**27.10.2020 ИС 2К Операционные системы**

1. **Виртуальная память**

В то время как для создания абстракции адресного пространства могут быть использованы базовые и ограничительные регистры, нужно решить еще одну проблему: управления ресурсоемким программным обеспечением. Несмотря на быстрый рост

объемов памяти, объемы, требующиеся программному обеспечению, растут намного быстрее. В 1980-е годы многие университеты работали на машинах VAX, имеющих память объемом 4 Мбайт, под управлением систем с разделением времени, которые одновременно обслуживали с десяток (более или менее удовлетворенных) пользователей. Теперь корпорация Microsoft рекомендует использовать как минимум 2 Гбайт памяти для 64-разрядной Windows 8. Тенденция к использованию мультимедиа предъявляет к объему памяти еще более весомые требования.

Последствия такого развития выразились в необходимости запуска программ, объем которых не позволяет им поместиться в памяти, при этом конечно же возникает потребность в системах, поддерживающих несколько одновременно запущенных программ, каждая из которых помещается в памяти, но все вместе они превышают имеющийся объем памяти. Свопинг — не слишком привлекательный выбор, поскольку обычный диск с интерфейсом SATA обладает пиковой скоростью передачи данных в несколько сотен мегабайт в секунду, а это означает, что свопинг программы объемом 1 Гбайт займет секунды, и еще столько же времени будет потрачено на загрузку другой программы в 1 Гбайт.

Проблемы программ, превышающих по объему размер имеющейся памяти, возникли еще на заре компьютерной эры, правда, проявились они в таких узких областях, как решение научных и прикладных задач (существенные объемы памяти требуются для моделирования возникновения Вселенной или даже для авиасимулятора нового самолета). В 1960-е годы было принято решение разбивать программы на небольшие части, называемые **оверлеями**. При запуске программы в память загружался только администратор оверлейной загрузки, который тут же загружал и запускал оверлей с порядковым номером 0. Когда этот оверлей завершал свою работу, он мог сообщить администратору загрузки оверлеев о необходимости загрузки оверлея 1 либо выше оверлея 0, находящегося в памяти (если для него было достаточно пространства), либо поверх оверлея 0 (если памяти не хватало). Некоторые оверлейные системы имели довольно сложное устройство, позволяя множеству оверлеев одновременно находиться в памяти. Оверлеи хранились на диске, и их свопинг с диска в память и обратно осуществлялся администратором загрузки оверлеев.

Хотя сама работа по свопингу оверлеев с диска в память и обратно выполнялась операционной системой, разбиение программ на части выполнялось программистом в ручном режиме. Разбиение больших программ на небольшие модульные части было очень трудоемкой, скучной и не застрахованной от ошибок работой. Преуспеть в этом деле удавалось далеко не всем программистам. Прошло не так много времени, и был придуман способ, позволяющий возложить эту работу на компьютер.

Изобретенный метод (Fotheringham, 1961) стал известен как **виртуальная память**. В основе виртуальной памяти лежит идея, что у каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые **страницами**. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы одновременное присутствие в памяти всех страниц необязательно. Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету. Когда программа ссылается на часть своего адресного пространства, которое *не находится* в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду.

**Тема Память**

В некотором смысле виртуальная память является обобщением идеи базового и ограничительного регистров. У процессора 8088 было несколько отдельных базовых регистров (но не было ограничительных регистров) для текста и данных программы. При использовании виртуальной памяти вместо отдельного перемещения только сегмента текста или только сегмента данных программы на физическую память в сравнительно небольших блоках может быть отображено все адресное пространство. Реализация виртуальной памяти будет показана далее.

Виртуальная память неплохо работает и в многозадачных системах, когда в памяти одновременно содержатся составные части многих программ. Пока программа ждет считывания какой-либо собственной части, центральный процессор может быть отдан другому процессу.

1. **Страничная организация памяти**

Большинство систем виртуальной памяти используют технологию под названием **страничная организация памяти** (paging), к описанию которой мы сейчас и приступим. На любом компьютере программы ссылаются на набор адресов памяти. Когда программа выполняет команду

MOV REG,1000

она осуществляет копирование содержимого ячейки памяти с адресом 1000 в *REG* (или наоборот, в зависимости от компьютера). Адреса могут генерироваться с использованием индексной адресации, базовых регистров, сегментных регистров и другими способами.

Такие сгенерированные программным способом адреса называются **виртуальными адресами**, именно они и формируют **виртуальное адресное пространство**. На компьютерах, не использующих виртуальную память, виртуальные адреса выставляются непосредственно на шине памяти, что приводит к чтению или записи слова физической памяти с таким же адресом. При использовании виртуальной памяти виртуальные адреса не выставляются напрямую на шине памяти. Вместо этого они поступают в **диспетчер памяти** (Memory Management Unit (**MMU**)), который отображает виртуальные адреса на адреса физической памяти (рис. 3.8).

Очень простой пример работы такого отображения показан на рис. 3.9. В этом примере у нас есть компьютер, генерирующий 16-разрядные адреса, от 0 и до 64K - 1. Это виртуальные адреса. Но у этого компьютера есть только 32 Кбайт физической памяти. И хотя для него можно написать программы объемом 64 Кбайт, целиком загрузить в память и запустить такие программы не представляется возможным. Но полный дубликат содержимого памяти всей программы, вплоть до 64 Кбайт, может размещаться на диске, позволяя вводить ее по частям по мере надобности.

Виртуальное адресное пространство состоит из блоков фиксированного размера, называемых **страницами**. Соответствующие блоки в физической памяти называются **страничными блоками**. Страницы и страничные блоки имеют, как правило, одинаковые размеры. В нашем примере их размер составляет 4 Кбайт, но в реальных системах используются размеры страниц от 512 байт до 1 Гбайт. При наличии 64 Кбайт виртуального адресного пространства и 32 Кбайт физической памяти мы получаем 16 виртуальных страниц и 8 страничных блоков. Перенос информации между оперативной памятью и диском всегда осуществляется целыми страницами. Многие

процессоры поддерживают несколько размеров страниц, которые могут быть смешаны и подобраны по усмотрению операционной системы. Например, архитектура х86-64 поддерживает страницы размером 4 Кбайт, 2 Мбайт и 1 Гбайт, поэтому для пользовательских приложений можно использовать страницы размером 4 Кбайт, а для ядра — одну страницу размером 1 Гбайт. Почему иногда лучше использовать одну большую страницу, а не много маленьких, будет объяснено позже.

На рис. 3.9 приняты следующие обозначения. Диапазон, помеченный 0К-4К, означает, что виртуальные или физические адреса этой страницы составляют от 0 до 4095. Диапазон 4К-8К ссылается на адреса от 4096 до 8191 и т. д. Каждая страница содержит строго 4096 адресов, которые начинаются с чисел, кратных 4096, и заканчиваются числами на единицу меньше чисел, кратных 4096.

К примеру, когда программа пытается получить доступ к адресу 0, используя команду МОУ 1\*Еб,0

диспетчеру памяти посылается виртуальный адрес 0. Диспетчер видит, что этот виртуальный адрес попадает в страницу 0 (от 0 до 4095), которая в соответствии со своим отображением представлена страничным блоком 2 (от 8192 до 12 287). Соответственно, он трансформируется в адрес 8192, который и выставляется на шину. Память вообще не знает о существовании диспетчера и видит только запрос на чтение или запись по адресу 8192, который и выполняет. Таким образом диспетчер памяти эффективно справляется с отображением всех виртуальных адресов между 0 и 4095 на физические адреса от 8192 до 12 287.

Аналогично этому команда МОУ 1\*Еб,8192

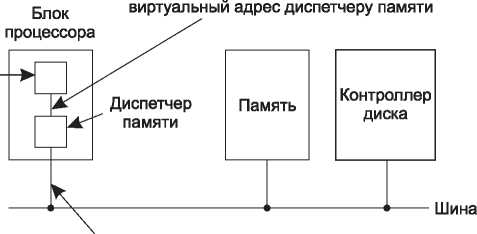
эффективно преобразуется в МОУ 1\*Еб,24576

поскольку виртуальный адрес 8192 (в виртуальной странице 2) отображается на 24 576 (в физической странице 6). В качестве третьего примера виртуальный адрес 20 500 отстоит на 20 байт от начала виртуальной страницы 5 (виртуальные адреса от 20 480 до 24 575) и отображается на физический адрес 12 288 + 20 = 12 308.

**Центральный**

**процессор**

**Центральный процессор передает**



**Диспетчер памяти посылает физический адрес в память**

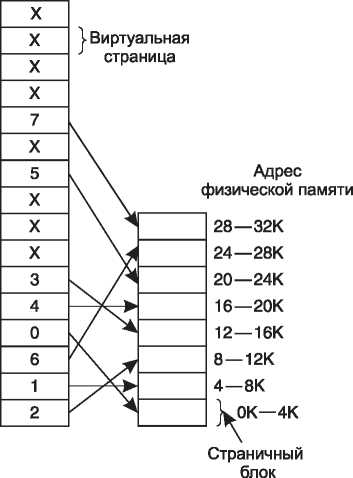
Рис. 3.8. Расположение и предназначение диспетчера памяти. Здесь он показан в составе  
микросхемы центрального процессора, как это чаще всего и бывает в наши дни. Но логически  
он может размещаться и в отдельной микросхеме, как было в прошлом

Рис. 3.9. Связь между виртуальными адресами и адресами физической памяти, получаемая  
с помощью таблицы страниц. Каждая страница начинается с адресов, кратных 4096,  
и завершается на 4095 адресов выше, поэтому 4К-8К на самом деле означает  
4096-8191, а 8К-12К означает 8192-12 287

**Виртуальное  
адресное пространство**

**60—64К  
56—60К  
52—56К  
48—52К  
44—48К  
40 —44К  
36—40К  
32—36К  
28 —32К  
24—28К  
20—24К  
16—20К  
12 —16К  
8— 12К  
4—8К**

1. **4К**



Сама по себе возможность отображения 16 виртуальных страниц на 8 страничных блоков за счет соответствующей настройки таблиц диспетчера памяти не решает проблемы превышения объема виртуальной памяти над объемом физической памяти. Поскольку в нашем распоряжении только 8 физических страничных блоков, то на физическую память могут отображаться только 8 виртуальных страниц (рис. 3.9). Остальные, отмеченные на рисунке крестиками, в число отображаемых не попадают. Реальное оборудование отслеживает присутствие конкретных страниц в физической памяти за счет **бита присутствия-отсутствия**.

А что происходит, если, к примеру, программа ссылается на неотображаемые адреса с помощью команды

MOV REG,32780

которая обращается к байту 12 внутри виртуальной страницы 8 (которая начинается с адреса 32 768)? Диспетчер памяти замечает, что страница не отображена (поскольку она на рисунке помечена крестиком), и заставляет центральный процессор передать управление операционной системе. Это системное прерывание называется **ошибкой отсутствия страницы** (page fault). Операционная система выбирает редко используемый страничный блок и сбрасывает его содержимое на диск (если оно еще не там). Затем она извлекает (также с диска) страницу, на которую была ссылка, и помещает ее в только что освободившийся страничный блок, вносит изменения в таблицы и заново запускает прерванную команду.

К примеру, если операционная система решит выселить содержимое страничного блока 1, она загрузит виртуальную страницу 8 начиная с физического адреса 4096 и внесет два изменения в карту диспетчера памяти. Сначала в запись о виртуальной странице 1 будет внесена пометка о том, что эта страница не отображена, чтобы при любом будущем обращении к виртуальным адресам в диапазоне от 4096 до 8191 вызывалось системное прерывание. Затем крестик в записи, относящейся к виртуальной странице 8, будет изменен на цифру 1, поэтому при повторном выполнении прерванной команды произойдет отображение виртуального адреса 32 780 на физический адрес 4108 (4096 + 12).

Теперь рассмотрим внутреннее устройство диспетчера памяти, чтобы понять, как он работает и почему мы выбрали размер страницы, кратный степени числа 2. На рис. 3.10 показан пример виртуального адреса 8196 (0010000000000100 в двоичной записи), отображенного с использованием карты диспетчера памяти с рис. 3.9. Входящий 16-разрядный виртуальный адрес делится на 4-битный номер страницы и 12-битное смещение. Выделяя 4 бита под номер страницы, мы можем иметь 16 страниц, а с 12 битами под смещение можем адресовать все 4096 байт внутри страницы.

Номер страницы используется в качестве индекса внутри **таблицы страниц** для получения номера страничного блока, соответствующего виртуальной странице. Если бит



**Физический адрес на выходе (24580)**

**Виртуальный адрес  
на входе (8196)**

Рис. 3.10. Преобразование диспетчером памяти виртуального адреса в физический

для 16 страниц по 4 Кбайт

присутствия-отсутствия установлен в 0, то вызывается системное прерывание. Если бит установлен в 1, из таблицы страниц берется номер страничного блока, который копируется в старшие 3 бита выходного регистра вместе с 12-битным смещением, которое копируется в неизменном виде из входящего виртуального адреса. Вместе они формируют 15-разрядный физический адрес. Затем значение выходного регистра выставляется на шине памяти в качестве физического адреса.

1. **Таблицы страниц**

При простой реализации отображение виртуальных адресов на физические может быть сведено к следующему: виртуальный адрес делится на номер виртуальной страницы (старшие биты) и смещение (младшие биты). К примеру, при 16-разрядной адресации и размере страниц 4 Кбайт старшие 4 бита могут определять одну из 16 виртуальных страниц, а младшие 12 бит — смещение в байтах (от 0 до 4095) внутри выбранной страницы. Но для страницы также можно выделить 3, или 5, или какое-нибудь другое количество битов. Различные варианты выделения подразумевают различные размеры страниц.

Номер виртуальной страницы используется в качестве индекса внутри таблицы страниц, который нужен для поиска записи для этой виртуальной страницы. Из записи в таблице страниц берется номер страничного блока (если таковой имеется). Номер страничного блока присоединяется к старшим битам смещения, заменяя собой номер виртуальной страницы, чтобы сформировать физический адрес, который может быть послан к памяти.

Таким образом, предназначение таблицы страниц заключается в отображении виртуальных страниц на страничные блоки. С математической точки зрения таблица страниц — это функция, в которой в качестве аргумента выступает номер виртуальной страницы, а результатом является номер физического блока. При использовании результата этой функции поле виртуальной страницы в виртуальном адресе можно заменить полем страничного блока, формируя таким образом адрес физической памяти.

Нас интересует только виртуальная память, а не полная виртуализация. Иными словами, нам пока не до виртуальных машин, что каждая виртуальная машина требует собственной виртуальной памяти, а в результате организация таблицы страниц становится гораздо сложнее, при этом привлекаются теневые или вложенные таблицы страниц, и не только это. Но как мы увидим далее, даже без таких загадочных конфигураций подкачка и виртуальная память остаются довольно сложными технологиями.

**Структура записи в таблице страниц**

Давайте перейдем от общего рассмотрения структуры таблицы страниц к подробностям отдельной записи в этой таблице. Точный формат записи сильно зависит от конструкции машины, но вид присутствующей в ней информации примерно одинаков для всех машин. На рис. 3.11 показан пример записи в таблице страниц. Размер варьируется от компьютера к компьютеру, но обычно он составляет 32 бита. Наиболее важным является поле *номера страничного блока* (Page frame number). В конечном счете цель страничного отображения и состоит в выдаче этого значения. Следующим по значимости является бит *присутствия-отсутствия.* Если он установлен в 1, запись имеет смысл и может быть использована. А если он установлен в 0, то виртуальная страница, которой принадлежит

эта запись, в данный момент в памяти отсутствует. Обращение к записи таблицы страниц, у которой этот бит установлен в 0, вызывает ошибку отсутствия страницы.

Биты *защиты* сообщают о том, какого рода доступ разрешен. В простейшей форме это поле состоит из 1 бита со значением 0 для чтения-записи и значением 1 только для чтения. При более сложном устройстве имеется 3 бита, по одному для разрешения чтения, записи и исполнения страницы.

Биты *модификации* и *ссылки* отслеживают режим использования страницы. Когда в страницу осуществляется запись, аппаратура автоматически устанавливает бит *модификации.* Этот бит имеет значение, когда операционная система решает регенерировать страничный блок. Если содержащаяся в нем страница подвергалась модификации (то есть является измененной), ее нужно сбросить обратно на диск. Если же она не подвергалась модификации (то есть является неизмененной), от нее можно отказаться, поскольку ее дисковая копия не утратила актуальности. Этот бит иногда называется **битом изменения**, поскольку он отражает состояние страницы.

**Блокирование**

**кэша Изменение Присутствие/отсутствие**

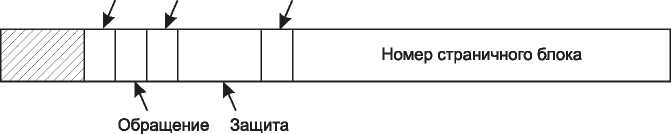


Рис. 3.11. Типичная запись таблицы страниц

Бит *ссылки* устанавливается при обращении к странице как для чтения, так и для записи. Он призван помочь операционной системе выбрать выселяемую страницу при возникновении ошибки отсутствия страницы. Страницы, к которым не было обращений, являются более предпочтительными кандидатами, чем востребуемые, и этот бит играет важную роль в ряде алгоритмов замещения страниц, которые будут рассмотрены далее в этой главе.

И наконец, оставшийся бит позволяет блокировать кэширование страницы. Эта возможность актуальна для тех страниц, которые отображаются на регистры устройств, а не на память. Если операционная система вошла в цикл ожидания отклика какого- нибудь устройства ввода-вывода на только что выданную ею команду, очень важно, чтобы аппаратура продолжала извлечение слова из устройства, а не использовала старую копию, попавшую в кэш. Благодаря этому биту кэширование может быть отключено. Те машины, у которых есть отдельное пространство ввода-вывода и которые не используют ввод-вывод с отображением данного пространства в память, в этом бите не нуждаются.

Заметьте, что адрес на диске, который используется для хранения страницы, в таблице страниц не фигурирует. Причина проста. В таблице страниц содержится только та информация, которая нужна оборудованию, чтобы перевести виртуальный адрес в физический. Информация, необходимая операционной системе для обработки ошибки отсутствия страницы, содержится в таблицах программного обеспечения внутри операционной системы. Оборудование в них не нуждается.

Перед более глубоким погружением в вопросы реализации стоит еще раз отметить, что в принципе виртуальная память создает новую абстракцию — адресное пространство,

которое является абстракцией физической памяти точно так же, как процесс является абстракцией физического процессора (ЦПУ). Виртуальная память может быть реализована за счет разбиения виртуального адресного пространства на страницы и отображения каждой страницы на какой-нибудь страничный блок физической памяти или содержания ее (временно) в неотображенном состоянии. Поэтому в этой главе речь идет в основном об абстракции, созданной операционной системой, и о том, как эта абстракция управляется.

1. **Ускорение работы страничной организации памяти**

После того как мы рассмотрели основы виртуальной памяти и страничной организации, настало время углубиться в подробности возможных вариантов реализации. В любой системе со страничной организацией памяти необходимо рассмотреть два основных вопроса.

1. Отображение виртуального адреса на физический должно быть быстрым.
2. Если пространство виртуальных адресов слишком обширное, таблица страниц будет иметь весьма солидный размер.

Первый пункт является следствием того, что отображение виртуальной памяти на физическую должно осуществляться при каждом обращении к памяти. Все команды в конечном счете должны поступать из памяти, и многие из них ссылаются на операнды, которые также находятся в памяти. Следовательно, при выполнении каждой команды необходимо обращаться к таблице страниц один, два или более раз. Если выполнение команды занимает, скажем, 1 нс, то поиск в таблице страниц, чтобы не стать главным узким местом, должен быть произведен не более чем за 0,2 нс.

Второй пункт следует из факта, что все современные компьютеры используют как минимум 32-разрядные виртуальные адреса, но все более обычными для настольных компьютеров и ноутбуков становятся 64-разрядные адреса. При размере страницы, скажем, 4 Кбайт 32-разрядное адресное пространство имеет 1 млн страниц, а 64-разрядное адресное пространство имеет намного больше страниц, чем вам может понадобиться. При 1 млн страниц в виртуальном адресном пространстве таблица страниц должна содержать 1 млн записей. Также следует помнить, что каждому процессу требуется собственная таблица страниц (поскольку у него собственное виртуальное адресное пространство).

Потребность в обширном и быстром отображении страниц является весьма существенным ограничением на пути создания компьютеров. Простейшая конструкция (по крайней мере, концептуально) состоит в использовании одной таблицы страниц, состоящей из массива быстродействующих аппаратных регистров, имеющей по одной записи для каждой виртуальной страницы, проиндексированной по номеру виртуальной страницы (см. рис. 3.10). При запуске процесса операционная система загружает регистры таблицей страниц этого процесса, которая берется из копии, хранящейся в оперативной памяти. Во время выполнения процесса таблице страниц не нужны никакие дополнительные ссылки на память. Преимуществами этого метода являются простота и отсутствие каких-либо обращений к памяти во время отображения. Его недостаток — в чрезмерных затратах при большом размере таблицы страниц, что зачастую просто непрактично. Еще один недостаток заключается в необходимости загрузки всей таблицы страниц при каждом переключении контекста, что полностью убьет производительность.

Другой крайностью является конструкция, при которой вся таблица страниц может целиком находиться в оперативной памяти. При этом аппаратуре нужно иметь лишь один регистр, указывающий на начало таблицы страниц. Такая конструкция позволяет отображению виртуальной памяти на физическую меняться при переключении контекста путем перезагрузки всего лишь одного регистра. Здесь, разумеется, есть и недостаток, поскольку для считывания записей таблицы страниц во время выполнения каждой команды требуется одно или несколько обращений к памяти, что существенно замедляет работу.

**Буферы быстрого преобразования адреса**

Теперь рассмотрим широко используемую систему, призванную ускорить страничную организацию памяти и обрабатывать большие виртуальные адресные пространства, отталкиваясь от прежней схемы. Отправной точкой для большинства оптимизирующих технологий является содержание таблицы страниц в памяти. Потенциально такая конструкция оказывает сильное влияние на производительность. Рассмотрим, к примеру, однобайтную команду, копирующую один регистр в другой. При отсутствии страничной организации эта команда осуществляет лишь одно обращение к памяти для извлечения самой команды. При страничной организации памяти понадобится как минимум еще одно обращение к памяти для доступа к таблице страниц. Поскольку скорость выполнения обычно ограничена скоростью, с которой центральный процессор может извлечь инструкцию и данные из памяти, необходимость осуществления при каждом обращении к памяти двух обращений снижает производительность вдвое. При таких условиях использовать страничную организацию памяти никто не станет.

Разработчики компьютерных систем были знакомы с этой проблемой много лет и наконец придумали решение, которое основывалось на том наблюдении, что большинство программ склонны большинство своих обращений направлять к небольшому количеству страниц, а не наоборот. Поэтому интенсивному чтению подвергается лишь небольшая часть записей таблицы страниц, а остальная часть практически не используется.

Найденное решение состояло в оснащении компьютеров небольшим устройством для отображения виртуальных адресов на физические без просмотра таблицы страниц. Состояние этого устройства, названного **буфером быстрого преобразования адреса** (Translation Lookaside Buffer (**TLB**)), которое иногда еще называют **ассоциативной памятью**, показано в табл. 3.1. Зачастую это устройство находится внутри диспетчера памяти и состоит из небольшого количества записей. В данном примере их 8, но их количество редко превышает 64. Каждая запись содержит информацию об одной странице, включающую номер виртуальной страницы, бит, устанавливающийся при модификации страницы, код защиты (разрешение на чтение, запись и выполнение) и физический страничный блок, в котором расположена страница. Эти поля имеют точное соответствие полям в таблице страниц, за исключением номера виртуальной страницы, который в таблице страниц не нужен. Еще один бит показывает задейство- ванность страницы (то есть используется она или нет).

Пример, который мог бы сформировать TLB, показанный в табл. 3.1, — это циклический процесс, занимающий виртуальные страницы 19, 20 и 21, поэтому их записи в TLB имеют коды защиты, разрешающие чтение и исполнение. Основные данные, используемые в этот момент (скажем, обрабатываемый массив), занимают страницы 129 и 130. Страница 140 содержит индексы, используемые при вычислениях, производимых с элементами массива. И наконец, стек занимает страницы 860 и 861.

**Таблица 3.1. Буфер быстрого преобразования адреса, используемый для ускорения страничного доступа к памяти**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Задействована | Виртуальная  страница | Изменена | Защищена | Страничный  блок |
| 1 | 140 | 1 | RW | 31 |
| 1 | 20 | 0 | R X | 38 |
| 1 | 130 | 1 | RW | 29 |
| 1 | 129 | 1 | RW | 62 |
| 1 | 19 | 0 | R X | 50 |
| 1 | 21 | 0 | R X | 45 |
| 1 | 860 | 1 | RW | 14 |
| 1 | 861 | 1 | RW | 75 |

Теперь рассмотрим работу TLB. Когда диспетчеру памяти предоставляется для преобразования виртуальный адрес, аппаратура сначала проверяет, не содержится ли номер его виртуальной страницы в TLB, одновременно (то есть параллельно) сравнивая его значение со всеми записями. Для этого потребуется специальное оборудование, имеющееся у всех диспетчеров памяти с TLB-буферами. Если будет найдено соответствие и биты защиты не будут препятствовать доступу, номер страничного блока будет взят непосредственно из TLB, без обращения к таблице страниц. Если номер виртуальной страницы присутствует в TLB, но команда пытается осуществить запись в страницу, предназначенную только для чтения, генерируется ошибка защиты.

Нас интересует, что же произойдет, если номер виртуальной страницы не будет найден в TLB. Диспетчер памяти обнаруживает его отсутствие и осуществляет обычный поиск в таблице страниц. Затем он выселяет одну из записей в TLB, заменяя ее только что найденной записью из таблицы страниц. Поэтому, если вскоре эта страница будет востребована снова, то во второй раз она уже будет найдена в TLB. Когда запись удаляется из TLB, бит модификации копируется обратно в таблицу страниц, находящуюся в памяти. Другие значения, за исключением бита ссылки, там уже присутствуют. Когда TLB загружается из таблицы страниц, все поля берутся из памяти.

**Программное управление буфером TLB**

До сих пор мы предполагали, что каждая машина со страничной виртуальной памятью имеет таблицы страниц, распознаваемые аппаратурой, плюс TLB. При такой конструкции управление TLB и обработка TLB-ошибок осуществляется исключительно оборудованием диспетчера памяти. Прерывания, передающие управление операционной системе, происходят, только если страница отсутствует в памяти.

В былые времена такое предположение вполне соответствовало действительности. Но многие современные RISC-машины, включая SPARC, MIPS и (теперь уже ставшие историей) HP PA, осуществляют практически все управление страницами программным образом. На этих машинах записи TLB загружаются операционной системой явным образом. Когда нужная запись в TLB отсутствует, диспетчер памяти, вместо того чтобы обращаться к таблицам страниц для поиска и извлечения сведений о нужной странице, просто генерирует TLB-ошибку и подбрасывает проблему операционной системе. Система должна отыскать страницу, удалить запись из TLB, внести новую запись и перезапустить команду, вызвавшую ошибку. И разумеется, все это долж

но быть сделано с использованием минимума команд, поскольку ошибка отсутствия записи в TLB случается намного чаще, чем ошибка отсутствия страницы в таблице.

Как ни странно, но при умеренно большом TLB (скажем, 64 записи) программное управление этим буфером оказывается вполне эффективным средством снижения количества ошибок отсутствия нужной записи. Основным преимуществом такого управления является существенное упрощение диспетчера памяти, освобождающее большую площадь микросхемы центрального процессора под кэш и другие функциональные узлы, способные повысить производительность.

Для улучшения производительности на машинах, осуществляющих программное управление TLB, в давние времена были разработаны различные стратегии. В одном из подходов прилагались усилия как по сокращению ошибок отсутствия нужных записей в TLB, так и по уменьшению издержек при возникновении этих ошибок (Bala et al., 1994). Иногда для сокращения количества ошибок отсутствия нужных записей в TLB операционная система может воспользоваться своей интуицией для определения того, какие из страниц, скорее всего, будут востребованы в следующую очередь, и заранее загрузить касающиеся их записи в TLB. Например, когда клиентский процесс отправляет сообщение серверному процессу на той же самой машине, высока вероятность того, что сервер вскоре также будет задействован. Располагая этими сведениями во время обработки системного прерывания, занимающегося отправкой *(send),* система может заодно с этим проверить, присутствуют ли в карте отображения страницы кода, данных и стека сервера, и отобразить их до того, как появятся предпосылки для возникновения ошибки TLB.

Обычный способ аппаратной или программной обработки ошибки TLB заключается в переходе к таблице страниц и осуществлении операций индексации для определения местоположения затребованной страницы. Проблема выполнения этого поиска программным способом состоит в том, что страницы, содержащие таблицу страниц, могут отсутствовать в TLB, что может стать причиной дополнительных TLB-ошибок в процессе обработки. От этих ошибок можно избавиться за счет использования довольно большого (например, 4 Кбайт) программного кэша записей TLB в фиксированном месте, сведения о странице которого всегда содержатся в TLB. При первой проверке программного кэша операционная система может существенно сократить количество TLB-ошибок.

При использовании программного управления TLB важно понять разницу между различными видами ошибок отсутствия записей. **Программная ошибка отсутствия** происходит, когда страница, к которой идет обращение, отсутствует в TLB, но присутствует в памяти. Для ее устранения требуется лишь обновление TLB и не требуется выполнение операций ввода-вывода с обращением к диску. Обычно устранение программной ошибки отсутствия требует 10-20 машинных команд и может быть завершено за несколько наносекунд. В отличие от нее **аппаратная ошибка отсутствия** происходит, когда сама страница отсутствует в памяти (и, разумеется, запись о ней отсутствует в TLB). Для получения страницы требуется обращение к диску, занимающее в зависимости от используемого диска несколько миллисекунд. Аппаратная ошибка отсутствия обрабатывается почти в миллион раз медленнее, чем программная. Просмотр отображения в иерархии таблиц страниц называется **просмотром таблиц страниц** (page table walk).

Фактически дела обстоят еще хуже. Ошибка отсутствия носит не только программный или аппаратный характер. Некоторые ошибки отсутствия имеют по сравнению

с другими ошибками отсутствия больше программный (или больше аппаратный) характер. Например, при просмотре таблицы страниц нужная страница в таблице страниц процесса отсутствует, и программа, таким образом, сталкивается с ошибкой отсутствия страницы. В данной ситуации существуют три возможности. Во-первых, страница фактически может находиться в памяти, но отсутствовать в таблице страниц процесса. Например, страница может быт взята с диска другим процессом. В таком случае нужно вместо нового обращения к диску просто соответствующим образом отобразить страницу в таблицах страниц. Это скорее программная ошибка отсутствия страницы, известная как **легкая ошибка отсутствия страницы** (minor page fault). Во-вторых, если страницу нужно брать с диска, это будет считаться серьезной ошибкой отсутствия страницы. В-третьих, вполне возможно, что программа просто обратилась по неверному адресу и в TLB вообще не нужно добавлять никакого отображения. В таком случае операционная система обычно прекращает выполнение программы с выдачей **ошибки сегментации**. И только в этом случае программа действительно делает что-то неправильно. Во всех остальных случаях ценой некоторой потери производительности все автоматически исправляется оборудованием и/или операционной системой.

1. **Таблицы страниц для больших объемов памяти**

Буферы TLB могут использоваться для увеличения скорости преобразования виртуальных адресов в физические при обычной системе хранения таблицы страниц в памяти. Но этим проблемы не исчерпываются. Возникают проблемы обслуживания очень больших виртуальных адресных пространств. Далее будут рассмотрены два способа, позволяющие справиться с этими проблемами.

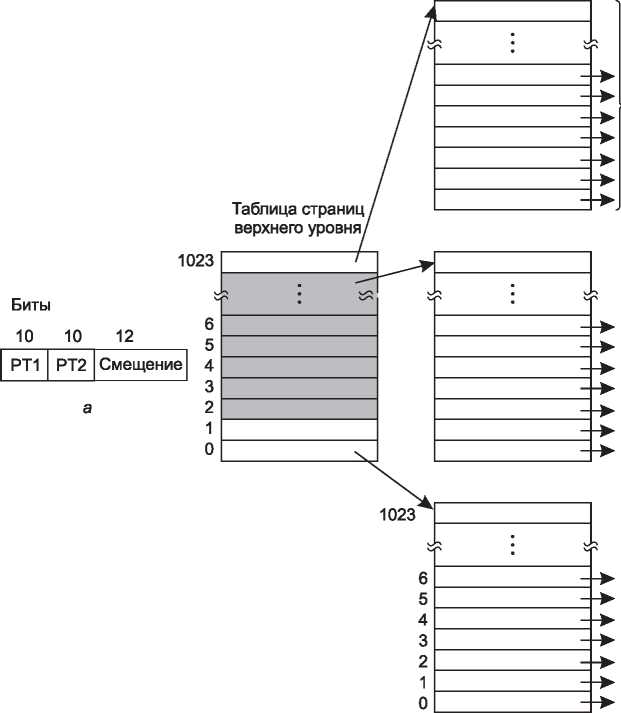
**Многоуровневые таблицы страниц**

В качестве первого подхода рассмотрим использование многоуровневой таблицы страниц, простой пример которой показан на рис. 3.12. В его левой части (на фрагменте *а)* показан 32-разрядный виртуальный адрес, разбитый на 10-битное поле PT1, 10-битное поле PT2 и 12-битное поле смещения. Поскольку под смещение отведено 12 бит, страницы имеют размер 4 Кбайт и их общее количество составляет 220.

Секрет метода использования многоуровневой таблицы страниц заключается в отказе от постоянного хранения всех таблиц страниц в памяти. В частности, вообще не должны храниться те таблицы, в которых нет необходимости. Предположим, к примеру, что процессу требуются 12 Мбайт: нижние 4 Мбайт памяти — для текста программы, следующие 4 Мбайт — для данных и верхние 4 Мбайт — для стека. Между верхней границей данных и дном стека образуется огромная неиспользуемая дыра.

На рис. 3.12, *б* показано, как работает двухуровневая таблица страниц. Слева показана таблица страниц верхнего уровня, содержащая 1024 записи, соотносящиеся с 10-битным полем PT1. Когда диспетчеру памяти предоставляется виртуальный адрес, то сначала он извлекает поле PT1 и использует его значение в качестве индекса для таблицы страниц верхнего уровня. Каждая из этих 1024 записей в таблице страниц верхнего уровня представляет 4 Мбайт, поскольку все 4-гигабайтное (то есть 32-разрядное) виртуальное адресное пространство было разбито на фрагменты по 4096 байт.

**Таблица страниц  
второго уровня**



**Таблица страниц  
для старших  
4 Мбайт памяти**

**К страницам**

*б*

Рис. 3.12. Многоуровневая таблица страниц: *а* — 32-разрядный адрес с двумя полями  
таблиц страниц; *б* — двухуровневая таблица страниц

Из записи, место которой определяется путем индексирования таблицы страниц верхнего уровня, извлекается адрес или номер страничного блока таблицы страниц второго уровня. Запись 0 таблицы страниц верхнего уровня указывает на таблицу страниц для текста программы, запись 1 — на таблицу страниц для данных, а запись 1023 — на таблицу страниц для стека. Другие (закрашенные) записи не используются. Поле PT2 теперь используется в качестве индекса на выбранную таблицу страниц второго уровня, предназначенного для поиска номера страничного блока для самой страницы.

В качестве примера рассмотрим 32-разрядный виртуальный адрес 0x00403004 (4 206 596 в десятичном формате), который соответствует 12 292-му байту внутри области данных. Этот виртуальный адрес соответствует PT1 = 1, PT2 = 2 и смещение равно 4. Диспетчер памяти сначала использует PT1 для обращения по индексу к та

блице верхнего уровня и извлекает запись 1, которая соответствует адресам от 4 Мбайт до 8 Мбайт - 1. Затем он использует PT2 для обращения по индексу к таблице страниц второго уровня, чтобы найти и извлечь запись 3, которая соответствует адресам от 12 288 до 16 383 внутри своего фрагмента размером 4 Мбайт (то есть соответствует абсолютным адресам от 4 206 592 до 4 210 687). Эта запись содержит номер страничного блока той страницы, которая содержит виртуальный адрес 0x00403004. Если эта страница не присутствует в памяти, то бит присутствия-отсутствия в записи таблицы страниц будет иметь нулевое значение, что вызовет ошибку отсутствия страницы. Если страница присутствует в памяти, то номер страничного блока, взятый из таблицы страниц второго уровня, объединяется со смещением (4) для построения физического адреса. Этот адрес выставляется на шину и отправляется к блоку памяти.

В отношении изображения на рис. 3.12 следует отметить одну интересную деталь. Хотя адресное пространство содержит более миллиона страниц, фактически востребованы только четыре таблицы: таблица верхнего уровня и таблицы второго уровня для памяти от 0 до 4 Мбайт - 1 (для текста программы), от 4 Мбайт до 8 Мбайт - 1 (для данных) и для верхних 4 Мбайт (выделенных под стек). Биты присутствия-отсутствия в остальных 1021 записи таблицы страниц верхнего уровня установлены в нуль, что при любом обращении к ним вызовет ошибку отсутствия страницы. При возникновении этой ошибки операционная система поймет, что процесс пытается обратиться к той памяти, обращение к которой не предполагалось, и предпримет соответствующие меры, например пошлет ему сигнал или уничтожит этот процесс. В данном примере мы выбрали для различных размеров округленные значения и размер поля PT1, равный размеру поля PT2, но в реальных системах, конечно, возможны и другие значения.

Система, показанная на рис. 3.12, в которой используется двухуровневая таблица страниц, может быть расширена до трех, четырех и более уровней. Дополнительные уровни придают ей большую гибкость. Например, 32-разрядный процессор Intel 80386 (выпущенный в 1985 году) способен был адресовать до 4 Гбайт памяти, используя двухуровневую таблицу страниц, которая состоит из каталога страниц, чьи записи указывают на таблицы страниц, которые, в свою очередь, указывают на фактические страничные блоки размером 4 Кбайт. Как в каталоге, так и в таблицах страниц содержится по 1024 записи, что в целом, как и требуется, дает 210 • 210 • 212 = 232 адресуемых байтов.

Спустя 10 лет с выпуском процессора Pentium Pro был введен еще один уровень: таблица указателей на каталоги страниц. Кроме всего прочего, каждая запись в каждом уровне иерархии таблиц страниц была расширена с 32 до 64 разрядов, что позволяло адресовать память за пределами 4-гигабайтной границы. Поскольку имелось всего 4 записи в таблице указателей на каталоги страниц, 512 записей в каждом каталоге страниц и 512 записей в каждой таблице страниц, общий объем возможной адресуемой памяти был по-прежнему ограничен максимальным значением 4 Гбайт. Когда же к семейству x86 была добавлена должная 64-разрядная поддержка (изначально это было сделано компанией AMD), дополнительный уровень *можно* было бы назвать указателем таблицы указателей на каталоги страниц. Это вполне вписалось бы в манеру присваивания названий производителями микросхем. К счастью, этого не произошло. И ими был выбран альтернативный вариант «страничное отображение уровня 4» (page map level 4) — конечно, имя не самое броское, зато короткое и более понятное. В любом случае, теперь эти процессоры используют все 512 записей во всех таблицах, выдавая объем адресуемой памяти 29 • 29 • 29 • 29 • 212 = 248 байтов. Они могли бы добавить и еще один уровень, но, возможно, подумали, что 256 Тбайт пока будет достаточно.

**Инвертированные таблицы страниц**

Альтернатива постоянно растущим уровням иерархии страничной адресации называется **инвертированными таблицами страниц**. Впервые они использовались такими процессорами, как PowerPC, UltraSPARC и Itanium (которые иногда называли Itanic, поскольку успех, на который в связи с их выходом надеялась компания Intel, так и не был достигнут). В данной конструкции имеется одна запись для каждого страничного блока в реальной памяти, а не одна запись на каждую страницу в виртуальном адресном пространстве. Например, при использовании 64-разрядных виртуальных адресов, страниц размером 4 Кбайт и оперативной памяти размером 4 Гбайт инвертированные таблицы требовали только 1 048 576 записей. В каждой записи отслеживается, что именно находится в страничном блоке (процесс, виртуальная страница).

Хотя инвертированные таблицы страниц экономят значительное количество пространства, по крайней мере в том случае, когда виртуальное адресное пространство намного объемнее физической памяти, у них есть один серьезный недостаток: преобразование виртуальных адресов в физические становится намного сложнее. Когда процесс *n* обращается к виртуальной странице р, аппаратура уже не может найти физическую страницу, используя *р* в качестве индекса внутри таблицы страниц. Вместо этого она должна провести поиск записи (n, р) по всей инвертированной таблице страниц. Более того, этот поиск должен быть проведен при каждом обращении к памяти, а не только при ошибках отсутствия страницы. Вряд ли можно признать просмотр таблицы размером 256 K записей при каждом обращении к памяти способом сделать ваш компьютер самым быстродействующим.

Решение этой дилеммы состоит в использовании TLB. Если в этом буфере можно будет хранить информацию обо всех интенсивно используемых страницах, преобразование может происходить так же быстро, как и при использовании обычных таблиц страниц. Но при отсутствии нужной записи в TLB программа должна просмотреть инвертированную таблицу страниц. Одним из приемлемых способов осуществления этого поиска является ведение хэш-таблицы, созданной на основе виртуальных адресов. На рис. 3.13 показано, что все находящиеся на данный момент в памяти виртуальные

**Традиционная таблица страниц с ячейкой для каждой из 252 страниц**

**256 Мбайт физической  
памяти имеют 216 страничных  
блоков размером 4 Кбайт**

**Хэш-таблица**

Индексация  
по виртуальным  
страницам

Индексация  
по значению  
хэш-функции  
на виртуальной  
странице

**Виртуальная**

**страница**

**Страничный**

**блок**

Рис. 3.13. Сопоставление традиционной таблицы страниц с инвертированной

страницы, имеющие одинаковые хэш-значения, связываются в одну цепочку. Если у хэш-таблицы столько же строк, сколько физических страниц у машины, средняя цепочка будет длиной всего лишь в одну запись, позволяя существенно ускорить отображение. Как только будет найден номер страничного блока, в TLB будет введена новая пара значений (виртуального, физического).

Инвертированные таблицы страниц нашли широкое применение на 64-разрядных машинах, поскольку даже при очень больших размерах страниц количество записей в обычных таблицах страниц будет для них просто гигантским. К примеру, при размере страниц 4 Мбайт и 64-разрядных виртуальных адресах понадобится 242 записей в таблице страниц.