Виртуализация/Эмуляция аппаратного обеспечения(CPU, Mem, I/O devices)

Цели эмуляции аппаратного обеспечения

Доля рынка встраиваемых систем постоянно увеличивается, в связи с чем возникает необходимость в разработке, отладке и сопровождению программного обеспечения для таких систем. Поскольку процессор, оперативная память и драйверы взаимодействия с устройствами ввода/вывода отличаются между машинами, на которой разрабатывается встраиваемое приложение и для которой оно предназначено – вопрос о запуске приложения для отладки становится очень актуальным. Эмуляция аппаратного обеспечения позволяет с высокой точностью повторить поведение процессора и устройств целевой машины и дает возможность разрабатывать программное обеспечение без необходимость постоянно запускать его на хосте, для которого ведется разработка.

Эмуляция центрального процессора

Существует два противоположных подхода к эмуляции системных областей процессора:

- Использование механизма страничной защиты для эмуляции системной области и запуск всех работающих с областью инструкций без эмуляции(например защита и эмуляция структуры данных, содержащих дескрипторы сегментов и запуск инструкций загрузки/чтения сегментных регистров без эмуляции)
- Разрешение свободной записи/чтения системной области и эмуляция всех работающих с областью инструкций(например, свободная запись/чтение в структуру данных, содержащую дескрипторы сегментов и эмуляция инструкций загрузки/чтения сегментных регистров)

Помимо системных областей, схема виртуализации также должна обеспечивать:

- Эмуляцию колец защиты процессора
- Эмуляцию исключений и прерываний процессора
- Эмуляцию набора инструкций процессора
- Эмуляция внешних устройств

Эмуляция центрального процессора

К методам эмуляции процессора относятся:

- Интерпретатор

Наиболее простая реализация эмуляции процессора, при таком подходе описывается работа моделируемого процессора - последовательно происходит считывание участка памяти(длиною в инструкцию моделируемого процессора), который указан в регистре PC. Считанная инструкция декодируется и, в соответствие с этой инструкцией, изменяется состояние моделируемого процессора(или памяти, доступной моделируемой машине), значение в регистре PC увеличивается на длину инструкции процессора и описанный шаг повторяется.

Очевидным минусом такого подхода является скорость исполнения команды, так как ее необходимо декодировать и определить исполняемую функцию, по полученному коду операции. Существуют некоторые подходы для ускорения интерпретатора, нацеленные на ускорение исполнения повторяющихся инструкций, путем инлайнинга кода функции. Плюсом же такого подхода, помимо относительной простоты(если известна архитектура моделируемого процессора) реализации является то, что реализацию интерпретатора можно скомпилировать под любую архитектуру и запустить на другом хосте.

Эмуляция центрального процессора

- Бинарный транслятор

Отличием такого подхода от интерпретатора является то, что код операции, считанный с программы, исполняемой моделируемым процессором, подменяется на эквивалентный код операции процессора, на котором эмулируется система, таким образом инструкция выполняется «мгновенно» процессором хоста. Для тех инструкций, эквивалента которым нет на хосте, компилируются блоки кода, которые будут исполняться при необходимости. В бинарном трансляторе также применяется метод кэширования кода инструкций, которые выполняются в цикле. Минусом такого подхода является то, что реализация такого эмулятора должна быть своя для каждой архитектуры набора команд процессора

```
switch(memory[PC++])
{
        case OPCODE1:
                opcode1();
                break;
        case OPCODE2:
                opcode2();
                break;

        ....

        case OPCODEn:
                opcodeN();
                break;
}
```

Эмулятор интерпретатор

Эмуляция центрального процессора

- Соответствие тактовой частоте

Одной из важнейших частей эмуляции является соответствие тактовой частоте моделируемого процессора. При слишком высокой скорости работы интерпретатора/транслятора, могут произойти нарушения в исполняемой программе, например она может ожидать в цикле готовности какого-либо внешнего ресурса и прерывание, инициированное при готовности этого ресурса, просто не дойдет вовремя. Также известна проблема прерываний синхронизации от внешних устройств, например видеооборудования, которое посылает сигнал vsync(синхронизация кадра) в конце каждого кадра. Если этот сигнал не будет обработан вовремя, качество и плавность смены кадров будет потеряна. Для соблюдения тактовой частоты необходимо вычислить количество циклов, выполняемых моделируемым процессором за определенный временной промежуток, и количество циклов, выполненных эмулятором за то же время. Если во втором случае число выше, эмулятор останавливается при достижении количества циклов реального процессора и ожидает следующего временного интервала. Вышесказанное можно описать следующей формулой:

$$c_1 * t_1 = c_2 * t_2 * k, \text{ где}$$

c_1 - количество циклов моделируемого процессора, выполненных на реальном процессоре

t_1 - время, за которое они выполнились на реальном процессоре

Эмуляция центрального процессора

c_2 - количество циклов моделируемого процессора

t_2 - время, за которое они выполнились при моделировании процессоре

k – коэффициент, на который нужно умножить время чтобы получить, в какой момент приостановить выполнение инструкций.

Таким образом получается:

$$k = \frac{c_1 * t_1}{c_2 * t_2},$$

Так как $t_1 = t_2$ (количество циклов было замерено за одинаковый интервал), получаем:

$$\mathrm{t} = t_1 - kt_1 = t_1(1 - k) = t_1(1 - \frac{c_1}{c_2}),$$

Полученное число показывает, что каждые $t_1 - t$ секунд работы интерпретатора/бинарного транслятора необходимо приостанавливать его работу на t секунд.

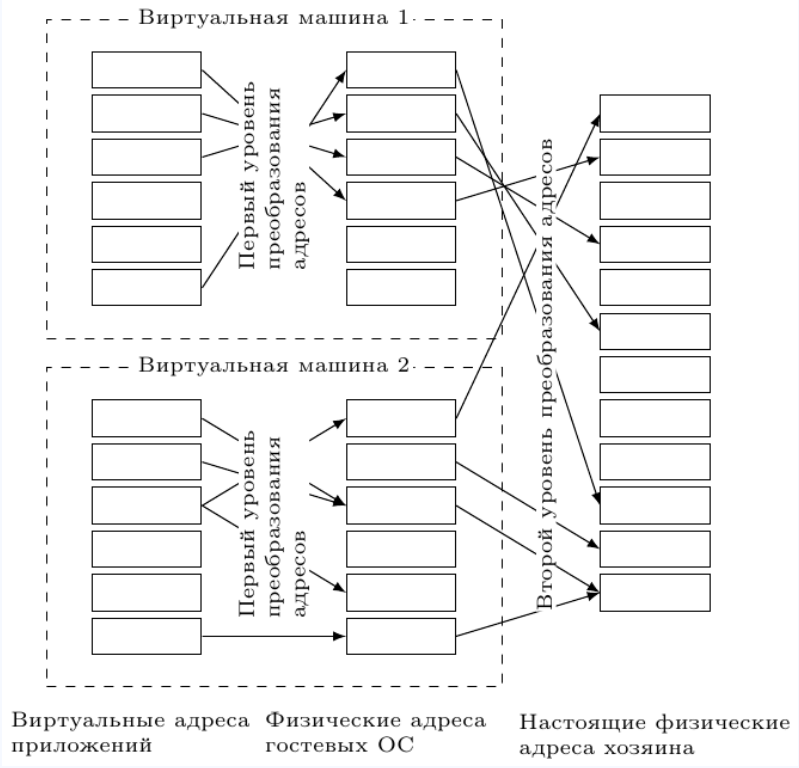
Эмуляция оперативной памяти

Часть оперативной памяти хоста разделяется на регистры(моделируемого процессора) и оперативную память гостя. Трансляция между памятью гостя и хоста происходит с помощью «карты памяти»(memory map).

Со стороны моделируемого процессора также может быть реализован блок управления памятью(Memory management unit), в задачи которого входит:

- Трансляция виртуального адреса памяти в физический
- Защита памяти
- Управление кэш-памятью
- Переключение банков памяти

Эмуляция оперативной памяти



Трансляция блоком управления памятью виртуальных адресов в физические адреса и их преобразование в физические адреса хоста

QEMU

QEMU – свободно распространяемая программа с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ

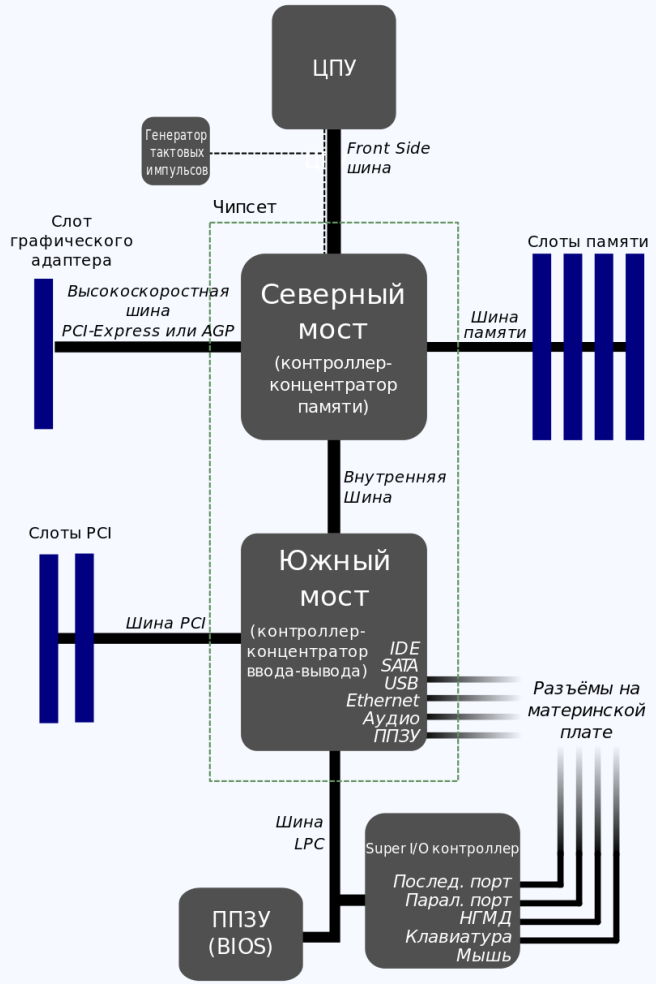
Данная программа позволяет запускать операционные системы, предназначенные под одну архитектуру, на другой(например, ARM на x86). Кроме процессора QEMU эмулирует различные периферийные устройства: сетевые карты, HDD, видео карты, USB и др.

Принцип работы QEMU немного отличается от описанных выше подходов:

- Бинарный код конвертируется в промежуточный платформонезависимый код
- Получившийся код конвертируется в инструкции для хоста и исполняется

Отличие заключается в том, что промежуточный платформонезависимый код позволяет легко расширять эмулятор новыми архитектурами набора команд процессора. Достигается это за счет исключение необходимости переводить бинарный код процессора непосредственно в инструкции для хоста.

QEMU

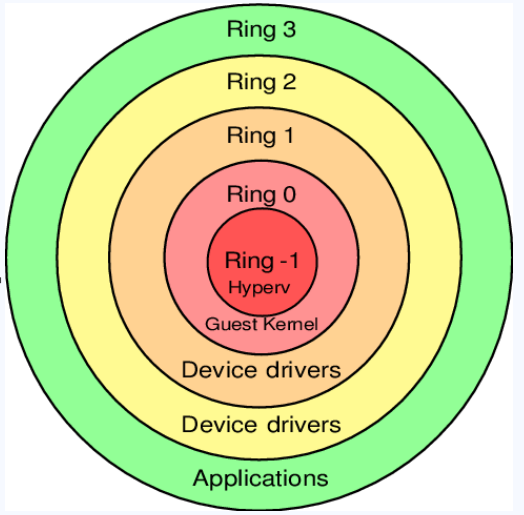

ЦПУ
Генератор тактовых импульсов
Front Side шина
Чипсет
Слот графического адаптера
Высокоскоростная шина PCI-Express или AGP
Северный мост (контроллер-концентратор памяти)
Шина памяти
Слоты памяти
Внутренняя Шина
Слоты PCI
Шина PCI
Южный мост (контроллер-концентратор ввода-вывода)
IDE
SATA
USB
Ethernet
Аудио
ППЗУ
Разъёмы на материнской плате
Шина LPC
Super I/O контроллер
Послед. порт
Парал. порт
НГМД
Клавиатура
Мышь
ППЗУ (BIOS)

Intel Virtualization Technology

Данная технология включает в себя:

- Возможность виртуализации центрального процессора

Без технологии Intel VT-x, гипервизоры работали на Ring 0, а гости на Ring 1. Ввиду отсутствия достаточных прав для нормального функционирования гостевой операционной системы на Ring 1, то при каждом привилегированном вызове от гстевой системы, гипервизору приходилось на лету его модифицировать и выполнять на Ring 0. Таким образом гостевой код не выполнялся на процессоре напрямую, что существенно замедляло работу гостевой ОС. Тогда несколько производителей процессоров, независимо друг от друга выпустили расширенный набор инструкций, позволяющий гостевой системы выполнять код напрямую, на процессоре хоста. Таким образом добавился новый уровень кольца – Ring -1, на котором работает гипервизор, а гостевые ОС на Ring 0, имея привилегированный доступ к процессору.



Intel Virtualization Technology

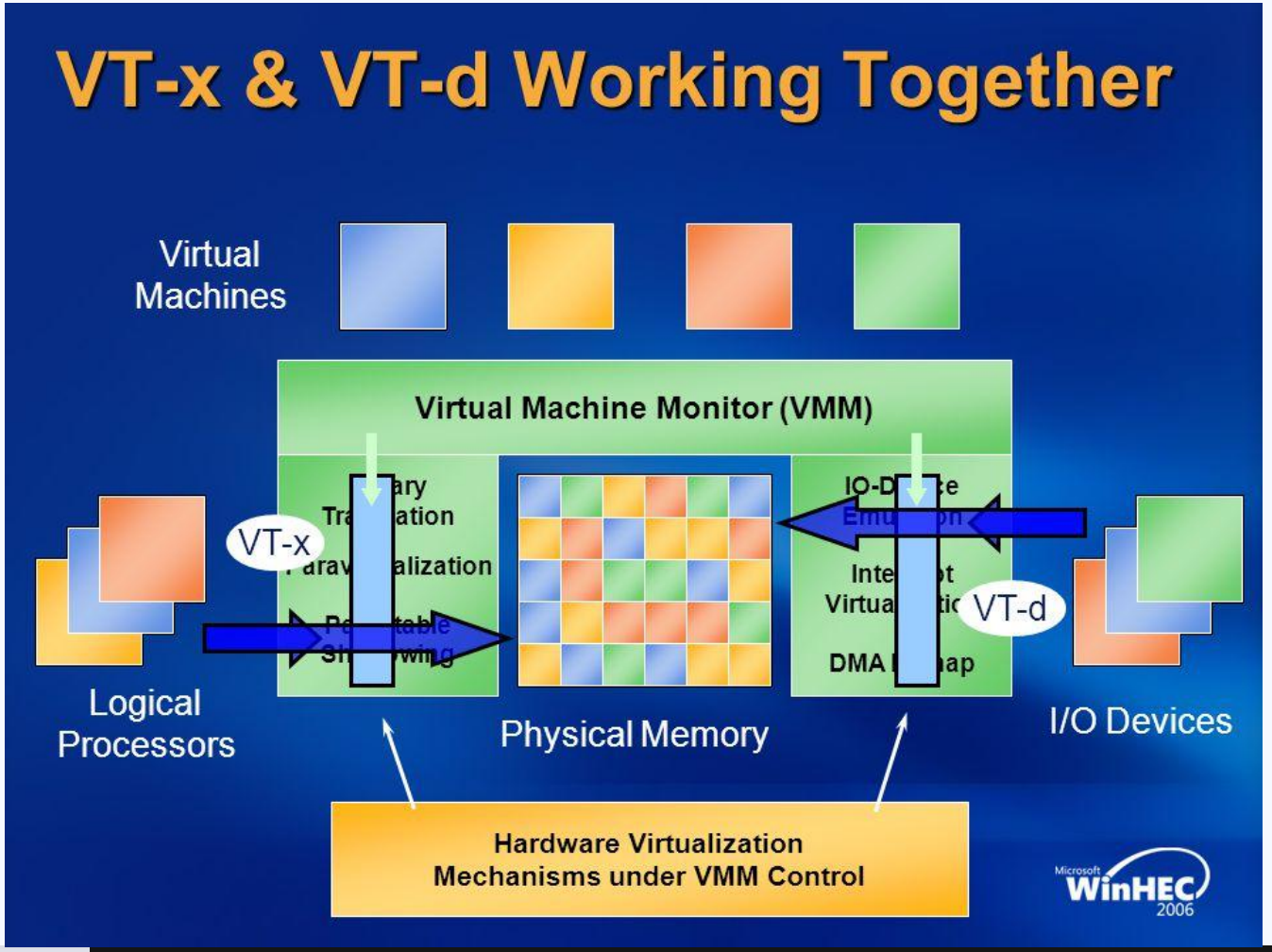
- Возможность виртуализации памяти

Позволяет использовать изоляцию и мониторинг памяти на основе каждой виртуальной машины. Включает переназначение прямого доступа к памяти(DMA) и таблицы Extended Page Tables(EPT), включая их расширения: доступные биты и биты очистки.

- Возможность виртуализации систем ввода-вывода

Ускоряют выгрузку многоядерной обработки пакетов на сетевые адаптеры, а также прямое назначение виртуальных машин виртуальным функциям, включая ввод-вывод диска.

Intel Virtualization Technology



Virtual machine control structure

Virtual machine control structure(VMCS) – структура данных, существующая для каждой виртуальной машины, управляемой VMM. VMCS определяет состояние каждого виртуального процессора и меняет состояние при каждом изменении контекста выполнения между разными виртуальными машинами. Эта структура является read-only для виртуальной машины и read-write для гипервизора.

RIP, RSP, RFLAGS	
CR3, CR4, DR7	
Selector	{ES, CS, SS, DS, FS, GS, LDTR, TR}
Base	{ES, CS, SS, DS, FS, GS, LDTR, TR, GDTR, IDTR}

Как видно из примера части структуры VMCS, она состоит только из следующих регистров: указатель на следующую инструкцию, указатель на стэк, регистр флагов и сегментных регистров. Другими словами структура состоит только из «управляющих регистров», отсутствуют такие, как: rax, rbx, r1 и т.д., то есть регистры, в которых располагают данные, над которыми работает программа.

Кольца защиты

Кольца защиты ограничивают возможность влияния приложений на работу гостевой ОС, и работу гостевой ОС на нормальное функционирования хоста, запрещая напрямую использовать все доступные регистры, вместо этого реализуя API для общения(с ОС и аппаратным обеспечением).

На рисунке показано, что приложения виртуальных машин работают на третьем уровне кольца, а гостевая операционная система на первом. Для нормального функционирования ОС недостаточно привилегий первого уровня, следовательно виртуальной машине необходимо постоянно обращаться к VMM, что существенно снижает ее скорость работы. С помощью технологии VT-x проблема решается таким образом, что VMM располагается на кольце, ниже нулевого уровня, а гостевая ОС, в свою очередь располагается на нулевом уровне, получая необходимые инструкции, позволяющие ей выполняться без постоянного обращения к VMM.