выполнение функций языка обеспечивается системой программирования. При ориентации на различные архитектуры целевой вычислительной системы в системе программирования могут потребоваться различные комбинации вызовов функций ОС для выполнения одних и тех же функций исходного языка. Это должно быть учтено в коде RTL. В общем случае для каждой архитектуры целевой вычислительной системы будет требоваться свой код RTL языка программирования. Выбор того или иного объектного кода RTL для подключения к результирующей программе система программирования обеспечивает автоматически.

Например, рассмотрим функции динамического выделения памяти в языках C и Pascal. В языке C это функции malloc, realloc и free (функции new и delete в C++), в Pascal – функции new и dispose. Если использовать эти функции в исходном тексте программы, то с точки зрения исходной программы они будут действовать одинаковым образом и зависеть только от семантики исходного кода. При этом для разработчика исходной программы не имеет значения, на какую архитектуру ориентирована его программа. Вышесказанное имеет значение для системы программирования, которая для каждой из этих функций должна подключить к результирующей программе объектный код библиотеки, выполняющий обращение к соответствующим функциям ОС. Не исключено даже, что для однотипных по смыслу функций в разных языках (например, malloc в C и new в языке Pascal выполняют схожие по смыслу действия) этот код будет выполнять схожие обращения к ОС. Однако для различ-

ных вариантов ОС код будет различен даже при использовании одного и того же исходного языка.

Проблема, главным образом, заключается в том, что большинство языков программирования предоставляют пользователю недостаточно широкий набор стандартизованных функций. Поэтому разработчик исходного кода существенно ограничен в выборе доступных функций API. Как правило, набора стандартных функций оказывается недостаточно для создания полноценной прикладной программы. Тогда разработчик может обратиться к функциям других библиотек, имеющихся в составе системы программирования. В этом случае нет гарантии, что функции, включенные в состав данной системы программирования, но не входящие в стандарт языка программирования, будут доступны в другой системе программирования, особенно если она ориентирована на другую архитектуру целевой вычислительной системы. Такая ситуация уже ближе к третьему варианту реализации API.

Например, те же функции malloc, realloc и free в языке C фактически не входят в стандарт языка. Они входят в состав стандартной библиотеки, которая де-факто входит во все системы программирования, построенные на основе языка C. Общепринятые стандарты существуют для многих часто используемых функций языка. Если же взять более специфические функции, такие как функции порождения новых процессов, то для них ни в C, ни в языке Pascal не окажется общепринятого стандарта.

5.4.2.3 Реализация функций API с помощью внешних библиотек

Реализация функций API с помощью внешних библиотек осуществляется посредством предоставления пользователю данных функций в виде библиотеки процедур и функций, созданной сторонним разработчиком [2]. Причем разработчиком такой библиотеки может выступать тот же самый производитель.

Система программирования ответственна только за подключение объектного кода библиотеки к результирующей про-

грамме. Причем внешняя библиотека может быть и динамически загружаемой, т.е. загружаемой во время выполнения программы.

С точки зрения эффективности выполнения этот метод реализации API имеет самые низкие результаты, поскольку внешняя библиотека обращается как к функциям ОС, так и к функциям RTL языка программирования. Только при очень высоком качестве внешней библиотеки ее эффективность становится сравнимой с библиотекой RTL.

Если говорить о переносимости исходного кода, то здесь требование только одно – используемая внешняя библиотека должна быть доступна в любой из архитектур вычислительных систем, на которые ориентирована прикладная программа. Тогда удается достигнуть переносимости. Это возможно, если используемая библиотека удовлетворяет какому-то принятому стандарту, а система программирования поддерживает этот стандарт.

Например, библиотеки, удовлетворяющие стандарту POSIX (см. п. 5.5.3), доступны в большинстве систем программирования для языка C, и если прикладная программа использует только библиотеки этого стандарта, то ее исходный код будет переносимым. Еще одним примером является широко известная библиотека графического интерфейса XLib, поддерживающая стандарт графической среды X Window.

Для большинства специфических библиотек отдельных разработчиков это не так. Если пользователь использует какую-то библиотеку, то она ориентирована на ограниченный набор доступных архитектур целевой вычислительной системы. Примерами могут служить библиотеки MFC (Microsoft Foundation Classes) фирмы Microsoft и VCL (Visual Controls Library) фирмы Borland, жестко ориентированные на архитектуру ОС типа Windows.

Тем не менее многие фирмы-разработчики сейчас стремятся создать библиотеки, которые бы обеспечивали переносимость исходного кода приложений между различными архитектурами целевой вычислительной системы. Пока еще такие библиотеки не получили широкого распространения, но есть несколько попыток их реализации: например, кросс-платформенная библиотека

компонент CLX (Component Library for Cross-platform, «кликс») производства фирмы Borland, ориентированная на архитектуру ОС типа Linux и ОС типа Windows.

В целом развитие функций прикладного API идет в направлении попыток создания библиотек API, обеспечивающих широкую переносимость исходного кода. Однако, учитывая корпоративные интересы различных производителей и сложившуюся ситуацию на рынке системного программного обеспечения, в ближайшее время вряд ли удастся достичь значительных успехов в этом направлении. Разработка широко применимого стандарта API пока еще остается делом будущего.

Поэтому разработчики системных программ вынуждены оставаться в довольно узких рамках ограничений стандартных библиотек языков программирования.

Что касается прикладных программ, то гораздо большую перспективу для них предоставляют технологии, связанные с разработками в рамках архитектуры «клиент-сервер» или трехуровневой архитектуры создания приложений. В этом направлении ведущие производители ОС, СУБД и систем программирования скорее достигнут соглашений, чем в направлении стандартизации API.

Отметим еще один очень важный момент: желательно, чтобы интерфейс прикладного программирования не зависел от системы программирования. Конечно, имелись персональные компьютеры, у которых базовой системой программирования выступал интерпретатор с языка Basic, но это скорее исключение. Обычно базовые функции API не зависят от системы программирования и могут использоваться из любой системы программирования, хотя и с применением соответствующих правил построения вызывающих последовательностей. В то же время в ряде случаев система программирования может сама генерировать обращения к функциям API. Например, мы можем написать в программе вызов функции по запросу 256 байт памяти

```
unsigned char * ptr = malloc (256).
```

Система программирования языка С сгенерирует целую последовательность обращений. Из кода пользовательской программы будет осуществлен вызов библиотечной функции malloc, код которой расположен в RTL языка С. Библиотека

времени выполнения в данном случае реализует вызов malloc уже как вызов системной функции API HeapAlloc

```
LPVOID HeapA11oc(
HANDLE hHeap,   // указатель на блок
DWORD dwFlags,  // свойства блока
DWORD dwBytes    // размер блока
```

Параметры выделяемого блока памяти в таком случае задаются системой программирования, и пользователь лишен возможности задавать их напрямую. С другой стороны, если это необходимо, возможно использование функций API прямо в тексте программы.

```
unsigned char * ptr = (LPVOID) HeapAlloc(
GetProcessHeap(). 0. 256).
```

В этом случае программирование вызова немного усложняется, но получаемый конечный результат будет, как правило, короче и, что самое важное, будет работать эффективнее. Следует отметить, что далеко не все возможности API доступны через обращения к функциям системы программирования. Непосредственное обращение к функциям API позволяет пользователю обращаться к системным ресурсам более эффективным способом. Однако это требует знания функций API, количество которых нередко достигает нескольких сотен.

Как правило, API не стандартизированы. В каждом конкретном случае набор вызовов API определяется, прежде всего, архитектурой ОС и ее назначением. В то же время принимаются попытки стандартизировать некоторый базовый набор функций, поскольку это существенно облегчает перенос приложений из одной ОС в другую. Таким примером может служить известный и, пожалуй, самый распространенный стандарт – стандарт POSIX. В этом стандарте перечислен большой набор функций, их параметров и возвращаемых значений. Стандартизированными, согласно POSIX, являются не только обращения к API, но и файловая система, организация доступа к внешним устройствам, набор системных команд (команд управления процессами). Использование в приложениях этого стандарта позволяет в дальнейшем легко переносить такие программы из одной ОС в другую путем простейшей перекомпиляции исходного текста.

Частным случаем попытки стандартизации API является внутренний корпоративный стандарт компании Microsoft, известный как WinAPI. Он включает в себя следующие реализации: Win16, Win32s, Win32, WinCE. С точки зрения WinAPI, который в силу ряда идеологических причин является обязательным графическим «оконным» интерфейсом пользователя, базовой задачей выступает окно. Таким образом, WinAPI изначально ориентирован на работу в графической среде. Однако базовые понятия дополнены традиционными функциями, в том числе частично поддерживается стандарт POSIX.

5.4.3 Платформенно-независимый интерфейс POSIX

Платформенно-независимый системный интерфейс для компьютерного окружения POSIX (Portable Operating System Interface for Computer Environments) – это стандарт IEEE[10], описывающий системные интерфейсы для открытых ОС, в том числе оболочки, утилиты и инструментарии [16]. Помимо этого, согласно POSIX, стандартизированными являются задачи обеспечения безопасности, задачи реального времени, процессы администрирования, сетевые функции и обработка транзакций. Стандарт базируется на UNIX-системах, но допускает реализацию и в других ОС. POSIX возник как попытка всемирно известной организации IEEE пропагандировать переносимость приложений в UNIX-средах путем разработки абстрактного, платформенно-независимого стандарта. Однако POSIX не ограничивается только UNIX-системами; существуют различные реализации этого стандарта в системах, которые соответствуют требованиям, предъявляемым стандартом IEEE Standard 1003.1-1990 (POSIX.1). Например, известная ОС реального времени QNX соответствует спецификациям этого стандарта, что облегчает перенос приложений в эту систему, но UNIX-системой не является ни в каком виде, ибо ее архитектура использует абсолютно иные принципы.

Этот стандарт подробно описывает систему виртуальной памяти VMS (Virtual Memory System,), многозадачность MPE

[10] IEEE (Institute of Electrical and Electronics Engineers) – институт инженеров по электротехнике и радиоэлектронике.

(MultiProcess Executing) и технологию переноса операционных систем CTOS (An Operating System produced Convergent Technology …). Таким образом, на самом деле POSIX представляет собой множество стандартов, именуемых POSIX.I – POSIX.12. В табл. 5.1 приведены основные направления, описываемые данными стандартами. Следует также особо отметить, что в POSIX. 1 предполагается язык С в качестве основного языка описания системных функций API.

Таблица 5.1 – Семейство стандартов POSIX

Стандарт	Стандарт ISO	Краткое описание
POSIX.0	Нет	Введение в стандарт открытых систем. Данный документ не является стандартом в чистом виде, а представляет собой рекомендации и краткий обзор технологий
POSIX.1	Да	Системный API (язык С)
POSIX.2	Нет	Оболочки и утилиты, одобренные IEEE
POSIX.3	Нет	Тестирование и верификация
POSIX.4	Нет	Задачи реального времени и нити
POSIX.5	Да	Использование языка ADA применительно к стандарту POSIX.1
POSIX.6	Нет	Системная безопасность
POSIX.7	Нет	Администрирование системы
POSIX.8	Нет	Сети «Прозрачный» доступ к файлам Абстрактные сетевые интерфейсы, не зависящие от физических протоколов вызова удаленных процедур RPC (Remote Procedure Calls)
POSIX.9	Да	Использование языка FORTRAN применительно к стандарту POSIX.1
POSIX.10	Нет	Super-computing Application Environment Profile (AEP)
POSIX.11	Нет	Обработка транзакций AEP
POSIX.12	Нет	Графический интерфейс пользователя GUI

Таким образом, программы, написанные с соблюдением данных стандартов, будут одинаково выполняться на всех POSIX-совместимых системах. Однако стандарт в некоторых случаях носит лишь рекомендательный характер. Часть стандартов описана очень строго, тогда как другая часть только поверхностно раскрывает основные требования.

Нередко программные системы заявляются как POSIX-совместимые, хотя таковыми их назвать нельзя. Причины кроются в формальности подхода к реализации POSIX-интерфейса в различных ОС. К сожалению, достаточно часто с целью увеличения производительности той или иной подсистемы либо из соображений введения фирменных технологий, которые ограничивают использование приложения соответствующей операционной средой, при программировании используются другие функции, не отвечающие стандарту POSIX.

Реализации POSIX API на уровне операционной системы различны. Если UNIX-системы в своем абсолютном большинстве изначально соответствуют спецификациям IEEE Standard 1003.1-1990, то WinAPI не является POSIX-совместимым. Однако для поддержки данного стандарта в операционной системе MS Windows NT введен специальный модуль поддержки POSIX API, работающий на уровне привилегий пользовательских процессов. Данный модуль обеспечивает конвертацию и передачу вызовов из пользовательской программы к ядру системы и обратно, работая с ядром через Win API. Прочие приложения, созданные с использованием WinAPI, могут передавать информацию POSIX-приложениям через стандартные механизмы потоков ввода/вывода (stdin, stdout).

Вопросы для самопроверки

1. Перечислите и поясните основные принципы построения ОС.
2. Расскажите о концепции построения микроядерной ОС. Какие основные функции должно выполнять микроядро ОС?
3. Расскажите о концепции построения ОС с монолитным ядром.

4. Какие задачи возлагаются на интерфейс прикладного программирования (API)?

5. Какими могут быть варианты реализации API? В чем их достоинства и недостатки?

6. Что такое библиотека времени выполнения?

7. Что такое POSIX? Какими преимуществами обладают программы, созданные с использованием только стандартных функций, предусмотренных POSIX?

Часть 2. РЕАЛЬНЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

6. ОПЕРАЦИОННЫЕ СИСТЕМЫ ФИРМЫ MICROSOFT

6.1 Операционная система MS DOS

6.1.1 История ОС MS DOS

Операционная система MS DOS (Microsoft Disk Operation System) – **однозадачная, однопроцессорная, однопользовательская система для управления 16-разрядным персональным микрокомпьютером IBM PC.** Первой разработкой MS DOS можно считать операционную систему для персональных ЭВМ, созданную фирмой Seattle Computer Products в 1980 г. В конце 1980 г. система, первоначально названная QDOS, была модифицирована и переименована в 86-DOS. Право на использование операционной системы 86-DOS было куплено Корпорацией Microsoft, заключившей контракт с фирмой IBM и обязавшейся разработать JC для новой модели персональных компьютеров, выпускаемых фирмой. Когда новый компьютер IBM PC приобрел широкую популярность (1981 г.), его операционная система представляла собой модифицированную версию системы 86-DOS, названную PC DOS, версия 1.0.

Вскоре после выпуска IBM PC на рынке стали появляться персональные компьютеры, «схожие с PC». Операционная система этих компьютеров, предоставленная корпорацией Microsoft в распоряжение фирм, производивших такие машины, представляла собой точную копию операционной системы PC DOS и получила название MS DOS версия 1.0. Единственное серьезное различие этих систем состояло в «уровне системы», что означало необходимость приобретения собственной ОС для каждой машины при использовании ОС MS DOS. Отличительные особенности каждой системы мог выявить только системный программист, в чьи обязанности входила работа по «подгонке» операционной системы к конкретной машине. Пользователь, работающий на разных машинах, не ощущал никакой разницы между ними.

С момента выпуска операционные системы PC DOS и MS DOS совершенствовались параллельно и аналогичным образом. В 1982 году появились *версии 1.1*. Главным преимуществом новой версии была возможность использования двухсторонних дискет (версия 1.0 позволяла работать только с односторонними дискетами), а также возможность пересылки принтеровского вывода на другие устройства. В 1983 году были разработаны *версии 2.0*. По сравнению с предыдущими они давали возможность использовать жесткий диск, обеспечивали усложненный иерархический каталог диска, включали встроенные устройства для дискет и систему управления файлами. MS DOS *версии 3.0*, выпущенной в 1984 г., предоставляла улучшенный вариант обслуживания жесткого диска и подсоединенных к компьютеру микрокомпьютеров. Последующие версии, включая версию 3.3, появившуюся в 1987 г., развивались в том же направлении. MS DOS *версии 5.0* предоставляла возможность использования памяти, расположенной выше 1Мб. В MS DOS *версии 6.0* были расширены возможности использования памяти, расположенной выше 1Мб, добавлены утилита оптимизации использования памяти Memmaker и средство увеличения эффективного дискового пространства DoubleSpace. В комплект поставки включены утилиты проверки и оптимизации жесткого диска ScanDisk и Defrag.

6.1.2 Основные части MS DOS

Основными частями операционной системы MS DOS являются [1]:

BIOS (Base Input Output System Data Area) – базовая система ввода-вывода, представляющая собой базовый набор стандартно присутствующих в системе драйверов, которые находятся в ПЗУ (постоянном запоминающем устройстве). BIOS является только частью ПЗУ. Драйвер – это программа, которая обеспечивает обслуживание определенных устройств, осуществляющих управление и обмен информацией. BIOS, с одной стороны, является внешним компонентом для MS DOS, но с другой стороны, операционная система MS DOS без него не работает. Некоторые операционные системы отказываются от BIOS, тогда ядро операционной системы работает с аппаратным обеспече-

нием («железом») напрямую. Таким образом организована работа операционных систем OS/2, UNIX, Windows NT;

загрузчик операционной системы (BootRecord) – это очень короткая программа, находящаяся в первом секторе диска с операционной системой MS DOS[11]. Функция данной программы заключается в считывании в память двух модулей операционной системы IO.SYS и MSDOS.SYS, которые и завершают процесс загрузки ядра MS DOS. Жесткий диск (винчестер) может быть разбит на несколько логических дисков (разделов), в связи с этим в загрузчике операционной системы выделяются две части: первая часть загрузчика находится в первом секторе жесткого диска и предназначается для выбора раздела жесткого диска, с которого следует продолжить загрузку; вторая – находится в первом секторе выбранного раздела и предназначается для считывания в память модулей DOS и передачи им управления;

ядро ОС включает следующие основные компоненты:

IO.SYS – систему ввода/вывода для устройств, которые не предусмотрены в BIOS;

MSDOS.SYS – обработчик прерываний, реализующий основные высокоуровневые функции MS DOS;

командный процессор (интерпретатор команд) – программа, которая обрабатывает команды, вводимые пользователем. Команды делятся на внутренние и внешние. Внутренние команды командный процессор выполняет сам, а для выполнения внешних команд ищет на диске программу с соответствующим именем и при обнаружении загружает в память и передает ей управление. По окончании работы программы командный процессор удаляет программу из памяти и выводит сообщение о готовности к выполнению последующих команд. Командный процессор находится в дисковом файле COMMAND.COM;

утилиты (внешние команды MS DOS) – специальные программы, которые являются частью операционной системы и выполняют некий стандартный набор команд, например форма-

[11] В настоящее время к персональному компьютеру может быть подключено до 2-х накопителей на гибких дисках и до 4-х накопителей на жестких дисках.

тирование и проверку дисков, отображение информации о доступной памяти и др.;

файлы конфигурации – это файлы CONFIG.SYS, AUTO-EXEC.BAT, в которых выполняется процесс загрузки и инициализации конкретной конфигурации системы, настроенный на определенную рабочую обстановку. В файле CONFIG.SYS может содержаться вызов дополнительных драйверов.

6.1.3 Последовательность загрузки MS DOS

Начальная загрузка MS DOS выполняется автоматически в следующих случаях [1]:

- при включении электропитания компьютера;
- при нажатии на клавишу «Reset» на корпусе компьютера (такая клавиша есть не у всех моделей компьютеров);
- при одновременном нажатии клавиш Ctrl-Alt-Del на клавиатуре.

При выполнении начальной загрузки MS DOS работают программы проверки оборудования POST, находящиеся в постоянной памяти компьютера. При обнаружении ошибок на экран выводится код ошибки. Самым длительным этапом проверки оборудования является проверка оперативной памяти.

После окончания работы программ тестирования оборудования запускается программа начальной загрузки *BootRecord*. Она пытается прочесть с диска программу загрузчика операционной системы. После того как с диска, с которого загружается операционная система, прочитана программа-загрузчик, программа *BootRecord* считывает в память компьютера модули операционной системы (для MS DOS – файлы IO.SYS и MSDOS.SYS) и передает им управление. Далее с того же диска читается файл конфигурации системы CONFIG.SYS и в соответствии с указаниями, содержащимися в этом файле, загружаются драйверы устройств и устанавливаются параметры операционной системы. Если файл CONFIG.SYS отсутствует, все параметры устанавливаются по умолчанию. После этого с диска, с которого загружается ОС, читается командный процессор (файл COMMAND.COM) и ему передается управление. Командный процессор выполняет командный файл AUTO-EXEC.BAT, если

этот файл имеется в корневом каталоге диска, с которого загружается ОС. В файле AUTOEXEC.BAT указывают команды и программы, выполняемые при каждом запуске компьютера. Например, в нем можно указать запуск программы, обеспечивающей ввод русских букв с клавиатуры. Если файл AUTOEXEC.BAT не найден в корневом каталоге диска, с которого загружается ОС, то MS DOS запрашивает у пользователя текущую дату и время. После выполнения файла AUTOEXEC.BAT процесс загрузки операционной системы заканчивается и MS DOS выдает приглашение.

6.1.4 Файл конфигурации MSDOS CONFIG.SYS

BIOS является общей и неизменяемой частью операционной системы. Изменение BIOS – нетривиальная задача, тесно связанная с особенностями аппаратуры конкретной модели персонального компьютера. Расширение BIOS с помощью дополнительного модуля MS DOS, который сравнительно легко модифицируется с учетом требований пользователя, придает гибкость операционной системе. С помощью механизма прерываний можно «перекрывать» вызов функций BIOS и дополнять дополнительными подпрограммами, обслуживающими новые внешние устройства. Необходимость подключения новых драйверов внешних устройств, а также изменение других параметров ОС указываются в файле конфигурации CONFIG.SYS.

Команды конфигурирования могут указывать:

1) дополнительные драйверы, которые необходимо подключить к MS DOS (DEVICE=HIMEM.SYS);

2) режим реагирования на нажатие клавиш Ctrl-Break (Ctrl-C) (BREAK=ON);

3) количество файлов, которые могут быть открыты одновременно (FILES=200);

4) количество буферов для обмена информацией с дисковым накопителем (BUFFERS=5);

5) имя файла, который будет играть роль нестандартного командного процессора вместо COMMAND.COM (SHELL=4DOS.COM).

6.1.5 Работа интерпретатора команд COMMAND.COM

В отличие от ядра файл с командным процессором может занимать на системном диске любое место и восприниматься как обычная программа. Основные функции командного процессора заключаются в следующем:

- прием и разбор команд, полученных с клавиатуры или из командного файла;
- исполнение встроенных команд MS DOS, находящихся внутри файла COMMAND.COM;
- загрузка и исполнение внешних программ MS DOS и прикладных программ;
- исполнение файла автозапуска AUTOEXEC.BAT.

Когда в качестве команды MS DOS командный процессор встречает имя файла, не совпадающее с именами встроенных команд, производится анализ типа этого файла, указанного в каталоге. Файлы типов COM и EXE считаются загрузочными и обрабатываются соответствующим образом, а файл BAT трактуется как командный.

Когда в качестве команды MS DOS используется имя командного файла, командный процессор последовательно читает и интерпретирует содержащиеся в нем строки, каждая из которых может содержать одну команду, метку или комментарий. Если в очередной строке стоит команда, осуществляющая вызов некоторой программы, то интерпретация командного файла приостанавливается и начинается работа вызванной программы. После ее завершения управление возвращается к командному процессору.

Язык команд MS DOS служит основным средством общения пользователя с операционной системой. После вызова какой-либо прикладной программы пользователь взаимодействует именно с вызванной программой, а не с MS DOS, вплоть до окончания задачи, после чего вновь вступает в действие командный процессор. Общий вид команды MS DOS:

команда аргумент1 ... аргументN /параметр1 ... /параметрN

Здесь *команда* – обязательный аргумент. Аргументы *аргумент1 ... аргументN* требуются не во всякой команде, и кроме того, в некоторых командах часть из них может опускаться. То же са-

мое относится к параметрам (флажкам) – *параметр1 ... параметрN*. Аргументы обычно указывают на объекты, с которыми имеет дело данная команда: имена дисков, каталогов, файлов, внешних устройств. Флажки служат для задания различных модификаций и режимов исполнения команд.

6.1.6 Командный файл автозапуска AUTOEXEC.BAT

В файле AUTOEXEC.BAT удобно размещать различные команды, которые осуществляют всю необходимую настройку системы. Рассмотрим фрагменты некоторых команд, которые полезно вносить в файл AUTOEXEC.BAT.

Фрагмент 1

```
@ECHO OFF
PATH C:\;C:\ARC;\C:\WORK
PROMPT $P$G
VER
DATE /T
```

Команда ECHO OFF отключает вывод на экран сообщений о выполнении команд в BAT-файле. Символ @ перед командой отключает вывод сообщения ECHO OFF на экран.

Начиная со второй строки этого файла, стоят команды, обеспечивающие создание определенной операционной обстановки.

Команда PATH устанавливает альтернативные маршруты поиска исполняемых файлов.

Команда PROMPT задает формат приглашения MS DOS. Задание формата приглашения осуществляется аргументом команды – строкой, состоящей из обычных текстовых и специальных управляющих символов. Управляющие символы снабжаются префиксом (знаком $), отличающим их от обычных символов. Наиболее часто используются следующие управляющие символы:

$P – выдача имени текущего каталога;

$T – выдача текущего времени;

$D – выдача текущей даты;

$G – выдача символа-разделителя >.

В рассматриваемом фрагменте строка PG в команде PROMPT обеспечивает выдачу приглашения следующей формы:

C:\WORK>.

Если бы строка имела вид TDBP$G, то в приглашение были бы включены сообщения о времени и дате:

8:50:01.01 Mon 01-01-02 | C:\WORK>.

Команда VER предназначена для вывода на экран номера текущей версии операционной системы.

Команда DATE /T выводит текущую дату на экран. Данная команда без параметров выведет текущую дату и предложит осуществить ввод новой текущей даты.

Часто система конфигурируется таким образом, чтобы ею могли пользоваться несколько человек. Настройка конфигурации в соответствии с кодом позволяет пользователю погрузиться в требуемую ему среду. В файле AUTOEXEC.BAT могут быть указаны такие параметры операционной обстановки, как имя конкретного рабочего каталога, имена альтернативных каталогов с различными вспомогательными файлами, имя первой исполняемой команды и др.

Фрагмент 2

В файл AUTOEXEC.BAT заносятся команды, позволяющие запрашивать код пользователя, а затем, проанализировав его, передавать управление на одну из ветвей:

```
@ECHO OFF
PATH C:\;C:\ARC;\C:\WORK
PROMPT $P$G
VER
IF EXIST MAIL TYPE MAIL
:BEGIN
        ECHO Пользователи этой машины
        ECHO 1 – Алексей 2 – Петр 3 – Иван 4 – остальные
        CHOICE /C:1234
        IF ERRORLEVEL 4 GOTO WORK
        IF ERRORLEVEL 3 GOTO IVAN
```

```
        IF ERRORLEVEL 2 GOTO PETER
        IF ERRORLEVEL 1 GOTO ALEX
        GOTO BEGIN
:IVAN
        CD IVAN
        START
:PETER
        CD PETER
        START
:ALEX
        CD ALEX
        START
:WORK
        CD WORK
```

В данном примере введена команда IF EXIST MAIL TYPE MAIL, которая обеспечивает вывод на экран файла MAIL, если он присутствует в корневом каталоге.

Строки, начинающиеся с символа двоеточие, такие как :BEGIN, :IVAN, :PETER, :ALEX, :WORK, – это метки, на которые осуществляется переход командой GOTO.

Команда ECHO с текстовой строкой предназначена для вывода этой строки на экран.

Команда CHOICE /C:1234 запрашивает ввод символов с клавиатуры и зависимости от ключей, расположенных за символом двоеточие, команда IF ERRORLEVEL осуществляет переход на ту или иную метку.

Команда CD осуществляет смену текущего каталога.

6.1.7 Командный язык MS DOS и файлы пакетной обработки

Собственно синтаксис командной строки операционной системы MS DOS подробно описан во множестве работ, посвященных этой системе. Там же можно найти описание команд MS DOS, их ключей, параметров и особенностей использования. Можно воспользоваться электронным справочником по операционной системе MS DOS – Teach Help, а также вызовом справ-

ки по команде, заложенной в ОС, используя символы «/?» в качестве аргумента для команды.

Командный файл (файл пакетной обработки) – это текстовый файл в коде ASCII, состоящий из группы команд MS DOS. Правила идентификации командных файлов совпадают с общими правилами идентификации файлов. Единственное исключение – командный файл всегда записывается на диск с расширением «.BAT» (BATch).

Обратиться к командному файлу крайне просто. Набирается команда старта, т.е. имя файла, и нажимается клавиша Enter. После введения команды файл выбирается из рабочего каталога указанного или рабочего диска. Если в рабочем каталоге его нет, то поиск файла будет производиться в каталогах, описанных командой PATH или APPEND. При нахождении файла первая из его команд загружается в память, отображается на экране и выполняется. Этот процесс повторяется последовательно для всех команд файла (от первой до последней команды).

Команды MS DOS подразделяются на внутренние, встроенные в саму операционную систему, и внешние, выполненные в виде отдельных файлов, имя которых и является командой.

К командам помощи в MS DOS относятся: <команда> /?, fasthelp, help. Командой смена текущего диска является команда: *<имя_диска>*. С диска c: начинаются логические диски жесткого диска(ов). Диски a: и b: – это дискеты.

Рассмотрим другие основные команды MS DOS.

Командой установки (+) или снятия (–) заданного(ных) атрибута(ов) является команда:

ATTRIB <+/-атрибут(ы)> <ключи> <имя файла(ов)>

где атрибуты показывают тип файла:

r – файл только для чтения (read only);

a – архивный файл (archive);

h – скрытый файл (hidden);

s – системный файл (system).

BACKUP – программа-оболочка для создания резервных копий.

CALIBRATE <дисковод:> <параметры> – проверка надежности жесткого диска.

CD – смена текущей директории (cd\ – зайти в корневую директорию; cd kat – зайти в директорию KAT текущей директории; cd .. – подняться на один уровень вверх по файловой структуре).

CHECKDISK – проверка жесткого диска на наличие ошибок.

CLS – очистка экрана монитора.

COPY <ключи> <источник> <ключи> <назначение> – копирование файла(ов) из источника в назначение. В качестве источника и назначения могут служить логические имена физических устройств:

CON – консоль (при вводе – клавиатура; при выводе – монитор);

LPT – параллельный порт;

COM и AUX – последовательный порт;

PRN – принтер.

В данной команде используются следующие значения ключей для файлов:

/a – файл расматривается как последовательность символов;

/b – файл расматривается как последовательность бит;

/v – копирование происходит с проверкой.

Команда *COPY* может служить и для обьединения файлов: *COPY имя_файла_1 имя_файла_n имя_общего_файла*.

DATE <нужная дата> – настройка системного календаря.

DEFRAG <имя диска> <параметры> – устранение фрагментации диска, где параметры: /f – полная оптимизация диска; /u – только дефрагментация.

DEL <имя файла> – удаление указанного файла.

DIR<путь> <ключи> – просмотр списка поддиректорий и файлов в текущей директории, где используются ***следующие значения ключей***:

/p – постраничный вывод информации;

/w – вывод на всю ширину экрана;

/a: – вывод файлов по заданным атрибутам, где после знака «:» приводятся следующие параметры:

r – файлы только для чтения;

h – скрытые файлы;

s – системные файлы;

a – архивные файлы:

d – директории:

/o: – отсортированный вывод файлов, где после знака «:» приводятся следующие параметры:

n – по имени;

e – по расширению;

s – по размеру;

d – по дате и времени создания;

g – вывод, начиная с директорий с сортировкой по имени;

a – по последней дате открытия;

/s – вывод с включением поддиректорий;

/b – вывод только имен файлов;

/l – использование нижнего регистра;

/v – отображение расширенных сведений о файлах и папках;

/4 – вывод четырех цифр года (если не указан ключ /V).

DISKCOPY <имя первого диска> <имя второго диска> <параметры> – копирование содержимого одного диска на другой, используется в основном для копирования дискет.

EDIT – вызов текстового редактора.

ERASE <имя файла> – удаление указанного файла.

FDISK <параметры> – разбиение жесткого диска на отдельные логические диски.

FC <параметры> <имя первого файла(ов) или директории(й)> <имя второго файла(ов) или директории(й)> – сравнение файлов или директорий.

FORMAT <диск:> <параметры> – форматирование дисков.

HELP <тема-справочника> – вызов интерактивного справочника.

LABEL <диск:> <метка (до 11 символов)> – задание метки диска.

MD <имя директории> – создает указанную директорию или подиректорию.

MEM – выдача справки о состоянии памяти компьютера (mem /c /p – выдача более подробной справки о расположении программ в памяти с постраничным выводом на экран).

MOVE <источник> <назначение> <ключи> – перемещение файла(ов) из источника в назначение.

PROMPT <текст> – задание приглашения MS DOS.

RAMDRIVE – создание в оперативной памяти виртуального диска.

REM <старое имя> <новое имя> – переименование файла(ов) или директории(й).

RD <имя директории> – удаление пустой директории.

SCANDISK – подобно *CHECKDISK* – проверка жесткого диска на наличие ошибок, но более продвинутая.

SMARTDRV – инициация ускорения работы жесткого диска и CD-ROM (по своей сути является программным кэшем).

SPEEDDISK <дисковод:> <режимы> – оптимизация размещения файлов на диске.

TIME <час минуты секунды> – настройка системного времени.

TYPE <имя файла> – просмотр файла в текстовом виде.

VER – предоставление справки об установленной на компьютере версии DOS или WINDOWS. Если на компьютере установлена ОС Windows, то команда ver выдаст "Windows <номер версии>".

VERIFY (on/off) – проверка записи файлов при копировании или перемещении.

VOL <диск:> – выдача метки указанного диска.

6.2 Операционная система Windows 95

Операционные системы Windows получили свое развитие в 90-х годах XX века. Первая версия была выпущена в ноябре 1985 г, но она не снискала популярности. В мае 1990 г. появилась весьма эффектная и вполне успешная версия Windows 3.0. Появление Windows 3.0 стало тем самым переломным этапом, в ходе которого мир внезапно открыл для себя возможности и достоинства Windows, вследствие чего было продано огромное количество копий системы. Это связано с тем, что аппаратные средства ОС Windows смогли обеспечить должный уровень производительности, достоинства графического интерфейса мгновенно стали очевидны для огромного числа пользователей.

Windows 3.1 была выпущена главным образом для того, чтобы покончить с проблемами, которые были выявлены в ходе широкомасштабного использования Windows 3.0.

Основная цель создания Windows 95 – разработка системы, обладающей удобным пользовательским интерфейсом [17]. Миссия Windows 95 состоит в том, чтобы максимально облегчить использование и обслуживание персонального компьютера, а также унифицировать разработку программного обеспечения и аппаратных средств. **Отличительной особенностью системы Windows 95 явилось преобразование ее из Windows 3.1 в полнофункциональную защищенную 32-разрядную операционную систему.** У пользователя исчезла необходимость в использовании MS DOS. В Windows 95 предусмотрена поддержка приложений MS DOS при помощи средств совместимости, а также для производителей аппаратных средств предоставлена возможность разрабатывать и совершенствовать свою продукцию не в обязательном строгом соответствии старой архитектуре IBM PC. В Windows 95 доступ к любым аппаратным средствам осуществляется при помощи драйверов устройств. Пользователь, если у него есть соответствующий драйвер, легко может доставить к системе новое устройство. Теперь нет необходимости в совместимости с устаревшими BIOS, если, конечно, данное устройство не должно поддерживать также и работу MS DOS.

Появилась спецификация «Plug and Play», созданная совместно фирмами Microsoft, Intel, Phoenix Technologies, Compaq и др. [17]. Цель ее создания состояла в сведении к минимуму проблем, связанных с настройкой и конфигурированием аппаратных средств. Интерфейс «Plug and Play» берет на себя все заботы по идентификации подключенного устройства и обеспечению данного устройства необходимыми аппаратными ресурсами, а также конфигурированию соответствующих драйверов устройств.

Появление 32-разрядной модели памяти позволило [17]:

- создать 32-разрядный интерфейс прикладного программирования Windows API (Application Programming Interface), совместимый с API, который поддерживает Microsoft Windows

NT, вследствие чего стало проще создавать программные приложения, поддерживаемые обеими операционными системами;

- избежать множества ограничений, которые были присущи ранним версиям Windows. Чрезвычайно ценные ранее системные ресурсы стали доступны в изобилии.

Архитектура программ, основанная на 32-разрядной адресации памяти, обеспечивает следующие возможности:

- **вытеснение (preemption).** Приложения Win32 представляют собой полностью вытесняемые программы. Это означает, что операционная система в любой момент может прервать их выполнение и переключиться на другую, обладающую более высоким приоритетом задачу. В общем случае это обеспечивает более плавную реакцию, а также повышение общей производительности системы и позволяет, например, избежать потери данных, которая может произойти в случае, если какому-нибудь приложению слишком долго не удается получить процессор в свое распоряжение;
- **отдельное адресное пространство**. Приложение Win32 выполняется в своей собственной, защищенной области памяти, и никакая другая программа не может нарушить целостность ее кода или данных;
- **поддержка потоков (Thread support).** Часто программам требуется одновременно выполнять два действия, например, производить резервное копирование текущего документа на диск и обеспечивать пользователю возможность редактирования текста, который он видит на экране. При работе под Windows 3.1 такого рода многозадачность в пределах одного приложения представляла собой достаточно сложную для реализации возможность, программируя которую, легко ошибиться.

Конфигурирование Windows 95 является довольно непростой задачей. Сведения для конфигурации находятся в файлах WIN.INI и SYSTEM.INI. Кроме этих файлов, что сами по себе управляют конфигурированием Windows, многие приложения используют собственные файлы инициализации или добавляют некоторую информацию в файл WIN.INI. При анализе содержимого файлов инициализации WIN.INI и SYSTEM.INI появляется ряд вопросов:

- Чем обусловлена необходимость такого большого количества условно необходимых параметров?
- А не могли бы принятые по умолчанию параметры исключить надобность введения новых?
- Неужели улучшенные параметры, принятые по умолчанию, не дадут возможность уменьшить количество дополнительных?
- Если какой-то параметр не нужен, зачем включать его в файл конфигурации?

Понятно, что приведение всех этих проблем в соответствие со здравым смыслом несколько запоздало, однако команда разработчиков Windows 95 взяла на вооружение подход, который в свое время разработали создатели Windows NT. Дело в том, что Windows NT использует специальный файл, называемый registry', в котором содержится вся информация, относящаяся к аппаратным средствам, операционной системе и конфигурации прикладных программ. Содержащиеся в registry параметры доступны прикладным программам через заранее определенный интерфейс прикладного программирования. При этом приложения могут добавлять и изменять свои собственные настройки конфигурации при помощи соответствующих функций API. Пользователь избавлен от необходимости редактировать файлы конфигурации, что автоматически исключает целый ряд ошибок. Windows 95 использует файлы регистрации, так же как и Windows NT, поэтому по мере того как разработчики будут совершенствовать свои программы для Windows 95, все проблемы с настройкой конфигурации должны исчезнуть.

В оболочку операционной системы Windows 95 были добавлены **новые возможности** [17]:

- универсальные механизмы связывания и встраивания объектов OLE 2 (Object Linking and Embedding), явившиеся первым шагом к документно-ориентированной архитектуре приложений. Оболочка Windows 95 поддерживает функции OLE 2 и полный набор возможностей «Drag and Drop» (в русскоязычной версии Windows 95 эта технология называется «Перетащить и Оставить»);
- поддержка интерфейса электронной почты;
- поддержка длинных имен файлов;

- наличие большого набора средств просмотра файлов, позволяющих пользователям заглянуть в файл определенного формата без необходимости запуска приложения, которым этот файл был создан;

- поддержка приложений MS DOS, которые будут жить вечно. Несмотря на то что Windows 95 с ее улучшенной оконной средой приближает их конец, все-таки поддержка приложений MS DOS в Windows 95 была заметно усовершенствована. В число новых возможностей входят действия по изменению размеров окон MS DOS, операции копирования и вставки, а также использование в приложениях MS DOS шрифтов True Type;

- наличие программного обеспечения, создающего условия для представления компьютера как полностью оборудованной машины-клиента NetWare с целью наиболее полной адаптации к сетевым средам. Кроме такой поддержки локальных вычислительных сетей Windows 95 имеет много других возможностей, относящихся к области коммуникаций, – от простейших операций, вроде набора номера телефона, до поддержки самых современных сверхпортативных компьютеров. При этом Windows 95 стремится наилучшим образом выполнять функции операционной системы машины-клиента и обеспечивает:

- поддержку действий машины-клиента для всех популярных сетей фирм Novell, Banyan, Microsoft и других;
- поддержку различных типов машин-клиентов, что позволяет одновременно подключать один и тот же компьютер к различным сетям, например к локальной сети Novell и к глобальной сети WAN (Wide Area Network), построенной с использованием протокола TCP/IP;
- возможность работы компьютера в качестве сервера в одноранговой сети, благодаря чему рабочие группы или небольшие фирмы избавляются от необходимости выделять специальный компьютер для выполнения функций сервера;
- поддержку электронной почты, основанную на интерфейсе прикладного программирования сообщений MAPI (Message Applica-tion Programming Interface), позволяющем работать как с факсимильными устройствами, так и с популярными сетями электронной почты;

- возможности удаленного взаимодействия и управления, которые обеспечивают эффективный доступ к локальной сети и управление ею посредством низкоскоростных соединений. При этом Windows распознает явление «блуждающего компьютера» при поддержке синхронизации версий файлов и эффективной передаче данных по низкоскоростным каналам.

Требование к оборудованию при использовании ОС Windows 95 состоят в следующем: рекомендуется процессор 486 и выше с тактовой частотой 25 Мгц и выше; не менее 8 мегабайт памяти (ОЗУ); не менее 40–45 мегабайт доступного пространства на жестком диске; монитор VGA или с более высоким разрешением; мышь Microsoft Mouse или совместимый манипулятор.

6.3 Операционная система Windows 98

Операционная система Microsoft Windows 98 – это обновление Windows, расширяющее функциональные возможности домашнего компьютера. Простой доступ к Internet, высокая производительность системы, новые служебные программы и средства диагностики увеличивают эффективность работы. Windows 98 улучшает качество воспроизведения графики, звука и мультимедийных приложений по сравнению с Windows 95. Поддержка шины USB (Universal Serial Bus) позволяет легко подключать и отключать внешние устройства, а также просматривать на компьютере телевизионные передачи.

Microsoft Windows 98 второго издания – это обновление популярной операционной системы Windows 98, в котором используются самые современные технологии Internet, имеются средства для создания домашних сетей, поддерживается новейшее оборудование. Во втором издании Windows 98 обеспечивается дополнительная поддержка аппаратных средств и предлагается целый ряд новых возможностей для работы с Internet [17]:

- Internet Explorer 5. Широко распространенные технологии обзора, созданные корпорацией Microsoft, позволяют значительно повысить быстродействие, качество и гибкость при работе в Internet;

- Windows NetMeeting 3. Последняя версия NetMeeting расширяет возможности проведения сетевых конференций, повышает быстродействие, обеспечивает безопасность и поддержку стандартов Internet;

- подключение к Internet с общим доступом ICS (Internet Connection Sharing). ICS – это комплекс передовых технологий, дающих возможность пользователям нескольких компьютеров одновременно получать доступ в Internet через одно общее подключение;

- расширенная поддержка оборудования. Усовершенствована встроенная поддержка таких стандартов, как шина USB, DVD, стандарт IEEE 1394, предназначенный для подключения высокоскоростных последовательных устройств и интерфейса управления питанием и конфигурациями ACPI (Advanced Configuration and Power Interface), поддержка технологий Digital Imaging и Microsoft WebTV для Windows, а также широкополосных сетевых подключений. Увеличено количество подключаемых устройств и упрощена работа с ними;

- простота использования и доступа в Internet. Динамическая справочная система на основе Web-технологии и 15 программ-мастеров упрощают использование компьютера. Web-совместимый интерфейс пользователя Windows 98 облегчает поиск, унифицируя представление информации в компьютере, локальной сети и Web;

- высокая производительность и надежность. Сокращение времени запуска приложений, новые средства очистки диска и повышения эффективности его работы. Все это стало возможным благодаря новшествам, превращающим Windows 98 в мощную и надежную ОС;

- пакет обновления Windows 98, предназначенный для исправления основных неполадок, в том числе связанных с проблемой 2000;

- совместимость с приложениями и технологиями, разработанными для более ранних версий Windows.

Для операционной системы Windows 98 предъявляются следующие **требования к оборудованию**: процессор 486DX/66 МГц или более мощный; 24 Мб оперативной памяти, обеспечи-

вающей минимальную производительность; 260 Мб свободного места на диске, необходимого при работе с файловой системой FAT16 для обычной установки (однако в зависимости от конфигурации системы и количества выбранных компонентов, может потребоваться от 210 до 400 МБ); монитор VGA или с более высоким разрешением; мышь Microsoft Mouse или совместимое указывающее устройство.

6.4 Windows Millennium Edition

Windows Millennium Edition (Windows Me) – **это операционная система для персональных компьютеров, позволяющая получить доступ ко всем многообразным возможностям электронного мира при использовании ПК в качестве домашнего компьютера:**

1) **возможности мультимедиа** [17]:

создание фильмов. Средство Windows Movie Maker позволяет создавать и редактировать видеофильмы, а также предоставлять другим пользователям доступ к ним с помощью электронной почты и Internet;

упрощенный ввод фотографий и предоставление доступа к ним. Новые возможности работы с цифровыми фотографиями позволяют загружать снимки со сканеров и из цифровых фотоаппаратов, а также выполнять над этими фотографиями простые операции, включая поворот и распечатку. После этого нетрудно переслать лучшие снимки по электронной почте родственникам и друзьям, разместить эти снимки на личном Web-узле или подготовить слайды для показа;

музыка и видео. При помощи проигрывателя Windows Media можно загружать из Internet музыкальные произведения или переписывать их с приобретенных компакт-дисков, создавать коллекции музыкальных произведений, используя так называемые списки воспроизведения, переписывать эти произведения на переносной проигрыватель или компакт-диск. Кроме того, имеется возможность прослушивания радиостанций или просмотра видео непосредственно через Internet;

возможности графики и звука. Используя компоненты Di-rectX 7, можно получить качественную трехмерную графику

и оптимальное для данного оборудования воспроизведение звука;

2) **многообразные средства доступа к Internet**, благодаря которым пользователь может поддерживать связь с окружающим миром, используя следующие возможности:

удобство работы в глобальной сети. Web-обозреватель Internet Explorer 5.5 позволяет просматривать Web-ресурсы с применением удобных средств поиска и других современных возможностей, в том числе ускоренного обзора и более удобной печати;

поддержка широкополосных подключений с использованием быстродействующих средств подключения к Internet, включая ADSL и кабельные модемы;

отправка электронной почты с помощью приложения Outlook Express;

служба немедленной передачи сообщений MSN Messenger, с помощи которой можно мгновенно установить, кто именно из ваших друзей или коллег подключен в данный момент к сети, и оперативно обменяться с ним сообщениями;

видеоконференции. Интеграция со службой немедленной передачи сообщений позволяет без видимых затруднений перейти от обмена текстовыми сообщениями к участию в видеоконференциях, организованных с помощью продукта NetMeeting 3.1;

интерактивные игры (нарды, червы, пики, шашки и реверси);

3) **возможности в области создания и эксплуатации домашней сети:**

упрощение работы с домашней сетью. Улучшенные инструментальные средства позволяют предоставлять общий доступ к одному подключению Internet нескольким компьютерам, даже если они работают под управлением разных версий системы Windows;

мастер домашней сети. Процесс создания домашней сети упрощается благодаря простым пошаговым инструкциям;

автоматический общий доступ к принтерам, подключенным к домашней сети;

интерактивные игры по сети. Усовершенствования, внесенные в домашнюю сеть, позволяют играть по ней нескольким игрокам;

4) **возможности по обеспечению надежности ОС для домашнего компьютера**:

восстановление системы. Функция System Restore предоставляет удобную возможность отката системы в предыдущее работоспособное состояние в случае, если в системе возникли неполадки и требуется немедленно восстановить ее работоспособность;

защита системных файлов. Встроенные средства защищают от потери или случайного изменения ключевых системных файлов;

автоматическое обновление с Web-узла Windows Update. Предусмотрен удобный способ регулярной автоматической установки обновлений (включая новые драйверы и расширения) непосредственно с Web-узла Windows Update, что позволяет всегда иметь на компьютере последнюю версию системы;

ускоренные загрузка и возобновление работы;

5) **возможности по поддержке нового аппаратного оборудования**:

«интеллектуальная» установка устройств. Файлы и драйверы распознаются автоматически, что ускоряет и упрощает их установку;

поддержка универсальных самонастраивающихся устройств UPnP (Universal Plug and Play), благодаря чему появляется возможность применять «интеллектуальные» устройства нового поколения;

помощь и поддержка в обеспечении доступа к справочным ресурсам, находящимся как в Internet, так и на самом компьютере, осуществляется централизованно и из удобного места;

настраиваемые меню. Рабочий стол меньше загроможден, поскольку различные меню «приспосабливаются» к манере работы пользователя (в меню отображаются только те пункты и программы, которые используются чаще всего);

поддержка новейшего оборудования. Предусмотрена встроенная поддержка последних моделей оборудования, таких как пятикнопочная мышь, Web-клавиатура, широкополосные USB-модемы.

6.5 Платформа Windows NT

6.5.1 Общие сведения

Ранее рассмотренные операционные системы Windows 95, Windows 98, Windows Me относятся к ОС, построенным на платформе (принципе построении ядра ОС) Windows 9x. Фирмой Microsoft альтернативно платформе Windows 9x была разработана новая платформа – **Windows NT**, являющаяся **наиболее мощной ОС Microsoft,** в которой **впервые нарушена традиция ориентации на Intel**, с тем чтобы дать производителям возможность выбирать в качестве основы для своих систем другие процессоры. Так, Microsoft и ее партнеры создали версии Windows NT для MIPS R4000, DEC Alpha, Power PC и ряда других мощных процессоров. Ни одна из вышеперечисленных микросхем не совместима с семейством процессоров Intel, поэтому единственный способ заставить существующие приложения Windows или MS DOS работать на одном из этих процессоров заключается в обеспечении Windows NT неким эмулятором процессора Intel. На самом деле, работающие под Windows NT компьютеры обычно используются вовсе не для запуска программ MS DOS, в связи с чем в Windows NT существует ряд ограничений на выполнение 16-разрядных приложений.

6.5.2 Windows NT Server 4.0

Windows NT Server 4.0, благодаря новым особенностям Option Pack, поставляемого вместе с Windows NT 4.0, является совершенной средой для написания и размещения сетевых программ [17]. Он наиболее простой в управлении среди всех серверных операционных систем. **Windows NT Server 4.0 был разработан с целью ускорения построения и разработки бизнес-приложений.** Option Pack встраивает прямо в Windows NT Server 4.0 Web-службы, транзакции, компоненты, очереди сообщений. Новые средства управления Windows NT Server 4.0 и Option Pack помогают создавать Web-сайты, управлять их содержанием, изучать посещаемость и способствуют их развитию.

Наличие нескольких Web-сайтов на одной машине, поддержка новых издательских технологий в Web, средства настройки делают Windows NT Server 4.0 лучшей средой для публикации и безопасного использования информации во внутренних сетях предприятий и в Internet.

Основные возможности Windows NT состоят в следующем:

- интерфейс пользователя ОС Microsoft Windows 95 встроен в Windows NT Server 4.0. Это делает взаимодействие с сервером более лёгким и похожим на работу с Windows 95 и Windows NT Workstation 4.0;
- административные мастера содержат стандартные средства по управлению сервером и помогают шаг за шагом в решении каждой задачи. Кроме того, Windows NT Server 4.0 имеет мастера для подключения новых пользователей, лицензирования, установки доступа к файлам и папкам и т.д.;
- эффективные средства диагностики, позволяют отслеживать поток данных на сервер и с сервера на уровне пакетов, облегчают устранение проблем с сетью и предоставляют возможность наблюдения за путём распространения информации;
- системные правила используются для создания единой конфигурации и контроля над рабочей средой пользователей и их действиями;
- менеджер задач используется для наблюдения за приложениями и задачами и получения сведений о производительности Windows NT. По каждой программе, запущенной с рабочей станции, выдаётся справка об объеме занятой памяти и использовании ресурсов процессора;
- Microsoft Index Server автоматически индексирует на сервере всё содержимое и показывает свойства файлов, включая HTML. Это работает для Internet, Intranet и простых серверов;
- разработанный для непрограммистов, подходящий для опытных разработчиков Web-сайтов, Microsoft FrontPage является быстрым и простым в пользовании продуктом для создания и управления профессиональными Web-сайтами.
- протокол PPTP (Point-to-Point Tunneling Protocol) позволяет использовать общедоступные сети, такие как Internet, для создания виртуальных частных сетей. Возможно дальнейшее

развитие протоколов для поддержки TCP/IP соединений и шифрования данных.

При использовании Windows NT Server 4.0 предъявляются следующие **требования к оборудованию**:

для Intel и совместимых с ним процессоров – процессор 486/33 Мгц и выше, Pentium или Pentium PRO; не менее 125 Мбайт свободного пространства на жёстком диске;

для RISC-систем – RISC-процессор, совместимый с Microsoft Windows NT Server версии 4.0; не менее 160 Мбайт свободного пространства на жёстком диске;

Общие требования для Intel-процессоров и RISC-систем: 32 Мб оперативной памяти; устройство для чтения компакт-дисков; VGA, Super VGA или другой графический видеоадаптер, совместимый с Windows NT Server 4.0.

6.5.3 Windows NT Workstation 4.0

Windows NT Workstation 4.0 предназначена для функционирования на клиентских местах в корпоративной системе. Обновленный пользовательский интерфейс Windows NT Workstation 4.0 предоставляет интерфейсные средства, подобные интерфейсу Windows 95: внешний вид, эффективность работы. К основным функциональным особенностям Windows NT Workstation 4.0 относятся [17]:

- возможность предоставления независимого адресного пространства 16-разрядным приложениям, организованным таким образом, что в случае сбоя одного из них остальные не пострадают;
- дополнительная защита особо важного кода операционной системы, драйверов устройств и данных из приложений.
- наличие инструментов удаленного управления и поиска неисправностей;
- возможность проведения администраторами политики поддержки стандартов, необходимых для системных настроек рабочего стола.

К оборудованию Windows NT Workstation 4.0 предъявляются следующие требования:

- системы на основе Intel Pentium или более быстрые;

- системные рабочие станции на основе RISC-процессоров Alpha AXP, MIPS R4X00 или PowerPC; 32 Мб оперативной памяти;
- 110 Мб свободного места на диске; адаптер монитора VGA или лучшего разрешения;
- манипулятор Microsoft Mouse или совместимое устройство.

6.6 Платформа Windows 2000

6.6.1 Windows 2000 Server

Платформа Windows 2000 представляет собой ОС нового поколения для делового использования на самых разнообразных компьютерах – от переносных компьютеров до высококлассных серверов, **основывается на технологии NT и является операционной системой для ведения коммерческой деятельности в Internet.** Windows 2000 Server строится на преимуществах сервера Windows NT Server 4.0, предоставляя при этом целый ряд улучшений, а также новые мощные возможности и наборы функций. Сервер Windows 2000 Server включает в себя службы стандартных каталогов, Web, службы приложений, сети, файлов и печати с мощными возможностями управления «от узла к узлу» и высокой надежностью, что является наилучшей основой для интеграции предприятия в Internet.

К новым возможностям относятся [17]:

- ***дефрагментация диска***. Изменяющаяся с течением времени фрагментация диска может серьезно снизить быстродействие сильно загруженного файлового сервера или вебсервера. Предлагаемые средства улучшают доступность и быстродействие дисков;
- ***выделение дисковых квот*** использования дискового пространства для пользователя и тома, что увеличивает доступность дискового пространства и облегчает планирование используемого места на диске;
- ***динамическое управление томами,*** включающее добавление новых томов, расширение существующих, удаление или добавление зеркала или восстановление дискового массива

RAID 5 при работающем сервере без влияния на работу конечного пользователя;

- ***иерархическое управление памятью,*** осуществляемое посредством автоматического перевода на менее дорогие носители данных, к которым в течение продолжительного времени не производилось обращений; максимизации дискового пространства для данных, к которым доступ осуществляется наиболее часто;
- ***служба репликации файлов FRS –*** репликация данных в сетевых папках и автоматическое обеспечение синхронизации между копиями. Служба FRS увеличивает быстродействие и доступность распределенной файловой системы;
- ***шифрование файловой системы,*** увеличивающее безопасность данных за счет шифрования диска (данные остаются зашифрованными даже при архивации);
- ***управление принтерами с помощью Веба,*** позволяющее осуществлять просмотр и управление принтерами из любого обозревателя;
- ***управление удаленным портом,*** предоставляющее возможность дистанционного управления и настройки портов принтера с любого компьютера, на котором установлена система Windows 2000 Professional;
- ***наличие программы разбора языка XML***. Создание приложений, позволяющих веб-серверу обмениваться данными формата XML[12] как с обозревателем Microsoft Internet Explorer, так и с любым сервером, обладающим возможностью интерпретации формата XML;
- ***перезапуск сеанса FTP,*** усиливающий быстродействие загрузки благодаря возможности возобновления загрузки с того места, на котором она была прервана;
- ***безопасная связь по сети.*** Зашифрованная связь по сети компании «от узла к узлу» с использованием стандарта IPSec помогает защитить передачу важных данных внутри компании от

[12] XML (Extensible Markup Language) – это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов.

намеренного или случайного просмотра. Служба Active Directory обеспечивает при помощи соответствующей политики централизованный контроль за использованием безопасной связи по сети, чтобы обеспечить возможности для ее развертывания;

- ***сетевое качество службы,*** предполагающее введение приоритетов для потоков трафика сетевых приложений в целях повышения эффективности, доступности и быстродействия сети;
- ***делегирование набора административных полномочий*** администраторами сети с помощью службы Active Directory, производимое для отдельных лиц в пределах организации с целью распределения функций управления и улучшения точности администрирования. Делегирование полномочий также помогает уменьшить число доменов, необходимых компаниям для поддержки большой организации с географически разнесенными пунктами;
- ***наличие системы защиты файлов Windows,*** предотвращающей возможную замену важных системных файлов при установке нового программного обеспечения;
- ***защита от записи в режиме ядра,*** исключающая возможность вмешательства ошибочных программ в работу системы;
- ***загрузка в безопасном режиме,*** позволяющая пользователям организовать проверку системы в процессе запуска путем изменения заданных по умолчанию настроек или удаления недавно установленного драйвера, использование которого привело к возникновению неполадок;
- ***наличие планировщика задач,*** организующего вызов любого сценария, документа или программы в любой момент времени или через любой промежуток времени, а также в ответ на такие события, как загрузка системы, вход пользователя или простой системы.

6.6.2 Windows 2000 Professional

Возможности операционной системы Windows 2000 Professional позволяют использовать ее как **основную операционную систему для современных настольных компьютеров и компьютеров Notebook**, используемых на предприятиях любого типа. При создании этой системы корпорацией Microsoft

были сохранены и улучшены все полезные возможности Windows 98 (технология «Plug and Play», простой и понятный пользовательский интерфейс, широкие возможности управления). Кроме того, они были дополнены системой безопасности, средствами управления и обеспечения надежности, характерными для системы Windows NT. Развертывание системы Windows 2000 Professional как на одном компьютере, так и в рамках всемирной сети позволяет повысить эффективность использования компьютерных технологий.

Возможности Windows 2000 Professional достаточно широки, в данном пособии остановимся на наиболее существенных из них [17]:

- защита основных файлов системы от перезаписи при установке приложений. В случае перезаписи файла система защиты Windows File Protection заменит перезаписанный файл правильной версией. Защита системных файлов обеспечивает надежную работу системы Windows 2000 и отсутствие системных сбоев, характерных для более ранних версий системы Windows;
- гарантированность подлинности драйверов данного устройства и уменьшение риска установки несертифицированных драйверов;
- уменьшение вероятности сбоев приложений и незапланированных перезагрузок системы;
- исключение большинства случаев, вызывающих принудительную перезагрузку системы для Windows NT 4.0 и Windows 9x. Во многих случаях даже при установке новых приложений перезагрузка системы не требуется;
- увеличение быстродействия на 25% по сравнению с Windows 9x в системах с оперативной памятью 64 Мб и более;
- использование полной 32-разрядной архитектуры, позволяющей выполнять одновременно больше задач, чем Windows 95 или Windows 98;
- поддержка до 4 Гб оперативной памяти (RAM) и до двух симметричных мультипроцессоров;
- возможность взаимодействия с более ранними версиями Windows на равноправной основе, включая разрешение совместного доступа к таким ресурсам, как папки, принтеры и периферийные устройства;

- поддержка службами Microsoft Windows 2000 клиента для доступа к UNIX-машинам;
- пофайловое шифрование с помощью созданного случайным образом ключа. Процессы шифрования и дешифрования производятся явным для пользователя образом.
- организация помощи в защите данных, передаваемых по сети. Средство IPSec, позволяющее организациям безопасно передавать данные через Internet, является важной частью системы обеспечения безопасности для виртуальных частных сетей VPN (Virtual Private Network);
- автоматическое нахождение прокси-сервера и настройка обозревателя Internet Explorer 5.01 для подключения к Internet.

Требования к оборудованию включают: рекомендованный минимум – pentium-совместимый процессор с частотой 133 Мгц или выше 64 Мб оперативной памяти RAM (принять во внимание, что с увеличением размера оперативной памяти увеличивается и быстродействие); жесткий диск объемом 2 Гб, на котором имеется не менее 650 Мб свободного места. Операционная система Windows 2000 Professional поддерживает однопроцессорные и двухпроцессорные системы.

6.7 Операционная система Windows XP

Корпорацией Microsoft представлено три выпуска операционной системы Windows XP:

1) *Windows XP Professional* – операционная система, предназначенная для корпоративных пользователей и обеспечивающая высокий уровень масштабируемости и надежности;

2) *Windows XP Home Edition* – эффективная платформа для работы с цифровыми мультимедийными материалами, являющаяся наиболее удачным выбором для пользователей домашних компьютеров и любителей компьютерных игр;

3) *Windows XP 64-Bit Edition* – 64-разрядная операционная система, способная удовлетворить самых требовательных пользователей, обладающих специальной технической подготовкой.

Приведем основные характеристики Windows XP [17]:

- основу системы Windows XP составляет код Windows NT и Windows 2000, характеризуемый 32-разрядной вычислительной архитектурой и полностью защищенной моделью памяти;
- средство проверки драйверов устройств в ОС Windows XP, созданное на основе аналогичного средства системы Windows 2000 и обеспечивающее более тщательное испытание драйверов;
- доступность критически важных структур ядра системы только для чтения, благодаря чему драйверы и приложения не могут повредить их. Код драйверов устройств также доступен только для чтения и снабжен защитой на уровне страниц;
- наличие механизма, позволяющего устанавливать и использовать одновременно несколько версий компонентов системы Windows;
- возможность безопасной передачи данных через Интернет с помощью системы IP-безопасности;
- обновленный внешний вид при сохранении ядра Windows 2000, предоставляющий возможность объединения и упрощения типичных задач; добавления новых визуальных подсказок, помогающих пользователю в работе с компьютером; возможность смены обновленного пользовательского интерфейса на классический интерфейс Windows 2000 одним нажатием кнопки администратором или пользователем системы;
- возможность настройки и оптимизации многочисленных функций операционной системы Windows XP, а также устранения неполадок.

К основным возможностям **Windows XP Home Edition** относятся:

- новое внешнее оформление;
- работа с цифровыми изображениями и обмен ими;
- наличие многофункционального средства работы с музыкальными материалами, обеспечивающее поиск, загрузку, хранение и воспроизведение цифровых музыкальных файлов;
- наличие всех необходимых средств для создания и просмотра видеоматериалов на своем ПК и обмена ими;
- простота организации общего доступа к компьютеру и создания домашней сети;

- наличие эффективных средств связи для передачи мгновенных сообщений, проведения голосовых и видеоконференций, а также организации совместной работы;
- наличие средств для решения возникающих проблем и получения помощи специалистов.

Windows XP Professional обладает всеми возможностями, присущими Windows XP Home Edition.

Остановимся на дополнительных преимуществах данной ОС:

- более высокий уровень безопасности, возможность шифрования файлов и папок с целью защиты корпоративной информации;
- поддержка мобильных устройств для обеспечения возможности работы в автономном режиме или подключения к компьютеру в удаленном режиме;
- встроенная поддержка высокопроизводительных многопроцессорных систем;
- возможность работы с серверами Microsoft Windows Server и системами управления предприятиями;
- эффективное взаимодействие с другими пользователями по всему миру благодаря возможностям многоязычной поддержки.

Требования к оборудованию состоят в следующем:

- компьютер с тактовой частотой процессора не менее 300 МГц; допустимый минимум – 233 МГц (система с одним или двумя процессорами); могут использоваться процессоры семейств Intel Penti-um/Celeron, AMD K6/Athlon/Duron или другие совместимые процессоры;
- рекомендуется не менее 128 МБб ОЗУ (допустимый минимум – 64 МБ, при этом быстродействие и некоторые возможности операционной системы могут быть ограничены);
- 1,5 Гб свободного места на жестком диске;
- видеоплата и монитор Super VGA с разрешением не менее 800×600 точек. Дисковод для компакт-дисков или дисков DVD;
- клавиатура и мышь Microsoft Mouse или совместимое устройство ввода.

В **Windows XP 64-Bit Edition** заложены следующие возможности:

- высокий уровень быстродействия и масштабируемости, позволяющий решать сложные задачи технического характера, наиболее требовательные к ресурсам компьютера;
- организация работы ОС на основе семейства процессоров Intel Itanium, благодаря чему обеспечивается поддержка дополнительного объема памяти, увеличивается скорость операций ввода-вывода, расширяются возможности для выполнения вычислений переменных с плавающей точкой;
- эффективная платформа для создания современных цифровых материалов;
- мощная компьютерная платформа для выполнения технических и аналитических разработок;
- отличная платформа для проведения финансового и статистического анализа.

6.8 Windows 2003 Server

Windows 2003 Server, созданный на основе доказавших свою надежность продуктов семейства Windows 2000 Server, осуществляет интеграцию многофункциональной среды выполнения приложений в целях создания современных веб-служб и подготовки бизнес-решений, позволяющих значительно повысить эффективность процессов обработки информации. При использовании платформы Windows 2003 Server разработчикам предоставляются следующие преимущества [17]:

- создание более эффективных приложений благодаря следующим возможностям:
  - собственной поддержке веб-служб XML на основе использования стандартов, таких как SOAP[13], WSDL[14] и UDDI[15],

[13] SOAP (Simple Object Access Protocol) – стандарт SOAP, описывающий протокол, предназначенный для обмена структурированной информацией в распределенных системах, таких как интернет.

[14] WSDL (Web Services Description Language) – язык для описания Интернет-сервисов.

в результате чего расширяются возможности для обеспечения взаимодействия приложений друг с другом;

- обширному набору интегрированных служб распределенных приложений, оптимизированных по быстродействию и масштабируемости и включающих новые усовершенствования, связанные с развертыванием, управлением и обеспечением безопасности;
- встроенным средствам обеспечения надежности на основе поддержки архитектуры слабо связанных и тесно связанных компонентов;
- тесной интеграции с серверами Microsoft 2003 Enterprise Server.

• сокращение времени разработки благодаря следующим возможностям:

- предоставлению интегрированного набора служб;
- собственной поддержке веб-служб XML (SOAP, WSDL, UDDI);
- использованию управляемого кода и других возможностей Microsoft Visual Studio 2003, что позволяет разработчикам сократить время программирования;
- интеграции со средой разработки Visual Studio 2003;
- использованию единого языка программирования;
- возможностям использования существующего оборудования независимо от применяемого языка программирования.

Семейство Microsoft Windows 2003 Server включает четыре продукта:

1) **Windows 2003 Web Server –** продукт, представляющий собой веб-сервер, в котором наибольшее внимание уделяется улучшению функциональных возможностей. Он разрабатывался с целью предоставления компаниям многофункциональной и устойчивой платформы для обслуживания и размещения веб-приложений, обеспечивающих при этом удобство развертывания

---

[15] UDDI (Universal Description, Discovery and Integration) – универсальный метод описания, обнаружения и интеграции web-сервисов для систем электронной коммерции. Бизнес-регистр UDDI представляет собой базу данных общего пользования, в которой компании сами себя регистрируют.

и управления. Благодаря использованию новаторской технологии Microsoft ASP2003 – одной из составляющих среды 2003 Framework – сервер Windows 2003 Web-Server предоставляет разработчикам платформу для быстрого создания и развертывания веб-служб XML и веб-приложений;

2) **Windows 2003 Standard Server – п**родукт, представляющий собой надежную операционную систему для сети, обеспечивающую быструю и простую реализацию бизнес-решений. Эта гибкая серверная ОС прекрасно подходит для удовлетворения повседневных требований как крупных, так и малых организаций. В операционной системе Windows 2003 Standard Server предоставляются технологии совместного использования файлов и принтеров, безопасного подключения к Интернету, централизованного развертывания приложений для настольных компьютеров и организации эффективной совместной работы между сотрудниками, партнерами и заказчиками. В Windows 2003 Standard Server реализована поддержка до 4 Гб оперативной памяти и симметричной многопроцессорной обработки с использованием двух процессоров;

3) **Windows 2003 Enterprise Server** – продукт, предназначенный для компаний среднего и крупного размера. В нем реализованы функциональные возможности, необходимые для поддержки инфраструктуры организации, бизнес-приложений и транзакций электронной коммерции. Windows 2003 Enterprise Server – операционная система, обладающая полным набором функциональных возможностей, обеспечивающая поддержку до восьми процессоров и предоставляющая возможности корпоративного уровня, такие как создание и поддержка класстеров, состоящих из четырех узлов, и организация оперативной памяти, достигающей объема в 32 ГБ. Данная система доступна также для 64-разрядных вычислительных платформ;

4) **Windows 2003 Datacenter Server –** продукт, предназначенный для компаний, предъявляющих высокие требования к масштабируемости и доступности. Предоставляет надежную основу для создания критически важных технических решений, обеспечивающих поддержку баз данных, программ планирования ресурсов предприятия ERP (En-terprise Resource Planning), обработку сложных транзакций в режиме реального времени

и консолидацию серверов. Это – самая эффективная и мощная серверная операционная система из всех, когда-либо разрабатывавшихся корпорацией Microsoft. В ней реализована поддержка симметричной многопроцессорной обработки с использованием до 32 процессоров. В качестве стандартных функций предоставляются службы балансировки нагрузки и создания кластеров, состоящих из восьми узлов. Операционная система Windows 2003 Datacenter Server доступна также для 64-разрядных вычислительных платформ.

Windows 2003 Server основывается на достоинствах, унаследованных от платформы Microsoft Windows NT. Продукты семейства Windows 2003 Server полноценно взаимоувязаны с серверами, принадлежащими семейству Windows 2000. Компании, которые в настоящее время уже развернули или осуществляют развертывание продуктов семейства Windows 2000 Server, смогут легко выполнить обновление до уровня Windows 2003 Server, что сразу же принесет им существенный выигрыш, обусловленный улучшением управляемости, повышением надежности, безопасности и быстродействия, а также возможностью использования интегрированных веб-служб XML.

Требования к оборудованию различных версий Windows 2003 Server приведены в таблице.

Требование	Windows 2003 Web-сервер	Windows 2003 Standard Server	Windows 2003 Enterprise Server	Windows 2003 Datacenter Server
Минимальная тактовая частота процессора	133 МГц	133 МГц	133 МГц для компьютеров с процессором x86 733 МГц для компьютеров с процессором Itanium	400 МГц для компьютеров с процессором x86 733 МГц для компьютеров с процессором Itanium
Рекомендуемая тактовая частота процессора	550 МГц	550 МГц	733 МГц	733 МГц

Требование	Windows 2003 Web-сервер	Windows 2003 Standard Server	Windows 2003 Enterprise Server	Windows 2003 Datacenter Server
Мини-мальный объем ОЗУ	128 Мб	128 Мб	128 Мб	512 Мб
Рекомен-дуемый минималь-ный объем ОЗУ	256 Мб	256 Мб	256 Мб	1 Гб
Макси-мальный объем ОЗУ	2 Гб	4 Гб	32 Гб для компьютеров с процессором x86 64 Гб для компьютеров с процессором Itanium	64 Гб для компьютеров с процессором x86 128 Гб для компьютеров с процессором Itanium
Поддержка нескольких процессоров	1 или 2	1 или 2	До 8	Не менее 8 Не более 32
Место на диске, необходимое для уста-новки	1,5 Гб	1,5 Гб	1,5 Гб для компьютеров с процессором x86 2,0 Гб для компьютеров с процессором Itanium	1,5 Гб для компьютеров с процессором x86 2,0 Гб для компьютеров с процессором Itanium

Вопросы для самопроверки

1. Назовите основные характеристики ОС MS DOS.
2. Перечислите основные части ОС MS DOS.
3. Что представляет собой BIOS?
4. В чем состоит назначение BootRecord?
5. Опишите ядро ОС MS DOS.
6. Приведите последовательность загрузки ОС MS DOS.
7. Опишите файл конфигурации MS DOS CONFIG.SYS и приведите основные команды конфигурирования. Поясните на-

значение файлов пакетной обработки и особенности командного файла автозапуска AUTOEXEC.BAT.

8. Перечислите основные функции командного процессора. Раскройте принцип работы командного процессора при обработке внутренних и внешних команд ОС MS DOS.

9. В какой версии Windows появилась поддержка технологии «Plug and Play»?

10. Что такое Windows API и что он позволяет?

11. Поддержка потоков, что это?

12. Как вы понимаете вытеснение задач в ОС Windows?

13. Что используется для хранения данных о конфигурации системы в ОС Windows?

14. В какой версии ОС Windows появилась поддержка длинных имен файлов?

15. Что такое OLE2 и «Drag and Drop»?

16. С какой версии ОС Windows появилась поддержка шины USB?

17. В какой версии ОС Windows появилась поддержка Universal Plug and Play?

18. На каких процессорах могут работать ОС Windows 9x и ОС Windows NT?

19. В чем отличие Windows NT Server от Windows NT Workstation?

20. В каких ОС семейства Windows можно вводить дисковые квоты и осуществлять поддержку массива RAID 5?

21. Приведите отличительные особенности версий Windows XP.

22. Из каких продуктов состоит семейство Microsoft Windows 2003 Server?

7. СЕМЕЙСТВО ОПЕРАЦИОННЫХ СИСТЕМ OS/2 WARP

7.1 Общее представление OS/2 Warp

Семейство операционных систем OS/2 Warp, созданных фирмой IBM, является одним из самых лучших ОС для ПК по очень большому числу параметров. Эти операционные системы появились раньше своих основных конкурентных систем, но тем не менее они не смогли стать самыми распространенными. Основная причина сложившейся ситуации заключается в отсутствии широкой рекламы и системы продвижения этого продукта на рынок, хотя качество операционной сисстемы было досточно высоким [2].

Во-первых, компания IBM не сочла необходимым продвигать свою ОС на рынок программного обеспечения, ориентированный на конечного пользователя, а решила продолжить свою практику работы исключительно с корпоративными клиентами. Рынок корпоративного ПО оказался существенно уже для ПК, чем рынок ПО для конечного пользователя, ибо компьютеры типа IBM PC, прежде всего, являются персональными.

Во-вторых, основные доходы компания IBM получала не от продажи системного ПО для ПК, а за счет продаж дорогостоящих серверов и другого оборудования. Доходы от продажи своей ОС не представлялись руководству компании IBM значимыми [2]. Для успеха на рынке ОС для ПК необходимо было обеспечить всестороннюю поддержку своей системы соответствующей учебной литературой, широкой рекламой, заинтересовать разработчиков программного обеспечения. Этого не произошло, и сегодня уже практически мало кто знает о системах OS/2. В то же время следует отметить, что фирмы, которые в свое время освоили эту систему и создали для нее соответствующее ПО, до сих пор не переходят на ныне чрезвычайно популярные ОС Windows NT, поскольку последние требуют существенно больше системных ресурсов и при этом функционируют медленнее.

Семейство 32-разрядных ОС для IBM-совместимых компьютеров начало свою историю с появления первой OS/2 версии

2.0 в 1992 году. Сейчас мы, как правило, имеем дело с четвертой версией ОС этого семейства. Все ОС данного семейства в своем названии имеют слово Warp, что переводится с английского как «основа». OS/2 Warp 4.0 практически представляет собой OS/2 Warp 3.0, вышедшую еще в 1994 году, с несколько улучшенными параметрами для DOS-задач и обновленными элементами объектно-ориентированного интерфейса. Для операционной системы OS/2 Warp 4.0 характерны:

- вытесняющая многозадачность (preemptive multitasking) и поддержка DOS- и Windows-приложений;
- интуитивно понятный и действительно удобный объектный пользовательский интерфейс;
- поддержка стандарта открытого объектного документооборота OpenDoc;
- поддержка стандарта OpenGL;
- поддержка и встроенная разработка на языке Java;
- поддержка шрифтов True Type (TTF);
- управление голосом без предварительной подготовки (технология Voice Type);
- полная поддержка глобальных сетей Интернет и технологии Интранет, доступ в CompuServe (американская почтовая служба);
- средства построения одноранговых сетей и клиентские части для IBM LAN Server, Windows, Lantastic, Novell Netware 4.1, в том числе поддержка службы каталогов;
- наличие системы удаленного доступа через модемные соединения;
- Mobile File System для поддержки мобильных пользователей;
- стандарт авторасознавания аппаратных устройств Plug-and-Play;
- набор офисных приложений (базы данных, электронные таблицы, текстовый процессор, генератор отчетов, деловая графика, встроенная система приема/передачи факсимильных сообщений, информационный менеджер);
- полная MultiMedia-поддержка, включающая систему работы с видеокамерой, расширенную систему помощи WarpGuide.

Однако наиболее заманчивы не перечисленные из рекламного буклета возможности системы, а удобная и надежная среда при работе с базами данных, возможность работы в сетях, организованной как клиентское рабочее место при взаимодействии с большими системами.

OS/2 Warp предлагает единый интерфейс для программирования прикладных программ (API), совместимый с рядом операционных систем, что позволяет снизить стоимость разработок. Все версии OS/2 и LAN Server, включая версии OS/2 Warp и OS/2 Warp Server 4.5, совместимы по восходящей линии, что позволяет экономить средства, необходимые для поддержания уже существующих прикладных программ.

Чрезвычайно важным для пользователей является тот факт, что компания IBM для всех версий своей ОС регулярно выпускает пакеты обновления (FixPak). Эти пакеты исправляют обнаруженные ошибки, а также вносят новые функции. Для пользователей такая практика сопровождения фирмой своей ОС более выгодна нежели используемая компанией Microsoft практика частого выпуска новых версий ОС, в которых обещается исправление обнаруженных ранее недостатков и появление новых функций, поскольку требуются значительные капиталовложения не только для приобретения новой системы, но и для ее освоения.

Так, например, для версии одной из своих самых удачных операционных систем Windows компания Microsoft выпустила всего только 6 пакетов обновления (ServicePak), тогда как для OS/2 Warp 3.0, которая вышла в свет в 1994 году, компания IBM выпустила уже несколько десятков FixPak. Для OS/2 Warp 4.0 вышло 15 FixPax.

Очень полезным как для управления приложениями, так и для создания несложных собственных программ является наличие системы программирования на языке высокого уровня Rexx, который иногда называют языком процедур. Можно сказать, что это встроенный командный язык, служащий для тех же целей, что и язык для пакетных (batch) файлов в среде DOS, но он обладает несравнимо большими возможностями. Это язык высокого уровня с нетипизированными переменными. Язык легко расширяем, любая программа OS/2 может добавлять в него новые

функции. Помимо встроенного интерпретатора с языка Rexx имеется система программирования Visual Rexx. Есть и объектно-ориентированная версия языка Rexx с соответствующим интерпретатором.

Наиболее сильное впечатление, которое можно получить при работе в OS/2, оставляет объектно-ориентированный графический пользовательский интерфейс, а особой популярностью у программистов эта система пользовалась вследствие достаточно хорошей организации виртуальных машин и высокого быстродействия при выполнении обычных DOS-приложений.

7.2 Особенности архитектуры OS/2 Warp

В OS/2 имеется несколько видов виртуальных машин для выполнения прикладных программ. **Собственные 32- и 16-разрядные программы OS/2 выполняются на отдельных виртуальных машинах в режиме вытесняющей многозадачности и взаимодействуют между собой с помощью средств DDE[16] OS/2.** Прикладные программы DOS и Win16 могут запускаться на отдельных виртуальных машинах в многозадачном режиме. При этом они поддерживают полноценные связи DDE и OLE 2.0 друг с другом и связи DDE с 32-разрядными программами OS/2. Кроме того, при желании можно запустить несколько программ Win16 на общей виртуальной машине Win16, где они работают в режиме невытесняющей многозадачности, как это реализовано в Windows 3.x.

Разнообразные сервисные функции API OS/2, в том числе модель системных объектов SOM (System Object Model), обеспечиваются с помощью системных динамических библиотек DLL, к которым можно обращаться без требующих затрат времени переходов между кольцами защиты. Ядро OS/2 предоставляет многие базовые сервисные функции API, обеспечивает поддержку файловой системы, управление памятью и имеет диспетчер аппаратных прерываний. **В ядре виртуальных DOS-**

[16] DDE (Dynamic Data Exchange) – универсальные механизмы динамического обмена данными. Используются разработчиками в качестве средства интеграции компонентов ПО.

машин (VDM-ядре) осуществляется эмуляция DOS и процессора 8086, а также управление VDM. Драйверы виртуальных устройств обеспечивают уровень аппаратной абстракции. Драйверы физических устройств напрямую взаимодействуют с аппаратурой.

Модуль реализации механизмов виртуальной памяти в ядре OS/2 поддерживает большие, постраничные, разбросанные адресные пространства, составленные из объектов памяти. **Каждый объект памяти управляется так называемым «пейджером» – задачей вне ядра, обеспечивающей резервное хранение страниц объекта памяти.** Адресные пространства управляются отображением или размещением объектов памяти внутри них. Ядро управляет защитой памяти и ее распределением на основе объектов памяти абстрактным образом вне зависимости от каких-либо конкретных аппаратных средств трансляции процессорных адресов. В частности, ядро интенсивно использует режим копирования при записи для придания программам способности делить объекты памяти без копирования большого числа страниц, когда новое адресное пространство получает доступ к объекту памяти. Новые копии страниц создаются только тогда, когда программа в одном из адресных пространств обновляет их. Когда ядро принимает страничный сбой в объекте памяти и не имеет страницы памяти в наличии, или когда оно должно удалить страницы из памяти по требованию других программ, работающих в машине, оно с помощью механизма межпроцессного взаимодействия уведомляет пейджер об объекте памяти, в котором произошел сбой. Теперь дело пейджера сервера приложений определить, каким образом предоставить или сохранить данные. Это позволяет системе устанавливать различную семантику для объектов памяти, основываясь на потребностях программ, которые их используют.

Ядро управляет средами исполнения для программ, обеспечивающих выполнение множественных заданий и потоков. Каждое задание имеет свое собственное адресное пространство или отображение. Оно назначает объекты памяти, которые задание отобразило на диапазон адресов внутри адресного пространства. Задание также является блоком размещения ресурсов и защиты, при этом заданиям придаются возможности

и права доступа к средствам межпроцессного взаимодействия системы. Для поддержки параллельного исполнения с другой программой в пределах одного адресного пространства ядро отделяет среду исполнения от действительно идущего потока инструкций**. Потоки вычислений, включая процессорные ресурсы, необходимые для их поддержки, называются потоками.** Таким образом, программа может быть загружена в задание и может быть исполнена в нескольких различных местах в коде в одно и то же время на мультипроцессоре или параллельной машине. Это приводит к повышению быстродействия приложения.

Система межпроцессного взаимодействия обеспечивает базовый механизм, позволяющий потокам работать в различных заданиях для связи друг с другом. Система межпроцессного взаимодействия поддерживает надежную доставку сообщений на порты. **Порты представляют собой защищенные каналы между заданиями.** Каждому заданию, использующему порт, приписывается набор прав на этот порт. Права могут быть различными для разных заданий. Только одно задание может получать данные по какому-либо порту, хотя любой поток внутри задания может выполнять операцию приема. Одно или более заданий могут иметь права передачи в порт. Ядро позволяет заданиям применять систему межпроцессного взаимодействия на передачу друг другу прав на порт. Вместо того чтобы копировать данные, сообщение содержит указатель на них, он называется указателем на данные вне линии. Когда ядро передает сообщение от передатчика к приемнику, оно заставляет передаваемую память появиться в адресном пространстве приемника и, как вариант, исчезнуть из адресного пространства передатчика. Ядро само структурировано как задание с потоками, и большинство системных сервисов реализованы как механизмы межпроцессного взаимодействия к ядру, а не как прямые системные вызовы.

7.3 Особенности интерфейса OS/2 Warp

В OS/2 Warp в качестве стандартной графической оболочки используется среда WPS (Workplace Shell), организованная более логично и удобно, чем известный Windows-

интерфейс. **Оболочка Workplace Shell основана на мощной системно-объектной модели SOM IBM-технологии,** специально разработанной для решения таких проблем, как жесткая привязка объектов к их клиентам и необходимость использования одного и того же языка программирования. Объекты Workplace Shell работают в среде SOM, доступ в которую можно реализовать почти на всех языках программирования, предусматривающих внешние процедуры, в том числе и на Rexx.

В отличие от GUI Windows, в которой ярлыки объектов никак не связаны между собой, в WPS объекты, имеющие аналогичные ярлыки (shadow в терминологии WPS), просто имеют дополнительные свойства быть многократно отображенными почти как самостоятельные объекты. Можно сделать несколько shadow-значков с уже существующей shadow-значков или объекта. При этом любые shadow-значки могут быть перемещены в любое место, и их связи с основным объектом не теряются. Аналогично и в GUI Windows. Но в WPS можно переместить основной объект, и его shadow-значки тоже изменят свои параметры, тогда как в GUI Windows произойдет разрушение связей, поскольку связи являются односторонними.

Про SOM можно сказать, что это не связанная ни с одним конкретным языком объектно-ориентированная технология для создания, хранения и использования двоичных библиотек классов. Ключевые слова здесь «двоичные» и «не связанная ни с одним конкретным языком». Хотя теперь многие считают OS/2 технологией прошлого, модель SOM на самом деле представляет собой одну из наиболее интересных разработок в области компьютерной индустрии даже на сегодняшний день. По существу, некоторые идеи, реализованные в OS/2 в начале 90-х годов прошлого столетия, сейчас только обещают быть реализованными в новом поколении ОС Windows с кодовым названием Whistler. Объектно-ориентированное программирование (ООП) заслужило безоговорочное признание в качестве основной парадигмы, однако его применению в коммерческом программном обеспечении препятствуют отсутствие в языках ООП средств для обращения к библиотекам классов, подготовленным на других языках, и необходимость поставлять с библиотеками классов исходные тексты. Многим независимым разработчикам

библиотек классов приходится продавать заказчикам исходные тексты, поскольку разные компиляторы по-разному отображают объекты. Настоящий потенциал SOM заключается в ее совместимости практически с любой платформой и любым языком программирования. **SOM соответствует спецификации CORBA** (Common Object Request Broker Architecture) – архитектуре посредника стандартного объектного запроса, которая определяет стандарт условий взаимодействия между прикладными программами в неоднородной сети.

Интересно отметить тот факт, что существует довольно много альтернативных оболочек для OS/2, начиная с FileBar, примитивной, но зато отлично работающей на компьютерах с 4 Мбайт памяти, и кончая мощной Object Desktop, которая значительно улучшает внешний вид экрана OS/2 и делает работу с системой более удобной.

Помимо оболочек, улучшающих интерфейс OS/2, имеется также ряд программ, расширяющих ее функциональность. Это, прежде всего, **Xfree86 для OS/2** – полноценная система X Window, которая может использоваться как X-терминал при работе в сети с UNIX-машинами, а также для запуска программ, перенесенных из UNIX в OS/2. К сожалению, таких программ немного, однако большое количество UNIX-программ поставляется вместе с исходными кодами, которые, как правило, практически не нужно изменять для перекомпиляции под Xfree86/OS2.

7.4 Серверная операционная система OS/2 Warp 4.5

Серверная операционная система компании IBM, выпущенная в 1999 году, носит название OS/2 WarpServer for e-Business, что подчеркивает ее основное назначение, но поскольку в процессе ее создания она носила кодовое название «Аврора» (Aurora), фактически все ее так теперь и называют.

Как известно, предыдущие версии системы OS/2 могли предоставить программисту только 512 Мбайт виртуального адресного пространства для 32-битовых задач. В свое время это было очень много. Однако сегодня, хотя и крайне редко, еще встречаются задачи, требующие столь большого объема оперативной памяти. Некоторые считают серьезным недостатком ог-

раничение в 512 Мбайт. Поэтому в последней версии системы это ограничение снято, и теперь объем виртуальной памяти может достигать 3 Гбайт (напомним, что в Windows NT 4.0 объем виртуального адресного пространства для задач пользователя составляет 2 Гбайт).

В этой системе разработчики постарались все прежние остатки старого 16-битового кода, который еще оставался в предыдущих версиях системы, заменить на полностью 32-битовые реализации, что повышает скорость работы системы. Прежде всего, сделана поддержка 32-битовых драйверов инсталлируемых файловых систем (IFS), ибо в предыдущих системах работа с ними велась через трансляцию вызовов 32bit–>–l6bit–>32bit. В то же время для обеспечения совместимости со старым программным обеспечением кроме 32-битового используется и 16-битовый API.

Для повышения надежности файловой подсистемы создана **новая журналирующая файловая система JFS** (Journaling File System). JFS введена для удовлетворения потребности в более живучей файловой системе для OS/2 Warp. JFS имеет большую безопасность в структурах данных благодаря технике, разработанной для СУБД. Работа с файловой системой происходит в режиме транзакций с ведением журнала транзакций. В случае системных сбоев имеется возможность обработки журнала транзакций, позволяющая производить занесение или сброс каких-либо изменений, произшедших во время системного сбоя. Эта система также повышает скорость восстановления файловой системы после сбоя. Сохраняя целостность файловой системы, эта система управления файлами не гарантирует восстановление пользовательских данных. Следует отметить, что файловая система JFS обеспечивает самую высокую скорость работы с файлами из всех известных систем, созданных для ПК, что очень важно для серверной ОС.

Для работы с дисками создан специальный менеджер дисков – LVM (Logical Volume Manager). Все устанавливаемые файловые системы содержатся в LVM. LVM осуществляет определение имен дисков для программ, которые этого требуют. Это позволяет избирательно назначить любую букву любому разделу. И даже больше – ОС не будет сама использовать имена

дисков. LVM в совокупности с JFS позволяет объединять несколько томов и далее несколько физических дисков в один большой логический том.

Вопросы для самопроверки

1. В каком году появилась 32-разрядная ОС от фирмы IBM? Как она называется? Расскажите о ее основных возможностях.
2. Что такое VDM-ядро и DMA в ОС OS/2 Warp?
3. Как называется стандартная графическая оболочка в ОС OS/2 Warp и каковы ее возможности?
4. Как называется новая файловая система в ОС OS/2 Warp 4.5 и в чем ее преимущества?

8. ОПЕРАЦИОННЫЕ СИСТЕМЫ СЕМЕЙСТВА UNIX

8.1 Общее представление семейства ОС UNIX

ОС UNIX[17] является удачной реализацией многопользовательской и многозадачной операционной системы. Она спроектирована как инструментальная система для разработки программного обеспечения. Система UNIX обладает простым, но очень мощным командным языком и независимой от устройств файловой системой. Системы и приложения, выполняющиеся в ней, легко переносимы. Компилятор с языка С для всех оттранслированных программ дает рентабельный и разделяемый код, что позволяет эффективно использовать имеющиеся в системе ресурсы. При создании ОС UNIX имелось три цели:

1) стремление сохранить простоту и обойтись минимальным количеством функций. Все реальные сложности оставлялись пользовательским программам;

2) использование общих механизмов во множестве случаев, например при обращении к файлам, прерываниях, именовании и др.;

3) предоставление возможности решать большие задачи, комбинируя более мелкие, а не разрабатывать программы зано-

[17] В 1965 г. подразделение компании AT&T Bell Telephone Laboratories (Bell Labs) совместно с General Electric Company и Массачусетским технологическим институтом начали разрабатывать новую операционную систему, названную **Multics** (Multi-user Timesharing Interactive Computing System*)*. Целью проекта было создание многозадачной ОС, способной обеспечить одновременную работу нескольких сотен пользователей. Но система Multics так и не была завершена, поскольку в 1969 г. Bell Labs вышла из проекта. Однако специалисты Bell Labs, принимавшие участие в проекте, – Кен Томпсон и Дэннис Ритчи – продолжили работу над созданием удобной среды программирования. Используя идеи и разработки, появившиеся в результате работы над Multics, они создали в 1969 году небольшую операционную систему, получившую **имя Unix**, созвучное Multics и придуманное другим членом группы разработчиков, Брайаном Керниганом. Система была целиком написана на ассемблере и применялась на компьютере PDP-7. К 1971 году в Bell Labs Unix была перенесена на более мощный компьютер PDP-11.

во. В системе UNIX имеется возможность направлять выход одной программы на вход другой (программные каналы – pipe).

UNIX-системы поставляются с большим набором системных и прикладных программ, включающим редакторы текстов, программные интерпретаторы командного языка, компиляторы с популярных языков программирования (C, C++, ассемблер, Perl, Fortran и др.), средства сортировки, ведения БД, административные и обслуживающие программы.

8.2 Основные понятия семейства ОС UNIX

Виртуальная машина. Каждому пользователю после входа в систему предоставляется виртуальный компьютер, в котором есть все необходимые ресурсы: процессор, память, устройства, файлы. Текущее состояние виртуального компьютера называется ***образом***, который включает [2]:

- образ памяти;
- значения общих регистров процессора;
- состояния открытых файлов;
- текущую директорию и другую информацию.

Образ процессора во время его выполнения размещается в основной памяти. В старых версиях системы UNIX образ мог быть выгружен на диск, если какому-либо приоритетному процессу потребуется место в основной памяти (прежде всего, выгружаются неиспользуемые страницы).

Образ памяти делится на три логических сегмента:

1) сегмент рентабельных процедур, начинающийся с нулевого адреса в виртуальном адресном пространстве процесса;

2) сегмент данных, располагающийся следом за сегментом процедур и способный расти в сторону больших адресов;

3) сегмент стека, растущий в сторону младших адресов.

Пользователь. Чтобы начать работать с системой, пользователь должен ввести со свободного терминала свое учетное имя (account name) и пароль (password). Для этого пользователь должен быть зарегистрирован в системе. Обычно регистрацию новых пользователей выполняет администратор. Пользователь не может изменить свое учетное имя, хотя может поменять пароль.

Каждому зарегистрированному пользователю соответствует каталог файловой системы, который называется домашним каталогом пользователя. При входе в систему пользователь получает неограниченный доступ к своему домашнему каталогу и всем файлам и каталогам, содержащимся в нем. Потенциально возможен доступ ко всем другим файлам, однако он может быть ограничен, если пользователь не имеет достаточно привилегий.

Интерфейс пользователя. Традиционный способ взаимодействия пользователя с системой UNIX основывается на использовании командных языков[18]. После входа пользователя в систему у него запускается командный интерпретатор. Обычно в системе поддерживается несколько командных интерпретаторов. Общее название для всех командных интерпретаторов ОС UNIX – shell (оболочка), поскольку любой интерпретатор представляет внешнее окружение ядра системы. Вызванный командный интерпретатор выдает приглашение на ввод пользователем командной строки[19], которая может содержать простую команду, конвейер команд или последовательность команд. Командные языки довольно просты, но могут использоваться для написания сложных программ, посредствам использования командных файлов (shell scripts).

Shell является языком программирования, так как имеет:

- переменные;
- управляющие структуры (типа if);
- подпрограммы (в том числе командные файлы);
- возможность передачи параметров;
- механизм обработки прерываний.

Привилегированный пользователь. Центральной частью системы UNIX является ядро (kernel). Ядро идентифицирует каждого пользователя по его идентификатору UID (User Identifier), уникальному целому значению, присвоенному пользователю при регистрации в системе. Кроме того, каждый поль-

---

[18] В связи с большим распространением графических оболочек в системе UNIX все чаще работают с графической оболочкой X-Windows.

[19] Как правило, это значок доллара – $, но может быть и любой другой символ.

зователь относится к группе пользователей, которая идентифицируется некоторым целым значением GID (Group Identifier). Данные идентификаторы сохраняются в учетных файлах системы[20] и приписываются процессу, в котором выполняется командный интерпретатор, запущенный при входе пользователя в систему. Эти значения наследуются каждым новым процессом, запущенным от имени данного пользователя, и используются системой для контроля прав доступа к файлам, выполнению программ и т.д.

Одна запись в файле с информацией о пользователях соответствует одному пользователю и имеет следующие текстовые поля, разделенные символом двоеточия:

- имя пользователя;
- пароль пользователя в закодированном виде;
- целочисленный идентификатор пользователя;
- целочисленный идентификатор группы;
- комментарий, который содержит сведения о месте работы пользователя;
- каталог пользователя;
- интерпретатор команд пользователя.

Пример записи файла с пустым комментарием:

```
mary:KmHu1hE:201:10::/users/mary:/bin/csh
```

При наличии комментария его синтаксис определяется учетными программами.

Одна запись в файле о группах пользователей соответствует одной группе и представляет собой строку текста со следующими полями, разделенными двоеточиями:

- имя группы;
- пароль группы в закодированном виде;
- целочисленный идентификатор группы;
- список имен пользователей группы, разделенных запятыми.

Пример записи файла для группы без пароля:

```
sect2115::10:mary,sas,temp,ges
```

---

[20] Имеются в виду файлы:
/etc/passwd – файл с информацией о пользователях;
/etc/group – файл с информацией о группе.

Администратору системы выделяется нулевое значение UID. Пользователь с таким значением UID называется суперпользователь (superuser) или root. Он имеет неограниченные права на доступ к любому файлу и на выполнение любой программы. На суперпользователя не распространяется ограничение на используемые ресурсы. Он может изменять эти ограничения для других пользователей, но на него эти ограничения не распространяются. Для обычного пользователя выставляются такие ограничения, как максимальный размер файла, максимальное число сегментов разделяемой памяти, максимально допустимое пространство на диске и т.д.

Атрибуты файлов. Каждый файл и каталог имеют владельца – обычно это пользователь, создавший их в первый раз. Владелец может затем назначить защиту файла со стороны трех классов пользователей:

1) собственно владельца;
2) группы пользователей, к которой принадлежит владелец;
3) всех пользователей, имеющих доступ к системе.

Каждый файл имеет 3 вида разрешения на доступ:

1) чтение (r) read – предоставление возможности читать (просматривать) содержимое файла или каталога (читать с ключом «-l» в команде ls);

2) запись (w) write – предоставление возможности менять содержимое файла или каталога (создавать или удалять файлы в каталоге);

3) выполнение (x) execute – предоставление возможности использовать файл как команду UNIX и проводить поиск в каталоге.

Все комбинации трех видов разрешения доступа для трех классов пользователей (9 комбинаций) при наличии всех прав записываются в формате: `rwx rwx rwx или 777`. Первые три параметра относятся к владельцу файла, три последующих – к группе, к которой принадлежит владелец файла, три последних – это права доступа остальных пользователей. Отсутствие права доступа указывается минусом, например: `r--r--r-` или `444` или

```
$ ls -l /bin
-r-xr-xr-x 1bin 1986 Nov.26 12:00 ar
```

. . .

Приведенная команда показывает режимы доступа.

Кроме девяти режимов защиты файла имеется три дополнительных режима, которые имеют смысл только для исполняемых файлов. Эти режимы может установить только суперпользователь, используя команду chmod (change mode).

Специальные режимы соответствуют восьмеричным значениям 4000, 2000 и 1000 первого параметра этой команды:

4000 – бит установки идентификатора пользователя; указывает, что при выполнении программы (файл) идентификатор пользователя устанавливается равным идентификатору владельца этого файла, а не пользователя, запустившего этот файл на исполнение;

2000 – бит установки идентификатора группы; указывает, что на время выполнения программы (файла) идентификатор группы устанавливается равным идентификатору группы, соответствующему групповой принадлежности этого файла;

1000 – бит разделения; указывает, что выполняемая программа (файл) разделяется многими пользователями и свопинговое пространство не сбрасывается, даже если никто программу не использует в данный момент.

Установка обоих битов идентификаторов пользователя и группы позволяет пользователю выполнять такие программы, которые должны иметь полномочия суперпользователя (например, программа mail создает каталог /usr/spool/mail). В противном случае возможности системы, доступные пользователю, были бы значительно ограничены.

Процессы. Процесс в ОС UNIX – программа, выполняемая в собственном виртуальном адресном пространстве. Когда пользователь входит в систему, автоматически создается процесс. Если командный интерпретатор встречает команду, соответствующую командному файлу, то он создает новый процесс и запускает в нем соответствующую программу.

Стандартные файлы. Многие команды работают по умолчанию со стандартными файлами:

Standard Input (S.I.) – стандартный поток ввода;

Standard Output (S.O.) – стандартный поток вывода;

Diagnostic Output (D.O.) – диагностический поток вывода.

Однако есть средства изменения умолчания, т.е. имеется возможность указать другие файлы вместо стандартных. Можно также в качестве команды D.O. использовать команду S.O. Эти средства называются перенаправлением ввода и вывода, при этом используются следующие обозначения: знак «<» – перенаправление ввода; знак «>» – перенаправление вывода. Стандартный вывод одной команды может быть стандартным вводом следующей команды, это обозначается знаком « | » (вертикальная черта).

Команды, которые могут вводить со стандартного ввода и выводить на стандартный вывод, называются **фильтрами**. Большинство команд является фильтрами. Имеются исключения: команда ls не может работать со стандартным вводом или команда lpr не может работать со стандартным выводом.

Режимы переднего и заднего плана. Обычно команды выполняются в режиме переднего плана (foreground), т.е. «пока вы ждете». Однако если во время выполнения некоторой команды вы хотите выполнять другие команды, то эту (первую) команду можно выполнить в режиме заднего плана (background).

Для того чтобы команда выполнялась в режиме заднего плана, необходимо ее закончить знаком &. Система UNIX создает процесс, который выполняется независимо от командного интерпретатора. Ответ 2042 – это идентификатор этого процесса (PID).

Лучше перенаправлять стандартный вывод процесса заднего плана в файл, поскольку имеется опасность совмещения вывода «переднего» и «заднего» плана на экране.

8.3 Выполнение процессов в ОС UNIX

Процесс в ОС UNIX, как уже говорилось ранее, – это программа, выполняемая в собственном виртуальном адресном пространстве. Когда пользователь входит в систему, автоматически создается процесс, в котором выполняется программа командного интерпретатора. Если командному интерпретатору встречается команда, соответствующая выполняемому, он создает новый процесс и запускает в нем соответствующую программу, начиная с функции main. Эта запущенная программа, в

свою очередь, может создать процесс и запустить в нем другую программу (она тоже должна содержать функцию main) и т.д.

Для образования нового процесса и запуска в нем программы используются два системных вызова API [2]:

1) **fork()**

2) **exec(имя_выполняемого_файла).**

Системный вызов fork приводит к созданию нового адресного пространства, состояние которого абсолютно идентично состоянию адресного пространства основного процесса, то есть в нем содержатся те же программы и данные. Для дочернего процесса заводятся копии всех сегментов данных. Сразу после выполнения системного вызова fork основной (родительский) и порожденный процессы являются абсолютно одинаковыми. Чтобы программа могла разобраться, в каком процессе (основном или порожденном) она теперь работает, функция fork возвращает разные значения: 0 в порожденном процессе и целое положительное число (идентификатор порожденного процесса) в основном процессе.

Теперь, если мы хотим запустить новую программу в порожденном процессе, нужно обратиться к ***системному вызову exec***, указав в качестве аргументов вызова имя файла, содержащего новую выполняемую программу, и, возможно, одну или несколько текстовых строк, которые будут переданы в качестве аргументов функции main новой программы. Выполнение системного вызова exec приводит к тому, что в адресное пространство порожденного процесса загружается новая выполняемая программа и запускается с адреса, соответствующего входу в функцию main. Другими словами, это приводит к замене текущего программного сегмента и текущего сегмента данных, которые были унаследованы при выполнении вызова fork, на новые соответствующие сегменты, заданные в файле. Прежние сегменты теряются. Это эффективный метод смены выполняемой процессом программы, но не самого процесса. Файлы, уже открытые до выполнения примитива exec, остаются открытыми после его выполнения.

Рассмотрим пример пользовательской программы, вызываемой как команда shell, которая выполняет в отдельном процессе стандартную команду shell ls и выдает на экран содержимое те-

кущего каталога файлов.

```
    main()
    {if(fork()==0) wait(0); /* родительский
процесс */
    else exec("ls", "ls",0); /* порожденный
процесс */
    }
```

Таким образом, с практической точки зрения процесс в UNIX является объектом, создаваемым в результате выполнения функции fork(). Каждый процесс, за исключением начального (нулевого), порождается в результате запуска другим процессом операции fork(). Каждый процесс имеет одного родителя, но может породить много процессов. Начальный (нулевой) процесс является особенным процессом, который создается в результате загрузки системы. После порождения нового процесса с идентификатором «1» нулевой процесс становится процессом подкачки и реализует механизм виртуальной памяти. Процесс с идентификатором «1», известный под именем init, является предком любого другого процесса в системе и связан с каждым процессом особым образом. Функция fork() создает иерархию процессов.

Процесс может выполняться в одном из двух состояний – ***пользовательском*** или ***системном***. В пользовательском состоянии процесс выполняет пользовательскую программу и имеет доступ к пользовательскому сегменту данных. В системном состоянии процесс выполняет программы ядра и имеет доступ к системному сегменту данных.

Когда требуется выполнить системную функцию, пользовательским процессом создается системный вызов. Фактически происходит вызов ядра системы как подпрограммы. С момента появления системного вызова процесс считается системным. Таким образом, пользовательский и системный процессы являются двумя фазами одного и того же процесса, но никогда не пересекаются между собой. Каждая фаза пользуется собственным стеком, содержащим аргументы, локальные переменные и другую информацию относительно функций, выполняемых в режиме задачи. Диспетчерский процесс не имеет пользовательской фазы.

В UNIX-системах используется разделение времени, то есть каждому процессу выделяется квант времени. Процесс либо завершается сам до истечения отведенного ему кванта времени, либо он откладывается по истечении кванта. Механизм диспетчеризации характеризуется достаточно справедливым распределением процессорного времени между всеми процессами. Пользовательским процессам приписываются приоритеты в зависимости от количества получаемого ими процессорного времени. Процессам, которые получили большое количество процессорного времени, назначают более низкие приоритеты, в то время как процессам, которые получили лишь небольшое количество процессорного времени, наоборот – повышают приоритет. Все системные процессы имеют более высокие приоритеты по сравнению с пользовательскими и поэтому всегда обслуживаются в первую очередь.

8.4 Межпроцессные коммуникации в UNIX

ОС UNIX в своей основе наиболее полно отвечает требованиям технологии «клиент-сервер». Эта универсальная модель служит основой построения любых сколь угодно сложных систем, в том числе и сетевых. Разработчики СУБД, коммуникационных систем, систем электронной почты, банковских систем и других продуктов во всем мире широко используют технологию «клиент-сервер». Для построения программных систем, работающих по принципам модели типа «клиент-сервер», в UNIX существуют следующие механизмы [2]:

- сигналы;
- семафоры;
- программные каналы;
- очереди сообщений;
- сегменты разделяемой памяти;
- вызовы удаленных процедур.

Сигналы. Если рассматривать выполнение процесса в виртуальном компьютере, который предоставляется каждому пользователю, то в такой системе должна существовать система прерываний, отвечающая стандартным требованиям. Система прерываний включает:

- обработку исключительных ситуаций;
- средства обработки внешних и внутренних прерываний;
- средства управления системой прерываний (маскирование и демаскирование).

Всем этим состояниям системы в UNIX отвечает техника сигналов, которая может не только воспринимать и обрабатывать сигналы, но и порождать их и посылать на другие машины (процессы). Сигналы могут быть синхронными, когда инициатор сигнала – сам процесс, и асинхронными, когда инициатор возникновения сигнала – интерактивный пользователь за терминалом. Источником асинхронных сигналов может быть также ядро, когда оно контролирует определенные состояния аппаратуры, рассматриваемые как ошибочные.

Сигналы можно рассматривать как простейшую форму межпроцессного взаимодействия, которое используется для передачи от одного процесса другому или от ядра ОС какому-либо процессу уведомления о возникновении определенного события.

Семафоры. Механизм семафоров, реализованный в ОС UNIX, является обобщением классического механизма семафоров общего вида, предложенного профессором Дейкстрой. Семафор представляет собой обрабатываемый ядром целочисленный объект, для которого определены следующие элементарные (неделимые) операции:

- инициализация семафора, в результате которой семафору присваивается неотрицательное значение;
- операция типа P, уменьшающая значение семафора. Если значение семафора опускается ниже нулевой отметки, выполняющий операцию процесс приостанавливает свою работу;
- операция типа V, увеличивающая значение семафора. Если значение семафора в результате операции становится больше или равно 0, один из процессов, приостановленных во время выполнения операции P, выходит из состояния приостановки;
- условная операция типа P, сокращенно CP (conditional P), уменьшающая значение семафора и возвращающая логическое значение «истина» в том случае, когда значение семафора остается положительным. Если в результате операции значение семафора должно стать отрицательным или нулевым, никаких

действий над ним не производится и операция возвращает логическое значение «ложь».

Программные каналы. Теоретически программный канал позволяет взаимодействовать любому числу процессов по принципу FIFO (First-In-First-Out). Процесс, читающий из программного канала, прочитает самые давние записанные в программный канал данные. В современных версиях ОС UNIX для реализации программных каналов применяются, как правило, очереди сообщений.

В UNIX различают два вида программных каналов – именованные и неименованные. Именованный программный канал может служить для общения и синхронизации произвольных процессов, знающих имя данного программного канала и имеющих соответствующие права доступа. Неименованным программным каналом могут пользоваться только создавший его процесс и его потомки.

Очереди сообщений. Для обеспечения возможности обмена сообщениями между процессами этот механизм поддерживается следующими типами системных вызовов:

- образование новой очереди сообщений или получение дескриптора существующей очереди;
- посылка сообщения (вернее, постановка в указанную очередь сообщений);
- прием сообщения (вернее, выборка сообщения из очереди сообщений);
- выполнение ряда управляющих действий.

Ядро хранит сообщения в виде связного списка (очереди), а дескриптор очереди сообщений является индексом в массиве заголовков очередей сообщений.

Разделяемая память. Для работы с разделяемой памятью используются четыре типа системных вызовов:

- создание нового сегмента разделяемой памяти или нахождение следующего сегмента с тем же ключом;
- подключение сегмента с указанным дескриптором к виртуальной памяти обращающегося процесса;
- отключение от виртуальной памяти ранее подключенного к ней сегмента с указанным виртуальным адресом начала;
- управление разнообразными параметрами, связанными с

существующим сегментом.

После того как сегмент разделяемой памяти подключен к виртуальной памяти процесса, этот процесс может обращаться к соответствующим элементам памяти с использованием обычных машинных команд чтения и записи, не прибегая к использованию дополнительных системных вызовов.

Вызовы удаленных процедур. Во многих случаях взаимодействие процессов носит характер «клиент-сервер». Один из процессов «клиент» запрашивает у другого процесса «сервера» некоторую услугу (сервис) и не продолжает свое выполнение до тех пор, пока эта услуга не будет выполнена, т.е. пока процесс-клиент не получит соответствующие результаты. Семантически такой режим взаимодействия эквивалентен вызову процедуры, отсюда и соответствующее название. ОС UNIX по своей идеологии идеально подходит для того, чтобы быть сетевой операционной системой. И на ее основе можно создавать распределенные системы и организовывать распределенные вычисления. Свойства переносимости позволяют создавать однородные по операциям сети, включающие разнородные компьютеры. Однако остается проблема разного представления данных в компьютерах разной архитектуры. Поэтому одной из основных идей вызовов удаленных процедур является автоматическое обеспечение преобразования форматов данных при взаимодействии процессов, выполняющихся на разнородных компьютерах.

Реализация технологии вызовов удаленных процедур RPC (Remote Procedure Call) достаточно сложна, поскольку этот механизм должен обеспечить работу взаимодействующих процессов, которые находятся на разных компьютерах. Если в случае обращения к процедуре, расположенной на том же компьютере, процесс общается с ней через стек или общие области памяти, то в случае удаленного вызова передача параметров процедуре превращается в передачу запроса по сети. Соответственно и получение результата также осуществляется посредством использования сетевых механизмов.

Удаленный вызов процедур включает следующие шаги:

- процесс-клиент осуществляет локальный вызов процедуры, которую называют «заглушкой» (stub). Задача модуля-заглушки – принять аргументы, преобразовать их в стандартную

форму и сформировать сетевой запрос. Упаковка аргументов и создание сетевого запроса называется сборкой (ling);

- сетевой запрос пересылается на удаленную систему, где соответствующий модуль ожидает такой запрос и при его получении извлекает параметры вызова процедуры (unmarshalling), а затем передает их серверу удаленной процедуры. После выполнения осуществляется обратная передача.

8.5 Операционная система Linux

Linux – это современная POSIX[21]-совместимая и UNIX-подобная операционная система для персональных компьютеров и рабочих станций. Linux – это свободно распространяемая версия UNIX, которая была разработана в начале 90-х годов студентом хельсинкского университета Линусом Торвальдсом (Linus Torvalds) и названа им созвучно собственному имени. Все компоненты системы, включая исходные тексты, распространяются с лицензией на свободное копирование и установку для неограниченного числа пользователей.

Linux был создан с помощью многих UNIX-программистов и энтузиастов из Интернета. К данному проекту добровольно подключились программисты, имеющие достаточный опыт практической работы и соответствующие способности для его развития. В создание операционной системы Linux внесли лепту программисты всего мира.

Изначально Linux создавался как самодельная UNIX-подобная реализация для ПК типа IBM PC с процессором i80386. Однако Linux стал настолько популярен и его на сегодняшний день поддерживает такое большое число компаний, что в настоящее время имеется реализация этой ОС практически для всех типов процессоров и компьютеров на их основе. На

[21] Стандарт института IEEE (Institute Electrical and Electronics Engineers), определяющий набор служб операционной системы. Программы, соответствующие стандарту POSIX, могут быть легко перемещены из одной системы в другую. Стандарт основывался на системных службах UNIX, но создавался таким образом, чтобы его реализация была бы возможной в других операционных системах.

базе ОС Linux создаются и встроенные системы, и суперкомпьютеры. Система поддерживает кластеризацию и большинство современных интерфейсов и технологий.

Linux поддерживает большинство свойств, присущих другим реализациям UNIX, кроме того, дополнительные свойства, которыми не обладает ни одна из реализаций UNIX.

ОС Linux достаточно хорошо совместима с рядом стандартов для UNIX на уровне исходных текстов, включая IEEE POSIX.l, System V и BSD. Такая совместимость учитывалась при его создании. Большинство свободно распространяемых по сети Интернет программ для UNIX может быть откомпилировано для Linux практически без особых изменений. Кроме того, все исходные тексты для Linux, включая ядро, драйверы устройств, библиотеки, пользовательские программы и инструментальные средства, распространяются свободно. Другие специфические внутренние черты Linux включают: контроль работ по стандарту POSIX, используемого оболочками csh и bash; псевдотерминалы pty; поддержку национальных и стандартных клавиатур, динамически загружаемых драйверами клавиатур.

Linux поддерживает различные типы файловых систем для хранения данных. Некоторые файловые системы, такие как файловая система *ext2fs,* были созданы специально для Linux. Поддерживаются также другие типы файловых систем, например Minix-l и Xenix. Реализована также система управления файлами на основе FAT и FAT32, позволяющая непосредственно обращаться к файлам, находящимся в разделах с этой файловой системой. Поддерживается и файловая система ISO 9660 CD-ROM для работы с дисками CD-ROM. Имеются системы управления файлами на томах с HPFS и NTFS, правда, они работают только на чтение файлов.

Linux, как и все UNIX-системы, обеспечивает полный набор протоколов TCP/IP для сетевой работы. Поддерживается весь спектр клиентов и услуг TCP/IP, таких как FTP, telnet, NNTP и SMTP. Достаточно часто на компьютерах, работающих под управлением Linux, реализуют DNS-сервер, WWW-сервера (Apache), файерволы для защиты локальных сетей при работе в Интернет, почтовые серверы, сервер DHCP.

Ядро Linux сразу было создано с учетом возможностей защищенного режима процессоров Intel 80386 и 80486. В частности, Linux использует парадигму описания памяти в защищенном режиме и другие новые свойства процессоров. В настоящее время имеются ядра для этой системы, оптимизированные для работы с процессорами Intel и AMD последнего поколения, хотя основные архитектурные особенности защищенного режима работы изменились мало.

Ядро также поддерживает универсальный пул памяти для пользовательских программ и дискового кэша. При этом для кэширования может использоваться вся свободная память, и наоборот, кэш уменьшается при работе больших программ. Этот механизм агрессивного кэширования позволяет увеличить производительность системы.

Выполняемые программы используют динамически связываемые библиотеки, то есть выполняемые программы могут совместно использовать библиотечную программу, представленную одним физическим файлом на диске. Это позволяет выполняемым файлам занимать меньше места на диске, особенно тем, которые многократно используют библиотечные функции. Есть также статические связываемые библиотеки для тех, кто желает пользоваться отладкой на уровне объектных кодов или иметь полные выполняемые программы, которые не нуждаются в разделяемых библиотеках. В Linux разделяемые библиотеки динамически связываются во время выполнения, позволяя программисту заменять библиотечные модули своими собственными.

Вопросы для самопроверки

1. Какие цели имелись при создании ОС UNIX?
2. Перечислите и поясните основные понятия системы UNIX.
3. Изложите основные архитектурные особенности ОС UNIX.
4. Из чего состоит образ памяти в ОС UNIX?
5. Что понимают под понятием Shell?
6. Что понимают под понятием Kernel?
7. Что понимают под понятием UID и GID?

8. Какие атрибуты файлов существуют в ОС UNIX?

9. Что понимают под процессом в ОС UNIX?

10. Расскажите о стандартных потоках и средствах их перенаправления в ОС UNIX. Что понимают под термином «фильтры»?

11. Расскажите о режимах переднего и заднего плана. Какие команды позволяют управлять процессами на заднем плане?

12. Что такое UNIX Reference Manual?

13. Расскажите о структуре файловой системы UNIX. Назовите ее основные компоненты.

14. Расскажите о процессе монтирования файловых систем.

15. Расскажите о системных вызовах Fork и Exec.

16. Расскажите о роли процесса под именем Init.

17. Какие состояния существуют у процессов в ОС UNIX?

18. Какие механизмы межпроцессных коммуникаций в ОС UNIX Вы знаете?

19. Что представляет собой ОС Linux? Какому стандарту она отвечает? На какой процессор была изначально рассчитана?

20. Как называется WWW-сервер для ОС Linux?

21. Какая файловая система была разработана специально для ОС Linux?

9. ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ. ОПЕРАЦИОННАЯ СИСТЕМА QNX

9.1 Общее представление ОС реального времени QNX

Мощной операционной системой реального времени (ОСРВ) [2], позволяющей проектировать сложные программные системы, работающие в реальном времени как на одном-единственном компьютере, так и в локальной вычислительной сети является операционная система QNX. Встроенные средства операционной системы QNX обеспечивают поддержку многозадачного режима на одном компьютере и взаимодействие параллельно выполняемых задач на разных компьютерах, работающих в среде локальной вычислительной сети. Основным языком программирования в системе является язык С. Основная операционная среда соответствует стандартам POSIX-интерфейса. Это позволяет с небольшими доработками перенести необходимое накопленное программное обеспечение в среду операционной системы QNX для организации работы в среде распределенной обработки.

ОС QNX является сетевой, мультизадачной, многопользовательской (многотерминальной) и масштабируемой системой. С точки зрения пользовательского интерфейса и API она очень похожа на UNIX. Однако QNX не является версией операционной системы UNIX, хотя почему-то многие так считают. Она была разработана, что называется, «с нуля» канадской фирмой QNX Software Systems Limited в 1989 году по заказу Министерства обороны США. Причем эта система построена на совершенно других архитектурных принципах, отличных от принципов, использованных при создании ОС UNIX.

QNX была первой коммерческой ОС, построенной на принципах микроядра и обмена сообщениями. Система реализована в виде совокупности независимых, но взаимодействующих через обмен сообщениями, процессов различного уровня (менеджеров и драйверов), каждый из которых реализует определенный вид сервиса. Эти идеи позволили добиться нескольких важнейших преимуществ [18]:

предсказуемости, означающей ее применимость к задачам жесткого реального времени. QNX является операционной системой, которая дает полную гарантию в том, что процесс с наивысшим приоритетом начнет выполняться практически немедленно и что критическое событие (например, сигнал тревоги) никогда не будет потеряно. Ни одна версия UNIX не может достичь подобного качества, поскольку нереентерабельный код ядра слишком велик. Любой системный вызов из обработчика прерывания в UNIX может привести к непредсказуемой задержке (то же самое касается Windows NT);

масштабируемости и эффективности, достигаемых оптимальным использованием ресурсов и означающим ее применимость для встроенных (embedded) систем; вы не увидите в каталоге /dev большого количества файлов, соответствующих ненужным драйверам. Драйверы и менеджеры можно запускать и удалять (кроме файловой системы, что очевидно) динамически, просто из командной строки. Вы можете иметь только тот сервис, который вам реально нужен, причем это не требует серьезных усилий и не порождает проблем;

расширяемости и надежности одновременно, поскольку написанный вами драйвер не нужно компилировать в ядро, рискуя вызвать нестабильность системы. Менеджеры ресурсов (сервис логического уровня) работают в кольце защиты 3, и вы можете добавлять свои, не опасаясь за систему. Драйверы работают в кольце с уровнем привилегий 1 и могут вызвать проблемы, но не фатального характера. Кроме того, их достаточно просто писать и отлаживать;

поддержки быстрого сетевого протокола FLEET, обеспечивающего прозрачность при обмене сообщениями, автоматическую отказоустойчивость, балансирование нагрузки и маршрутизацию между альтернативными путями доступа (см. подробно подраздел 9.3);

поддержки компактной графической подсистемы Photon, построенной на тех же принципах модульности, что и сама ОС, позволяющей получить полнофункциональный GUI (расширенный Motif), работающий вместе с POSIX-совместимой ОС всего в 4Мбайт памяти, начиная с i80386 процессора.

Приведем основные принципы, реализация которых обязательна при разработке ОС реального времени. Первым обязательным требованием к архитектуре ОСРВ является **многозадачность** в истинном смысле этого слова. Очевидно, что варианты с псевдомногозадачностью (а точнее – невытесняющей многозадачностью) типа MS Windows 3.x или Novell NetWare неприемлемы, поскольку они допускают возможность блокировки или даже полного развала системы одним неправильно работающим процессом. Для предотвращения блокировок ОСРВ должна использовать квантование времени, то есть вытесняющую многозадачность, что является достаточно легкой задачей. Вторая проблема – **организация надежных вычислений –** может быть эффективно решена при полном использовании возможностей процессоров Intel 80386 и старше, что предполагает работу ОС в 32-разрядном режиме процессора. Для эффективного обслуживания прерываний ОС должна использовать алгоритм диспетчеризации, обеспечивающий **вытесняющее планирование, основанное на приоритетах**. Наконец, крайне желательны эффективная **поддержка сетевых коммуникаций** и **наличие развитых механизмов взаимодействия между процессами,** поскольку реальные технологические системы обычно управляются целым комплексом компьютеров и/или контроллеров. Весьма важно также, чтобы ОС поддерживала **множественные потоки управления** (не только *мультипрограммный,* но и *мультизадачный* режимы), а также **симметричную мультипроцессорность**.

И наконец, при соблюдении всех перечисленных условий ОС должна быть способна **работать на ограниченных аппаратных ресурсах***,* поскольку одна из ее основных областей применения – это встроенные системы. К сожалению, данное условие обычно реализуется путем урезания стандартных сервисных средств.

9.2 Особенности архитектуры системы QNX

QNX – это ОС реального времени, позволяющая эффективно организовать распределенные вычисления. В системе реализована концепция связи между задачами на основе сообщений,

посылаемых от одной задачи к другой, причем задачи эти могут находиться как на одном и том же компьютере, так и на удаленных, но связанных между собой локальной вычислительной сетью. Реальное время и концепция связи между процессами в виде сообщений оказывают решающее влияние на разрабатываемое для QNX программное обеспечение и на программиста, стремящегося с максимальной выгодой использовать преимущества системы. Микроядро имеет объем в несколько десятков килобайт (в одной из версий – 10 Кбайт, в другой – менее 32 Кбайт), то есть это одно из самых маленьких ядер среди всех существующих операционных систем. В этом объеме помещаются [19]:

- механизм передачи сообщений между процессами (IPC);
- redirect (перенаправлять) прерываний;
- блок планирования выполнения задач;
- сетевой интерфейс для перенаправления сообщений (менеджер Net).

Механизм передачи межпроцессных сообщений предназначен для пересылки сообщений между процессами и является одной из важнейших частей ОС, так как все общение между процессами, в том числе и системными, происходит через сообщения. Сообщение в QNX – это последовательность байтов произвольной длины (0–65535 байтов) произвольного формата. Протокол обмена сообщениями организуется как блокировка задачи для ожидания сообщения. Одна задача посылает другой сообщение и при этом блокируется сама, ожидая ответа. Первая задача разблокировывается, обрабатывает сообщение и отвечает, разблокируя при этом вторую задачу.

Сообщения и ответы, пересылаемые между процессами при их взаимодействии, находятся в теле отправляющего их процесса до того момента, когда они могут быть приняты. Это значит, что, с одной стороны, уменьшается вероятность повреждения сообщения в процессе передачи, а с другой – уменьшается объем оперативной памяти, необходимый для работы ядра. Кроме того, уменьшается число пересылок из памяти в память, что разгружает процессор. Особенностью процесса передачи сообщений является то, что в сети, состоящей из нескольких компьютеров под управлением QNX, сообщения могут прозрачно переда-

ваться процессам, выполняющимся на любом из узлов. Определены в QNX еще и два дополнительных метода передачи сообщений – ***метод представителей*** (Proxy) и ***метод сигналов*** (Signal).

Представители используются в случаях, когда процесс должен передать сообщение, но не должен при этом блокироваться на передачу. В таком случае вызывается функция qnx_proxy_attach() и создается представитель. Он накапливает в себе сообщения, которые должны быть доставлены другим процессам. Любой процесс, знающий идентификатор представителя, может вызвать функцию Trigger(), после чего будет доставлено первое в очереди сообщение. Функция Trigger() может вызываться несколько раз, и каждый раз представитель будет доставлять следующее сообщение. При этом представитель содержит буфер, в котором может храниться до 65535 сообщений.

Сигналы уже давно используются в ОС UNIX. Система QNX поддерживает **множество сигналов**, совместимых с POSIX, большое количество сигналов, традиционно используемых в UNIX. Поддержка этих сигналов реализована для совместимости с переносимыми приложениями, и ни один из системных процессов QNX их не генерирует. Также системой QNX поддерживается несколько сигналов, специфичных для самой QNX. По умолчанию любой сигнал, полученный процессом, приводит к завершению процесса, кроме нескольких, которые по умолчанию игнорируются. Но процесс, имеющий приоритет уровня «superuser», может защититься от нежелательных сигналов. В любом случае процесс может содержать обработчик для каждого возможного сигнала. Сигналы удобно рассматривать как разновидность программных прерываний.

Redirect прерываний является частью ядра и занимается перенаправлением аппаратных прерываний в связанные с ними процессы. Благодаря такому подходу возникает один побочный эффект – с аппаратной частью компьютера работает ядро, оно перенаправляет прерывания процессам – обработчикам прерываний. Обработчики прерываний обычно встроены в процессы, хотя каждый из них исполняется асинхронно с процессом, в который он встроен. Обработчик исполняется в контексте процесса и имеет доступ ко всем глобальным переменным процесса.

При работе обработчика прерываний прерывания разрешены, но обработчик приостанавливается только в том случае, если произошло более высокоприоритетное прерывание. Если это позволяется аппаратной частью, к одному прерыванию может быть подключено несколько обработчиков, и каждый из них получит управление при возникновении прерывания.

Этот механизм позволяет пользователю избежать работы с аппаратным обеспечением напрямую и тем самым не допустить конфликтов между различными процессами, работающими с одним и тем же устройством. Для обработки сигналов от внешних устройств чрезвычайно важно минимизировать время между возникновением события и началом непосредственной его обработки. Этот фактор существен в любой области применения – от работы терминальных устройств до возможности обработки высокочастотных сигналов,

Блок планирования выполнения задач (диспетчер задач) занимается обеспечением многозадачности. В этой части ОС QNX предоставляет разработчику огромный простор для выбора той методики выделения ресурсов процессора задаче, которая обеспечит наиболее подходящие условия для критических приложений или обеспечит такие условия для некритических приложений, что они выполнятся за разумное время, не мешая работе критических приложений.

К выполнению своих функций как диспетчера ядро приступает в следующих случаях:

- при выходе какого-либо процесса из блокированного состояния;
- при истечении кванта времени для процесса, владеющего CPU;
- при прерывании работающего процесса каким-либо событием.

Диспетчер выбирает процесс для запуска среди неблокированных процессов в порядке значений их приоритетов, которые располагаются в диапазоне от 0 (наименьшего) до 31 (наибольшего). Обслуживание каждого из процессов зависит от метода диспетчеризации, с которым он работает. Уровень приоритета и метод диспетчеризации могут динамически меняться во время работы. В QNX существуют три метода диспетчеризации:

1) FIFO (первым пришел – первым обслужен);

2) round-robin, при котором процессу выделяется определенный квант времени для работы;

3) адаптивный, который является наиболее используемым.

Первый наиболее близок к кооперативной многозадачности, то есть процесс выполняется до тех пор, пока он не перейдет в состояние ожидания сообщения, состояние ожидания ответа на сообщение или не отдаст управление ядру. При переходе в одно из таких состояний процесс помещается последним в очередь процессов с таким же уровнем приоритета, а управление передается процессу с наибольшим приоритетом.

Во втором варианте все происходит так же, как и в предыдущем, с той разницей, что период, в течение которого процесс может работать без перерыва, ограничивается неким квантом времени.

Процесс, работающий с адаптивным методом, в QNX ведет себя следующим образом. Когда процесс полностью использовал выделенный ему квант времени, его приоритет снижается на 1, если в системе есть процессы с тем же уровнем приоритета, готовые к исполнению. При этом если процесс с пониженным приоритетом остается не обслуженным в течение секунды, его приоритет увеличивается на 1; если же процесс блокируется, ему возвращается оригинальное значение приоритета.

По умолчанию процессы запускаются в режиме адаптивной многозадачности. В этом же режиме работают все системные утилиты QNX. Процессы, работающие в разных режимах многозадачности, могут одновременно находиться в памяти и исполняться. Важным элементом реализации многозадачности является приоритет процесса. Обычно приоритет процесса устанавливается при его запуске. Но есть дополнительная возможность, называемая «вызываемый клиентом приоритет». Как правило, она реализуется для серверных процессов, исполняющих запросы на какое-либо обслуживание. При этом приоритет процесса-сервера устанавливается только на время обработки запроса и становится равным приоритету процесса-клиента.

Сетевой интерфейс в системе QNX является неотъемлемой частью ядра. Он, конечно, взаимодействует с сетевым адаптером через сетевой драйвер, но базовые сетевые сервисы реали-

зованы на уровне ядра. При этом передача сообщения процессу, находящемуся на другом компьютере, ничем не отличается с точки зрения приложения от передачи сообщения процессу, выполняющемуся на том же компьютере. Благодаря такой организации сеть превращается в однородную вычислительную среду. При этом для большинства приложений не имеет значения, с какого компьютера они были запущены, на каком исполняются и куда поступают результаты их работы. Такое решение принципиально отличает QNX от остальных ОС, которые тоже имеют все необходимые средства для работы в сети, и делает системы, работающие под ее управлением, по-настоящему распределенными.

Все сервисы QNX, не реализованные непосредственно в ядре, работают как стандартные процессы в полном соответствии с основными концепциями микроядерной архитектуры. С точки зрения операционной системы системные процессы ничем не отличаются от всех остальных, как, впрочем, и драйверы устройств. Единственное, что нужно сделать при организации нового драйвера устройства в QNX для того, чтобы он стал частью ОС, – это изменить конфигурационный файл системы так, чтобы драйвер запускался при загрузке.

9.3 Основные механизмы QNX

QNX является сетевой операционной системой и позволяет организовать эффективные распределенные вычисления. Для организации сети на каждой машине, называемой узлом, помимо ядра и менеджера процессов должен быть запущен уже упомянутый нами выше менеджер Net. Менеджер Net не зависит от аппаратной реализации сети. Данная аппаратная независимость обеспечивается за счет использования сетевых драйверов. В QNX имеются драйверы для различных сетей, например Ethernet, Arcnet, Token Ring. Кроме этого, имеется возможность организации сети через последовательный канал или модем.

В QNX последней, четвертой, версии полностью реализовано встроенное сетевое взаимодействие «точка-точка». Например, находясь на машине А, вы можете скопировать файл с гибкого диска, подключенного к машине В, на жесткий диск, под-

ключенный к машине С. По существу, сеть из машин QNX действует как один мощный компьютер. Любые ресурсы (модемы, диски, принтеры) могут быть добавлены к системе простым их подключением к любой машине в сети. QNX поддерживает одновременную работу в сетях Ethernet, Arcnet, Serial и Token Ring и обеспечивает более чем один-единственный путь для коммуникации, а также балансировку нагрузки в сетях. Если кабель или сетевая плата выходят из строя таким образом, что связь через сеть прекращается, то система будет автоматически перенаправлять данные через другую сеть. Это происходит в режиме «on-line», предоставляя пользователю автоматическую сетевую избыточность и увеличивая скорость коммуникаций во всей системе.

Каждому узлу в сети соответствует уникальный целочисленный идентификатор – логический номер узла. Любой поток в сети QNX имеет прозрачный доступ (при наличии достаточных привилегий) ко всем ресурсам сети, то же самое относится и к взаимодействию потоков. Для взаимодействия потоков, находящихся на разных узлах сети, используются те же самые вызовы ядра, что и для потоков, выполняемых на одном узле. В том случае, если потоки находятся на разных узлах сети, ядро переадресует запрос менеджеру сети. Для организации обмена в сети используется надежный и эффективный протокол транспортного уровня FLEET. Каждый из узлов может принадлежать одновременно нескольким QNX-сетям. В том случае, если сетевое взаимодействие может быть реализовано несколькими путями, для передачи выбирается незагруженная и более скоростная сеть.

Сетевое взаимодействие является узким местом в большинстве операционных систем и обычно создает значительные проблемы для систем реального времени. Для того чтобы обойти это препятствие, разработчики QNX создали собственную специальную сетевую технологию FLEET и соответствующий протокол транспортного уровня FTL (FLEET Transport Layer). Этот протокол не базируется ни на одном из распространенных сетевых протоколов типа IPX или NetBios и обладает рядом качеств, которые делают его уникальным. Основные его качества за-

шифрованы в аббревиатуре FLEET [20] и представлены в таблице.

Составляющие аабревиатуры «FLEET»

Название составляющего процесса сетевой технологии FLEET	Содержание процесса
Fault-Tolerant Networking	QNX может одновременно использовать несколько физических сетей. При выходе из строя любой из них данные будут автоматически перенаправлены «на лету» через другую
Load-Balancing on the Fly	При наличии нескольких физических соединений QNX автоматически распараллеливает передачу пакетов по соответствующим сетям
Efficient Performance	Специальные драйверы, разрабатываемые фирмой QSSL для широкого спектра оборудования, позволяют с максимальной эффективностью использовать сетевое оборудование.
Extensible Architecture	Любые новые типы сетей могут быть поддержаны путем добавления соответствующих драйверов
TransparcntDistributed Processing	Благодаря отсутствию разницы между передачей сообщений в пределах одного узла и между узлами нет необходимости вносить какие-либо изменения в приложения для того, чтобы они могли взаимодействовать через сеть

Благодаря этой технологии сеть компьютеров с QNX фактически можно представлять как один виртуальный суперкомпьютер. Все ресурсы любого из узлов сети автоматически доступны другим, и для этого не нужны специальные «фокусы» с использованием технологии RPC. Это значит, что любая программа может быть запущена на любом узле, при этом ее входные и выходные потоки могут быть направлены на любое устройство на любых других узлах.

Более подробную информацию о работе с QNX можно получить в [6, 21, 22].

Вопросы для самопроверки

1. В каком году была создана ОС QNX? Каковы ее основные возможности?
2. Какие блоки входят в стандартный состав микроядра ОС QNX?
3. Какие методы диспетчеризации существуют в ОС QNX?
4. Какова максимальная длина сообщений в ОС QNX?
5. Сколько исключений резервирует ОС QNX под системные нужды?
6. Благодаря чему ОС QNX приобретает конфигурационную независимость и не зависит от аппаратной реализации сети?
7. Благодаря какой технологии сеть компьютеров с QNX фактически можно представлять как один виртуальный суперкомпьютер?

СПИСОК СОКРАЩЕНИЙ

АЛУ – арифметико-логическое устройство.

ИТ – информационные технологии.

НЖМД – накопители на жестких магнитных дисках.

ОЗУ – оперативное запоминающее устройство.

ООП – объектно-ориентированное программирование.

ОС – операционные системы.

ПЗУ – постоянно запоминающее устройство.

ПО – программное обеспечение.

СУБД – системы управления базами данных.

СУФ – системы управления файлами.

ACPI (Advanced Configuration and Power Interface) – расширенный интерфейс управления питанием и конфигурациями.

AEP (Super-computing Application Environment Profile) – стандарт POSIX.

AGP (Accelerated Graphic Port) – ускоренный графический порт.

API (Application Program Interface) –интерфейс прикладного программирования.

BIOS (Base Input Output System) – базовая система ввода-вывода.

CDFS (Compact Disk File System) – файловая система компактного диска.

CLX (Component Library for Cross-platform, «кликс») – кросс-плат-форменная библиотека компонент производства фирмы Borland.

CORBA (Common Object Request Broker Architecture) – архитектура посредника стандартного объектного запроса.

CPU (central processing unit) – центральный процессор.

CTOS (An Operating System produced Convergent Technology …) – технология переноса операционных систем.

DDE (Dynamic Data Exchange) – универсальные механизмы динамического обмена данными.

DLL (Dynamic Link Library) – динамически загружаемый библиотечный модуль.

DMA (Direct Memory Access) – канал прямого доступа к памяти.

DO (Diagnostic Output) – диагностический поток вывода.

DRAM (Dynamic Random Access Memory) – динамическое оперативное запоминающее устройство.

EMS (Expanded Memory Specification) – отображаемая память.

ERP (Enterprise Resource Planning) – планирование ресурсов предприятий.

ESCD (Extended System Configuration Data) – область энергонезависимой памяти.

FAT (File Allocation Table) – таблица размещения файлов.

FCFS (First Come – First Served) – дисциплина обслуживания «первым пришел – первым обслужился».

FLEET (Fault-Tolerant Networking, Load-Balancing on the Fly, Efficient Performance, Extensablc Architecture, Transparcnt Distributed Processing) – сетевая технология в ОС QNX.

FTL (FLEET Transport Layer) – протокол транспортного уровня (см. FLEET).

GDI (Graphics Device Interface) – интерфейс графических устройств.

GDT (Global Descriptor Table) – глобальная таблица дескрипторов.

GID (Group Identifier) – идентификатор группы.

GUI (Graphical User Interface) – пользовательский графический интерфейс.

HDD (Hard Disk Drive) – жесткий диск.

HMA (High Memory Area) – высокая область памяти.

HPFS (High Performance File System) – файловая система высокого исполнения.

ICS (Internet Connection Sharing) – подключение к Internet с общим доступом.

IDT (Interrupt Descriptor Table) – таблица дескрипторов прерываний.

IEEE (Institute Electrical and Electronic Engineers) – институт инженеров по электротехнике и радиоэлектронике

IEEE 1394 – стандарт, предназначенный для подключения высокоскоростных последовательных устройств.

IFS (Installing File System) – инсталлируемая файловая система.

IPC (Inter-Process Communication) – межпроцессные коммуникации.

IIS (Internet Information Server) – WWW-сервер.

JCL (Job Control Language) – язык управления заданиями.

JFS (Journaling File System) – журналирующая файловая система.

LDT (Logical Disk Table) – таблица логического диска.

LSB (Least Significant Bit) – младший бит.

LVM (Logical Volume Manager) – менеджер томов (логических дисков).

MAPI (Message Applica-tion Programming Interface) – интерфейс прикладного программирования.

MBR (Master Boot Record) – главная загрузочная запись.

MESI (M (Modified), E (Exclusive), S (Shared), I (Invalid)) – протокол кэша.

MFC (Microsoft Foundation Classes) – библиотека классов Microsoft.

MFT (Master File Table) – файл метаданных в NTFS.

MSB (Most Significant Bit) – старший бит.

MTBF (Mean Time Before Failure) – ожидаемое время до отказа.

MTRR (Memory Type Range Registers) – регистры, описывающие свойства областей памяти.

NCSC (National Computer Security Center) – национальный центр компьютерной безопасности США.

NSB (Non-System Bootstrap) – внесистемный загрузчик.

NTFS (New Technology File System) – файловая система на основе новой технологии.

OLE (Object Linking and Embedding) – технология документно-ориентированной архитектуры приложений.

PAT (Page Attribute Table) – таблица атрибутов страниц памяти.

PID (Process Identificator) – идентификатор процесса.

PE (Protect Enable) – бит включения защищенного режима.

POSIX (Portable Operating System Interface for Computer Environments) – платформенно-независимый системный интерфейс для компьютерного окружения.

PPTP (Point-to-Point Tunneling Protocol) – протокол для создания виртуальных частных сетей.

PT (Partition Table) – таблица разделов.

RAM (Random Access Memory) – оперативное запоминающее устройство.

ROM (Read Only Memory) – постоянное запоминающее устройство.

RPC (Remote Procedure Call) – вызов удаленных подпрограмм.

RPM (Revolutions Per Minute) – обороты в минуту.

RR (round robin) – карусельная дисциплина обслуживания.

RTL (Run Time Library) – библиотека времени выполнения.

S.I. (Standard Input) – стандартный поток ввода.

SJN (Shortest Job Next) – следующее короткое задание (дисциплина обслуживания заданий).

SMBR (Secondary MBR) – следующая загрузочная запись.

SMI# (System Management Interrupt) – сигнал переключения в режим SMM.

SMM (System Management Mode) – режим системного управления.

SMRAM (System Management Random Access Memory) – системное управление оперативным запоминающим устройством.

S.O. (Standard Output) – стандартный поток вывода.

SOAP (Simple Object Access Protocol) – стандарт, описывающий протокол, предназначенный для обмена структурированной информацией в распределенных системах.

SOM (System Object Model) – модель системных объектов.

SRT (Shortest Remaining Time) – кратчайшее остаточное время (дисциплина обслуживания заданий)

TPI (Track Per Inch) – количество треков на дюйм.

TR (Task Register) – регистр задачи.

TSS (Task State Segment) – сегмент состояния задачи.

TTF (True Type Font) – наименование типа шрифтов.

UC (Uncacheable) – некэшируемая память.

UDDI (Universal Description, Discovery and Integration) – универсальный метод описания, обнаружения и интеграции web-сервисов для систем электронной коммерции.

UDF (Universal Disk Format) – универсальный дисковый формат.

UID (User Identifier) – пользовательский идентификатор.

UMA (Upper Memory Area) – верхняя область памяти.

UPnP (Universal Plug and Play) – технология поддержки универсальных самонастраивающихся устройств.

USB (Universal Serial Bus) – универсальная последовательная шина.

VCL (Visual Controls Library) – библиотека классов Borland.

VDM-машина (Virtual DOS Machine) – виртуальная DOS машина.

VMM (Virtual Memory Manager) – диспетчер виртуальной памяти.

VMS (Virtual Memory System) – система виртуальной памяти.

VPN (Virtual Private Network) – виртуальные частные сети.

WAN (Wide Area Network) – глобальная сеть.

WB (Write-back) – кэширования память с обратной записью.

WC (Write Combining) – кэширования память с комбинируемой записью.

WOW (Windows on Windows) – окна в окнах (сеанс 16-разрядных прикладных Windows-программ ОС Windows NT).

WP (Write protected) – кэширования память с защищенной записью.

WPS (Workplace Shell) – стандартная графическая оболочка OS/2 Warp.

WSDL (Web Services Description Language) – язык для описания Интернет сервисов.

WT (Write-through) – кэширования память со сквозной записью.

XML (Extensible Markup Language) – язык разметки.

XMS (Extended Memory Specification) – расширенная память.

ЛИТЕРАТУРА

1. Брябрин В. М. Программное обеспечение персональных ЭВМ. – 3-е изд., стереотип. – М.: Наука, 1990. – 272 с.

2. Гордеев А.В. Системное программное обеспечение / А.В. Гордеев, А.Ю. Молчанов. – СПб.: Питер, 2002. – 736 с.

3. Кейлингерт П. Элементы операционных систем. Введение для пользователей. – М.: Мир, 1985. – 295 с.

4. Мэдник С. Операционные системы / С. Мэдник, Дж. Донован. – М.: Мир, 1978. – 792 с.

5. Бэкон Д. Операционные системы / Д. Бэкон, Т. Харрис. – СПб.: Питер; Киев: Издательская группа BHV, 2004. – 800 с.

6. Алексеев Д. Практика работы с QNX / Д. Алексеев, Е. Ведревич, А. Волков и др. – М.: Издательский Дом «КомБук», 2004. – 432 с.

7. IA-32 Intel Architecture Software. Developer's Manual, Volume 1–4.

8. Озеров В. Советы по Дельфи. – http://www.webmachine.ru/delphi.

9. Гордеев А.В. Управление процессами в операционных системах реального времени: Учебное пособие / А.В. Гордеев, В.А. Штепен. – Л.: ЛИАП, 1988. – 76 с.

10. Робачевский А.М. Операционная система UNIX. – СПб.: BHV–Санкт-Петербург, 1997. – 528с.

11. Олифер Н.А. Сетевые операционные системы / Н.А. Олифер, В.Г. Олифер. – СПб.: Питер, 2001.

12. Гук М.. Аппаратные средства IBM PC: Энциклопедия. – 2-е изд. – СПб.: Питер, 2002. – 928 с.

13. Гук М. Процессоры Pentium III, Athlon и другие / М. Гук, В. Юров. – СПб.: Питер, 2000. – 480с.

14. Фигурнов В. Э. IBM PC для пользователя: Краткий курс. – М.: ИНФРА-М, 2003. – 480 с.

15. Федоров А. Microsoft Windows и файловые системы // Компьютер Пресс. – 2000. – № 7. – С. 54–60.

16. http://standards.ieee.org/regauth/posix/index.html

17. http://www.microsoft.ru/rus/

18. ОС QNX: Обзор системы. – http://www.lgg.ru/~nigl/QNX/doc

19. Обухов И. QNX: Как надо делать операционные системы / PC Week Re. – 1998. – № 7.

20. http://support.qnx.com

21. Зыль С. Операционная система реального времени QNX: от теории к практике. – СПб.: БХВ-Петербург, 2004. – 192 с.

22. Роб Кёртен. Введение в QNX/Neutrino 2. – СПб.: Петрополис, 2002. – 512 с.

КОНТРОЛЬНЫЕ РАБОТЫ

Контрольная работа №1

1. Изучить структуру системных таблиц реального режима Windows и организацию цепочек блоков памяти.

Познакомиться с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виде шестнадцатиричного дампа –PEEK.COM (во время работы доступен HELP – F1, карта памяти – F8 и информация о блоке памяти – F6).

Найти в памяти таблицу таблиц (для получения ее адреса – запусти lol.com), познакомиться с ее содержимым и посмотреть указатель на 1 MCB (см. пп.3.2.1, электронный справочник THelp).

Проследить в памяти цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти (F8).

Написать отчет с описанием процесса нахождения блоков MCB. В отчете указать цепочку блоков памяти с их адресами и размерами.

2. Письменно ответить на вопросы согласно полученному варианту.

Вариант 1
1. В чем отличие между понятиями процесса и задачи?
2. Расскажите о механизме кэширования памяти.

Вариант 2
1. Изобразите диаграмму состояний процесса, поясните все возможные переходы из одного состояния в другое.
2. Опишите распределение оперативной памяти в Windows NT.

Вариант 3
1. Объясните значения следующих терминов: task (задача), process (процесс), thread (поток, нить). Как они между собой соотносятся?
2. Опишите распределение оперативной памяти в Windows 9x.

Вариант 4

1. Для чего каждая задача получает соответствующий дескриптор? Какие поля, как правило, содержатся в дескрипторе процесса (задачи)?
2. Назовите основные принципы фон-неймановской архитектуры вычислительных машин.

Вариант 5

1. Объясните понятие ресурса. Почему понятие ресурса является одним из фундаментальных при рассмотрении ОС? Какие виды и типы ресурсов вы знаете?
2. Опишите оценки качества диспетчеризации.

Вариант 6

1. Как вы считаете: сколько и каких списков дескрипторов задач может быть в системе? От чего должно зависеть это число?
2. Сравните механизмы диспетчеризации задач в ОС Windows NT и UNIX. В чем заключаются основные различия?

Вариант 7

1. Перечислите дисциплины обслуживания прерываний; объясните, как можно реализовать каждую из этих дисциплин.
2. В чем отличие алгоритмов диспетчеризации с вытесняющей и невытесняющей многозадачностью?

Вариант 8

1. С какой целью в ОС вводится специальный системный модуль, иногда называемый супервизором прерываний?
2. Какие дисциплины диспетчеризации задач вы знаете? Опишите их.

Вариант 9

1. Приведите классификацию ОС?
2. Какие стратегии диспетчеризации вы знаете?

Вариант 10
1. Дайте объяснение понятиям операционной среды и операционной системы.
2. Перечислите и поясните основные функции ОС, которые связаны с управлением задачами.

Контрольная работа №2

1. Изучить структуру файловой системы FAT.

Подготовиться к работе, используя справочные материалы данного учебного пособия (пп. 4.3.2) и электронный справочник THelp.

Познакомиться с основным меню DE.EXE (Norton Utilites).

Исследовать и описать средства работы с диском на уровне DOS (кластер, логический сектор, BOOT, FAT, ROOT DIR).

Исследовать и описать структуру загрузочного сектора системной и обычной дискеты.

Исследовать и описать структуру таблицы распределения файлов.

Исследовать и описать структуру корневого каталога.

Исследовать и описать изменения в системной области диска при создании и удалении файла и способы восстановления удалённых файлов.

Сформулировать принцип восстановления удаленных файлов в MS-DOS, условия восстановления и рекомендации пользователю по работе в системе, увеличивающие шансы успешного восстановления.

ВНИМАНИЕ. Речь идет не об использовании стандартной утилиты – например, UNDELETE, а об АЛГОРИТМЕ восстановления.

2. Письменно ответить на вопрос согласно полученному варианту.

Вариант 1
Перечислите и поясните основные принципы построения ОС.

Вариант 2
Расскажите о концепции построения микроядерной ОС. Какие основные функции должно выполнять микроядро ОС?

Вариант 3
Расскажите о концепции построения ОС с монолитным ядром.

Вариант 4.
Какие задачи возлагаются на интерфейс прикладного программирования (API)?

Вариант 5
Какими могут быть варианты реализации API? В чем их достоинства и недостатки?

Вариант 6
Что такое библиотека времени выполнения?

Вариант 7
Что такое POSIX? Какими преимуществами обладают программы, созданные с использованием только стандартных функций, предусмотренных POSIX?

Вариант 8
Опишите структуру магнитного диска (разбиение дисков на разделы). Сколько разделов может быть на магнитном диске? Каково назначение разделов магнитного диска?

Вариант 9
Приведите основные характеристики файловой системы ОС UNIX?

Вариант 10
Приведите основные характеристики HPFS. За счет чего в файловой системе HPFS обеспечена высокая производительность?

Контрольная работа №3

1. Разработать командный файл согласно полученному варианту.

При разработке учтите возможность обработки различных ошибок, например, неправильного запуска ваших программ (с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки. При выполнении работы используйте электронный справочник THelp.

2. Письменно ответить на вопрос согласно полученному варианту.

Вариант 1
1. Разработать командный файл, создающий, копирующий или удаляющий файл, указанный в командной строке, в зависимости от выбранного ключа /n , /c , /d.
2. Перечислите основные части ОС MS DOS.

Вариант 2
1. Разработать командный файл, добавляющий вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей /b /e).
2. В чем состоит назначение BootRecord?

Вариант 3
1. Разработать командный файл, регистрирующий время своего запуска в файле протокола run.log и автоматически запускающий некоторую программу (например, антивирусную и т.п.) по пятницам или 13 числам.
2. Приведите последовательность загрузки ОС MS DOS.

Вариант 4
1. Разработать командный файл, который в интерактивном режиме мог бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.
2. Опишите файл конфигурации MS DOS CONFIG.SYS и приведите основные команды конфигурирования. Поясните назначе-

ние файлов пакетной обработки и особенности командного файла автозапуска AUTOEXEC.BAT.

Вариант 5
1. Разработать командный файл, который при запуске выполнял какие-либо действия только один раз в сутки.
2. Перечислите основные функции командного процессора. Раскройте принцип работы командного процессора при обработке внутренних и внешних команд ОС MS DOS.

Вариант 6
1. Разработать командный файл, который получал в качестве параметра какое-либо имя, проверял, определена ли такая переменная среды или нет, и выводил соответствующее сообщение.
2. Что такое Windows API и что он позволяет?

Вариант 7
1. В некотором файле хранится список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос – копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да», то программа копирует файл.
2. Как вы понимаете вытеснение задач в ОС Windows?

Вариант 8
1. Разработать командный файл (аналог команды tail в Unix). Командный файл печатает конец файла. По умолчанию – 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.
2. Что такое OLE2 и «Drag and Drop»?

Вариант 9
1. Разработать командный файл, создающий, копирующий или удаляющий каталог, указанный в командной строке, в зависимости от выбранного ключа /n , /c , /d.
2. В каких ОС семейства Windows можно вводить дисковые квоты и осуществлять поддержку массива RAID 5?

Вариант 10
1. Разработать командный файл, копирующий произвольное число файлов, заданных аргументами из текущего каталога в каталог C:\Temp.
2. Назовите основные характеристики ОС MS DOS.

Контрольная работа №4

1. Познакомтесь с панелью управления Windows, изучите возможности изменения основных параметров Windows.

Изучить основные компоненты панели управления Windows (Экран, Язык и региональные стандарты, Система, Сетевые подключения, Свойства папки, Администрирование, Панель задач и меню пуск, Дата и время).

Изучить возможность использования справочной системы Windows.

Изучить основные возможности Internet Explorer и Outlook Express.

Напишите реферат на тему использование Интернет и электронной почты.

2. Письменно ответить на вопросы согласно полученному варианту

Вариант 1
1. В каком году появилась 32-разрядная ОС от фирмы IBM? Как она называется? Расскажите о ее основных возможностях.
2. Какие атрибуты файлов существуют в ОС UNIX?
3. Какие блоки входят в стандартный состав микроядра ОС QNX?

Вариант 2
1. Что такое VDM-ядро и DMA в ОС OS/2 Warp?
2. Что понимают под процессом в ОС UNIX?

3. Благодаря какой технологии сеть компьютеров с QNX фактически можно представлять как один виртуальный суперкомпьютер?

Вариант 3
1. Как называется стандартная графическая оболочка в ОС OS/2 Warp и каковы ее возможности?
2. Расскажите о стандартных потоках и средствах их перенаправления в ОС UNIX. Что понимают под термином «фильтры»?
3. Расскажите о процессе Init в ОС UNIX.

Вариант 4
1. Как называется новая файловая система в ОС OS/2 Warp 4.5 и в чем ее преимущества?
2. Какие цели имелись при создании ОС UNIX?
3. Какие механизмы межпроцессных коммуникаций в ОС UNIX Вы знаете?

Вариант 5
1. Перечислите и поясните основные понятия системы UNIX.
2. Какие состояния существуют у процессов в ОС UNIX?
3. В каком году была создана ОС QNX? Каковы ее основные возможности?

Вариант 6
1. Изложите основные архитектурные особенности ОС UNIX.
2. Расскажите о системных вызовах Fork и Exec.
3. Какова максимальная длина сообщений в ОС QNX?

Вариант 7
1. Из чего состоит образ памяти в ОС UNIX?
2. Что представляет собой ОС Linux? Какому стандарту она отвечает? На какой процессор была изначально рассчитана?
3. Благодаря чему ОС QNX приобретает конфигурационную независимость и не зависит от аппаратной реализации сети?

Вариант 8
1. Что понимают под понятием Shell?
2. Какая файловая система была разработана специально для ОС Linux?
3. Сколько исключений резервирует ОС QNX под системные нужды?

Вариант 9
1. Что понимают под понятием Kernel?
2. Как называется WWW-сервер для ОС Linux?
3. Какие методы диспетчеризации существуют в ОС QNX?

Вариант 10
1. Расскажите о режимах переднего и заднего плана. Какие команды позволяют управлять процессами на заднем плане?
2. Какими преимуществами обладает ОС QNX благодаря принципам построения микроядра.
3. Расскажите о системно-объектной модели SOM, разработанной фирмой IBM.