Виртуальная память.

Практически любой программе для работы нужна оперативная память. Пока потребности невелики и программе хватает той памяти, что готова предоставить операционная система, никаких проблем нет. Но что делать, если потребность в памяти больше, чем доступно в моменте или даже больше, чем общий объём оперативной памяти в системе ? Начинаются ухищрения.

1) Оверлеи (overlay). Если тело программы достаточно велико и в ней можно выделить независимые модули, обычно не все эти модули должны быть загружены постоянно, по возможности часть из них можно выгружать и загружать вновь по требованию. Следит за этим центральный модуль.
2) Хранение части не нужных в данный момент данных в более медленной памяти - на диске, на магнитной ленте, магнитном барабане. Программа сама решает что ей требуется в данный момент и загружает/выгружает данные по мере необходимости. Это называется кэширование[1]. В сущности, все СУБД (системы управления базами данных) работают так. Однако, далеко не все программы так сложны как СУБД, но многим из них требуется много памяти. Приходится продумывать логику работы с данными и это требует немалых ресурсов на создание - отладку - поддержание такой логики.

Проблема нехватки памяти общая, но каждая программа была вынуждена решать ее самостоятельно с соответствующими издержками. Поэтому уже давно программисты начали задумываться над тем, как создать универсальный механизм кэширования памяти. Этот механизм называется “виртуальная память”, термин ввёл Фриц Рудольф Гюнтш (Fritz-Rudolf Güntsch) в 1956 г. Находится этот механизм в ядре операционной системы.

Пусть у нас есть 10 страниц[2] быстрой памяти и 100 страниц медленной. К каждой из быстрых страниц приписан номер медленной страницы, на которую она смотрит. При этом обратиться можно к любой из медленных страниц. Если в данный момент у нее нет быстрой пары, какую-то из быстрых страниц придётся освободить (обычно ту, что не использовалась дольше всего, для этого придётся поддерживать статистику использования страниц), записав её содержимое в медленную память, если она была изменена. После чего загрузить новую страницу и прописать новый номер.

Невозможно (слишком дорого) перед каждым обращением к памяти проверять доступность страницы чтобы запустить описанный механизм. Поэтому это делается аппаратно. При обращении к памяти в случае отсутствия нужной страницы инициируется аппаратное прерывание, в процессе обработки которого операционная система подгружает нужную страницу. После выхода из прерывания, программа как ни в чем ни бывало продолжает работу.

---

[1] от cash - наличные, аналогия с деньгами, часть которых лежит в кармане, а остальные в банке, куда потребуется идти, если карманных не хватит
[2] страница - область памяти обычно фиксированного размера (4 килобайта), тот квант информации, которыми обмениваются между собой быстрая и медленная области памяти

Адресация в медленных страницах называется **виртуальной**. Адресация в быстрых страницах - **физической**.

Так было на момент появления механизма, сейчас, конечно, всё немного усложнилось. Виртуальная память больше не имеет прямого отображения в медленную (в качестве которой обычно выступает диск). Пока страниц было мало, к каждой можно было привязать адрес на диске, но со временем потребовался механизм перекодировки.Виртуальные пространства разных процессов теперь смотрят в единое дисковое пространство, которое называется файлом подкачки (swap file). Раньше частенько у каждой программы был собственный файл подкачки. Виртуальные пространства стали настолько велики, что требуется как минимум двухуровневый механизм с собственным кэшем для перекодировки виртуального адреса в физический. Но принципиально ничего не изменилось.

Описанный механизм конвертации называется **MMU** (memory management unit), в некоторых микроконтроллерах [3]его нет, как нет и виртуальной памяти - виртуальные адреса тождественны физическим.

Немного надо сказать о так называемой “модели памяти”. Существует определённая путаница, когда при разговоре о виртуальной памяти возникает термин “сегментированная память”.

Сегментированная.

В этой модели для обращения к памяти требуется два значения - база и смещение от неё. Как ни странно, первая коммерчески доступная система с виртуальной памятью (Burroughs B5000) использовала именно сегментированный подход.

Такой на первый взгляд странный метод доступа к памяти нужен был чтобы обойти ограничения аппаратуры. Например, в процессоре Intel 8086 регистры 16-разрядные, но программе доступен 1 Мб адресного пространства (т.е. шина данных 20 разрядов). В архитектуру помимо регистров общего назначения введены так называемые сегментные регистры: CS - сегмент кода, служит для вычисления адресов функций, переходов …, DS - сегмент данных, SS - сегмент стека и ES - для прочих нужд. При необходимости, например, загрузить слово из памяти в регистр, использовалась команда

```
mov [ds:ax], 42
```

которая вычисляла настоящий адрес как

```
addr = ds * 16 + ax + 42
```

Умножение на 16 - это сдвиг базового адреса вверх на 4 разряда.
Так на 16-разрядной архитектуре удавалось пользоваться 20-разрядной шиной данных.

[3] микроконтроллер - относительно простой микропроцессор для промышленной автоматизации, содержит минимум функциональных устройств, часто не содержит MMU (без виртуальной памяти)

Странично организованная сегментированная виртуальная память требует наличия при каждом сегменте механизма перекодировки из виртуального адреса в физический/адрес на диске. Поддерживать для диска непрерывное следование страниц для каждого сегмента (что позволило бы обойтись банальными сдвигами) невозможно из-за (нарастающей в этом случае) фрагментации[4] файла подкачки. Этот механизм должен быть применим и к большим сегментам и к маленьким, что влечет избыточность и перерасход ресурсов. Кроме того, задача вытеснения на диск части данных становится весьма непростой, учитывая, что статистика использования распределена по многочисленным сегментам.

В результате на данный момент из систем с сегментированной памятью более-менее активно используется только AS/400 от IBM. Последовавшая за ней архитектура POWER поддерживает(вала?) сегментированную память только в режиме совместимости.

В архитектуре x86 сегментированная память присутствовала от рождения. Правда, аппаратная защита памяти[5] появилось только в 80286, при этом ширина адреса выросла с 20 до 24 разрядов (при 16-разрядных регистрах). Впрочем, этот эксперимент был признан неудачным и все дальнейшие процессоры семейства имели режим плоской виртуальной памяти (сегментированная память только в режиме совместимости).

Плоская.

Плоская память подразумевает что к любому месту памяти мы можем обратиться по единому адресу (с точностью до выравнивания[6]).

Плоскую память можно рассматривать как вырожденный случай сегментированной - единственный сегмент во всё виртуальное адресное пространство. А значит и единый механизм перекодировки адресов.

Все более-менее распространённые современные архитектуры используют плоскую модель памяти, то же касается операционных систем. В частности, ядро Linux вообще никогда не поддерживало сегментирование памяти.

Итак, как же всё это работает?
Мы здесь не будем рассматривать кэширование памяти, хотя оно довольно сильно

---

[4] при выделении/освобождении памяти в едином пространстве адресов (в том числе в файле подкачки) неизбежно накапливаются неиспользуемые области памяти, которые замедляют работу и затрудняют выделение больших областей. Это называется фрагментация памяти. Бороться с ней можно, но приводит к перерасходу памяти.

[5] аппаратная защита памяти - во многих архитектурах на сегмент/страницу памяти можно установить атрибуты защиты (например - только для чтения), при попытке нарушения возникает аппаратное прерывание, т.е. исключительная ситуация, которая обрабатывается ядром ОС

[6] выравнивание - в некоторых архитектурах прочитать/записать слово в/из памяти можно только по адресу, кратному размеру слова.

связано с перекодировкой адресов. Сделаем это позже.

TLB (translation lookaside buffer)

На первом этапе делается проверка на наличие информации по предъявленному адресу в структуре под названием TLB (таблица трансляции адресов, translation lookaside buffer). Это небольшой кэш, в котором хранятся пары недавно протестированных адресов (виртуальный => физический).

Архитектура процессора может допускать несколько размеров страниц, например, для SPARC V8 это 4K, 256K и 16Mb. В некоторых системах страницы всех размеров обрабатываются в едином TLB, в других на каждый размер страницы имеет свой TLB (запросы к которым выполняются параллельно, заранее ведь неизвестно какого размера страница). При этом, очевидно, чем больше размер страницы тем меньше размер TLB. Типичный размер TLB для 4K страниц - 512 элементов.

И, как водится, в таком довольно простом механизме полно подводных камней.

- у разных процессов[7] (не потоков[8], threads) виртуальные адресные пространства пересекаются, поэтому одного лишь адреса недостаточно, каждая запись в TLB дополнительно содержит идентификатор ее процесса. Соответственно, при конвертации адреса требуется совпадение и процесса (с идентификатором текущего процесса, PCID) тоже. В случае x86-64 (начиная с Westmere microarchitecture, 2010) размер PCID составляет 12 разрядов. Для сравнения в DEC Alpha-21264 (1995) ASN (address space number, аналог PCID) - 8 разрядов.
- в случае наличия нескольких ядер (core) в процессоре, каждое из них должно обладать своим TLB и, строго говоря, своим MMU.
- в некоторых архитектурах разделяют TLB для данных (dTLB) и кода (iTLB), что логично, учитывая разные сценарии использования.
- при смене на ядре выполняемого процесса (не потока), все принадлежащие старому процессу записи в TLB должны быть вычищены. Именно поэтому операционная система старается запускать разные потоки одного процесса на одних и тех же ядрах.
- некоторые записи достаточно важны для того, чтобы предохранить их от стирания в кэше TLB, в AMD64 для этого предназначен механизм глобальных страниц (global pages) - разряд (8) в записи TLB. Механизм global pages появился раньше PCID и их действия отчасти пересекаются.

В современных процессорах TLB бывают и в несколько уровней (аналогично кэш - памяти), например, в Haswell (x86-64 2013) для 4K страниц: L1 dTLB(64), L1 iTLB(128), L2 dTLB(1024) элементов.

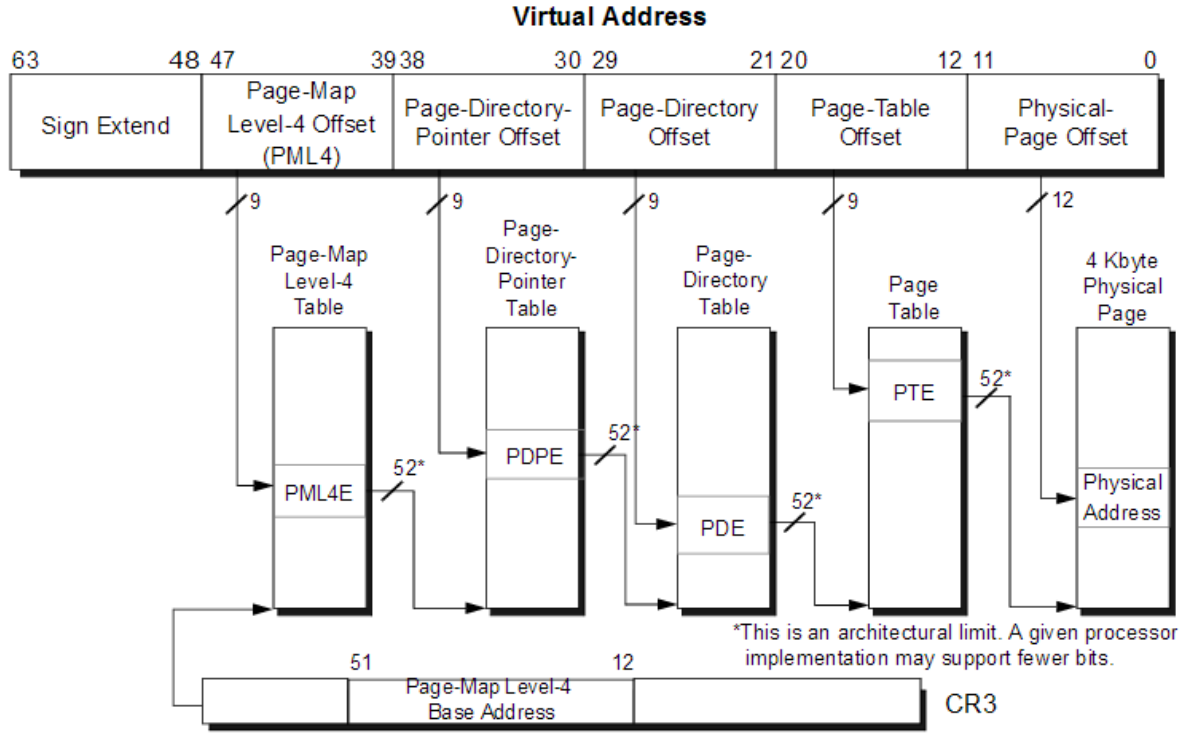
---

[7] процесс - программа, исполняемая операционной системой

[8] поток - во многих ОС процесс может параллельно выполняться на нескольких ядрах в так называемых потоках (threads). Процесс сам создаёт потоки, указывая через системный вызов процедуры для исполнения

Page walk

При промахе в кэше TLB, приходится обращаться в древовидную[9] структуру под названием “таблица страниц” (page table). Эта процедура называется “page walk”. Рассмотрим на примере AMD64. В обычном, так называемом “длинном режиме” (64 разрядном) допускается работа со страницами виртуальной памяти трёх размеров - 4K, 2Mb и 1Gb.

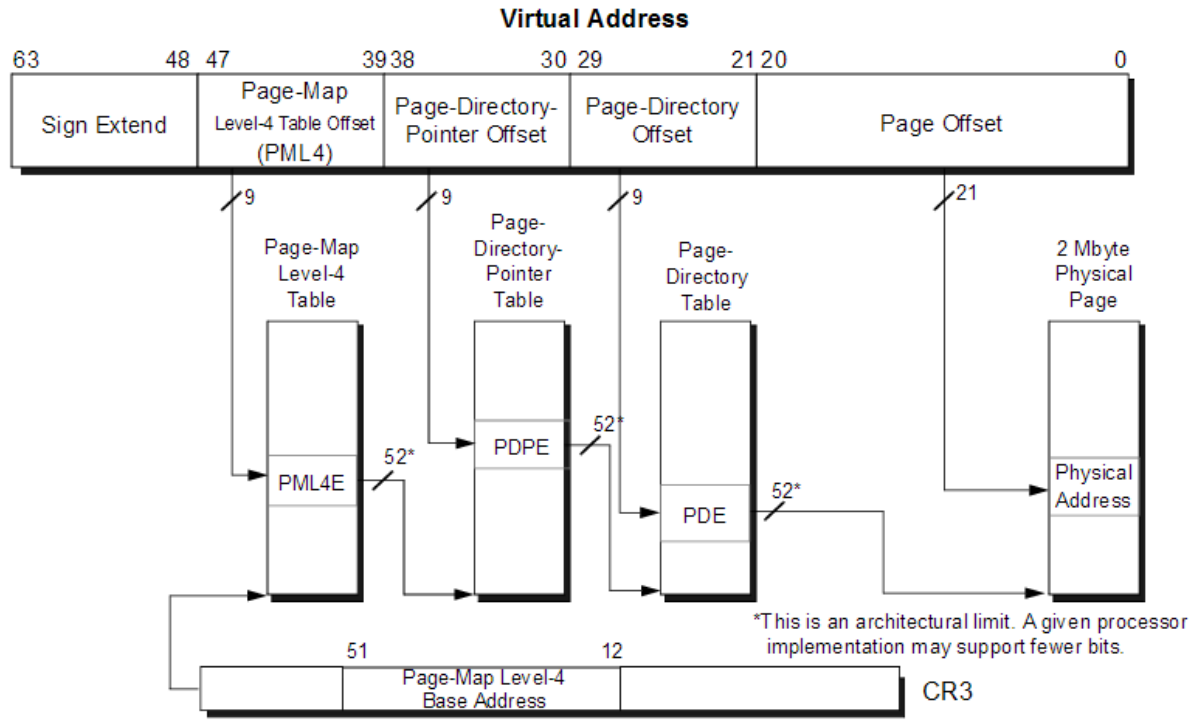


Фиг.1 Трансляция 4K страниц, AMD64

Вершина дерева находится в регистре CR3, используются только 48 младших разрядов виртуального адреса из 64. При работе с 4K страницами дерево полностью сбалансировано и имеет 4 уровня страниц. Группы по 9 разрядов используются как индекс в странице (дерева) соответствующего уровня (512 x 8 = 4K на страницу). Листовая страница (PTE) ссылается на искомую страницу физической памяти, которая расширяется 12 разрядами сдвига внутри страницы. Результирующий физический адрес ограничен 52 разрядами.

Но как же так, при начале трансляции виртуального адреса мы не знаем на страницу какого типа он указывает, что делать, если физическая страница имеет фактический размер 2M?

[9] дерево - распространенная структура данных, состоящая из элементов, у которых может быть несколько потомков, но только один родитель. Растёт дерево из единственного элемента, называемого корнем (root).



Фиг.2 Трансляция 2M страниц, AMD64

Оказывается, размер 2M выбран не случайно. Разница между 4K и 2M - 512 раз (9 двоичных разрядов), фактически для 2M страниц дерево перекодировки имеет глубину 3, на физические страницы смотрит уровень PDE. Но ведь дерево одно для всех типов страниц, как же их различать? Ответ прост, каждый элемент страниц дерева занимает 64 разряда, под данные используется 52 из них, один из незанятых разрядов (в данном случае это 7 разряд) можно использовать как признак конечного элемента. Остальные разряды также используются, часть из них доступна программно, часть - это флаги защиты, младший разряд - это признак нахождения нижеследующей страницы в памяти (а не в файле подкачки).

Итак, если встречаем единицу в 7-м разряде по мере спуска по дереву, значит нашли результат. Если это произошло на уровне PTE, то имеем дело с 4K страницей и адрес надо дополнить 12 разрядами из исходного виртуального значения. В случае PDE - 2M страница и сдвиг на странице 21 разряд. Ну а в случае PDPE - 1Gb страница и оффсет в ней 30 разрядов.

Так кто же заполняет и поддерживает состояние TLB и дерева (таблицы) страниц? Операционная система. Когда программа обращается за выделением виртуальной памяти нужного размера, ядру ОС следует предпринять примерно следующее:

- выбрать размер страниц, до недавнего времени вариант был только один - 4K
- округлить вверх требуемый размер до размера страницы
- найти в виртуальном адресном пространстве свободный участок подходящего размера, благо, при (пусть даже) 48 разрядном адресном пространстве это не сложно
- найти в физическом адресном пространстве нужное количество подходящих страниц

- соотнести физические и виртуальные страницы
- запомнить эти пары в таблице (дереве) страниц
- внести по крайней мере первую страницу выделенного блока в TLB, раз программа попросила память, значит сейчас ей воспользуется

Хозяйке на заметку. Любая из страниц дерева page table (кроме верхней) может быть выгружена из оперативной памяти на диск. В этом случае её старый адрес в физической памяти не имеет значения, а на его место (в таблице страниц) можно записать её положение в файле подкачки. Это же касается и пользовательских страниц.

Немного посчитаем. Допустим, программа на C попросила выделить (malloc) 4Gb памяти, что совсем не экстраординарный размер. 4Gb - это миллион страниц по 4K плюс две тысячи страниц PTE (+8Mb), которые нужно внести в дерево. Таким образом рост адресного пространства влечет за собой разбухание дерева page table, а это дополнительная нагрузка и на саму память и на её кэш.

Представляется соблазнительным использовать страницы большего чем 4K размера, чтобы сэкономить и на page walk и на TLB. Однако это сопряжено с трудностями, например, при вытеснении на диск придётся записать не 4K, а 2Mb, что заметно дольше. Или что делать, если при поднятии из файла подкачки 1Gb страницы не окажется непрерывного физического пространства в 1Gb (фрагментация), потребуется длительная и неоднозначная (нет гарантий успешности) процедура вытеснения каких-то данных на диск. В результате и в linux и во windows механизм использования больших страниц существовал (locked pages(windows), hugetlbfs(linux)), но требовал усилий от клиентского кода, такую память нельзя было получить malloc-ом, большим страницам было отказано в подкачке ...

THP (transparent huge pages)

В ядре linux-2.6.38 появилась (Andrea Arcangeli, 2012) остроумная техника - прозрачные большие страницы (transparent huge pages). Суть её в следующем:

- при любой возможности ядро старается использовать большие (2Mb) страницы
- при выделении памяти, насколько это возможно, создаются большие страницы
- при чтении 4K страницы из файла подкачки, по возможности создаётся большая страница, при этом объединяются мелкие страницы и упоминание о них удаляется из TLB и page table
- большие страницы подлежат подкачке, но при этом могут расщепляться на маленькие
- системные вызовы mlock, mprotect так же приводят к расщеплению большой страницы на маленькие

Происходит всё это прозрачно для пользовательских программ. Замеры показывают, что за счет сокращения page table и высоты дерева в общем случае имеется выигрыш в производительности порядка 10%. Тем не менее, нередко, такая техника замедляет работу приложений, особенно тех, что рассчитаны (явно или неявно) на работу с мелкими страницами. Так что при необходимости THP можно (и нужно) выключить.

Почему страницы по 4К

Есть простой и неправильный ответ на этот вопрос.
“Потому что i386 32-разрядный и дерево page table у него двухуровневое,
12 (разрядов в 4К странице) **+ 10** (1024 32-разрядных указателей на 4К странице дерева первого уровня) **+ 10** (1024 32-разрядных указателей на 4К странице дерева второго уровня) **= 32**.”

Ответ неправильный, т.к. на i386 свет клином не сошелся.
В SPARC V8 таблица страниц трёхуровневая и распределена так: 8 разрядов в верхнем уровне, по 6 в нижних и 12 на страницу. Тоже 4К.

Правда, SPARC (1987) появился после i386 (1985), а что было до него?
32-разрядный VAX (1977) имел физические страницы по 512 байт. Операционная система BSD группировала их в пакеты по 8 штук, так что с точки зрения программиста страница имела размер в те же 4К.

Еще раньше, PDP-11(1970). 16-разрядное адресное пространство было разделено на 8 областей по 8К, которые формально можно считать страницами.

Еще раньше, Multics (GE-645, 1965) имела два размера страниц - 64 и 1024 36-разрядных слов.
Вообще, кажущаяся нам естественной разрядность компьютеров в степень двойки окончательно сложилась в 70-х годах. Размер страницы в 4К - в начале 80-х.

До этого делали кто во что горазд и размер страницы был предметом научных споров и битв математических моделей. В конце концов пришли к выводу, что 4К - оптимальный размер для задач, типичных для 80-х годов.

С тех пор прошло немало времени, и задачи подросли и системы стали 64-разрядными. Неслучайно разработчики архитектуры ARM ввели два базовых размера страниц - 4К или 64К. Intel патентует таблицу страниц, которая поддерживает и то и другое одновременно. Возможно, нас ждут изменения.