ВИРТУАЛЬНАЯ ПАМЯТЬ

Трудность изучения этой темы заключается в том, что управление памятью представляет собой сложную связь между аппаратным обеспечением и программным обеспечением операционной системы.

ТЕРМИНОЛОГИЯ, СВЯЗАННАЯ С ВИРТУАЛЬНОЙ ПАМЯТЬЮ

Виртуальная память - Схема распределения памяти, в которой вторичная память может адресоваться так, как если бы она была частью основной памяти.

Адреса, которые программа может использовать для обращения к памяти, отличаются от адресов, используемых системой управления памятью для идентификации физической памяти, и генерируемые программой адреса автоматически транслируются в соответствующие машинные адреса.

Размер виртуальной памяти ограничен схемой адресации компьютерной системы и количеством доступной вторичной памяти, но не фактическим количеством ячеек основной памяти

Виртуальный адрес - Адрес, присвоенный местоположению в виртуальной памяти, который позволяет обращаться к данному местоположению так, как если бы это была часть основной памяти

Виртуальное адресное пространство - Виртуальная память, назначенная процессу

Адресное пространство - Диапазон адресов памяти, доступный процессу

Реальный адрес - Адрес ячейки основной памяти

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ И УПРАВЛЯЮЩИЕ СТРУКТУРЫ

Сравнивая простую страничную организацию и сегментацию с одной стороны и фиксированное и динамическое распределение памяти - с другой, мы видим основание для фундаментального

прорыва в технологии управления памятью. Ключевыми для него являются следующие характеристики страничной организации и сегментации.

1. Все ссылки на память в рамках процесса представляют собой логические адреса, которые динамически транслируются в физические адреса во время выполнения. Это означает, что процесс может быть выгружен на диск и вновь загружен в основную память, так что в разные моменты времени выполнения он может находиться в разных местах основной памяти.

2. Процесс может быть разбит на ряд частей (страниц или сегментов), которые не обязательно должны располагаться в основной памяти единым непрерывным блоком. Это обеспечивается за счет динамической трансляции адресов и использования таблицы страниц или сегментов.

Если в системе выполняются указанные характеристики, то наличие всех страниц или сегментов процесса в основной памяти одновременно не является обязательным условием.

Если фрагмент (сегмент или страница), в котором находится следующая выбираемая команда, и фрагмент, в котором находится ячейка памяти, к которой обращается программа, располагаются в основной памяти, то как минимум некоторое время выполнение программы может продолжаться.

Рассмотрим, каким образом это может осуществляться. Пока что мы говорим об этой технологии в общем, так что используем термин блок для обозначения страницы или сегмента - в зависимости от того, имеем ли мы дело со страничной организацией или с сегментацией.

Предположим, что наступило время загрузки нового процесса в память. Операционная система начинает ее с размещения в памяти только одного или нескольких блоков, включая блок, содержащий начало программы.

Часть процесса, располагающаяся в некоторый момент времени в основной памяти, называется **резидентным множеством** (resident set) процесса. Во время выполнения процесса все происходит так, как если бы все ссылки были только на резидентное множество процесса. При помощи таблицы сегментов или страниц процессор всегда может определить, располагается ли блок, к которому требуется обращение, в основной памяти. Если процессор сталкивается с логическим адресом, который не находится в основной памяти, он генерирует прерывание, свидетельствующее об ошибке доступа к памяти.

Операционная система переводит прерванный процесс в заблокированное состояние и получает управление. Чтобы продолжить выполнение прерванного процесса, операционной системе необходимо загрузить в основную память блок, содержащий вызвавший проблемы логический адрес. Для этого операционная система использует запрос на чтение с диска (во время выполнения которого может продолжаться выполнение других процессов).

После того как необходимый блок загружен в основную память, выполняется прерывание ввода-вывода, передающее управление операционной системе, которая, в свою очередь, переводит заблокированный процесс в состояние готовности.

Естественно, возникает вопрос об эффективности использования такой технологии, когда выполнение процесса постоянно прерывается только из-за того, что в основной памяти размещены не все его блоки.

Следствия применения новой стратегии:

1. В основной памяти может поддерживаться большее количество процессов. Поскольку в основную память загружаются только некоторые из блоков каждого процесса, мы можем разместить в ней больше процессов. Это, в свою очередь, приводит к более эффективному использованию процессора, поскольку повышается вероятность наличия активных процессов в любой момент времени.

2. Процесс может быть больше, чем вся основная память. Преодолено одно из наиболее существенных ограничений в программировании. Обычно программист должен изначально рассчитывать, какое количество памяти потребуется его программе. Если разработанная программа окажется слишком большой, программист должен принять соответствующие меры по разделению программы на части, которые могли бы быть загружены в память в отдельности, с использованием той или иной оверлейной стратегии. В случае использования виртуальной памяти на основе страничной организации или сегментации эта функция передается операционной системе и аппаратному обеспечению. Программе (и программисту) при этом доступен огромный объем памяти, по сути, представляющий собой размер дискового пространства. Операционная система при необходимости автоматически загружает нужные блоки процесса в память.

Поскольку процесс выполняется только в основной памяти, эта память называется также ***реальной*** (real memory). Однако программист или пользователь имеет дело с потенциально гораздо большей памятью - выделенной на диске. Эта память известна как ***виртуальная*** (virtual memory). Виртуальная память обеспечивает очень эффективную многозадачность и облегчает работу пользователя, снимая жесткие ограничения относительно объема основной памяти.

Таблица 8.2. Характеристики страничной организации и сегментации

Простая страничная организация	Страничная организация с виртуальной памятью	Простая сегментация	Сегментация с виртуальной памятью
Основная память разделена на небольшие блоки фиксированного размера, именуемые кадрами		Основная память не разделена	
Программа разбита на страницы компилятором или системой управления памятью		Сегменты программы определены программистом при компиляции (решение о разбивке на сегменты принимается программистом)	
Внутренняя фрагментация в кадрах		Внутренняя фрагментация отсутствует	

Простая страничная организация	Страничная организация с виртуальной памятью	Простая сегментация	Сегментация с виртуальной памятью
Внешняя фрагментация отсутствует		Внешняя фрагментация	
Операционная система должна поддерживать таблицу страниц для каждого процесса, указывающую, какой кадр занят данной страницей процесса		Операционная система должна поддерживать таблицу сегментов для каждого процесса, указывающую адрес загрузки и длину каждого сегмента	
Операционная система должна поддерживать список свободных кадров		Операционная система должна поддерживать список свободных блоков памяти	
Для вычисления абсолютного адреса процессор использует номер страницы и смещение		Для вычисления абсолютного адреса процессор использует номер сегмента и смещение	

Простая страничная организация	Страничная организация с виртуальной памятью	Простая сегментация	Сегментация с виртуальной памятью
Для работы процесса все его страницы должны находиться в основной памяти (кроме случая использования оверлеев)	Для работы процесса не все его страницы должны находиться в основной памяти; они могут загружаться при необходимости Считывание страницы в основную память может требовать записи страницы на диск	Для работы процесса все его сегменты должны находиться в основной памяти (кроме случая использования оверлеев)	Для работы процесса не все его сегменты должны находиться в основной памяти; они могут загружаться при необходимости Считывание сегмента в основную память может требовать записи одного или нескольких сегментов на диск

Локальность и виртуальная память

Преимущества виртуальной памяти весьма привлекательны, но насколько практична данная схема?

Одно время на эту тему шли оживленные дебаты, однако опыт ее использования многими операционными системами продемонстрировал бесспорную жизнеспособность этой схемы - виртуальная память стала неотъемлемым компонентом большинства современных операционных систем.

Чтобы понять, в чем заключается ключевое свойство виртуальной памяти и почему она вызвала такие бурные дебаты, обратимся вновь к задачам операционной системы, связанным с виртуальной памятью.

Рассмотрим большой процесс, состоящий из длинного кода и ряда массивов данных.

В каждый небольшой промежуток времени выполнение программы сосредоточивается в малой части кода (например, в одной из подпрограмм), и обращается эта часть кода, как правило, только к одному или двум массивам данных.

В таком случае загружать в память все данные и код, в то время как перед приостановкой и выгрузкой процесса из памяти будут использоваться только небольшие их части, - расточительство.

Память будет использоваться гораздо эффективнее, если загружать в нее только необходимые части программы. Соответственно, в этом случае при обращении к данным (или к коду), которых в настоящий момент нет в основной памяти, происходит прерывание выполнения программы, которое говорит операционной системе о необходимости загрузки в основную память затребованной части программы.

Таким образом, в основной памяти в каждый момент времени находится только некоторая, как правило, небольшая, часть данного процесса, и следовательно, в основной памяти может быть одновременно размещено большее количество процессов.

К тому же при этом получается определенная экономия времени, так как неиспользуемые части процессов не приходится постоянно выгружать из памяти и загружать вновь.

Однако операционная система должна очень разумно работать с такой схемой управления памятью. В установившемся состоянии практически вся основная память занята фрагментами процессов, чтобы процессор и операционная система могли работать с как можно большим количеством процессов одновременно.

Следовательно, при загрузке в основную память некоторого блока другой блок должен быть выгружен оттуда.

Если из памяти выгрузить блок, который тут же потребуется вновь, операционная система будет заниматься постоянным перемещением одних и тех же блоков в основную память и на диск. Большое количество таких перебросок приводит к ситуации, известной как **снижение пропускной способности** (thrashing): процессор в основном занимается не выполнением процессов, а выгрузкой и загрузкой в основную память.

Задаче устранения этого нежелательного эффекта посвящен ряд исследовательских работ, выполнявшихся в 1970-х годах и приведших к появлению различных сложных, но эффективных алгоритмов.

По сути, они сводятся к попыткам определить на основании последних событий в системе, какие блоки памяти потребуются в ближайшем будущем.

Эти методы базируются на ***принципе локальности***, который гласит, что обращения к коду и данным в процессе имеют тенденцию к кластеризации.

Следовательно, логично допустить, что в течение некоторого небольшого времени для работы будет требоваться только небольшая часть процесса; кроме того, можно сделать правильные предположения о том, какие именно части процесса потребуются для работы в ближайшем будущем, и тем самым избежать снижения пропускной способности.

Принцип локальности позволяет надеяться на эффективность работы схемы виртуальной памяти. Для этого требуется, чтобы, ***во-первых***, имелась аппаратная поддержка страничной организации и/или сегментации, а ***во-вторых***, операционная система должна включать программное обеспечение для управления перемещением страниц и/или сегментов между вторичной и основной памятью.

Страничная организация

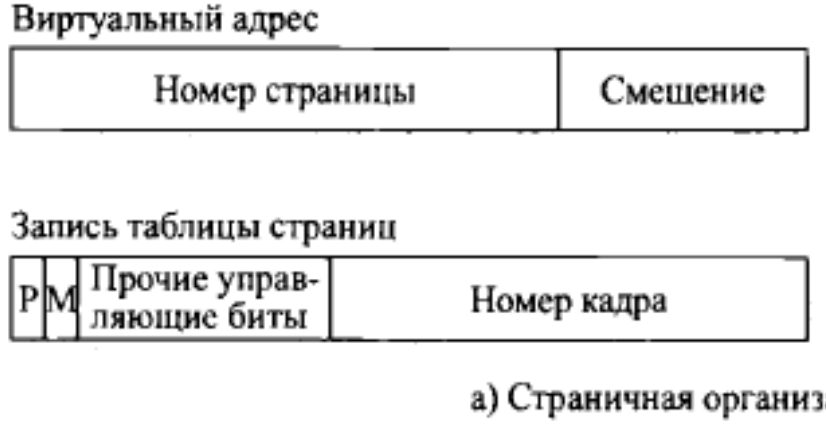
Термин виртуальная память обычно ассоциируется с системами, использующими страничную организацию, хотя используется и виртуальная память на основе сегментации.

В случае простой страничной организации основная память делится на ряд кадров одинакового размера. Каждый процесс делится на ряд страниц того же одинакового размера, что и размер кадров. Процесс загружается путем загрузки всех его страниц в доступные, хотя и не всегда смежные, кадры в памяти.

В случае страничной организации виртуальной памяти мы снова имеем страницы одинакового размера, равного размеру кадров, однако для выполнения не все страницы обязаны быть загружены в кадры основной памяти.

При рассмотрении простой страничной организации мы указывали, что каждый процесс имеет собственную таблицу страниц, которая создается при загрузке всех страниц процесса в основную память.

Каждая запись в таблице страниц содержит номер кадра соответствующей страницы в памяти. Такая же таблица страниц, связанная с каждым из процессов, требуется и при организации виртуальной памяти на базе страничной организации - однако в этом случае структура записей таблицы становится несколько более сложной (рис. 8.1, а).



а) Страничная организация

Поскольку в основной памяти могут находиться только некоторые из страниц процесса, в каждой записи таблицы должен

иметься бит P, указывающий на присутствие соответствующей страницы в основной памяти.

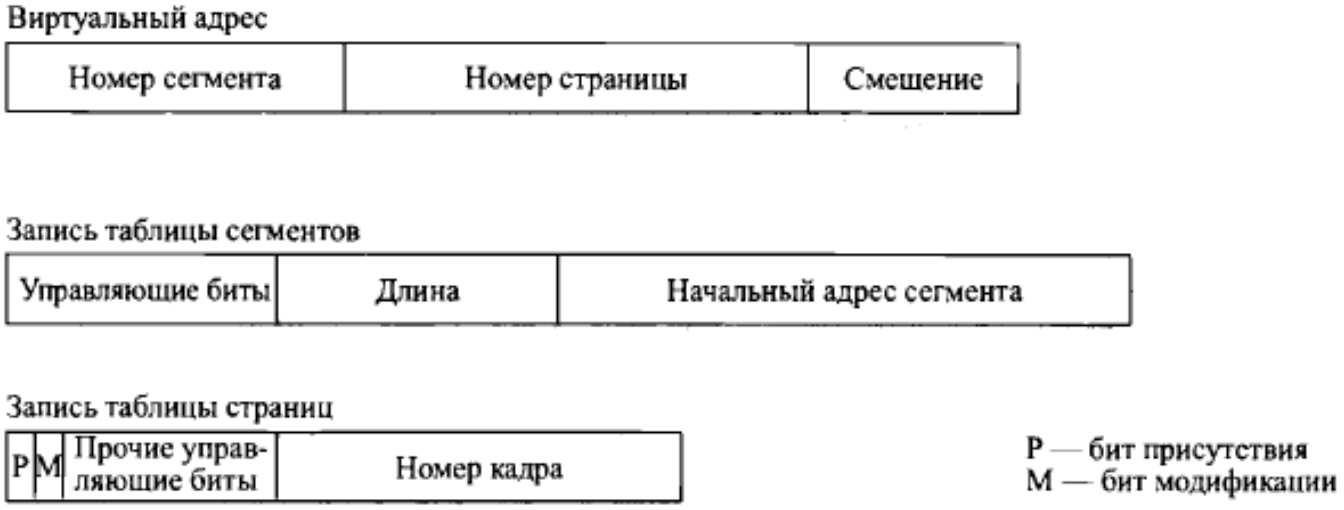
Если данная страница располагается в основной памяти, то в записи таблицы содержится номер ее кадра.

Другим управляющим битом в записи таблицы страниц является бит модификации, M, который указывает, было ли изменено содержимое данной страницы со времени последней загрузки в основную память.

Если изменений не было, то, когда наступит время замены страницы в занимаемом ею в данный момент кадре, записывать эту страницу на диск не понадобится, так как на диске уже есть ее точная копия.

В записи таблицы страниц могут быть и другие управляющие биты, например, служащие для целей защиты или совместного использования памяти на уровне страниц.

б) Сегментация



в) Комбинация страничной организации и сегментации

Рис. 8.1. Типичные форматы систем управления памятью

Структура таблицы страниц

Базовый механизм чтения слова из памяти включает в себя трансляцию виртуального, или логического, адреса, состоящего из номера страницы и смещения, в физический адрес, который представляет собой номер кадра и смещение, с использованием таблицы страниц.

Поскольку таблица страниц имеет переменную длину, зависящую от размера процесса, разместить ее в регистрах не представляется возможным, и таблица страниц должна располагаться в основной памяти.

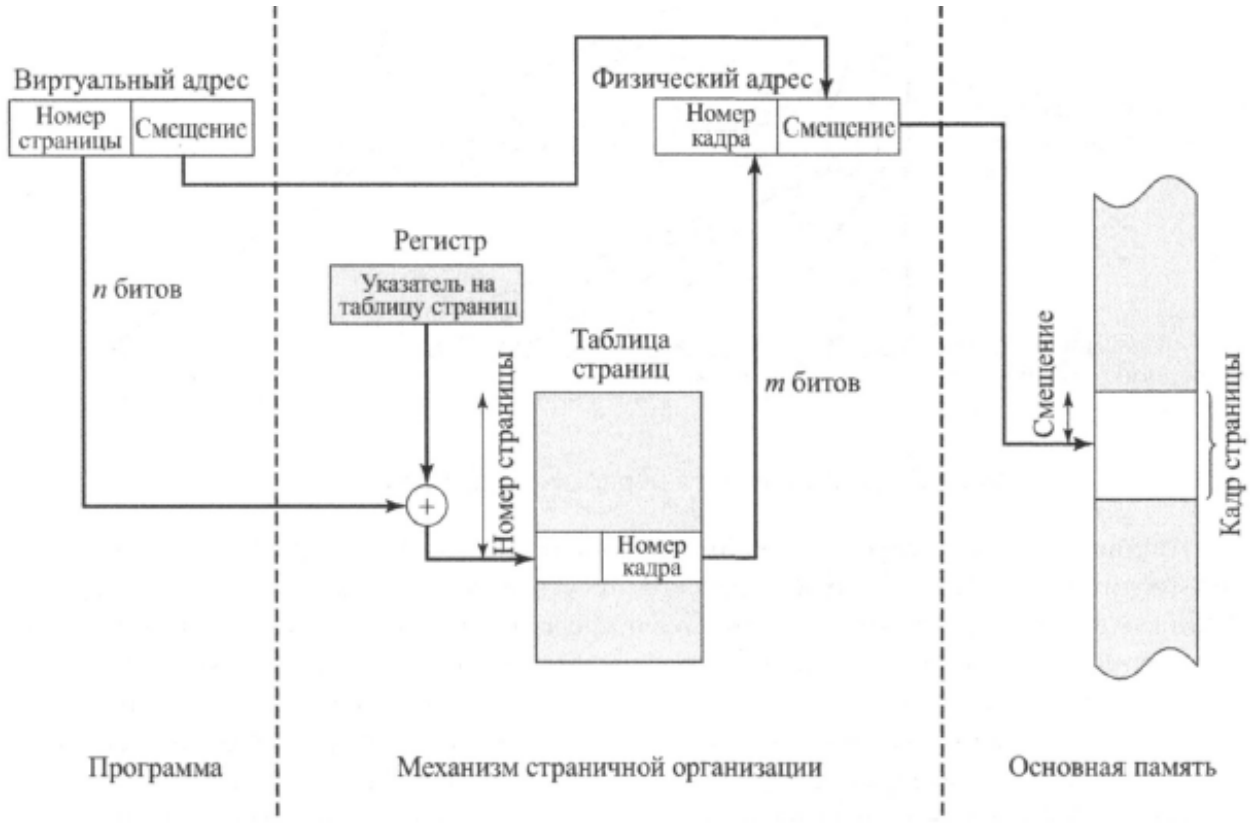


Рис. 8.2. Трансляция адреса в системе со страничной организацией

На рис. 8.2 показана аппаратная реализация этого механизма. При выполнении некоторого процесса стартовый адрес его таблицы страниц хранится в регистре, а номер страницы из виртуального адреса используется в качестве индекса элемента, в котором ищется соответствующий номер кадра.

Затем этот номер объединяется со смещением из виртуального адреса для получения реального физического адреса интересующей нас ячейки памяти.

Как правило, поле номера страницы длиннее, чем поле номера кадра ($n>m$). Это неравенство вытекает из того факта, что количество страниц в процессе может превышать количество кадров в основной памяти.

В большинстве систем для каждого процесса имеется одна таблица страниц. Однако каждый процесс может использовать большой объем виртуальной памяти. Так, например, в архитектуре VAX (Virtual Address Extension) каждый процесс может иметь до 2^{31} = 2 Гбайт виртуальной памяти. При использовании страниц размером

2^9 = 512 байт оказывается, что нам требуется до 2^{22} записей в таблице страниц для каждого процесса.

Понятно, что такое количество памяти, отводимое таблицам страниц, неприемлемо. Для преодоления этой проблемы большинство схем виртуальной памяти хранят таблицы страниц не в реальной, а в виртуальной памяти. Это означает, что сами таблицы страниц становятся объектами страничной организации, как и любые другие страницы.

При работе процесса как минимум часть его таблицы страниц должна располагаться в основной памяти, в том числе запись о странице, выполняющейся в настоящий момент.

Некоторые процессоры используют двухуровневую схему для больших таблиц страниц. При такой схеме имеется каталог таблиц страниц, в котором каждая запись указывает на таблицу страниц.

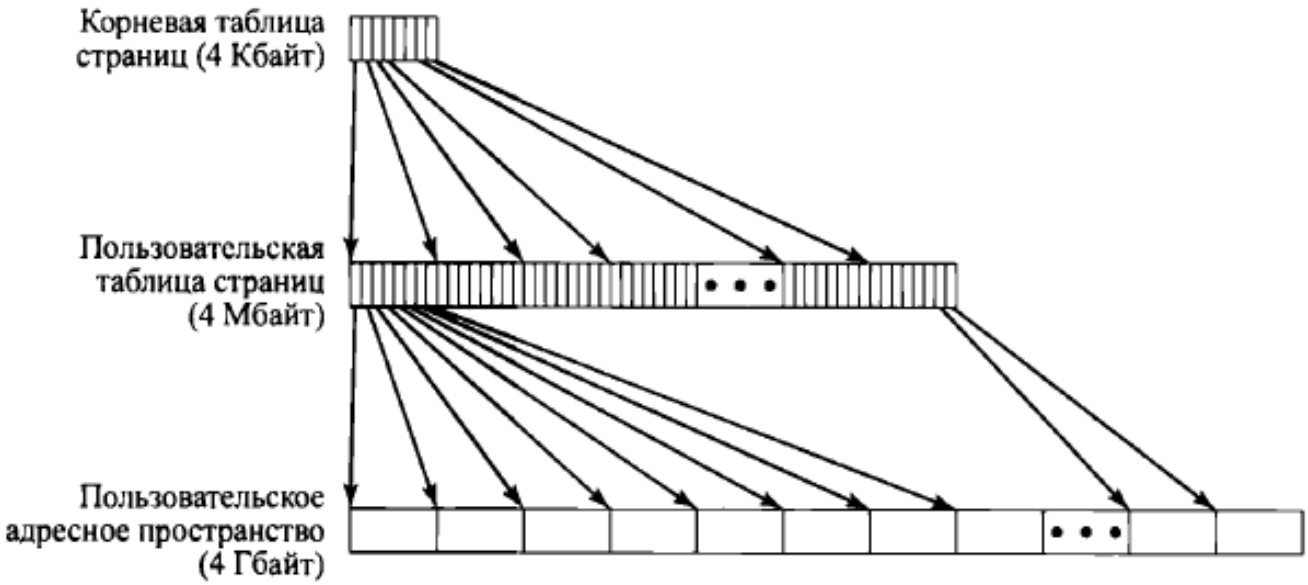


Рис. 8.3. Двухуровневая иерархическая таблица страниц

На рис. 8.4 показаны действия, выполняемые при трансляции адреса в двухуровневой системе. Корневая страница всегда остается в основной памяти. Первые 10 бит виртуального адреса используются для индекса корневой таблицы для поиска записи о странице пользовательской таблицы. Если нужная страница отсутствует в основной памяти, генерируется ошибка доступа к странице. Если же необходимая страница находится в основной памяти, то следующие 10 бит виртуального адреса используются как

индекс для поиска записи о странице, на которую ссылается исходный виртуальный адрес.

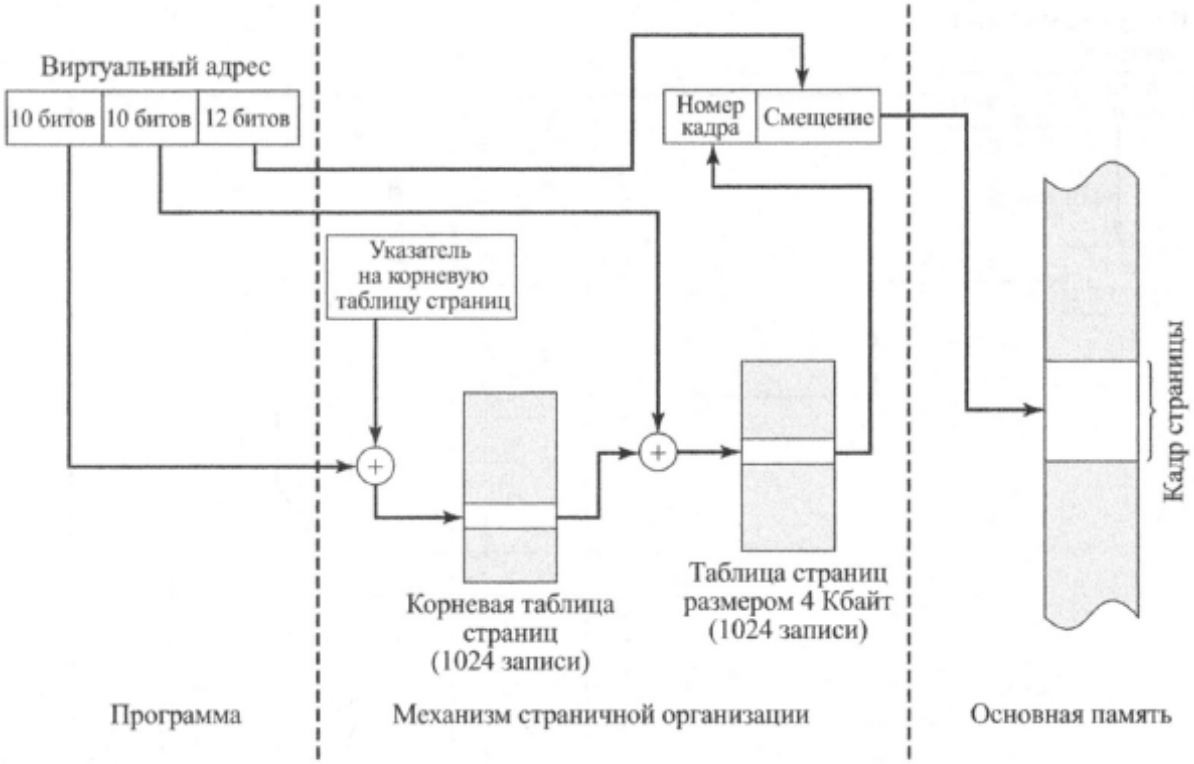


Рис. 8.4. Трансляция адреса в системе с двухуровневой страничной организацией

Инвертированная таблица страниц

Недостатком таблиц страниц рассматриваемого типа является то, что их размер пропорционален размеру виртуального адресного пространства.

Альтернативным подходом к использованию одно - или двухуровневых таблиц страниц является применение инвертированной таблицы страниц. Варианты этого подхода применялись на машинах Power PC, UltraSPARC.

При таком подходе часть виртуального адреса, представляющая собой номер страницы, отображается в хеш-таблицу с использованием простой функции хеширования. Хеш-таблица содержит указатель на инвертированную таблицу страниц. Каждому кадру страницы реальной памяти при этом соответствует одна запись в инвертированной таблице страниц.

Таким образом, для хранения таблиц требуется фиксированная часть основной памяти, независимо от размера и количества процессов и поддерживаемых виртуальных страниц.

Поскольку на одну и ту же запись хеш-таблицы могут отображаться несколько виртуальных адресов, для обработки переполнения используется технология цепочек (которые на практике обычно достаточно коротки - как правило, от одной до двух записей).

Структура таблицы страниц называется инвертированной, потому что она индексирует записи страницы таблицы номерами кадров, а не номерами виртуальных страниц.

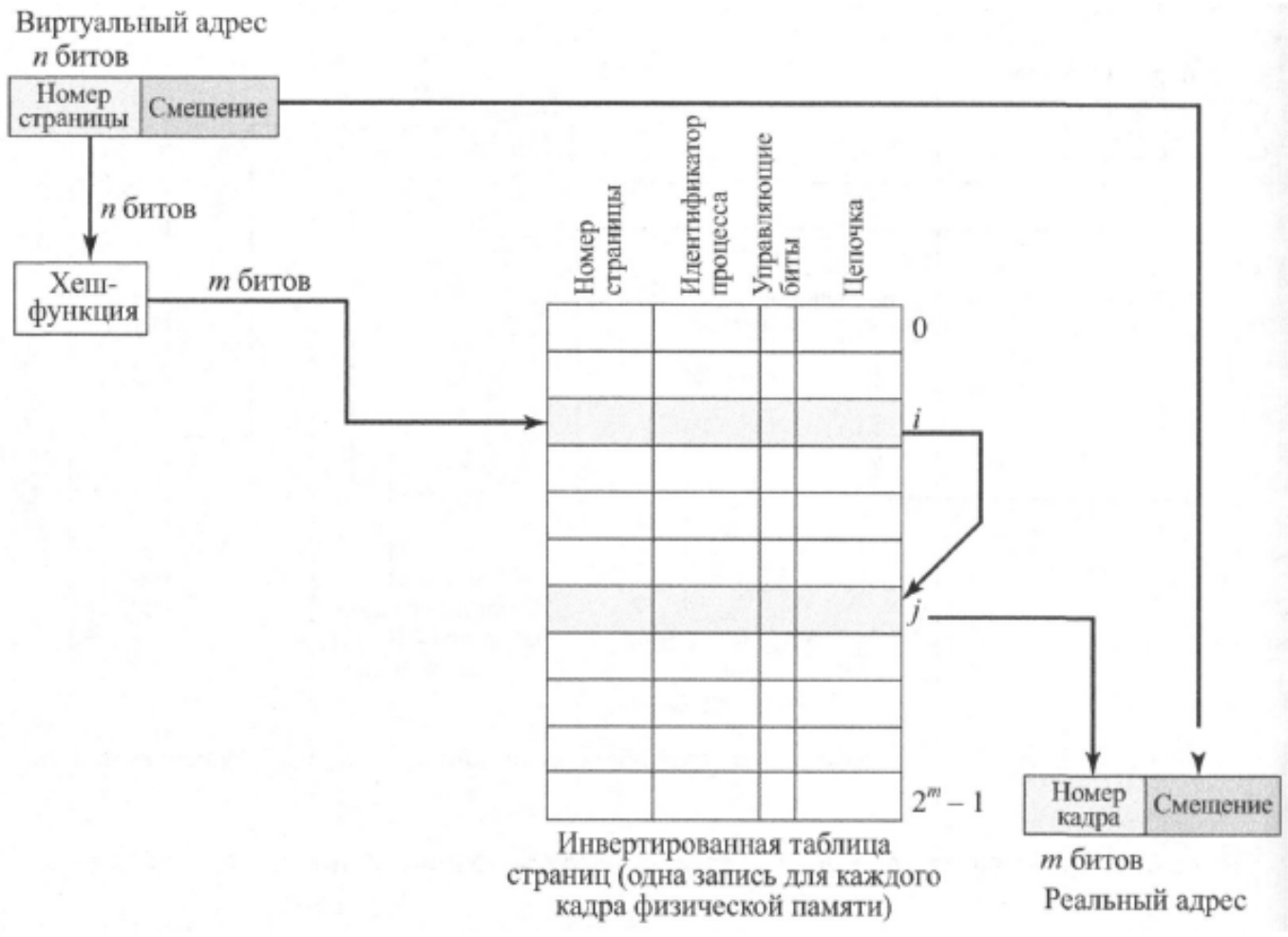


Рис. 8.5. Структура инвертированной таблицы страниц

Для физической памяти размером 2^m кадров инвертированная таблица страниц содержит 2^m записей, так что i-я запись относится к кадру i. Каждая запись в таблице страниц включает следующую информацию.

- ✓ **Номер страницы.** Часть виртуального адреса, относящаяся к номеру страницы.
- ✓ **Идентификатор процесса.** Процесс, владеющий этой страницей. Комбинация номера страницы и идентификатора процесса идентифицирует страницу в виртуальном адресном пространстве определенного процесса.
- ✓ **Управляющие биты.** Это поле включает флаги, такие как корректность, модифицированность и иные, а также информацию о защите и блокировках.
- ✓ **Указатель цепочки.** Это поле нулевое (вероятно, указывается отдельным битом), если для этой записи нет других связанных цепочкой записей. В противном случае поле содержит значение индекса (число от 0 до 2^m -1) следующей записи в цепочке.

В этом примере виртуальный адрес включает n-битный номер страницы, где n > m. Хеш-функция отображает n-битный номер в m - битное число, которое используется в качестве индекса инвертированной таблицы страниц.

Буфер быстрой переадресации

Каждый виртуальный адрес может привести к обращению к двум физическим адресам: к одному - для выборки соответствующей записи из таблицы страниц и еще к одному - для обращения к адресуемым данным.

Таким образом, простая схема виртуальной памяти, по сути, удваивает время обращения к памяти. Для преодоления этой проблемы большинство реально использующихся схем виртуальной памяти использует специальный высокоскоростной кеш для записей таблицы страниц, который обычно называют **буфером быстрого преобразования адреса** или просто буфером быстрой переадресации (**translation lookaside buffer - TLB**).

Этот кеш функционирует так же, как и обычный кещ памяти, и содержит те записи таблицы страниц, которые использовались

последними. Организация аппаратной поддержки использования TLB показана на рис. 8.6.

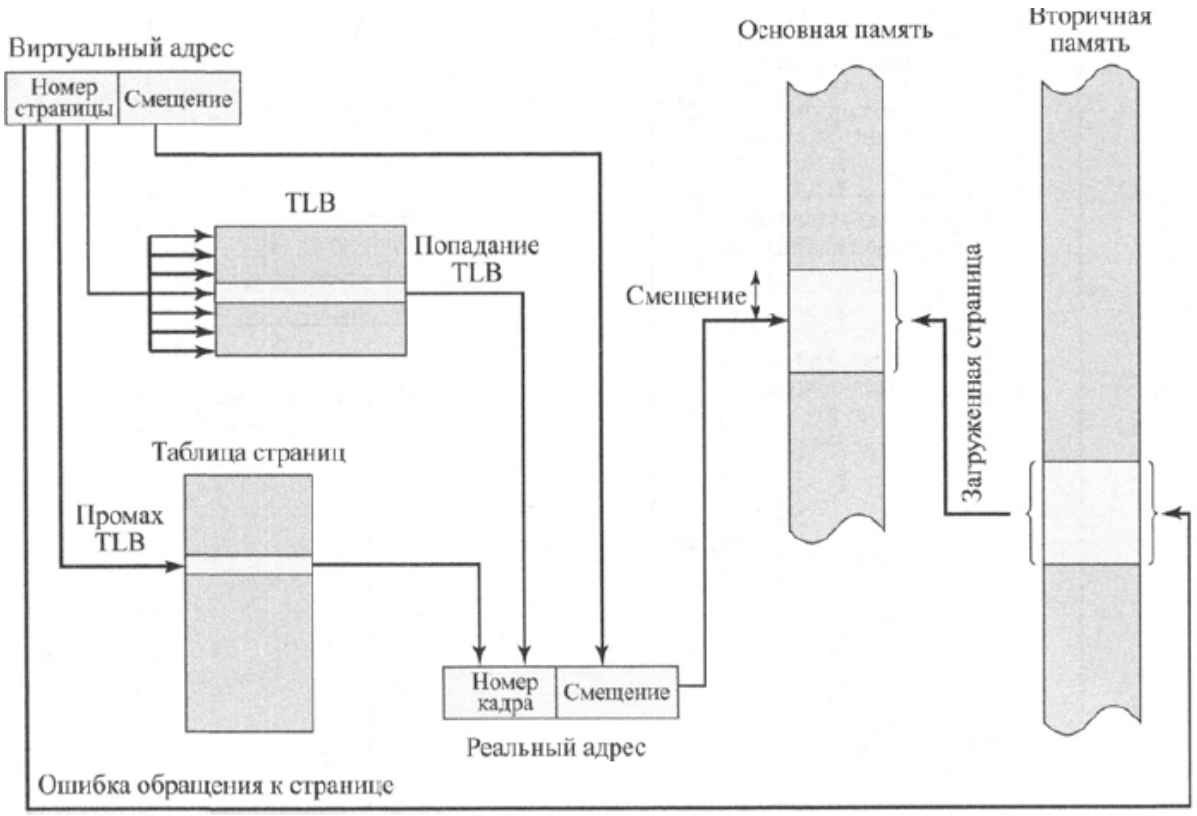


Рис. 8.6. Использование TLB

Получив виртуальный адрес, процессор сначала просматривает TLB. Если требуемая запись найдена (**попадание**), процессор получает номер кадра и формирует реальный адрес.

Если запись в TLB не найдена **(промах)**, то процессор использует номер страницы в качестве индекса для таблицы страниц процесса и просматривает соответствующую запись. Если бит присутствия в ней установлен, значит, искомая страница находится в основной памяти, и процессор просто получает номер кадра из записи таблицы страниц и формирует реальный адрес, одновременно внося использованную запись таблицы страниц в TLB.

И наконец, если бит присутствия не установлен, значит, искомой страницы в основной памяти нет, и процессор генерирует ошибку обращения к странице. В этот момент подключается операционная система, которая загружает требуемую страницу в основную память и обновляет таблицу страниц.

На рис. 8. 7 приведена диаграмма использования TLB. На ней показано, что если требующаяся страница отсутствует в основной памяти, то прерывание ошибки обращения к странице вызывает соответствующую программу обработки.

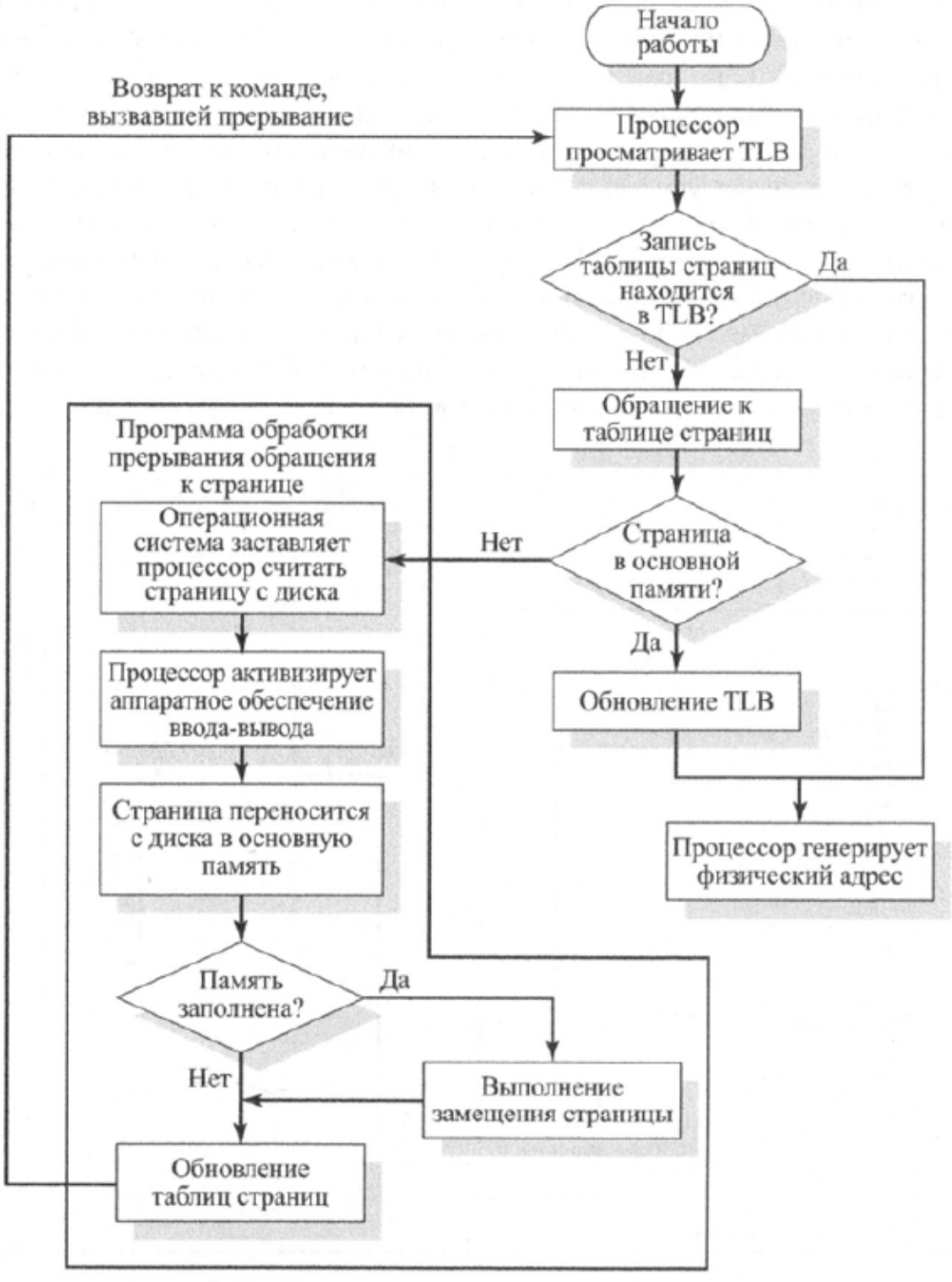


Рис. 8.7. Диаграмма трансляции адреса с использованием TLB

Для упрощения диаграммы в ней не отражен тот факт, что в процессе выполнения операций ввода-вывода операционная система

может параллельно с медленными операциями дискового ввода-вывода выполнять другой процесс.

Исходя из принципа локальности большинство обращений к виртуальной памяти будут сосредоточены в недавно использованных страницах, и соответствующие записи будут находиться в кеше, так что с помощью TLB существенно повышается эффективность работы виртуальной памяти. В реальной организации TLB имеется ряд дополнительных деталей.

Так, поскольку TLB содержит только некоторые из записей таблицы страниц, индексация записей в TLB на основе номера страницы не представляется возможной; вместо этого каждая запись TLB должна наряду с полной информацией из записи таблицы страниц включать номер страницы.

Процессор аппаратно способен опрашивать ряд записей TLB одновременно для определения того, какая из них соответствует заданному номеру страницы. Такая методика известна как ***ассоциативное отображение*** (associative mapping), в отличие от прямого отображения, или индексирования, применяемого для поиска в таблице страниц,

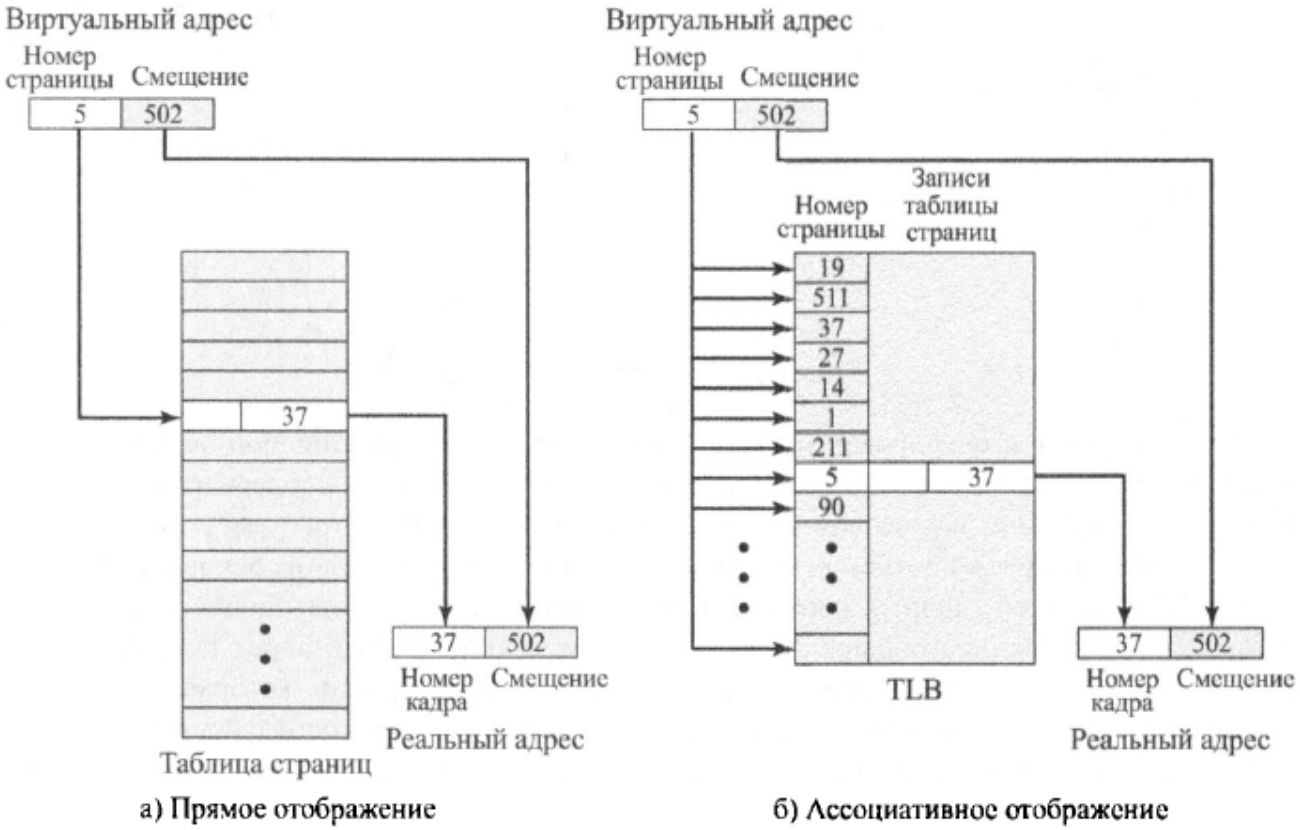


Рис. 8.8. Прямой и ассоциативный поиск записи таблицы страниц

Дизайн TLB должен также предусматривать способ организации записей в кеше и принятия решения о том, какая из старых записей должна быть удалена при внесении в кеш новой записи.

Механизм виртуальной памяти должен взаимодействовать с кешем (кешем основной памяти, не TLB). Это взаимодействие продемонстрировано на рис. 8.9.

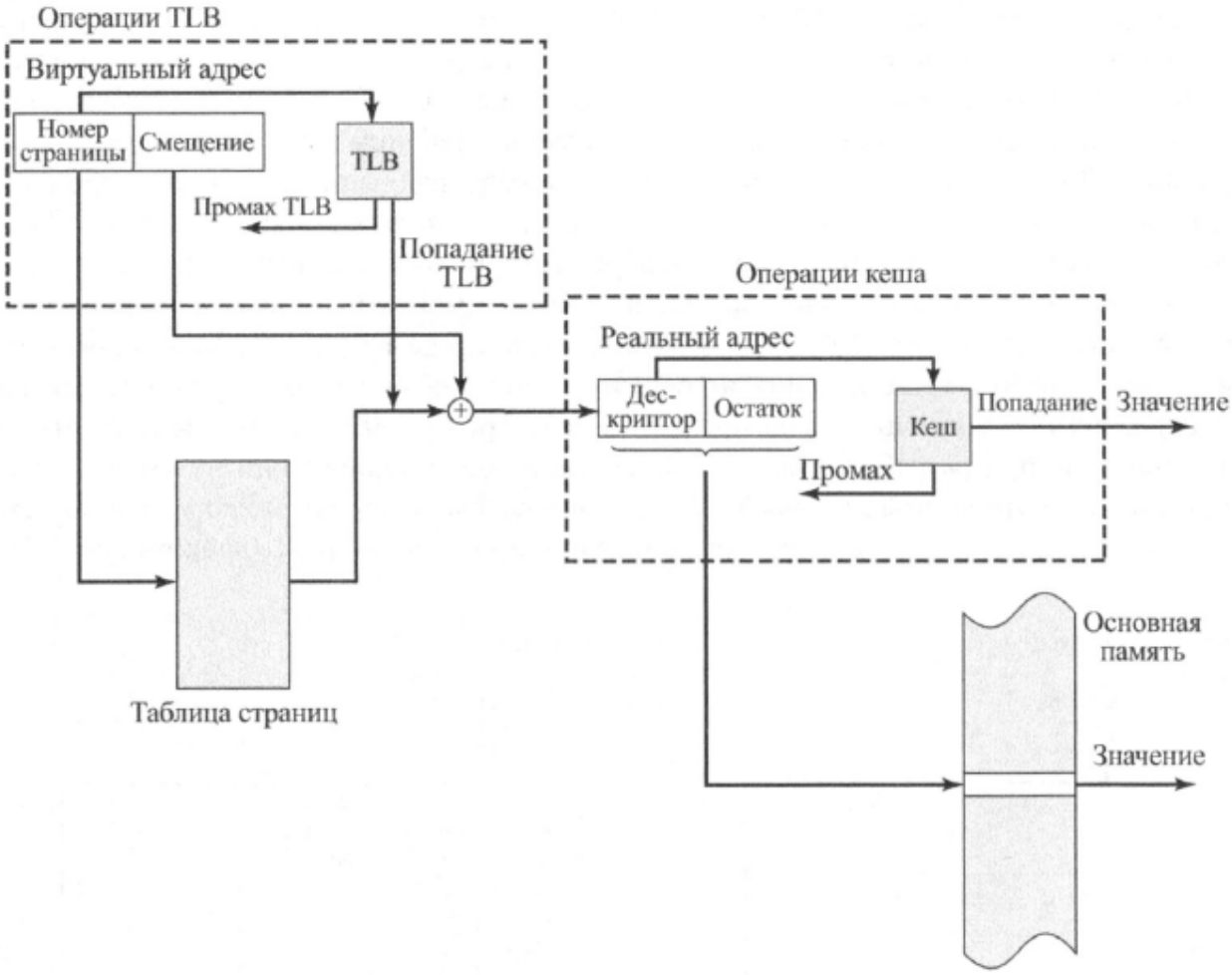


Рис. 8.9. Работа кеша и TLB

Виртуальный адрес, вообще говоря, представляет собой пару "номер страницы - смещение". Сначала происходит обращение к TLB для выяснения, имеется ли в нем соответствующая запись таблицы страниц. При положительном результате путем объединения номера кадра и смещения генерируется реальный (физический) адрес (если требуемой записи в TLB нет, ее получают из таблицы страниц).

После того как сгенерирован реальный адрес, представляющий собой дескриптор и остаток адреса, выполняется обращение к кешу для выяснения, не содержится ли в нем блок с интересующим нас словом.

Если ответ положительный, то требуемое значение передается процессору; в противном случае происходит выборка слова из основной памяти.

Сложное аппаратное обеспечение процессора, вовлечено в единственное обращение к памяти, когда виртуальный адрес преобразуется в реальный. Это приводит к обращению к записи таблицы страниц, которая может оказаться в TLB, в основной памяти или на диске. Само слово, к которому осуществляется обращение, тоже может оказаться в разных местах - в кеше, в основной памяти или на диске.

Если слово находится только на диске, страница, содержащая это слово, должна быть загружена в основную память, а блок, содержащий слово, - в кеш . Кроме того, должна быть обновлена запись для данной страницы в таблице страниц.

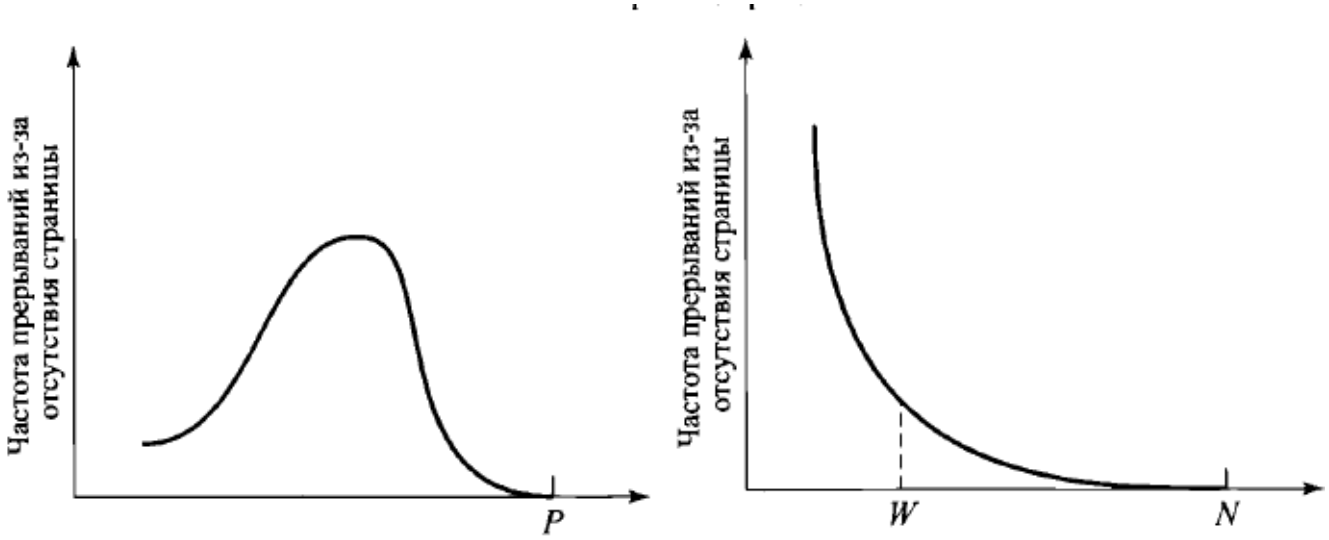
Размер страницы

Весьма важным вопросом при разработке является выбор размера страниц. Здесь следует учесть сразу несколько факторов. Один из них - внутренняя фрагментация. Понятно, что внутренняя фрагментация, которую желательно уменьшить для оптимизации использования основной памяти, находится в прямой зависимости от размера страницы.

С другой стороны, чем меньше размер страниц, тем больше их требуется для процесса, что означает увеличение таблицы страниц. Для больших программ в загруженной многозадачной среде это может означать, что часть таблиц страниц активных процессов будет находиться в виртуальной памяти и при обращении к памяти будет возникать двойное прерывание из-за отсутствия страницы: сначала - при получении требуемой записи из таблицы страниц, а затем - при обращении к странице процесса.

Еще одним фактором, который следует учесть, являются физические характеристики большинства устройств вторичной памяти, приводящие к тому, что передача больших блоков осуществляется более эффективно.

Вопрос усложняется еще и тем, что на частоту возникновения прерывания из-за отсутствия страницы в основной памяти влияет размер страницы.



P — размер процесса
W — размер рабочего множества
N — общее количество страниц процесса

Рис. 8.10. Типичное поведение страничной организации

На рис. 8.10, *а* показано обычное поведение частоты возникновения прерываний из-за отсутствия страницы с учетом принципа локальности. Если размер страницы очень мал, то в памяти размещается относительно большое количество страниц процесса.

Через некоторое время страницы в памяти будут содержать части процесса, сосредоточенные вблизи последних обращений, и частота возникновения прерывания из-за отсутствия страницы должна быть невелика.

По мере увеличения размера страницы каждая отдельная страница будет содержать данные, которые располагаются все дальше и дальше от последних выполненных обращений к памяти. Соответственно, действие принципа локальности ослабевает, и наблюдается рост количества прерываний из-за отсутствия страницы. В конце концов, когда размер страницы начинает становиться сравнимым с размером процесса (точка P на графике), прерывания из-за отсутствия страницы становятся все более и более редкими, а по достижении размера этого процесса прекращаются вовсе.

Следует учитывать также влияние количества кадров, выделенных процессу.

На рис. 8.10, *б* показано, что для фиксированного размера страницы частота возникновения прерываний из-за отсутствия страницы уменьшается с ростом числа страниц, находящихся в основной памяти. Таким образом, на программную стратегию (объем памяти, выделяемой процессу) влияет аппаратное решение (размер страницы).

Таблица 8.3. Примеры размеров страниц

Компьютер	Размер страницы
Atlas	512 48-битовых слов
Honeywell-Multics	1024 36-битовых слова
IBM 370/XA и 370/ESA	4 Кбайт
Семейство VAX	512 байт
IBM AS/400	512 байт
DEC Alpha	8 Кбайт
MIPS	От 4 Кбайт до 16 Мбайт
UltraSPARC	От 8 Кбайт до 4 Мбайт
Pentium	От 4 Кбайт до 4 Мбайт
Intel Itanium	От 4 Кбайт до 256 Мбайт
Intel core i7	От 4 Кбайт до 1 Гбайт

Решение об используемом размере страниц связано с размером физической основной памяти и размером программы. Растет не только объем основной памяти в компьютерах, но и адресное пространство, используемое приложениями.

Эта тенденция наиболее заметна в персональных компьютерах и рабочих станциях, в которых особенно резко проявляется увеличение размеров и возрастание сложности используемых приложений. Кроме того, современные технологии программирования, используемые в больших программах, приводят к снижению локальности ссылок процесса.

В качестве примеров можно привести следующие.

- Объектно-ориентированные технологии, стимулирующие использование множества мелких модулей кода и данных с

обращениями к большому количеству объектов за относительно короткое время.

- Многопоточные приложения, приводящие к внезапным изменениям в потоке команд и обращениям к памяти, разбросанным по разным адресам.

Результативность поиска в TLB определенного размера с ростом размера процессов и уменьшением локальности снижается. При таком положении дел TLB может стать узким местом, ограничивающим производительность.

Один из способов повышения производительности TLB - использование большого TLB с большим количеством записей. Однако увеличение размера TLB связано с другими аспектами аппаратного решения вопросов обращения к памяти, с такими как размер кеша основной памяти и количество обращений к памяти при выполнении одной команды, что заставляет сделать вывод о невозможности роста размера TLB такими же темпами, как и увеличение размера основной памяти.

Альтернативой может быть использование больших размеров страниц, с тем, чтобы каждая запись в TLB ссылалась на большой блок памяти. Однако мы уже видели, что использование больших размеров страниц может привести к потере производительности.

Ряд разработчиков пришли к использованию множественных размеров страниц, и некоторые из микропроцессоров поддерживают эту методику. Множественные размеры страниц обеспечивают необходимую для эффективного использования TLB гибкость. Большие непрерывные области адресного пространства процесса, например программный код, могут отображаться с использованием небольшого количества больших страниц, в то время как для отображения стеков потоков могут использоваться страницы малого размера.

Однако большинство коммерческих операционных систем все еще поддерживают только один размер страниц, независимо от

способности аппаратного обеспечения работать со страницами разного размера.

Причина этого отставания в том, что с размером страниц связано большое количество разнообразных аспектов операционных систем, и переход на множественный размер страниц оказывается очень сложным.

Сегментация

Значение виртуальной памяти

Сегментация позволяет программисту рассматривать память как область, состоящую из множества адресных пространств, или сегментов.

Сегменты могут иметь разные (на самом деле динамические) размеры. Обращения к памяти используют адреса, представляющие собой пары (*номер сегмента, смещение*).

Такая организация имеет ряд преимуществ по сравнению с несегментированным адресным пространством:

1. Упрощается обработка растущих структур данных. Если программисту заранее не известен размер структур данных, с которыми предстоит работать, и есть возможность использовать сегментацию, структуре данных может быть назначен ее собственный сегмент, размер которого операционная система будет увеличивать или уменьшать по мере необходимости. Если сегмент, размер которого следует увеличить, находится в основной памяти и для его увеличения нет свободного места, операционная система может переместить его в большую область или выгрузить на диск (в этом случае увеличенный сегмент будет загружен вновь при первой возможности).
2. Программы могут изменяться и перекомпилироваться независимо от компиляции или компоновки всего множества программ (что осуществляется при использовании множественных сегментов).

3. Упрощается совместное использование кода и данных разными процессами. Программист может поместить код утилиты или необходимые данные в отдельный сегмент, к которому будут обращаться другие процессы.
4. Улучшается защита. Так как сегмент представляет собой точно определенные множества программ или данных, программист или системный администратор может назначать права доступа просто и удобно.

Организация

При рассмотрении простой сегментации мы отмечали, что каждый процесс имеет собственную таблицу сегментов, и при загрузке всех сегментов процесса в основную память создается таблица сегментов процесса, которая также загружается в основную память. В каждой записи таблицы сегментов указаны начальный адрес соответствующего сегмента в основной памяти и его длина. Такая же таблица сегментов нужна и при схеме виртуальной памяти, основанной на сегментации.

б) Сегментация

Типичным приемом является использование отдельной таблицы сегментов для каждого процесса. Записи таблицы сегментов в этом случае усложняются (рис. 8.1, б). Поскольку в основной памяти могут находиться не все сегменты процесса, в каждой записи требуется наличие бита присутствия, указывающего, располагается ли данный сегмент в основной памяти.

Если сегмент расположен в основной памяти, то запись включает его начальный адрес и длину.

Еще один бит, необходимый в данной схеме, - бит модификации, указывающий, было ли изменено содержимое сегмента со времени его последней загрузки в основную память. Если изменений не было, то при выгрузке сегмента нет необходимости в его записи на диск.

Могут иметься и другие управляющие биты, например при организации защиты или совместного использования на уровне сегментов.

Основной механизм чтения слова из памяти включает трансляцию виртуального, или логического, адреса, состоящего из номера сегмента и смещения, в физический адрес с использованием таблицы сегментов. Поскольку таблица сегментов имеет переменную длину, зависящую от размера процесса, мы не можем рассчитывать на ее хранение в регистрах, и для хранения таблицы сегментов используется основная память.

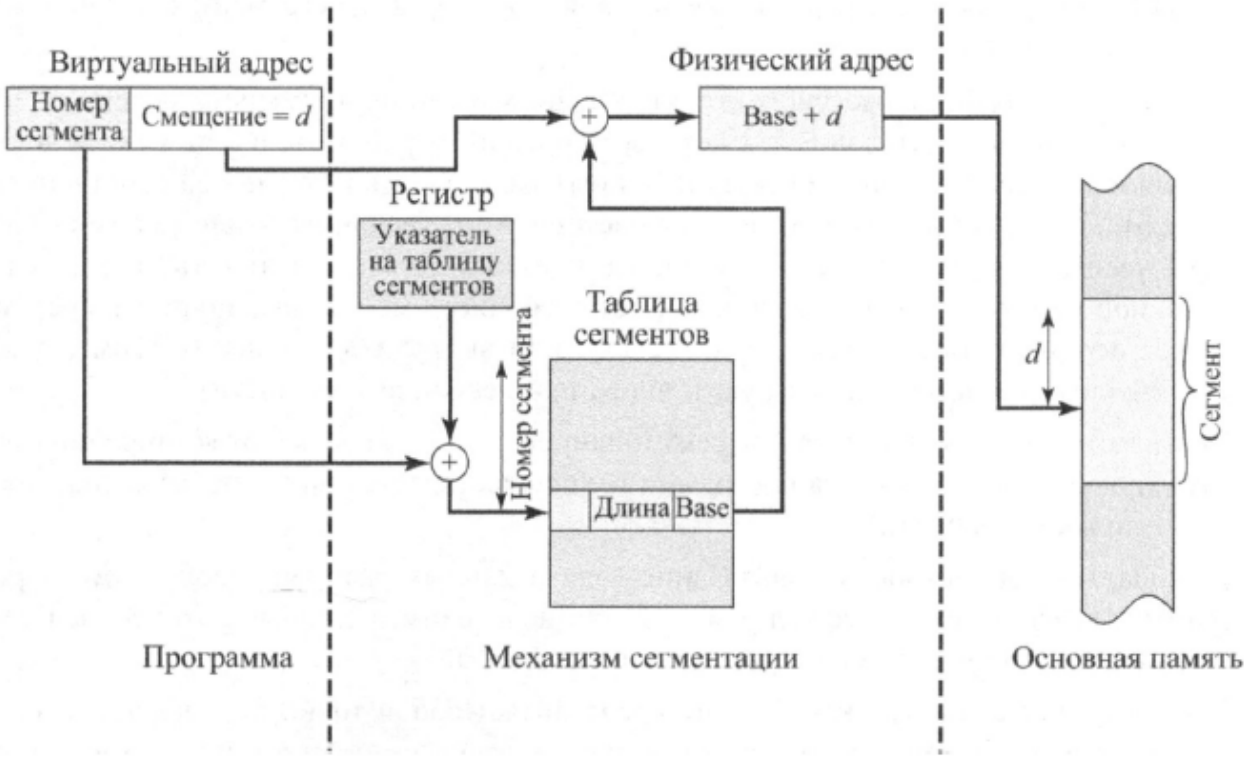


Рис. 8.11. Трансляция адреса в системе с сегментацией

На рис. 8.11 предложена аппаратная реализация описываемой схемы (обратите внимание на подобие с рис. 8.2).

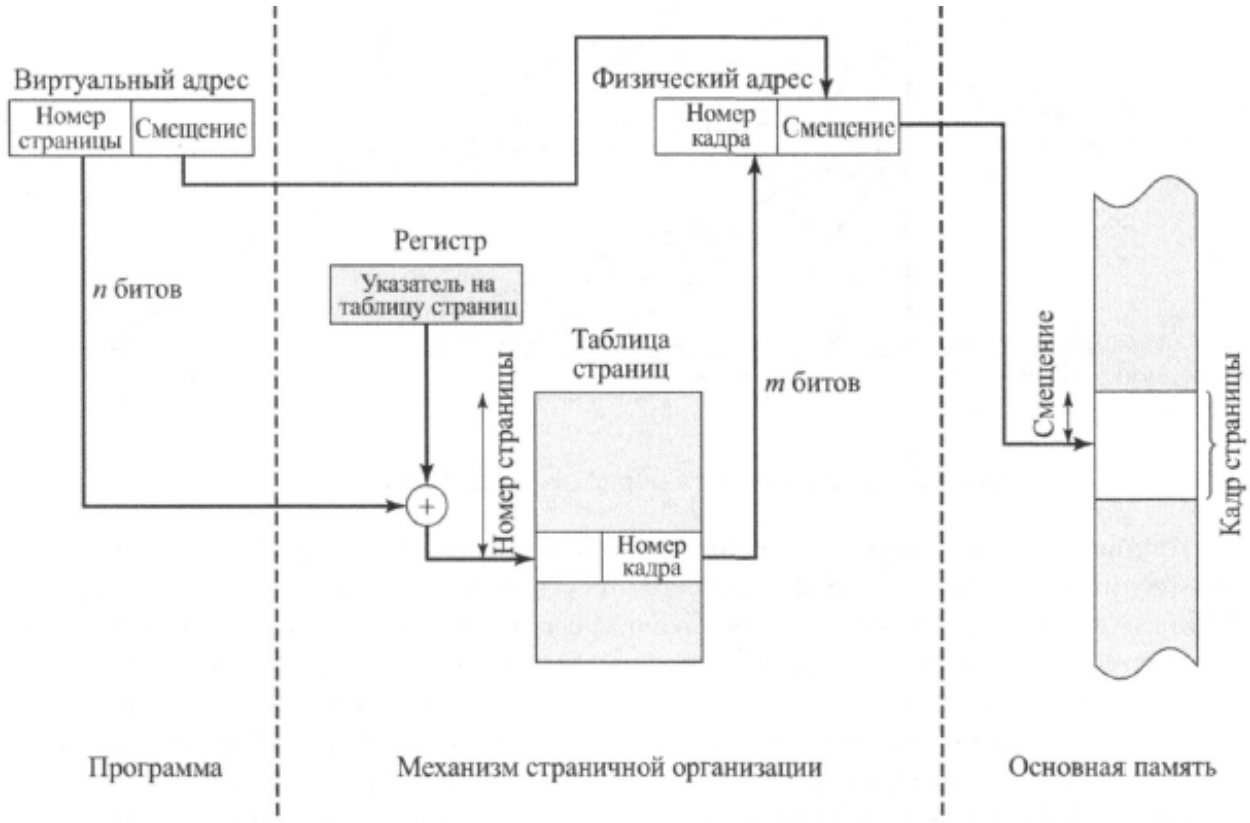


Рис. 8.2. Трансляция адреса в системе со страничной организацией

Когда запускается определенный процесс, в регистре хранится стартовый адрес его таблицы сегментов. Номер сегмента из виртуального адреса используется в качестве индекса таблицы, позволяющего определить начальный адрес сегмента. Для получения физического адреса к начальному адресу сегмента добавляется смещение из виртуального адреса.

Комбинация сегментации и страничной организации

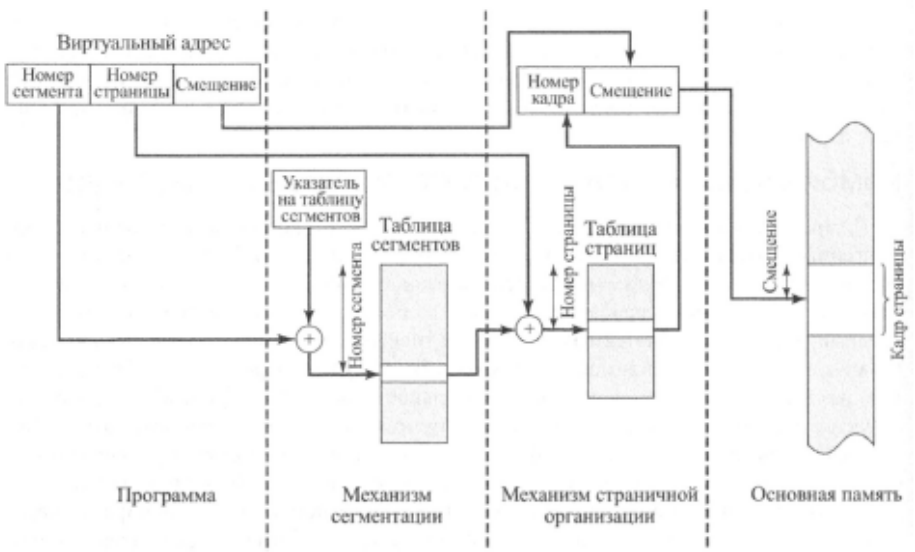


Рис. 8.12. Трансляция адреса при совместном использовании сегментации и страничной организации

И страничная организация, и сегментация имеют свои достоинства. Страничная организация, прозрачная для программиста, устраняет внешнюю фрагментацию и таким образом обеспечивает эффективное использование основной памяти. Кроме того, поскольку перемещаемые в основную память и из нее блоки имеют фиксированный, одинаковый размер, облегчается создание эффективных алгоритмов управления памятью.

Сегментация, являясь видимой для программиста, имеет перечисленные в предыдущем разделе достоинства, включающие модульность, возможность обработки растущих структур данных, а также поддержку совместного использования и защиты памяти. Некоторые вычислительные системы, будучи оснащенными соответствующим аппаратным обеспечением и операционной системой, используют достоинства обоих методов.

В такой комбинированной системе адресное пространство пользователя разбивается на ряд сегментов по усмотрению программиста. Каждый сегмент, в свою очередь, разбивается на ряд страниц фиксированного размера, соответствующего размеру кадра основной памяти.

Если размер сегмента меньше размера страницы, он занимает страницу целиком. С точки зрения программиста, логический адрес в этом случае состоит из номера сегмента и смещения в нем.

С позиции операционной системы смещение в сегменте следует рассматривать как номер страницы определенного сегмента и смещение в ней.

На рис. 8. 12 предложена структура для поддержки комбинации сегментации и страничной организации.

С каждым процессом связаны одна таблица сегментов и несколько (по одной на сегмент) таблиц страниц. При работе определенного процесса в регистре процессора хранится начальный адрес соответствующей таблицы сегментов.

Получив виртуальный адрес, процессор использует его часть, представляющую номер сегмента, в качестве индекса в таблице

сегментов для поиска таблицы страниц данного сегмента. После этого часть адреса, представляющая собой номер страницы, используется для поиска соответствующего кадра основной памяти в таблице страниц; затем часть адреса, представляющая смещение, используется для получения искомого физического адреса путем добавления к начальному адресу кадра.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ОПЕРАЦИОННОЙ СИСТЕМЫ

Особенности разработки части программного обеспечения операционной системы, управляющей памятью, зависят от ответа на три основных вопроса.

1. Будет ли использоваться виртуальная память.

2. Будет ли использоваться сегментация, страничная адресация или обе указанные технологии.

3. Какие алгоритмы будут использованы для различных аспектов управления памятью.

Ответы на первые два вопроса тесно связаны с используемой аппаратной платформой. Так, ранние реализации UNIX не использовали виртуальную память, поскольку процессоры, на которых работали эти операционные системы, не поддерживали ни сегментации, ни страничной организации, а без аппаратной поддержки преобразования адресов ни сегментация, ни страничная организация никакой практической ценности не представляют.

Ответ на третий из поставленных вопросов связан с программным обеспечением операционной системы.

В табл. 8.4 перечислены основные рассматриваемые в этом разделе стратегии. В каждом случае ключевым вопросом становится производительность: требуется сократить количество прерываний из-за отсутствия страницы в памяти, поскольку их обработка приводит к существенным накладным расходам, которые включают как минимум принятие решения о том, какие резидентные страницы

должны быть замещены, и операции ввода-вывода по замене страниц в основной памяти.

Таблица 8.4. Стратегии операционной системы для виртуальной памяти

Стратегия выборки	**Управление резидентным множеством**
По требованию	Размер резидентного множества
Предварительная выборка	Фиксированный
	Переменный
Стратегия размещения	Область видимости замещения
	Глобальная
Стратегия замещения	Локальная
Основные алгоритмы	
Оптимальный	**Стратегия очистки**
Дольше всех неиспользовавшиеся	По требованию
Первым вошел — первым вышел	Предварительная очистка
Часовой	
Буферизация страниц	**Управление загрузкой**
	Степень многозадачности

Кроме того, операционная система должна активировать на время выполнения медленных операций ввода-вывода другой готовый к работе процесс.

Ни одна из перечисленных в табл. 8.4 стратегий не является "наилучшей".

Задача управления памятью чрезвычайно сложна. Кроме того, производительность каждого определенного набора стратегий зависит от размера основной памяти, относительной скорости основной и вторичной памяти, размера и количества конкурирующих за ресурсы процессов и поведения отдельных программ.

Последняя характеристика зависит от природы приложения, использованных языка программирования и компилятора, стиля программиста, а для интерактивных программ - от динамичности пользователя.

Стратегия выборки

Стратегия выборки определяет, когда страница должна быть передана в основную память. Два основных варианта - по

требованию и предварительно. При ***выборке по требованию*** страница передается в основную память только тогда, когда выполняется обращение к ячейке памяти, расположенной на этой странице.

В случае ***предварительной выборки*** загружается не только страница, вызвавшая прерывание обращения. Предварительная выборка использует характеристики большинства устройств вторичной памяти, таких как диски, у которых имеются время поиска и задержка, связанная с вращением диска. Если страницы процесса расположены во вторичной памяти последовательно, то гораздо более эффективной будет загрузка в основную память нескольких последовательных страниц за один раз, чем загрузка этих же страниц по одной в течение некоторого промежутка времени. Естественно, эта стратегия не дает никакого выигрыша, если обращения к дополнительно загруженным страницам не происходит.

Стратегия размещения

Стратегия размещения определяет, где именно в физической памяти будут располагаться части процесса. В случае "чистой" сегментации стратегия размещения является весьма важным вопросом, решения которого в виде стратегий первого подходящего, очередного подходящего и других.

Однако для систем, использующих только страничную организацию или страничную организацию в сочетании с сегментацией, стратегия размещения обычно не так важна, поскольку аппаратная трансляция адреса и аппаратное обращение к памяти одинаково результативны при любых сочетаниях "страница-кадр".

Стратегия замещения

Эта тема достаточно сложна методологически, поскольку включает ряд взаимосвязанных вопросов.

- Какое количество кадров должно быть выделено каждому активному процессу.

• Должно ли множество страниц, которые потенциально могут быть замещены загружаемыми страницами, ограничиваться одним процессом или в качестве кандидатов на замещение могут рассматриваться все кадры страниц основной памяти.

• Какие именно страницы из рассматриваемого множества следует выбрать для замещения.

Вопросы стратегии замещения представляют собой, пожалуй, наиболее полно изученный аспект управления памятью.

Когда все кадры основной памяти заняты и требуется разместить новую страницу в процессе обработки прерывания из-за отсутствия страницы, стратегия замещения определяет, какая из находящихся в настоящее время в основной памяти страниц должна быть выгружена, чтобы освободить кадр для загружаемой страницы.

Все стратегии направлены на то, чтобы выгрузить страницу, обращений к которой в ближайшем будущем не последует.

В соответствии с принципом локальности часто наблюдается сильная корреляция между множеством страниц, к которым в последнее время были обращения, и множеством страниц, к которым будут обращения в ближайшее время.

Таким образом, большинство стратегий пытаются определить будущее поведение программы на основе ее прошлого поведения. При рассмотрении разных стратегий следует учитывать, что чем более совершенный и интеллектуальный алгоритм использует стратегия, тем выше будут накладные расходы при его реализации.

Перед рассмотрением различных алгоритмов следует упомянуть об одном ограничении: некоторые кадры основной памяти могут быть заблокированы. Блокировка кадра означает, что страница, хранящаяся в данный момент в этом кадре, не может быть замещена.

Большинство ядер операционных систем хранятся в заблокированных кадрах, так же как и основные управляющие структуры. Кроме того, в заблокированных кадрах основной памяти могут располагаться буферы ввода-вывода и другие критические по

отношению ко времени доступа данные и код. Блокировка осуществляется путем установки соответствующего бита у каждого кадра. Этот бит может находиться как в таблице кадров, так и быть включенным в текущую таблицу страниц.

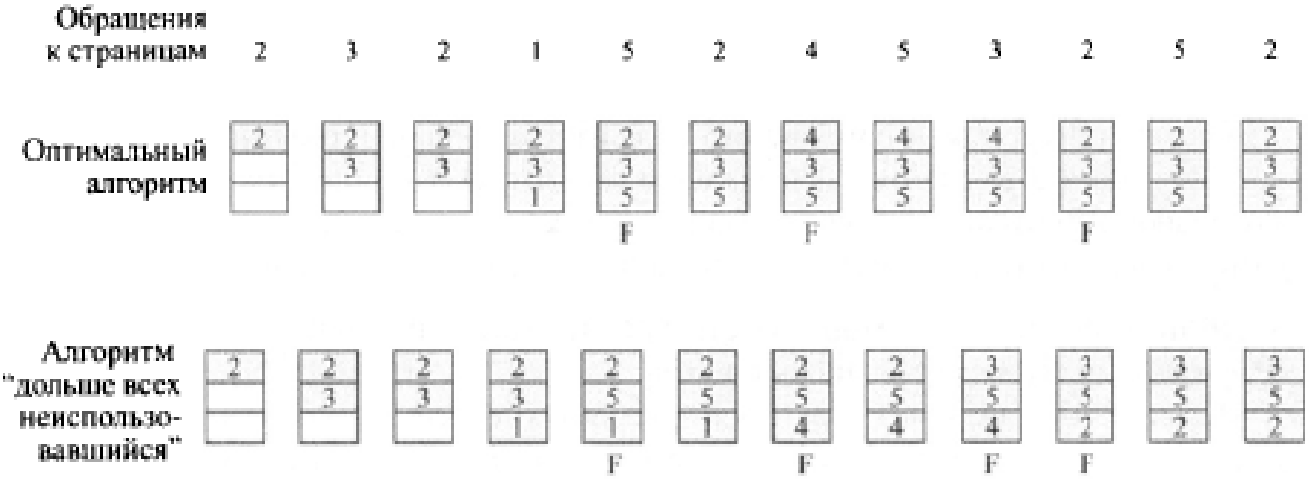
Основные алгоритмы

Независимо от стратегии управления резидентным множеством имеется ряд основных алгоритмов, используемых для выбора замещаемой страницы.

- Оптимальный алгоритм
- Алгоритм дольше всех неиспользовавшегося элемента
- Алгоритм "первым вошел - первым вышел"
- Часовой алгоритм

Оптимальная стратегия состоит в выборе для замещения той страницы, обращение к которой будет через наибольший промежуток времени по сравнению со всеми остальными страницами. Можно показать, что этот алгоритм приводит к минимальному количеству прерываний из-за отсутствия страницы. Понятно, что реализовать такой алгоритм невозможно, поскольку для этого системе требуется знать все будущие события.

Однако этот алгоритм является стандартом , с которым сравниваются реальные алгоритмы. На рис. 8.14 приведен пример оптимальной стратегии.



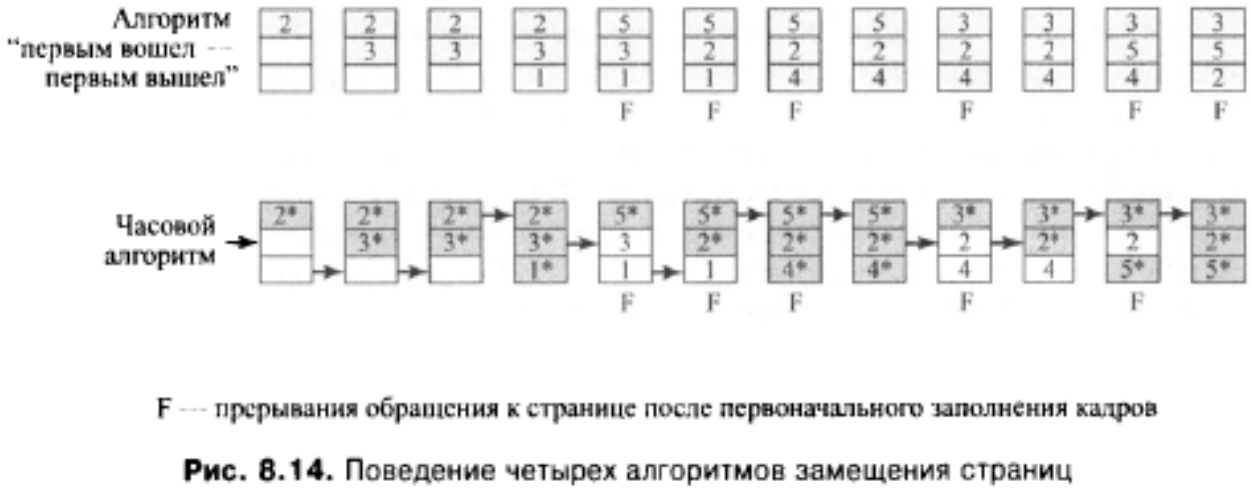
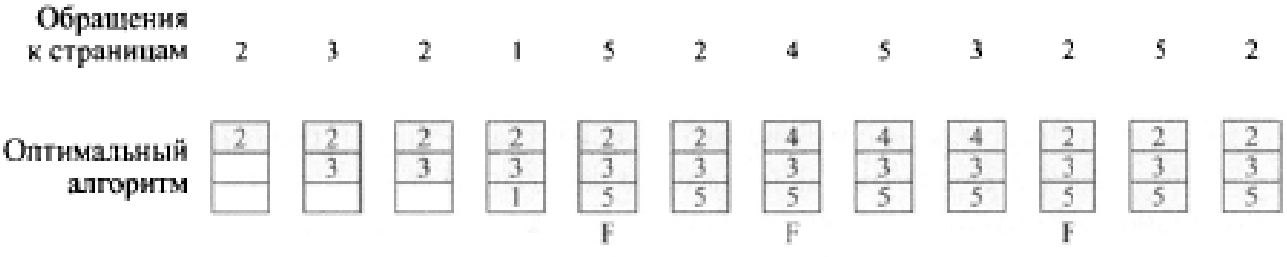


Рис. 8.14. Поведение четырех алгоритмов замещения страниц

Предполагается, что для данного процесса используется фиксированное распределение кадров (фиксированный размер резидентного множества, состоящего из трех кадров). Выполнение процесса приводит к обращениям к пяти различным страницам. В процессе работы обращения к страницам выполняются в следующем порядке:

2 3 2 1 5 2 4 5 3 2 5 2

Это означает, что сначала выполняется обращение к странице 2, затем - к странице 3 и т.д. Оптимальная стратегия приводит после заполнения всего множества кадров к трем прерываниям обращения к странице (обозначенным на рисунке буквами "F").



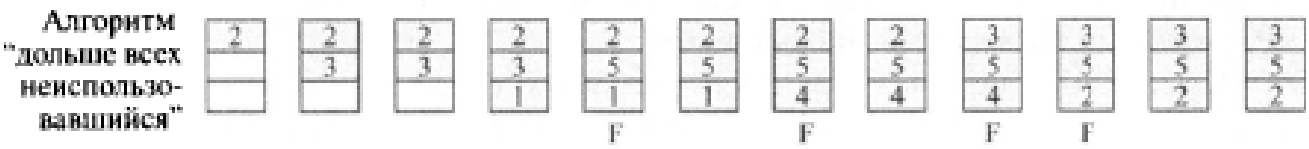
Стратегия дольше всех неиспользовавшегося элемента замещает в памяти ту страницу, обращений к которой не было дольше, чем к другим. Согласно принципу локальности можно ожидать, что эта страница не будет использоваться и в ближайшем будущем.

Эта стратегия и в самом деле недалека от оптимальной. Основная проблема заключается в сложности ее реализации. Один из

вариантов реализации предполагает отмечать время последнего обращения к странице; это должно делаться при каждом обращении к памяти, независимо от того, к чему выполняется обращение - к коду или к данным. Даже в случае аппаратной поддержки этой схемы накладные расходы слишком велики.

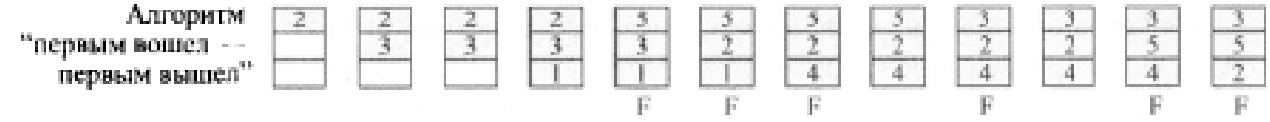
Еще один вариант предполагает поддержание стека обращений к страницам, что тоже обходится недешево для производительности системы.

2 3 2 1 5 2 4 5 3 2 5 2



На рис. 8.14 приведен пример выполнения алгоритма дольше всех неиспользовавшегося элемента с тем же потоком данных, что и для оптимального алгоритма. В этом примере возникают четыре прерывания обращения к странице.

2 3 2 1 5 2 4 5 3 2 5 2



Стратегия "первым вошел - первым вышел" рассматривает кадры страниц процесса как циклический буфер с циклическим же удалением страниц из него. Все, что требуется для реализации этой стратегии, - это указатель, циклически проходящий по кадрам страниц процесса. Таким образом, это одна из простейших в реализации стратегий замещения.

Логика ее работы заключается в том, что замещается страница, находящаяся в основной памяти дольше других. Однако далеко не всегда эта страница редко используется; очень часто некоторая область данных или кода интенсивно используется программой, и

страницы из этой области при использовании описанной стратегии будут загружаться и выгружаться вновь и вновь.

На рис. 8.14 описанная стратегия приводит к шести прерываниям отсутствия страницы. Заметим, что предыдущая стратегия распознает, что чаще других используются страницы 2 и 5, в то время как стратегия "первым вошел - первым вышел" на это не способна.

Хотя стратегия дольше всех неиспользовавшегося элемента и близка к оптимальной, она трудна в реализации и приводит к значительным накладным расходам.

Стратегия "первым вошел - первым вышел" реализуется очень просто, но относительно редко приводит к хорошим результатам.

В течение долгого времени разработчики операционных систем испытывали различные алгоритмы, пытаясь достичь увеличения производительности стратегии дольше всех неиспользовавшегося элемента при значительном снижении накладных расходов. Многие из этих алгоритмов представляют собой варианты схемы, известной как часовая стратегия (clock policy).

В простейшей схеме часовой стратегии с каждым кадром связывается один дополнительный бит, известный как бит использования. Когда страница впервые загружается в кадр, бит использования устанавливается равным 1.

При последующих обращениях к странице, вызвавших прерывание из-за отсутствия страницы, этот бит также устанавливается равным 1. При работе алгоритма замещения множество кадров, являющихся кандидатами на замещение (текущий процесс, локальная область видимости, вся основная память или глобальная область видимости) рассматривается как циклический буфер, с которым связан указатель. При замещении страницы указатель перемещается к следующему кадру в буфере.

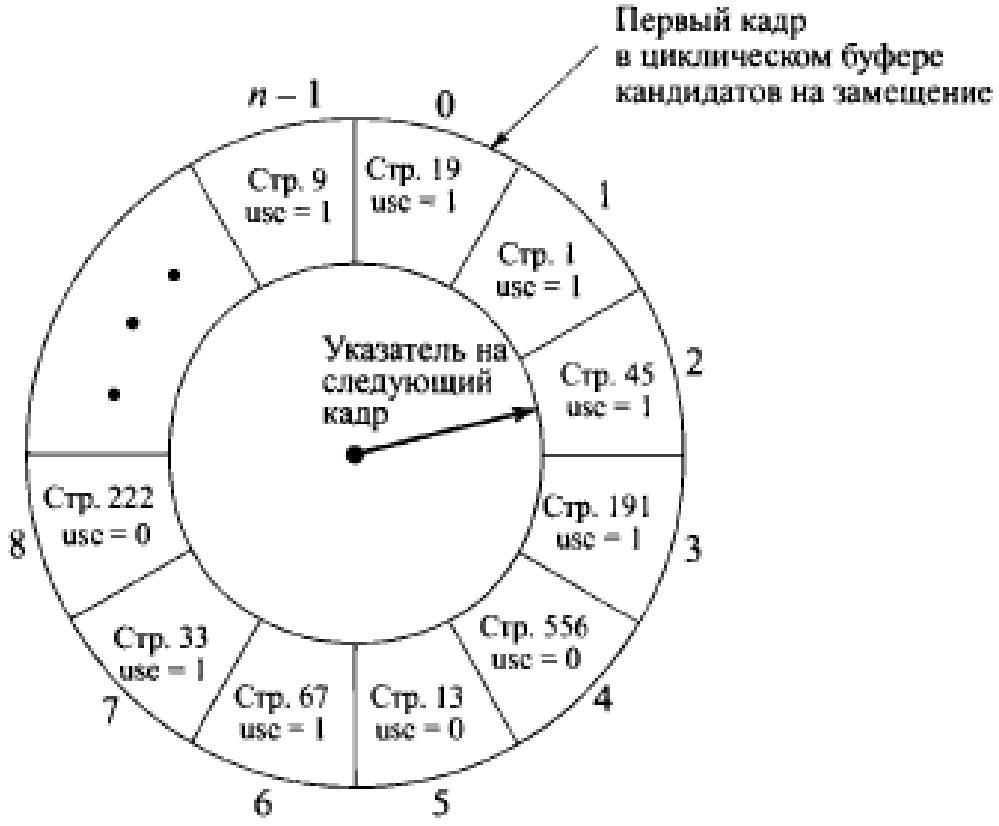
Когда наступает время замещения страницы, операционная система сканирует буфер для поиска кадра, бит использования которого равен 0.

Всякий раз, когда в процессе поиска встречается кадр с битом использования, равным 1, он сбрасывается в 0. Первый же встреченный кадр с нулевым битом использования выбирается для замещения.

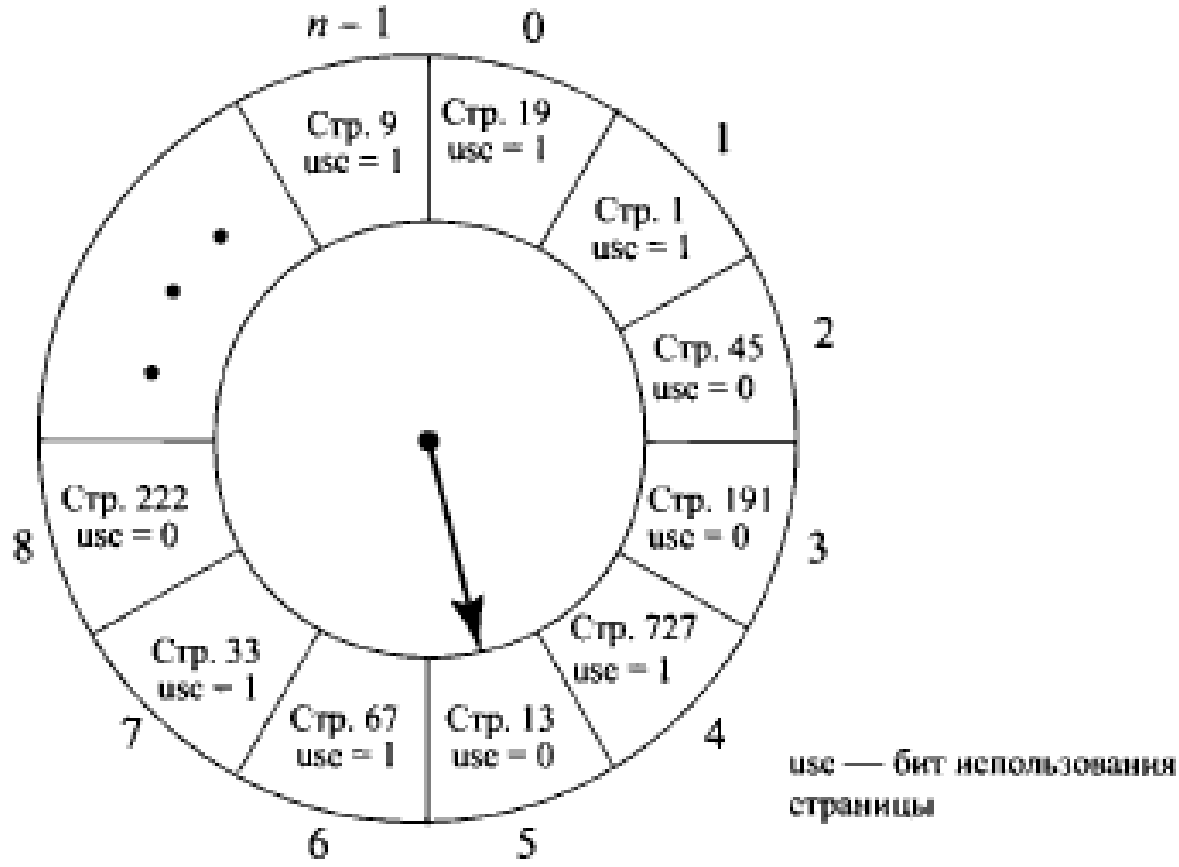
Если все кадры имеют бит использования, равный 1, указатель совершает полный круг и возвращается к начальному положению, заменяя страницу в этом кадре.

Как видим, эта стратегия схожа со стратегией «первым вошел - первым вышел»: но отличается тем, что кадры, имеющие установленный бит использования, пропускаются алгоритмом.

Буфер кадров страниц представлен в виде круга, откуда и произошло название стратегии. Ряд операционных систем используют различные варианты часовой стратегии.



а) Состояние буфера непосредственно перед замещением страницы



б) Состояние буфера непосредственно после замещения страницы

Рис. 8.15. Пример работы часового алгоритма

Буферизация страниц

Хотя алгоритмы дольше всех неиспользовавшегося и часовой и превосходят алгоритм "первым вошел - первым вышел", они оба сложны и имеют высокие накладные расходы по сравнению с последним.

Кроме того, следует учитывать, что стоимость замещения модифицированной страницы выше стоимости замещения немодифицированной, которую не надо записывать во вторичную память.

Есть еще одна интересная стратегия, которая может повысить производительность страничной организации при использовании простейшего алгоритма замещения. Это ***буферизация страниц***.

В качестве алгоритма замещения страниц используется простейший алгоритм "первым вошел - первым вышел".

Для повышения его производительности замещаемая страница не теряется, а вносится в один из двух списков: в список свободных страниц, если страница не модифицировалась, или в список модифицированных страниц.

Заметим, что физически страница не перемещается - вместо этого ее запись удаляется из таблицы страниц и переносится в список свободных или модифицированных страниц.

Список свободных страниц представляет собой список кадров страниц, доступных для чтения. VMS пытается постоянно поддерживать некоторое небольшое количество свободных кадров. Когда страница считывается в кадр, используется кадр, расположенный в начале списка; при этом страница, находившаяся в нем ранее, уничтожается.

Размер резидентного множества

При использовании страничной виртуальной памяти для подготовки процесса к выполнению нет необходимости (да это может быть и невозможно) размещать в основной памяти все его страницы.

Следовательно, операционная система должна принять решение о том, какое количество страниц следует загрузить, т.е. какое количество памяти выделяется конкретному процессу. Здесь играет роль ряд факторов.

• Чем меньше памяти выделяется процессу, тем больше процессов может одновременно находиться в основной памяти. Это увеличивает вероятность того, что операционная система в любой момент времени найдет как минимум один готовый к выполнению процесс и, следовательно, снижаются затраты времени на свопинг процессов.

• При относительно небольшом количестве страниц процесса, размещенных в основной памяти, несмотря на принцип локальности частота возникновения прерываний из-за отсутствия страницы будет достаточно велика.

• После определенного предела дополнительное выделение основной памяти некоторому процессу в соответствии с принципом локальности не будет приводить к сколь-нибудь значительному снижению частоты возникновения прерываний из-за отсутствия страницы.

С учетом этих факторов в современных операционных системах используются два типа стратегий. Стратегия *фиксированного* распределения выделяет процессу фиксированное количество кадров основной памяти, в пределах которого он выполняется. Это количество определяется в момент начальной загрузки (при создании процесса) и может быть определено на основании типа процесса (интерактивный, пакетный и т.п.) либо на основании указаний программиста или системного администратора. При использовании стратегии фиксированного распределения прерывание из-за отсутствия страницы приводит к замещению требуемой страницей одной из страниц процесса.

Стратегия ***переменного*** распределения позволяет количеству выделенных процессу кадров страниц изменяться во время работы процесса. В идеале процессу, который страдает от большого количества прерываний из-за отсутствия страницы (принцип локальности для данного процесса выполняется слабо), выделяются дополнительные кадры страниц; и напротив, у процесса, при работе которого таких прерываний относительно мало (что указывает на то, что поведение процесса с точки зрения локальности достаточно хорошее), могут быть изъяты кадры в расчете на то, что это не намного увеличит частоту возникновения прерываний.

Стратегия переменного распределения представляется более мощной, однако трудности данного подхода состоят в том, что операционная система при этом должна отслеживать поведение процессов. Это приводит к очень высоким накладным расходам, зависящим от возможностей аппаратного обеспечения конкретной платформы.

УПРАВЛЕНИЕ ПАМЯТЬЮ в UNIX

Поскольку операционная система UNIX разрабатывалась как машинно-независимая, система управления памятью в UNIX варьируется от одной операционной системы к другой.

Ранние версии UNIX использовали переменное распределение памяти без применения виртуальной памяти. Текущие реализации UNIX используют страничную виртуальную память.

Страничная система обеспечивает реализацию возможностей виртуальной памяти, распределяя кадры основной памяти среди процессов, а также среди буферов диска. Хотя описанная схема эффективно работает с пользовательскими процессами и дисковым вводом-выводом, страничная виртуальная память мало приспособлена для управления памятью ядра.

Для этой цели используется распределение памяти ядра. Рассмотрим оба механизма.

Страничная система

Структуры данных

Для страничной виртуальной памяти UNIX использует ряд структур данных, которые (с минимальной коррекцией) являются машинно-независимыми (см. рис. 8.20 и табл. 8.6).

• **Страница таблиц**. Обычно для каждого процесса используется одна таблица страниц, в которой каждой странице виртуальной памяти процесса соответствует одна запись.

• **Дескриптор дискового блока**. В этой таблице каждой странице процесса соответствует запись, описывающая дисковую копию этой страницы.

• **Таблица кадров страниц**. Описывает каждый кадр реальной памяти; таблица проиндексирована номерами кадров. Эта таблица используется алгоритмом замещения.

• **Таблица использования свопинга**. Для каждого устройства свопинга имеется своя таблица, в которой для каждой страницы на этом устройстве имеется своя запись.

Номер кадра страницы	Возраст	Копирование при записи	Модифицирована	Обращения	В памяти	Защищенность

а) Запись таблицы страниц

Номер устройства свопинга	Номер блока устройства	Тип памяти

б) Дескриптор дискового блока

Состояние страницы	Количество ссылок	Логическое устройство	Номер блока	Указатель на данные кадра

в) Запись таблицы кадров

Количество ссылок	Номер страницы/ единицы памяти

б) Запись таблицы свопинга

Рис. 8.20. Структуры данных системы управления памятью UNIX SVR4

Таблица 8.6. Параметры управления памятью UNIX SVR4

Запись таблицы страниц
Номер кадра страницы Указывает кадр в реальной памяти.
Возраст Указывает, как долго страница находится в памяти без обращения к ней. Длина и содержимое данного поля зависят от используемого процессора.
Копирование при записи Устанавливается, когда страница совместно используется несколькими процессами. Если один из процессов производит запись в страницу, сначала должны быть сделаны отдельные копии страницы для каждого из совместно использующих ее процессов. Эта возможность позволяет отложить операцию копирования до тех пор, пока она не станет необходима, и избежать ее в тех случаях, когда она не является таковой.
Модифицирована Указывает, изменено ли содержимое страницы.
Обращения Указывает, что к странице были обращения. Этот бит устанавливается равным нулю при первой загрузке страницы и может периодически сбрасываться алгоритмом замещения страниц.

В памяти	Указывает, что страница находится в основной памяти.
Защищенность	Указывает, что разрешена операция записи.
Дескриптор дискового блока	
Номер устройства свопинга	Номер логического устройства вторичной памяти, хранящей соответствующую страницу. Позволяет использовать для свопинга больше одного устройства.
Номер блока устройства	Расположение блока страницы на устройстве вторичной памяти.
Тип памяти	Вторичная память может представлять собой модуль свопинга или выполнимый файл. В последнем случае имеется признак, указывающий, должна ли распределяемая виртуальная память быть предварительно очищенной.
Запись таблицы кадров	
Состояние страницы	Указывает, свободен ли кадр или содержит страницу. В этом случае указывает статус страницы: на устройстве свопинга, в выполнимом файле или выполняется прямое обращение к памяти.
Количество ссылок	Количество процессов, обращающихся к странице.
Логическое устройство	Логическое устройство, содержащее копию страницы.
Номер блока	Расположение блока копии страницы на логическом устройстве.
Указатель на данные кадра	Указатель на другие записи таблицы в списке свободных страниц и в хеш-очереди страниц.
Таблица свопинга	
Количество ссылок	Количество записей таблицы страниц, указывающих на страницы на устройстве свопинга.
Номер страницы/единицы памяти	Идентификатор страницы в модуле вторичной памяти.

Большинство полей, определенных в табл. 8.6, не требуют пояснений.

Замещение страниц

Для замещения страниц используется таблица кадров. Для создания списков в этой таблице применяются несколько указателей. Все доступные кадры объединены в один список свободных кадров, в которых могут размещаться страницы. Когда количество доступных страниц становится ниже некоторого порогового значения, ядро в качестве компенсации отдает ряд страниц загруженных процессов.

Алгоритм замещения страниц, использованный в SVR4, представляет собой усовершенствованный часовой алгоритм, известный под названием "часовой алгоритм с двумя стрелками" (рис. 8.21).

Рис. 8.21. Часовой алгоритм замещения с двумя стрелками

Этот алгоритм использует бит обращений из записи таблицы страниц для каждой из страниц памяти, которая может быть выгружена из основной памяти (т.е. не заблокирована). Этот бит устанавливается равным 0 при первоначальной загрузке страницы и равным 1 - при обращении к ней для чтения или записи. Передний

указатель проходит по страницам, содержащимся в списке пригодных для выгрузки страниц, и устанавливает бит обращений каждой из них равным 0.

Несколько позже по тем же страницам проходит задний указатель и проверяет бит обращений. Если он равен 1, значит, к данной странице было обращение между проверками ее передним и задним указателями, и такая страница игнорируется. Страница же, бит обращений которой остался равным 0, переносится в список выгружаемых страниц.

Работа алгоритма определяется двумя параметрами.

1. Частота сканирования. Частота, с которой указатели сканируют список страниц (в страницах в секунду).

2. Охват. Промежуток между передним и задним указателями.

Эти параметры в процессе загрузки операционной системы получают значения по умолчанию, основанные на количестве физической памяти. Частота сканирования в процессе работы может изменяться, чтобы соответствовать изменяющимся условиям работы системы. Параметр линейно изменяется от минимального значения до максимального, определенных при настройке системы, с изменением количества свободной памяти от максимального до минимального. Иными словами, чем меньше свободной памяти в системе, чем чаще выполняется сканирование. Охват определяет интервал между указателями и вместе с частотой сканирования определяет промежуток времени, в течение которого к странице должно произойти обращение, чтобы она оставалась в основной памяти.

Распределение памяти ядра

Ядро в процессе работы часто генерирует и уничтожает маленькие таблицы и буферы, память для каждого из которых выделяется динамически.

Беркли (Barkley) и Ли (Lee) из AT&T предложили модификацию, известную как "ленивая" система двойников, которая

принята в SVR4. Авторами было замечено, что UNIX часто демонстрирует устойчивое состояние памяти ядра, т.е. количество требующихся блоков определенного размера мало меняется со временем. Таким образом, вполне возможна ситуация, когда освобождающийся блок размером 2^i сливается со своим двойником в блок размером 2^{i+1} , который тут же вновь разделяется на два блока размером 2^i в соответствии с запросом системы.

Чтобы избежать излишних слияний и разделений блоков, слияние освобожденных блоков откладывается до того момента, когда оно оказывается действительно необходимым (и тогда производится максимально возможное количество слияний блоков).

Алгоритм схемы приведен на рис. 8.22.

```
Начальное значение D_i равно 0
После выполнения операций значение D_i
изменяется следующим образом.

(I) Если следующая операция является запросом
    на выделение блока:
        если имеется свободный блок, выбрать его
          если выбранный блок локально свободен,
            то D_i := D_i + 2
            иначе D_i := D_i + 1
        в противном случае
          получить два блока путем разделения большего
            блока на два меньших (рекурсивная операция).
          выбрать один из них, пометив второй как локально
            свободный.
          D_i остается неизменным (при этом значение D для
            других размеров блоков может измениться в связи
            с использованием рекурсивности).
(II) Если следующая операция — запрос на освобождение блока:
        при D_i ≥ 2
          пометить блок как локально свободный и освободить
            его локально
          D_i := D_i -2
        при D_i = 1
          пометить блок как глобально свободный и освободить
            его глобально; если это возможно, выполнить
            слияние блоков.
          D_i := 0
        при D_i = 0
          Пометить блок как глобально свободный и освободить
            его глобально; если это возможно, выполнить
            слияние блоков.
          Выбрать один локально свободный блок размером 2^i и
            освободить его глобально; если это возможно,
            выполнить слияние блоков.
          D_i := 0
```

Рис. 8.22. Алгоритм "ленивой" системы двойников

УПРАВЛЕНИЕ ПАМЯТЬЮ В LINUX

Многие характеристики схем управления памятью других реализаций UNIX применимы и к Linux, однако эта операционная система имеет и свои, присущие только ей, особенности. Вообще говоря, система управления памятью в Linux весьма сложна, и здесь мы дадим только краткое описание двух основных аспектов управления памятью в Linux: *виртуальной памяти процесса* и *распределение памяти ядра.*

Основной единицей памяти является физическая страница, которая представлена в ядре Linux соответствующей структурой. Размер страницы зависит от архитектуры; обычно он равен 4 Кбайт.

Linux также поддерживает возможность Hugepages, позволяющую задавать большие размеры страниц (например, 2 Мбайт). Существует несколько проектов, которые используют Hugepages для повышения производительности.

Виртуальная память Linux

Адресация виртуальной памяти

Linux использует трехуровневую структуру таблицы страниц, состоящую из следующих типов таблиц (каждая отдельная таблица имеет размер, равный одной странице).

• **Каталог страниц.** Активный процесс имеет единый каталог страниц, размер которого равен одной странице. Каждая запись в каталоге страниц указывает на одну страницу промежуточного каталога страниц. Каталог страниц активного процесса должен находиться в активной памяти.

• **Промежуточный каталог страниц.** Промежуточный каталог страниц может охватывать несколько страниц. Каждая запись промежуточного каталога указывает на одну страницу таблицы страниц.

• **Таблица страниц.** Таблица страниц также может охватывать несколько страниц. Каждая запись таблицы страниц указывает на одну виртуальную страницу процесса.

Для использования трехуровневой структуры таблицы страниц виртуальный адрес в Linux рассматривается как состоящий из четырех частей (рис. 8.23).

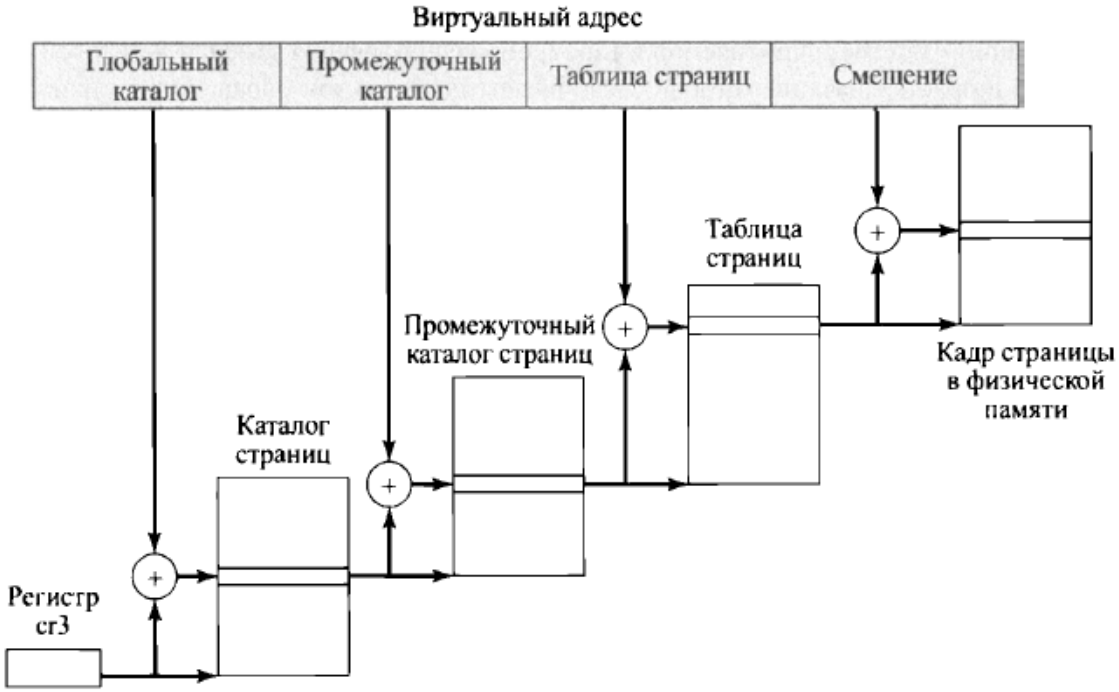


Рис. 8.23. Трансляция адреса в схеме виртуальной памяти Linux

Левое (наиболее значащее) поле используется в качестве индекса в каталоге страниц; следующее поле служит в качестве индекса в промежуточном каталоге страниц. Третье поле представляет собой индекс таблицы страниц, а четвертое - смещение в пределах выбранной страницы памяти.

Структура таблицы страниц Linux платформонезависима и разработана для работы с 64-разрядным процессором Alpha, который обеспечивает аппаратную поддержку трехуровневой страничной организации. При использовании 64-разрядных адресов использование только двух уровней может привести к тому, что таблицы и каталоги страниц будут очень большими. 32-разрядная архитектура x86 обладает только двухуровневым механизмом страничной организации, и программное обеспечение Linux использует двухуровневую схему путем определения размера промежуточного каталога, равного одной странице.

Обратите внимание, что все ссылки на дополнительный уровень косвенности устраняются оптимизатором во время компиляции, а не во время выполнения.

Таким образом, при использовании обобщенного трехуровневого дизайна на платформах, которые аппаратно

поддерживают только два уровня, снижение производительности не наблюдается.

Распределение страниц

Чтобы повысить эффективность чтения страниц из основной памяти и записи страниц в нее, Linux определяет механизм для работы со смежными блоками страниц, отображаемых на смежные блоки кадров страниц. С этой целью Linux использует систему двойников. Ядро поддерживает список групп смежных кадров фиксированного размера; группа может состоять из 1, 2, 4, 8, 16 или 32 кадров страниц.

При выделении и освобождении страниц в основной памяти доступные группы разделяются и объединяются с использованием алгоритма двойников.

Алгоритм замещения страниц

Алгоритм замещения страниц в Linux до версии 2.6.28 был основан на часовом алгоритме. В случае использования простого часового алгоритма с каждой страницей основной памяти связаны биты использования и модификации.

В схеме, применяемой в Linux, бит использования заменен 8-битовой переменной возраста, значение которой увеличивается при каждом обращении к странице.

В фоновом режиме Linux периодически сканирует страницы основной памяти и уменьшает значения их переменных возраста. Страницы с нулевым значением представляют собой "старые" страницы, к которым некоторое время не было обращений и которые являются наиболее подходящими кандидатами для замещения. Чем больше значение возраста страницы, тем чаще она использовалась в последнее время и тем менее она подходит для замещения.

Таким образом, алгоритм, используемый в Linux, представляет собой разновидность стратегии замещения наименее часто используемых страниц.

Начиная с Linux версии 2.6.28 алгоритм замены страницы, описанный в предыдущем абзаце, был пересмотрен, и в ядро был внесен новый алгоритм, именуемый алгоритмом разделения последнего использованного (split LRU).

Одной из проблем старого алгоритма является то, что периодическое сканирование пула страниц с ростом памяти потребляет все большее и большее количество времени процессора.

Новый алгоритм использует два флага, добавленных к каждой записи таблицы страниц: PG active и PG_referenced. Вся физическая память делится на различные "зоны" Linux, основанные на их адресах. В каждой зоне имеются два связанных списка, которые используются диспетчером памяти. а именно - списки активных и неактивных страниц.

Демон ядра kswapd периодически запускается в фоновом режиме для выполнения утилизации страниц в каждой зоне. Этот демон сканирует записи таблицы страниц, на которые отображены системные кадры страниц. Для всех записей таблицы страниц, помеченных как доступные, установлен бит PG_referenced. Этот бит устанавливается процессором при первом обращении к странице.

На каждой итерации kswapd он проверяет, установлен ли бит доступа к странице в записи таблицы страниц. Каждый раз при чтении бита доступности страницы kswapd очищает этот бит. Мы можем собрать все шаги управления страницами памяти воедино на рис. 8.24.

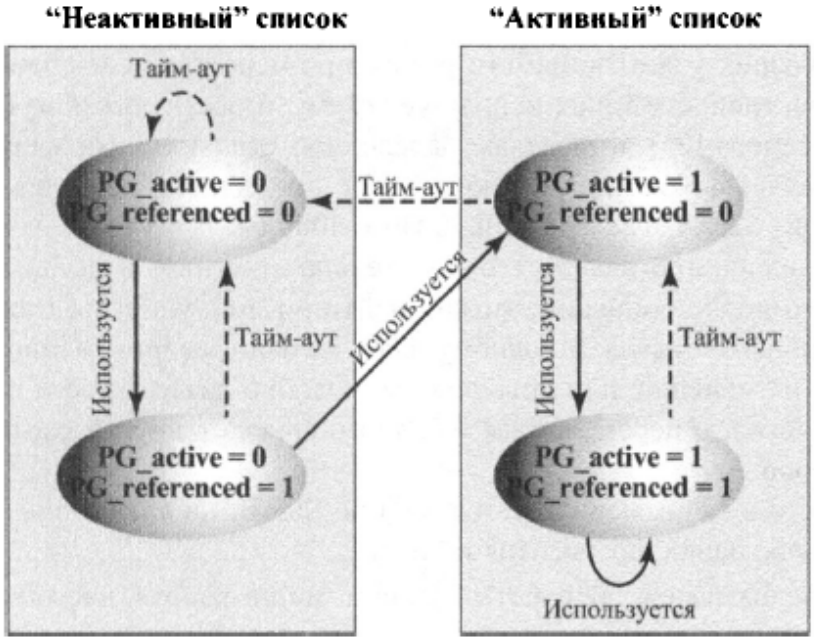


Рис. 8.24. Утилизация страниц в Linux

1. При первом обращении к странице в неактивном списке устанавливается флаг PG_referenced.
2. При следующем обращении к странице она перемещается в активный список. Таким образом, для объявления страницы активной требуется два обращения к ней. Точнее говоря, чтобы страница стала активной, к ней должны быть выполнены два обращения в различных сканированиях.
3. Если второе обращение не произошло достаточно быстро, бит PG_referenced сбрасывается.
4. Аналогично для перемещения активных страниц в список неактивных требуется два тайм-аута. Страницы в списке неактивных доступны для замещения страниц с использованием одной из разновидностей алгоритма последнего использовавшегося.

Распределение памяти ядра

Возможности работы с памятью ядра Linux управляют физическими кадрами страниц основной памяти. Возможные владельцы кадра включают пользовательские процессы (т.е. кадр является частью виртуальной памяти процесса, который в настоящий

момент располагается в реальной памяти), динамически выделенную память для данных ядра, статический код ядра и кеш страниц.

Фундаментом распределения памяти ядра в Linux является механизм распределения страниц, используемый для управления пользовательской виртуальной памятью.

Здесь, как и в схеме виртуальной памяти, используется алгоритм двойников, так что память для нужд ядра может выделяться и освобождаться на уровне страниц. Поскольку минимальное количество памяти, которое может быть выделено таким образом, составляет одну страницу, такое распределение неэффективно в связи с частыми запросами на выделение небольших участков памяти разного размера с малым временем жизни.

Для повышения эффективности Linux использует схему, известную как кусочное распределение (slab allocation) в пределах выделенной страницы. На машинах x86 размер страницы составляет 4 Кбайт, а участки памяти, выделяемые в пределах страницы, могут иметь размеры 32, 64, 128, 252, 508, 2040 и 4080 байт.

Такой SLAB-аллокатор является относительно сложным и детально здесь не рассматривается. По сути, Linux поддерживает множество связанных списков, по одному для участков каждого размера. Участки могут быть разделены на меньшие или объединены, подобно разделению и слиянию блоков в алгоритме двойников, и перемещаться из одного списка в другой соответственно изменению их размеров.

Наиболее часто в Linux используется именно SLAB, но есть и иные аллокаторы для распределения небольших фрагментов памяти.

1. SLAB. Предназначен для работы с кешем, минимизируя, насколько возможно, количество промахов кеша.

2. SLUB (SLAB без очереди). Максимально простой с минимальным количеством команд.

3. SLOB (simple Jist of Ьlocks - простой список блоков). Максимально компактный; предназначен для систем с ограничениями памяти.

РЕЗЮМЕ

Для эффективного использования процессора и систем ввода-вывода желательно поддерживать в основной памяти как можно больше процессов одновременно. Кроме того, желательно освободить программиста от ограничений, накладываемых на размер создаваемых им программ.

Решение обеих задач состоит в использовании виртуальной памяти. При использовании виртуальной памяти все адреса являются логическими, транслируемыми в процессе работы в физические. Это позволяет процессу располагаться в произвольном месте основной памяти, а также изменять свое местоположение в памяти в процессе работы.

Кроме того, виртуальная память позволяет разделить процесс на несколько частей, которые не обязательно должны располагаться в смежных блоках основной памяти, более того - не все они должны находиться в основной памяти во время работы процесса.

Виртуальная память опирается на два основных подхода - страничную организацию и сегментацию.

При страничной организации каждый процесс разделяется на относительно малые страницы фиксированного размера.

Сегментация же позволяет использовать части различного размера. В одной схеме управления памятью могут совместно использоваться и сегментация, и страничная организация.

Схема управления виртуальной памятью требует как программной, так и аппаратной поддержки. Аппаратная поддержка, обеспечиваемая процессором, включает динамическое преобразование виртуальных адресов в физические и генерацию прерывания при отсутствии адресуемой страницы или сегмента в основной памяти. Это прерывание обрабатывается программным обеспечением управления памятью.

При разработке системы управления памятью следует решить множество связанных с ней вопросов, включающих следующие.

• **Стратегия выборки.** Страницы могут загружаться в основную память как по требованию процесса, так и с использованием стратегии предварительной выборки, при которой происходит загрузка страниц кластерами.

• **Стратегия размещения.** В случае выбора системы с использованием только сегментации все вновь загружаемые сегменты должны быть размещены в доступном пространстве в памяти.

• **Стратегия замещения.** При заполнении памяти следует принять решение о том, какая страница (или страницы) будет замещена загружаемыми в память.

• **Управление резидентным множеством.** Операционная система должна решать, какой именно объем памяти должен быть отведен тому или иному процессу при его загрузке в память. Этот объем может быть выделен статически, в момент создания процесса, либо изменяться динамически в процессе работы.

• **Стратегия очистки.** Измененные страницы процесса должны быть записаны при их замещении; однако возможно использование стратегии предварительной очистки, при которой за одну операцию производится запись кластера страниц.

• **Управление загрузкой.** Управление загрузкой заключается в определении количества процессов, которые должны быть резидентны в основной памяти в данный момент.