Денис Силаков

KVM и AQEMU – аппаратная виртуализация на домашней машине

На протяжение последнего десятилетия пальму первенства в области виртуализации на настольных машинах прочно удерживала VMware со своей Workstation. Основной свободной альтернативой с дружественным интерфейсом традиционно считается VirtualBox. Другие свободные решения – такие, как Xen, QEMU или KVM – применяются в основном на серверах, отпугивая многих пользователей-неспециалистов необходимостью разбираться в дебрях опций командной строки. Однако ситуация не столь плачевна, как может показаться – ведь и для инструментов командной строки существуют графические оболочки-надстройки.

В этой статье я расскажу о KVM (Kernel-based Virtual Machine) – как он работает, чем примечателен, а также как его использовать совместно с графической оболочкой AQEMU, получив вполне реальную альтернативу VirtualBox.

Немного теории

Виртуализацию – возможность запускать несколько ОС на одной физической машине так, что они не подозревают о существовании друг друга – впервые реализовали еще 60х годах прошлого века для систем IBM System/360 и 370. Идея тех реализаций была достаточно простой и основывалась на аппаратном разграничении привилегий – каждый процесс, выполняемый на компьютере, работает на определенном уровне привилегий, и в зависимости от этого уровня, ему разрешается либо запрещается выполнение тех или иных инструкций. При попытке выполнить «запрещенную» инструкцию генерируется аппаратное исключение, которое может быть перехвачено и обработано программами с более высоким уровнем привилегий.

В классической схеме виртуализации, получившей название “Trap-and-Emulate” («лови и эмулируй») виртуализируемые ОС переносятся на один из непривилегированных уровней, а на уровень с наивысшими привилегиями помещается специальная программа – монитор виртуальных машин (ВМ), или гипервизор. Попытка выполнить привилегированную инструкцию в ВМ приводит к аппаратному исключению. Это исключение перехватывается гипервизором, который эмулирует выполнение инструкции для конкретной ВМ. Непривилегированные же инструкции выполняются «как есть», без участия гипервизора. Как показала практика, доля инструкций, которые необходимо эмулировать, в реальных системах достаточно мала, поэтому производительность такой реализации достаточно высока.

Однако есть одна тонкость – чтобы описанная реализация работала, все инструкции процессора, которые могут изменить состояние ресурсов машины, должны быть привилегированными. Также не должно быть инструкций, которые ведут себя по-разному на разных уровнях привилегий, не вызывая аппаратных исключений. Эти условия были впервые сформулированы разработчиками систем виртуализации IBM и по их именам получили название критерия Попека-Голдберга.

Увы, архитектура Intel x86 (и ее 64-битное расширение) этому критерию не удовлетворяет. Поэтому долгое время считалось, что разработать эффективное (в плане производительности) ПО для виртуализации на этой платформе нельзя. (Неэффективное – пожалуйста; например,

программа Bochs просто эмулирует каждую инструкцию, выполняемую ВМ, но ее производительность может быть в сотни раз меньше, чем у реальной машины).

Тем не менее, возможность виртуализации оказалась востребованной и на платформе x86, так что в конце концов были разработаны пути обхода ограничений аппаратуры. Сначала VMware предложила эффективную реализацию динамической бинарной трансляции, осуществляющую анализ и эмуляцию инструкций от ВМ «на лету». Альтернативным подходом явилась паравиртуализация, основанная на внесении модификаций в код ОС, запускаемой внутри ВМ. Целью таких модификаций является замена инструкции, которые необходимо эмулировать, на прямые обращения к гипервизору – то есть при таком подходе гипервизору не надо ничего перехватывать, ОС сама отдает ему управление в случае необходимости.

Однако бинарная трансляция в общем случае все-таки привносит замедление в работу ВМ и может серьезно нагружать процессор, а паравиртуализация подходит только для ОС, код которых можно модифицировать. Поэтому, наблюдая все возрастающий интерес к виртуализации, Intel и AMD решили пойти навстречу разработчикам и доработать архитектуры своих процессоров, существенно упростив создание гипервизоров. Опять же, к тому времени (примерно 2005 год) гонка мегагерцев несколько приутихла/приелась, и надо было предоставить рынку что-то новенькое.

Решения, предложенные Intel и AMD, с архитектурной точки зрения достаточно просты – в дополнение к существующим уровням привилегий («кольцам защиты» в терминологии x86), в процессоры был добавлен новый «корневой» (“root”) режим. Работающие в нем программы обладают б*о*льшими привилегиями, чем процессы, размещающиеся на классических уровнях привилегий, и могут перехватывать все инструкции, выполняемые этими процессами. Фактически в процессоры был добавлен еще один уровень привилегий, на котором и предлагается размещать гипервизор.

Именно на использовании новых свойств процессоров Intel и AMD и базируется KVM, добавляющий функционал гипервизора непосредственно в ядро Linux.

KVM и QEMU

KVM реализован как модуль ядра (точнее, несколько модулей – основной kvm и платформо-специфичные kvm-intel и kvm-amd). Его код включен в основную ветку ядра, начиная с версии 2.6.20, и сегодня KVM доступен в репозиториях большинства дистрибутивов. После установки KVM у вас должны появиться соответствующие модули, для загрузки которых можно выполнить в консоли из-под root'a команду “modprobe kvm-amd” (или kmd-intel – в зависимости от архитектуры машины). Модуль kvm будет автоматически загружен как зависимость kvm-amd/kvm-intel.

Если при попытке загрузить модули вы получите ошибку "Operation not supported", то аппаратная поддержка виртуализации в машине отсутствует или отключена. Для верности, можно посмотреть флаги в /proc/cpuinfo – среди них должны быть vmx (для Intel) либо svm (для AMD), так что вывод команды

```
grep 'vmx\|svm' /proc/cpuinfo
```

должен быть не пуст. Если таких флагов нет, можно посмотреть в BIOS – возможно, нужные

возможности присутствуют, но отключены.

Модули KVM сами по себе не являются законченным решением виртуализации – их роль заключается в перехвате инструкций ВМ, операций ввода/вывода и отслеживании ряда других событий. Но для эмуляции полноценной машины необходимо выполнять еще достаточно много действий – в частности, необходима эмуляция различных устройств (жесткого диска, сетевой карты и других). Все такие действия возлагаются на QEMU – еще один свободный эмулятор, изначально использовавший бинарную трансляцию и особой производительностью не блиставший (хотя ее можно было увеличить за счет модуля KQEMU – изначально закрытого, но бесплатного; позже модуль был открыт под лицензией GPL). Поскольку QEMU к моменту создания KVM уже имел все необходимые возможности по эмуляции различных устройств, то разработчики KVM решили не изобретать велосипед, а научить два инструмента работать сообща.

В свою очередь разработчики QEMU пришли к выводу, что использование KVM дает больше возможностей, чем ускорение с помощью KQEMU (который, к слову, был чисто программным и не нуждался в аппаратной поддержке) и теперь, при возможности, его используют. Естественно, это возможно не всегда, а только при наличии аппаратной поддержки и совпадении архитектуры эмулируемой машины с архитектурой машины физической.

Действительно, KVM дает существенный выигрыш в производительности эмуляции процессора. Но проект достаточно молодой, и проблемы с ним встречаются. В сети можно найти довольно много сообщений о том, что KVM/QEMU в определенных конфигурациях системы сильно нагружают процессор, даже если гостевая система простаивает. У меня такая ситуация наблюдалась на одной из машин, когда для гостя-WinXP была включена эмуляция звуковой карты AC97. Не стоит также забывать, что потери в скорости могут возникать из-за некачественной эмуляции устройств в QEMU — при высокой нагрузке на диск или сетевую карту внутри ВМ могут проявится серьезные задержки. Для улучшения эмуляции устройств можно воспользоваться Virtio (http://wiki.libvirt.org/page/Virtio), но его использование — тема отдельного разговора.

Сам QEMU – это консольный набор инструментом с богатым набором опций. Много возможностей – это хорошо и практично, но для «домашнего» использования нет необходимости изучать их все. К счастью, существуют графические средства управления виртуальными машинами KVM/QEMU, берущие часть забот на себя и предлагающие пользователю интерфейс, схожий с привычными VMware и VirtualBox.

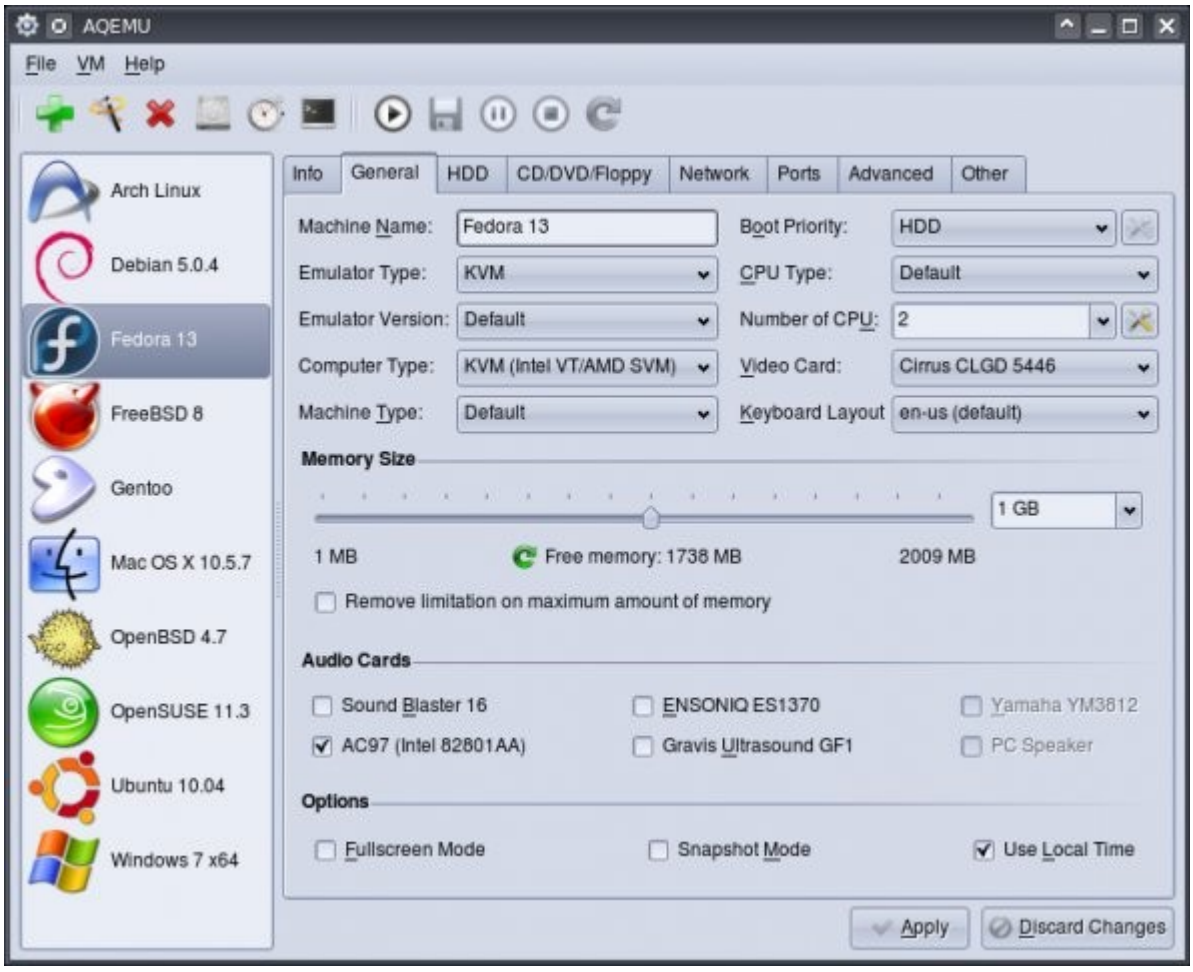
AQEMU

Наиболее "раскрученным" средством управления виртуальными машинами на сегодняшний день является Virtual Machine Manager (Vitr-Manager), разрабатываемый в RedHat и основанный на библиотеке libvirt. Однако мне Virt-Manager кажется достаточно аскетичным и предназначенным, скорее, для системных администраторов, которым необходимо управлять большим количеством ВМ.

Гораздо больше мне понравился AQEMU – продукт отечественного производства, получить который можно на http://sf.net/projects/aqemu. AQEMU включен в репозитории многих дистрибутивов, но версии в них могут обновляться с запозданием. Здесь я буду рассматривать версию 0.8.1 — последнюю на момент написания статьи.

При первом запуске AQEMU появится мастер настроек, где надо выбрать язык интерфейса (несколько непривычно – обычно язык определяется текущей локалью), некоторые глобальные настройки (можно все оставить по умолчанию), а также произвести поиск имеющихся средств виртуализации. Если вы корректно установили и загрузили модули KVM, то он (KVM) должен появиться в списке.

Далее потребуется создать ВМ – это можно сделать с помощью мастера (меню ВМ → Новая ВМ → Мастер). Для начала, стоит выбрать обычный режим настройки. Тип эмулятора следует установить в KVM (кому не повезло с его запуском – можно выбрать QEMU, но высокой производительности ожидать не стоит). Далее необходимо указать шаблон ВМ (можно выбрать один из имеющихся – представлены все популярные системы – либо просто указать примерный год выпуска ОС), задать имя машины, размер и местоположение файла для жесткого диска и указать, необходима ли сеть. Обратите внимание, что файл с образом жесткого диска является динамически расширяемым – то есть в начале он места практически не занимает, а его размер увеличивается по мере необходимости. Физический размер файла можно всегда посмотреть в свойствах ВМ, на вкладке HDD.



Скриншот с сайта http://sourceforge.net/projects/aqemu/

После создания ВМ с помощью мастера, можно еще поиграться с ее настройками – коих насчитывается довольно много. В первую очередь, многих заинтересует вкладка CD/DVD/Floppy, где можно указать файл с образом диска для использования в качестве CD. Также можно задать размер оперативной памяти и параметры сети. Из последних стоит задать

MAC-адрес, особенно если планируется использовать несколько ВМ – иначе будет использоваться значение по умолчанию, что может привести к конфликтам. Впрочем, выдумывать самому ничего не надо – просто нажмите на кнопку с зеленой стрелочкой справа (в старых версиях – с плюсиком) – и AQEMU сгенерирует MAC.

На этом в первом приближении все – машина готова к запуску. Перед запуском убедитесь, что у вас не работают другие средства виртуализации (вроде VMware или VirtualBox). Подобные программы также могут использовать аппаратную виртуализацию, что приведет к конфликту.

Традиционно, AQEMU каждую ВМ запускает в отдельном окне – как VirtualBox. В последних версиях помимо этого появляется дополнительная панель 'Emulator Control', позволяющая управлять QEMU во время работы ВМ (отключать/подключать внешние устройства, посылать сигналы о нажатии клавиш и прочее). Также существует экспериментальная возможность использования встроенного VNC-клиента, позволяющий отображать экран ВМ непосредственно в окне AQEMU – получается похоже на VMware Workstation.

Заключение

Итак, KVM в связке с AQEMU предоставляет нам удобный и практичный аналог VirtualBox со схожей производительностью. Некоторые возможности (например, поддержка 3D-ускорения) в KVM отсутствуют, но многим пользователям они и не понадобятся. Так что на мой взгляд, попробовать KVM/AQEMU стоит – благо, это не так уж и сложно.