

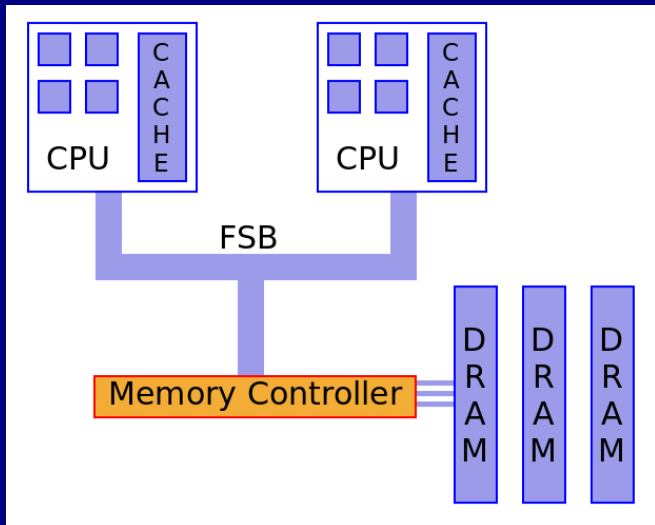
# Operating Systems

## Hardware Memory and Protection

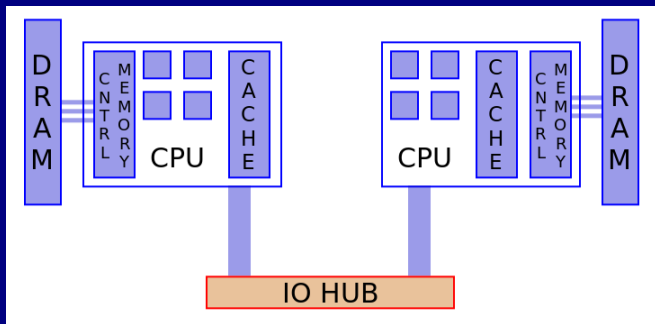
Me

October 4, 2016

# Memory Controller (Old, 200x)



# Memory Controller (New, 201x)



- ▶ CPU видит память через Memory Controller
  - ▶ подключите к Memory Controller хоть чайник, CPU будет считать его памятью.

# Неоднородность памяти

- ▶ Запросы в "память" могут переадресовываться устройствам и иметь побочные эффекты:
  - ▶ например, запись по адресу 0xb8000 приводит к выводу на экран;
  - ▶ обращение по адресу 0xfe00000 переадресуется Local APIC-у;
  - ▶ обращение по адресу 0xfec00000 переадресуется IO APIC-у.
- ▶ Память по каким-то адресам может отсутствовать в принципе:
  - ▶ регионы зарезервированные для устройств отображенных в память и неиспользованные;
  - ▶ ISA Hole, участки зарезервированные для PCI и прочих устройств.

# Карта памяти

- ▶ ОС управляет памятью и должна иметь информацию о доступной памяти:
  - ▶ чтобы настроить алокаторы памяти;
- ▶ разные системы используют разные интерфейсы для получения карты памяти:
  - ▶ BIOS, INT 0x15 функция 0xe820;
  - ▶ UEFI, GetMemoryMap();
  - ▶ multiboot (другой загрузчик) может предоставлять карту памяти;
  - ▶ карта памяти может передаваться ОС как аргумент при старте;
  - ▶ **A tour beyond BIOS memory map in UEFI BIOS.**

# Защита памяти

- ▶ От кого/чего нужно защищать память?
  - ▶ память можно защищать от чтения/записи/исполнения;
  - ▶ память можно защищать от непривилегированного кода;
  - ▶ память одних приложений можно защищать от действий других.
- ▶ Зачем защищать память?
  - ▶ защита от непривилегированного кода защищает память ядра ОС от действий пользовательских приложений - ошибка в приложении не должна приводить к проблемам в ядре;
  - ▶ ошибка в одном приложении не должна вприводить к проблемам в другом.

# Средства защиты памяти

- ▶ Сегментация
  - ▶ сейчас не особо применяется для защиты памяти;
  - ▶ полезно рассмотреть для домашнего задания.
- ▶ Paging
  - ▶ используется для организации трансляции адресов - дает уровень косвенности при работе с памятью;
  - ▶ позволяет защищать память на уровне отдельно взятых "страниц";
  - ▶ является основным инструментом для защиты памяти сейчас;
  - ▶ на x86 используется "совместно" с сегментацией.
- ▶ Другие архитектурно зависимые средства...

# Сегментация на примере x86

## Сегментные регистры

- ▶ В x86 при любом обращении к памяти явно или неявно используется один из сегментных регистров:
  - ▶ DS, ES, CS, SS, FS, GS.
- ▶ Если регистр не указан явно, то регистр выбирается в зависимости от операции:
  - ▶ для обычных чтения и записи данных используются DS или ES;
  - ▶ для операций работы со стеком (push, pop и др.) используется SS;
  - ▶ для чтения следующей инструкции из памяти используется CS;
  - ▶ FS и GS... Ну это отдельная история...



# Сегментация на примере x86

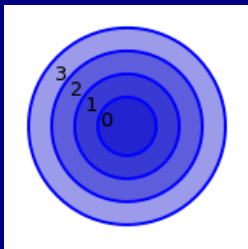
## Селектор сегмента



- ▶ Сегментный регистр хранит 16-битное значение - селектор сегмента:
  - ▶ RPL - Requested Privilege Level (DS, ES, FS, GS, SS);
  - ▶ CPL - Current Privilege Level (CS);
  - ▶ TI - Table Indicator (0 - GDT, 1 - LDT);
  - ▶ Index - индекс записи в таблице (GDT или LDT, в зависимости от TI).

# Сегментация на примере x86

Уровни привилегий 1/2



- ▶ В x86 есть 4 уровня привилегий (aka ring0 - ring3):
  - ▶ 0-ой - наивысший уровень привилегий, на этом уровне можно выполнять любые инструкции и обращаться к любой памяти;
  - ▶ 3-ий - низший уровень, на этом уровне некоторые инструкции не доступны, и доступ к памяти может быть ограничен.

# Сегментация на примере x86

## Уровни привилегий 2/2

- ▶ На уровне 0, *обычно*, работает ядро ОС:
  - ▶ ядро ОС должно иметь полный контроль над системой;
  - ▶ два младших бита в CS у ядра ОС, *обычно*, равны 0;
- ▶ на уровне 3, *обычно*, работают пользовательские приложения:
  - ▶ редакторы текста, браузеры, компиляторы и прочее;
  - ▶ в том числе и QEMU (если оставить в стороне KVM);
  - ▶ два младших бита в CS у приложений, *обычно*, равны 3;
- ▶ уровни 1 и 2 используются редко и в очень специфичных приложениях.

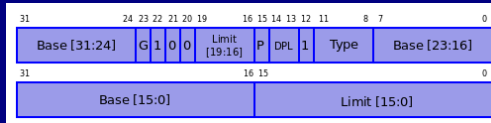
# Сегментация на примере x86

## Сегментные дескрипторы

- ▶ Index - индекс записи в GDT и LDT:
  - ▶ в обеих таблица записи имеют одинаковый формат и значение, т. ч. для нас между ними нет разницы.
- ▶ Смысл записей в GDT/LDT немного меняется в зависимости от режима работы процессора
  - ▶ в Real Mode нет защиты памяти;
  - ▶ в данном контексте нас интересуют только Protected Mode и Long Mode.
- ▶ Дескрипторы в GDT/LDT делятся на системные и все остальные
  - ▶ системные дескрипторы используются для специфичных нужд x86;
  - ▶ в данном контексте нас не интересуют системные дескрипторы.

# Сегментация на примере x86

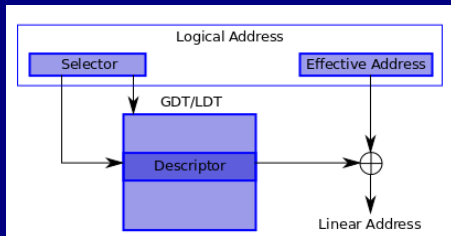
## Сегментные дескрипторы в Protected Mode 1/2



- ▶ Protected Mode дескриптор содержит
  - ▶ Base и Limit - описывают границы региона физической памяти;
  - ▶ DPL - Descriptor Priviledge Level;
  - ▶ TYPE - тип сегмента (сегмент кода/данных, разрешено ли чтение/запись);
  - ▶ P - бит присутствия (должен быть 1 для валидных сегментов);
  - ▶ G - бит гранулярности (1 - Limit в 4Кб блоках, 0 - Limit в байтах).

# Сегментация на примере x86

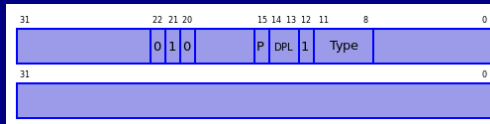
## Сегментные дескрипторы в Protected Mode 2/2



- ▶ Указатель в программе (ака логический адрес) состоит из двух частей:
  - ▶ сегментного регистра (CS, DS, ES, SS, FS, GS);
  - ▶ смещения (ака эффективный адрес).
- ▶ Логический адрес преобразуется в линейный адрес
  - ▶ при преобразовании проверяется, что эффективный адрес не больше Limit.

# Сегментация на примере x86

## Сегментные дескрипторы в Long Mode



- ▶ Long Mode дескриптор содержит почти то же самое что и в Protected Mode
  - ▶ Base и Limit потеряли свой смысл, т. е. трансляция логического адреса в линейный не выполняется и границы сегмента не проверяются;
  - ▶ кроме того нельзя ограничить память, описываемую сегментным дескриптором, т. е. сегментация как самостоятельный способ защиты теряет свой смысл;
  - ▶ кроме регистров FS и GS.

# Сегментация на примере x86

## Проверка уровня привилегий при доступе к данным

- ▶ При загрузке селектора в сегментный регистр (DS, ES, FS, GS) CPU выполняет проверку уровня привилегий:
  - ▶ в проверке участвуют текущий CPL, RPL в значении селектора и DPL в дескрипторе сегмента, соответствующего селектору;
  - ▶ DPL должен быть численно больше или равен и CPL и RPL, в противном случае генерируется исключение;
  - ▶ ОС задает селекторы приложения, и приложение очень ограничено в их изменении.
- ▶ Для сегмента стека SS проверка немного другая:
  - ▶ RPL и CPL должны совпадать с DPL;
  - ▶ на практике для данных и стека часто используют один дескриптор.

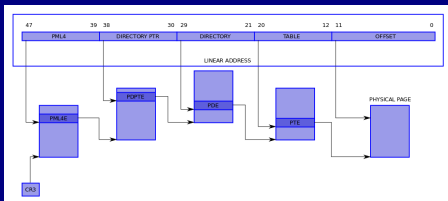


# Страничная организация памяти

- ▶ Страничная организация памяти (Paging) -  
постраничная трансляция адресов
  - ▶ физическое адресное пространство разбивается на страницы;
  - ▶ логическое/линейное адресное пространство так же разбивается на страницы;
  - ▶ ОС настраивает отображение страниц логического/линейного адресного пространства на страницы физического;
  - ▶ CPU транслирует логический/линейный адрес в физический пользуясь отображением при обращениях в память;
  - ▶ в отображении так же можно задать уровень привилегий доступа, права на запись и исполнение.

# Paging на примере x86

## Таблица страниц



- ▶ Для трансляции линейного адреса в физический используется иерархическая структура:
  - ▶ узел дерева - 4KB таблица, 512 записей в каждой;
  - ▶ физический адрес корня хранится в специальном регистре *cr3*;
  - ▶ запись в таблице содержит физический адрес таблицы следующего уровня;
  - ▶ смещение в таблице берется из линейного адреса.

# Paging на примере x86

## Замечания о таблице страниц

- ▶ Различные линейные адреса легко могут быть отображены на один физический:
  - ▶ например, мы можем создать одну страничку заполненную нулями и отобразить все странички, которые должны быть занулены на нее.
- ▶ *Обычно* у каждого пользовательского приложения своя таблица страниц:
  - ▶ таким образом память одних приложений может быть изолирована от памяти других приложений;
  - ▶ при этом разные таблицы страниц могут также ссылаться на одну и ту же физическую память.

# Paging на примере x86

## PML4 Entry



- ▶ P (Present) - если 0, то запись не валидна, иначе запись валидна;
- ▶ R/W - если 0, то запись запрещена, иначе разрешена;
- ▶ S/U - если 0, то доступ разрешен только для привилегированного кода;
- ▶ XD - если 1, то исполнение кода запрещено (специальная фича, требует включения);
- ▶ A - автоматически выставляется в 1, если запись была использована для CPU для трансляции адресов;
- ▶ Physical Address - физический адрес таблицы следующего уровня;

# Paging на примере x86

## PDPT и PDT Entries



- ▶ PS (Page Size) - если 1, то Physical Address хранит адрес 1GB или 2MB страницы (в PDPT 1GB, в PDT 2MB) физической памяти, в противном случае адрес таблицы следующего уровня;
- ▶ т. е. не все страницы в отображении всегда одного размера.

# Paging на примере x86

## Замечания о таблице страниц

- ▶ Одна запись в PML4 описывает 512 GB линейного адресного пространства:
  - ▶ все запреты в PML4E распространяются на весь 512 GB регион независимо от того, что написано в записях нижних уровней;
- ▶ одна запись в PDPT описывает 1 GB линейного адресного пространства;
- ▶ одна запись в PDT описывает 2 MB линейного адресного пространства;
- ▶ одна запись в PD описывает 4 KB линейного адресного пространства:
  - ▶ 4 KB - наименьшая возможная гранулярность в x86.

# Канонический адрес

- ▶ Из линейного адреса в x86 используется всего 48 бит, но указатель 64 битный
  - ▶ биты 63-48 должны быть равны биту 47;
  - ▶ т. е. если бит 47 установлен, то и биты 64-48 тоже должны быть равны 1, в противном случае они должны быть равны 0.
- ▶ На адреса в x86 можно смотреть как на числа со знаком:
  - ▶ есть два региона допустимых адресов  $[0, 2^{47})$  и  $[-2^{47}, 0)$ ;
  - ▶ отрицательные 64-битное число  $x$  представляется с использованием дополнения до 2 как беззнаковое число  $2^{64} - x$ .

# Page Fault

- ▶ Ошибки трансляции адресов:
  - ▶ при трансляции адресов бит P может быть сброшен в одной из записей;
  - ▶ может произойти нарушение привелегий - непривилегированный код пытается читать/писать память только для привилегированного кода;
  - ▶ нарушение прав доступа, запись/исполнение в/из памяти доступной только на чтение.
- ▶ При подобных ошибках полезно знать:
  - ▶ адрес, к которому происходило обращение;
  - ▶ какое обращение к памяти происходило (чтение/запись/исполнение) или какого рода ошибка произошла.



# Page Fault в x86

Код ошибки при Page Fault



- ▶ P - 0, если Page Fault произошел из-за сброшенного бита P в одной из записей;
- ▶ R/W - 0, если Page Fault произошел при чтении, в противном случае Page Fault произошел при записи;
- ▶ S/U - 0, если Page Fault произошел в привилегированном коде, в противном случае Page Fault произошел в непривилегированном коде.

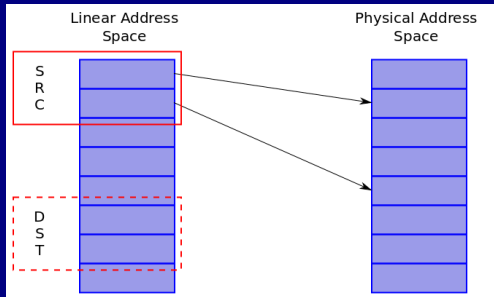
# Page Fault в x86

Адрес доступа и адрес возврата

- ▶ Адрес, обращение к которому вызвало Page Fault, сохраняется в специальный регистр *cr2*.
- ▶ Адрес возврата, который сохраняется на стек перед вызовом обработчика исключения, содержит адрес инструкции, которая привела к Page Fault
  - ▶ т. е. по возвращению из обработчика исключения инструкция будет выполнена заново;
  - ▶ т. е. обработчик исключения может поправить отображение после чего инструкция будет выполнена заново.

# Page Fault в x86

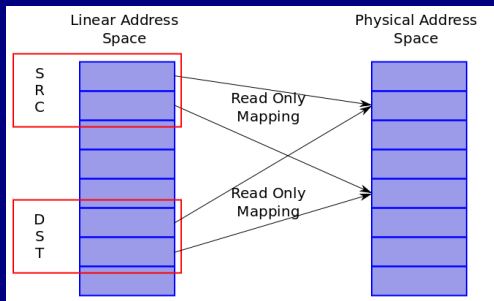
Пример: оптимизация тетсру больших регионов 1/3



- ▶ Хотим скопировать SRC в DST (для простоты регионы выровнены на 4 KB)
  - ▶ SRC каким-то образом отображено на физическую память;
  - ▶ отображение DST нас не особо интересует.

# Page Fault в x86

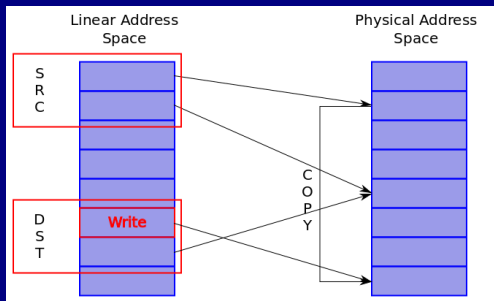
Пример: оптимизация тетсру больших регионов 2/3



- ▶ Вместо копирования памяти отображаем DST на те же страницы, что и SRC
  - ▶ оба отображения должны быть Read Only, чтобы модификация приводила к Page Fault.

# Page Fault в x86

Пример: оптимизация тетсру больших регионов 3/3



- ▶ Попытка записи в SRC или DST приводит к Page Fault:
  - ▶ в обработчике Page Fault мы можем на самом деле скопировать страницу памяти;
  - ▶ таким образом мы реально копируем только те страницы, которые будут изменяться и только тогда, когда они будут изменяться.

# Q&A