Глава 3. Управление памятью в операционных системах

Оперативная память — это важнейший ресурс любой вычислительной системы, поскольку без нее (как, впрочем, и без центрального процессора) невозможно выполнение ни одной программы. В главе 1 мы уже отмечали, что память является разделяемым ресурсом. От выбранных механизмов распределения памяти между выполняющимися процессорами в значительной степени зависит эффективность использования ресурсов системы, ее производительность, а также возможности, которыми могут пользоваться программисты при создании своих программ. Желательно так распределять память, чтобы выполняющаяся задача имела возможность обратиться по любому адресу в пределах адресного пространства той программы, в которой идут вычисления. С другой стороны, поскольку любой процесс имеет потребности в операциях ввода-вывода, и процессор достаточно часто переключается с одной задачи на другую, желательно в оперативной памяти расположить достаточное количество активных задач с тем, чтобы процессор не останавливал вычисления из-за отсутствия очередной команды или операнда. Некоторые ресурсы, которые относятся к неразделяемым, из-за невозможности их совместного использования делают *виртуальными*. Таким образом, чтобы иметь возможность выполняться, каждый процесс может получить некий виртуальный ресурс. Виртуализация ресурсов делается программным способом средствами операционной системы, а значит, для них тоже нужно иметь ресурс памяти. Поэтому вопросы организации разделения памяти для выполняющихся процессов и потоков являются очень актуальными, ибо выбранные и реализованные алгоритмы решения этих вопросов в значительной степени определяют и потенциальные возможности системы, и общую ее производительность, и эффективность использования имеющихся ресурсов.

Память и отображения, виртуальное адресное пространство

Если не принимать во внимание программирование на машинном языке (эта технология практически не используется уже очень давно), то можно сказать, что программист обращается к памяти с помощью некоторого набора логических имен, которые чаще всего являются символьными, а не числовыми, и для которого отсутствует отношение порядка. Другими словами, в общем случае множество переменных в программе не упорядочено, хотя отдельные переменные могут иметь частичную упорядоченность (например, элементы массива). Имена переменных и входных точек программных модулей составляют пространство символьных имен. Иногда это адресное пространство называют *логическим*.

С другой стороны, при выполнении программы мы имеем дело с физической оперативной памятью, собственно с которой и работает процессор, извлекая из нее команды и данные и помещая в нее результаты вычислений. *Физическая память* представляет собой упорядоченное множество ячеек реально существующей оперативной памяти, и все они пронумерованы, то есть к каждой из них можно обратиться, указав ее порядковый номер (адрес). Количество ячеек физической памяти ограниченно и фиксированно.

Системное программное обеспечение должно связать каждое указанное пользователем символьное имя с физической ячейкой памяти, то есть осуществить отображение пространства имен на физическую память компьютера. В общем случае это отображение осуществляется в два этапа (рис. 3. 1): сначала системой программирования, а затем операционной системой. Это второе отображение осуществляется с помощью соответствующих аппаратных средств процессора — подсистемы управления памятью, которая использует дополнительную информацию, подготавливаемую и обрабатываемую операционной системой. Между этими этапами обращения к памяти имеют форму *виртуального* адреса. При этом можно сказать, что множество всех допустимых значений виртуального адреса для некоторой программы определяет ее *виртуальное адресное пространство,* или *виртуальную память.* Виртуальное адресное пространство программы зависит, прежде всего, от архитектуры процессора и от системы программирования и практически не зависит от объема реальной физической памяти компьютера. Можно еще сказать, что адреса команд и переменных в машинной программе, подготовленной к выполнению системой программирования, как раз и являются виртуальными адресами.

Как мы знаем, система программирования осуществляет трансляцию и компоновку программы, используя библиотечные программные модули. В результате работы системы программирования полученные виртуальные адреса могут иметь как двоичную форму, так и символьно-двоичную. Это означает, что некоторые программные модули (их, как правило, большинство) и их переменные получают ка-

кие-то числовые значения, а те модули, адреса для которых пока не могут быть определены, имеют по-прежнему символьную форму, и их окончательная привязка к физическим ячейкам будет осуществлена на этапе загрузки программы в память перед ее непосредственным выполнением.

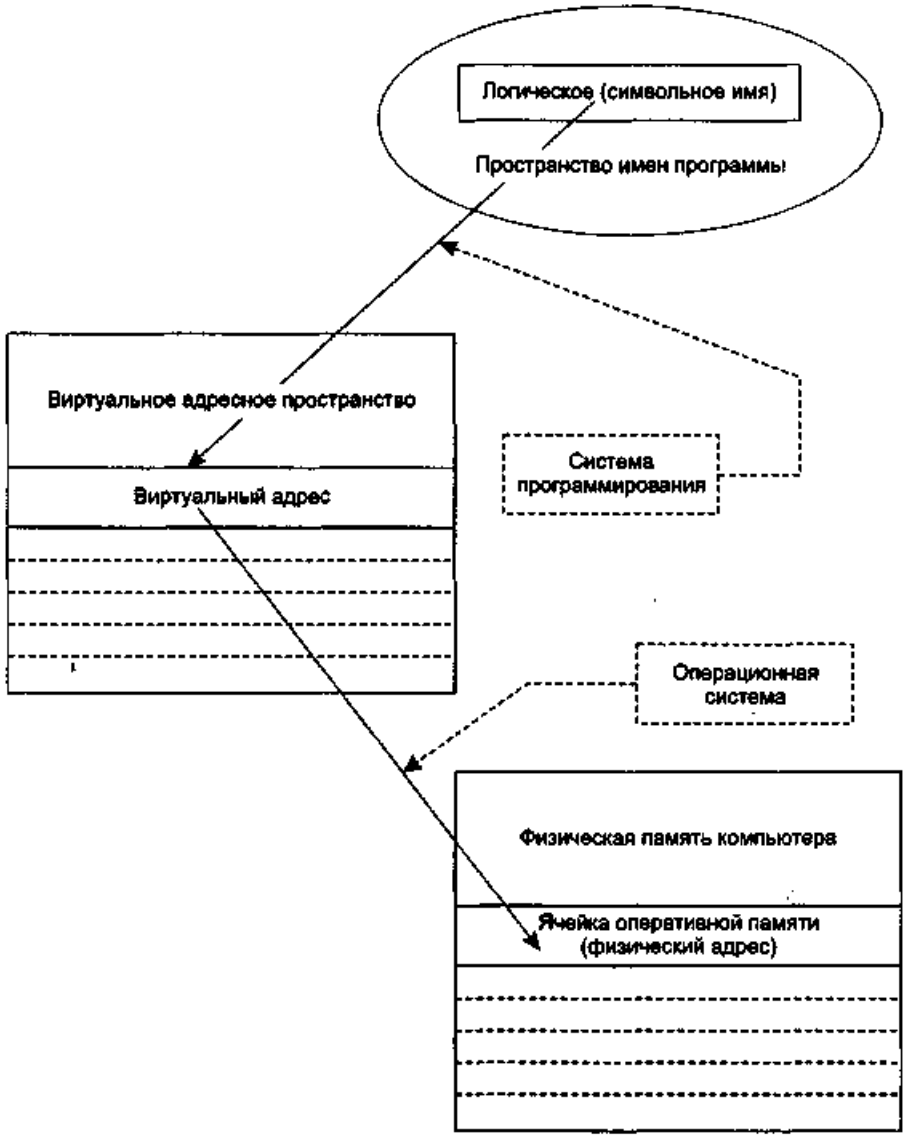

Рис. 3. 1. Память и отображения

Одним из частных случаев отображения пространства символьных имен на физическую память является полная тождественность виртуального адресного пространства физической памяти. При этом нет необходимости осуществлять второе ото-

бражение. В таком случае говорят, что система программирования генерирует *абсолютную двоичную программу;* в этой программе все двоичные адреса таковы, что программа может исполняться только тогда, когда ее виртуальные адреса будут точно соответствовать физическим. Часть программных модулей любой операционной системы обязательно должна быть абсолютными двоичными программами. Эти программы размещаются по фиксированным адресам физической памяти, и с их помощью уже можно впоследствии реализовывать размещение остальных программ, подготовленных системой программирования таким образом, что они могут работать на различных физических адресах (то есть на тех адресах, на которые их разместит операционная система). В качестве примера таких программ можно назвать программы загрузки операционной системы.

Другим частным случаем этой общей схемы трансляции адресного пространства является тождественность виртуального адресного пространства исходному логическому пространству имен. Здесь уже отображение выполняется самой операционной системой, которая во время исполнения использует таблицу символьных имен. Такая схема отображения используется чрезвычайно редко, так как отображение имен на адреса необходимо выполнять для каждого вхождения имени (каждого нового имени), и особенно много времени тратится на квалификацию имен. Данную схему можно было встретить в интерпретаторах, в которых стадии трансляции и исполнения практически неразличимы. Это характерно для простейших компьютерных систем, в которых вместо операционной системы использовался встроенный интерпретатор (например, Basic).

Возможны и промежуточные варианты. В простейшем случае транслятор-компилятор генерирует относительные адреса, которые, по сути, являются виртуальными адресами, с последующей настройкой программы на один из непрерывных разделов. Второе отображение осуществляется перемещающим загрузчиком. После загрузки программы виртуальный адрес теряется, и доступ выполняется непосредственно к физическим ячейкам. Более эффективное решение достигается в том случае, когда транслятор вырабатывает в качестве виртуального адреса относительный адрес и информацию о начальном адресе, а процессор, используя подготавливаемую операционной системой адресную информацию, выполняет второе отображение не один раз (при загрузке программы), а при каждом обращении к памяти.

Термин *виртуальная память* фактически относится к системам, которые сохраняют виртуальные адреса во время исполнения. Так как второе отображение осуществляется в процессе исполнения задачи, то адреса физических ячеек могут изменяться. При правильном применении такие изменения улучшают использование памяти, избавляя программиста от деталей управления ею, и даже повышают надежность вычислений.

Если рассматривать общую схему двухэтапного отображения адресов, то с позиции соотношения объемов упомянутых адресных пространств можно отметить наличие следующих трех ситуаций:

* объем виртуального адресного пространства программы V_v меньше объема физической памяти V_p ($V_v < V_p$);
* объем виртуального адресного пространства программы V_v равен объему физической памяти V_p ($V_v = V_p$);

* объем виртуального адресного пространства программы V_v больше объема физической памяти V_p ($V_v > V_p$).

Первая ситуация ($V_v < V_p$) ныне практически не встречается, но, тем не менее, это реальное соотношение. Скажем, не так давно 16-разрядные мини-ЭВМ имели систему команд, в которых пользователи-программисты могли адресовать до $2^{16} = 64$ Кбайт адресов (обычно в качестве адресуемой единицы выступала ячейка памяти размером с байт). А физически старшие модели этих мини-ЭВМ могли иметь объем оперативной памяти в несколько мегабайтов. Обращение к памяти столь большого объема осуществлялось с помощью специальных регистров, содержимое которых складывалось с адресом операнда (или команды), извлекаемым из поля операнда или указателя команды (и/или определяемым по значению поля операнда или указателя команды). Соответствующие значения в эти специальные регистры, выступающие как базовое смещение в памяти, заносила операционная система. Для одной задачи в регистр заносилось одно значение, а для второй (третьей, четвертой и т. д.) задачи, размещаемой одновременно с первой, но в другой области памяти, заносилось, соответственно, другое значение. Вся физическая память таким образом разбивалась на разделы объемом по 64 Кбайт, и на каждый такой раздел осуществлялось отображение своего виртуального адресного пространства.

Вторая ситуация ($V_v = V_p$) встречается очень часто, особенно характерна она была для недорогих вычислительных комплексов. Для этого случая имеется большое количество методов распределения оперативной памяти.

Наконец, в наше время мы уже достигли того, что ситуация превышения объема виртуального адресного пространства программы над объемом физической памяти ($V_v > V_p$) характерна даже для персональных компьютеров, то есть для самых распространенных и недорогих машин. Теперь это самая обычная ситуация, и для нее имеется несколько методов распределения памяти, отличающихся как сложностью, так и эффективностью.

Простое непрерывное распределение и распределение с перекрытием

Общие принципы управления памятью в однопрограммных операционных системах

Простое непрерывное распределение — это самая простая схема, согласно которой вся память условно может быть разделена на три области:

* область, занимаемая операционной системой;
* область, в которой размещается исполняемая задача;
* незанятая ничем (свободная) область памяти.

Изначально являясь самой первой схемой, схема простого непрерывного распределения памяти продолжает и сегодня быть достаточно распространенной. Эта схема предполагает, что операционная система не поддерживает мультипрограммирование, поэтому не возникает проблемы распределения памяти между несколькими

задачами. Программные модули, необходимые для всех программ, располагаются в области самой операционной системы, а вся оставшаяся память может быть предоставлена задаче. Эта область памяти получается непрерывной, что облегчает работу системы программирования. Поскольку в различных однотипных вычислительных комплексах может быть разный состав внешних устройств (и, соответственно, они содержат различное количество драйверов), для системных нужд могут быть отведены отличающиеся объемы оперативной памяти, и получается, что можно не привязывать жестко виртуальные адреса программы к физическому адресному пространству. Эта привязка осуществляется на этапе загрузки задачи в память.

Для того чтобы для задач отвести как можно больший объем памяти, операционная система строится таким образом, чтобы постоянно в оперативной памяти располагалась только самая нужная ее часть. Эту часть операционной системы стали называть *ядром*. Прежде всего, в ядро операционной системы входят основные модули супервизора. Для однопрограммных систем понятие супервизора вырождается в модули, получающие и выполняющие первичную обработку запросов от обрабатывающих и прикладных программ, и в модули подсистемы памяти. Ведь если программа по ходу своего выполнения запрашивает некоторое множество ячеек памяти, то подсистема памяти должна их выделить (если они есть), а после освобождения этой памяти подсистема памяти должна выполнить действия, связанные с возвратом памяти в систему. Остальные модули операционной системы, не относящиеся к ее ядру, могут быть обычными диск-резидентными (или транзитными), то есть загружаться в оперативную память только по необходимости, и после своего выполнения вновь освобождать память.

Такая схема распределения влечет за собой два вида потерь вычислительных ресурсов — потеря процессорного времени, потому что процессор простаивает, пока задача ожидает завершения операций ввода-вывода, и потеря самой оперативной памяти, потому что далеко не каждая программа использует всю память, а режим работы в этом случае однопрограммный. Однако это очень недорогая реализация, которая позволяет отказаться от многих функций операционной системы. В частности, такая сложная проблема, как защита памяти, здесь почти не стоит. Единственное, что желательно защищать — это программные модули и области памяти самой операционной системы.

Если есть необходимость создать программу, логическое адресное пространство которой должно быть больше, чем свободная область памяти, или даже больше, чем весь возможный объем оперативной памяти, то используется распределение с перекрытием — так называемые *оверлейные структуры* (от *overlay* — перекрытие, расположение поверх чего-то). Этот метод распределения предполагает, что вся программа может быть разбита на части — сегменты. Каждая оверлейная программа имеет одну главную (main) часть и несколько сегментов (segments), причем в памяти машины одновременно могут находиться только ее главная часть и один или несколько не перекрывающихся сегментов.

Пока в оперативной памяти располагаются выполняющиеся сегменты, остальные находятся во внешней памяти. После того как текущий (выполняющийся) сегмент завершит свое выполнение, возможны два варианта: либо он сам (если данный сег-

мент не нужно сохранить во внешней памяти в его текущем состоянии) обращается к операционной системе с указанием, какой сегмент должен быть загружен в память следующим; либо он возвращает управление главному сегменту задачи, и уже тот обращается к операционной системе с указанием, какой сегмент сохранить (если это нужно), а какой сегмент загрузить в оперативную память, и вновь отдает управление одному из сегментов, располагающихся в памяти. Простейшие схемы сегментирования предполагают, что в памяти в каждый конкретный момент времени может располагаться только один сегмент (вместе с главным модулем). Более сложные схемы, используемые в больших вычислительных системах, позволяют располагать в памяти несколько сегментов. В некоторых вычислительных комплексах могли существовать отдельно сегменты кода и сегменты данных. Сегменты кода, как правило, не претерпевают изменений в процессе своего исполнения, поэтому при загрузке нового сегмента кода на место отработавшего последний можно не сохранять во внешней памяти, в отличие от сегментов данных, которые сохранять необходимо.

Первоначально программисты сами должны были включать в тексты своих программ соответствующие обращения к операционной системе (их называют системными вызовами) и тщательно планировать, какие сегменты могут находиться в оперативной памяти одновременно, чтобы их адресные пространства не пересекались. Однако с некоторых пор такого рода обращения к операционной системе системы программирования стали подставлять в код программы сами, автоматически, если в том возникает необходимость. Так, в известной и популярной в недалеком прошлом системе программирования Turbo Pascal программист просто указывал, что данный модуль является оверлейным. И при обращении к нему из основной программы модуль загружался в память и получал управление. Все адреса определялись системой программирования автоматически, обращения к DOS для загрузки оверлеев тоже генерировались системой Turbo Pascal.

Распределение оперативной памяти в MS DOS

Как известно, MS DOS[1] — это однопрограммная операционная система для персонального компьютера типа IBM PC. В ней, конечно, можно организовать запуск резидентных, или TSR-задач[2], в результате которого в памяти будет находиться не одна программа, но в целом система MS DOS предназначена для выполнения только одного вычислительного процесса. Поэтому распределение памяти в ней построено по схеме простого непрерывного распределения. Система поддерживает механизм распределения памяти с перекрытием (оверлейные структуры).

Как известно, в IBM PC использовался 16-разрядный микропроцессор i8088, который за счет введения сегментного способа адресации позволял указывать

[1] Версий однопрограммных дисковых операционных систем (Disks Operating System, DOS) для персональных компьютеров было много. Одних только MS DOS (систем от Microsoft) более 10. Однако несмотря на существенные различия все их чаще всего именуют одинаково — MS DOS.

[2] TSR (Terminate and Stay Resident) — резидентная в памяти программа, которая благодаря изменениям в таблице векторов прерываний позволяет перехватывать прерывания и в случае обращения к ней выполнять необходимые действия. Подробно об этом можно прочесть, например, в [3, 23, 24, 35].

адрес ячейки памяти в пространстве объемом до 1 Мбайт. В последующих персональных компьютерах (IBM PC AT, AT386 и др.) было принято решение поддерживать совместимость с первыми, поэтому при работе в DOS прежде всего рассматривают первый мегабайт. Вся эта память разделялась на несколько областей, что иллюстрирует рис. 3.2. На этом рисунке показано, что памяти может быть и больше, чем 1 Мбайт, но более подробное рассмотрение этого вопроса мы здесь опустим, отослав желающих изучить данную тему глубже к монографии [2].

Если не вдаваться в детали, можно сказать, что в состав MS DOS входят следующие основные компоненты.

* Подсистема BIOS (Base Input-Output System — базовая подсистема ввода-вывода), включающая в себя помимо программы POST (Power On Self Test — самотестирование при включении компьютера)[1] программные модули обработки прерываний, с помощью которых можно управлять основными контроллерами на материнской плате компьютера и устройствами ввода-вывода. Эти модули часто называют обработчиками прерываний. По своей функциональной сути они представляют собой драйверы. BIOS располагается в постоянном запоминающем устройстве компьютера. В конечном итоге почти все остальные модули MS DOS обращаются к BIOS. Если и не напрямую, то через модули более высокого уровня иерархии.
* Модуль расширения BIOS — файл IO.SYS (в других DOS-системах он может называться иначе, например _BIO.COM).
* Основной, или базовый, модуль обработки прерываний DOS — файл MSDOS.SYS. Именно этот модуль в основном реализует работу с файловой системой.
* Командный процессор (интерпретатор команд) — файл COMMAND.COM.
* Утилиты и драйверы, расширяющие возможности системы.
* Программа загрузки MS DOS — загрузочная запись (Boot Record, BR), расположенная на дискете или на жестком диске (подробнее о загрузочной записи и о других загрузчиках см. главу 6).

Вся память в соответствии с архитектурой IBM PC условно может быть разбита на следующие три части.

* В самых младших адресах памяти (первые 1024 ячейки) размещается таблица векторов прерывания (см. раздел «Система прерываний 32-разрядных микропроцессоров i80x86» в главе 4). Это связано с аппаратной реализацией процессора i8088. В последующих процессорах (начиная с i80286) адрес таблицы прерываний определяется через содержимое соответствующего регистра, но для обеспечения полной совместимости с первым процессором при включении или аппаратном сбросе в этот регистр заносятся нули. При желании, однако, в случае использования современных микропроцессоров i80x86 вектора прерываний можно размещать и в других областях.

[1] После выполнения программы POST, входящей в состав ROM BIOS, опрашиваются устройства, которые могут содержать программы для загрузки операционной системы.

0000-003FF	1 Кбайт	Таблица векторов прерываний	
00400-005FF	512 байт	Глобальные переменные BIOS; глобальные переменные DOS	В ранних версиях здесь располагались глобальные переменные интерпретатора Бейсик
00600-0A000	35-60 Кбайт	Модуль IO. SYS; Модуль MSDOS. SYS: - обслуживающие функции; - буферы, рабочие и управляющие области; - устанавливаемые драйверы; Резидентная часть COMMAND. COM: - обработка программных прерываний; - системная программа загрузки; - программа загрузки транзитной части COMMAND. COM	Размер этой области зависит от версии MSDOS и, главное, от конфигурационного файла CONFIG. SYS
	580 Кбайт	Область памяти для выполнения программ пользователя и утилит MS DOS. В эту область попадают программы типа *. COM и *. EXE	Объем этой области очень зависит от объема, занимаемого ядром ОС. Программа может перекрывать транзитную область COMMAND. COM
		Область расположения стека исполняющейся программы	Стек «растет» снизу вверх
	18 Кбайт	Транзитная часть командного процессора COMMAND. COM	Собственно командный интерпретатор
A0000-C7FFF	160 Кбайт	Видеопамять. Область и размер используемого видеобуфера зависят от текущего режима	При работе в текстовом режиме область памяти A0000-B0000 свободна и может быть использована в программе
C8000-EOOOO	96 Кбайт	Зарезервировано для расширения BIOS	
F0000-FFFF	64 Кбайт	Область ROM BIOS (System BIOS)	Обычно объем этой области равен 32 Кбайт, но может достигать и 128 Кбайт, занимая младшие адреса
Более 100000		High Memory Area. При наличии драйвера HIMEM. SYS здесь можно расположить основные системные файлы MS DOS, освобождая тем самым область основной памяти в первом мегабайте	Может использоваться при наличии специальных драйверов. Используются спецификации XMS и EMS

Рис. 3.2. Распределение оперативной памяти в MS DOS

* Вторая часть памяти отводится для программных модулей самой системы MS DOS и для программ пользователя. Эту область памяти мы рассмотрим чуть

позже, здесь только заметим, что она называется основной, или стандартной, памятью (conventional memory).

* Наконец, третья часть адресного пространства отведена для постоянных запоминающих устройств и функционирования некоторых устройств ввода-вывода. Эта область памяти получила название UMA (Upper Memory Area — область памяти, адрес которой выше основной).

В младших адресах основной памяти размещается то, что можно условно назвать ядром этой операционной системы — системные переменные, основные программные модули, блоки данных для буферизации операций ввода-вывода. Для управления устройствами, драйверы которых не входят в базовую подсистему ввода-вывода, загружаются так называемые *загружаемые,* или *устанавливаемые,* драйверы. Перечень устанавливаемых драйверов определяется специальным конфигурационным файлом CONFIG.SYS. После загрузки расширения BIOS — файла IO.SYS — последний (загрузив модуль MSDOS.SYS) считывает файл CONFIG.SYS и уже в соответствии с ним подгружает в память необходимые драйверы. Кстати, в конфигурационном файле CONFIG.SYS могут иметься операторы, указывающие на количество буферов, отводимых для ускорения операций ввода-вывода, и на количество файлов, которые могут обрабатываться (для работы с файлами необходимо зарезервировать место в памяти для хранения управляющих структур, с помощью которых выполняются операции с записями файла). В случае использования микропроцессоров i80x86 и наличия в памяти драйвера HIMEM.SYS модули IO.SYS и MSDOS.SYS могут быть размещены за пределами первого мегабайта в области, которая получила название *HMA* (High Memory Area — область памяти с большими адресами).

Память с адресами, большими чем 10FFFFh, может быть использована в DOS-программах при выполнении их на микропроцессорах, имеющих такую возможность (например, микропроцессор i80286 имел 24-разрядную шину адреса, а i80386 — уже 32-разрядную). Но для этого с помощью специальных драйверов необходимо переключать процессор в другой режим работы, при котором он сможет использовать адреса выше 10FFFFh. Широкое распространение получили две основные спецификации: *XMS* (Extended Memory Specification) и *EMS* (Expanded Memory Specification). Последние годы система MS DOS практически перестала применяться. Теперь ее используют в основном для запуска некоторых утилит, с помощью которых подготавливают дисковые устройства, или для установки других операционных систем. И поскольку основным утилитам, необходимым для обслуживания персонального компьютера, спецификации EMS и XMS, как правило, не нужны, мы не будем здесь их рассматривать.

Остальные программные модули MS DOS (в принципе, большинство из них является утилитами) оформлены как обычные исполняемые файлы. Например, утилита форматирования диска представляет собой и двоичный исполняемый файл, и команду операционной системы. В основном такого рода утилиты являются транзитными модулями, то есть загружаются в память только на время своей работы, хотя среди них имеются и TSR-программы.

Для того чтобы предоставить больше памяти программам пользователя, в MS DOS применено то же решение, что и во многих других простейших операционных

системах, — командный процессор COMMAND.COM состоит из двух частей. Первая часть является резидентной и размещается в области ядра, вторая часть транзитная и размещается в области старших адресов раздела памяти, выделяемой для программ пользователя. И если программа пользователя перекрывает собой область, в которой была расположена транзитная часть командного процессора, то последний при необходимости восстанавливает в памяти свою транзитную часть, поскольку после выполнения программы она возвращает управление резидентной части COMMAND.COM.

Поскольку размер основной памяти относительно небольшой, то очень часто системы программирования реализуют оверлейные структуры. Для этого в MS DOS поддерживаются специальные вызовы.

Распределение памяти статическими и динамическими разделами

Для организации мультипрограммного и/или мультизадачного режима необходимо обеспечить одновременное расположение в оперативной памяти нескольких задач (целиком или частями). Память задаче может выделяться одним сплошным участком (в этом случае говорят о методах *неразрывного распределения памяти)* или несколькими порциями, которые могут быть размещены в разных областях памяти (тогда говорят о методах *разрывного распределения)*.

Начнем с методов неразрывного распределения памяти. Самая простая схема распределения памяти между несколькими задачами предполагает, что память, не занятая ядром операционной системы, может быть разбита на несколько непрерывных частей — *разделов* (partitions, regions). Разделы характеризуются именем, типом, границами (как правило, указываются начало раздела и его длина).

Разбиение памяти на несколько непрерывных (неразрывных) разделов может быть фиксированным (статическим) либо динамическим (то есть процесс выделения нового раздела памяти происходит непосредственно при появлении новой задачи). Вначале мы кратко рассмотрим статическое распределение памяти на разделы.

Разделы с фиксированными границами

Разбиение всего объема оперативной памяти на несколько разделов может осуществляться единовременно (то есть в процессе генерации варианта операционной системы, который потом и эксплуатируется) или по мере необходимости оператором системы. Однако и во втором случае при разбиении памяти на разделы вычислительная система более ни для каких целей в этот момент не используется. Пример разбиения памяти на несколько разделов приведен на рис. 3.3.

В каждом разделе в каждый момент времени может располагаться по одной программе (задаче). В этом случае по отношению к каждому разделу можно применить все те методы создания программ, которые используются для однопрограммных систем. Возможно использование оверлейных структур, что позволяет создавать большие сложные программы и в то же время поддерживать *коэффициент мульти-*

программирования[1] на должном уровне. Первые мультипрограммные операционные системы строились по этой схеме. Использовалась эта схема и много лет спустя при создании недорогих вычислительных систем, поскольку является несложной и обеспечивает возможность параллельного выполнения программ. Иногда в некотором разделе размещалось по нескольку небольших программ, которые постоянно в нем и находились. Такие программы назывались *ОЗУ-резидентными* (или просто *резидентными)*. Та же схема используется и в современных встроенных системах; правда, для них характерно, что все программы являются резидентными, и внешняя память во время работы вычислительного оборудования не используется.

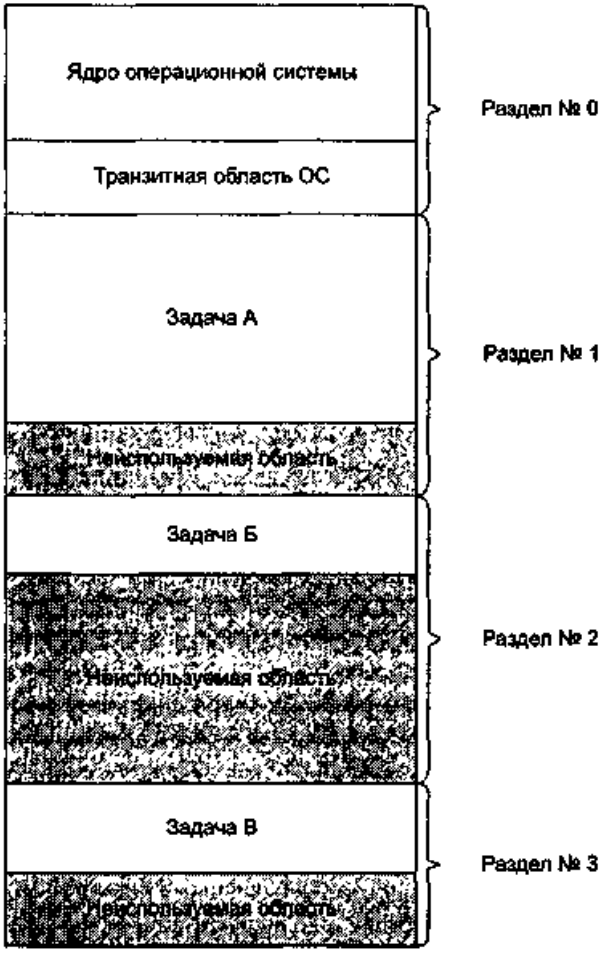

Рис. 3.3. Распределение памяти разделами с фиксированными границами

Под коэффициентом мультипрограммирования (т) понимают количество параллельно выполняемых программ. Обычно на практике для загрузки центрального процессора до уровня 90 % необходимо, чтобы коэффициент мультипрограммирования был не менее 4-5. А для того чтобы наиболее полно использовать и остальные ресурсы системы, желательно иметь m на уровне 10-15.

При небольшом объеме памяти и, следовательно, небольшом количестве разделов увеличить число параллельно выполняемых приложений (особенно когда эти приложения интерактивны и во время своей работы фактически не используют процессорное время, а в основном ожидают операций ввода-вывода) можно за счет *замены их в памяти,* или *свопинга* (swapping). При свопинге задача может быть целиком выгружена на магнитный диск (перемещена во внешнюю память), а на ее место загружается либо более привилегированная, либо просто готовая к выполнению другая задача, находившаяся на диске в приостановленном состоянии. При свопинге из основной памяти во внешнюю (обратно) перемещается вся программа, а не ее отдельная часть.

Серьезная проблема, которая возникает при организации мультипрограммного режима работы вычислительной системы, — защита как самой операционной системы от ошибок и преднамеренного вмешательства процессов в ее работу, так и самих процессов друг от друга.

В самом деле, программа может обращаться к любым ячейкам в пределах своего виртуального адресного пространства. Если система отображения памяти не содержит ошибок, и в самой программе их тоже нет, то возникать ошибок при выполнении программы не должно. Однако в случае ошибок адресации, что случается не так уж и редко, исполняющаяся программа может начать «обработку» чужих данных или кодов с непредсказуемыми последствиями. Одной из простейших, но достаточно эффективных мер является введение регистров защиты памяти. В эти регистры операционная система заносит граничные значения области памяти раздела текущего исполняющегося процесса. При нарушении адресации возникает прерывание, и управление передается супервизору операционной системы. Обращения задач к операционной системе за необходимыми сервисами осуществляются не напрямую, а через команды программных прерываний, что обеспечивает передачу управления только в предопределенные входные точки кода операционной системы и в системном режиме работы процессора, при котором регистры защиты памяти игнорируются. Таким образом, выполнение функции защиты требует введения специальных аппаратных механизмов, используемых операционной системой.

Основным недостатком рассматриваемого способа распределения памяти является наличие порой достаточно большого объема неиспользуемой памяти (см. рис. 3.3). Неиспользуемая память может быть в каждом из разделов. Поскольку разделов несколько, то и неиспользуемых областей получается несколько, поэтому такие потери стали называть *фрагментацией памяти.* В отдельных разделах потери памяти могут быть очень значительными, однако использовать фрагменты свободной памяти при таком способе распределения не представляется возможным. Желание разработчиков сократить столь значительные потери привело их к следующим двум решениям:

* выделять раздел ровно такого объема, который нужен под текущую задачу;
* размещать задачу не в одной непрерывной области памяти, а в нескольких областях.

Второе решение было реализовано в нескольких способах организации виртуальной памяти. Мы их обсудим в следующем разделе, а сейчас кратко рассмотрим первое решение.

Разделы с подвижными границами

Чтобы избавиться от фрагментации, можно попробовать размещать в оперативной памяти задачи плотно, одну за другой, выделяя ровно столько памяти, сколько задача требует. Одной из первых операционных систем, в которой был реализован такой способ распределения памяти, была OS MVT[1] (Multiprogramming with a Variable number of Tasks — мультипрограммирование с переменным числом задач). В этой операционной системе специальный планировщик (диспетчер памяти) ведет список адресов свободной оперативной памяти. При появлении новой задачи диспетчер памяти просматривает этот список и выделяет для задачи раздел, объем которой либо равен необходимому, либо чуть больше, если память выделяется не ячейками, а некими дискретными единицами. При этом модифицируется список свободных областей памяти. При освобождении раздела диспетчер памяти пытается объединить освобождающийся раздел с одним из свободных участков, если таковой является смежным.

При этом список свободных участков памяти может быть упорядочен либо по адресам, либо по объему. Выделение памяти под новый раздел может осуществляться одним из трех основных способов:

* первый подходящий участок;
* самый подходящий участок;
* самый неподходящий участок.

В первом случае список свободных областей упорядочивается по адресам (например, по возрастанию адресов). Диспетчер просматривает список и выделяет задаче раздел в той области, которая первой подойдет по объему. В этом случае, если такой фрагмент имеется, то в среднем необходимо просмотреть половину списка. При освобождении раздела также необходимо просмотреть половину списка. Правило «первый подходящий» приводит к тому, что память для небольших задач преимущественно будет выделяться в области младших адресов, и, следовательно, это увеличит вероятность того, что в области старших адресов будут образовываться фрагменты достаточно большого объема.

Способ «самый подходящий» предполагает, что список свободных областей упорядочен по возрастанию объема фрагментов. В этом случае при просмотре списка для нового раздела будет использован фрагмент свободной памяти, объем которой наиболее точно соответствует требуемому. Требуемый раздел будет определяться по-прежнему в результате просмотра в среднем половины списка. Однако оставшийся фрагмент оказывается настолько малым, что в нем уже вряд ли удастся разместить еще какой-либо раздел. При этом получается, что вновь образованный фрагмент попадет в начало списка, и в последующем его придется каждый раз проверять на пригодность, тогда как его малый размер вряд ли окажется подходящим. Поэтому в целом такую дисциплину нельзя назвать эффективной.

Как ни странно, самым эффективным способом, как правило, является последний, по которому для нового раздела выделяется «самый неподходящий» фрагмент сво-

[1] Эта операционная система была одной из самых распространенных в больших ЭВМ класса IBM 360 (370).

бодной памяти. Для этой дисциплины, список свободных областей упорядочивается по убыванию объема свободного фрагмента. Очевидно, что если есть такой фрагмент памяти, то он сразу же и будет найден, и, поскольку этот фрагмент является самым большим, то, скорее всего, после выделения из него раздела памяти для задачи оставшуюся область памяти можно будет использовать в дальнейшем.

Однако очевидно, что при любой дисциплине обслуживания, по которой работает диспетчер памяти, из-за того что задачи появляются и завершаются в произвольные моменты времени и при этом имеют разные объемы, в памяти всегда будет наблюдаться сильная фрагментация. При этом возможны ситуации, когда из-за сильной фрагментации памяти диспетчер задач не сможет образовать новый раздел, хотя суммарный объем свободных областей будет больше, чем необходимо для задачи. В этой ситуации можно организовать так называемое *уплотнение памяти*. Для уплотнения памяти все вычисления приостанавливаются, и диспетчер памяти корректирует свои списки, перемещая разделы в начало памяти (или, наоборот, в область старших адресов). При определении физических адресов задачи будут участвовать новые значения базовых регистров, с помощью которых и осуществляется преобразование виртуальных адресов в физические. Недостатком этого решения является потеря времени на уплотнение и, что самое главное, невозможность при этом выполнять сами вычислительные процессы.

Данный способ распределения памяти, тем не менее, применялся достаточно длительное время в нескольких операционных системах, поскольку в нем для задач выделяется непрерывное адресное пространство, а это упрощает создание систем программирования и их работу. Применяется этот способ и ныне при создании систем на базе контроллеров с упрощенной (по отношению к мощным современным процессорам) архитектурой. Например, при разработке операционной системы для современных цифровых АТС, которая использует 16-разрядные микропроцессоры Intel.

Сегментная, страничная и сегментно-страничная организация памяти

Методы распределения памяти, при которых задаче уже может не предоставляться сплошная (непрерывная) область памяти, называют *разрывными*. Идея выделять память задаче не одной сплошной областью, а фрагментами позволяет уменьшить фрагментацию памяти, однако этот подход требует для своей реализации больше ресурсов, он намного сложнее. Если задать адрес начала текущего фрагмента программы и величину смещения относительно этого начального адреса, то можно указать необходимую нам переменную или команду. Таким образом, виртуальный адрес можно представить состоящим из двух полей. Первое поле будет указывать на ту часть программы, к которой обращается процессор, для определения местоположения этой части в памяти, а второе поле виртуального адреса позволит найти нужную нам ячейку относительно найденного адреса. Программист может либо самостоятельно разбивать программу на фрагменты, либо можно автоматизировать эту задачу, возложив ее на систему программирования.

Сегментный способ организации виртуальной памяти

Первым среди разрывных методов распределения памяти был *сегментный*. Для этого метода программу необходимо разбивать на части и уже каждой такой части выделять физическую память. Естественным способом разбиения программы на части является разбиение ее на логические элементы — так называемые *сегменты*. В принципе, каждый программный модуль (или их совокупность, если мы того пожелаем) может быть воспринят как отдельный сегмент, и вся программа тогда будет представлять собой множество сегментов. Каждый сегмент размещается в памяти как до определенной степени самостоятельная единица. Логически обращение к элементам программы в этом случае будет состоять из имени сегмента и смещения относительно начала этого сегмента. Физически имя (или порядковый номер) сегмента будет соответствовать некоторому адресу, с которого этот сегмент начинается при его размещении в памяти, и смещение должно прибавляться к этому базовому адресу.

Преобразование имени сегмента в его порядковый номер осуществит система программирования. Для каждого сегмента система программирования указывает его объем. Он должен быть известен операционной системе, чтобы она могла выделять ему необходимый объем памяти. Операционная система будет размещать сегменты в памяти и для каждого сегмента она должна вести учет о местонахождении этого сегмента. Вся информация о текущем размещении сегментов задачи в памяти обычно сводится в *таблицу сегментов,* чаще такую таблицу называют *таблицей дескрипторов сегментов задачи*. Каждая задача имеет свою таблицу сегментов. Достаточно часто эти таблицы называют таблицами дескрипторов сегментов, поскольку по своей сути элемент таблицы описывает расположение сегмента.

Таким образом, виртуальный адрес для этого способа будет состоять из двух полей — номера сегмента и смещения относительно начала сегмента. Соответствующая иллюстрация приведена на рис. 3.4 для случая обращения к ячейке, виртуальный адрес которой равен сегменту с номером 11 со смещением от начала этого сегмента, равным 612. Как мы видим, операционная система разместила данный сегмент в памяти, начиная с ячейки с номером 19700.

Итак, каждый сегмент, размещаемый в памяти, имеет соответствующую информационную структуру, часто называемую *дескриптором сегмента*. Именно операционная система строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов, и при размещении каждого из сегментов в оперативной или внешней памяти отмечает в дескрипторе текущее местоположение сегмента. Если сегмент задачи в данный момент находится в оперативной памяти, то об этом делается пометка в дескрипторе. Как правило, для этого используется *бит присутствия* P (от слова «present»). В этом случае в поле адреса диспетчер памяти записывает адрес физической памяти, с которого сегмент начинается, а в поле длины сегмента (limit) указывается количество адресуемых ячеек памяти. Это поле используется не только для того, чтобы размещать сегменты без наложения друг на друга, но и для того, чтобы контролировать, не обращается ли код исполняющейся задачи за пределы текущего сегмента. В случае превышения длины сегмен-

та вследствие ошибок программирования мы можем говорить о нарушении адресации и с помощью введения специальных аппаратных средств генерировать сигналы прерывания, которые позволят фиксировать (обнаруживать) такого рода ошибки.

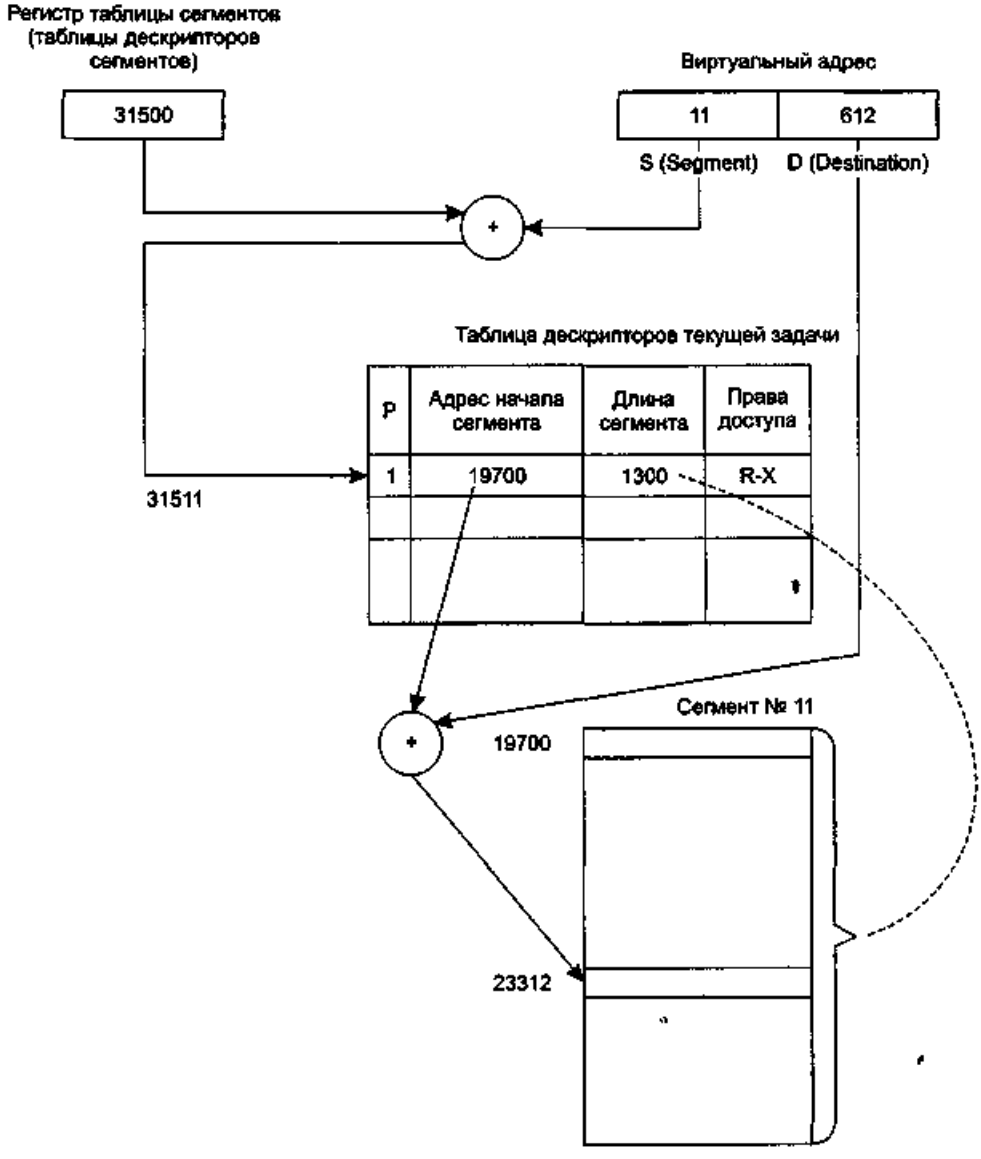

Рис. 3.4. Сегментный способ организации виртуальной памяти

Если бит присутствия в дескрипторе указывает, что сегмент находится не в оперативной, а во внешней памяти (например, на жестком диске), то названные поля

адреса и длины используются для указания адреса сегмента в координатах внешней памяти. Помимо информации о местоположении сегмента, в дескрипторе сегмента, как правило, содержатся данные о его типе (сегмент кода или сегмент данных), правах доступа к этому сегменту (можно или нельзя его модифицировать, предоставлять другой задаче), отметка об обращениях к данному сегменту (информация о том, как часто или как давно этот сегмент используется или не используется, на основании которой можно принять решение о том, чтобы предоставить место, занимаемое текущим сегментом, другому сегменту).

При передаче управления следующей задаче операционная система должна занести в соответствующий регистр адрес таблицы дескрипторов сегментов этой задачи. Сама таблица дескрипторов сегментов, в свою очередь, также представляет собой сегмент данных, который обрабатывается диспетчером памяти операционной системы.

При таком подходе появляется возможность размещать в оперативной памяти не все сегменты задачи, а только задействованные в данный момент. Благодаря этому, с одной стороны, общий объем виртуального адресного пространства задачи может превосходить объем физической памяти компьютера, на котором эта задача будет выполняться; с другой стороны, даже если потребности в памяти не превосходят имеющуюся физическую память, можно размещать в памяти больше задач, поскольку любой задаче, как правило, все ее сегменты единовременно не нужны. А увеличение *коэффициента мультипрограммирования* μ, как мы знаем, позволяет увеличить загрузку системы и более эффективно использовать ресурсы вычислительной системы. Очевидно, однако, что увеличивать количество задач можно только до определенного предела, ибо если в памяти не будет хватать места для часто используемых сегментов, то производительность системы резко упадет. Ведь сегмент, находящийся вне оперативной памяти, для участия в вычислениях должен быть перемещен в оперативную память. При этом если в памяти есть свободное пространство, то необходимо всего лишь найти нужный сегмент во внешней памяти и загрузить его в оперативную память. А если свободного места нет, придется принять решение — на место какого из присутствующих сегментов будет загружаться требуемый. Перемещение сегментов из оперативной памяти на жесткий диск и обратно часто называют *свопингом сегментов*.

Итак, если требуемого сегмента в оперативной памяти нет, то возникает прерывание, и управление передается через диспетчер памяти программе загрузки сегмента. Пока происходит поиск сегмента во внешней памяти и загрузка его в оперативную, диспетчер памяти определяет подходящее для сегмента место. Возможно, что свободного места нет, и тогда принимается решение о выгрузке какого-нибудь сегмента и выполняется его перемещение во внешнюю память. Если при этом еще остается время, то процессор передается другой готовой к выполнению задаче. После загрузки необходимого сегмента процессор вновь передается задаче, вызвавшей прерывание из-за отсутствия сегмента. Всякий раз при считывании сегмента в оперативную память в таблице дескрипторов сегментов необходимо установить адрес начала сегмента и признак присутствия сегмента.

При поиске свободного места используется одна из вышеперечисленных дисциплин работы диспетчера памяти (применяются правила «первого подходящего»

и «самого неподходящего» фрагментов). Если свободного фрагмента памяти достаточного объема нет, но, тем не менее, сумма этих свободных фрагментов превышает требования по памяти для нового сегмента, то в принципе может быть применено «уплотнение памяти», о котором мы уже говорили в подразделе «Разделы с фиксированными границами» раздела «Распределение памяти статическими и динамическими разделами».

В идеальном случае размер сегмента должен быть достаточно малым, чтобы его можно было разместить в случайно освобождающихся фрагментах оперативной памяти, но достаточно большим, чтобы содержать логически законченную часть программы с тем, чтобы минимизировать межсегментные обращения.

Для решения проблемы замещения (определения того сегмента, который должен быть либо перемещен во внешнюю память, либо просто замещен новым) используются следующие дисциплины[1]:

* правило *FIFO* (First In First Out — первый пришедший первым и выбывает);
* правило *LRU* (Least Recently Used — дольше других неиспользуемый);
* правило *LFU* (Least Frequently Used — реже других используемый);
* *случайный* (random) выбор сегмента.

Первая и последняя дисциплины являются самыми простыми в реализации, но они не учитывают, насколько часто используется тот или иной сегмент, и, следовательно, диспетчер памяти может выгрузить или расформировать тот сегмент, к которому в самом ближайшем будущем будет обращение. Безусловно, достоверной информация о том, какой из сегментов потребуется в ближайшем будущем, в общем случае быть не может, но вероятность ошибки для этих дисциплин многократно выше, чем у второй и третьей, в которых учитывается информация об использовании сегментов.

В алгоритме FIFO с каждым сегментом связывается очередность его размещения в памяти. Для замещения выбирается сегмент, первым попавший в память. Каждый вновь размещаемый в памяти сегмент добавляется в хвост этой очереди. Алгоритм учитывает только время нахождения сегмента в памяти, но не учитывает фактическое использование сегментов. Например, первые загруженные сегменты программы могут содержать переменные, требующиеся на протяжении всей ее работы. Это приводит к немедленному возвращению к только что замещенному сегменту.

Для реализации дисциплин LRU и LFU необходимо, чтобы процессор имел дополнительные аппаратные средства. Минимальные требования — достаточно, чтобы при обращении к дескриптору сегмента для получения физического адреса, с которого сегмент начинает располагаться в памяти, соответствующий *бит обращения* менял свое значение (скажем, с нулевого, которое устанавливает операционная система, в единичное). Тогда диспетчер памяти может время от времени просматривать таблицы дескрипторов исполняющихся задач и собирать для соответствующей обработки статистическую информацию об обращениях к сегмен-

[1] Их называют «дисциплинами замещения».

там. В результате можно составить список, упорядоченный либо по длительности простоя (для дисциплины LRU), либо по частоте использования (для дисциплины LFU).

Важнейшей проблемой, которая возникает при организации мультипрограммного режима, является защита памяти. Для того чтобы выполняющиеся приложения не смогли испортить саму операционную систему и другие вычислительные процессы, необходимо, чтобы доступ к таблицам сегментов с целью их модификации был обеспечен только для кода самой ОС. Для этого код операционной системы должен выполняться в некотором привилегированном режиме, из которого можно осуществлять манипуляции дескрипторами сегментов, тогда как выход за пределы сегмента в обычной прикладной программе должен вызывать прерывание по защите памяти. Каждая прикладная задача должна иметь возможность обращаться только к собственным и к общим сегментам.

При сегментном способе организации виртуальной памяти появляется несколько интересных возможностей.

Во-первых, при загрузке программы на исполнение можно размещать ее в памяти не целиком, а «по мере необходимости». Действительно, поскольку в подавляющем большинстве случаев алгоритм, по которому работает код программы, является разветвленным, а не линейным, то в зависимости от исходных данных некоторые части программы, расположенные в самостоятельных сегментах, могут быть не задействованы; значит, их можно и не загружать в оперативную память.

Во-вторых, некоторые программные модули могут быть разделяемыми. Поскольку эти программные модуля являются сегментами, относительно легко организовать доступ к таким общим сегментам. Сегмент с разделяемым кодом располагается в памяти в единственном экземпляре, а в нескольких таблицах дескрипторов сегментов исполняющихся задач будут находиться указатели на такие разделяемые сегменты.

Однако у сегментного способа распределения памяти есть и недостатки. Прежде всего (см. рис. 3.4), для доступа к искомой ячейке памяти приходится тратить много времени. Мы должны сначала найти и прочитать дескриптор сегмента, а уже потом, используя полученные данные о местонахождении нужного нам сегмента, вычислить конечный физический адрес. Для того чтобы уменьшить эти потери, используется кэширование — те дескрипторы, с которыми мы имеем дело в данный момент, могут быть размещены в сверхоперативной памяти (специальных регистрах, размещаемых в процессоре).

Несмотря на то что рассмотренный способ распределения памяти приводит к существенно меньшей фрагментации памяти, нежели способы с неразрывным распределением, фрагментация остается. Кроме того, много памяти и процессорного времени теряется на размещение и обработку дескрипторных таблиц. Ведь на каждую задачу необходимо иметь свою таблицу дескрипторов сегментов. А при определении физических адресов приходится выполнять операции сложения, что требует дополнительных затрат времени.

Поэтому следующим способом разрывного размещения задач в памяти стал способ, при котором все фрагменты задачи считаются равными (одинакового разме-

ра), причем длина фрагмента в идеале должна быть кратна степени двойки, чтобы операции сложения можно было заменить операциями конкатенации (слияния). Это — страничный способ организации виртуальной памяти. Этот способ мы детально рассмотрим ниже.

Примером использования сегментного способа организации виртуальной памяти является операционная система OS/2 первого поколения[1], которая была создана для персональных компьютеров на базе процессора i80286. В этой операционной системе в полной мере использованы аппаратные средства микропроцессора, который специально проектировался для поддержки сегментного способа распределения памяти.

OS/2 v.1 поддерживала распределение памяти, при котором выделялись сегменты программы и сегменты данных. Система позволяла работать как с именованными, так и с неименованными сегментами. Имена разделяемых сегментов данных имели ту же форму, что и имена файлов. Процессы получали доступ к именованным разделяемым сегментам, используя их имена в специальных системных вызовах. Операционная система OS/2 v.1 допускала разделение программных сегментов приложений и подсистем, а также глобальных сегментов данных подсистем. Вообще, вся концепция системы OS/2 была построена на понятии разделения памяти: процессы почти всегда разделяют сегменты с другими процессами. В этом состояло существенное отличие системы OS/2 от систем типа UNIX, которые обычно разделяют только реентерабельные программные модули между процессами.

Сегменты, которые активно не использовались, могли выгружаться на жесткий диск. Система восстанавливала их, когда в этом возникала необходимость. Так как все области памяти, используемые сегментом, должны были быть непрерывными, OS/2 перемещала в основной памяти сегменты таким образом, чтобы максимизировать объем свободной физической памяти. Такое переразмещение сегментов называется уплотнением памяти (компрессией). Программные сегменты не выгружались, поскольку они могли просто перезагружаться с исходных дисков. Области в младших адресах физической памяти, которые использовались для запуска DOS-программ и кода самой OS/2, в компрессии не участвовали. Кроме того, система или прикладная программа могла временно фиксировать сегмент в памяти с тем, чтобы гарантировать наличие буфера ввода-вывода в физической памяти до тех пор, пока операция ввода-вывода не завершится.

Если в результате компрессии памяти не удавалось создать необходимое свободное пространство, то супервизор выполнял операции фонового плана для перекачки достаточного количества сегментов из физической памяти, чтобы дать возможность завершиться исходному запросу.

Механизм перекачки сегментов использовал файловую систему для выгрузки данных из физической памяти и обратно. Ввиду того что перекачка и компрессия влияли на производительность системы в целом, пользователь мог сконфигурировать систему так, чтобы эти функции не выполнялись.

[1] OS/2 v.1 начала создаваться в 1984 году и поступила в продажу в 1987 году.

Было организовано в OS/2 и динамическое присоединение обслуживающих программ. Программы OS/2 используют команды удаленного вызова. Ссылки, генерируемые этими вызовами, определяются в момент загрузки самой программы или ее сегментов. Такое отсроченное определение ссылок называется *динамическим присоединением.* Загрузочный формат модуля OS/2 представляет собой расширение формата загрузочного модуля DOS. Он был расширен, чтобы поддерживать необходимое окружение для свопинга сегментов с динамическим присоединением. Динамическое присоединение уменьшает объем памяти для программ в OS/2, одновременно делая возможными перемещения подсистем и обслуживающих программ без необходимости повторного редактирования адресных ссылок к прикладным программам.

Страничный способ организации виртуальной памяти

Как уже упоминалось, при страничном способе организации виртуальной памяти все фрагменты программы, на которые она разбивается (за исключением последней ее части), получаются одинаковыми. Одинаковыми полагаются и единицы памяти, которые предоставляются для размещения фрагментов программы. Эти одинаковые части называют *страницами* и говорят, что оперативная память разбивается на физические страницы, а программа — на виртуальные страницы. Часть виртуальных страниц задачи размещается в оперативной памяти, а часть — во внешней. Обычно место во внешней памяти, в качестве которой в абсолютном большинстве случаев выступают накопители на магнитных дисках (поскольку они относятся к быстродействующим устройствам с прямым доступом), называют *файлом подкачки,* или *страничным файлом* (paging file). Иногда этот файл называют *swap-файлом,* тем самым подчеркивая, что записи этого файла — страницы — замещают друг друга в оперативной памяти. В некоторых операционных системах выгруженные страницы располагаются не в файле, а в специальном разделе дискового пространства[1].

Разбиение всей оперативной памяти на страницы одинаковой величины, причем кратной степени двойки, приводит к тому, что вместо одномерного адресного пространства памяти можно говорить о двухмерном. Первая координата адресного пространства — это номер страницы, вторая координата — номер ячейки внутри выбранной страницы (его называют индексом). Таким образом, физический адрес определяется парой (P_p, i), а виртуальный адрес — парой (P_v, i), где P_v — номер виртуальной страницы, P_p — номер физической страницы, i — индекс ячейки внутри страницы. Количество битов, отводимое под индекс, определяет размер страницы, а количество битов, отводимое под номер виртуальной страницы, — объем потенциально доступной для программы виртуальной памяти. Отображение, осуществляемое системой во время исполнения, сводится к отображению P_v в P_p и приписыванию к полученному значению битов адреса, задаваемых величиной i. При этом

[1] В UNIX-системах для этих целей выделяется специальный раздел, но кроме него могут быть использованы и файлы, выполняющие те же функции, если объема раздела недостаточно.

нет необходимости ограничивать число виртуальных страниц числом физических, то есть не поместившиеся страницы можно размещать во внешней памяти, которая в данном случае служит расширением оперативной.

Для отображения виртуального адресного пространства задачи на физическую память, как и в случае сегментного способа организации, для каждой задачи необходимо иметь *таблицу страниц* для трансляции адресных пространств. Для описания каждой страницы диспетчер памяти операционной системы заводит соответствующий дескриптор, который отличается от дескриптора сегмента прежде всего тем, что в нем нет поля длины — ведь все страницы имеют одинаковый размер. По номеру виртуальной страницы в таблице дескрипторов страниц текущей задачи находится соответствующий элемент (дескриптор). Если бит присутствия имеет единичное значение, значит данная страница размещена в оперативной, а не во внешней памяти, и мы в дескрипторе имеем номер физической страницы, отведенной под данную виртуальную. Если же бит присутствия равен нулю, то в дескрипторе мы будем иметь адрес виртуальной страницы, расположенной во внешней памяти. Таким образом и осуществляется трансляция виртуального адресного пространства на физическую память. Этот механизм трансляции иллюстрирует рис. 3.5.

Защита страничной памяти, как и в случае сегментного механизма, основана на контроле уровня доступа к каждой странице. Как правило, возможны следующие уровни доступа:

* только чтение;
* чтение и запись;
* только выполнение.

Каждая страница снабжается соответствующим кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

При обращении к виртуальной странице, не оказавшейся в данный момент в оперативной памяти, возникает прерывание, и управление передается диспетчеру памяти, который должен найти свободное место. Обычно предоставляется первая же свободная страница. Если свободной физической страницы нет, то диспетчер памяти по одной из вышеупомянутых дисциплин замещения (LRU, LFU, FIFO, случайный доступ) определит страницу, подлежащую расформированию или сохранению во внешней памяти. На ее месте он разместит новую виртуальную страницу, к которой было обращение из задачи, но которой не оказалось в оперативной памяти.

Напомним, что алгоритм LFU выбирает для замещения ту страницу, на которую не было ссылки на протяжении наиболее длительного периода времени. Алгоритм LRU ассоциирует с каждой страницей время ее последнего использования. Для замещения выбирается та страница, которая дольше всех не использовалась.

Для использования дисциплин LRU и LFU в процессоре должны быть соответствующие аппаратные средства. В дескрипторе страницы размещается бит обращения (на рис. 3.5 подразумевается, что этот бит расположен в последнем поле), который становится единичным при обращении к дескриптору.

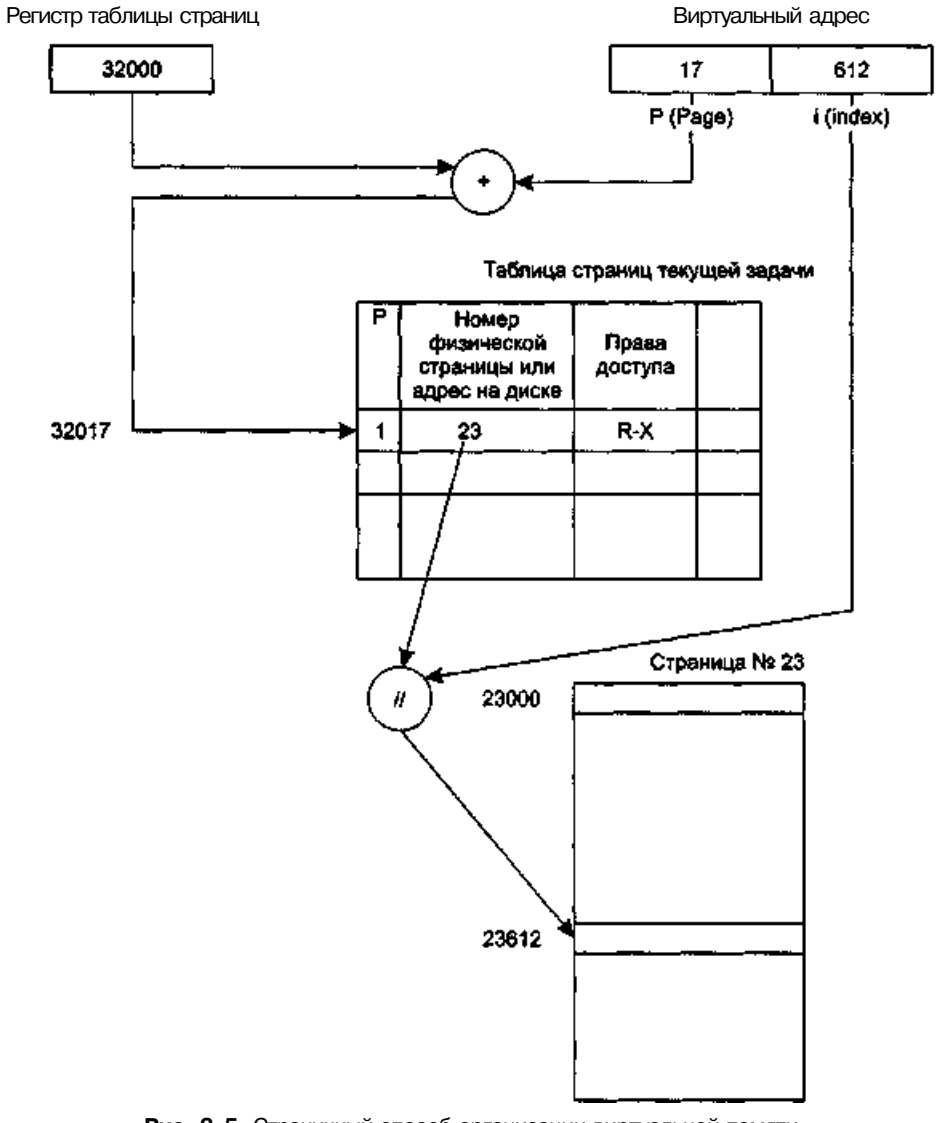

Рис. 3.5. Страничный способ организации виртуальной памяти

Если объем физической памяти небольшой и даже часто требуемые страницы не удается разместить в оперативной памяти, возникает так называемая «пробуксовка». Другими словами, *пробуксовка* — это ситуация, при которой загрузка нужной страницы вызывает перемещение во внешнюю память той страницы, с которой мы тоже активно работаем. Очевидно, что это очень плохое явление. Чтобы его не допускать, желательно увеличить объем оперативной памяти (сейчас это просто, поскольку стоимость модуля оперативной памяти многократно снизилась), уменьшить количество параллельно выполняемых задач или прибегнуть к более эффективным дисциплинам замещения.

Для абсолютного большинства современных операционных систем характерна дисциплина замещения страниц LRU как самая эффективная. Так, именно эта дисциплина используется в OS/2 и в Linux. Однако в операционных системах Windows NT/2000/XP разработчики, желая сделать их максимально независимыми от аппаратных возможностей процессора, отказались от этой дисциплины и применили правило FIFO. А для того чтобы хоть как-то компенсировать неэффективность правила FIFO, была введена «буферизация» тех страниц, которые должны быть записаны в файл подкачки на диск[1] или просто расформированы. Принцип буферизации прост. Прежде чем замещаемая страница действительно окажется во внешней памяти или просто расформированной, она помечается как кандидат на выгрузку. Если в следующий раз произойдет обращение к странице, находящейся в таком «буфере», то страница никуда не выгружается и уходит в конец списка FIFO. В противном случае страница действительно выгружается, а на ее место в «буфер» попадает следующий «кандидат». Величина такого «буфера» не может быть большой, поэтому эффективность страничной реализации памяти в Windows NT/2000/XP намного ниже, чем в других операционных системах, и явление пробуксовки начинается даже при существенно большем объеме оперативной памяти.

В ряде операционных систем с пакетным режимом работы для борьбы с пробуксовкой используется метод «рабочего множества». *Рабочее множество* — это множество «активных» страниц задачи за некоторый интервал T, то есть тех страниц, к которым было обращение за этот интервал времени. Реально количество активных страниц задачи (за интервал T) все время изменяется, и это естественно, но, тем не менее, для каждой задачи можно определить среднее количество ее активных страниц. Это количество и есть рабочее множество задачи. Наблюдения за исполнением множества различных программ показали [11, 17, 22], что даже если интервал T равен времени выполнения всей работы, то размер рабочего множества часто существенно меньше, чем общее число страниц программы. Таким образом, если операционная система может определить рабочие множества исполняющихся задач, то для предотвращения пробуксовки достаточно планировать на выполнение только такое количество задач, чтобы сумма их рабочих множеств не превышала возможности системы.

Как и в случае с сегментным способом организации виртуальной памяти, страничный механизм приводит к тому, что без специальных аппаратных средств он существенно замедляет работу вычислительной системы. Поэтому обычно используется кэширование страничных дескрипторов. Наиболее эффективным механизмом кэширования является ассоциативный кэш. Именно такой ассоциативный кэш и создан в 32-разрядных микропроцессорах i80x86. Начиная с i80386, который поддерживает страничный способ распределения памяти, в этих микропроцессорах имеется кэш на 32 страничных дескриптора. Поскольку размер страницы в этих

[1] В системе Windows NT файл с выгруженными виртуальными страницами носит название PageFile.sys. Таких файлов может быть несколько. Их совокупный размер должен быть не менее, чем объем физической памяти компьютера плюс 11 Мбайт, необходимых для самой Windows NT. В системах Windows 2000 размер файла PageFile.sys намного превышает объем установленной физической памяти и часто достигает многих сотен мегабайтов.

микропроцессорах равен 4 Кбайт, возможно быстрое обращение к памяти размером 128 Кбайт.

Итак, основным достоинством страничного способа распределения памяти является минимальная фрагментация. Поскольку на каждую задачу может приходиться по одной незаполненной странице, очевидно, что память можно использовать достаточно эффективно; этот метод организации виртуальной памяти был бы одним из самых лучших, если бы не два следующих обстоятельства.

Первое — это то, что страничная трансляция виртуальной памяти требует существенных накладных расходов. В самом деле, таблицы страниц нужно тоже размещать в памяти. Кроме того, эти таблицы нужно обрабатывать; именно с ними работает диспетчер памяти.

Второй существенный недостаток страничной адресации заключается в том, что программы разбиваются на страницы случайно, без учета логических взаимосвязей, имеющихся в коде. Это приводит к тому, что межстраничные переходы, как правило, осуществляются чаще, нежели межсегментные, и к тому, что становится трудно организовать разделение программных модулей между выполняющимися процессами.

Для того чтобы избежать второго недостатка, постаравшись сохранить достоинства страничного способа распределения памяти, был предложен еще один способ — сегментно-страничный. Правда, за счет увеличения накладных расходов на его реализацию.

Сегментно-страничный способ организации виртуальной памяти

Как и в сегментном способе распределения памяти, программа разбивается на логически законченные части — сегменты — и виртуальный адрес содержит указание на номер соответствующего сегмента. Вторая составляющая виртуального адреса — смещение относительно начала сегмента — в свою очередь может быть представлено состоящим из двух полей: виртуальной страницы и индекса. Другими словами, получается, что виртуальный адрес теперь состоит из трех компонентов: сегмента, страницы и индекса. Получение физического адреса и извлечение из памяти необходимого элемента для этого способа иллюстрирует рис. 3.6.

Из рисунка сразу видно, что этот способ организации виртуальной памяти вносит еще большую задержку доступа к памяти. Необходимо сначала вычислить адрес дескриптора сегмента и прочитать его, затем определить адрес элемента таблицы страниц этого сегмента и извлечь из памяти необходимый элемент и уже только после этого можно к номеру физической страницы приписать номер ячейки в странице (индекс). Задержка доступа к искомой ячейке получается, по крайней мере, в три раза больше, чем при простой прямой адресации. Чтобы избежать этой неприятности, вводится кэширование, причем кэш, как правило, строится по ассоциативному принципу. Другими словами, просмотры двух таблиц в памяти могут быть заменены одним обращением к ассоциативной памяти.

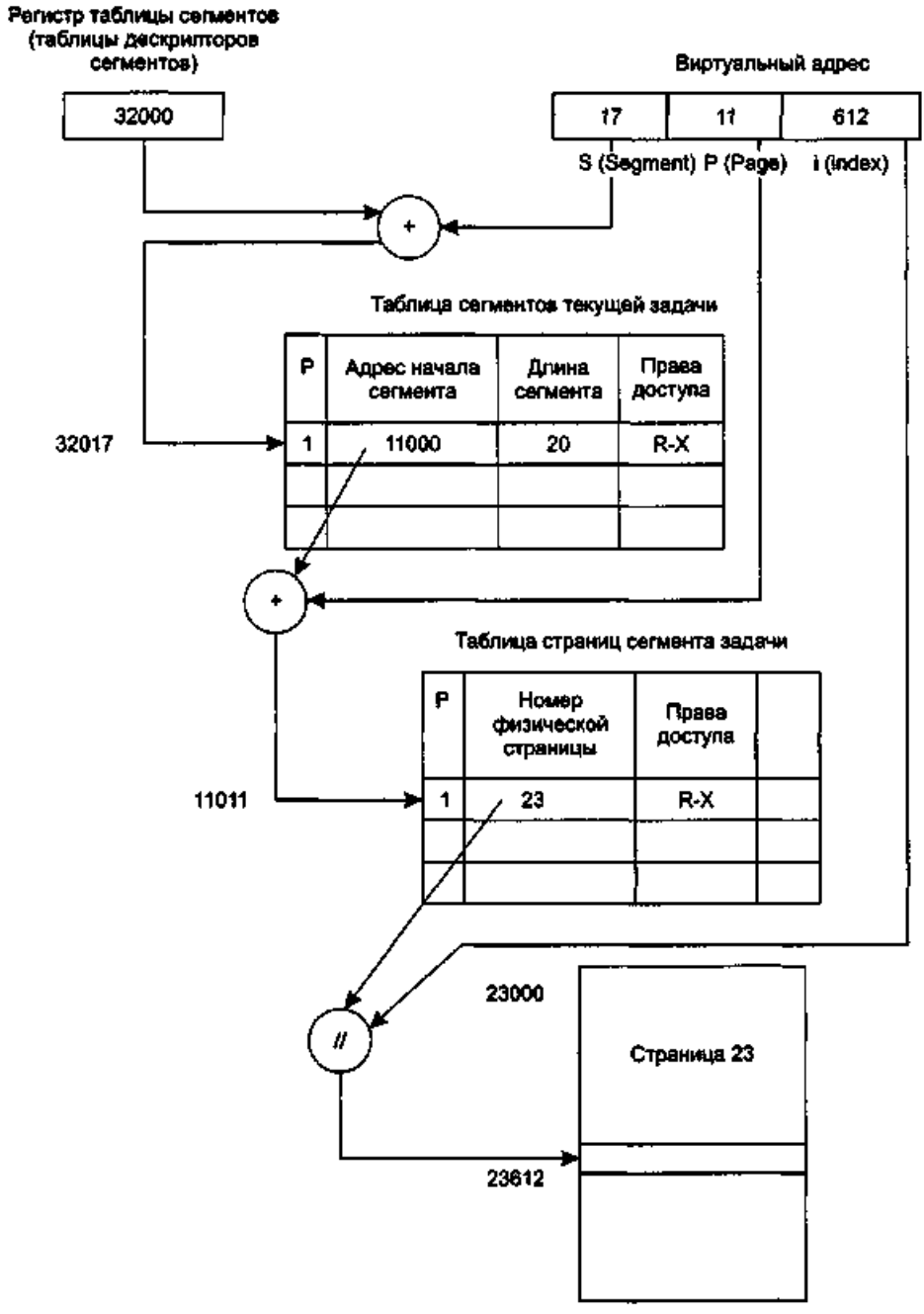

Рис. 3.6. Сегментно-страничный способ организации виртуальной памяти

Напомним, что принцип действия ассоциативного запоминающего устройства предполагает, что каждой ячейке памяти такого устройства ставится в соответ-

ствие ячейка, в которой записывается некий ключ (признак, адрес), позволяющий однозначно идентифицировать содержимое ячейки памяти. Сопутствующую ячейку с информацией, позволяющей идентифицировать основные данные, обычно называют *полем тега*. Просмотр полей тега всех ячеек ассоциативного устройства памяти осуществляется одновременно, то есть в каждой ячейке тега есть необходимая логика, позволяющая посредством побитовой конъюнкции найти данные по их признаку за одно обращение к памяти (если они там, конечно, присутствуют). Часто поле тегов называют аргументом, а поле с данными — функцией. В данном случае в качестве аргумента при доступе к ассоциативной памяти выступают номер сегмента и номер виртуальной страницы, а в качестве функции от этих аргументов получаем номер физической страницы. Остается приписать номер ячейки в странице к полученному номеру, и мы получаем адрес искомой команды или операнда.

Оценим достоинства сегментно-страничного способа. Разбиение программы на сегменты позволяет размещать сегменты в памяти целиком. Сегменты разбиты на страницы, все страницы сегмента загружаются в память. Это позволяет сократить число обращений к отсутствующим страницам, поскольку вероятность выхода за пределы сегмента меньше вероятности выхода за пределы страницы. Страницы исполняемого сегмента находятся в памяти, но при этом они могут находиться не рядом друг с другом, а «россыпью», поскольку диспетчер памяти манипулирует страницами. Наличие сегментов облегчает разделение программных модулей между параллельными процессами. Возможна и динамическая компоновка задачи. А выделение памяти страницами позволяет минимизировать фрагментацию.

Однако поскольку этот способ распределения памяти требует очень значительных затрат вычислительных ресурсов и его не так просто реализовать, используется он редко, причем в дорогих мощных вычислительных системах. Возможность реализовать сегментно-страничное распределение памяти заложена и в семейство микропроцессоров i80x86, однако вследствие слабой аппаратной поддержки, трудностей при создании систем программирования и операционной системы практически в персональных компьютерах эта возможность не используется.

Контрольные вопросы и задачи

1. Что такое «виртуальный адрес», «виртуальное адресное пространство»? Чем (в общем случае) определяется максимально возможный объем виртуального адресного пространства программы?
2.. Имеются ли виртуальные адреса в программах, написанных для работы в среде DOS? Приведите примеры абсолютной двоичной программы для таких операционных систем, как MS DOS и Windows NT/2000/XP.
3. Изложите способ распределения памяти в MS DOS.
4. Что дает использование оверлеев при разработке DOS-приложений?
5. Объясните и сравните алгоритмы «первый подходящий», «самый подходящий» и «самый неподходящий», используемые при поиске и выделении фрагмента памяти.

6. Что такое «фрагментация памяти»? Какой метод распределения памяти позволяет добиться минимальной фрагментации и почему?
7. Что такое «уплотнение памяти»? Когда оно применяется?
8. Объясните сегментный способ организации виртуальной памяти. Что представляет собой (в общем случае) дескриптор сегмента?
9. Что представляет собой динамическое присоединение программ? Что оно дает?
10. Сравните сегментный и страничный способы организации виртуальной памяти. Перечислите достоинства и недостатки каждого.
11. Какие дисциплины применяются для решения задачи замещения страниц? Какие из них являются наиболее эффективными и как они реализуются?
12. Что такое «рабочее множество»? Что позволяет разрешить реализация этого понятия?
13. В каких случаях возникает «пробуксовка»? Почему системы Windows NT/2000/XP требуют для своей нормальной работы существенно большего объема оперативной памяти?