Operating Systems

Виртуализация

Me

April 26, 2016

План лекции

- История вопроса.
- Зачем нужна виртуализация.
- Эффективная виртуализация. Trap and Emulate.
- Бинарная трансляция.
- Виртуализация памяти.

История вопроса

- Представим, что вам нужно разработать новую ОС
  - вам потребуется отлаживать ОС, что не всегда возможно с реальным железом (его может не быть, оно может быть дорогим);
  - вам поможет симулятор - эмулирующий реальное оборудование;
  - симуляция очень сильно замедляет исполнение;

История вопроса

- Представим, что вам нужно разработать новую ОС
  - вам потребуется отлаживать ОС, что не всегда возможно с реальным железом (его может не быть, оно может быть дорогим);
  - вам поможет симулятор - эмулирующий реальное оборудование;
  - симуляция очень сильно замедляет исполнение;
- Что если эмулируемая платформа (*guest*) и платформа, на которой выполняется эмуляция (*host*) совпадают?
  - Можно ли ускорить симуляцию в этом случае?
  - На сколько эффективной может быть такая эмуляция?

История вопроса

- А зачем вообще эмуляция, если у нас уже есть нужное железо?
  - пользовательское программное обеспечение полагается на ОС;
  - хочется запускать ПО для разных ОС на одной машине;
  - т. е. хотим запускать несколько ОС на одной машине.

История вопроса

- А зачем вообще эмуляция, если у нас уже есть нужное железо?
  - пользовательское программное обеспечение полагается на ОС;
  - хочется запускать ПО для разных ОС на одной машине;
  - т. е. хотим запускать несколько ОС на одной машине.
- Виртуальные машины изначально были разработаны для решения описанной проблемы
  - гиппервизор (Virtual Machine Monitor) - создает иллюзию нескольких экземпляров одного оборудования;
  - виртуальная машина (Virtual Machine) - изолированное окружение для запуска экземпляра ОС;

Зачем нужна виртуализация

- Зачем виртуализация используется сейчас?
  - История историей, но сейчас компьютеры стали гораздо дешевле;
  - запуск приложений для других ОС на *desktop*;
  - тестирование и отладка может быть удобнее в виртуальной машине;
  - консолидация ресурсов;

Зачем нужна виртуализация

- Зачем виртуализация используется сейчас?
  - История историей, но сейчас компьютеры стали гораздо дешевле;
  - запуск приложений для других ОС на *desktop*;
  - тестирование и отладка может быть удобнее в виртуальной машине;
  - консолидация ресурсов;
- Очень часто ваше ПО не делает ничего полезного:
  - например, сервер может простаивать в ожидании подключений;
  - в итоге аппаратные ресурсы простаивают в пустую;
  - запустив несколько ОС на одной физической машине можно увеличить утилизацию;

Эффективная виртуализация

- Итак нам нужна виртуализация, как сделать ее эффективной?
  - Как вообще понимать эффективность виртуализации?
  - Когда возможна эффективная виртуализация?
  - Интуитивно, при эффективной виртуализации как можно больше кода должны исполняться нативно (без интерпретации);

Эффективная виртуализация

- Итак нам нужна виртуализация, как сделать ее эффективной?
  - Как вообще понимать эффективность виртуализации?
  - Когда возможна эффективная виртуализация?
  - Интуитивно, при эффективной виртуализации как можно больше кода должны исполняться нативно (без интерпретации);
- В 74 году два товарища - Попек и Голдберг, потрудились сформулировать формальный критерий:
  - оригинальная статья "Formal Requirements for Virtualizable Third Generation Architectures";
  - как и для любого формализма нам потребуется несколько вводных определений;

Модель системы

- В системе существуют два режима работы - привилегированный и непривилегированный
  - в привилегированном режиме можно исполнять любые инструкции;
  - в непривилегированном некоторые инструкции приводят к генерации исключения (a. k. a. *trap*);
  - таким образом инструкции делятся на привилегированные и непривилегированные;

Модель системы

- В системе существуют два режима работы - привилегированный и непривилегированный
  - в привилегированном режиме можно исполнять любые инструкции;
  - в непривилегированном некоторые инструкции приводят к генерации исключения (a. k. a. *trap*);
  - таким образом инструкции делятся на привилегированные и непривилегированные;
- Среди инструкций также выделяют чувствительные инструкции:
  - управляющие инструкции изменяют конфигурацию ресурсов системы (например, изменение таблицы страниц);
  - инструкции чувствительные к конфигурации инструкции;

Критерий виртуализуемости

- Эффективный *VMM* может быть построен для заданной архитектуры при условии, что все чувствительные инструкции являются привилегированными
  - все инструкции, которые мы хотели бы обрабатывать особым образом генерируют *trap*;

Критерий виртуализуемости

- Эффективный *VMM* может быть построен для заданной архитектуры при условии, что все чувствительные инструкции являются привилегированными
  - все инструкции, которые мы хотели бы обрабатывать особым образом генерируют *trap*;
- *VMM* организован следующим образом:
  - *VM* работает в непривилегированном режиме как обычный процесс;
  - при попытке исполнить "опасную" инструкцию генерируется *trap*;
  - *VMM* перехватывает управление и "эмулирует" инструкцию;

x86 рушит наши планы

- В x86 есть несколько чувствительных непривилегированных инструкций
  - например, инструкция *SGDT* позволяет получить указатель на *GDT*;
  - код ОС может считывать и проверять значение указателя *GDT*;
  - код ОС может считать оттуда совсем не то, что она туда положила;

x86 рушит наши планы

- В x86 есть несколько чувствительных непривилегированных инструкций
  - например, инструкция *SGDT* позволяет получить указатель на *GDT*;
  - код ОС может считывать и проверять значение указателя *GDT*;
  - код ОС может считать оттуда совсем не то, что она туда положила;
- x86 довольно популярная платформа
  - виртуализуемая или нет - мне надо!
  - в новых версиях x86 появилась поддержка эффективной виртуализации;
  - но и до появления этой поддержки компании VMWare удавалось довольно эффективно виртуализовывать x86.

Бинарная трансляция

- Вернемся к старой и медленной симуляции
  - зная, что *guest* и *host* используют одну архитектуру мы можем довольно быстро "переписать" код;
  - нам нужно найти в коде "опасные" инструкции и заменить их на вызовы гиппервизора;

Бинарная трансляция

- Вернемся к старой и медленной симуляции
  - зная, что *guest* и *host* используют одну архитектуру мы можем довольно быстро "переписать" код;
  - нам нужно найти в коде "опасные" инструкции и заменить их на вызовы гиппервизора;
- Как найти опасные инструкции?
  - Как вообще найти код среди всей памяти в системе?
  - Что является кодом, что является данными определяется в момент исполнения.
  - Мы знает точку входа, далее мы можем отслеживать куда передается управление.

Бинарная трансляция

- Транслятор работает с *базовыми блоками*:
  - базовый блок - линейная последовательность инструкций заканчивающаяся инструкцией перехода;
  - трансляция базового блока - замена "опасных" инструкций на вызов *VMM* и, возможно, вставка какого-то эпилога;
  - для каждого транслированного блока поддерживается его адрес и размер в *VM*;
  - транслированные блоки можно кешировать;
  - транслировать код, выполняющийся в непривилегированном режиме, обычно, не нужно;

Пример

```
1       addl   $4,              %esp
2       movl   %cr0,            %eax
3       orl    $(1 << 31), %eax
4       movl   %eax, %cr0
5       pushl  $enable_64bit_gdt
6       call   videomem_puts
```

```
1       addl   $4,              %esp
2       call   trap_get_cr0
3       orl    $(1 << 31), %eax
4       pushl %eax
5       call   trap_set_cr0
6       pushl  $enable_64bit_gdt
7       pushl  $videomem_puts
8       call   back_to_vmm
```

- В примере *VMM* перехватывает операции с регистром *%cr0*;
- Инструкция перехода в конце транслируется в вызов *VMM*

Аппаратная поддержка виртуализации

- Intel-VT и AMD-V - расширения архитектуры для поддержки эффективной виртуализации
  - детали различаются, но обе технологии позволяют перехватывать чувствительные инструкции;
  - первое поколение Intel-VT и AMD-V проигрывало по скорости бинарной трансляции из-за большой стоимости передачи управления между *VMM* и ОС;
  - т. е. эффективная вирутализация оказалась не такой эффективной;

Виртуализация памяти

- *VMM* должен разделить физическую память между *VM*
  - так чтобы у каждая ОС считала, что имеет дело с физической памятью;
  - используя *paging* мы можем делать с памятью все, что захотим;
  - но современные ОС сами используют *paging*;

Виртуализация памяти

- *VMM* должен разделить физическую память между *VM*
  - так чтобы у каждая ОС считала, что имеет дело с физической памятью;
  - используя *paging* мы можем делать с памятью все, что захотим;
  - но современные ОС сами используют *paging*;
- Мы можем перехватывать подмену корня таблицы страниц:
  - обычно это привилегированная инструкция, например, запись в *%cr3* в x86;
  - как отслеживать изменения на других уровнях? (вы уже можете сами догадаться, как это сделать)

Адресные пространства

- Раньше у нас было два адресных пространства:
  - *Virtual Address Space* - адресное пространство, которое создает ОС для себя и процессов;
  - *Physical Address Space* - физическая память;

Адресные пространства

- Раньше у нас было два адресных пространства:
  - *Virtual Address Space* - адресное пространство, которое создает ОС для себя и процессов;
  - *Physical Address Space* - физическая память;
- С появлением *VMM* ситуация меняется:
  - *Virtual Address Space (VA)* - адресное пространство, которое создает ОС;
  - *Physical Address Space (PA)* - то, что ОС считает физической памятью;
  - *Machine Address Space (MA)* - настоящая физическая память;

Адресные пространства

- Теперь нам нужно поддерживать два отображения:
  - оборудование поддерживает аппаратно только одно отображение;
  - как эффективно поддержать оба?

Адресные пространства

- Теперь нам нужно поддерживать два отображения:
  - оборудование поддерживает аппаратно только одно отображение;
  - как эффективно поддержать оба?
- Мы можем отслеживать два типа изменений отображения *VA* на *PA*:
  - изменение корня таблицы страниц;
  - изменение записи в одной из таблиц отображения;

Shadow Tables

- ОС подменяет корень отображения
  - *VMM* знает отображение *PA* на *MA*;
  - *VMM* знает корень отображения и может пройтись по всему дереву таблиц страниц;
  - *VMM* просто создает новое отображение *VA* напрямую на *MA*;

Shadow Tables

- ОС подменяет корень отображения
  - *VMM* знает отображение *PA* на *MA*;
  - *VMM* знает корень отображения и может пройтись по всему дереву таблиц страниц;
  - *VMM* просто создает новое отображение *VA* напрямую на *MA*;
- ОС подменяет запись в таблице
  - в этом случае просто обновляем созданное отображение;

Пример

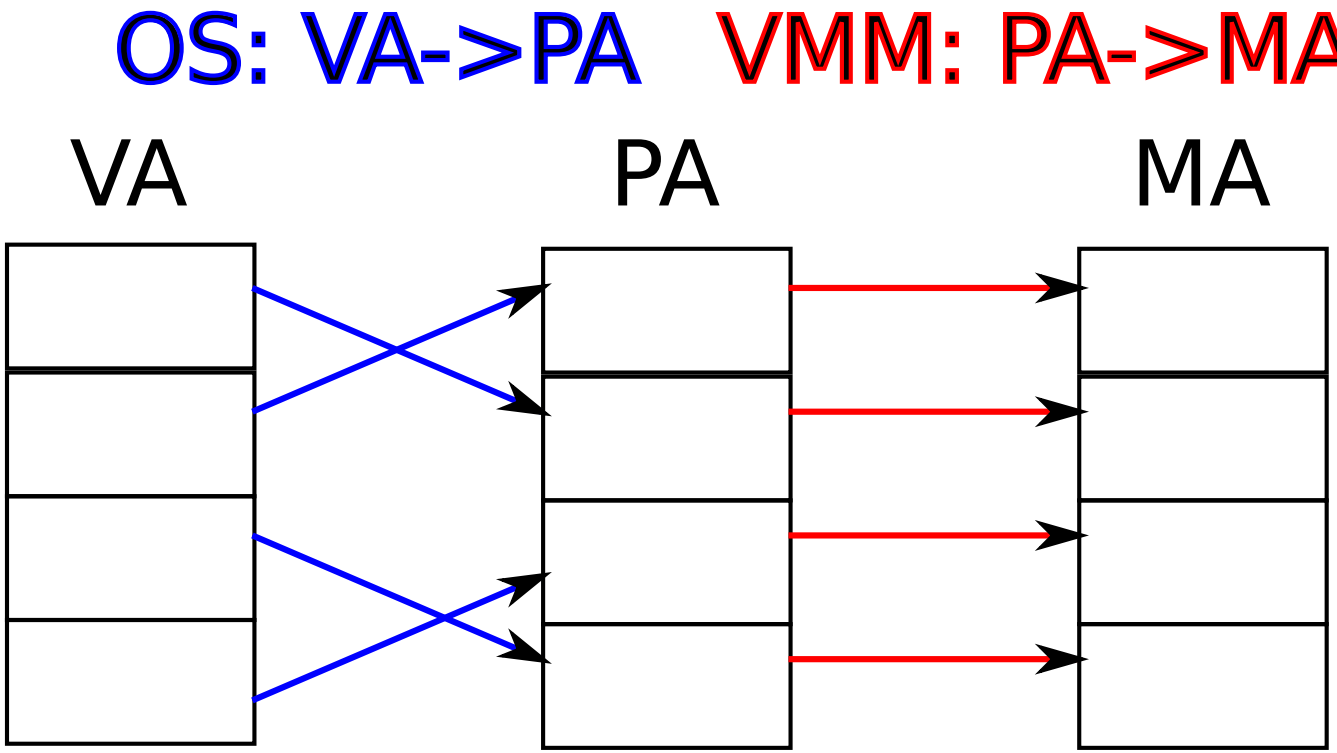

Figure : Shadow Page Table

Пример

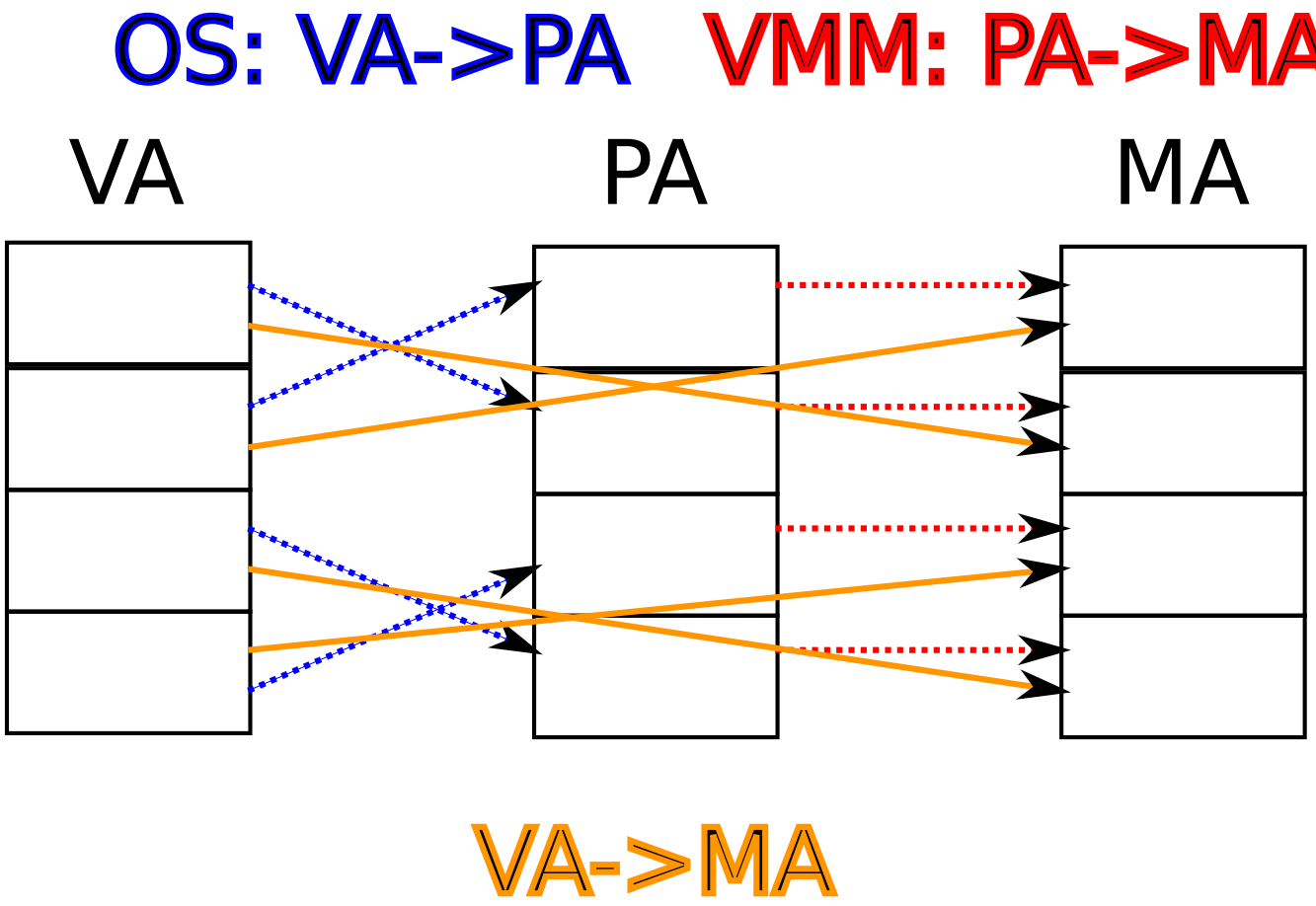

Figure : Shadow Page Table

Аппаратная поддержка

- В процессоры x86 были добавлены расширения *EPT* и *RVI*, которые добавляют аппаратную поддержку трансляции памяти
  - как обычно, обе технологии делают одно и тоже, разница в деталях;
  - фактически они позволяют гиппервизору задать отображение *PA* на *MA*, так чтобы ОС ничего о нем не знала;
  - отображение *PA* на *MA* меняется редко, поэтому *VMM* почти не приходится вмешиваться;

Balloon (bonus)

- Перераспределение памяти в ОС:
  - в ОС, если вы хотите освободить память вы делаете *swapping*;
  - т. е. сохраняете содержимое страниц на диске, и отдаете страницы памяти кому-то другому;

Balloon (bonus)

- Перераспределение памяти в ОС:
  - в ОС, если вы хотите освободить память вы делаете *swapping*;
  - т. е. сохраняете содержимое страниц на диске, и отдаете страницы памяти кому-то другому;
- Перераспределение памяти в *VMM*:
  - *VMM* запускает несколько экземпляров ОС, которые умеют делать сложный *swapping*;
  - напишем для ОС драйвер, который знает о существовании *VMM*, назовем его *balloon*;
  - если *VMM* хочет забрать часть памяти у ОС, просим *balloon* выделить нужное количество памяти (надуваем *balloon*);
  - если *VMM* решает вернуть память, просто просим драйвер освободить память (сдуваем *balloon*);

Осталось за кадром

- Паравиртуализация
  - гиппервизору полезно иногда знать, что делает ОС внутри *VM*;
  - паравиртуализация предполагает модификацию ОС, так чтобы она знала о *VMM* и умела с ним взаимодействовать;

Осталось за кадром

- Паравиртуализация
  - гиппервизору полезно иногда знать, что делает ОС внутри *VM*;
  - паравиртуализация предполагает модификацию ОС, так чтобы она знала о *VMM* и умела с ним взаимодействовать;
- Контейнерная виртуализаци
  - ОС нужна нам для запуска пользовательских процессов;
  - если все процессы предназначены для одной ОС, то нам не нужен *VMM*;
  - мы можем организовать изолированные окружения для групп процессов прямо в ядре ОС;