МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

Кафедра инженерной кибернетики

С.В. Назаров, А.И. Широков

Технологии многопользовательских операционных систем

Монография

Под редакцией профессора С.В. Назарова

Москва 2012

УДК 004.45
Н19

Р е ц е н з е н т
к.т.н., с.н.с., доц. кафедры моделирования
в экономике и управлении РГГУ *В.В. Муромцев*

Назаров, С.В.
Н19 Технологии многопользовательских систем : моногр. / С.В. Назаров, А.И. Широков ; под ред. С.В. Назарова. – М. : Изд. Дом МИСиС, 2012. – 296 с.
ISBN 978-5-87623-633-3

В монографии представлено описание основных технологий современных операционных систем: методы, алгоритмы и средства управление памятью современного компьютера; организация подсистемы ввода-вывода; архитектура и средства управления файловой системой; этапы загрузки операционных систем; файлы, сохраняющие параметры операционных систем; средства языка программирования оболочки Linux; безопасность, диагностика и восстановление операционных систем после отказов.

Рассматриваемые технологии иллюстрируются примерами из двух наиболее распространенных представителей программных систем этого класса: семейств UNIX/Linux и Windows.

Издание рассчитано в первую очередь на специалистов, которые обеспечивают реализацию технологий многопользовательских операционных систем, а также будет полезно для студентов разных специальностей, обучающихся по информационным технологиям.

УДК 004.45

ISBN 978-5-87623-633-3

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Монография состоит из пяти глав, содержание которых достаточно подробно освещает основополагающие принципы технологий многопользовательских операционных систем. В первой главе рассматривается архитектура памяти современных персональных компьютеров, в том числе методы, алгоритмы и средства управления памятью. Особое внимание уделено методам реализации виртуальной памяти: рассмотрена страничная и сегментная организация виртуальной памяти, а так же их комбинация. Уделено внимание вопросам оптимизации функционирования виртуальной памяти.

Во второй главе описывается эволюция технологий подсистем ввода-вывода и многоуровневая архитектура таких подсистем. Уделяется внимание вопросам согласования скоростей работы ядра компьютера и периферийных устройств. Подробно рассмотрены назначение, задачи и технологии подсистемы ввода-вывода; организация логического интерфейса между внешними устройствами и системой; назначение и способы реализации драйверов.

В третьей главе обсуждаются методы и средства организации файловых систем. Подробно рассмотрены технологии, связанные с файловыми системами современных персональных компьютеров. Освещены такие вопросы, как архитектура файловых систем, организация файлов и доступа к ним, логическая и физическая организация файловых систем современных компьютеров, реализация файловых операций и контроля доступа к файлам.

Четвертая глава содержит четыре части. Первая из них посвящена загрузке операционных систем. Подробно описаны этапы такого процесса для операционных систем Linux и Windows XP. Далее приводятся средства языка программирования, поддерживаемого оболочкой Linux. Это необходимо для понимания устройства системных файлов операционных систем семейства UNIX/Linux, описанных в третьей части. И в заключительной части рассматривается описание реестра Windows.

Последняя, пятая, глава описывает технологии обеспечения безопасности операционных систем, их диагностики и восстановления после отказов. В этой части на основе системного подхода даны основные определения безопасности компьютерных систем, классификация угроз для компьютерных систем и зловредного программного

обеспечения. Приводятся основные технологии, обеспечивающие безопасное функционирование современных компьютеров, методы и средства из восстановления.

Работа над монографией распределилась следующим образом. Первая, вторая, третья и пятая главы написаны С.В. Назаровым. В третьей главе разделы 3.6, 3.10 и 3.11 написаны А.И. Широковым. В четвертой главе раздел 3.4 написан С.В. Назаровым, а разделы 4.1, 4.2 и 4.3 – А.И. Широковым.

Общее редактирование книги выполнено С.В. Назаровым.

ГЛАВА 1. УПРАВЛЕНИЕ ПАМЯТЬЮ. МЕТОДЫ, АЛГОРИТМЫ И СРЕДСТВА

1.1. Организация памяти современного компьютера

Со времен создания первых ЭВМ основная память в компьютерной системе организована как *линейное* (одномерное) адресное пространство, состоящее из последовательности слов, а начиная с машин третьего поколения – байтов [1]. Аналогично организована и внешняя память. Такая организация отражает особенности используемого аппаратного обеспечения, но в недостаточной степени соответствует современной технологии создания программ. Большинство программ организованы в виде модулей, некоторые из них неизменны (только для чтения, только для исполнения), а другие содержат данные, которые могут быть изменены.

Если операционная система и аппаратное обеспечение могут эффективно работать с пользовательскими программами и данными, представленными модулями, то это обеспечивает ряд преимуществ:

• модули могут быть созданы и скомпилированы независимо друг от друга, при этом все ссылки из одного модуля в другой разрешаются системой во время работы программы;

• разные модули могут получать разные степени защиты (только чтение, только исполнение) за счет весьма умеренных накладных расходов;

• возможно применение механизма, обеспечивающего совместное использование модулей разными процессами (для случая сотрудничества процессов в работе над одной задачей).

Память – важнейший ресурс вычислительной системы, требующий эффективного управления. Несмотря на то что в наши дни память среднего домашнего компьютера в тысячи раз превышает память больших ЭВМ 1970-х гг., программы увеличиваются в размере быстрее, чем память. Достаточно сказать, что только операционная система занимает сотни и тысячи мегабайт, не говоря о прикладных программах и базах данных, которые могут занимать в вычислительных системах десятки и сотни гигабайт.

Перефразированный закон Паркинсона гласит: «Программы расширяются, стремясь заполнить весь объем памяти, доступный для их

поддержки» (сказано это было об ОС). В идеале программисты хотели бы иметь неограниченную по размеру и скорости обращения память, которая была бы энергонезависимой, т.е. сохраняла свое содержимое при отключении электричества, а также недорого бы стоила. Однако в реальности пока такой памяти нет. В то же время на любом этапе развития технологии производства запоминающих устройств действуют следующие достаточно устойчивые соотношения:

- чем меньше время доступа, тем дороже бит;
- чем выше емкость, тем ниже стоимость бита;
- чем выше емкость, тем больше время доступа.

Чтобы найти выход из сложившийся ситуации, необходимо опираться не на отдельно взятые компоненты или технологию, а выстроить иерархию запоминающих устройств, показанную на рис. 1.1. При перемещении слева направо происходит следующее[1]:

- снижается стоимость бита;
- возрастает емкость;
- возрастает время доступа;
- снижается частота обращений процессора к памяти.

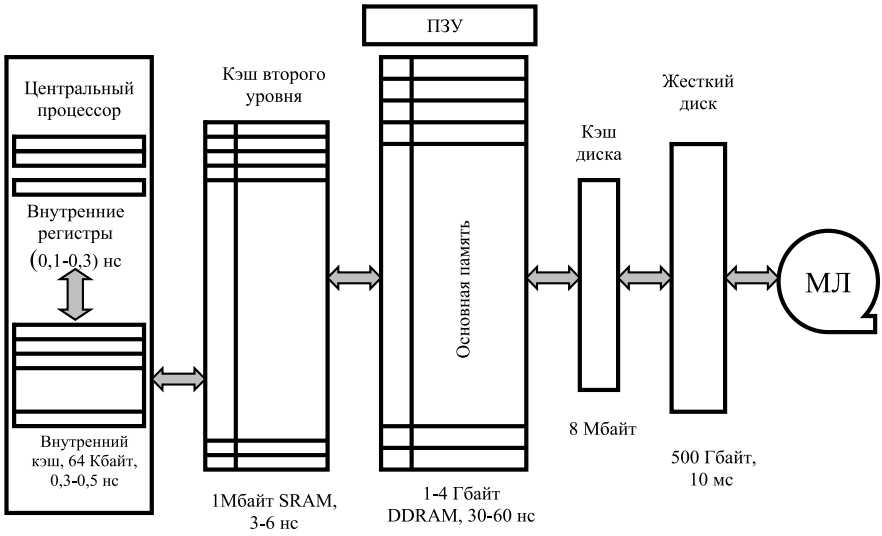

Рис. 1.1. Иерархия видов памяти

[1] Даны примерные характеристики офисного ПК 2008–2011 г.

Предположим, процессор имеет доступ к памяти двух уровней. На первом уровне содержится E_1 слов, и он характеризуется временем доступа $T_1 = 1$ нс. К этому уровню процессор может обращаться непосредственно. Однако если требуется получить слово, находящееся на втором уровне, то его сначала нужно передать на первый уровень. При этом передается не только требуемое слово, но и блок данных, содержащий это слово. Поскольку адреса, к которым обращается процессор, имеют тенденцию собираться в группы (циклы, подпрограммы), процессор обращается к небольшому повторяющему набору команд. Таким образом, работа процессора с вновь полученным блоком памяти будет осуществляться в течение достаточно длительного времени.

Обозначим через $T_2 = 10$ нс время обращения ко второму уровню памяти, а через P – отношение числа нахождений нужного слова в быстрой памяти к числу всех обращений. Пусть в нашем примере $P = 0{,}95$ (т.е. 95 % обращений приходится на быструю память, что вполне реально), тогда среднее время доступа к памяти можно выразить так:

$$T_{ср} = 0{,}95 \cdot 1 \text{ нс} + 0{,}05 \cdot (1 \text{ нс} + 10 \text{ нс}) = 1{,}50 \text{ нс}.$$

Этот принцип можно применять не только к памяти с двумя уровнями. В реальности так и происходит. Объем оперативной памяти существенно сказывается на характере протекания вычислительного процесса, так как он ограничивает число одновременно выполняющихся программ, т.е. *уровень мультипрограммирования*. Если предположить, что процесс проводит часть p своего времени в ожидании завершения операции ввода-вывода, то степень загрузки Z центрального процессора (ЦП) в идеальном случае будет выражаться зависимостью

$$Z = 1 - p^n,$$

где n – число процессов.

На рис. 1.2 показана зависимость $Z = p(n)$ для различного времени ожидания завершения операции ввода-вывода (20, 50 и 80 %) и числа процессов n. Большое количество задач, необходимое для высокой загрузки процессора, требует большого объема оперативной памяти. В условиях, когда для обеспечения приемлемого уровня мультипрограммирования имеющейся памяти недостаточно, был предложен метод организации вычислительного процесса, при котором образы некоторых процессов целиком или частично временно выгружаются на диск.

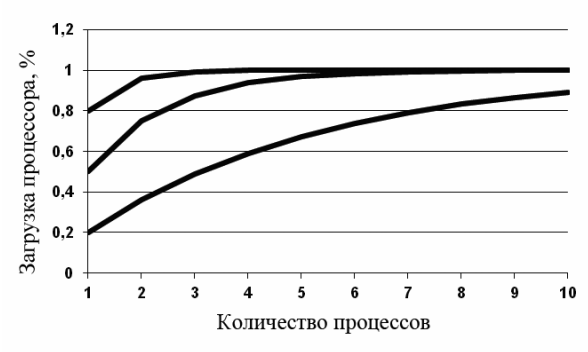

Рис. 1.2. Загрузка процессора при различном числе процессов

Очевидно, что имеет смысл временно выгружать неактивные процессы, находящиеся в ожидании каких-либо ресурсов, в том числе очередного кванта времени центрального процессора. К моменту, когда пройдет очередь выполнения выгруженного процесса, его образ возвращается с диска в оперативную память. Если при этом обнаруживается, что свободного места в оперативной памяти не хватает, то на диск выгружается другой процесс.

Такая подмена (виртуализация) оперативной памяти дисковой памяти позволяет повысить уровень мультипрограммирования, поскольку объем оперативной памяти теперь не столь жестко ограничивает число одновременно выполняемых процессов. При этом суммарный объем оперативной памяти, занимаемой образами процессов, может существенно превосходить имеющийся объем оперативной памяти.

В данном случае в распоряжение прикладного программиста предоставляется виртуальная оперативная память, размер которой намного превосходит реальную память системы и ограничивается только возможностями адресации используемого процесса (в ПК на базе Pentium 2^{32} = 4 Гбайт). Вообще виртуальным (кажущимся) называется ресурс, обладающий свойствами (в данном случае большим объемом ОП), которых в действительности у него нет. Виртуализация оперативной памяти осуществляется совокупностью аппаратных и программных средств вычислительной системы (схемами процессора и операционной системой) автоматически без участия программиста и не сказывается на логике работы приложения.

Виртуализация памяти возможна на основе двух возможных подходов [2]:

• ***свопинг*** (swapping) – образы процессов выгружаются на диск и возвращаются в оперативную память целиком;

• ***виртуальная память*** (virtual memory) – между оперативной памятью и диском перемещаются части образов (сегменты, страницы, блоки и т.п.) процессов.

Недостатки свопинга:

• избыточность перемещаемых данных и, как следствие, замедление работы системы и неэффективное использование памяти;

• невозможность загрузить процесс, виртуальное пространство которого превышает имеющуюся в наличии свободную память.

Достоинство свопинга по сравнению с виртуальной памятью – меньшие затраты времени на преобразование адресов в кодах программ, поскольку оно производится один раз при загрузке с диска в память (однако это преимущество может быть незначительным, так как при очередной загрузке выполняется только часть кода и полного преобразования кода в большинстве случаев не требуется).

Виртуальная память не имеет указанных недостатков, но ее ключевой проблемой является преобразование виртуальных адресов в физические, что приводит к существенным затратам времени на этот процесс, если не принять специальных мер.

1.2. Функции ОС по управлению памятью

Под памятью в данном случае подразумевается оперативная (основная) память компьютера. В однопрограммных операционных системах основная память разделяется на две части. Одна часть предназначена для операционной системы (резидентный монитор, ядро), а вторая – для выполняющейся в текущий момент времени программы. В многопрограммных ОС «пользовательская» часть памяти – важнейший ресурс вычислительной системы – должна быть распределена для размещения нескольких процессов, в том числе процессов ОС. Эта задача распределения выполняется динамически специальной подсистемой управления памятью (memory management) операционной системы. Эффективное управление памятью жизненно важно для многозадачных систем. Если в памяти будет находиться небольшое число процессов, то значительную часть времени процессы будут

находиться в состоянии ожидания ввода-вывода и загрузка процессора будет низкой.

В первых ОС управление памятью сводилось просто к загрузке программы и ее данных из некоторого внешнего накопителя (перфоленты, магнитной ленты или магнитного диска) в ОЗУ. При этом память разделялась между программой и ОС. На рис. 1.3 показаны три варианта такой схемы. Первый вариант раньше применялся в мэйнфреймах и мини-компьютерах. Второй вариант сейчас используется в некоторых карманных компьютерах и встроенных системах, третий вариант был характерен для первых персональных компьютеров с MS DOS.

Рис. 1.3. Варианты распределения памяти

С появлением мультипрограммирования задачи ОС, связанные с распределением имеющейся памяти между несколькими одновременно выполняющимися программами, существенно усложнились.

Функциями ОС по управлению памятью в мультипрограммных системах являются:

• отслеживание (учет) свободной и занятой памяти;

• первоначальное и динамическое выделение памяти процессам приложений и самой операционной системе и освобождение памяти по завершении процессов;

• настройка адресов программы на конкретную область физической памяти;

• полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП недостаточны для размещения всех процессов, и возвращение их в ОП;

• защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;

• дефрагментация памяти.

Перечисленные функции особого пояснения не требуют, остановимся только на задаче преобразования адресов программы при ее загрузке в ОП. Для идентификации переменных и команд на разных этапах жизненного цикла программы используются *символьные* имена, *виртуальные* (математические, условные, логические – все это синонимы) и *физические* адреса (рис. 1.4).

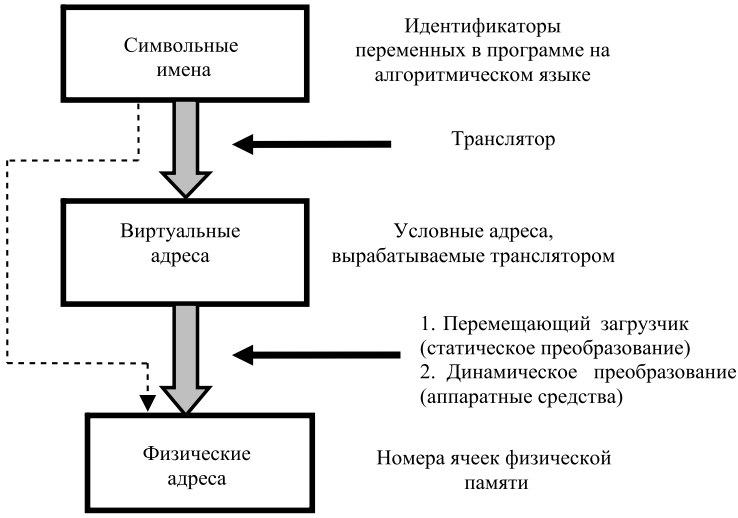

Рис. 1.4. Типы адресов

Символьные имена присваивает пользователь при написании программ на алгоритмическом языке высокого уровня или языке Ассемблер. Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Поскольку во время трансляции неизвестно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, считая по умолчанию, что начальным адресом программы будет нулевой адрес.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности будут расположены переменные и команды.

Совокупность виртуальных адресов процесса называется виртуальным адресным пространством. Диапазон адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для Pentium адресное пространство составляет объем, равный 2^{32} байт, с диапазоном адресов от 0000.0000_{16} до $FFFF.FFFF_{16}$).

Существует два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические. В первом случае такое преобразование выполняется один раз для каждого процесса во время начальной загрузки программы в память. Преобразование осуществляет перемещающий загрузчик на основании имеющихся у него данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставляемой транслятором об адресно-зависимых элементах программы. Второй способ заключается в том, что программа загружается в память в виртуальных адресах. Во время выполнения программы при каждом обращении к памяти операционная система преобразует виртуальные адреса в физические.

1.3. Распределение памяти

Существует ряд базовых вопросов управления памятью, которые в различных ОС решаются по-разному. Например, следует ли назначать каждому процессу одну непрерывную область физической памяти или можно выделять память участками? Должны ли сегменты программы, загруженные в память, находиться на одном месте в течение всего периода выполнения процесса или их можно время от времени сдвигать? Что делать, если сегменты программы не помещаются в имеющейся памяти? Как сократить затраты ресурсов системы на управление памятью? Имеется и ряд других не менее интересных проблем управления памятью [1–4].

Ниже приводится классификация методов распределения памяти, в которой выделено два класса методов – с перемещением сегментов процессов между ОП и ВП (диском) и без перемещения, т.е. без привлечения внешней памяти (рис. 1.5). Данная классификация учитывает только основные признаки методов. Для каждого метода может быть использовано несколько различных алгоритмов его реализации.

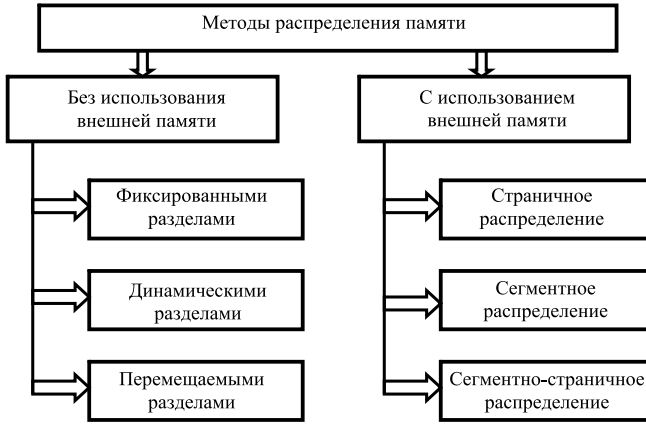

Рис. 1.5. Классификация методов распределения памяти

На рис. 1.6 показано два примера фиксированного распределения. Одна возможность состоит в использовании разделов одинакового размера. В этом случае любой процесс, размер которого не превышает размер раздела, может быть загружен в любой доступный раздел. Если все разделы заняты и нет ни одного процесса в состоянии готовности или работы, ОС может выгрузить процесс из любого раздела и загрузить другой процесс, обеспечивая тем самым процессор работой.

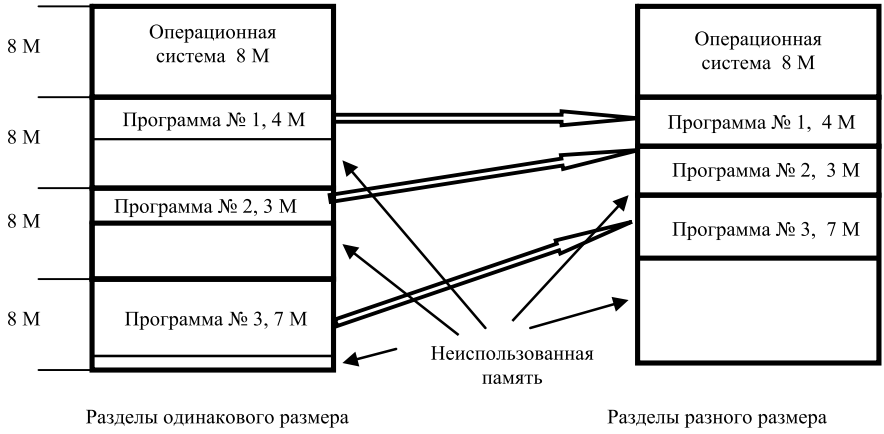

Рис. 1.6. Варианты фиксированного распределения памяти:
М – Мбайт

При использовании разделов с одинаковым размером имеется две трудности:

1. Программа может быть слишком велика для размещения в разделе. В этом случае программист должен разрабатывать программу, использующую оверлеи, чтобы в любой момент времени требовался только один раздел памяти. Когда требуется модуль, отсутствующий в данный момент в ОП, пользовательская программа должна сама его загрузить в раздел памяти программы. Таким образом, в данном случае управление памятью во многом возлагается на программиста.

2. Использование ОП крайне неэффективно. Любая программа, независимо от ее размера, занимает раздел целиком. При этом могут оставаться неиспользованные участки памяти большого размера. Этот феномен появления неиспользованной памяти называется *внутренней фрагментацией* (internal fragmentation).

Бороться с этими трудностями (хотя и невозможно устранить их полностью) можно посредством использования разделов разных размеров. В этом случае программа размером до 8 Мбайт может обойтись без оверлеев, а разделы малого размера позволяют уменьшить внутреннюю фрагментацию при загрузке небольших программ. В том случае, когда разделы имеют одинаковый размер, размещение процессов тривиально – в любой свободный раздел. Если все разделы заняты процессами, которые не готовы к немедленной работе, любой из них может быть выгружен для освобождения памяти для нового процесса.

Когда разделы имеют разные размеры, есть два возможных подхода к назначению процессов разделам памяти. Простейший путь состоит в том, чтобы каждый процесс размещался в наименьшем разделе, способном вместить данный процесс (в этом случае в задании пользователя указывается размер требуемой памяти). При таком подходе для каждого раздела требуется очередь планировщика, в которой хранятся выгруженные из памяти процессы, предназначенные для данного раздела памяти. Достоинство такого способа заключается в возможности распределения процессов между разделами ОП так, чтобы минимизировать внутреннюю фрагментацию.

Недостаток заключается в том, что отдельные очереди для разделов могут привести к неоптимальности распределения памяти для системы в целом. Например, если в некоторый момент времени нет ни одного процесса размером от 7 до 12 Мбайт, то раздел размером 12 Мбайт будет пустовать, в то время как он мог бы использоваться

меньшими процессами. Поэтому более предпочтительным является использование одной очереди для всех процессов. В момент, когда требуется загрузить процесс в ОП, выбирается наименьший доступный раздел, способный вместить данный процесс.

В целом можно отметить, что схемы с фиксированными разделами относительно просты, поэтому предъявляются минимальные требования к операционной системе; накладные расходы работы процессора на распределение памяти невелики. Однако у этих схем имеются серьезные недостатки:

1. Количество разделов, определенное в момент генерации системы, ограничивает количество активных процессов (т.е. уровень мультипрограммирования).

2. Поскольку размеры разделов устанавливаются заранее во время генерации системы, небольшие задания приводят к неэффективному использованию памяти. В средах, где заранее известны потребности в памяти всех задач, применение рассмотренной схемы может быть оправданно, но в большинстве случаев эффективность этой технологии крайне низка.

Для преодоления сложностей, связанных с фиксированным распределением, был разработан альтернативный подход, известный как динамическое распределение. В свое время этот подход был использован фирмой IBM в операционной системе для мэйнфреймов в OS/MVT (мультипрограммирование с переменным числом задач – Multiprogramming with a Variable number of Tasks). Позже этот же подход к распределению памяти был использован в ОС ЕС ЭВМ [5].

При динамическом распределении образуется переменное количество разделов переменной длины. При размещении процесса в основной памяти для него выделяется строго необходимое количество памяти. В качестве примера рассмотрим использование 64 Мбайт основной памяти (рис. 1.7). Изначально вся память пуста, за исключением области, используемой ОС. Первые три процесса загружаются в память начиная с адреса, где заканчивается ОС, и используют столько памяти, сколько требуется данному процессу. После этого в конце ОП остается свободный участок памяти, слишком малый для размещения четвертого процесса. В некоторый момент времени все процессы в памяти оказываются неактивными, и операционная система выгружает второй процесс, после чего остается достаточно памяти для загрузки нового, четвертого процесса.

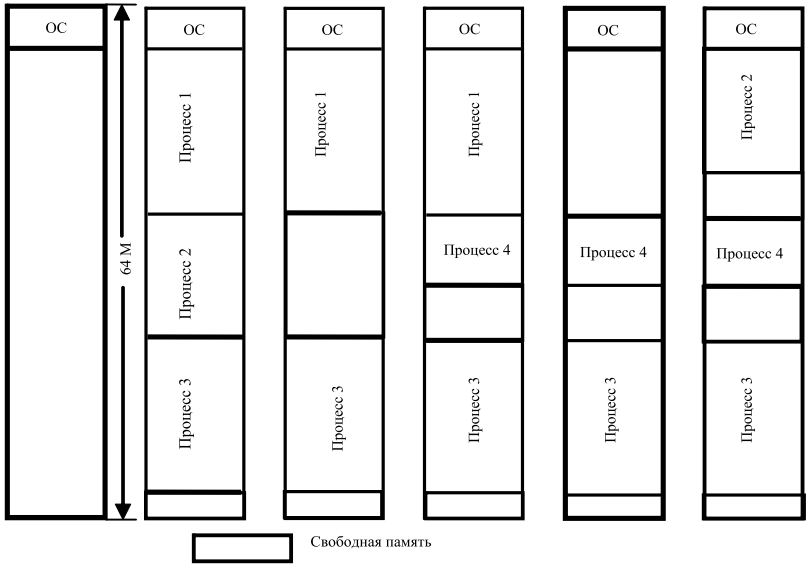

Рис. 1.7. Вариант использования памяти

Поскольку процесс 4 меньше процесса 2, появляется еще свободный участок памяти. После того как в некоторый момент времени все процессы оказались неактивными, но стал готовым к работе процесс 2, свободного места в памяти для него не находится, а ОС вынуждена выгрузить процесс 1, чтобы освободить необходимое место и разместить процесс 2 в ОП. Как показывает данный пример, этот метод эффективен на начальных этапах работы, но затем вспомогательные работы могут занимать значительную часть процессорного времени. В конечном счете он приводит к наличию множества мелких свободных участков памяти, в которых нет возможности разместить какой-либо новый процесс. Это явление называется внешней фрагментацией (external fragmentation), и отражает тот факт, что сильно фрагментированной становится память, внешняя по отношению ко всем разделам.

Один из методов преодоления внешней фрагментации – уплотнение (compaction) процессов в ОП. Осуществляется это перемещением всех занятых участков так, чтобы вся свободная память образовала единую свободную область. В дополнение к функциям, которые ОС выполняет при распределении памяти динамическими разделами, в данном случае она должна еще время от времени копировать содер-

жимое разделов из одного места в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется уплотнением, или сжатием.

Перечислим функции операционной системы по управлению памятью в этом случае:

1. Перемещение всех занятых участков в сторону старших или младших адресов при каждом завершении процесса или для вновь создаваемого процесса в случае отсутствия раздела достаточного размера.

2. Коррекция таблиц свободных и занятых областей.

3. Изменение адресов команд и данных, к которым обращаются процессы при их перемещении в памяти за счет использования относительной адресации.

4. Аппаратная поддержка процесса динамического преобразования относительных адресов в абсолютные адреса основной памяти.

5. Защита памяти, выделяемой процессу, от взаимного влияния других процессов.

Уплотнение может выполняться либо при каждом завершении процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, а во втором – реже выполняется процедура сжатия.

Так как программа перемещается по оперативной памяти в ходе своего выполнения, то в данном случае невозможно выполнить настройку адресов с помощью перемещающего загрузчика. Здесь более подходящим оказывается динамическое преобразование адресов. Достоинствами распределения памяти перемещаемыми разделами являются эффективное использование оперативной памяти, исключение внутренней и внешней фрагментации; недостатком – дополнительные расходы по обслуживанию вспомогательных процессов ОС.

При использовании фиксированной схемы распределения процесс всегда будет назначаться одному и тому же разделу памяти после его выгрузки и последующей загрузки в память. Это позволяет использовать простейший загрузчик, замещающий при загрузке процесса все относительные ссылки абсолютными адресами памяти, определенными на основе базового адреса загруженного процесса.

Ситуация усложняется, если размеры разделов равны (или неравны) и существует единая очередь процессов; каждый из них по ходу работы может занимать разные разделы. Такая же ситуация возможна и при динамическом распределении. В этих случаях расположение команд и

данных, к которым обращается процесс, не является фиксированным и изменяется всякий раз при выгрузке, загрузке или перемещении процесса. Для решения этой проблемы в программах используются относительные адреса. Это означает, что все ссылки на память в загружаемом процессе даются относительно начала этой программы. Таким образом, для корректной работы программы требуется аппаратный механизм, который бы транслировал относительные адреса в физические в процессе выполнения команды, которая обращается к памяти.

Обычно используемый способ трансляции показан на рис. 1.8. Когда процесс переходит в состояние выполнения, в специальный регистр процесса, называемый базовым, загружается начальный адрес процесса в основной памяти. Кроме того, используется «граничный» (bounds) регистр, в котором содержится адрес последней ячейки программы. Эти значения заносятся в регистры при загрузке программы в основную память. При выполнении процесса относительные адреса в командах обрабатываются процессором в два этапа. Сначала к относительному адресу прибавляется значение базового регистра для получения абсолютного адреса. Затем полученный абсолютный адрес сравнивается со значением в граничном регистре. Если полученный абсолютный адрес принадлежит данному процессу, команда может быть выполнена. В противном случае генерируется соответствующее данной ошибке прерывание.

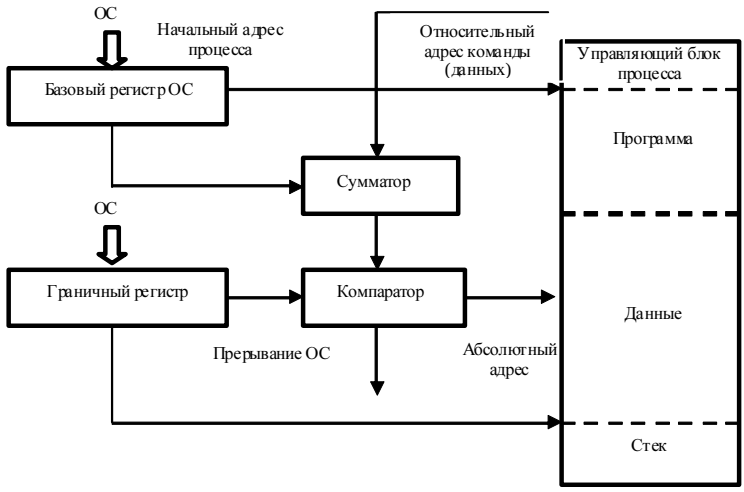

Рис. 1.8. Преобразование адресов

1.4. Страничная организация виртуальной памяти

Большинство систем виртуальной памяти используют технику, называемую страничной организацией памяти [1, 2]. Любой процесс, реализуемый в компьютере, может обратиться к множеству адресов в памяти. Адреса могут формироваться с использованием индексации, базовых регистров, сегментных регистров и другими путями. Эти программно формируемые адреса, называемые виртуальными адресами, формируют виртуальное адресное пространство. На компьютерах без виртуальной памяти виртуальные адреса подаются непосредственно на шину памяти и вызывают для чтения или записи слово в физической памяти с тем же самым адресом.

Когда используется виртуальная память, виртуальные адреса не передаются напрямую шиной памяти. Вместо этого они передаются *диспетчеру памяти* (MMU – Memory Management Unit), который отображает виртуальные адреса на физические адреса памяти, как показано на рис. 1.9. Здесь диспетчер памяти показан как часть микросхемы процессора, что обычно и бывает чаще всего. Но логически он мог бы быть отдельной микросхемой, как было в недавнем прошлом.

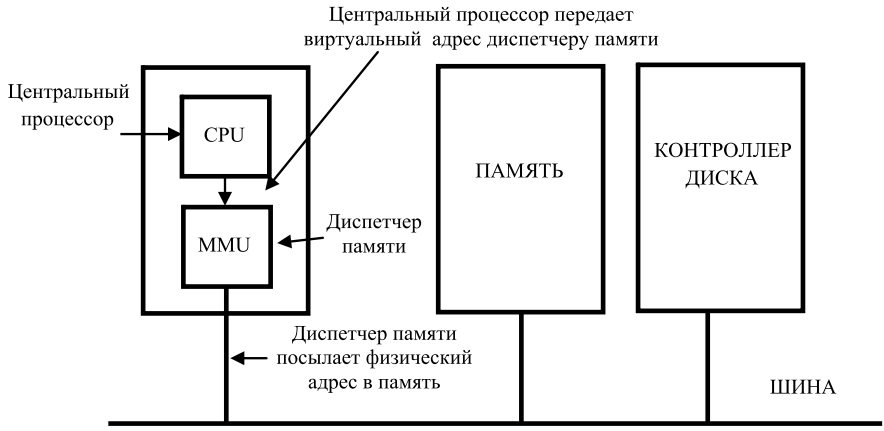

Рис. 1.9. Диспетчер памяти

Все имеющееся в настоящее время множество реализаций виртуальной памяти различается в основном способом структуризации виртуального адресного пространства.

Сам термин «виртуальная память» ассоциируется с системами, использующими страничную организацию. Впервые сообщение о виртуальной памяти на основе страничной организации появилось в 1962 г. в работе Kilburn I и др. «One-Level Storage System», и вскоре после этого виртуальная память стала широко использоваться в коммерческих системах.

В настоящее время выделяют три метода реализации виртуальной памяти.

1. ***Страничная виртуальная память*** организует перемещение данных между основной памятью и диском страницами – частями виртуального адресного пространства фиксированного и сравнительно небольшого размера.

2. ***Сегментная виртуальная память*** предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера с учетом смыслового значения данных.

3. ***Сегментно-страничная виртуальная память*** использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных является страница.

Для временного хранения сегментов и страниц на диске отводится специальная область либо специальный файл (страничный файл, или файл подкачки – paging file). Текущий размер страничного файла является важным параметром, оказывающим влияние на возможности операционной системы: чем больше страничный файл, тем больше приложений может одновременно выполнять ОС (при фиксированном размере оперативной памяти). Однако необходимо понимать, что увеличение числа одновременно работающих приложений в результате увеличения размера страничного файла замедляет их работу, так как значительная часть времени при этом тратится на перемещение данных на диск и обратно.

Размер страничного файла в современных ОС является настраиваемым параметром, который выбирается администратором системы для достижения компромисса между уровнем программирования и быстродействием системы.

При страничной организации виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами

(Virtual pages). В общем случае размер виртуального адресного пространства не кратен размеру страницы, поэтому последняя страница дополняется фиксированной областью. Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 1024, 2408, 4096 байт и т.д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в операционную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно находятся на смежных физических страницах. Для каждого процесса ОС создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса (рис. 1.10).

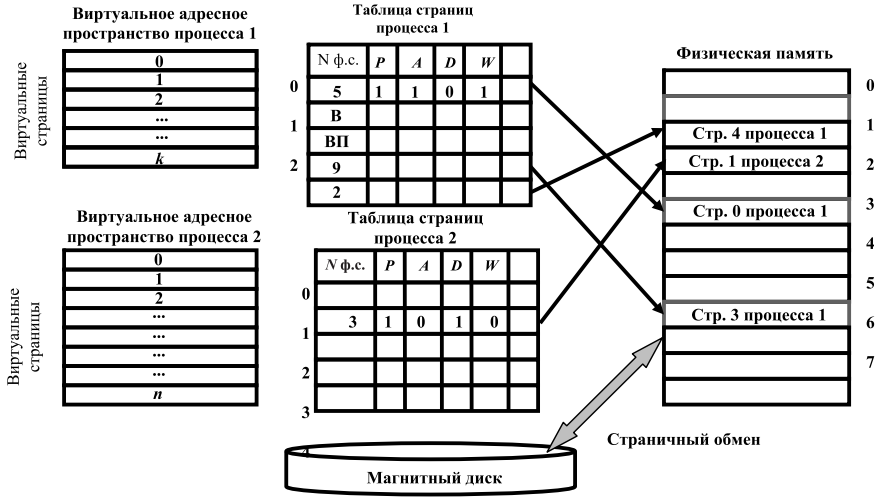

Рис. 1.10. Таблицы страниц виртуальной памяти

Запись таблицы (дескриптор страницы) включает следующую информацию:

• номер физической страницы (*N* ф.с.), в которую загружена данная виртуальная страница;

• признак присутствия *P*, устанавливаемый равным единице, если данная страница находится в оперативной памяти;

• признак модификации страницы *D*, который устанавливается равным единице всякий раз, когда производится запись по адресу, относящемуся к данной странице;

• признак обращения *A* к странице, называемый также битом доступа, который устанавливается равным единице при каждом обращении по адресу, относящемуся к данной странице;

• другие управляющие биты (*W*), служащие, например, для целей защиты или совместного использования памяти на уровне страниц.

Перечисленные признаки в большинстве моделей процессов устанавливаются аппаратно схемами процессора при выполнении операций с памятью. Информация из таблицы страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический. Сами таблицы страниц, так же как и описываемые ими страницы, размещаются в оперативной памяти.

Поскольку процесс может использовать большой объем виртуальной памяти (например, в Windows 2000 он равен 2^{32} = 4 Гбайт) при использовании страницы объемом 4 Кбайт (2^{12}) потребуется 2^{20} записей в таблице страниц для каждого процесса. Понятно, что выделять такое количество оперативной памяти под таблицы страниц нецелесообразно. Для преодоления этой проблемы большинство схем виртуальной памяти хранит таблицы страниц не в реальной, а в виртуальной памяти. Это означает, что сами таблицы страниц становятся объектами страничной организации. При работе процесса как минимум часть его таблицы страниц должна располагаться в основной памяти, в том числе запись о странице, выполняющейся в настоящий момент. Адрес таблицы страниц включается в контекст процесса. При активизации очередного процесса ОС загружает адрес его таблицы страниц в специальный регистр.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит страничное прерывание.

Выполняющий процесс переводится в состояние ожидания, и активизируется процесс из очереди процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается ее загрузить в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно. Если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную память, обнуляется ее бит присутствия и анализируется ее признак модификации. Если удаляемая страница за время последнего требования в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой виртуальной страницы, и никакой записи на диск не производится. Физическая страница объявляется свободной. Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, чтобы невозможно было использовать содержимое выгруженной страницы. Для хранения информации о положении вытесненной страницы в страничном файле ОС может использовать специальные поля таблицы страниц.

Виртуальный адрес при страничном распределении может быть представлен в виде пары (P, S_v), где P – номер виртуальной страницы процесса (нумерация страниц начинается с нуля), а S_v – смещение в пределах виртуальной страницы (рис. 1.11). Физический адрес также может быть представлен в виде пары (N, S_f), где N – номер физической страницы, а S_f – смещение а пределах физической страницы. Задача подсистемы виртуальной памяти заключается в отображении пары значений (P, S_v) в пару (N, S_f).

Чтобы понять механизм реализации этого отображения, следует остановиться на двух базисных свойствах страничной организации. Первое свойство заключается в том, что объем страницы, как виртуальной, так и физической, выбирается равным степени двойки – 2^k (k = 8 и выше). Отсюда следует, что смещение S_v и S_f может быть получено отделением k младших разделов в двоичной записи виртуального и соответственно физического адреса страницы. При этом оставшиеся старшие разделы адреса представляют собой двоичную

запись номера виртуальной и соответственно физической страницы. Дополнив эти номера нулями, можно получить начальный адрес виртуальной и физической страниц.

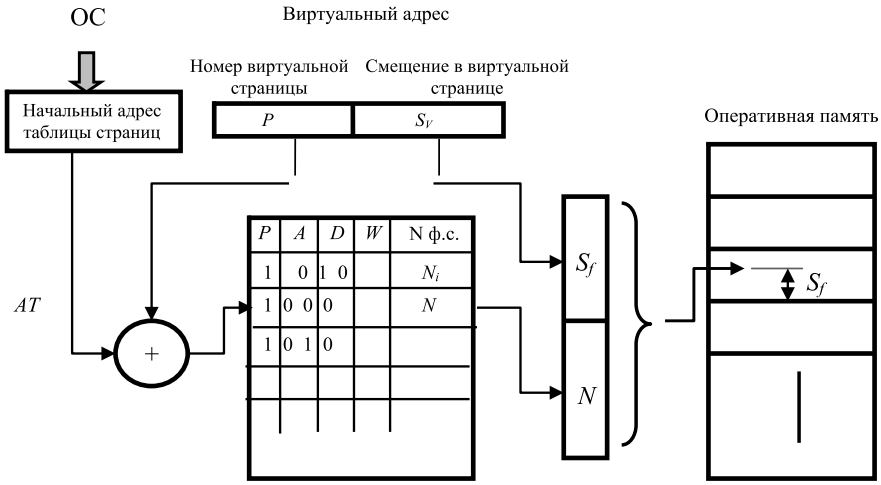

Рис. 1.11. Преобразование виртуального адреса

Второе свойство – линейность адресного пространства виртуальной и физической страницы – приводит к тому, что $S_f = S_v$. Отсюда следует простая схема преобразования виртуального адреса в физический.

При обращении к памяти по некоторому виртуальному адресу (P, S_v) аппаратные схемы процессора выполняют следующие действия:

1. Из специального регистра процессора извлекается начальный адрес AT таблицы страниц активного процесса. С помощью сумматора Σ по значениям AT, P, L (длина отдельной записи в таблице страниц) определяется адрес нужной записи в таблице страниц:

$$A = AT + (P \cdot L).$$

2. Считывается номер соответствующей физической страницы – N.

3. К номеру физической страницы присоединяется смещение S_v.

В итоге полученный физический адрес оперативной памяти представляется в виде пары значений (N, S_f).

Процедура преобразования виртуального адреса в физический без принятия специальных мер (кэширование активных страниц) зани-

мает один цикл оперативной памяти, который затрачивается на считывание номера физической страницы из таблицы страниц. Поэтому любое обращение к ОП будет занимать два цикла вместо одного при работе без виртуальной памяти.

Другим фактором, влияющим на производительность систем, являются затраты времени на обработку страничных прерываний. При неправильно выбранной стратегии замещения страниц может возникнуть ситуация, когда система тратит большую часть времени впустую, на подкачку страниц из оперативной памяти на диск и обратно.

1.5. Оптимизация функционирования страничной виртуальной памяти

В настоящее время известно несколько методов повышения эффективности функционирования страничной виртуальной памяти. К ним относятся [2]:

1. Более сложная структуризация виртуального адресного пространства, например двухуровневая (типичная для 32-битовой адресации).

2. Использование специального высокоскоростного кэша для хранения части записей таблицы страниц, который обычно называют буфером быстрого преобразования адреса, или буфером поиска трансляции (translation lookaside buffer – TLB).

3. Выбор оптимального размера страницы виртуальной памяти.

4. Эффективное управление страничным обменом.

Остановимся на возможностях реализации этих методов.

Рассмотрим вариант двухуровневой страничной организации. При такой схеме имеется каталог таблиц страниц, в котором каждая запись указывает на таблицу страниц. Таким образом, если размер каталога – *X*, а максимальный размер таблицы страниц – *Y*, то процесс может состоять максимум из *X* и *Y* страниц. Обычно максимальный размер таблицы страниц определяется условием ее размещения в одной странице (такой подход используется в процессоре Pentium).

На рис. 1.12 приведен пример двухуровневой схемы, типичной для 32-битовой адресации. Подобная схема позволяет существенно сохранить размер пользовательской таблицы страниц, размещаемой в основной памяти. В данном случае виртуальное адресное пространство пользовательского процесса может составлять 2^{32} = 4 Гбайт. При объеме страницы 2^{12} = 4 Кбайт в этом пространстве размещается $2^{32}/2^{12} = 2^{20}$ страниц. Таким образом, пользовательская таблица стра-

ниц будет иметь 2^{20} 4-байтных записей общим объемом 4 Мбайт. Разместить такие таблицы для нескольких процессов в ОП нереально. При двухуровневой схеме это и не требуется. В основной памяти постоянно находится корневая таблица, содержащая 1024 записей, указывающих на начальный адрес пользовательской таблицы страниц (ее объем, как указано выше, 4 Мбайт). Указание на начальный адрес корневой таблицы (активного процесса) заносится в регистр процессора. Первые 10 бит виртуального адреса используются для индексации в корневой таблице при поиске записей о странице пользовательской таблицы.

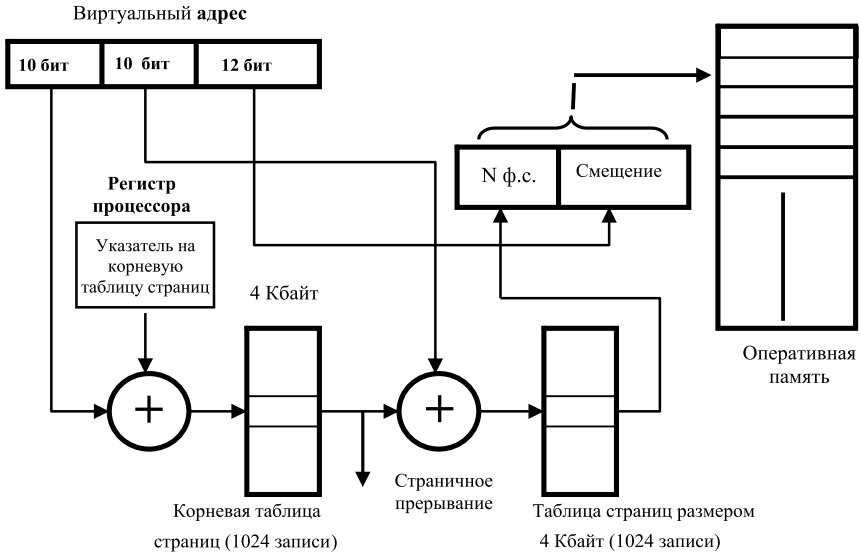

Рис. 1.12. Двухуровневая схема таблиц страниц

Если страница находится в ОП, то следующие 12 бит виртуального адреса используются для задания смещения в физической странице ОП. В противном случае генерируется страничное прерывание, но уже из-за отсутствия нужной страницы процесса в ОП.

Таким образом, двухуровневая схема, сокращая объем памяти для хранения таблицы страниц, в общем случае замедляет преобразование виртуального адреса из-за большего числа возможных страничных прерываний (даже если нет страничного прерывания, требуется три цикла ОП вместо двух при одноуровневой страничной организации).