Томский межвузовский центр дистанционного образования

Ю.Б. Гриценко

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Часть 2

Учебное пособие

ТОМСК — 2009

Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

Ю.Б. Гриценко

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Часть 2

Учебное пособие

2009

Рецензенты: канд. техн. наук, доцент кафедры теоретических основ информатики Томского государственного университета **Гусев И.С.**;
д-р техн. наук, профессор, заведующий кафедрой теоретических основ информатики Томского государственного университета **Костюк Ю.Л.**

Корректор: Осипова Е.А.

Гриценко Ю.Б.
Операционные системы: Учебное пособие. В 2-х частях. — Томск: Томский межвузовский центр дистанционного образования, 2009. — Ч.2. — 230 с.

Данное учебное пособие содержит вторую часть курса «Операционные системы», изучаемого студентами специальности 230102 «Автоматизированные системы обработки информации и управления», обучающимися по дистанционной форме. Рассмотрены вопросы организации вычислительных задач, управления памятью и устройствами ввода-вывода. Основное внимание уделено механизмам планирования процессов, диспетчеризации задач и программной модели микропроцессора Intel P6.

ОГЛАВЛЕНИЕ

3 Управление устройствами ввода-вывода.....152

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).

Программное обеспечение — 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. Например, одна из классификаций предлагает все программы, созданные для ЭВМ, разделить на следующие основные классы:

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

В работах других авторов, например Дж. Бэкона и Т. Харриса, предлагается выделить класс ***системного программного обеспечения*** из всего множества ПО (другой класс — прикладное программное обеспечение), к которому и будут относиться операционные системы. В некоторых случаях программное обеспечение, в особенности системное, не может рассматривать-

ся отдельно от аппаратного обеспечения, которое поддерживает его работу и в большей мере определяет его структуру.

Изучение дисциплины «Операционные системы» представляет собой основу для изучения всего процесса управления информационными ресурсами.

В содержание дисциплины входит изучение как теоретического материала: структур, методов и алгоритмов построения современных операционных сред и систем (ОС); так и изучение возможностей функционирования современных популярных 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 основано 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. В настоящее время наблюдается возросший интерес со стороны ученных и коммерческих фирм к развитию теорий построения ОС.

Для изучения дисциплины «Операционные системы» необходимо иметь навыки программирования на языке высокого уровня Си или Паскаль.

Структура учебного пособия

Учебное пособие состоит из двух частей. Первая часть ориентирована на рассмотрение функциональных и архитектурных особенностей популярных операционных систем. Вторая же часть посвящена общей теории построения операционных систем.

Часть 2:

В первой главе рассмотрены основные понятия организации вычислительных задач. К этим понятиям относятся: понятия процесса, потока, ресурса, режима мультипрограммирования, планирования и диспетчеризации задач. Здесь же раскрыты темы взаимодействия и синхронизации задач, организации прерываний и управление задачами в ОС Windows.

Вторая глава содержит описание процесса управления памятью, который включает в себя описание: иерархии памяти, программной модели микропроцессора Intel P6, режимов функционирования микропроцессора Inte l P6 и управления памятью в ОС Windows.

В третьей главе изложен процесс управления устройствами ввода-вывода. В этой главе представлены классификация и основные характеристики устройств ввода-вывода; организация операций ввода-вывода; физическая и логическая структуры магнитного диска и описание процесса управления устройствами ввода-вывода и файловыми системами в ОС Windows.

В конце учебного пособия приведены задания для выполнения контрольных работ по второй части данного курса.

1 ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ

1.1 Процессы. Ресурсы. Режим мультипрограммирования

Понятие «вычислительный процесс» (или просто — «процесс») является одним из основных при рассмотрении операционных систем. Под **процессом** понимают выполняемую программу, включающую текущие значения счетчика команд, регистров и переменных [1]. Впервые концепция процесса была реализована в операционных системах 1960-х годов (MULTICS, 1965; THE, 1968; RC4000, 1970).

По принципу выполнения различают последовательные процессы и параллельные. ***Последовательный процесс***, иногда называемый «задачей», — это выполнение отдельной программы с ее данными на последовательном процессоре [2]. 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**. Понятие 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 [3]. 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).

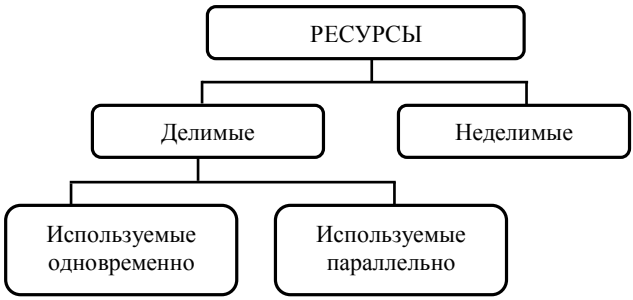

Рис. 1.1 — Классификация ресурсов

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 мо-

гут 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 на одном процессоре попеременно выполняется несколько программ. При мультипрограммировании повышается пропускная способность системы, но время выполнения отдельного процесса никогда не превышает времени выполнения данного процесса в однопрограммном режиме. Всякое раз-

деление ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождающегося ресурса.

Операционная система поддерживает режим мультипрограммирования и организует процесс эффективного использования ресурсов путем управления очередью запросов к ним, 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.

Общую схему выделения ресурсов можно представить следующим образом:

1. 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 устройство печати данных).

2. 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 [3]:

– 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).

3. 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.

4. 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 обслужи-*

вать максимально возможное количество процессов и наиболее полно использовать ресурсы [2].

Необходимо отличать системные управляющие процессы, представляющие работу супервизора операционной системы и занимающиеся распределением и управлением ресурсами, 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 микроядерных ОС 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). Аналогично и в UNIX-системах выполнение системных программных модулей тоже имеет статус системных процессов, которые получают ресурсы для своего исполнения в первую очередь.

Интересная форма мультипрограммной работы связана с мультипроцессорной обработкой. **Мультипроцессорная обработка** — это способ организации вычислительного процесса в системе с несколькими процессорами, при котором несколько

задач могут одновременно выполняться на разных процессорах системы. Концепция мультипроцессирования не нова, она известна с 70-х годов, однако стала доступной в широком масштабе лишь последнее десятилетие, особенно с появлением многопроцессорных ПК.

В отличие от мультипрограммной обработки в мультипроцессорных системах несколько задач выполняется одновременно, так как имеется несколько процессоров. Однако это не исключает мультипрограммной обработки на каждом процессоре. При этом резко усложняются все алгоритмы управления ресурсами, то есть операционная система.

Современные ОС, как правило, поддерживают мультипроцессирование (Sun Solaris 2.X, Santa Cruz Operation Open Server 3.x, OS/2, Windows NT/2000/2003, NetWare, начиная с версии 4.1) [6].

Анализ операционных систем 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 [3]:

– ***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 [3]. Рассмотрим переходы из одного состояния в другое более подробно.

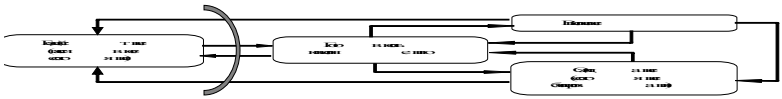

Рис. 1.2 — Граф состояний процесса

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.

Процесс, который может исполняться, как только ему будет предоставлен процессор, а для диск-резидентных задач в некоторых системах — и оперативная память, находится в состоянии готовности. Считается, что такому процессу уже выделены все необходимые ресурсы за исключением процессора.

Из состояния выполнения процесс может выйти по одной из следующих причин [3]:

– ***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

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

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, как уже было сказано выше, 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.

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 место-

нахождение ***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.

1.2 Потоки. Волокна

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

лять 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 пространстве, 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 (следовательно, и более эффективное использование ресурсов центрального процессора, и меньшее суммарное время выполнения задач) теперь уже часто реализуется 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; напри-

мер 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 [3].

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 отличие 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 относящих-

ся 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.

Переключение потоков в ОС занимает довольно много времени, так как для этого необходимо переключение в режим ядра, а затем возврат в режим пользователя. Достаточно велики затраты процессорного времени на планирование и диспетчеризацию потоков (об этом далее). Для предоставления сильно облегченного псевдопараллелизма в Windows 2000 было введено использование волокон (Fiber) (рис. 1.3). Волокна подобны потокам, но планируются в пространстве пользователя создавшей их программой. У каждого потока может быть несколько волокон, с той разницей, что когда волокно логически блокируется, оно помещается в очередь блокированных волокон, после чего для работы выбирается другое волокно в контексте того же потока. При этом ОС «не знает» о смене волокон, так как все тот же поток продолжает работу [6].

В завершение можно привести ***несколько советов***, представленных В. Озеровым на сайте http://www.webmachine.ru/delphi, ***по использованию потоков при создании приложений***.

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.

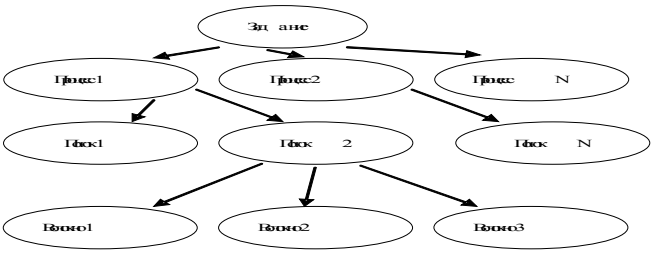

Рис. 1.3 — Иерархия заданий в ОС Windows 2000

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.

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

1.3 Планирование процессов и диспетчеризация задач

1.3.1 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 коммуникаций.

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** [3]. 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 ожиданий), да и имеющиеся в системе ресурсы будут использоваться более эффективно. Итак, возникает **задача планирования вы-**

числительного процесса, то есть организация бесконфликтного выполнения множества процессов. ***Основная цель планирования вычислительного процесса заключается в распределении времени процессора или нескольких процессоров между выполняющимися заданиями пользователей таким образом, чтобы удовлетворить требованиям, предъявляемым пользователями к вычислительной системе*** [6].

Все виды планирования, используемые в современных ОС, в зависимости от временного масштаба делятся на долгосрочное, среднесрочное, краткосрочное и планирование ввода-вывода. Рассматривая частоту работы планировщика, можно сказать, что долгосрочное планирование выполняется сравнительно редко, среднесрочное — несколько чаще. Краткосрочный планировщик, называемый часто диспетчером (dispatcher), работает чаще всего, определяя, какой процесс или поток будет выполняться следующим. Ниже приведен перечень функций, выполняемых планировщиком каждого вида (табл. 1.1) [6].

Таблица 1.1 — **Соответствие видов планирования и выполняемых функций**

Вид планирования	***Выполняемые функции***
Долгосрочное	Решение о добавлении задания (процесса) в пул выполняемых в системе
Среднесрочное	Решение о добавлении процесса к числу процессов, полностью или частично размещенных в основной памяти
Краткосрочное	Решение о том, какой из доступных процессов (потоков) будет выполняться процессором
Планирование ввода-вывода	Решение о том, какой из запросов процессов (потоков) на операцию ввода-вывода будет выполняться свободным устройством ввода-вывода

Задача планирования процессов возникла на этапе создания первых пакетных ОС при планировании пакетов задач, которые должны были выполняться на компьютере и оптимально ис-

пользовать его ресурсы. В настоящее время актуальность задачи оптимального использования ресурсов не так велика.

Выделяют два типа планирования: статическое и динамическое.

На первый план уже давно вышли задачи **динамического** (или краткосрочного) планирования, то есть текущего наиболее эффективного распределения ресурсов, возникающего практически при каждом событии. ***При динамическом планировании решения принимаются во время работы системы на основе анализа текущей ситуации, без использования предложений о мультипрограммной смеси.***

Организация работ по выполнению задач динамического планирования получила название ***диспетчеризации***.

Другой тип планирования — **статический** — может быть ***использован только в специализированных системах с заданным набором задач, например в управляющих вычислительных системах или системах реального времени. В этом случае статический планировщик принимает решение не во время работы системы, а заранее*** [6].

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 задач, находящихся в очереди гото-

вых к выполнению, которые должны быть переданы на исполнение. В большинстве современных операционных систем долгосрочный планировщик отсутствует.

1.3.2 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*** [3]:

– обеспечение 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).

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 задачи 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. 1.4.

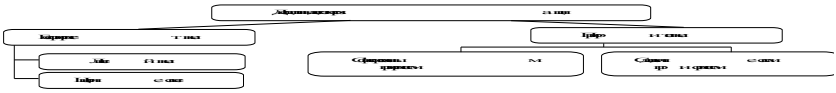

Рис. 1.4 — Дисциплины диспетчеризации

Приоритеты имеют следующие свойства [3]:

– 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 динамические приоритеты позволяют реализовать гарантии обслуживания задач.

Рассмотрим кратко основные, наиболее часто используемые, дисциплины диспетчеризации.

Бесприоритетные дисциплины диспетчеризации

Самой простой в реализации является **дисциплина FCFS**[2] (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. 1.5): одна очередь образуется из новых задач, а вторая очередь — из ранее выполнявшихся, но попавших в состояние ожидания. 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.

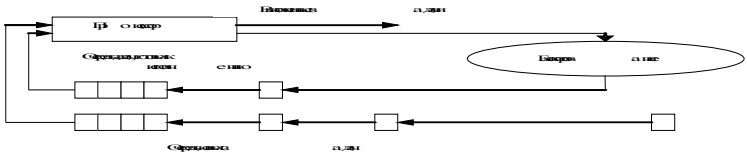

Рис. 1.5 — Дисциплина диспетчеризации FCFS

К достоинствам этой дисциплины диспетчеризации, прежде всего, можно отнести простоту реализации и малые расходы системных ресурсов на формирование очереди задач. Однако эта

[2] В некоторых публикациях ее иногда называют FIFO (first input — first output).

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.

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 заказанного 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 кото-

рой 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 задаче. Снятая задача ставится в конец очереди задач, готовых к выполнению (рис. 1.6). Для оптимальной работы системы необходимо правильно выбрать закон, по которому кванты времени выделяются задачам.

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 бу-

дет 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.

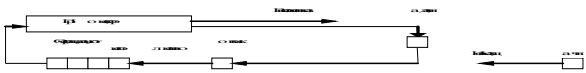

Рис. 1.6 — Дисциплина обслуживания RR

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 предоставляется задаче с максимальным приоритетом, а при равенстве приоритетов он действует по принципу очередности) [8]. 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, кото-

рая не предусматривает никаких действий по ее выполнению, но 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 [3].

Приоритетные дисциплины диспетчеризации

Появление второй группы методов диспетчеризации связано с необходимостью распределения времени процессора между потоками, имеющими разную важность, то есть разный приоритет. В таких случаях для потоков с равным приоритетом используется один из указанных выше методов диспетчеризации, а передача управления между потоками с разным приоритетом осуществляется с помощью приоритетных дисциплин диспетчеризации.

В наиболее простом случае если в состояние готовности переходят два потока с разными приоритетами, то процессорное время передается тому, у которого более высокий приоритет. Данная дисциплина называется **приоритетной многозадачностью**, но ее использование в таком виде связано с рядом сложностей. При наличии в системе одной группы потоков с одним приоритетом и другой группы с другим, более низким приоритетом, при карусельной диспетчеризации каждой группы в системе с приоритетной многозадачностью потоки низкоприоритетной группы могут вообще не получить доступа к процессору. Ре-

шением этой проблемы могут быть дисциплины диспетчеризации с динамическими приоритетами.

Так называемым общим случаем дисциплины с динамическими приоритетами может выступать **адаптивная многозадачность**, широко применяющаяся в интерфейсных системах. Суть дисциплины заключается в том, что приоритет потока, не выполняющегося какой-то период времени, повышается на единицу. Восстановление исходного приоритета происходит после выполнения потока в течение одного кванта времени или при блокировке потока. Таким образом, при карусельной многозадачности, очередь (или «карусель») более приоритетных потоков не может полностью заблокировать выполнение очереди менее приоритетных потоков.

Дальнейшим развитием дисциплины на основе адаптивной многозадачности стала дисциплина **спорадической многозадачности**. В ней задается для потоков бюджет времени выполнения и время восстановления бюджета. Когда поток с высоким приоритетом отработал свой бюджет выполнения, его приоритет понижается, давая возможности работать потокам с более низкими приоритетами. По прошествии времени для восстановления бюджета времени выполнения приоритет у потока с высоким приоритетом восстанавливается, и управление вновь передается ему. Адаптивная и спорадическая многозадачность были реализованы в **ОС реального времени QNX**.

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 не относится к ОС реального времени. Приоритет процесса вычисляется следующим образом [9]:

1. Во-первых, в вычислении участвуют значения двух полей дескриптора процесса — p_nice и p_cpu. Первое из них назнача-

ется пользователем явно или формируется по умолчанию с помощью системы программирования. Второе поле формируется диспетчером задач 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 операционной системой, и предназначены для поддержки приложений реального времени.

2. Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение **приоритета сна**, выбираемое ядром из диапазона системных приоритетов и связанное с событием, 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.

3. После завершения системного вызова перед возвращением в режим задачи ядро восстанавливает приоритет режима за-

дачи, сохраненный перед выполнением системного вызова. Это может привести к понижению приоритета, что, в свою очередь, вызовет переключение контекста.

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.

4. Каждую секунду ядро пересчитывает текущие приоритеты готовых к запуску процессов, приоритеты которых меньше некоторого порогового значения (в описываемом 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.

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 приоритет (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 [3]. 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, то есть нет готовых к выполнению задач с таким приоритетом, осуществляется переход к следующей очереди. Следовательно, если есть задачи, требующие процессор немедленно, они будут обслужены в первую 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 пере-

менного 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.

1.3.3 Вытесняющаяся и не вытесняющаяся многозадачность

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 многозадачность** (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 [3]. 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:

1) определяет момент снятия с выполнения текущей задачи;

2) сохраняет ее контекст в дескрипторе задачи;

3) 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 [10]. 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 [10]. 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.

1.3.4 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 обслужен 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 [3]:

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 [3]:

– ***загрузка (использование) центрального процессора.*** 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) [3]: указания разной степени категоричности о снятии текущего

процесса с процессора, а также указания о процессе, который должен быть выбран взамен текущего.

1.4 Взаимодействие и синхронизация задач

1.4.1 Способы взаимодействия задач

В однопроцессорных системах в режиме мультипрограммирования процессы чередуются, обеспечивая эффективное выполнение программ. В многопроцессорных системах возможно не только чередование, но и перекрытие процессов. При использовании обеих технологий выполнение процессов и потоков в мультипрограммной среде всегда имеет асинхронный характер, невозможно предсказать относительную скорость выполнения задач. Это справедливо как по отношению к потокам одного процесса, выполняющим общий программный код, так и по отношению к потокам разных процессов, каждый из которых выполняет собственную программу [6].

Способы взаимодействия задач можно классифицировать по степени осведомленности одной задачи о существовании другой [11]:

1. ***Конкуренция.*** Задачи не осведомлены о наличии друг друга (например, процессы разных заданий одного или различных пользователей). Это независимые задачи, не предназначенные для совместной работы. Хотя эти задачи и не работают совместно, ОС должна решать вопросы конкурентного использования ресурсов. Например, два независимых приложения могут затребовать доступ к одному и тому же диску или принтеру. ОС должна регулировать такие обращения.

2. ***Сотрудничество с использованием разделения.*** Задачи косвенно осведомлены о наличии друг друга (например, потоки одного задания). Эти задачи не обязательно должны быть осведомлены о наличии друг друга с высокой точностью, однако они разделяют доступ к некоторому объекту, например буферу ввода-вывода, к файлу или базе данных. Такие задачи демонстрируют сотрудничество при разделении общего объекта.

3. ***Сотрудничество с использованием связи.*** Задачи непосредственно осведомлены о наличии друг друга (например, по-

токи, работающие последовательно или поочередно в рамках одного задания). Такие задачи способны общаться один с другим, с использованием идентификаторов процессов, и изначально созданы для совместной работы. Эти задачи также демонстрируют сотрудничество при работе.

В работе Д. Бэкона и Т. Харриса [5] сформулированы ***пять условий, которые должны выполняться для хорошего программного алгоритма организации взаимодействия процессов, имеющих критические участки*** (процессы могут проходить их в произвольном порядке):

1. Задача должна быть решена чисто программным способом на обычной машине, не имеющей специальных команд взаимоисключения. При этом предполагается, что основные инструкции языка программирования являются атомарными операциями.

2. Не должно существовать никаких предположений об относительных скоростях выполняющихся процессов или числе процессоров, на которых они исполняются.

3. Если процесс исполняется в своем критическом участке, то не существует никаких других процессов, которые исполняются в своих соответствующих критических секциях. Это условие получило название условия взаимоисключения.

4. Процессы, которые находятся вне своих критических участков и не собираются входить в них, не могут препятствовать другим процессам входить в их собственные критические участки. Если нет процессов в критических секциях и имеются процессы, желающие войти в них, то только те процессы, которые не исполняются в остаточных секциях, должны принимать решение о том, какой процесс войдет в свою критическую секцию. Такое решение не должно приниматься бесконечно долго.

5. Не должно возникать бесконечного ожидания для входа процесса в свой критический участок. От того момента, когда процесс запросил разрешение на вход в критическую секцию, и до момента, когда он это разрешение получил, другие процессы могут пройти через свои критические участки лишь ограниченное число раз.

Описание соответствующего алгоритма в данном случае означает описание способа организации пролога и эпилога для

критической секции. Критический участок должен сопровождаться прологом и эпилогом, которые не имеют отношения к активности одиночного процесса. Во время выполнения пролога процесс должен получить разрешение на вход в критический участок, а во время выполнения эпилога — сообщить другим процессам, что он покинул критическую секцию.

1.4.2 Реализация взаимоисключений

Для **реализации взаимоисключений** существует несколько способов, приведем некоторые из них:

– ***Запрет прерываний.*** Наиболее простой способ, вокруг критической секции создаются пролог и эпилог с функцией запрета прерываний. Поскольку выход процесса из состояния исполнения без его завершения осуществляется по прерыванию, то внутри критической секции никто не может вмешаться в его работу. Таким образом, запретив прерывание, процесс может считывать и сохранять совместно используемые данные, не опасаясь вмешательства другого процесса. Однако этот способ практически не применяется, так как опасно доверять управление системой пользовательскому процессу — он может надолго занять процессор, а результат сбоя в критической ситуации может привести к краху ОС, а следовательно, и всей системы. Кроме того, нужного результата можно не достичь в многопроцессорной системе, так как запрет прерываний будет относиться только к одному процессору, а остальные процессоры продолжают работать и сохранять доступ к разделенным данным [6].

– ***Введение блокирующих переменных.*** Для синхронизации потоков программист может использовать глобальные блокирующие переменные. С этими переменными можно работать, не обращаясь к системным вызовам. К каждому набору критических данных ставится в соответствие двоичная переменная. Поток может войти в критическую секцию только тогда, когда значение этой переменной равно 0, одновременно изменяя ее значение на 1 — закрывая доступ. При выходе из критической секции поток сбрасывает ее значение в 0 — открывая доступ. К сожалению, такое решение не удовлетворяет условию взаимоисключе-

ния. Например, если поток протестировал значение переменной и принял решение о входе в критическую секцию. И в этот момент времени, еще до закрытия доступа, планировщик передал управление другому потоку, который также проверяет переменную доступа и тоже принимает решение о входе в критическую секцию. В результате получаем два процесса, одновременно выполняющихся в критических секциях [6].

– ***Строгое чередование.*** В данном подходе также используется общая переменная для двух потоков с начальным значением 0.Только теперь она играет не роль замка для критической секции, а явно указывает, какой поток может следующим войти в нее. Отработавший поток меняет значение переменной для входа следующего потока. При таком подходе взаимоисключение гарантируется, но нет возможности добавить еще один поток.

– ***Установка флагов готовности.*** Недостаток предыдущего алгоритма заключается в том, что процессы ничего не знают о состоянии друг друга в текущий момент времени. Исправить эту ситуацию можно путем введения массива флагов. Пусть два процесса имеют разделяемый массив флагов готовности входа процессов в критический участок:

shared int ready{2} = {0, 0}.

Когда i-й процесс готов войти в критическую секцию, он присваивает элементу массива ready{i} значение, равное 1. После выхода из критической секции он сбрасывает это значение в 0. Процесс не входит в критическую секцию, если другой процесс уже готов к входу в критическую секцию или находится в ней.

Полученный алгоритм обеспечивает взаимоисключение, позволяет процессу, готовому к входу в критический участок, войти в него сразу после завершения эпилога в другом процессе, но все равно при нем может возникнуть тупиковая ситуация (deadlock). Пусть процессы практически одновременно подошли к выполнению пролога. После выполнения присваивания ready[0] = 1 планировщик передал процессор от процесса 0 процессу 1, который также выполнил присваивание ready[1] = 1. После этого оба процесса бесконечно долго ждут друг друга на входе в критическую секцию [6].

– ***Реализация алгоритма Петерсона.*** Первое решение проблемы, удовлетворяющее всем требованиям и использующее идеи ранее рассмотренных алгоритмов, было предложено датским математиком Деккером (Dekker). В 1981 году Петерсон (Peterson) предложил более изящное решение. Оба процесса имеют доступ к массиву флагов готовности и к переменной очередности[6]:

```
shared int ready [2] = {0, 0};
shared int turn;
while (some condition)&&
{
  ready[i]=1;
  turn=1-I;
  while(ready[1-i]&&turn==1-i);
  critical section
  ready[i]=0;
  remainder section
}
```

При исполнении пролога критической секции процесс Pi заявляет о своей готовности выполнить критический участок и одновременно предлагает другому процессу приступить к его выполнению. Если оба процесса подошли к прологу практически одновременно, то они оба объявят о своей готовности и предложат выполняться друг другу. При этом одно из предложений всегда последует после другого. Тем самым работу в критическом участке продолжит процесс, которому было сделано последнее предложение.

– ***Реализация алгоритма булочной (Bakery algorithm).*** Алгоритм Петерсона дает решение задачи корректной организации взаимодействия двух процессов. Алгоритм булочной предполагает реализацию для N процессов. Основная его идея выглядит так. Каждый вновь прибывающий клиент (он же процесс) получает талончик на обслуживание с номером. Клиент с наименьшим номером на талончике обслуживается следующим. К сожалению, из-за неатомарности операции вычисления следующего номера алгоритм булочной не гарантирует, что у всех процессов будут талончики с разными номерами. В случае ра-

венства номеров на талончиках у двух или более клиентов первым обслуживается клиент с меньшим значением имени (имена можно сравнивать в лексикографическом порядке) [6].

Наличие аппаратной поддержки взаимоисключений позволяет упростить алгоритмы и повысить их эффективность точно так же, как это происходит и в других областях программирования. Многие вычислительные системы имеют специальные команды процессора, которые позволяют проверить и изменить значение машинного слова (Test-and-Set) или поменять местами значения двух машинных слов в памяти (Swap), выполняя эти действия как атомарные операции [6].

1.4.3 Механизмы синхронизации процессов

Помимо вышеперечисленных недостатков у алгоритмов, реализующих взаимоисключения, построенных средствами общих языков программирования, имеются и другие серьезные недостатки.

Допустим, что в вычислительной системе находятся два действующих процесса: один из них — **H** — с высоким приоритетом, другой — **L** — с низким приоритетом.

Пусть планировщик устроен так, что процесс с высоким приоритетом вытесняет низкоприоритетный процесс всякий раз, когда он готов к исполнению, и занимает процессор на все время своего выполнения (если не появится процесс с еще большим приоритетом). Тогда в случае, когда процесс **L** находится в своей критической секции, а процесс **H**, получив процессор, подошел к входу в критическую область, получаем тупиковую ситуацию. Процесс **H** не может войти в критическую область, находясь в цикле, а процесс **L** не получает управления, чтобы покинуть критический участок.

Одним из первых механизмов, предложенных для синхронизации поведения процессов, стали семафоры, концепцию которых описал Дейкстра (Dijkstra) в 1968 году в работе, посвященной архитектуре маленькой, но тщательно структурированной операционной системы THE.

Семафоры Дейкстры

Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции: P (от датского слова proberen — проверять) и V (от verhogen — увеличивать). Классическое определение этих операций выглядит следующим образом[6]:

P(S): пока S == 0 процесс блокируется;
S = S - 1;
V(S): S = S + 1;

Эта запись означает следующее: при выполнении операции P над семафором S сначала проверяется его значение. Если оно больше 0, то из S вычитается 1. Если оно меньше или равно 0, то процесс блокируется до тех пор, пока S не станет больше 0, после чего из S вычитается 1. При выполнении операции V над семафором S к его значению просто прибавляется 1.

Подобные переменные-семафоры могут быть с успехом применены для решения различных задач организации взаимодействия процессов. В ряде языков программирования они были непосредственно введены в синтаксис языка (например, в ALGOL-68), в других случаях применяются через использование системных вызовов. Соответствующая целая переменная располагается внутри адресного пространства ядра операционной системы. Операционная система обеспечивает атомарность операций P и V, используя, например, метод запрета прерываний на время выполнения соответствующих системных вызовов. Если при выполнении операции P заблокированными оказались несколько процессов, то порядок их разблокирования может быть произвольным, например FIFO.

Мониторы Хоара и Хансена

Для того чтобы облегчить труд программистов по анализу правильности использования семафоров, в 1974 году Хоаром были предложены механизмы более высокого уровня, получившие названия мониторов [6].

Мониторы представляют собой тип данных, который может быть внедрен в объектно-ориентированные языки программирования. Монитор обладает своими собственными переменными, определяющими его состояние. Значения этих переменных извне монитора могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в своей работе только данные, находящиеся внутри монитора, и свои параметры.

Важной особенностью мониторов является то, что в любой момент времени только один процесс может быть активен, т.е. находиться в состоянии «готовность» или «исполнение», внутри данного монитора. Поскольку мониторы представляют собой особые конструкции языка программирования, то компилятор может отличить вызов функции, принадлежащей монитору, от вызовов других функций и обработать его специальным образом, добавив к нему пролог и эпилог, реализующие взаимоисключение. Так как обязанность конструирования механизма взаимоисключений возложена на компилятор, а не на программиста, работа программиста при использовании мониторов существенно упрощается, а вероятность появления ошибок становится меньше.

Однако одних только взаимоисключений недостаточно для того, чтобы в полном объеме реализовать решение задач, возникающих при взаимодействии процессов. Нужны еще и средства организации очередности процессов. Для этого в мониторах было введено понятие условных переменных (condition variables), над которыми можно совершать две операции wait и signal, до некоторой степени похожие на операции P и V над семафорами.

Если функция монитора не может выполняться дальше, пока не наступит некоторое событие, она выполняет операцию wait над какой-либо условной переменной. При этом процесс, выполнивший операцию wait, блокируется, становится неактивным и другой процесс получает возможность войти в монитор.

Когда ожидаемое событие происходит, другой процесс внутри функции-метода совершает операцию signal над той же самой условной переменной. Это приводит к пробуждению ра-

нее заблокированного процесса, и он становится активным. Если несколько процессов дождались операции signal для этой переменной, то активным становится только один из них. Что нужно предпринять для того, чтобы не оказалось двух одновременно активных процессов (разбудившего и пробужденного) внутри монитора? Хоар предложил, чтобы пробужденный процесс подавлял исполнение разбудившего процесса, пока он сам не покинет монитор. Несколько позже Хансен предложил другой механизм: разбудивший процесс покидает монитор немедленно после исполнения операции signal.

1.4.4 Взаимоблокировки (тупики)

Коффман и другие исследователи доказали, что ***для возникновения тупиковой ситуации должны выполняться четыре условия*** [12]:

1. Условие взаимного исключения. Каждый ресурс в данный момент отдан ровно одному процессу или доступен.
2. Условие удерживания и ожидания. Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.
3. Условие отсутствия принудительной выгрузки ресурсов. У процесса нельзя забрать принудительно ранее полученные ресурсы. Процесс, владеющий ими, должен сам освободить ресурсы.
4. Условие циклического ожидания. Должна существовать круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

Для того чтобы произошла взаимоблокировка, должны выполняться все эти четыре условия. Если хотя бы одно отсутствует, тупиковая ситуация невозможна. ***При столкновении с взаимоблокировками используются четыре стратегии*** [6]:

1. Пренебрежение проблемой в целом.
2. Обнаружение и восстановление. Позволить взаимоблокировке произойти, обнаружить ее и предпринять какие-либо действия.

3. Избегание тупиковых ситуаций с помощью аккуратного распределения ресурсов.

4. Предотвращение с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки.

Если взаимоблокировки случаются в среднем раз в пять лет, а сбои ОС, компиляторов и неисправности аппаратуры происходят в среднем один раз в неделю, то подходит первая стратегия. К этому надо добавить, что большинство операционных систем потенциально страдает от взаимоблокировок, которые не обнаруживаются, не говоря уже об автоматическом выходе из тупика.

Вторая стратегия представляет собой обнаружение и восстановление. При использовании этого метода система не пытается предотвратить попадание в тупиковые ситуации. Вместо этого она позволяет произойти взаимоблокировке, старается определить, когда это случилось, и затем совершает некоторые действия по возврату системы к состоянию, имевшему место до того, как система попала в тупик.

Если все же был обнаружен тупик. Существуют методы, которые можно использовать для его ликвидации [6]:

– ***Первый*** — принудительная выгрузка ресурсов. Способность забирать ресурс у процесса, отдавать его другому процессу, а затем возвращать назад так, что исходный процесс не замечает того, что в значительной мере зависит от свойств ресурса. Выйти из тупика, таким образом, зачастую трудно или невозможно.

– ***Второй подход*** — восстановление через откат. В этом способе процессы должны периодически создавать контрольные точки, позволяющие запустить процесс с его предыстории. Когда взаимоблокировка обнаружена, достаточно просто понять, какие ресурсы нужны процессам. Чтобы выйти из тупика, процесс, занимающий необходимый ресурс, откатывается к тому моменту времени, перед которым он получил данный ресурс, для чего запускается одна из его контрольных точек. Вся работа, выполненная после этой контрольной точки, теряется. Если возобновленный процесс вновь пытается получить данный ресурс, ему придется ждать, когда ресурс станет доступным.

– ***Третий подход*** — восстановление путем уничтожения одного или более процессов. Это грубейший, но простейший выход из тупика. Проблема, какой процесс уничтожать.

Идеальной была бы такая организация вычислительного процесса, при которой не возникали бы тупики за счет безопасного распределения ресурсов. Подобные алгоритмы базируются на концепции безопасных состояний.

1.4.5 Синхронизация потоков, принадлежащих разным процессам

Рассмотренные способы синхронизации, основанные на глобальных переменных процесса, обладают существенным недостатком — они не подходят для синхронизации потоков различных процессов. В таких случаях ОС должна предоставлять потокам системные объекты синхронизации, которые были бы видны для всех потоков, даже если они принадлежат разным процессам и работают в разных адресных пространствах.

Примерами таких синхронизирующих объектов являются системные семафоры, мьютексы, события, таймеры и др. Набор таких объектов определяется конкретной ОС. Чтобы разные процессы могли разделять синхронизирующие объекты, используются различные методы. Некоторые ОС возвращают указатель на объект. Этот указатель может быть доступен всем родственным процессам, наследующим характеристики общего родительского процесса. В других ОС процессы в запросах на создание объектов синхронизации указывают имена, которые должны им быть присвоены. Далее эти имена используются различными процессами для манипуляций объектами синхронизации. В этом случае работа с синхронизирующими объектами подобна работе с файлами. Их можно создавать, открывать, закрывать, уничтожать.

Для синхронизации могут быть использованы такие объекты ОС, как файлы, процессы и потоки. Все эти объекты могут находиться в двух состояниях: ***сигнальном и несигнальном*** — ***свободном***. Смысл, вкладываемый в понятие «сигнальное состояние», зависит от типа объекта. Так, например, поток переходит в сигнальное состояние, когда он завершается. Процесс переходит в сигнальное состояние, когда завершились все его потоки.

Файл переходит в сигнальное состояние, когда завершается операция ввода-вывода для этого файла. Для остальных объектов сигнальное состояние устанавливается в результате выполнения специальных системных вызовов. Приостановка и активизация потоков осуществляются в зависимости от состояния синхронизирующих объектов ОС.

Потоки с помощью специального системного вызова — Wait(X), где X — указатель на объект синхронизации, — сообщают операционной системе о том, что они хотят синхронизировать свое выполнение с состоянием объекта X. Системный вызов, с помощью которого поток может перевести объект синхронизации в сигнальное состояние, назовем Set(X).

Поток, выполнявший системный вызов Wait(X), переводится операционной системой в состояние ожидания до тех пор, пока объект X не перейдет в сигнальное состояние. Поток может ждать сигнального состояния не одного объекта, а нескольких. Может случиться, что установки некоторого объекта в сигнальное состояние ожидают несколько потоков.

Синхронизация тесно связана с планированием потоков. ***Во-первых***, любое обращение потока к системному вызову Wait(X) приводит к переводу его в очередь ожидающих потоков, а из очереди готовых потоков выбирается и активизируется новый поток

Во-вторых, при переходе объекта в сигнальное состояние (в результате выполнения некоторого потока — системного или прикладного) ожидающий этот объект поток переводится в очередь готовых к выполнению потоков. Таким образом, в обоих случаях происходит перепланирование потоков, в том числе изменение их приоритетов и квантов времени, если это предусмотрено в ОС.

Круг событий, с которыми потоку может потребоваться синхронизировать свое выполнение, не исчерпывается завершением потока, процесса или операции ввода-вывода. Рассмотрим более подробно различные объекты синхронизации:

Мьютекс (mutex — сокращение от mutual exclusion — взаимное исключение) — упрощенный семафор, неспособный считать, он может управлять лишь взаимным исключением доступа к совместно используемым ресурсам или кодам. Реализация

мьютекса полезна в случае потоков, действующих только в пространстве пользователя.

Объект **событие** обычно используется не для доступа к данным, а для того чтобы оповестить другие потоки о том, что некоторые действия завершены.

Сигналы дают возможность задаче реагировать на событие, источником которого может быть ОС или другая задача. Синхронные сигналы чаще всего приходят от системы прерывания процессора и свидетельствуют о действиях процесса, блокируемых аппаратурой, например деление на нуль, ошибка адресации, нарушение защиты памяти и т.д. Примером асинхронного сигнала является сигнал с терминала. Во многих ОС предусматриваются оперативное снятие процесса с выполнения (Ctrl+Break) для выработки сигнала ОС и направление его процессу.

Обработка сигналов аналогична обработке аппаратных прерываний ввода-вывода. Сигналы обеспечивают логическую связь между процессами, а также между процессами и пользователями (терминалами). Поскольку посылка сигнала предусматривает знание идентификатора процесса, то взаимодействие посредством сигналов возможно только для членов группы, состоящей из родительского и дочерних процессов. Процесс может послать сигнал всей своей группе за один системный вызов.

Другим средством взаимодействия процессов является **канал (труба)** — псевдофайл, который может использоваться для связи двух процессов. Когда процесс А хочет отправить данные процессу В, он пишет их канал, как если бы это был выходной файл. Процесс В читает данные из канала, как если бы он был входным файлом. Подобное средство взаимодействия используется в операционной системе UNIX.

Почтовые ящики, используемые в Windows 2000 и выше, в некоторых аспектах подобны каналам, однако, в отличие от каналов, являются однонаправленными. Они позволяют отправляющему процессу использовать широковещание для рассылки сообщений сразу многим получателям.

Для прямой и непрямой адресации достаточно двух примитивов, чтобы описать передачу сообщений по линии связи — send и receive. В случае прямой адресации их можно обозначать так:

– send(P, message) — послать сообщение message процессу P;

– receive(Q, message) — получить сообщение message от процесса Q.

В случае непрямой адресации мы будем обозначать их так:

– send(A, message) — послать сообщение message в почтовый ящик A;

– receive(A, message) — получить сообщение message из почтового ящика A.

Примитивы send и receive уже имеют скрытый от наших глаз механизм взаимоисключения. Более того, в большинстве систем они уже имеют и скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Надо отметить, что, несмотря на простоту использования, передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.

Сокеты (ОС Windows 2000 и выше) подобны каналам, с тем отличием, что они при нормальном использовании соединяют процессы на разных компьютерах. Например, один процесс пишет в сокет, а другой на удаленной машине читает из него. В принципе сокеты можно использовать для соединения процессов на одной машине, но это связано с большими накладными расходами.

Вызов удаленной процедуры (Remote Procedure Call, RPC) представляет собой способ, которым процесс A просит процесс B вызвать процедуру в адресном пространстве процесса B от имени процесса A и вернуть результат процессу A.

Наконец, процессы могут **совместно использовать память** для одновременного отображения одного и того же файла. Все, что один процесс будет писать в этот файл, будет появляться в адресном пространстве других процессов. С помощью такого механизма легко реализовать общий буфер, применяемый в задаче производителя и потребителя. Запись в этот файл одним из процессов мгновенно становится видной остальным.

1.5 Прерывания

Прерывания представляют собой механизм, позволяющий координировать параллельное функционирование отдельных устройств вычислительной системы и реагировать на особые состояния, возникающие при работе процессора [3]. 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 [3]:

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 доста-

точно 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. 1.7). 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.

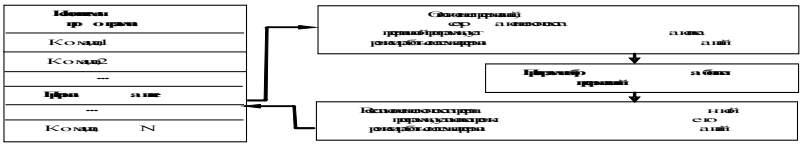

Рис. 1.7 — Обработка прерываний

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 маски (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:

1. Супервизор прерываний, прежде всего, сохраняет в дескрипторе текущей задачи рабочие регистры процессора, определяющие контекст прерываемого вычислительного процесса.

2. Далее он определяет ту подпрограмму, которая должна выполнить действия, связанные с обслуживанием настоящего (текущего) запроса на прерывание.

3. Наконец, перед тем как передать управление этой подпрограмме, супервизор прерываний устанавливает необходимый режим обработки прерывания.

4. После выполнения подпрограммы обработки прерывания управление вновь передается супервизору, на этот раз уже на тот модуль, который занимается диспетчеризацией задач.

5. 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. 1.8).

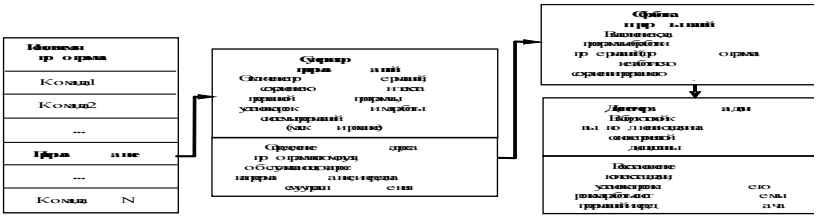

Рис. 1.8 — Обработка прерываний при участии супервизоров ОС

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».

Однако если бы контекст процессов сохранялся просто в стеке, как это обычно реализуется аппаратурой, а не в описанных выше дескрипторах задач, то 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 дескрипторе, то есть

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

1.6 Управление задачами в ОС Windows

1.6.1 Информация об организации вычислительных задач

Современные операционные системы содержат встроенные средства, предоставляющие информацию о компонентах вычислительного процесса. Диспетчер задач (Task Manager) операционных систем Windows (например, Windows XP) позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до выполняющихся прикладных программ (приложений) и процессов. Однако диспетчер задач не позволяет отслеживать потоки.

Для запуска диспетчера задач и просмотра компонентов вычислительного процесса нужно выполнить следующие действия [13]:

1. Щелкнуть правой кнопкой мыши по панели задач и выбрать строку «Диспетчер задач», или нажать клавиши Ctrl+Alt+Del, или нажать последовательно Пуск → Выполнить → taskmgr (рис. 1.9).

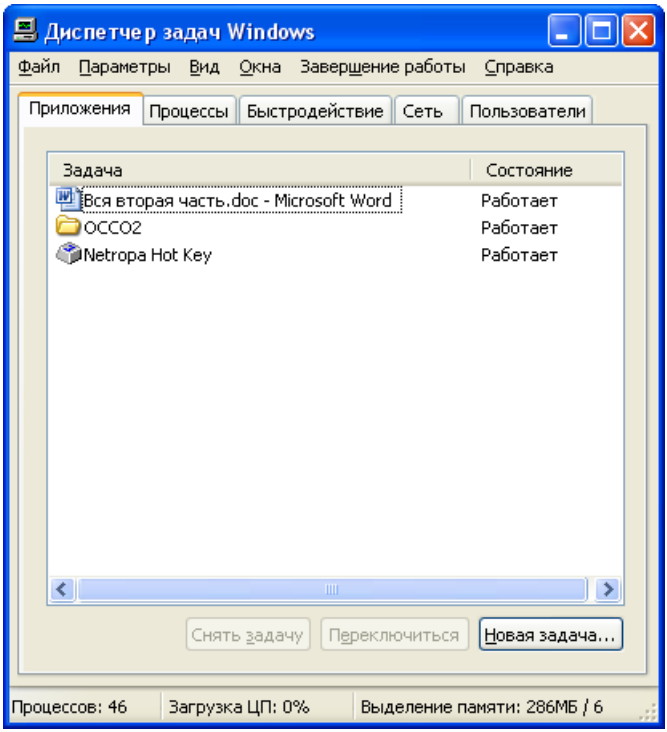

Рис. 1.9 — Окно диспетчера задач в ОС Windows XP

2. Для просмотра приложений перейти на вкладку «Приложения». Здесь можно завершить приложение (кнопка «Снять задачу»), переключиться на другое приложение (кнопка «Переключиться») и создать новую задачу (кнопка «Новая задача»). В последнем случае после нажатия кнопки «Новая задача» в появившемся окне (рис. 1.10) нужно ввести имя задачи.

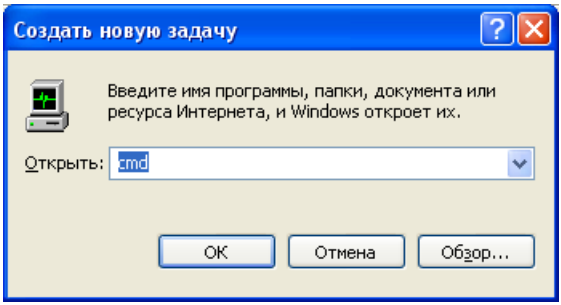

Рис. 1.10 — Окно создания новой задачи в ОС Windows XP

3. Просмотр (мониторинг) процессов осуществляется переходом на вкладку «Процессы». Таблица процессов включает в себя все процессы, запущенные в собственном адресном пространстве, в том числе все приложения и системные сервисы. Обратите внимание на процесс «Бездействие системы» — фиктивный процесс, занимающий процессор при простое системы.

4. Если требуется просмотреть 16-разрядные процессы, то в меню «Параметры» необходимо выбрать команду «Отображать 16-разрядные задачи».

5. Для выбора просматриваемых показателей (характеристик) с помощью команды «Выбрать столбцы» (меню «Вид») необходимо установить флажки рядом с показателями, которые требуется отображать (рис. 1.11).

В качестве примера можно рассмотреть процессы приложения MS Word. Для этого нужно выполнить следующие действия [13]:

1. Запустить MS Word. Щелкнуть правой клавишей мыши по названию приложения и в появившемся контекстном меню выбрать строку «Перейти к процессам». Произойдет переход на вкладку «Процессы». Можно просмотреть число потоков и другие характеристики процесса.

2. Изменить приоритет процесса. На вкладке «Процессы» необходимо щелкнуть правой клавишей мыши по названию процесса и выбрать в контекстном меню строку «Приоритет». Изменив приоритет, можно увидеть в колонке «Базовый приоритет» его новое значение.

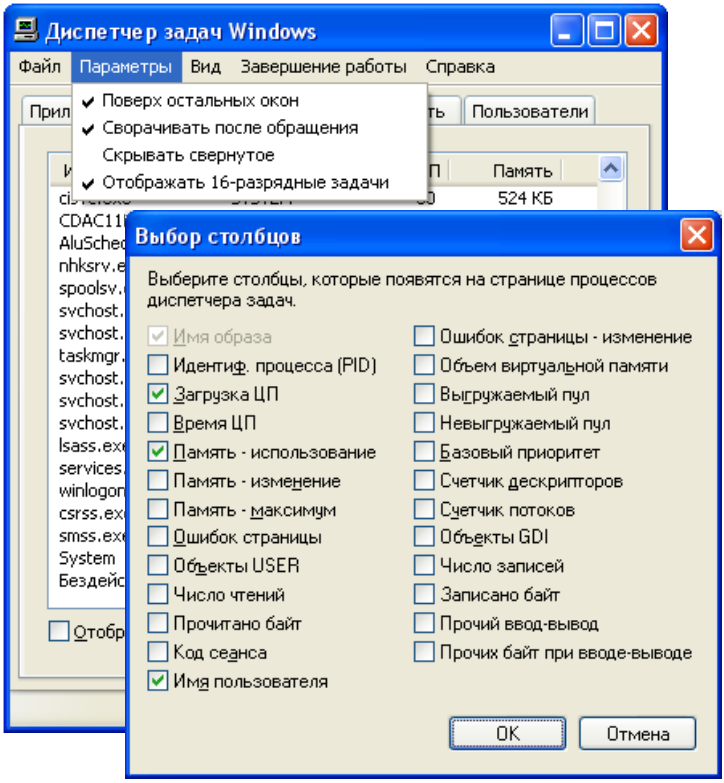

Рис. 1.11 — Окно диспетчера задач в ОС Windows XP
на вкладке «Процессы» с окном настройки отображения столбцов

3. Изменить скорости обновления данных. Войти в меню «Вид» и выбрать команду «Скорость обновления». Установить требуемую скорость (высокая — каждые полсекунды, обычная — каждую секунду, низкая — каждые 4 секунды, приостановить — обновления нет). Следует иметь в виду, что с повышением скорости мониторинга возрастают затраты ресурсов компьютера на работу операционной системы, что, в свою очередь, вносит погрешность в результаты мониторинга.

Диспетчер задач позволяет получить обобщенную информацию об использовании основных ресурсов компьютера. Для этого необходимо сделать следующее [13]:

1. Перейти на вкладку «Быстродействие» (рис. 1.12). Верхние два окна показывают интегральную загрузку процессора и хронологию загрузки. Нижние два окна — те же показатели, но по использованию памяти.

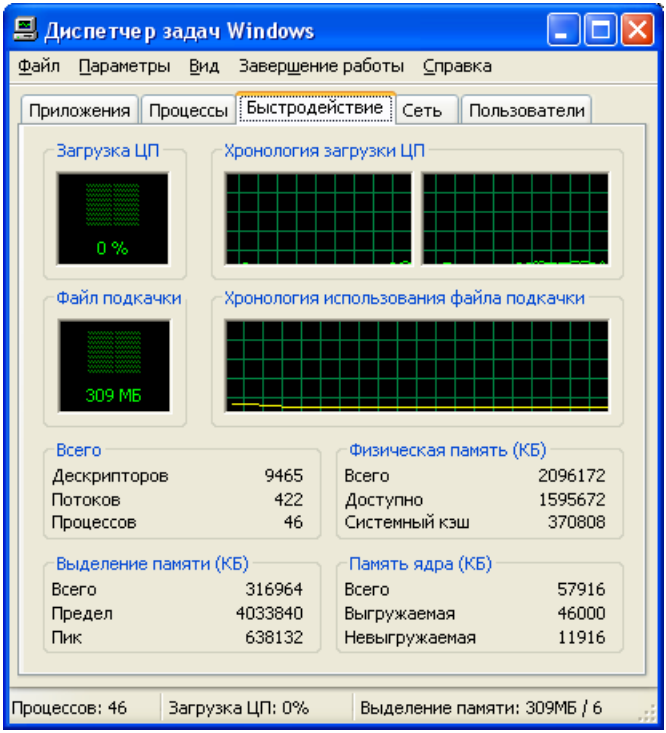

Рис. 1.12 — Окно диспетчера задач в ОС Windows XP на вкладке «Быстродействие»

2. Для просмотра использования процессора в режиме ядра (красный цвет) войти в меню «Вид» и щелкнуть на строке Вывод времени ядра.

В нижней части окна вкладки «Быстродействие» отображается информация о количестве процессов и потоков, участвующих в мультипрограммном вычислительном процессе, об общем количестве дескрипторов (описателей) объектов, созданных опе-

рационной системой, а также информация о доступной и выделенной памяти для реализации приложений. Кроме того, приводятся сведения о выделении памяти под ядро операционной системы с указанием выгружаемой и невыгружаемой памяти ядра и объеме системного кэша.

Также в диспетчере задач имеются вкладки для отображения состояния сети (вкладка «Сеть») и информации о вошедших в систему пользователях (вкладка «Пользователи»).

Ряд программ как производителей операционных систем, так и сторонних производителей могут предоставить более детальную информацию о компонентах вычислительного процесса и механизмы управления им: Process Explorer, Process Viewer, Microsoft Spy++, CPU Stress, Scheduling Lab, Job Lab и др.

На рис. 1.13 показано окно с получением информации о потоках в программе Process Explorer. В данной программе можно получить исчерпывающую информацию о количестве и состоянии задач в операционной системе Windows.

Любой поток состоит из двух компонентов [13]:

– объекта ядра, через который операционная система управляет потоком. Там же хранится статистическая информация о потоке;

– стека потока, который содержит параметры всех функций и локальные переменные, необходимые потоку для выполнения кода.

Создав объект ядра «поток», система присваивает счетчику числа его пользователей начальное значение, равное двум. Затем система выделяет стеку потока память из адресного пространства процесса (по умолчанию резервирует 1 Мбайт адресного пространства процесса и передает ему всего две страницы памяти, далее память может добавляться). После этого система записывает в верхнюю часть стека два значения (стеки строятся от старших адресов памяти к младшим). Первое из них является значением параметра **pvParam**, который позволяет передать функции потока какое-либо инициализирующее значение. Второе значение определяет адрес функции потока **pfnStartAddr**, с которой должен будет начать работу создаваемый поток.

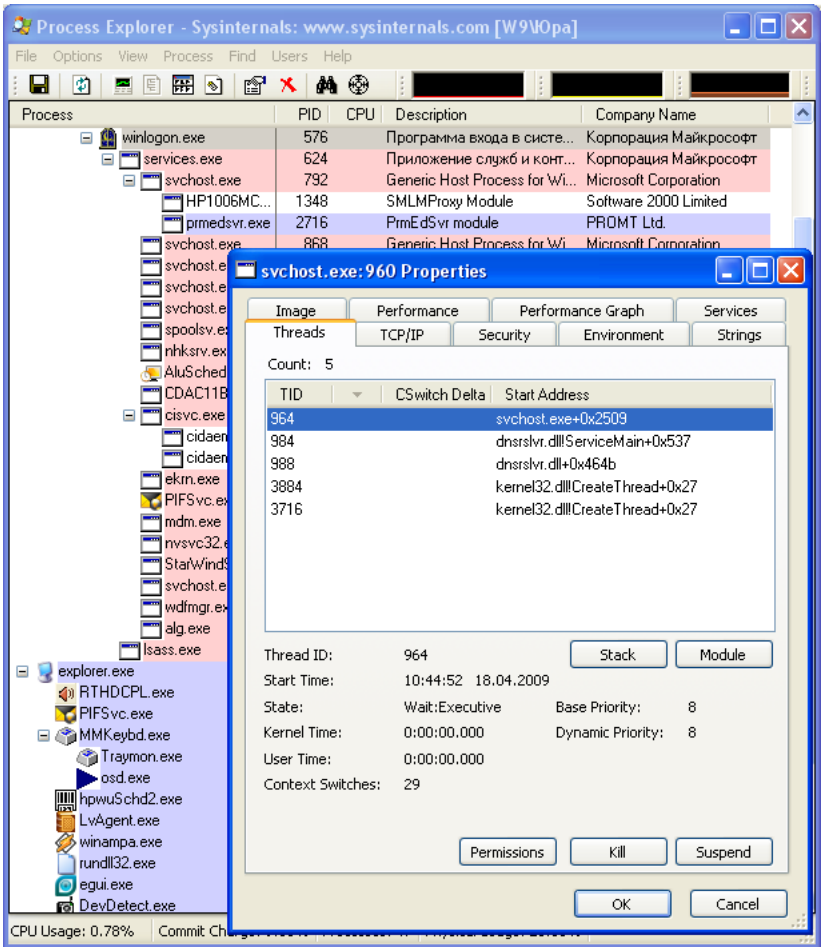

Рис. 1.13 — Окно с информацией о потоках
в программе Process Explorer

У каждого потока собственный набор регистров процессора, называемый контекстом потока. Контекст отображает состояние регистров процессора на момент последнего исполнения потока и записывается в структуру **CONTEXT**, которая содержится в объекте ядра «ПОТОК».

Указатель команд (IP) и указатель стека (SP) — два самых важных регистра в контексте потока. Когда система инициализирует объект ядра «ПОТОК», указателю стека в структуре CONTEXT присваивается тот адрес, по которому в стек потока было записано значение pfnStartAddr, а указателю команд — адрес недокументированной функции **BaseTbreadStart** (находится в модуле Kerne132.dll).

Новый поток начинает выполнение этой функции, в результате чего система обращается к функции потока, передавая ей параметр pvParam. Когда функция потока возвращает управление, BaseTbreadStart вызывает **ExitTbread**, передавая ей значение, возвращенное функцией потока. Счетчик числа пользователей объекта ядра «ПОТОК» уменьшается на 1, и выполнение потока прекращается.

При инициализации первичного потока его указатель команд устанавливается на другую недокументированную функцию — **BaseProcessStart**. Она почти идентична BaseTbreadStart. Единственное различие между этими функциями в отсутствии ссылки на параметр pvParam. Функция BaseProcessStart обращается к стартовому коду библиотеки C/C++/C#, который выполняет необходимую инициализацию, а затем вызывает входную функцию main, wmain, WinMain, Main. Когда входная функция возвращает управление, стартовый код библиотеки C/C++/C# вызывает **ExitProcess** [13].

1.6.2 Исследование производительности

В операционной системах семейства Windows имеются средства, позволяющие детально анализировать вычислительные процессы. К таким средствам относятся **«Системный монитор»** и **«Оповещения и журналы производительности»**. Для доступа к этим средствам нужно выполнить последовательность действий: Пуск → Панель управления → Администрирование → Производительность.

Откроется окно Производительность, содержащее две оснастки: «Системный монитор» и «Оповещения и журналы производительности» (рис. 1.14).

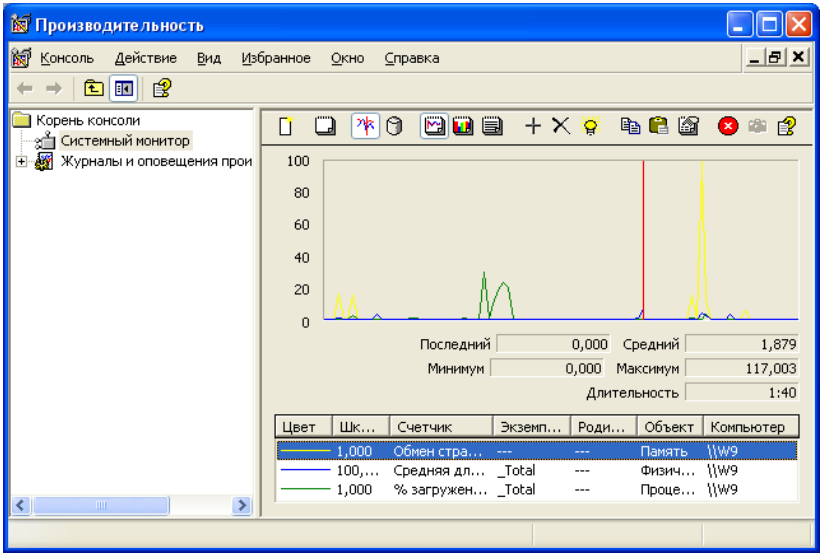

Рис. 1.14 — Окно «Производительность» в ОС Windows XP на вкладке «Быстродействие»

Системный монитор позволяет анализировать вычислительный процесс, используя различные счетчики. Объектами исследования являются практически все компоненты компьютера: процессор, кэш, задание, процесс, поток, физический диск, файл подкачки, очереди сервера, протоколы и др.

Для просмотра и выбора объектов мониторинга и настройки счетчиков нужно выполнить следующие действия:

1. Открыть оснастку «Производительность». По панели результатов (правая панель) щелкнуть правой клавишей мыши и выбрать в контекстном меню строку «Добавить счетчики» или щелкнуть по кнопке «Добавить» (значок +) на панели инструментов.

2. В появившемся окне «Добавить счетчики» (рис. 1.15) выбрать объект мониторинга, например процессор, а затем выбрать нужные счетчики из списка «Выбрать счетчики из списка», например «% времени прерываний», нажимая кнопку Добавить, для потока можно определить:

– число контекстных переключений в сек;
– состояние потока (для построения графа состояний и переходов);
– текущий приоритет (для анализа его изменения);
– базовый приоритет;
– % работы в привилегированном режиме и др.

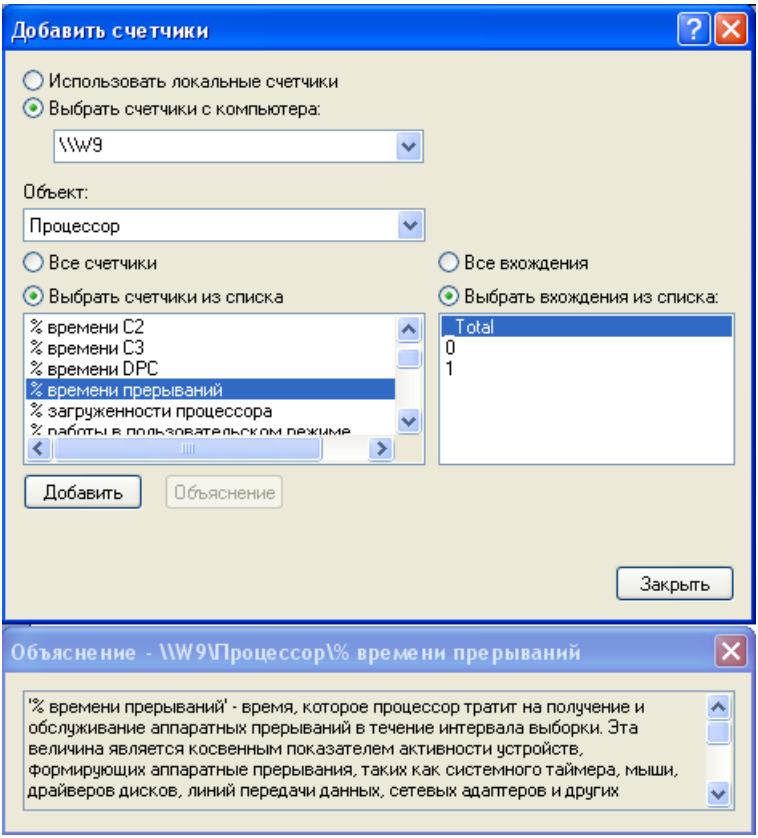

Рис. 1.15 — Окно «Добавить счетчики» в программе оценки производительности в ОС Windows XP

Нажав кнопку «Объяснение», можно получить информацию о счетчике. При выборе нескольких однотипных объектов,

например потоков, нужно их указать в правом поле «Выбрать вхождения из списка».

Для удобства работы предусмотрена настройка вида отображаемой информации.

Просмотр информации производительности возможен в виде графика, гистограммы и отчета. Для настройки внешнего вида окна нужно щелкнуть по графику правой кнопкой мыши и выбрать команду «Свойства».

На вкладке «Общие» можно задать вид информации (график, гистограмма, отчет), отображаемые элементы (легенда, строка значений, панель инструментов), данные отчета и гистограммы (максимальные, минимальные и т.д.), период обновления данных и др.

На вкладке «Источник» задается источник данных. На вкладке «Данные» можно для каждого счетчика задать цвет, ширину линии, масштаб и др.

На вкладке «График» можно задать заголовок, вертикальную и горизонтальную сетку, диапазон значений вертикальной шкалы. На вкладках «Цвета и шрифты» можно изменить набор цветов и шрифт.

Режимы «График» и «Гистограмма» не всегда удобны для отображения результатов анализа, например при большом количестве счетчиков, меняющих свое значение в разных диапазонах величин. Режим «Отчет» позволяет наблюдать реальные значения счетчиков, так как не использует масштабирующих множителей. В этом режиме доступна только одна опция — изменение интервала опроса.

Полученная с помощью «Монитора производительности» информация позволяет наглядно произвести экспресс-анализ функционирования нужного компонента вычислительного процесса или устройства компьютера.

Оснастка «Оповещения и журналы производительности» содержит три компонента:

Журналы счетчиков, Журналы трассировки и Оповещения — которые можно использовать для записи и просмотра результатов исследования вычислительного процесса. Данные, созданные при помощи оснастки, можно просматривать как в процессе сбора, так и после его окончания.

Файл журнала счетчиков состоит из данных для каждого указанного счетчика на указанном временном интервале. Для создания журнала необходимо выполнить следующие действия [13]:

1) запустить оснастку «Производительность»;

2) дважды щелкнуть по значку «Оповещения и журналы производительности»;

3) выбрать значок «Журналы счетчиков», щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры журнала»;

4) в открывшемся окне ввести произвольное имя журнала и нажать кнопку «ОК»;

5) в новом окне на вкладке «Общие» добавить нужные счетчики и установить интервал съема данных;

6) на вкладке «Файлы» журналов можно выбрать размещение журнала, имя файла, добавить комментарий, указать тип журнала и ограничить его объем. Возможны следующие варианты:

– текстовый файл — CVS (данные сохраняются с использованием запятой в качестве разделителя);

– текстовый файл — TSV (данные сохраняются с использованием табуляции в качестве разделителя);

– двоичный файл для регистрации прерывающейся информации;

– двоичный циклический файл для регистрации данных с перезаписью;

7) на вкладке «Расписание» выбрать режим запуска и остановки журнала (вручную или по времени). Для запуска команды после закрытия журнала установить флажок «Выполнить команду» и указать путь к исполняемому файлу;

8) после установки всех значений нажать кнопки «Применить» и «ОК».

В отличие от журналов счетчиков журналы трассировки находятся в ожидании определенных событий. Для интерпретации содержимого журнала трассировки необходимо использовать специальный анализатор.

Для создания журнала трассировки необходимо выполнить следующие действия:

1) запустить оснастку «Производительность»;

2) щелкнуть по значку «Журналы трассировки»;

3) щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры журнала»;

4) в открывшемся окне ввести произвольное имя журнала и нажать кнопку «ОК»; по умолчанию файл журнала создается в папке PerfLogs в корневом каталоге и к имени журнала присоединяется серийный номер;

5) на вкладке «Общие» указать путь и имя созданного журнала (по умолчанию оно уже есть);

6) на этой же вкладке выбрать «События», протоколируемые системным поставщиком или указать другого поставщика;

7) на вкладке «Файлы журналов» выбрать тип журнала:

– файл циклической трассировки (журнал с перезаписью событий, расширение etl);

– файл последовательной трассировки (данные записываются, пока журнал не достигнет предельного размера, расширение etl);

8) на этой же вкладке выбрать и размер файла;

9) на вкладке «Дополнительно» можно указать размер буфера журнала;

10) на вкладке «Расписание» выбрать режим запуска и остановки журнала (вручную или по времени).

В ряде случаев для обнаружения неполадок в организации вычислительного процесса удобно использовать оповещения. С помощью этого компонента можно установить оповещения для выбранных счетчиков. При превышении или снижении относительно заданного значения выбранными счетчиками оснастка посредством сервиса «Messenger» оповещает пользователя.

Для создания оповещений необходимо выполнить следующие действия:

1) щелкнуть по значку «Оповещения»;

2) щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры оповещений»;

3) в открывшемся окне ввести произвольное имя оповещения и нажать кнопку «ОК»;

4) в появившемся окне на вкладке «Общие» можно задать комментарий к оповещению и выбрать нужные счетчики;

5) в поле «Оповещать» выбрать предельные значения для счетчиков;

6) в поле «Снимать показания» выбрать период опроса счетчиков;

7) на вкладке «Действие» можно выбрать действие, которое будет происходить при запуске оповещения, например послать сетевое сообщение и указать имя компьютера;

8) на вкладке «Расписание» выбрать режим запуска и остановки наблюдения.

Если в компьютере произойдет событие, предусмотренное в оповещениях, в журнал событий «Приложение» будет сделана соответствующая запись. Для ее просмотра нужно зайти в оснастку «Просмотр событий», где и можно увидеть сведения о событии.

1.6.3 Средства командной строки Windows XP Professional

В ОС Windows XP Professional введен дополнительный набор средств командной строки, составляющий около 30 команд. В таблице 1.2 приведены команды из этого набора, относящиеся как-либо к процессу управления вычислительными задачами.

Таблица 1.2 — **Дополнительные средства командной строки Windows XP для управления задачами**

Команда	Свойства
eventcreate	Позволяет администратору создать особое событие в указанном журнале событий
eventquery	Выводит список событий и их свойств из одного или нескольких журналов событий
eventtriggers	Отображает и настраивает события триггеров на локальных и удаленных компьютерах
logman	Управляет и задает расписания для счетчиков производительности и журнала трассировки событий на локальном или удаленных компьютерах

Окончание табл. 13

Команда	Свойства
openfiles	Запрашивает, отображает и/или отключает открытые файлы
perfmon	Открывает консоль «Производительность» с настройками системного монитора из файлов настройки версии для Windows NT 4.0
relog	Сохраняет данные счетчиков производительности из файлов журналов в файлы других форматов, такие как текстовый-TSV (разделитель — запятая), текстовый-CSV (разделитель — табуляция), двоичный-BIN или SQL
sc	Извлекает и настраивает информацию о службах. Проверяет и отлаживает служебные программы
schtasks	Настраивает выполнение команд и программ через заданные интервалы или в указанное время. Добавляет и удаляет задания из расписания, запускает и останавливает задания по требованию, отображает и изменяет задания в расписании
shutdown	Выключает и перезагружает локальный или удаленный компьютер
systeminfo	Запрашивает основные сведения о конфигурации системы
taskkill	Завершает одно или несколько заданий или процессов
tasklist	Отображает список приложений, служб и код процесса (PID) для всех заданий, выполняющихся в данный момент на локальном или удаленном компьютере
tracerpt	Обрабатывает журналы трассировки событий и данные, поступающие в реальном масштабе времени от поставщиков трассировочных данных, и позволяет создавать отчеты с анализом данных трассировки и файлы CSV (с разделителем запятой) для создающихся событий
typeperf	Производит запись данных счетчика производительности в командное окно или в файл журнала поддерживаемого формата

Вопросы для самопроверки

1. Дайте определение понятию вычислительный процесс.

2. Что понимают под понятием задачи в теории операционных систем?

3. Какую цель преследует определение концепции процесса?

4. Что понимают под понятием «ресурс» в теории операционных систем?

5. Приведите классификацию ресурсов.

6. Дайте определение терминам «мультипрограммирование» и «мультизадачность».

7. Как можно представить общую схему выделения ресурсов?

8. Приведите основные функции ядра операционной системы.

9. Что понимают под понятием «мультипроцессорная обработка»?

10. В каких состояниях может находиться процесс? Приведите граф состояний процесса.

11. Для чего необходим дескриптор процесса?

12. Дайте определение понятиям потоков или тредов.

13. Что обеспечивает многопоточность?

14. Что в себя включает контекст процесса?

15. Какие советы можно дать по использованию потоков при создании приложений?

16. Какие функции выполняют ОС, связанные с управлением задачами?

17. Что подразумевается под задачей планирования вычислительного процесса?

18. Какие существуют виды планирования вычислительного процесса?

19. Какие существуют типы планирования вычислительного процесса?

20. Что понимают под стратегией планирования вычислительного процесса, и какие стратегии планирования вам известны?

21. Приведите классификацию дисциплин диспетчеризации.

22. В чем состоит отличие вытесняющей многозадачности от невытесняющей?

23. Какими параметрами можно оценить качество диспетчеризации?

24. Приведите классификацию способов взаимодействия задач.

25. Какие условия должны выполняться для хорошего программного алгоритма организации взаимодействия процессов, имеющих критические участки?
26. Какие способы существуют для реализации взаимоисключений?
27. Какие существуют механизмы синхронизации процессов?
28. Какие условия должны выполняться для возникновения тупиковой ситуации?
29. Какие методы существуют для ликвидации тупиков?
30. Как можно синхронизировать потоки, принадлежащие разным процессам?
31. Дайте определение понятию «прерывание».
32. Какие шаги включает механизм обработки прерывания независимо от архитектуры процессора?
33. Как происходит обработка прерывания в мультипрограммной ОС?
34. С помощью, каких программных средств можно получить информацию о процессах в ОС Windows XP?
35. Каким средством можно оценить производительность в ОС Windows XP?

2 УПРАВЛЕНИЕ ПАМЯТЬЮ

2.1 Основные понятия

2.1.1 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. Однако подавляющее большинство современных компьютеров основаны именно на указанных принципах, включая и сложные многопроцессорные комплексы, которые можно рассматривать как объединение фон-неймановских машин.

В дополнение к принципам фон-Неймана все современные ЭВМ обладают некоторыми общими и индивидуальными свойствами архитектуры. **К числу общих архитектурных свойств и принципов** можно отнести [14]:

- Принцип хранимой программы. Согласно ему, код программы и ее данные находятся в одном адресном пространстве в оперативной памяти.
- Принцип микропрограммирования. В состав процессора входит блок микропрограммного управления. Этот блок для каждой машинной команды имеет набор действий-сигналов, ко-

торые нужно сгенерировать для физического выполнения требуемой машинной команды.

- Линейное пространство памяти — совокупность ячеек памяти, которым последовательно присваиваются номера (адреса) 0, 1, 2, …
- Последовательное выполнение программ. Процессор выбирает из памяти команды строго последовательно. Для изменения прямолинейного хода выполнения программ необходимо использовать специальные команды (команды условного и безусловного перехода).
- Процессор не различает команды и данные, поэтому важно в программе всегда четко разделять пространство данных и команд.
- Безразличие к целевому назначению данных. Машине все равно, какую логическую нагрузку несут обрабатываемые ею данные.

Перечень индивидуальных свойств и принципов микропроцессоров весьма велик, наиболее значимыми являются [14]:

- Суперскалярная архитектура. Важным элементом архитектуры, появившимся в микропроцессоре Intel 486 (i486), стал конвейер — специальное устройство, при котором выполнение команд в микропроцессоре разбивается на несколько этапов. В i486 они следующие:
  - выборка команд из кэш-памяти или оперативной;
  - декодирование команды;
  - генерация адреса;
  - выполнение операции с помощью АЛУ (арифметико-логическое устройство);
  - запись результата.

Преимуществом такого подхода является то, что очередная команда после ее выборки попадает в блок декодирования. Таким образом, блок выборки свободен и может выбрать следующую команду. В результате на конвейере могут находиться в различной стадии выполнения пять команд. Микропроцессоры, имеющие один конвейер, называются скалярными, а два и более — суперскалярными.

- Раздельное кэширование кода и данных. Кэширование — это способ увеличения быстродействия системы за счет хранения часто используемых данных и кодов в так называемой «кэш-памяти первого уровня», находящейся внутри микропроцессора.
- Предсказание правильного адреса перехода. Под переходом понимают запланированное алгоритмом изменение последовательного характера выполнения программы. Типичная программа на каждые 6—8 команд содержит одну команду перехода. Последствия этого предсказать нетрудно: при наличии конвейера через каждые 6—8 команд его нужно очищать и заполнять заново в соответствии с адресом перехода. Все преимущества конвейеризации теряются. Поэтому в архитектуру Pentium был введен блок предсказания переходов. Вероятность правильного предсказания перехода составляет около 80 %.

2.1.2 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

[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.

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:

– 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 форм 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 таблице 2.1 приведены представления одной тетрады (4 бит) в различных системах исчисления.

Таблица 2.1 — **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

Рассмотрим перевод шестнадцатеричных чисел в десятичные числа.

$$\begin{array}{l} \mathrm{E} \longrightarrow 14 \times 16 = 224 \\ + \\ 6 \longrightarrow \quad 6 \times 1 = \quad 6 \\ \hline \mathrm{E6h} \qquad\qquad = 230 \end{array}$$

Возьмем число «E6h», «E» — это шестнадцатеричная цифра 14, и там 16 таких цифр (по аналогии с 10 для десятичного числа). Таким образом «E6h» — это четырнадцать раз по шестнадцать и шесть единиц. Для отделения разрядов шестнадцатеричного числа можно применять следующие четыре числа.

$$16^3 = 4096$$
$$16^2 = 256$$
$$16^1 = 16$$
$$16^0 = 1$$

Теперь научимся переводить десятичные числа в шестнадцатеричную форму.

$$1069 / 16 = 66 \quad (1069 - 66 * 16 = 13)$$
$$66 / 16 = 4 \quad (66 - 4 \times 16 = 2)$$
$$4 / 16 = 0 \quad (4 - 0 \times 16 = 4)$$

1069 = 42Dh

Для этого 1069 разделим на 16, остаток от деления будет давать нам первую шестнадцатеричную цифру, далее целую часть от деления разделим еще раз на шестнадцать, на этот раз остаток даст нам вторую цифру и т.д. Итоговое число собирается последовательно от последнего остатка отделения к первому.

2.1.3 Иерархия памяти

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 (рис. 2.1). Входящие в ее состав запоминающие устройства различаются емкостью, быстродействием и скоростью.

Самыми быстродействующими являются **регистры центрального процессора**, и они же имеют наименьшую емкость. Обращение к ним происходит при выполнении почти каждой машинной команды.

Кэш-память процессора в целом отличается меньшим объемом, но более высоким быстродействием по сравнению с основной памятью. Она функционирует как буфер между ***процессором*** и ***основной памятью*** и содержит копии областей памяти, которые использовались последними, — их адреса и рас-

положенные по этим адресам данные. Каждый раз, когда процессору требуется обратиться к памяти по заданному адресу, он сначала обращается к кэшу. Если происходит промах кэша, то есть этот адрес в кэше отсутствует, производится обращение к основной памяти. Данные считываются из памяти в указанный в команде регистр процессора, а их копия записывается в кэш. Для большей части программного обеспечения характерна временная локализация доступа, то есть значительная вероятность повторного обращения по одним тем же адресам в течение короткого промежутка времени. Если повторное обращение произойдет достаточно скоро, нужные данные все еще будут в кэше и их не придется считывать из основной памяти. Такая ситуация называется «попаданием в кэш».

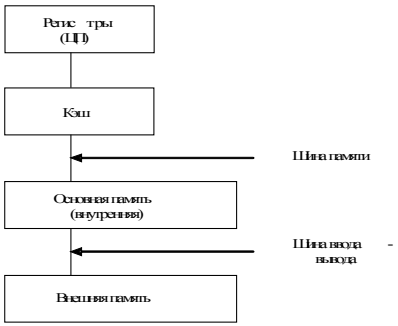

Рис. 2.1 — Иерархия памяти

Передача информации между основной памятью и кэш-памятью осуществляется по шине памяти, которая состоит из шины данных и шины адреса.

В памяти информация хранится в массиве ячеек. Минимальной адресуемой единицей является байт — каждый байт памяти имеет свой уникальный адрес.

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). 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 [15]:

– **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.

Информация из внешней памяти во внутреннюю память попадает по шине ввода-вывода.

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 (ОЗУ — оперативно запоминающее устройство),* информация в которой может изменяться процессором в любой момент времени, и *постоянную (ПЗУ — постоянно запоминающее устройство),* информацию, которую 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)***.

2.2 Программная модель микропроцессора Intel Pentium

2.2.1 Состав программной модели

Любая выполняющаяся программа получает в свое распоряжение определенный набор ресурсов микропроцессора. Эти ресурсы необходимы для выполнения и хранения в памяти команд программы, данных и информации о текущем состоянии программы и микропроцессора. Набор этих ресурсов представляет собой **программную модель микропроцессора** [7].

Программные модели более ранних микропроцессоров (i486, Pentium) отличаются меньшим размером адресуемого пространства оперативной памяти (2^{32}—1, так как разрядность их шины адреса составляет 32 бита) и отсутствием некоторых групп регистров.

Программную модель микропроцессора Intel составляют [14]:

– ***пространство адресуемой памяти*** (для Pentium III — до 2^{36}—1 байт);

– набор ***регистров для хранения данных общего назначения*** (eax/ax/ah/al, ebx/bx/bh/bl, edx/dx/dh/dl, ecx/cx/ch/cl, ebp/bp, esi/si, edi/di, esp/sp). Регистры этой группы используются для хранения данных и адресов;

– набор ***сегментных регистров*** (cs, ds, ss, es, fs, gs). Регистры этой группы используются для хранения адресов сегментов в памяти;

– набор ***регистров состояния и управления*** (это регистры, которые содержат информацию о состоянии микропроцессора, исполняемой программы и позволяют изменить это состояние:

- регистр флагов eflags/flags;
- регистр указатель команды eip/ip;

– ***системные регистры*** — это регистры для поддержания различных режимов работы, сервисных функций, а также регистры, специфичные для определенной модели микропроцессора;

– набор ***регистров устройства вычислений с плавающей точкой*** (сопроцессора) (st(0), st(l), st(2), st(3), st(4), st(5), st(6), st(7)). Регистры этой группы предназначены для написания программ, использующих тип данных с плавающей точкой;

– набор ***регистров целочисленного MMX-расширения*** (mmx0, mmx1, mmx2, mmx3, mmx4, mmx5, mmx6, mmx7), отображенных на регистры сопроцессора (впервые появились в архитектуре микропроцессора Pentium MMX);

– набор ***регистров MMX-расширения с плавающей точкой*** (xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7) (впервые появились в архитектуре микропроцессора Pentium Ш);

– ***программный стек***. Это специальная информационная структура, работа с которой предусмотрена на уровне машинных команд.

Программные модели ранних микропроцессоров Intel составляют лишь небольшую часть приведенной программной модели. Так, в программную модель микропроцессора i8086 входят 8- и 16-битные регистры общего назначения, сегментные регистры, регистры flags, ip и адресное пространство памяти размером до 1 Мбайт.

Для обеспечения работоспособности программ, написанных для младших 16-разрядных моделей микропроцессоров фирмы Intel, начиная с i8086 микропроцессоры i386 и Pentium имеют, в основном, 32-разрядные регистры. Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в их обозначениях — они имеют приставку e (Extended).

2.2.2 Регистры общего назначения

Регистры общего назначения используются в программах для хранения:

– операндов логических и арифметических операций;
– компонентов адреса;
– указателей на ячейки памяти.

Все эти регистры доступны для хранения операндов без особых ограничений, хотя при определенных условиях некоторые из них все же имеют жесткое функциональное назначение, закрепленное на уровне логики работы машинных команд. Среди всех этих регистров особо следует выделить регистр esp. Его не следует использовать явно для хранения каких-либо операндов программы, так как в нем хранится указатель на положение вершины стека программы. Все регистры этой группы позволяют обращаться к своим «младшим» частям (рис. 2.2).

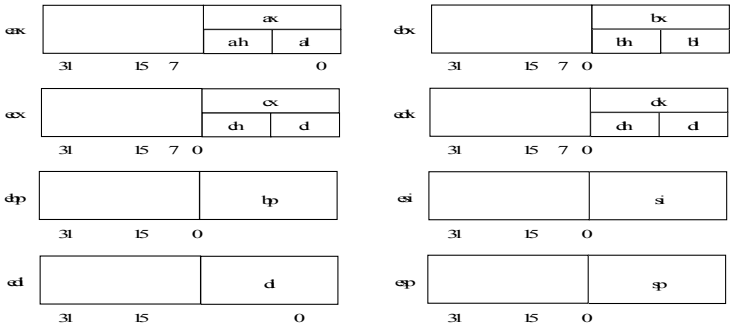

Рис. 2.2 — Регистры общего назначения

Рассматривая этот рисунок, заметьте, что использовать для самостоятельной адресации можно только младшие 16- и 8-битные части этих регистров. Старшие 16 битов этих регистров как самостоятельные объекты недоступны. Перечислим регистры, относящиеся к группе регистров общего назначения. Так как эти регистры физически находятся в микропроцессоре внутри ариф-

метико-логического устройства (АЛУ), то их еще называют регистрами АЛУ:

– eax/ax/ah/al (Accumulator register) — *аккумулятор.* Применяется для хранения промежуточных данных. В некоторых командах использование этого регистра обязательно;

– ebx/bx/bh/bl (Base register) — *базовый регистр.* Применяется для хранения базового адреса некоторого объекта в памяти;

– ecx/cx/ch/cl (Count register) — *регистр-счетчик.* Применяется в командах, производящих некоторые повторяющиеся действия. Его использование зачастую неявно и скрыто в алгоритме работы соответствующей команды. К примеру, команда организации цикла loop, кроме передачи управления команде, находящейся по некоторому адресу, анализирует и уменьшает на единицу значение регистра ecx/cx;

– edx/dx/dh/dl (Data register) — *регистр данных.* Так же как и регистр eax/ax/ah/ al, он хранит промежуточные данные. В некоторых командах его использование обязательно; для некоторых команд это происходит неявно.

Следующие два регистра используются для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:

– esi/si (Source Index register) — *индекс источника.* Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;

– edi/di (Destination Index register) — *индекс приемника (получателя).* Этот регистр в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

В архитектуре микропроцессора на программно-аппаратном уровне поддерживается такая структура данных, как стек. Для работы со стеком в системе команд микропроцессора есть специальные команды, а в программной модели микропроцессора для этого существуют специальные регистры:

– esp/sp (Stack Pointer register) — *регистр указателя стека.* Содержит указатель вершины стека в текущем сегменте стека;

– ebp/bp (Base Pointer register) — *регистр указателя базы кадра стека.* Предназначен для организации произвольного доступа к данным внутри стека.

2.2.3 Сегментные регистры

В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, fs, gs. Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно, такая организация памяти называется сегментной. Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры.

Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:

- *Сегмент кода.* Содержит команды программы. Для доступа к этому сегменту служит регистр cs (code segment register) — *сегментный регистр кода.* Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (то есть эти команды загружаются в конвейер микропроцессора).
- *Сегмент данных.* Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр ds (data segment register) — *сегментный регистр данных,* который хранит адрес сегмента данных текущей программы.
- *Сегмент стека.* Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит регистр ss (stack segment register) — *сегментный регистр стека,* содержащий адрес сегмента стека.

- *Дополнительный сегмент данных.* Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах es, fs, gs (extension data segment registers).

2.2.4 Регистры состояния и управления

В микропроцессор включены несколько регистров, которые постоянно содержат информацию о состоянии как самого микропроцессора, так и программы, команды которой в данный момент загружены на конвейер. Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние самого микропроцессора. К этим регистрам относятся:

– eflags/flags (flag register) — регистр флагов. Разрядность eflags/flags — 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются флагами. Младшая часть этого регистра полностью аналогична регистру flags для i8086;

– eip/ip (Instruction Pointer register) — указатель команд. Регистр eip/ip имеет разрядность 32/16 бит и содержит смещение следующей подлежащей выполнению команды относительно содержимого сегментного регистра cs в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и

возврата из процедур. Возникновение прерываний также приводит к модификации регистра eip/ip.

Исходя из особенностей использования флаги регистра eflags/flags можно разделить на три группы:

– 8 флагов состояния. Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния регистра eflags отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм. В таблице 1.4 приведены флаги состояния и указано их назначение;

– 1 флаг управления. Обозначается как df (Directory Flag). Он находится в десятом бите регистра eflags и используется цепочечными командами. Значение флага df определяет направление поэлементной обработки в этих операциях: от начала строки к концу (df = 0) либо, наоборот, от конца строки к ее началу (df = 1). Для работы с флагом df существуют специальные команды eld (снять флаг df) и std (установить флаг df). Применение этих команд позволяет привести флаг df в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьшение счетчиков при выполнении операций со строками;

– 5 системных флагов, управляющих вводом/выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это приведет к прерыванию работы программы.

Знание флагов регистра eflags/flags необходимо при создании программ на языке низкого уровня Ассемблер. Более подробную информацию по флагам можно взять из любого учебника по программированию на языке Ассемблер для микропроцессоров на платформе Intel x86, например в учебнике В. Юрова [14].

Внимание! Языки Ассемблер для различных микропроцессоров различны.

2.2.5 Системные регистры микропроцессора

Название этих регистров говорит о том, что они выполняют специфические функции в системе. Использование системных регистров жестко регламентировано. Именно они обеспечивают работу защищенного режима. Их также можно рассматривать как часть архитектуры микропроцессора, которая намеренно оставлена видимой для того, чтобы квалифицированный системный программист мог выполнить самые низкоуровневые операции.

Системные регистры можно разделить на три группы [14]:

– четыре регистра управления;
– четыре регистра системных адресов;
– восемь регистров отладки.

Регистры управления. В группу регистров управления входят 4 регистра: cr0, cr1, cr2, cr3.

Эти регистры предназначены для общего управления системой. Регистры управления доступны только программам с уровнем привилегий 0 — ядру операционной системы.

Хотя микропроцессор имеет четыре регистра управления, доступными являются только три из них — исключается ***cr1***, функции которого пока не определены (он зарезервирован для будущего использования).

Регистр ***cr0*** содержит системные флаги, управляющие режимами работы микропроцессора и отражающие его состояние глобально, независимо от конкретных выполняющихся задач.

Регистр ***cr2*** используется при страничной организации оперативной памяти для регистрации ситуации, когда текущая команда обратилась по адресу, содержащемуся в странице памяти, отсутствующей в данный момент времени в памяти. В такой ситуации в микропроцессоре возникает исключительная ситуация с номером 14 и линейный 32-битный адрес команды, вызвавшей это исключение, записывается в регистр cr2. Имея эту информацию, обработчик исключения 14 определяет нужную страницу,

осуществляет ее подкачку в память и возобновляет нормальную работу программы.

Регистр ***cr3*** также используется при страничной организации памяти. Это так называемый регистр каталога страниц первого уровня. Он содержит 20-битный физический базовый адрес каталога страниц текущей задачи. Этот каталог содержит 1024 32-битных дескриптора, каждый из которых содержит адрес таблицы страниц второго уровня. В свою очередь, каждая из таблиц страниц второго уровня содержит 1024 32-битных дескриптора, адресующих страничные кадры в памяти. Размер страничного кадра — 4 Кбайт.

Регистры системных адресов. Эти регистры еще называют регистрами управления памятью. Они предназначены для защиты программ и данных в мультизадачном режиме работы микропроцессора.

При работе в защищенном режиме микропроцессора адресное пространство делится:

– на глобальное — общее для всех задач;

– локальное — отдельное для каждой задачи.

Этим разделением и объясняется присутствие в архитектуре микропроцессора следующих системных регистров:

– регистра таблицы глобальных дескрипторов ***gdtr (Global Descriptor Table Register)***, имеющего размер 48 бит и содержащего 32-битовый (биты 16—47) базовый адрес глобальной дескрипторной таблицы GDT и 16-битовое (биты 0—15) значение предела, представляющее собой размер в байтах таблицы GDT;

– регистра таблицы локальных дескрипторов ***ldtr (Local Descriptor Table Register)***, имеющего размер 16 бит и содержащего так называемый селектор дескриптора локальной дескрипторной таблицы LDT. Этот селектор является указателем в таблице GDT, который и описывает сегмент, содержащий локальную дескрипторную таблицу LDT;

– регистра таблицы дескрипторов прерываний ***idtr (Interrupt Descriptor Table Register)***, имеющего размер 48 бит и содержащего 32-битовый (биты 16—47) базовый адрес дескрипторной таблицы прерываний IDT и 16-битовое (биты 0—15) зна-

чение предела, представляющее собой размер в байтах таблицы IDT;

– 16-битового регистра задачи ***tr (Task Register)***, который подобно регистру ldtr, содержит селектор, то есть указатель на дескриптор в таблице GDT. Этот дескриптор описывает текущий сегмент состояния задачи (TSS — Task Segment Status). Этот сегмент создается для каждой задачи в системе, имеет жестко регламентированную структуру и содержит контекст (текущее состояние) задачи. Основное назначение сегментов TSS — сохранять текущее состояние задачи в момент переключения на другую задачу.

Регистры отладки. Это группа регистров, предназначенных для аппаратной отладки. Средства аппаратной отладки впервые появились в микропроцессоре i486. Аппаратно микропроцессор содержит восемь регистров отладки, но реально из них используются только 6.

Регистры ***dr0, dr1, dr2, dr3*** имеют разрядность 32 бита и предназначены для задания линейных адресов четырех точек прерывания. Используемый при этом механизм следующий: любой формируемый текущей программой адрес сравнивается с адресами в регистрах dr0...dr3, и при совпадении генерируется исключение отладки с номером 1.

Регистр ***dr6*** называется регистром состояния отладки. Биты этого регистра устанавливаются в соответствии с причинами, которые вызвали возникновение последнего исключения с номером 1.

Регистр ***dr7*** называется регистром управления отладкой. В нем для каждого из четырех регистров контрольных точек отладки имеются поля, с помощью которых можно уточнить следующие условия, при которых следует сгенерировать прерывание:

– место регистрации контрольной точки — только в текущей задаче или в любой задаче;

– тип доступа, по которому инициируется прерывание: только при выборке команды, при записи или при записи/чтении данных.

2.3 Режимы функционирования процессора Intel x86

2.3.1 Перечень режимов функционирования процессора Intel x86

Режимы работы процессора. Процессоры семейства Intel x86, начиная с 80286, имеют несколько режимов работы:

- Реальный режим. Предназначен для совместимости с младшими моделями процессоров.
- Защищенный режим. Основной режим работы процессоров. Именно в нем доступны все особенности новых моделей процессоров, такие, как многозадачность, защита программ пользователей, возможность работы с большим объемом памяти, виртуальная память и т.д.
- Режим системного управления (SMM). В этом режиме доступны дополнительные возможности процессора.
- Режим Virtual-86. Этот режим схож с реальным режимом, однако может быть включен только в защищенном режиме. В этом режиме возможно выполнение нескольких приложений реального режима.

2.3.2 Реальный режим работы процессоров Intel x86

После рестарта процессор находится в реальном режиме. Именно в реальном режиме функционирует базовая система ввода/вывода, загрузчики операционных систем, а также и некоторые операционные системы, например MS DOS.

Адресация ячеек памяти в реальном режиме. В реальном режиме адрес имеет размер 20 бит и, следовательно, максимальный объем адресуемой памяти составляет 1 Мб. Для формирования 20-битового адреса в памяти используются два 16-битовых регистра: сегментный регистр и регистр смещения (рис. 2.3).

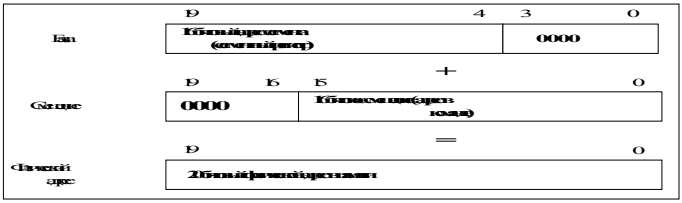

Рис. 2.3 — Преобразование логического адреса
в физический адрес в реальном режиме

Сегментный регистр определяет базовый адрес сегмента. В нем содержатся старшие 16 бит базового адреса сегмента, младшие четыре бита предполагаются равными 0. В регистре смещения находится смещение внутри сегмента (относительно базового адреса сегмента). Смещение может также задаваться непосредственной константой. Смещение добавляется к базовому адресу сегмента, и, таким образом, получается 20-битовый адрес в памяти. Более просто можно описать вычисление следующим образом — 16-битовое значение сегментного регистра сдвигается на 4 бита влево и к нему добавляется значение смещения, в результате получается 20-битовое значение. Именно такая схема применялась в 8086 и 8088 процессорах.

Такое представление 20-битного адреса двумя 16-битными числами в процессорах 8086/88 поддерживается и в реальном режиме всех последующих процессоров x86.

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.

Распределение оперативной памяти в MS DOS. Согласно сегментной модели исполнительный (линейный) адрес вычисляется по формуле, приведенной на рис. 2.3. Таким образом, обеспечивался доступ к адресному пространству Addr = 00000—FFFFFh при помощи пары 16-битных регистров.

Заметим, что при Seg = 0FFFFh и Offset = 0FFFFh 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 бит и даже 36 бит. В реальном режиме процессора, используемом в DOS, применяется та же сегментная модель памяти, и формально был доступен лишь 1 Мбайт памяти. 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 (64 Кб — 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.

Распределение памяти PC, непосредственно адресуемой процессором, представляется следующим образом [15] (рис. 2.4):

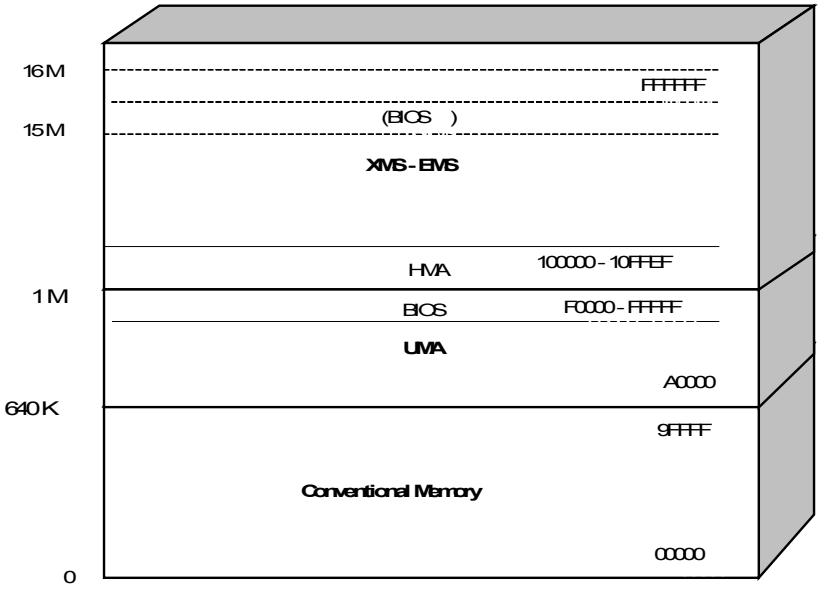

Рис. 2.4 — Распределение памяти в MS DOS

– 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 непрерывной (нефрагментированной) памяти, о которой мечтают программисты.

– Память выше 100000h — дополнительная (расширенная) память Extended Memory, непосредственно доступная только в защищенном режиме для компьютеров с процессорами i286 и

выше. В ней выделяется область 100000h—10FFEFh — высокая память HMA.

Стандартная память — Conventional memory. При работе в среде операционных систем типа MS DOS стандартная память является самой дефицитной в PC. На ее небольшой объем (типовое значение 640 Кбайт) претендуют и BIOS, и ОС реального режима, а оставшаяся часть предназначена для прикладного ПО. Стандартная память распределяется следующим образом [15]:

– 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.

В процессорах семейства Intel предусмотрено 256 прерываний и соответственно 256 векторов прерываний. Все вектора прерываний объединяются в таблицу, состоящую из 256 4-байтовых элементов и занимающую 1 Кб. В реальном режиме эта таблица находится в самом начале памяти по адресу 0000:0000 или просто по 0-му физическому адресу.

Каждый элемент таблицы состоит из двух полей. Первые два байта представляют собой значение, заносимое в регистр IP, последние два байта содержат значение, заносимое в регистр CS.

Управление пользовательской памятью осуществляется с использованием специализированных структур: таблицы таблиц — list of list (табл. 2.2) и управляющих блоков памяти — memory control block (табл. 2.3). Данные структуры являются недокументированными.

Таблица 2.2 — **Структура таблицы таблиц**

Смещение	Длина	Содержимое

–2	2	Сегментный адрес первого блока управления памятью MCB (memory control block)
0	4	Указатель на первый DPB (disk parameter blockout)
+4	4	Указатель на список таблиц открытых файлов
+8	4	Указатель на первый драйвер DOS (CLOCK$)
…	…	…

Таблица 2.3 — **Структура блока управления памятью**

Смещение	Длина	Содержимое
+0	1	Тип блока: '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 (1 параграф = 16 байт = 10h байт)
+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 адресу (Сегментный адрес блока + Размер + 1):0000;

– блоки Z-типа: (Сегментный адрес блока + Размер + 1): 0000 = конец памяти (a000H = 640K).

– +1 при вычислении адреса следующего блока — это шапка блока, равная 16 байтам или 10h (см. табл. 2.3).

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.

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

Верхняя память — UMA (Upper Memory Area)

Верхняя память имеет области различного назначения, которые могут быть заполнены буферной памятью адаптеров, постоянной памятью или оставаться незаполненными. Первоначально 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 [15]:

– 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 может занимать разные адресные пространства.

Теневая память — Shadow ROM и Shadow RAM

В области верхней памяти UMA обычно располагаются устройства с медленной памятью [15]:

– 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 i286, где она была реализована аппаратно. Процессоры класса i386+ позволяют ее реализовать программно, с помощью страничной переадресации. Затенение ОЗУ и ПЗУ устройств выполняется по-разному.

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 памяти отно-

сится 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 периферийной платы ISA (Industry Standard Architecture), и то не во всех режимах.

2.3.3 Защищенный режим работы процессоров Intel x86

32-битовые процессоры Intel обеспечивают значительную поддержку для операционных систем и программного обеспечения для системной разработки. Эта поддержка — часть системной архитектуры и включает возможности для помощи в организации следующих операций:

- Управление памятью.
- Защита программных модулей.
- Многозадачность.
- Обработка прерываний и исключений.
- Управление кэшированием.
- Управление аппаратными ресурсами и питанием.
- Отладка и отслеживание производительности.

Управление памятью в архитектуре Intel делится на две части: сегментацию и трансляцию страниц. ***Сегментация*** предоставляет механизм для изолирования индивидуального кода, данных и стека. Таким образом, несколько программ могут выполняться одновременно на одном процессоре, не перекрывая друг друга. ***Трансляция страниц*** предоставляет механизм для реализации виртуальной памяти с подкачкой страниц по запросу, где части программы отображаются, как необходимо, на физическую память. Трансляция страниц так же может использоваться для изоляции между несколькими задачами. В защищенном режиме обязательно должна использоваться какая-нибудь форма сегментации.

Как показано на рисунке 2.5, сегментация обеспечивает механизм для разделения процессорного адресного пространства (называемого линейным пространством) на меньшие защищен-

ные адресные пространства. Сегменты могут использоваться для содержания кода, данных, стека и системных структур данных.

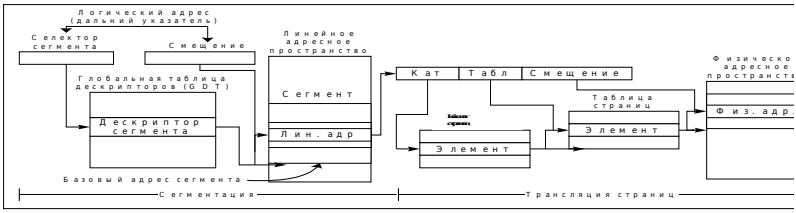

Рис. 2.5 — Сегментация и трансляция страниц

Все сегменты содержатся внутри линейного адресного пространства процессора. Для доступа к любому байту информации в каком-либо сегменте необходим логический адрес (иногда его называют дальним указателем). Логический адрес состоит из селектора сегмента и смещения. Селектор сегмента — это уникальный идентификатор сегмента. Селектор сегмента также является смещением в таблице дескрипторов. Каждый сегмент имеет дескриптор сегмента, который определяет размер, права доступа, уровень привилегий, тип сегмента и расположение его первого байта внутри линейного пространства. Смещение добавляется к базовому адресу сегмента для доступа к конкретному байту внутри сегмента. Базовый адрес плюс смещение формируют линейный адрес внутри линейного адресного пространства процессора [7].

Если трансляция страниц не используется, то линейное адресное пространство непосредственно отображается на физическое адресное пространство процессора. Физическое адресное

пространство определяется, как диапазон адресов, которые процессор может генерировать на шине адреса.

В многозадачных операционных системах линейное пространство гораздо больше, чем пространство физической памяти, поэтому необходим метод виртуализации линейного адресного пространства. Эта виртуализация линейного адресного пространства производится через механизм трансляции страниц.

Трансляция страниц поддерживает среду с ***виртуальной памятью***, где большое линейное пространство эмулируется за счет маленького количества физической памяти и некоторого количества дисковой памяти. При этом все сегменты делятся на 4-килобайтовые страницы, которые могут находиться либо в памяти, либо на диске. Когда задача пытается обратиться к странице, которой нет в памяти, происходит исключение и операционная система загружает недостающую страницу в память, после чего выполнение задачи возобновляется.

Для более экономного использования памяти используется двухуровневая таблица схемы трансляции страниц. На верхнем уровне находится каталог страниц, который содержит ссылки на таблицы страниц. Таблицы страниц содержат ссылки на сами страницы.

Трансляция страниц — это единственный способ, при котором возможно выполнение нескольких задач реального режима.

Упрощенно **схему по формированию физического адреса памяти на 32-разрядных процессорах в защищенном режиме** можно представить на рисунке 2.6.

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.

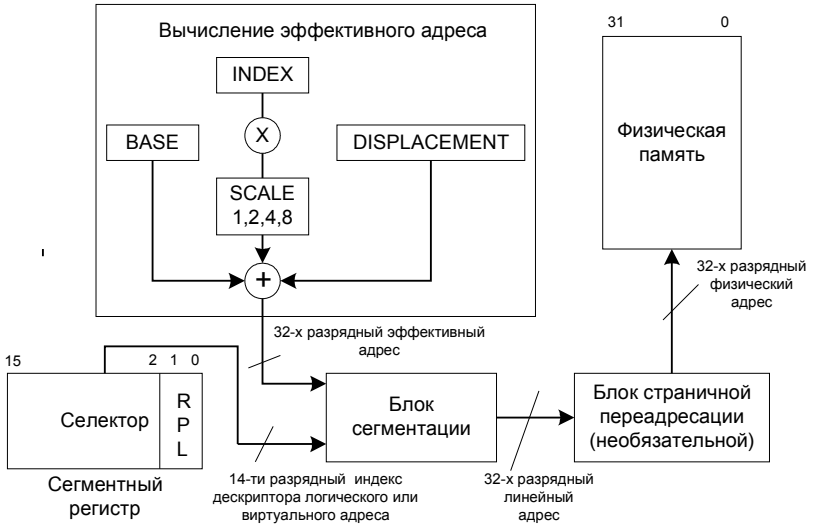

Рис. 2.6 — Формирование адреса памяти 32-разрядных процессоров в защищенном режиме

Блок сегментации транслирует логическое адресное пространство в 32-битное пространство линейных адресов.

Для трансляции логического адреса в линейный процессор выполняет следующие действия:

- Использует селектор сегмента как смещение в таблицах GDT или LDT для определения дескриптора сегмента и считывает последний в процессор.
- Делает проверки прав доступа к сегменту и попадание в диапазон адресов сегмента по полю предел.
- Добавляет базовый адрес сегмента к смещению для формирования линейного адреса.

Селектор сегмента — это 16-битовый идентификатор сегмента. Он не указывает напрямую на сегмент, а вместо этого он определяет сегмент. Селектор сегмента содержит следующие поля [7]:

- Index (биты с 3 по 15). Выбирает один из 8192 дескрипторов из GDT или LDT. Процессор умножает значение индекс

на 8 и добавляет результат к базовому адресу GDT или LDT (из регистров GDTR или LDTR соответственно).

- TI (бит 2). Определяет таблицу дескрипторов для использования. Нулевое состояние определяет GDT, единичное состояние определяет LDT.
- RPL (биты 0 и 1). Указывает уровень привилегий селектора. Диапазон привилегий — от 0 до 3, где 0-й — самый привилегированный.

Дескриптор сегмента — это структура данных в GDT или LDT, которая предоставляет процессору размер, положение и информацию доступа и статуса.

Процессор распознает следующие типы системных дескрипторов [7]:

- Дескриптор сегмента локальной таблицы дескрипторов (LDT).
- Дескриптор сегмента состояния задачи (task state descriptor).
- Дескриптор шлюза вызова (call-gate descriptor).
- Дескриптор шлюза прерывания (interrupt-gate descriptor).
- Дескриптор шлюза ловушки (trap-gate descriptor).
- Дескриптор шлюза задачи (task-gate descriptor).

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 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 адресо-

вать 4/64 Гбайт физической памяти, но в реальном режиме доступен только 1 Мбайт, начинающийся с младших адресов.

Механизм сегментации, поддерживаемый в архитектуре Intel, может быть использован для реализации множества сегментных моделей:

- ***Базовая Flat модель***. Самая простая модель — базовая Flat модель, в которой операционная система и приложения имеют доступ к непрерывному не сегментированному адресному пространству. Эта базовая модель прячет механизм сегментации от системного разработчика и прикладного программиста.

Для реализации этой модели необходимо, по крайней мере, два дескриптора сегмента. Один для ссылки на сегмент кода, другой — для сегмента данных. Тем не менее два этих сегмента отображаются на все адресное пространство.

- ***Защищенная Flat модель***. Защищенная Flat модель похожа на базовую модель, за исключением того, что пределы сегментов выставляются так, чтобы включить только пространство адресов, физически существующее для памяти. Эта модель предоставляет минимальный уровень аппаратной защиты против программных ошибок и может использоваться совместно с трансляцией страниц для разделения различных задач.
- ***Многосегментная модель***. Многосегментная модель использует все возможности механизма сегментации для обеспечения аппаратной защиты кода, структур данных, программ и задач. В данном случае каждая программа имеет свою таблицу дескрипторов сегментов и свои сегменты. Сегменты могут принадлежать одной программе, а могут и разделяться между несколькими программами. Доступ ко всем сегментам контролируется на аппаратном уровне. Проверка доступа может использоваться не только для защиты от ссылок, выходящих за границы сегмента, а также для защиты от выполнения операций в некоторых сегментах. Информация прав доступа может использоваться для назначения колец защиты.

Механизм переключения в защищенный режим. После аппаратного сброса процессор находится в реальном режиме. В этом месте процесса инициализации несколько базовых структур данных и модулей кода должны быть загружены в физиче-

скую память для обеспечения дальнейшей инициализации процессора. Перед переключением процессора в защищенный режим программа инициализации должна загрузить минимальное число структур защищенного режима для обеспечения надежной работы процессора в защищенном режиме. Эти структуры данных включают следующие [7]:

– IDT защищенного режима.
– GDT.
– TSS.
– (опционально) LDT.
– Если используется трансляция страниц, то каталог страниц и хотя бы одну таблицу страниц.
– Один или более модулей кода, содержащих необходимые обработчики прерываний и исключений.

Программа инициализации должна также загрузить следующие регистры:

– GDTR.
– (опционально) IDTR. Может быть загружен после переключения в защищенный режим.
– CR1 — CR4.
– Регистры типа диапазонов адресов (только для Pentium Pro).

После инициализации вышеперечисленных структур и регистров процессор может переключиться в защищенный режим посредством установки флага PE в регистре CR0.

Механизмы защиты. В защищенном режиме архитектура Intel предоставляет механизм защиты на обоих уровнях — сегментном и трансляции страниц. Этот механизм обеспечивает возможность ограничения доступа к определенным сегментам или страницам в зависимости от уровней привилегий.

При использовании механизма защиты к каждой ссылке в память применяются различные проверки защиты. Все проверки выполняются до начала цикла обращения к памяти, любое нарушение приводит к возникновению исключения.

Сегментный механизм защиты распознает 4 уровня привилегий от 0 до 3. Больший номер означает меньший уровень при-

вилегий. Процессор использует уровни привилегий для предотвращения доступа менее привилегированных программ к более привилегированным программам или их данным.

Для выполнения проверок между сегментами кода и сегментами данных процессор распознает следующие три типа уровней привилегий [7]:

- Текущий уровень привилегий (CPL). CPL — это уровень привилегий текущей выполняемой программы или задачи. Обычно CPL равен уровню привилегий кодового сегмента, в котором происходит выполнение. Процессор изменяет CPL, когда управление передается в кодовый сегмент с другим уровнем привилегий.
- Уровень привилегий дескриптора (DPL). DPL — это уровень привилегий сегмента или шлюза. Когда программа обращается к сегменту или шлюзу, DPL сегмента или шлюза сравнивается с CPL и RPL сегмента или шлюза. DPL интерпретируется по-разному в зависимости от типа сегмента или шлюза.
- Запрашиваемый уровень привилегий. RPL — это переопределяемый уровень привилегий, который назначается селектору сегмента. Процессор проверяет RPL вместе с CPL для определения доступности сегмента.

Уровни привилегий проверяются во время загрузки селекторов сегментов в сегментные регистры.

Обработка прерываний и исключений. Таблица дескрипторов прерываний (IDT) ассоциирует каждый вектор исключения или прерывания с дескриптором шлюза для процедуры или задачи, предназначенной для обслуживания ассоциированных векторов прерываний. Как и GDT, и LDT, IDT — это массив 8-байтовых дескрипторов (в защищенном режиме). В отличие от GDT первый элемент IDT может содержать дескриптор. Так как имеется только 256 векторов прерываний и исключений, нет необходимости более чем в 256 дескрипторах. IDT может содержать менее 256 дескрипторов, так как дескрипторы требуются только для векторов прерываний и исключений, которые могут произойти. Все пустые дескрипторные слоты в IDT должны иметь флаг присутствия P, равный 0.

IDT может содержать дескрипторы шлюзов трех видов [7]:

- Дескриптор шлюза задачи.
- Дескриптор шлюза прерывания.
- Дескриптор шлюза ловушки.

Формат шлюза задачи, используемый в IDT, точно такой же, как и в GDT и LDT. Шлюз задачи содержит селектор сегмента TSS для обработчика исключения и/или прерывания.

Шлюзы прерываний и ловушек очень похожи на шлюзы вызова. Они содержат дальний указатель, который процессор использует для передачи управления в процедуру обработчика. Эти шлюзы отличаются только тем, как процессор оперирует с флагом IF (флаг прерываний) в регистре EFLAGS. При использовании шлюза прерывания процессор сбрасывает IF флаг, а последующая инструкция IRET его восстанавливает. В случае шлюза ловушки процессор не изменяет содержимое этого флага.

Управление задачами. Архитектура Intel обеспечивает механизм для сохранения состояния задачи, для выборки задачи для исполнения и для переключения из одной задачи в другую. В защищенном режиме процессор все время работает в контексте какой-нибудь задачи. Даже простейшие системы должны определить, по крайней мере, одну задачу. Более сложные системы могут использовать возможности процессора по управлению задачами для поддержки многозадачных приложений.

Задача состоит из двух частей: ***пространства выполнения задачи и сегмента состояния задачи (TSS)***. Пространство выполнения задачи состоит из сегмента кода, сегмента стека, одного или более сегментов данных.

TSS определяет сегмент, который определяет пространство выполнения задачи, и предоставляет место для сохранения информации состояния. В многозадачных системах TSS также предоставляет механизм для связывания задач.

Следующие элементы определяют состояние текущей выполняемой задачи [7]:

- Пространство выполнения текущей задачи, определяемое селекторами сегментов в сегментных регистрах (CS, DS, SS, ES, FS и GS).

- Состояние регистров общего назначения.
- Состояние регистра EFLAGS.
- Состояние регистра EIP.
- Состояние управляющего регистра CR3.
- Состояние регистра задачи.
- Состояние регистра LDTR.
- Базовый адрес карты В/В и карта В/В.
- Указатели стека для стеков привилегий 0, 1 и 2.
- Ссылка на предыдущую задачу.

Процессор определяет пять структур данных:

- Сегмент состояния задачи (TSS).
- Дескриптор шлюза задачи.
- Дескриптор TSS.
- Регистр задачи.
- Флаг NT (флаг вложенности задачи) в регистре EFLAGS.

В защищенном режиме TSS и дескриптор TSS должны быть созданы, по крайней мере, для одной задачи, и селектор сегмента TSS должен быть загружен в регистр задачи.

Процессор передает управление на другую задачу в любом из следующих четырех случаев [7]:

- Текущая программа, задача или процедура выполняет JMP (команда безусловного перехода) или CALL (команда вызова подпрограммы) инструкции на дескриптор TSS в GDT.
- Текущая программа, задача или процедура выполняет JMP или CALL инструкции на дескриптор шлюза задачи в GDT или LDT.
- Вектор прерывания или исключения указывает на дескриптор шлюза задачи в GDT.
- Текущая задача запускает инструкцию IRET (команда выхода из обработчика прерываний), когда NT флаг в регистре EFLAGS установлен.

2.3.4 Режим системного управления (SMM)

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[4]. 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 [7]:

– 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 устройства с плавающей точкой 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#.

2.3.5 Режим Virtual-86

[4] 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.

Процессоры архитектуры Intel (начиная с i386) предоставляют два способа для запуска программ, созданных для i8086 процессора:

- Реальный режим.
- Virtual-8086 режим.

Virtual-8086 режим является специальным типом задачи, которая запускается в защищенном режиме. Когда операционная система переключается в Virtual-8086 задачу, процессор эмулирует i8086 процессор. Среда выполнения процессора во время состояния эмуляции такая же, как в реальном режиме. Основное различие между двумя режимами — это то, что в Virtual-8086 режиме эмулятор использует некоторые сервисы защищенного режима.

Включение Virtual-8086 режима. Процессор работает в Virtual-8086 режиме, когда VM флаг в регистре EFLAGS установлен. Этот флаг может быть установлен, когда процессор переключается в новую задачу или возвращается в Virtual-8086 режим после IRET инструкции.

Программное обеспечение не может изменить состояние VM флага непосредственно в регистре EFLAGS. Вместо этого оно изменяет этот флаг в имидже регистра EFLAGS, хранимом в TSS или в стеке, перед выполнением инструкции IRET.

Структура Virtual-8086 задачи. Virtual-8086 задача состоит из следующих элементов [7]:

- 32-битовый TSS для задачи.
- 8086 программа.
- Virtual-8086 монитор.
- Сервисы 8086 операционной системы.

TSS новой задачи должен быть 32-битовым, так как 16-битовый TSS не содержит старших битов регистра EFLAGS. Процессор входит в virtual-8086 режим для запуска 8086 программы и возвращается в защищенный режим для запуска virtual-8086 монитора.

Virtual-8086 монитор — это 32-битовый код защищенного режима, который выполняется на CPL=0. Монитор состоит из инициализации, обработчиков прерываний и исключений, процедур эмуляции ввода/вывода.

Транслирование страниц в Virtual-8086 задачах. Так как программа, запущенная в virtual-8086 режиме, может использовать только 20-битовые линейные адреса, то необходимо, чтобы процессор преобразовывал эти адреса в 32-битовые перед отображением их в физические.

Трансляция страниц может не использоваться для одной virtual-8086 задачи, однако трансляция страниц полезна или необходима в следующих ситуациях [7]:

- Когда запускаются несколько Virtual-8086 задач. Трансляция страниц позволяет отобразить нижние 1 Мб на разные участки физической памяти.
- При эмуляции отображения 64 Кб за 1 Мб на первые 64Кб.
- При разделении сервисов i8086 операционной системы и кода ПЗУ, общих для всех программ i8086.
- При перенаправлении или отлавливании ссылок на устройства ввода/вывода отображаемых в память.

Выход из Virtual-8086 режима. Процессор может выйти из virtual-8086 режима только через прерывание или исключение. Далее приводятся ситуации, в которых прерывание или исключение приведет к выходу из virtual-8086 режима [7]:

- Процессор обслуживает аппаратные прерывания, приостанавливая выполнение 8086 приложения. При приеме аппаратного прерывания процессор переключается в защищенный режим и либо переключается на задачу, либо передает управление через шлюз прерывания или ловушки.
- Процессор обслуживает исключения, вызванные кодом Virtual-8086 задачи, или обслуживает прерывания, принадлежащие этой задаче.
- Процессор обслуживает программные прерывания, генерируемые кодом virtual-8086 задачи (такие, как прерывания MS DOS).
- Аппаратный сброс, инициированный через ножки процессора RESET или INIT, является специальным типом прерывания. При сигналах RESER или INIT процессор переключается в реальный режим.

- Запуск команды HLT в virtual-8086 режиме вызовет прерывание основной защиты.

Обработка прерываний и исключений в virtual-8086 режиме. При генерации прерывания или исключения процессор переключается в защищенный режим и выполняет переход на обработчик защищенного режима. При завершении обработки прерывания процессор возвращает управление назад в 8086-программу. Обработчик прерывания может сам выполнить весь сервис, а может отразить его в назад в virtual-8086 задачу для выполнения.

Начиная с процессоров Pentium, предоставляются дополнительные возможности, такие, как вызов обработчиков напрямую, как в реальном режиме, поддержка виртуальных прерываний и т.д.

2.4 Управление памятью в ОС Windows

2.4.1 Использование отладчиков

В состав различных операционных систем и сред разработки программного обеспечения входят программы-отладчики. Эти программы позволяют анализировать и оценивать состояние аппаратных и программных средств компьютера в различных стадиях вычислительного процесса. Динамика измерения состояний ресурсов может отслеживаться в пошаговом режиме.

В операционной системе MS DOS таким отладчиком являлась программа Debug, в последних версиях ОС Windows такой программой является Ntsd.exe. Программа Debug используется для однопрограммных однопользовательских режимов работы и 16-разрядных приложений. Программа Ntsd.exe предназначена для работы в многопрограммном многопользовательском режиме работы вычислительной системы с 32-разрядными приложениями.

Вызов Debug можно выполнить в командной строке (рис. 2.7). После чего компьютер переходит в режим эмуляции MS DOS (Virtual-8086).

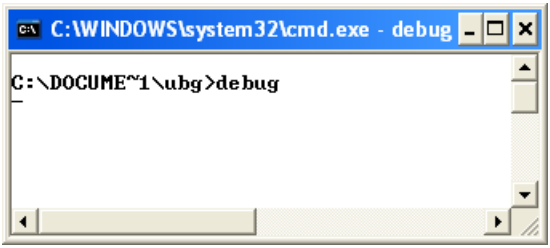

Рис. 2.7 — Вызов Debug

Программа вызывается и размещается в памяти компьютера, начиная с ячейки 100h. Знак «–» является приглашением к вводу команд. Список всех команд можно получить, если в строке приглашения набрать вопросительный знак (рис. 2.8).

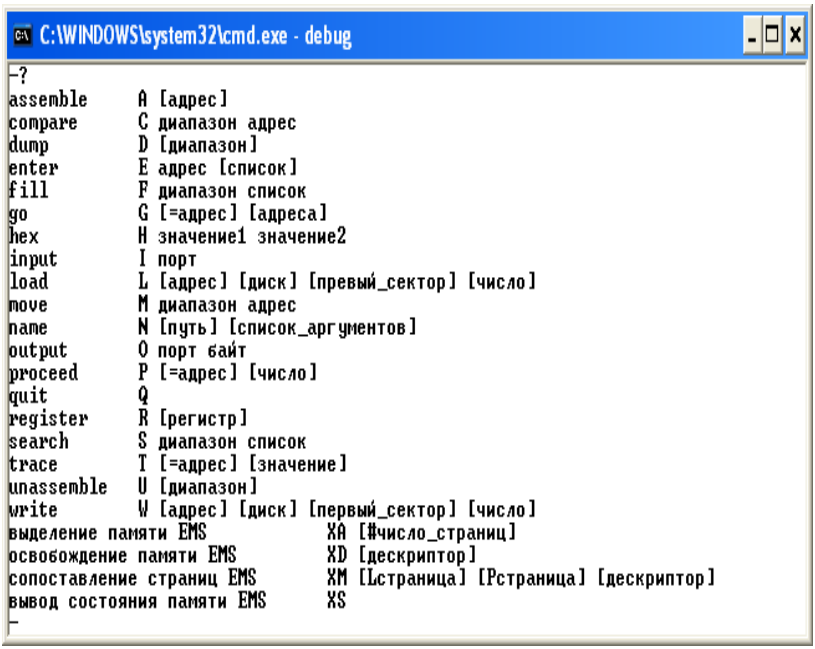

Рис. 2.8 — Вызов списка команд в Debug

Например, команда R предоставляет информацию о текущем состоянии всех программно доступных регистров компьютера (рис. 2.9).

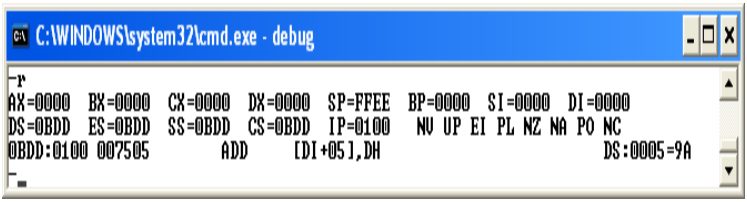

Рис. 2.9 — Вызов информации по регистрам в Debug

На рисунке 2.9 во второй строке после регистра IP представлены мнемонические значения разрядов регистра FLAGS, разряды которого используются для ветвления вычислительного процесса. На третьей строке содержится указание об очередной подготовленной для выполнения команде: адрес команды в памяти, шестнадцатеричный код представления команды и мнемонический код команды.

С помощью команды Debug можно также просмотреть дамп (диапазон) памяти. Например, набираем команду: D 40:13 (рис. 2.10), что соответствует адресу 0040:0013. Первые значащие нули адреса при наборе можно отбросить.

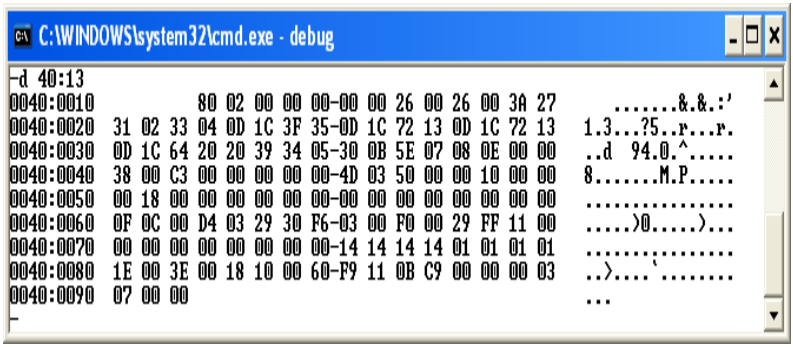

Рис. 2.10 — Просмотр дампа памяти в Debug

В каждой строке текста содержатся адреса 16 байтов, их шестнадцатеричное и символьное представление в коде ASCII. По набранному адресу в двух байтах в MS DOS содержится информация об объеме основной памяти. Необходимо помнить об LH-порядке следования байт. То есть полученные значения 80 и 02 меняем местами. Получается объем основной памяти, равный 0280h Кбайт, что в десятичном представлении соответствует 640 Кбайт.

В рассматриваемой программе имеется также возможность для ввода ассемблерных программ в память компьютера и перевода их в машинный код. Введем команду: A 100

По ней будет установлен начальный адрес в виде xxxx:0100 (xxxx означает произвольное значение регистра CS). После этого можно вводить покомандно всю программу, например:

XXXX:0100 MOV AL,20
XXXX:0102 MOV BL,16
XXXX:0104 SUB AL,BL
XXXX:0106 RET

По окончанию ввода нужно еще раз нажать клавишу Enter и операция ввода будет прекращена (рис. 2.11).

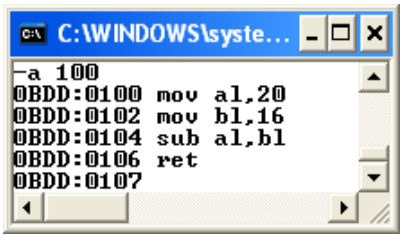

Рис. 2.11 — Ввод ассемблерной программы в Debug

Введем команду дисассемблирования: U 0100,0106, где 100 и 106 обозначают начальный и конечный адреса фрагмента программы, подлежащего переводу в машинный код. На рисунке 2.12 представлен результат выполнения введенной команды.

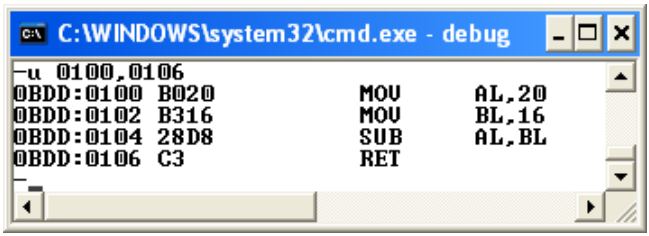

Рис. 2.12 — Дисассемблирование программы в Debug

Подводя итог, сделаем вывод, что с помощью программы Debug можно рассматривать не только дамп памяти, но также соответствие ассемблерных и машинных кодов команд программы.

2.4.2 Получение общей информации об использовании памяти

Диспетчер задач Windows позволяет просматривать общее использование памяти на вкладке Быстродействие (рис. 1.12). Здесь к информации о памяти относятся три раздела:

1. Выделение памяти.
2. Физическая память.
3. Память ядра.

В первом разделе содержится три статистических параметра виртуальной памяти. Параметр «Всего» — это общий объем виртуальной памяти, используемой как приложениями, так и операционной системой. Параметр «Предел» — это объем доступной виртуальной памяти. И, наконец, параметр «Пик» — наибольший объем памяти, использованный в течение сессии с момента последней загрузки.

Во втором разделе содержатся параметры, несущие информацию о текущем состоянии физической памяти машины. Эта статистика не имеет никакого отношения к файлу подкачки, следовательно, может являться хорошим индикатором ситуаций, когда его увеличение не даст эффекта. Параметр «Всего» — это объем памяти, обнаруженный операционной системой на компьютере. Параметр «Доступно» отражает память, доступную для использования процессами. Эта величина не включает в

себя память, доступную приложениям за счет файла подкачки. Каждое приложение требует определенного объема физической памяти и не может использовать только ресурсы файла подкачки. Параметр «Системный кэш» сообщает объем, доступный кэш-памяти системы. Это объем физической памяти, оставленный операционной системой после удовлетворения своих потребностей.

В третьем разделе отображается информация о потребностях компонентов операционной системы, обладающих наивысшим приоритетом. Эти компоненты обычно работают с сервисом низкого уровня, типа прямого доступа к жесткому диску. Параметры «Память ядра» отображают потребности ключевых служб операционной системы. Параметр «Всего» — это объем виртуальной памяти, необходимой операционной системе. Параметр «Выгружаемая» несет информацию об общем объеме памяти, использованной системой за счет файла подкачки. Параметр «Невыгружаемая» — объем физической памяти, потребляемой операционной системой.

Все приведенные параметры относятся лишь к привилегированным службам, а не ко всему сервису системы в целом. Многие компоненты ОС работают как приложения. В большинстве случаев параметры вкладки «Память ядра» должны оставаться без изменений, если не меняется что-либо в ядре операционной системы (например, устанавливается новое устройство в компьютер).

С помощью «Диспетчера задач» можно также узнать объемы памяти, используемые процессами. Для этого нужно перейти на вкладку Процессы, которая показывает список исполняемых процессов и занимаемую ими память, в том числе физическую память, пиковое, максимальное, использование памяти и виртуальную память.

Однако конкретное размещение процесса в виртуальной памяти с помощью «Диспетчера задач» узнать невозможно, нельзя также увидеть свободные, занятые страницы и блоки памяти, их размер и атрибуты защиты.

Информация, которую способен вывести «Диспетчер задач», не является полной. В ряде случаев ее достаточно для

оптимизации системы, но есть несколько ограничений, характерных для «Диспетчера задач» [13]:

1. Список процессов не полон. В окне «Диспетчера задач» представлены только процессы, зарегистрированные в Windows. В частности, в этот список не включаются драйверы устройств и некоторые системные службы.

2. Требования к памяти отражают текущее состояние процесса. В списке отражены объемы памяти, занимаемые приложениями в текущий момент времени, а не их максимальные значения.

3. Отсутствуют статистические данные. Поскольку в «Диспетчере задач» не выводятся временные характеристики, а только мгновенная картина потребления памяти, нет возможности отследить ее изменение.

Более обширную информацию по сравнению с «Диспетчером задач» можно получить утилитой TaskList (рис. 2.13).

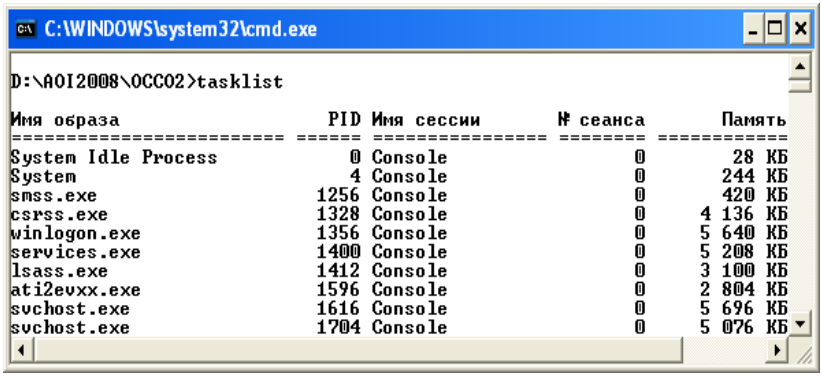

Рис. 2.13 — Запуск утилиты TaskList в окне командной строки

Получить информацию по возможностям утилиты TaskList можно так же, как и по любой команде, используемой в CMD (рис. 2.14).

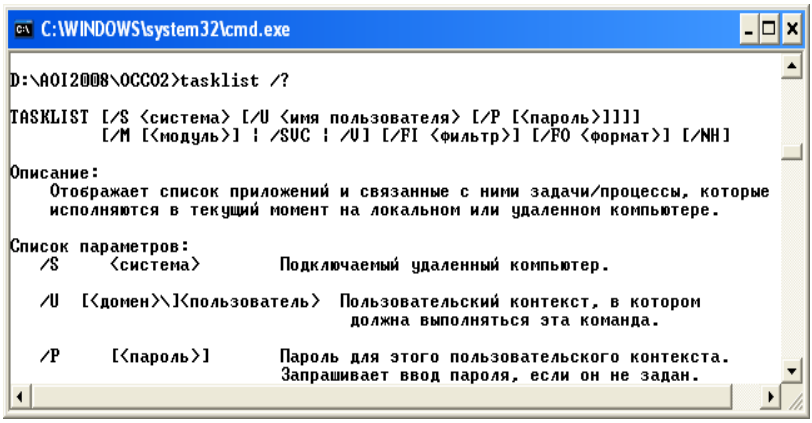

Рис. 2.14 — Вызов помощи по утилите TaskList

В операционной системе Windows XP через меню Пуск \ Все программы \ Стандартные \ Служебные можно вызвать приложение «Сведения о системе», в котором можно получить сведения об основных характеристиках организации памяти в компьютере (рис. 2.15).

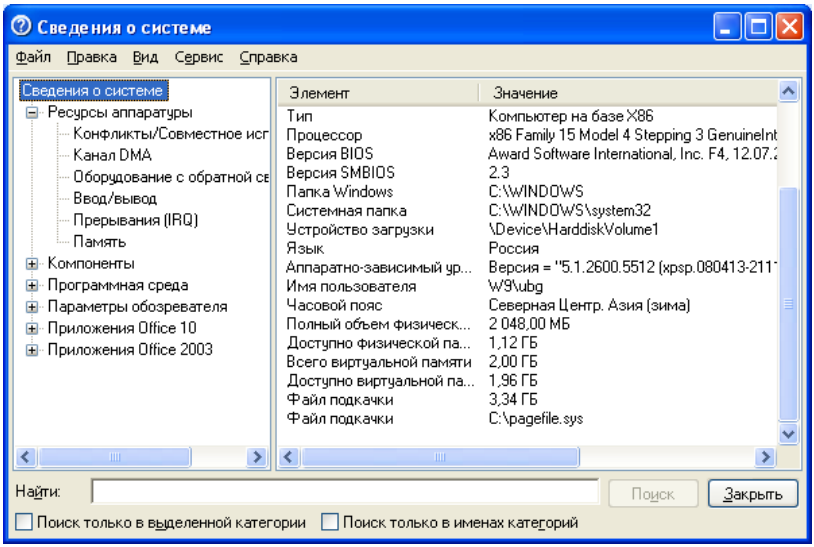

Рис. 2.15 — Окно приложения «Сведения о системе»

В этом приложении можно узнать полный объем физической, виртуальной и свободной в данный момент памяти, размещение и объем файла подкачки и т.п.

2.4.3 Архитектура памяти в ОС Microsoft Windows 9x

С точки зрения базовой архитектуры операционные системы Windows 9x являются 32-разрядными, многопотоковыми ОС с вытесняющей многозадачностью. При загрузке графического 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.

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 9x каждая 32-разрядная прикладная программа выполняется в своем собственном адресном пространстве, 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. Компоненты системы Windows 95, относящиеся к кольцу 0 (наиболее привелигированному), 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.

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 (Application Program Interface) 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 рисунка 2.16.

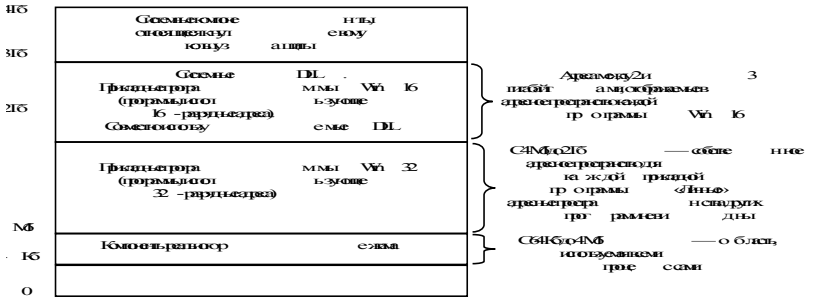

Рис. 2.16 — Модель распределения памяти в ОС Windows 9x

2.4.4 Архитектура памяти в ОС Microsoft Windows на платформе NT

В операционных системах на платформе Windows NT тоже используется плоская модель памяти. Однако схема распределения возможного виртуального адресного пространства в системах Windows NT разительно отличается от модели памяти Windows 9x. Прежде всего, в отличие от Windows 9x в Windows 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 рисунке 2.17.

Прикладным программам выделяется 2 Гбайта локального (собственного) линейного (неструктурированного) адресного пространства от границы 64 Кбайт до 2 Гбайт.

Нижние 64 Кбайта каждого виртуального адресного пространства в обычном состоянии не отображаются на физическую память. Это делается для облегчения перехвата программных ошибок (выявления недействительныхуказателей).

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).

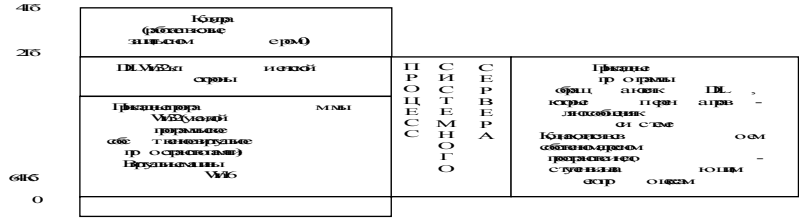

Рис. 2.17 — Модель распределения памяти в ОС Windows NT

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 Гбайта 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-раз-рядные 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 вытесняющей многозадачности, основанной на управлении отдельными потоками. Множественные 16-разрядные прикладные программы Win-dows в одном сеансе WOW выполняются в соответствии с кооперативной моделью многозадачности. Windows NT может также выполнять в многозадачном режиме несколько сеансов DOS.

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. 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. 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.

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 выделить 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 памяти относится 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.

2.4.5 Использование механизмов работы с памятью в ОС на платформе Microsoft Windows NT

Операционная система Windows реализует три механизма работы с памятью [13]:

1) виртуальную память — для операций с большими массивами объектов или структур;

2) проецируемые в память файлы — для операций с большими потоками данных (обычно из файлов) и для совместного использования данных несколькими процессами на одном компьютере;

3) кучи — для работы с множеством малых объектов.

Функции, работающие **с виртуальной памятью**, позволяют напрямую резервировать регион адресного пространства, передавать ему физическую память из страничного файла и присваивать любые допустимые атрибуты защиты.

Адресное пространство, выделенное процессу в момент создания, практически все свободно (не зарезервировано). Поэтому чтобы воспользоваться какой-нибудь его частью, нужно выделить в нем определенные области, используя функцию Win32 API VirtualAlloc. Выделенные области всегда начинаются с 64-килобайтных страниц (предполагается, что это сведет к минимуму проблемы переноса системы на компьютеры будущего с большим размером страниц). Количество выделенного адресного пространства может быть меньше, чем 64 Кбайта, но оно должно состоять из целого числа страниц.

Для использования зарезервированного региона (области) адресного пространства ему нужно выделить физическую память и спроецировать ее на этот регион. Такая операция называется передачей физической памяти. Делается это с помощью той же функции VirtualAlloc. Передавая физическую память, нет необходимости отводить ее целому региону. Например, зарезервировав регион размером 64 Кбайта, можно передать физическую память только его второй и пятой страницам. Если физическая память, переданная зарезервированному региону, больше не нужна, ее освобождают вызовом функции VirtualFree.

Рассматривая физическую память современных операционных систем, нужно помнить, что они «умеют» имитировать память за счет дискового пространства, создавая страничный файл (paging file). С точки зрения прикладной программы, страничный файл просто увеличивает объем памяти, которой она может пользоваться. Следовательно, физическую память следует

рассматривать как данные, хранимые в дисковом файле со страничной структурой.

Поэтому, когда приложение передает физическую память какому-нибудь региону адресного пространства, она на самом деле выделяется из файла, размещенного на жестком диске. Таким образом, размер страничного файла — главный фактор, определяющий количество физической памяти, доступной приложению. Реальный объем оперативной памяти имеет меньшее значение. Однако надо помнить о том, что процессор проецирует виртуальный адрес на физический только тогда, когда данные находятся в оперативной памяти.

Чем чаще системе приходится копировать страницы памяти в страничный файл и обратно, тем больше нагрузка на жесткий диск и тем медленнее работает операционная система. Получается так, что система будет тратить все свое время на подкачку страниц вместо выполнения программы. Поэтому, увеличив объем оперативной памяти, можно снизить частоту обращения к диску и тем самым увеличить общую производительность системы.

Не следует думать, что страничный файл сильно увеличивается при одновременном выполнении нескольких программ за счет того, что система при каждом запуске приложения резервирует регионы адресного пространства, передает им физическую память, а затем копирует код и данные из файла программы (расположенного на диске) в физическую память, переданную из страничного файла. Операционная система действует не так, иначе на загрузку и подготовку программы к запуску уходило бы слишком много времени.

На самом деле при запуске приложения система открывает его исполняемый файл и определяет объем кода и данных. Затем резервирует регион адресного пространства и помечает, что физическая память, связанная с этим регионом, — сам EXE-файл, то есть вместо выделения какого-то пространства из страничного файла система использует образ EXE-файла как зарезервированный регион адресного пространства программы. Благодаря этому приложение загружается очень быстро, а размер страничного файла удается существенно уменьшить.

Образ исполняемого файла (т.е. EXE- или DLL-файл), размещенный на диске и применяемый как физическая память для того или иного региона адресного пространства, называется **проецируемым в память файлом** (memory-mapped file). При загрузке EXE· или DLL-файлов система автоматически резервирует регион адресного пространства и проецирует на него образ файла. Помимо этого, система позволяет проецировать на регион адресного пространства и файлы данных.

Проецируемые файлы применяются в следующих случаях [13]:

– загрузка и выполнение EXE- и DLL-файлов. Это позволяет существенно экономить как на размере страничного файла, так и на времени, необходимом для подготовки приложения к выполнению;

– организация доступа к файлу данных, размещенному на диске. При этом можно исключить операции файлового ввода-вывода и буферизацию его содержимого;

– разделение данных между несколькими процессами, выполняемыми на одной машине.

Рассмотрим процесс загрузки и выполнения EXE- и DLL-файлов. При выполнении функции CreateProcess (создать процесс) операционная система действует следующим образом [13]:

1. Отыскивается EXE-файл, указанный в вызове функции, и, если он не найден, новый процесс не создается, а функция возвращает значение FALSE.

2. Если файл найден, создается новый объект ядра «процесс».

3. Создается адресное пространство нового процесса.

4. Резервируется такой регион адресного пространства, чтобы в него поместился данный EXE-файл. Желательное расположение этого региона указывается внутри самого EXE-файла. По умолчанию базовый адрес EXE-файла — 00400000h. При создании исполняемого файла приложения базовый адрес может быть изменен параметром компоновщика /BASE.

5. Система отмечает, что физическая память, связанная с зарезервированным регионом, — EXE-файл на диске, а не страничный файл.

Спроецировав EXE-файл на адресное пространство процесса, система обращается к разделу EXE-файла, содержащему список DLL, которые обеспечивают программе необходимые функции. После этого система, вызывая функцию LoadLibrary, поочередно загружает указанные DLL-модули. Каждый раз, когда для загрузки DLL вызывается LoadLibrary, операционная система выполняет действия, аналогичные описанным в строках 4 и 5 процесса загрузки.

Если система по какой-либо причине не свяжет EXE-файл с необходимыми ему DLL, на экране появится соответствующее сообщение, а адресное пространство процесса и объект «процесс» будут освобождены. При этом функция CreateProcess вернет значение FALSE.

После увязки EXE- и DLL-файлов с адресным пространством процесса начинает исполняться стартовый код EXE-файла. Подкачку страниц, буферизацию и кэширование выполняет система. Например, если код в EXE-файле переходит к команде, не загруженной в оперативную память, возникает ошибка. Обнаружив ее, система перекачивает нужную страницу кода из образа файла на страницу оперативной памяти. Затем отображает страницу оперативной памяти на должный участок адресного пространства процесса, позволяя потоку продолжить выполнение кода. Все эти операции скрыты от приложения и периодически повторяются при каждой попытке процесса обратиться к коду или данным, отсутствующим в оперативной памяти.

Операционная система позволяет проецировать на адресное пространство процесса и файл данных. Для этого нужно выполнить три операции [13]:

1. Создать или открыть объект ядра «файл», идентифицирующий дисковый файл, который предполагается использовать как проецируемый в память. Для этого используется функция CreateFile.

2. Создать объект ядра «проекция файла» с помощью функции CreateFileMapping, чтобы сообщить системе размер файла и способ доступа к нему.

3. Указать системе, как спроецировать в адресное пространство процесса объект «проекция файла» — целиком или частично. Это делает функция MapViewOfFile.

Закончив работу с проецируемым в память файлом, следует выполнить так же три операции [13]:

1. Сообщить системе об отмене проецирования на адресное пространство процесса объекта ядра «проекция файла». Для этого предусмотрена функция UnmapViewOfFile.
2. Закрыть объект «проекция файла».
3. Закрыть объект файл. Последние два действия осуществляются с помощью функции CloseHandle.

Опишем теперь возможность ***разделения данных между несколькими процессами, выполняемыми на одной машине.*** Создавать файл на диске и хранить там данные только с этой целью неудобно. Поэтому в Windows предусмотрена возможность проецирования файлов непосредственно на физическую память из страничного файла, а не из специально создаваемого дискового файла. Этот способ проще стандартного, основанного на создании дискового файла, проецируемого в память.

Создав объект «проекция файла» и спроецировав его представление на адресное пространство своего процесса, его можно использовать так же, как и любой другой регион памяти. Для того чтобы данные стали доступны другим процессам, нужно вызвать функцию CreateFileMapping и передать в параметре pszName строку с нулевым символом в конце. Тогда посторонние процессы, если им понадобится доступ к этому же файлу, смогут вызвать CreateFileMapping или OpenFileMapping и передать ей то же имя.

Когда необходимость в доступе к объекту «проекция файла» отпадет, процесс должен вызвать функцию CloseHandle. Как только все описатели объекта будут закрыты, система освободит память, переданную из страничного файла.

Использование виртуальной памяти, как это показано выше, не всегда удобно. Например, связанные списки и деревья проще обрабатывать, используя **кучи (heaps)** [13]. Преимущество динамически распределяемой памяти заключается в том, что она позволяет игнорировать гранулярность выделения памяти и размер страниц и сосредоточиться непосредственно на задаче. Но есть и недостаток в использовании этого механизма управления памятью, заключающийся в более медленном выделении и освобождении блоков памяти. Кроме того, в этом слу-

чае теряется прямой контроль над передачей физической памяти и ее возвратом системе.

Куча — это регион зарезервированного адресного пространства. Первоначально большей его части физическая память не передается. По мере того как программа занимает эту область под данные, специальный диспетчер, управляющий кучами (heap manager), постранично передает ей физическую память из страничного файла. При освобождении блоков в куче диспетчер возвращает системе соответствующие страницы физической памяти.

При инициализации процесса операционная система создает в его адресном пространстве стандартную кучу и автоматически уничтожает ее по завершении процесса. Размер стандартной кучи по умолчанию — 1 Мбайт. Но система позволяет увеличивать этот размер, для чего надо указывать компоновщику при сборке программы ключ /HEAP. Поскольку стандартную кучу процесса используют многие Windows-функции, а потоки приложения могут одновременно вызвать массу таких функций, то доступ к этой куче возможен только по очереди. Это несколько снижает производительность многопоточной программы.

Организация использования куч во многом зависит от языка и системы программирования, которые использовались при разработке приложения. Поэтому использование куч лучше изучать в курсах программирования.

2.4.6 Управление файлом подкачки на платформе Microsoft Windows NT

Файл подкачки — это область жесткого диска, используемая Windows для хранения данных оперативной памяти. Он создает иллюзию, что система располагает большим объемом оперативной памяти, чем это есть на самом деле. Единой стратегии работы с файлом подкачки не существует. Многое определяется назначением и настройкой компьютера.

По умолчанию Windows удаляет файл подкачки после каждого сеанса работы и создает его в процессе загрузки операционной системы. Размер файла постоянно меняется по мере выполнения приложений и контролируется операционной систе-

мой. Обычно используется единственный файл подкачки, расположенный на том же диске, что и операционная система. Такой подход не является лучшим и, более того, практически всегда плох. В этом случае возникает несколько проблем [13]:

1. Память может неожиданно оказаться исчерпанной из-за того, что приложение создало на жестком диске файл большого объема или операционная система без предупреждения увеличила потребности файла подкачки. Большой файл подкачки приводит к дефициту дискового пространства и к увеличению непроизводительных затрат на организацию страничного обмена.

2. Файл подкачки фрагментируется, что приводит не только к медленному считыванию жесткого диска, но и к дополнительным перемещениям считывающей головки диска, а в итоге — к существенному снижению производительности.

3. Файл подкачки фрагментируется сам по себе и очень быстро, причем так, что одна и та же область памяти может оказаться в разных местах жесткого диска. В этом случае даже отдельные приложения не могут получить доступ к памяти без нескольких обращений к диску.

4. Производительность системы падает и в том случае, если операционная система установлена не на самом быстром из жестких дисков, имеющихся в компьютере.

Наличие двух жестких дисков может дать значительное преимущество при настройке файла подкачки. Для максимально эффективного использования файла подкачки нужно так его настроить, чтобы он располагался на жестком диске в виде достаточно протяженных фрагментов (это уменьшает количество перемещений считывающей головки, радикально влияющих на производительность). Кроме того, файл подкачки необходимо периодически удалять, чтобы избежать его фрагментации.

Для установки размера файла подкачки нужно выполнить следующую последовательность действий. Щелкнуть правой клавишей мыши по значку «Мой компьютер» и выбрать в контекстном меню строку «Свойства». На экране появится окно «Свойства системы». Перейти на вкладку «Дополнительно» и нажать кнопку «Параметры» в рамке «Быстродействие» (рис. 2.18). В появившемся окне «Параметры быстродействия» на вкладке «Дополнительно» нажать кнопку «Изменить». Предва-

рительно следует выбрать принцип распределения времени процессора (для оптимизации работы программ, если это пользовательский компьютер, или служб, работающих в фоновом режиме, если это сервер). Кроме того, следует задать режим использования памяти. Для пользовательского компьютера — оптимизировать работу программ, для сервера — системного кэша.

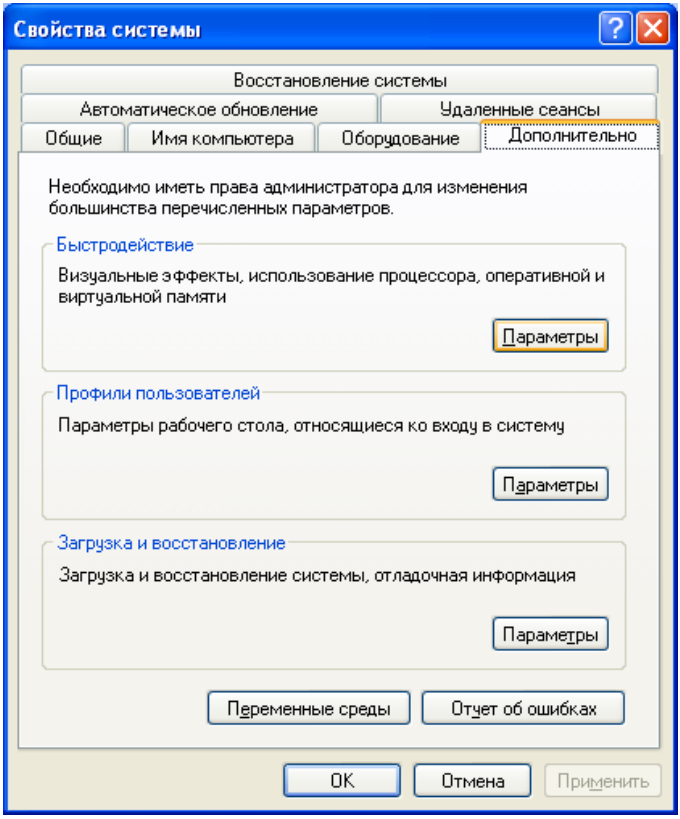

Рис. 2.18 — Вид окна «Свойства системы» на вкладке «Быстродействие»

Определение размера файла подкачки до сих пор вызывает многочисленные дискуссии. Основное правило заключается в том, что при небольшом объеме оперативной памяти файл под-

качки должен быть достаточно большим. При большом объеме оперативной памяти (512 Мбайт и более) файл подкачки можно уменьшить. Существует возможность вообще ликвидировать файл подкачки, выполнив определенную настройку реестра. Однако в этом случае объем оперативной памяти должен быть достаточно большим, лишь немногие системы обладают подобными ресурсами.

По рекомендациям некоторых авторов можно установить исходный размер файла подкачки, равный размеру физической памяти, а максимальный размер не более двух размеров физической памяти. После этого следует нажать кнопку «Задать» (рис. 2.19) и убедиться в том, что новое значение файла подкачки установлено. Далее щелкнуть на кнопке «OK». Windows XP выведет сообщение о том, что данное изменение требует перезагрузки компьютера. Теперь нужно перезагрузить компьютер.

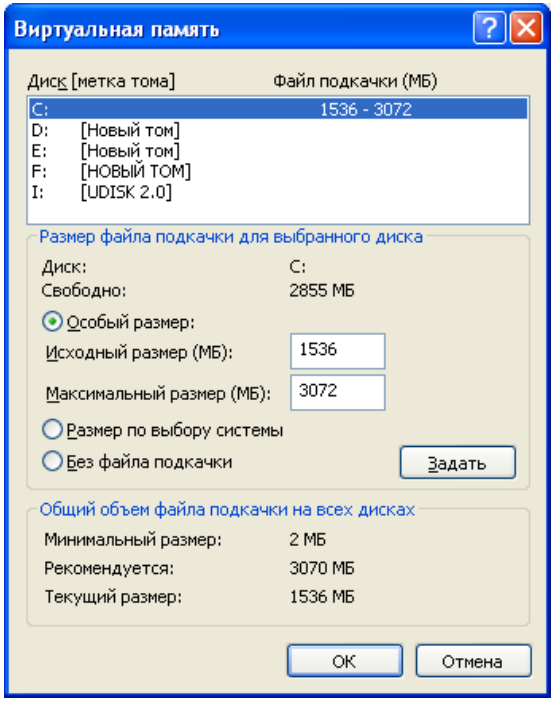

Рис. 2.19 — Вид окна «Виртуальная память»

Следует иметь в виду, что при первом создании файла подкачки жесткий диск, как правило, не готов к его размещению. Это обусловлено фрагментацией жесткого диска.

Поэтому нужно вначале выполнить дефрагментацию диска и лишь затем создать файл подкачки, чтобы поместить его в единственную область диска. Последовательность действий может быть, например, такой [13]:

1) если в компьютере имеется единственный жесткий диск, установить минимальный размер файла подкачки (2 Мбайт);

2) если имеется два жестких диска, переместить файл подкачки на более медленный диск;

3) провести дефрагментацию диска (во втором случае — быстрого). Для полной дефрагментации нужно выполнить несколько проходов;

4) присвоить файлу подкачки желаемый размер.

В результате работа с файлом подкачки станет максимально быстрой, а процессорная мощность и дисковое пространство будут использоваться эффективно.

Существует несколько приемов, позволяющих оптимизировать использование файла подкачки для повышения производительности. Файл подкачки следует по возможности размещать на отдельном жестком диске.

При наличии нескольких жестких дисков файл подкачки следует разделить, т.к. это повышает скорость работы с ним. Если при наличии двух жестких дисков разделить файл подкачки, то поскольку доступ к данным на обоих жестких дисках осуществляется одновременно, это значительно повысит производительность. Однако если имеются два жестких диска, из которых один быстрее другого, возможно, более эффективным решением будет размещение файла подкачки только на более быстром жестком диске. Определить наиболее производительную конфигурацию для данной системы можно экспериментальным путем.

Вопросы для самопроверки

1. Опишите машину Фон-Неймана и ее принципы.
2. Что понимают под определением «суперскалярная архитектура»?

3. Назовите, какие единицы информации вы знаете?

4. Как осуществляется перевод из десятичной системы исчисления в шестнадцатеричную и обратно?

5. Опишите структуру памяти на современных персональных компьютерах.

6. Что в себя включает программная модель микропроцессора?

7. Какие регистры входят в группу регистров общего назначения?

8. Какие регистры входят в группу сегментных регистров?

9. Какие регистры входят в группу регистров состояния и управления?

10. Какие регистры входят в группу системных регистров?

11. Приведите перечень режимов функционирования процессора Intel x86.

12. Как строится адресация ячеек памяти в реальном режиме?

13. Приведите распределение оперативной памяти в MS DOS.

14. Приведите распределение Conventional memory.

15. Для чего использовалась верхняя память UMA?

16. Для чего использовалась теневая память Shadow ROM и Shadow RAM?

17. Опишите процесс сегментации страниц в защищенном режиме.

18. Опишите процесс трансляции страниц в защищенном режиме.

19. Опишите процесс формирования адреса памяти 32-разрядных процессоров в защищенном режиме.

20. Какие сегментные модели могут быть реализованы с использованием механизма сегментации в защищенном режиме?

21. Как выглядит механизм переключения в защищенный режим?

22. Опишите механизм защиты в защищенном режиме.

23. Опишите механизм обработки прерываний в защищенном режиме.

24. Какие элементы определяют состояние текущей выполняемой задачи?

25. Что собой представляет режим системного управления?

26. Что собой представляет режим Virtual-86?

27. Для каких целей можно использовать утилиту debug.exe d ОС Windows?

28. С помощью каких средств можно получить информацию об использовании памяти в ОС Windows?

29. Опишите архитектуру памяти ОС Windows 9x?

30. Опишите архитектуру памяти ОС Windows на платформе NT?

31. Какие существуют в ОС Windows механизмы работы с памятью?

32. Как можно настроить управление файлом подкачки в ОС Windows?

33. Опишите процесс работы виртуальной памяти в ОС Windows.

34. Опишите процесс проецирования файлов в память в ОС Windows.

35. Опишите процесс распределения куч в ОС Windows.

3 УПРАВЛЕНИЕ УСТРОЙСТВАМИ ВВОДА-ВЫВОДА

3.1 Описание устройств ввода-вывода

3.1.1 Классификация устройств ввода-вывода

Внешние устройства, выполняющие операции ввода-вывода, можно разделить на три группы [6]:

1) ***устройства, работающие с пользователем.*** Используются для связи пользователя с компьютером. Сюда относятся принтеры, дисплеи, клавиатура, манипуляторы (мышь, трекбол, джойстики) и т.п.;

2) ***устройства, работающие с компьютером.*** Используются для связи с электронным оборудованием. К ним можно отнести дисковые устройства и устройства с магнитными лентами, датчики, контроллеры, преобразователи;

3) ***коммуникации.*** Используются для связи с удаленными устройствами. К ним относятся модемы и адаптеры цифровых линий.

По другому признаку устройства ввода-вывода можно разделить на ***блочные и символьные***. **Блочными** являются устройства, хранящие информацию в виде блоков фиксированного размера, причем у каждого блока есть адрес и каждый блок может быть прочитан независимо от остальных блоков [6]. 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).

Символьные устройства принимают или передают поток символов без какой-либо блочной структуры (принтеры, сетевые карты, мыши и т.д.) [6].

Однако некоторые из устройств не попадают ни в одну из этих категорий, например часы, мониторы и др. И все же модель блочных и символьных устройств является настолько общей, что может использоваться в качестве основы для достижения независимости от устройств некоторого программного обеспечения операционных систем, имеющего дело с вводом-выводом. Например, файловая система имеет дело с абстрактными блоч-

ными устройствами, а зависимую от устройств часть оставляет программному обеспечению низкого уровня.

По методу доступа к информации устройства внешней памяти разделяются на устройства ***с прямым*** (или непосредственным) и ***последовательным*** доступом [15]. **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.

Разнообразие внешних устройств приводит, по сути, к невозможности разработки единого и согласованного подхода к проблеме ввода-вывода как с точки зрения операционной системы, так и с точки зрения пользовательских процессов. На рисунке 3.1 даны сведения по скорости передачи данных различных устройств ввода-вывода [6].

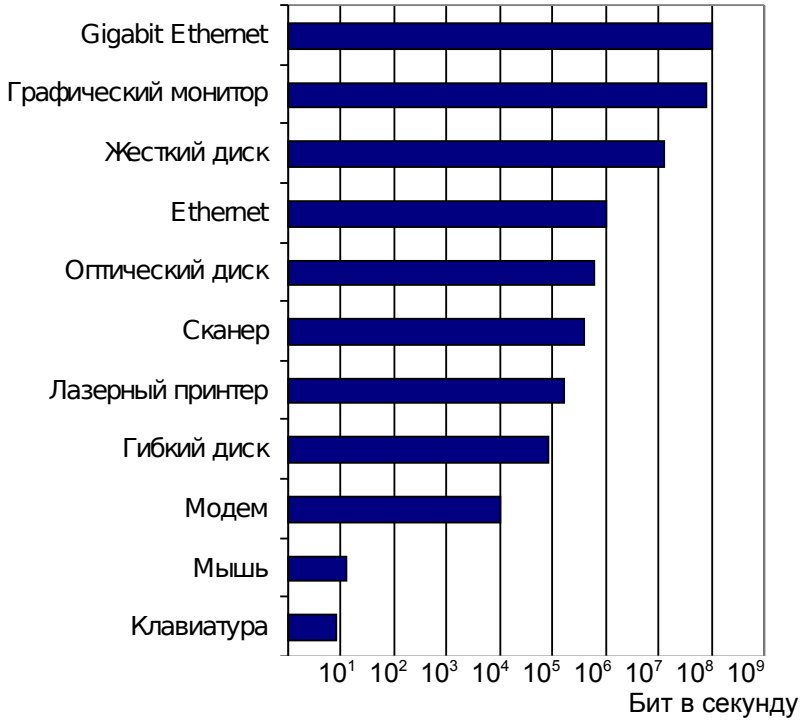

Рис. 3.1 — Соотношения скоростей передачи данных у различных устройств ввода-вывода

3.1.2 Основные характеристики устройств внешней памяти

Главная характеристика устройств — **емкость хранения**, 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 [15]. 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[5] 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

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

числу секторов на треке, но об этих понятиях будет сказано чуть позже.

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.

3.1.3 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 [15]. 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** [15].

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 по-

скольку 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 указывается минимальное и максимальное значение скорости, соответствующее 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 высокопроизводительных 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.

3.2 Организация работы устройств ввода-вывода

3.2.1 Организация операций ввода-вывода

Обмен данными между пользователями, приложениями и периферийными устройствами компьютера выполняет специальная подсистема ОС — подсистема ввода-вывода. Собственно для выполнения этой задачи и были разработаны первые системные программы, послужившие прототипами операционных систем.

Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие внешними устройствами, и файловая система. В работе подсистемы ввода-вывода активно участвует диспетчер прерываний. Более того, основная нагрузка диспетчера прерываний обусловлена именно подсистемой ввода-вывода, поэтому диспетчер прерываний иногда считают частью подсистемы ввода-вывода [6].

На подсистему ввода-вывода возлагаются следующие функции [6]:

– организация параллельной работы устройств ввода-вывода и процессора;

– согласование скоростей обмена и кэширование данных;

– разделение устройств и данных между процессами (выполняющимися программами);

– обеспечение удобного логического интерфейса между устройствами и остальной частью системы;

– поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера;

– динамическая загрузка и выгрузка драйверов без дополнительных действий с операционной системой;

– поддержка нескольких различных файловых систем;

– поддержка синхронных и асинхронных операций ввода-вывода.

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами [6]:

1. ***С помощью программируемого ввода-вывода.*** В этом случае, когда процессору встречается команда, связанная с вводом-выводом, он выполняет ее, посылая соответствующие команды контроллеру ввода-вывода. Это устройство выполняет требуемое действие, а затем устанавливает соответствующие биты в регистрах состояния ввода-вывода и не посылает никаких сигналов, в том числе сигналов прерываний. Процессор периодически проверяет состояние модуля ввода-вывода с целью проверки завершения операции ввода-вывода.

Таким образом, процессор непосредственно управляет операциями ввода-вывода, включая опознание состояния устройства, пересылку команд чтения-записи и передачу данных. Процессор посылает необходимые команды контроллеру ввода-вывода и переводит текущий процесс в состояние ожидания завершения операции ввода-вывода. Недостатками такого метода являются большие потери процессорного времени, связанные с управлением вводом-выводом.

2. ***Ввод-вывод, управляемый прерываниями.*** Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять текущий процесс, если нет необходимости в ожидании выполнения операции ввода-вывода. В противном случае текущий процесс приостанавливается до получения сигнала прерывания о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд.

Такой ввод-вывод намного эффективнее, чем программируемый ввод-вывод, так как при этом исключается ненужное ожидание с бесполезным простоем процессора, однако и в этом случае ввод-вывод потребляет еще значительное количество про-

цессорного времени, потому что каждое слово, которое передается из памяти в модуль ввода-вывода (контроллер) или обратно, должно пройти через процессор.

3. ***Прямой доступ к памяти*** (direct memory access — DMA). В этом случае специальный модуль прямого доступа к памяти управляет обменом данными между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных.

В настоящее время в персональных и других компьютерах используется третий способ ввода-вывода, поскольку в структуре компьютера имеется DMA-контроллер или подобное ему устройство, обслуживающее, как правило, запросы по передаче данных от нескольких устройств ввода-вывода на конкурентной основе [6].

3.2.2 Драйверы

Первоначально термин «драйвер» применялся в достаточно узком смысле; под драйвером понимается программный модуль, который [6]:

– входит в состав ядра ОС, работая в привилегированном режиме;

– непосредственно управляет внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;

– обрабатывает прерывания от контроллера устройства;

– предоставляет прикладному программисту удобный логический интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных;

– взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться, и т.п.

Согласно этому определению драйвер вместе с контроллером устройства и прикладной программой воплощали идею многослойного подхода к организации программного обеспечения. Контроллер представлял низкий слой управления устройством, выполняющий операции в терминах блоков и агрегатов устройства (например, передвижение головки дисковода, побитную передачу байта по двухпроводному кабелю). Драйвер выполнял более сложные операции, преобразуя данные, адресуемые в терминах номеров цилиндров, головок и секторов диска, в линейную последовательность блоков. В результате прикладная программа работала с данными, преобразованными в достаточно понятную форму, файлами, таблицами баз данных, текстовыми окнами на мониторе и т.п., не вдаваясь в детали представления этих данных в устройствах ввода-вывода [6].

В описанной схеме драйверы не делились на слои. ***Постепенно, по мере развития операционных систем и усложнения структуры подсистемы ввода-вывода, наряду с традиционными драйверами в ОС появились так называемые высокоуровневые драйверы, которые располагаются в общей модели подсистемы ввода-вывода над традиционными драйверами.*** Появление таких драйверов можно считать развитием идеи многоуровневой организации подсистемы ввода-вывода, когда ее функции декомпозируются между несколькими модулями в соседних слоях иерархии (таких примеров много, например семиуровневая модель сетевых протоколов) [6].

Традиционные драйверы, которые стали называть аппаратными, низкоуровневыми, или драйверами устройств, освобождаются от высокоуровневых функций и занимаются только низкоуровневыми операциями. Эти низкоуровневые операции составляют фундамент, на котором можно построить тот или иной набор операций в драйверах более высоких уровней [6].

При таком подходе повышается гибкость и расширяемость функции по управлению устройством. Например, если различным приложениям необходимо работать с различными логическими модулями одного и того же физического устройства, то для этого в системе достаточно установить несколько драйверов на одном уровне, работающих над одним аппаратным драйвером. Несколько драйверов, управляющих одним устройством,

но на разных уровнях, можно рассматривать как один многоуровневый драйвер [6].

На практике используют от двух до пяти уровней драйверов, поскольку с увеличением числа уровней снижается скорость выполнения операций ввода-вывода [6].

Высокоуровневые драйверы оформляются по тем же правилам и придерживаются тех же внутренних интерфейсов, что и аппаратные драйверы. Как правило, высокоуровневые драйверы не вызываются по прерываниям, так как взаимодействуют с устройством через посредничество аппаратных драйверов.

В модулях подсистемы ввода-вывода кроме драйверов могут присутствовать и другие модули, например **дисковый кэш**. Достаточно специфичные функции кэша делают нецелесообразным оформление его в виде драйвера, взаимодействующего с другими модулями ОС только с помощью услуг менеджера ввода-вывода. Другим примером модуля, который чаще всего не оформляется в виде драйвера, является диспетчер окон графического интерфейса. Иногда этот модуль вообще выносится из ядра ОС и реализуется в виде пользовательского интерфейса [6].

3.2.3 Файловые системы

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 [3]. 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 персональных компьютерах.

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. 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 указани-

ем 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 соответствующая 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.

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых входят обычные файлы, содержащие информацию произвольного ха-

рактера (текст, графика, звук и др.), файлы-каталоги, специальные файлы, именованные конвейеры, отображаемые в память файлы и др.

Обычные файлы, или просто файлы или регулярные файлы, содержат информацию, которую в них заносит пользователь или которая образуется в результате работы системных и пользовательских программ. Большинство ОС не контролируют содержимое и структуру регулярных файлов, которые в основном являются ASCII-файлами либо двоичными файлами. ***ASCII-файлы*** состоят из текстовых строк. Они могут отображаться на экране и выводиться на печать без какого-либо преобразования, и могут редактироваться практически любым текстовым редактором. ***Двоичные файлы*** имеют определенную внутреннюю структуру, которая известна программе, использующей данный файл. При выводе двоичного файла на принтер получается случайный набор символов.

Каталоги — это системные файлы, обеспечивающие поддержку структуры файловой системы. Они содержат системную справочную информацию о наборе файлов, сгруппированных пользователем по какому-либо неформальному признаку (договоры, рефераты, курсовые проекты и т.п.). Во многих ОС в каталог могут входить другие файлы, в том числе другие каталоги, за счет чего образуется древовидная структура, удобная для поиска требуемого файла. Каталоги устанавливают соответствие между именами файлов и их характеристиками, используемыми файловой системой для управления файлами. В число таких характеристик входит тип файла, права доступа к файлу, его распоряжение на диске, размер, дата и время создания и др.

Специальные файлы — это фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к последовательным устройствам ввода-вывода, таким, как терминалы, принтеры и др. (например, MS DOS рассматривает монитор и клавиатуру как файлы со стандартным именем con — консоль, а принтер — как файл prn). Блочные специальные файлы используются для моделирования дисков.

Именованные конвейеры (каналы) представляют собой циклические буферы, позволяющие выходной файл одной программы соединить с входным файлом другой программы.

Наконец, ***отображаемые файлы*** — это обычные файлы, отображенные на адресное пространство процесса по указанному виртуальному адресу.

Файлы относятся к абстрактному механизму. Они предоставляют способ сохранять информацию на запоминающем устройстве и считывать ее позднее снова. При этом от пользователя должны скрываться такие детали, как способ и место хранения информации, а также детали работы устройства.

Наиболее важной характеристикой любого механизма абстракции является именование управляемых объектов. Правила именования файлов меняются от одной ОС к другой, но, в основном, все современные операционные системы поддерживают использование в качестве имен файлов 8-символьные текстовые строки. Часто в именах разрешается использование цифр и специальных символов. В некоторых файловых системах различаются прописные и строчные символы, тогда как в других, например в MS DOS, нет.

Во многих операционных системах имя файла состоит из двух частей, разделенных точкой. Часть имени после точки называется ***расширением*** файла и обычно означает его тип. Так, в MS DOS имя файла может содержать от 1 до 8 символов, а расширение от 0 (отсутствует) до 3.

В некоторых ОС, например Windows, расширение указывает на программу, создавшую файл. Другие ОС, например UNIX, не принуждают пользователя строго придерживаться расширений.

В иерархически организованных файловых системах обычно используются три типа имен файлов: простые, составные и относительные.

Простое (короткое) символьное имя идентифицирует файл в пределах одного каталога. Несколько файлов могут иметь одно и то же простое имя, если они принадлежат разным каталогам.

Составное (полное) символьное имя представляет собой цепочку, содержащую имя диска и имена всех каталогов, через которые проходит путь от корневого каталога до данного файла.

Относительное имя файла определяется через текущий каталог, т.е. каталог, в котором в данный момент времени работает пользователь. Таким образом, относительных имен у файла может быть достаточно много, и все они являются частью полного имени.

Понятие файла включает не только хранимые им данные и имя, но и информацию, описывающую свойства файла. Эта информация составляет **атрибуты (дескриптор) файла**. Список атрибутов может быть различным в различных ОС.

3.3 Организация дисковых устройств

3.3.1 Физическая структура магнитного диска

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*** (sector), 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 операционных систем, как правило, поддерживает размер сектора — 512 байт. 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, то можно обрабатывать секторы и с другими размерами.

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.

3.3.2 Логическая структура магнитного диска

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. Раз-

делы 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 компьютера 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:

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) [3]. 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 lBEh 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 таблице 3.1.

Таблица 3.1 — **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 таблице 3.1 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 следует 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 принадлежность 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. 3.2).

Таблица 3.2 — **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 таблице 3.3 приведены наиболее известные идентификаторы.

Таблица 3.3 — **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

Окончание табл. 3.3

System ID, идентификатор	Тип раздела	System ID, идентификатор	Тип раздела
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	HiddenFAT16(<32Мбайт)	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		

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 ко-

личество. Каждый логический диск «управляется» своим логическим приводом. Каждому логическому диску на винчестере соответствует своя (относительная) ***логическая*** нумерация. Физическая же адресация жесткого диска сквозная.

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 используется утилита fdisk.exe, а в Windows на платформе NT используется «Панель управления», далее «Администрирование», «Управление компьютером». Существует также очень удобная, но не входящая в состав операционных систем утилита Partition Magic (фирмы PowerQuest), 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.

3.4 Обзор файловых систем

3.4.1 Файловая система FAT

FAT16

Файловая система FAT16 была разработана еще до создания MS DOS и в настоящее время поддерживается всеми операционными системами Microsoft для обеспечения совместимости,

а так же большим количеством операционных систем других производителей. 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.

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 таблице 3.4 показана зависимость размера кластера от размера тома. Размер кластера может отличаться от значения по умолчанию, но должен иметь одно из значений, указанных в таблице 3.4.

Таблица 3.4 — **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 рисунке 3.2 показано, как организован том при использовании файловой системы FAT16.

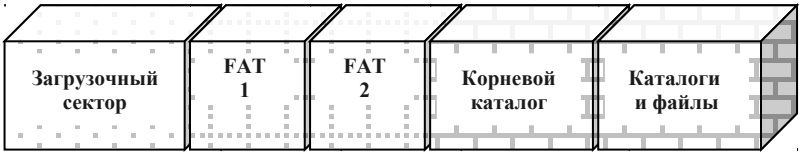

Рис. 3.2 — Структура тома при использовании FAT16

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 таблице 3.5.

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.

Таблица 3.5 — **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

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 рисунке 3.3: 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.

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).

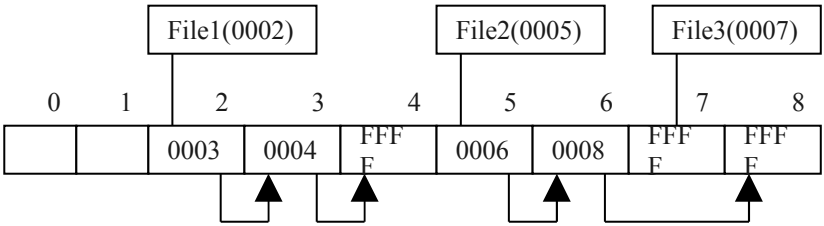

Рис. 3.3 — Пример расположения файлов по кластерам в FAT16

Среди ***преимуществ FAT16*** можно отметить следующие:

– 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:

– 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-мя 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.

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 Кбайта подходит для дисков объемом от 512 Мбайт до 8 Гбайт (табл. 3.6). 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.

При применении FAT32 максимальный размер файла может достигать 4 Гбайт минус 2 байта. Если при использовании FAT16 максимальное число вхождений в корневой каталог огра-

ничивалось 512, то FAT32 позволяет увеличить это число до 65 535.

Таблица 3.6 — **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

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 соответствующее 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:

– 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 Кбайта 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:

– 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.

3.4.2 Файловая система NTFS

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

ния 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 таблице 3.7 показаны размеры кластеров по умолчанию для томов различной емкости.

Таблица 3.7 — **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}. 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 Тбайта (2^{32} × 512 байт = = 2^{41}). ***Таким образом, размер тома в 2 Тбайта 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 пользователи 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 записывает 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 таблице 3.8.

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.

Таблица 3.8 — **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, необходимых для восстановления NTFS
Volume	$Volume	3	Информация о томе — метка и номер версии
Attribute definitions	$AttrDef	4	Таблица имен атрибутов и описание
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

Окончание табл. 3.8

Системные файлы	Имя файла	Запись MFT	Содержание записи MFT
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

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 Ownership** *(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 также 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 Кбайта. 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.

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.

3.4.3 Файловая система HPFS

HPFS (High Performance File System) — высокопроизводительная файловая система — впервые появилась в OS/2 1.2 и LAN Manager [3]. 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 кон-

кретного 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 сек-

торы (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 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 [3]. 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 (точнее, 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-х 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 Кбайтам. 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 сектора 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 Кбайта 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 Кбайта 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. Программы (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 Мбайтам. 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) [3].

Если файловая система HPFS сталкивается с проблемой в процессе записи данных на диск, то происходят следующие действия:

1. На экран выводится соответствующее сообщение об ошибке.

2. HPFS сохраняет информацию, которая должна 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.

3. 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.

4. 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 дефектных 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.

5. 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.

6. 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, со-

держат 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.***

3.4.4 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 рисунке 3.4.

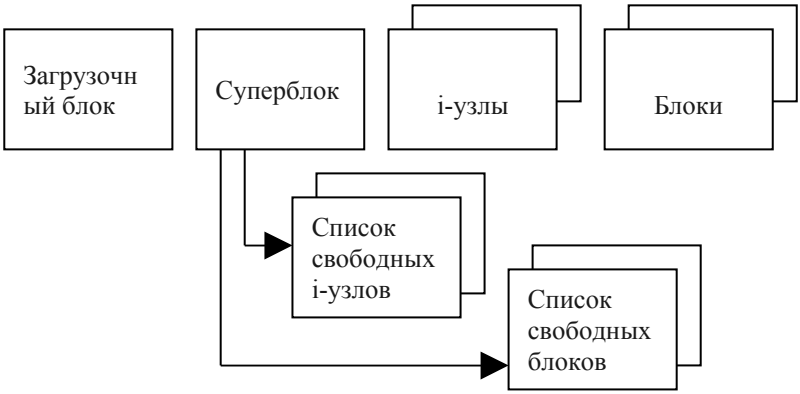

Рис. 3.4 — Структура файловой системы UNIX

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. 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×(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×(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×(10 + 128 + 128^2 + 128^3)). Другие версии системы UNIX могут отличаться количеством ссылок в i-узле, косвенных блоках и размером блока данных.

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 под-

ключает 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;

– могут возникнуть ситуации, когда некоторый блок является частью файла и одновременно присутствует в списке свободных блоков;

– могут существовать некоторые файлы, 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, соответствующий 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 являются 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.

3.4.5 Файловые системы для CD-ROM

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.

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.

Файловая система UDF

UDF (Universal Disk Format — универсальный дисковый формат) — спецификация формата файловой системы, независимой от операционной системы (ОС) для хранения файлов на оптических носителях (оптических дисках). UDF является реализацией стандарта ISO/IEC 13346 (известного также как ECMA-167).

Формат UDF призван заменить ISO 9660. ISO 9660 имеет некоторые ограничения, которые делают его несовместимым с DVD, CD-RW и другими новыми форматами дисков. UDF разработан так, чтобы избавить от этих ограничений. UDF позволяет дозаписывать файлы на CD-R- или CD-RW-дисках, один файл одновременно, без существенных потерь дискового пространства, используя метод пакетной записи. Также UDF учитывает возможность выборочного стирания некоторых файлов на перезаписываемых носителях CD-RW, освобождая место на диске. В стандарте ISO 9660 такое не предусмотрено. UDF также лучше подходит для DVD, так как имеет лучшую поддержку для дисков большого объёма — в частности ISO 9660 не поддерживает файлы размером более 2 Гб.

UDF разработан и развивается Optical Storage Technology Association (OSTA).

Существует несколько версий формата UDF:

– 1.02 (поддерживается Windows 98, многими версиями ОС корпорации Apple, возможно использовать для DVD-RAM и магнитооптических дисков);

– 1.50 (поддерживается Windows 2000, Windows XP и Windows Server 2003; может быть несовместим с Windows 98 или Apple);

– 2.01 (поддерживается Windows XP и Windows Server 2003; может быть несовместим с Windows 98, Windows 2000 или Apple);

– 2.50 (поддерживается Windows Vista; может быть несовместим с более ранними версиями Windows и др. платформами);

– 2.60 (поддерживается Windows Vista, Mac OS X 10.5, NetBSD).

3.5 Управление устройствами ввода-вывода и файловыми системами в ОС Windows

3.5.1 Диспетчер устройств и драйвера устройств

Задача системы ввода-вывода ОС Windows заключается в предоставлении основных средств (каркаса) для эффективного управления широким спектром устройств ввода-вывода. Основу этих средств образует набор независимых от устройств процедур для определенных аспектов ввода-вывода и набор загруженных драйверов для общения с устройствами. Формирует этот каркас ***менеджер ввода-вывода***, который предоставляет остальной операционной системе независимый от устройств ввод-вывод, вызывая для выполнения физического ввода-вывода соответствующий драйвер [13].

Файловые системы формально являются драйверами устройств, работающих под управлением менеджера ввода-вывода. В операционной системе Windows существует два драйвера для файловых систем FAT и NTFS, которые независимы друг от друга и управляют различными разделами диска или различными дисками [13].

Чтобы гарантировать, что драйверы устройств хорошо работают с остальной частью ОС, корпорация Microsoft определила для драйверов модель **Windows Driver Model**, которой должны соответствовать драйверы устройств. Разработчикам драйве-

ров предоставляется набор инструментов, который должен помочь в создании драйверов, удовлетворяющих требованиям этой модели [13].

Существует набор утилит, позволяющий контролировать работу программ, управляющих аппаратными устройствами. Так, утилита ***Drivers из набора средств Microsoft Windows Resource Kit*** позволяет получить детальную информацию о загруженных драйверах в текстовом формате.

Корпорацией Microsoft разработана утилита ***Bootvis***, позволяющая выявлять проблемы, возникающие в процессе загрузки операционной системы. Эта утилита выполняет трассировку всех этапов загрузки системы, в том числе этапов загрузки системного ядра, драйверов устройств и запуска процессов. Утилита не входит в стандартную поставку Windows, но ее можно загрузить из Интернета (http://download.microsoft.com:80 /download/ whistler/BTV/1.0/WXP/EN-US/BootVis-Tool.exe) [13].

В самой операционной системе Windows имеется программа «**Диспетчер устройств**», которую используют для обновления драйверов (или программного обеспечения) оборудования, изменения настроек оборудования, а также для устранения неполадок. Драйверы устройств для аппаратных продуктов с эмблемой «Для Microsoft Windows XP» или какой-либо другой более поздней версии снабжаются цифровой подписью корпорации Microsoft, которая подтверждает, что данный продукт проверен на совместимость с Windows и не изменился после проведения проверки. В окне диспетчера устройств представлено графическое отображение оборудования, установленного на компьютер. Для открытия окна диспетчера устройств нужно щелкнуть правой клавишей мыши по значку «Мой компьютер» и выбрать в контекстном меню строку «Свойства». В открывшемся окне «Свойства системы» следует перейти на вкладку «Оборудование» и нажать кнопку «Диспетчер устройств».

В окне «Диспетчера устройств» (рис. 3.5) можно, раскрывая соответствующие узлы, видеть устройства, которые либо подключены и работают, либо отключены. «Диспетчер устройств» обычно используется для проверки состояния оборудования, подключения-отключения оборудования и обновления драйве-

ров устройств, установленных на компьютере. Кроме того, возможности диагностики «Диспетчера устройств» могут использоваться опытными пользователями, обладающими глубокими знаниями о компьютерном оборудовании, для разрешения конфликтов устройств и изменения параметров ресурсов, однако при этом следует соблюдать большую осторожность [13].

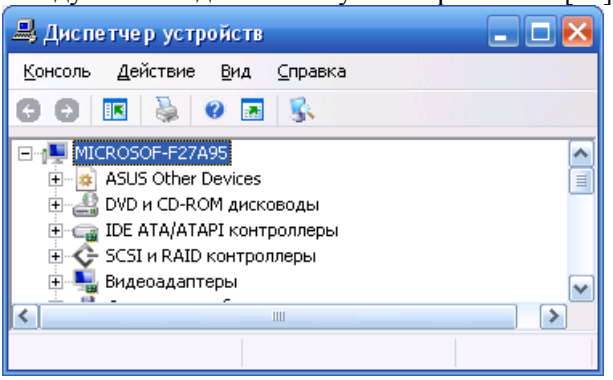

Рис. 3.5 — Окно программы «Диспетчер устройств»

При установке устройства «***Plug and Play***» Windows автоматически настраивает его, обеспечивая его правильную работу с другими установленными на компьютере устройствами. В ходе процесса настройки Windows назначает устанавливаемому устройству уникальный набор системных ресурсов. Эти ресурсы могут включать в себя один или несколько из следующих параметров:

- номера строк запросов на прерывание (IRQ);
- каналы прямого доступа к памяти (DMA);
- адреса портов ввода/вывода (1/0);
- диапазоны адресов памяти.

При установке устройств не «Plug and Play» автоматическая настройка ресурсов не производится. Некоторые типы устройств требуется настраивать вручную. Необходимые инструкции содержатся в руководстве, поставляемом вместе с устройством. Изменять параметры ресурсов вручную обычно не рекомендуется, поскольку при этом значения фиксируются, что

снижает возможности Windows по выделению ресурсов для других устройств. Если зафиксировано слишком много значений параметров для отдельных ресурсов, Windows не сможет автоматически устанавливать новые устройства «Plug and Play» [13].

Используя «Диспетчер устройств», можно отключать подсоединенные к компьютеру устройства и удалять их из конфигурации компьютера. Хотя для удаления устройства «Plug and Play» обычно достаточно его отключить или удалить из конфигурации, для удаления некоторых устройств необходимо сначала выключить компьютер.

Удаление устройств, не являющихся устройствами «Plug and Play», обычно состоит из двух шагов: отмена установки устройства с помощью диспетчера устройств и удаление устройства из конфигурации компьютера.

Не обязательно удалять устройство, которое требуется отключить, не отсоединяя от компьютера. Не отменяя установку самонастраиваемого устройства, его можно просто отключить. При отключении такого устройства оно физически остается подключенным к компьютеру, но Windows обновляет системный реестр таким образом, что драйверы отключенного устройства не загружаются при запуске компьютера. При включении устройства драйверы снова становятся доступными. Эта возможность полезна при необходимости переключения между двумя устройствами, например сетевым адаптером и модемом, или при устранении неполадок в оборудовании.

3.5.2 Диски и файловая система

Для получения доступа к просмотру состояния и управлению дисками нужно щелкнуть правой клавишей мыши по значку «Мой компьютер», выбрать строку «Управление» и щелкнуть по ней. В открывшемся окне щелкнуть по строке «Управление дисками» (рис. 3.6). В правой части окна будут отображены все дисковые устройства компьютера и основные параметры их состояния.

В окне можно управлять разделами дисковых устройств. Можно создать или удалить раздел или логический диск, можно сделать первичный раздел активным, чтобы при перезагрузке

операционной системы обращение к загрузочной записи осуществлялось с указанного раздела. Активный раздел может быть только один. Здесь же можно отформатировать диск и изменить букву или путь диска. Все эти действия вызываются щелчком правой кнопки мыши по выбранному разделу в окне, представленном на рисунке 3.6.

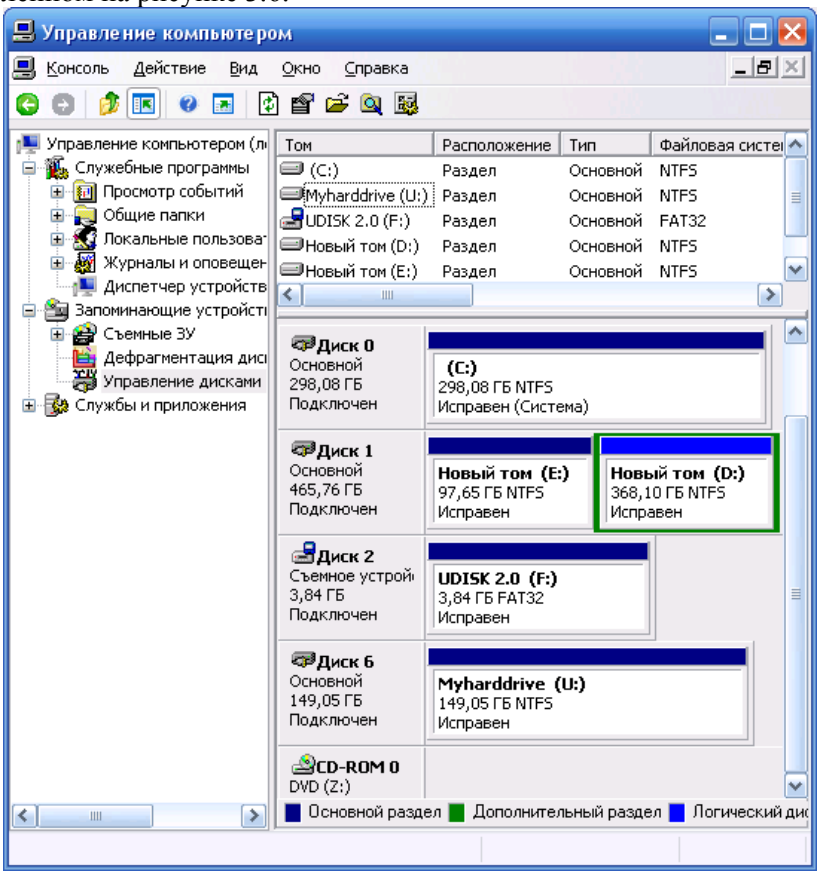

Рис. 3.6 — Вид окна «Управление компьютером» на вкладке «Управление дисками»

При работе с жестким диском всегда имеет место фрагментация. С течением времени после установки программ диск за-

полняется, а после их удаления файлы фрагментируются и операционной системе приходится искать свободные фрагменты на диске для размещения файлов. Это может привести к заметному снижению быстродействия компьютера. Негативный эффект фрагментации устраняется с помощью встроенной в Windows программы дефрагментации, запустить которую можно, указав предварительно имя диска, в левой панели оснастки «Управление компьютером» (рис. 3.7) [13].

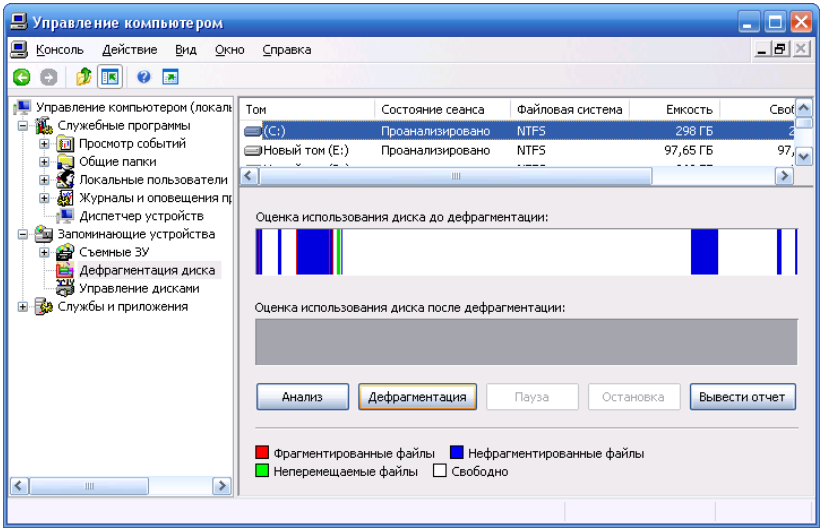

Рис. 3.7 — Вид окна «Управление компьютером» на вкладке «Дефрагментация диска»

Результаты дефрагментации можно просмотреть, нажав на кнопку «Вывести отчет», которая становится доступной после завершения дефрагментации.

3.5.3 Дисковые квоты

При совместном использовании дисковой памяти несколькими пользователями, работающими на одном компьютере, необходим контроль расходования дискового пространства. В

Windows на платформе NT эта проблема решается квотированием дискового пространства по каждому тому (независимо от количества физических дисков) и для каждого пользователя.

После установки квот дискового пространства пользователь сможет хранить на томе ограниченный объем данных, в то время как на этом томе может оставаться свободное пространство. Если пользователь превышает выданную ему квоту, в журнал событий вносится соответствующая запись. Затем, в зависимости от конфигурации системы, пользователь либо сможет записать информацию на том (более мягкий режим), либо ему будет отказано в записи.

Устанавливать и просматривать квоты на диске можно только в разделе NTFS 5.0 и при наличии необходимых полномочий (задаваемых с помощью локальных или доменных групповых политик) у пользователя, устанавливающего квоты.

Чтобы установить квоты, нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой мыши по конфигурируемому тому и выбрать в контекстном меню команду «Свойства». В появившемся окне перейти на вкладку «Квота» (рис. 3.8).

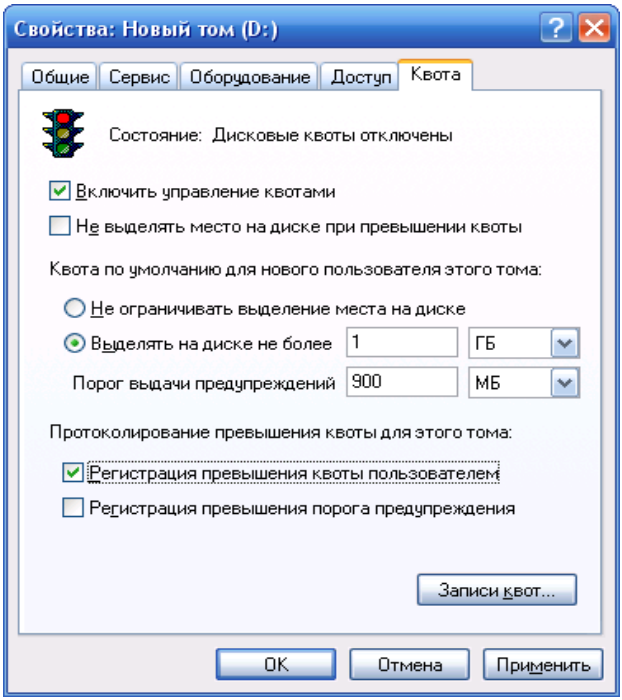

Рис. 3.8 — Вид окна по просмотру свойств диска на вкладке «Квота»

2. Установить флажок «Включить управление квотами». В этом случае будет установлен мягкий режим контроля используемого дискового пространства. Для задания жесткого режима контроля нужно установить флажок «Не выделять место на диске при превышении квоты». На этой же вкладке устанавливается размер выделяемой квоты и порог, превышение которого вызовет запись предупреждений в журнале событий.

Чтобы узнать, какие пользователи превысили выделенную им квоту (в мягком режиме), нужно нажать кнопку «Записи квот», где будет отражен список пользователей с параметрами квот и объемом используемого ими пространства диска.

3.5.4 Обеспечение надежности хранения данных на дисковых накопителях с файловой системой NTF 5.0

Устанавливая пользователям определенные **разрешения для файлов и каталогов (папок)**, администраторы системы могут защищать конфиденциальную информацию от несанкционированного доступа[6]. Каждый пользователь имеет определенный набор разрешений на доступ к конкретному объекту файловой системы (рис. 3.9). Администратор может назначить себя владельцем любого объекта файловой системы.

Действующие разрешения в отношении конкретного файла или каталога образуются из всех прямых и косвенных разрешений, назначенных пользователю для данного объекта с помощью логической функции ИЛИ.

Пользователь может назначить себя владельцем какого-либо объекта файловой системы, если у него есть необходимые права, а также передать права владельца другому пользователю.

Точки соединения (аналог монтирования в UNIX) позволяют отображать целевую папку (диск) в пустую папку, находящуюся в пространстве имен файловой системы NTFS 5.0 локального компьютера. Целевой папкой может служить любой допустимый путь Windows 2000[7] или выше. Точки соединений прозрачны для приложений, это означает, что приложение или пользователь, осуществляющий доступ к локальной папке NTFS, автоматически перенаправляется к другой папке.

---

[6] Для практического исследования данных возможностей необходимо использовать Windows Server на платформе 2000 или выше.

[7] Поддерживаются только в NTFS 5.0.

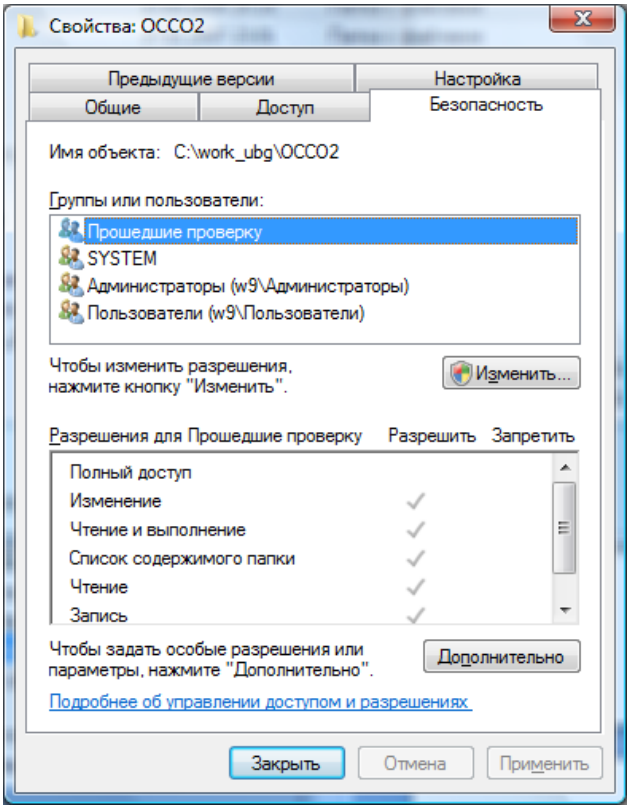

Рис. 3.9 — Вид окна установки разрешений на доступ к конкретному объекту файловой системы

Для работы с точками соединения на уровне томов можно использовать стандартные средства системы — ***утилиту Mountvol*** (рис. 3.10) и оснастку «Управление дисками». Для монтирования папок нужна утилита Linkd (из Windows 2000 Resource Kit).

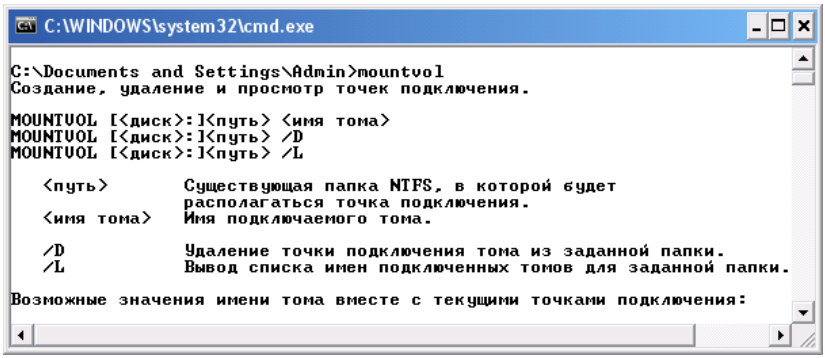

Рис. 3.10 — Вызов утилиты Mountvol

С помощью утилиты Mountvol можно выполнить следующие действия:

– отобразить корневую папку локального тома в некоторую целевую папку NTFS, т.е. подключить или монтировать том;

– вывести на экран информацию о целевой папке точки соединения NTFS, использованной при подключении тома;

– просмотреть список доступных для использования томов файловой системы;

– уничтожить точки подключения томов.

Оснастка «Управление дисками» позволяет также создать соединения для дисков компьютера.

Шифрующая файловая система EFS (Encrypting File System). Поскольку шифрование и дешифрование выполняются автоматически, пользователь может работать с файлом так же, как и до установки его криптозащиты. Все остальные пользователи, которые попытаются получить доступ к зашифрованному файлу, получат сообщение об ошибке доступа, поскольку они не владеют необходимым личным ключом, позволяющим им расшифровать файл [13].

Шифрование информации задается в окне свойств файла или папки. В окне свойств файла на вкладке «Общие» нужно нажать кнопку «Другие». Появится окно диалога «Дополнитель-

ные атрибуты». В группе «Атрибуты сжатия и шифрования» необходимо установить флажок «Шифровать содержимое для защиты данных» и нажать кнопку ОК. Далее следует нажать кнопку ОК в окне свойств зашифровываемого файла или папки. Появится окно, в котором надо указать режим шифрования [13].

При шифровании папки можно указать следующие режимы применения нового атрибута: «Только к этой папке» или «К этой папке и ко всем вложенным папкам и файлам». Для дешифрования файла или папки на вкладке «Общие» окна свойств соответствующего объекта нажать кнопку «Другие» и в открывшемся окне сбросить флажок «Шифровать содержимое для защиты данных» [13].

В процессе шифрования файлов и папок система EFS формирует специальные атрибуты (Data Decryption Field — Поле дешифрования данных), содержащие список зашифрованных ключей (FEK — File Encryption Key), что позволяет организовать доступ к файлу со стороны нескольких пользователей. Для шифрования набора FEK используется открытая часть пары ключей каждого пользователя. Информация, требуемая для дешифрования, привязывается к самому файлу. Секретная часть ключа пользователя используется при дешифровании FEK. Она хранится в безопасном месте, например на смарт-карте или устройстве высокой степени защищенности [13].

FEK применяется для создания ключей восстановления, которые хранятся в другом специальном атрибуте — DRF (Data Recovery Field — Поле восстановления данных). Сама процедура восстановления выполняется довольно редко (при уходе пользователя из организации или забывании секретной части ключа) [13].

Система EFS имеет встроенные средства восстановления зашифрованных данных в условиях, когда неизвестен личный ключ пользователя. Пользователи, которые могут восстанавливать зашифрованные данные в условиях утраты личного ключа, называются агентами восстановления данных. Они обладают сертификатом (X.509 v.3) на восстановление данных и личным ключом, с помощью которого выполняется операция восстановления зашифрованных данных [13].

Вопросы для самопроверки

1. В чем различия между блочными и символьными устройствами?
2. Как классифицируются устройства внешней памяти по методу доступа?
3. Приведите названия основных характеристик внешней памяти.
4. Какие характеристики существуют у накопителей на жестком магнитном диске?
5. Какие функции возлагаются на подсистему ввода-вывода?
6. Какими способами для персональных компьютеров могут выполняться операции ввода-вывода?
7. Что понимают под термином «драйвер»?
8. Что понимают под термином «файловая система»?
9. Опишите физическую структуру магнитного диска.
10. Опишите логическую структуру магнитного диска.
11. Расскажите об основных свойствах файловой системы FAT16.
12. Расскажите об основных свойствах файловой системы FAT32.
13. Расскажите об основных свойствах файловой системы NTFS.
14. Расскажите об основных свойствах файловой системы HPFS.
15. Расскажите об основных свойствах файловой системы UNIX.
16. Какие файловые системы существуют для CD-ROM?
17. Как устроен механизм подключения периферийных устройств в ОС Windows?
18. С помощью каких средств в ОС Windows можно управлять дисками?
19. Какие версии в ОС Windows могут поддерживать дисковые квоты?
20. Как в ОС Windows можно осуществить монтирование и шифрование дисков.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ РАБОТ

Контрольная работа № 1

Выполняется в виде электронного теста.

Контрольная работа № 2

Вариант 1

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Исследовать мультипрограммный вычислительный процесс на примере выполнения самостоятельно разработанных трех задач (например, заданий по курсу программирования).

2. Для одной из задач определить PID, загрузку ЦП, время ЦП, базовый приоритет процесса, использование памяти. Изменить приоритет процесса и установить, влияет ли это на время выполнения приложения.

3. Монопольно выполнить каждую из трех задач, определить время их выполнения.

4. Запустить одновременно (друг за другом) три задачи, определить время выполнения пакета.

Письменно ответьте на вопросы:

1. В каком случае суммарное время выполнения задач больше? При последовательном выполнении или одновременном выполнении?

2. Как изменилось время выполнения каждой отдельной задачи?

3. Как изменится время выполнения отдельной задачи при изменении ее приоритета?

4. Окажет ли влияние изменение приоритета одной задачи на время выполнения другой задачи? Объяснить результаты.

Вариант 2

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Запустить некоторое количество программ. Используя возможности оснастки «Производительность», получить диаграммы, характеризующие использование процессора при его нагрузке различным количеством потоков, меняя их активность и уровни приоритета.

2. Исследовать свои задачи (например, задания по курсу программирования). Определить характеристики процессов: % загрузки процессора (в пользовательском и привилегированном режиме), % времени прерываний, количество прерываний, базовый приоритет, обращения к диску, время выполнения процесса.

3. Исследовать свои приложения с записью результатов в Журнал счетчиков, выбрав следующие счетчики: % загруженности, работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.

4. Выполнить следующие действия:

– Запустить журнал (частота съема данных 10 сек, файл типа CVS).

– Запустить исследуемую программу.

– Через 2—3 мин остановить журнал.

– Просмотреть результаты, открыв файл журнала в Excel. Объяснить полученные результаты.

– Исследовать программу еще раз, указав тип журнала — двоичный (чтобы потом можно было просмотреть диаграммы).

5. Создать журнал трассировки для исследования своего приложения. Создать оповещения по выбранным счетчикам для своего приложения. Просмотреть журнал событий. Объяснить полученные результаты.

Вариант 3

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Используя программу Msconfig (входящую в комплект поставки Windows XP), проведите анализ, чем различаются составы загружаемых программ в различных режимах загрузки. Результаты представьте в виде таблицы сравнения.

2. Самостоятельно исследуйте интерфейс прикладного программирования операционных систем Windows и Unix. В области управления процессами (создание, завершение, приостановка, изменение приоритета и т.п.). Приведите названия функций, управляющих процессами в ОС Windows и Unix.

Письменно ответьте на вопросы:

1. Чем отличается от всех этих режимов режим с включенным протоколированием загрузки?

2. Какой из режимов содержит больше всего загружаемых программ в процессе загрузки?

3. Какой из режимов содержит меньше всего загружаемых программ в процессе загрузки?

Вариант 4

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Познакомьтесь с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виде шестнадцатеричного дампа, — например DEBUG (см. пункт 2.4.1) или PEEK.COM (во время работы доступен HELP — F1, карта памяти — F8 и информация о блоке памяти — F6).

2. Найдите в памяти таблицу таблиц (для получения ее адреса — запустите программу lol.com), познакомьтесь с ее содержимым и посмотрите указатель на 1 MCB (управляющий блок памяти). Структура таблицы таблиц и MCB приведена в пункте 2.3.2.

3. Проследите в памяти всю цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти (F8).

4. Напишите отчет о найденной цепочке блоков памяти с их адресами и размерами.

Письменно ответьте на вопросы:

1. Что содержится в последнем блоке памяти?

2. Чему равен размер основной памяти?

3. Как вычисляется физический адрес в реальном режиме работы процессора intel x86?

Вариант 5

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Познакомьтесь с работой программы DEBUG (см. пункт 2.4.1).

2. Напишите в ней программу представленную на рисунке 2.11.

3. Выполните программу с использованием трассировки (команда T) и проанализируйте выполнение каждой очередной команды.

4. Узнайте дату «прошивки» ROM BIOS, для этого поверьте дамп памяти по адресу ffff5h.

5. Время, измеряемое компьютером, формируется на основе отсчетов счетчика часов реального времени. Четыре байта этого счетчика располагаются в оперативной памяти, начиная с адреса 0046Ch. Значения счетчика времени корректируется по каждому сигналу от таймера с частотой 18,2 импульса в секунду. Определите значение счетчика. Запишите два показания счетчика примерно через одну минуту. Определите разницу между этими числами с учетом шестнадцатеричных значений. Переведите результат в десятичный вид, разделите полученное значение на величину 60*18,2 и убедитесь, что темп изменения отсчетов действительно соответствует темпу изменения реального времени.

Письменно ответьте на вопросы:

1. Как и почему меняется содержимое регистров после выполнения очередной команды?

2. Как вычисляются адреса команд?

3. Как числовая информация размещается в памяти?

Вариант 6

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Определите объем установленной физической памяти, виртуальной памяти, величину файла подкачки и его размещение на компьютере.

2. Определите, какие области физической памяти использует системная плата.

3. Проведите анализ памяти с использованием утилиты TaskList.

4. Проведите дефрагментацию жесткого диска, на который предполагается поместить файл подкачки, установите его желаемое значение и перезагрузите компьютер.

5. Оцените полученный эффект в результате оптимизации файла подкачки. Для этого используйте оснастку «Производительность», в ней добавьте счетчики «Файл подкачки \ % использования», «Файл подкачки \ % использования (пик)» и другие.

Письменно ответьте на вопросы:

1. Почему процесс настройки файла подкачки является важным для организации быстродействия персонального компьютера?

2. Значение каких параметров памяти можно получить с использованием утилиты TaskList?

3. Какие механизмы работы с памятью существуют в ОС на платформе Microsoft Windows NT?

Вариант 7

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. С помощью «Диспетчера устройств» определить, какие системные ресурсы используются портами COM (последовательный порт) и LPT (параллельный порт).

2. Просматривая параметры порта LPT, обратите внимание на включенную кнопку «Не использовать прерывание в любом случае». Просмотрите все вкладки. Выполните аналогичные действия для последовательного порта.

3. Выполните аналогичные действия для стандартного контроллера гибких дисков или других устройств хранения информации и объясните полученные данные.

Письменно ответьте на вопросы:

1. Объясните, почему для порта принтера используется канал DMA и не используется прерывание, а для последовательного порта используется прерывание и есть возможность установки скорости работы порта?

2. Объясните, почему несколько устройств используют один и тот же номер IRQ. Как операционная система их различает? Как меняется уровень приоритета по шинам IRQ? Какие устройства используют DMA? В какие области памяти производится ввод-вывод? Почему?

Вариант 8

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Изучите работу утилиты mountvol.exe и оснастку «Управление дисками».

2. Создайте две пустые папки на диске C. Выполните подключение к одной из них устройства CD-ROM с помощью утилиты mountvol.exe, а к другой диска D с помощью оснастки «Управление дисками».

3. Удалите точки соединения — первую с помощью оснастки, вторую с помощью утилиты.

4. Опишите процесс установки дисковых квот.

Письменно ответьте на вопросы:

1. К чему приводит фрагментация жестких дисков? Какими средствами можно выполнить дефрагментацию дисков.

2. Какие атрибуты могут быть у файла в файловой системе NTFS 5.0? Найти и изучить самостоятельно по любой операционной системе старше Windows 2000.

Вариант 9

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Познакомиться с основным меню DE.EXE (Norton Utilites).

2. Исследовать и описать средства работы с гибким магнитным диском при использовании FAT (кластер, логический сектор, BOOT, FAT, ROOT DIR).

3. Исследовать и описать структуру загрузочного сектора системной и обычной дискеты.

4. Исследовать и описать структуру таблицы распределения файлов и структуру корневого каталога.

5. Исследовать и описать изменения в системной области диска при создании и удалении файла и способы восстановления удалённых файлов.

6. Сформулировать принцип восстановления удаленных файлов на дискете при использовании FAT, 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.

Письменно ответьте на вопросы:

1. Назовите преимущества и недостатки NTFS и FAT. Ответ оформите в виде таблицы сравнений по характеристикам.

2. Назовите преимущества и недостатки HPFS и FAT. Ответ оформите в виде таблицы сравнений по характеристикам.

Вариант 10

Выполните практическую часть. Опишите процесс выполнения, сопровождая экранными формами.

1. Изучить процесс шифрования данных с использованием шифрующей файловой системы EFS.

2. Создать нескольких пользователей на вашем компьютере (условно пользователи А, Б и т.д.).

3. Выполнить следующие действия пользователем А: создать папку Folder1 на диске D: и в ней с помощью программы Блокнот создать три файла: File1.txt, File2.txt, File3.txt.

4. Выполнить аналогичные действия пользователем Б (предварительно перезагрузив компьютер): создать папку Folder2 на диске D: и в ней с помощью программы Блокнот создать три файла: File1.txt, File2.txt, File3.txt.

5. Установить пользователем А (предварительно перезагрузив компьютер) следующие разрешения для пользователя Б:

– полный доступ к папке Folder1, кроме чтения дополнительных атрибутов;

– чтение и выполнение для файла File1.txt;

– разрешить чтение и выполнение, но запретить запись для файла File2.txt;

– разрешить запись атрибутов, чтение разрешений, запретить запись данных и выполнение файлов для файла File3.txt.

– передать права владения файлом File1.txt пользователю Б.

6. Установить пользователем Б (предварительно перезагрузив компьютер) следующие разрешения для пользователя А:

– чтение и запись для Folder2;

– полный доступ для файла File1.txt;

– разрешить только выполнение для файла File3.txt;

– разрешить запись атрибутов, чтение разрешений, запретить запись данных и выполнение файлов для файла File3.txt.

– передать права владения файлом File3.txt пользователю А.

7. Проверить возможности доступа к созданным папкам и файлам для каждого пользователя.

8. Загрузить компьютер пользователем А, зашифровать File2.txt в папке Folder1. Сменить пользователя А на пользователя Б, попробовать прочитать File2.txt. Объяснить результат.

Письменно ответьте на вопросы:

1. Сформулируйте основные причины появления файловых систем?
2. Какими единицами операционная система выделяет файлам пространство на диске?
3. Могут ли на одном диске быть разные файловые системы?
4. Как определить величину максимального и минимального кластеров FAT-системы?

ГЛОССАРИЙ

API — Application Program Interface
DDE — Dynamic Data Exchange
DFS — Distributed File System
DMA — Direct Memory Access
DPB — Disk Parameter Blockout
DRAM — Dynamic Random Access Memory
DRF — Data Recovery Field
EFS — Encrypting File System
FAT — File Allocation Table
FCFS — First Come — First Served
FEK — File Encryption Key
FIFO — First Input — First Output
GDI — Graphics Device Interface
GDT — Global Descriptor Table
GDTR — Global Descriptor Table Register
HMA — High Memory Area
HPFS — High Performance File System
IDT — Interrupt Descriptor Table
IDTR — Interrupt Descriptor Table Register
ISA — Industry Standard Architecture
JCL — Job Control Language
LDT — Local Descriptor Table
LDTR — Local Descriptor Table Register
MCB — Memory Control Block
MBR — Master Boot Record
NSB — Non-System Bootstrap
NTFS — New Technology File System
OLE — Object Linking and Embedding
PT — Partition Table
PID — Process Identification
RAM — Random Access Memory
ROM — Read Only Memory
RPC — Remote Procedure Call
RR — Round Robin
SJN — Shortest Job Next
SMBR — Secondary Master Boot Record

SMI — System Management Interrupt
SMM — System Management Mode
SRT — Shortest Remaining Time
TR — Task Register
TSS — Task State Segment
UMA — Upper Memory Area
UMB — Upper Memory Block
VMM — Virtual Memory Manager
WOW — Windows on Windows

ОЗУ — оперативно запоминающее устройство
ОС — операционные системы
ПЗУ — постоянно запоминающее устройство

СПИСОК ЛИТЕРАТУРЫ

1. Дейтел Г. Введение в операционные системы. — М.: Мир, 1987. — Т.1. — 440 с.

2. Кейлингерт П. Элементы операционных систем. Введение для пользователей. — М.: Мир, 1985. — 295 с.

3. Гордеев А.В., Молчано А.Ю. Системное программное обеспечение. — СПб.: Питер, 2002. — 736 с.

4. Мэдник С., Донован Дж. Операционные системы. — М.: Мир, 1978. — 792 с.

5. Бэкон Д., Харрис Т. Операционные системы. — СПб.: Питер; Киев: Издательская группа BHV, 2004. — 800 с.

6. Назаров С.В. Операционные среды, системы и оболочки. Основы структурной и функциональной организации: Учеб. пособие. — М.: КУДИЦ-ПРЕСС, 2007.—504 с.: ил.

7. IA-32 Intel Architecture Software. Developer's Manual, Volume 1—4.

8. Гордеев А.В., Штепен В.А. Управление процессами в операционных системах реального времени: Учеб. пособие. — Л.: ЛИАП, 1988. — 76 с.

9. Робачевский А.М. Операционная система UNIX. — СПб.: BHV, 1997. — 528 с.

10. Олифер Н.А., Олифер В.Г. Сетевые операционные системы. — СПб.: Питер, 2001. — 538 с.

11. Столингс В. Операционные системы: Пер. с англ. — 4-е изд. — М.: Издательский дом «Вильямс», 2002. — 848 с.

12. Таненбаум Э. Современные операционные системы: Пер. с англ. — 2-е изд. — СПб.: Питер, 2002. — 1040 с.

13. Назаров С.В., Гудыно Л.П., Кириченко А.А. Операционные системы: Практикум / Под ред. С.В. Назарова — М.: КУДИЦ-ПРЕСС, 2008. — 464 с.

14. Assembler / В. Юров. — СПб., 2001. — 624 с.

15. Гук М.. Аппаратные средства IBM PC: Энциклопедия. — 2-е изд. — СПб.: Питер, 2002. — 928 с.