Визитка

ДЕНИС СИЛАКОВ, к. ф.-м. н., член рабочей группы LSB, старший архитектор ЗАО «РОСА», занимается автоматизацией разработки ОС «РОСА»

Виртуализация на платформе x86

Одновременным запуском Windows и Linux на одной машине уже никого не удивишь. Разберемся, как работают эти технологии, ставшие такими привычными

Сегодня вряд ли можно найти человека, относящегося к миру ИТ, но не знающего про виртуализацию – возможность запускать несколько ОС на одной физической машине так, что они не подозревают о существовании друг друга. Большинство пользователей ПК познакомились с этой технологией в конце 90-х годов прошлого века, когда компания VMware представила свои продукты для создания виртуальных машин в ОС Windows. С тех пор в сфере виртуализации для платформы x86 произошли фундаментальные изменения, и сегодня эта технология является основой облачных сервисов. Но как же устроены виртуальные машины? Знакомися с теорией и историей вопроса.

Предыстория

Начнем с того, что идея использовать одну физическую машину для параллельной работы нескольких ОС возникла гораздо раньше не только Windows, но и вообще персональных компьютеров. Первые промышленные реализации виртуальных машин были представлены в 60-х годах прошлого века инженерами IBM для платформ System/360 и 370. Эти реализации основывались на аппаратном разграничении привилегий: каждый процесс, выполняемый на машине, работает на определенном уровне, в зависимости от которого ему разрешается либо запрещается выполнение тех или иных инструкций. При попытке выполнить недопустимую для данного уровня инструкцию генерируется аппаратное исключение, которое может быть перехвачено и обработано программами с более высоким уровнем привилегий.

В предложенной инженерами IBM классической схеме виртуализации, получившей название «Trap and Emulate» («лови и эмулируй»), виртуализируемые ОС (следуя современной терминологии, будем называть их «гостевыми») запускаются на одном из непривилегированных уровней, а на уровень с наивысшими привилегиями помещается специальная программа – монитор виртуальных машин (ВМ) или гипервизор.

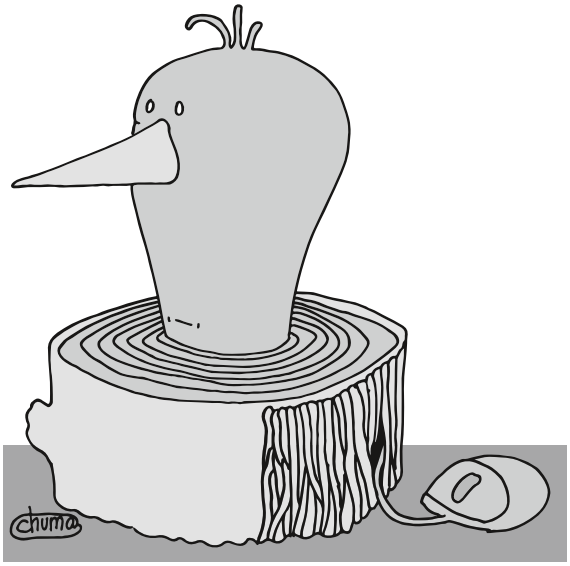
Если гостевая ОС вызывает непривилегированную инструкцию, никаких препятствий не возникает, и эта инструкция обрабатывается непосредственно аппаратурой. Попытка же выполнить привилегированную инструкцию приводит к аппаратному исключению. Это исключение перехватывается гипервизором, который эмулирует выполнение инструкции для конкретной ВМ.

Фактически гипервизор выполняет для гостевых ОС ту же роль, что сами ОС выполняют для работающих внутри них процессов. В частности, ему необходимо обеспечивать взаимодействие с различными устройствами, ведь нескольким гостевым системам нужно как-то разделять физические устройства (клавиатуру, монитор, принтер и так далее), и диспетчером в тут и выступает гипервизор. В зависимости от реализации виртуальным машинам может предоставляться либо эмуляция устройств, либо доступ к реальному «железу».

Набор инструкций для эмуляции определяется аппаратной архитектурой платформы. В случае мейнфреймов IBM доля инструкций, которые необходимо эмулировать при реальной работе, была мала, поэтому производительность гипервизоров достаточно высока. Как результат, еще в 70-х годах прошлого века на основе этих мейнфреймов строились структуры, архитектурно очень похожие на нынешние облачные сервисы. На одном мейнфрейме можно было при необходимости запускать несколько полностью независимых виртуальных машин для решения прикладных задач.

У классической схемы виртуализации есть одна тонкость – чтобы ее можно было реализовать на конкретной аппаратуре, все инструкции процессора, которые могут изменить состояние ресурсов машины, должны быть привилегированными. Также не должно быть инструкций, которые ведут себя по-разному в зависимости от состояния этих ресурсов (например, различных системных регистров). В частности, не должно быть инструкций, которые работают по-разному на разных уровнях привилегий, не вызывая аппаратных исключений.

В более формальном виде требования к платформе, необходимые для создания на ней эффективного гипервизора, были впервые сформулированы разработчиками систем виртуализации IBM и по их именам получили название критерия Попека – Голдберга.



Развитие средств поддержки виртуализации на уровне аппаратуры способствовало росту облачных технологий

Эти требования достаточно очевидны, однако для многих реальных процессоров они не выполняются, особенно если разработчики последних не предполагают, что их изделия будут использоваться для развертывания виртуальных сред. В число таких процессоров попали Intel 8086 и все его потомки, выпущенные в течение почти 27 лет после его появления – до реализации аппаратной поддержки виртуализации. Более того, со временем создание виртуальных машин для архитектуры Intel x86 только затруднялось, в частности, после внедрения режима DMA, позволяющего работать с устройствами в обход процессора.

Долгое время процессоры архитектуры x86 действительно использовались только в персональных компьютерах, но Intel все активнее проявлял себя на рынке серверов, где виртуализация очень даже востребована. Создать классический гипервизор на платформе x86 было невозможно, но ведь не обязательно следовать уже известной модели. Не дожидаясь помощи от производителей процессоров, сторонние разработчики предложили альтернативные варианты запуска ВМ, обходящие ограничения аппаратуры.

Далее мы расскажем о наиболее популярных походах, а также о том, какую помощь разработчики виртуализированных сред получили в конце концов от производителей процессоров. Но прежде – более подробная информация о том, чем плоха изначальная архитектура x86 с точки зрения виртуализации.

«Нехорошие» инструкции x86

Для разграничения привилегий в процессорах x86 используются так называемые кольца защиты. Всего их предусмотрено четыре, однако обычно используются только два: в кольце 0 (самом привилегированном) работает операционная система, в кольце 3 – пользовательские приложения.

Для реализации гипервизора по классической схеме ОС необходимо было бы переместить в кольцо с более высоким номером. Но одной из проблем платформы x86 является то, что номер кольца содержится в сегментном регистре CS (и SS, если он используется), и любой процесс может узнать номер кольца (то есть свой уровень привилегий) из этого регистра, не прибегая к помощи привилегированных инструкций, например, посредством прямого вызова MOV или через стек посредством PUSH/POP. А как поведет себя ОС, обнаружив, что работает не в нулевом кольце, вопрос сложный, особенно в случае закрытых систем. Ведь традиционно ядра ОС для x86 проектируются исходя из того, что они работают в кольце 0.

Схожая проблема имеется с инструкцией STR, возвращающей запрошенный процессом уровень привилегий (ведь ОС может не понять, почему она работает в кольце 3, хотя запросила кольц 0), с командами PUSHF и POPF (позволяющими посмотреть флаги, контролирующие состояние процессора), а также с инструкцией SMSW, считывающей Machine Status Word (флаг, на основе которого можно узнать, работает машина в защищенном или реальном режиме x86). Правда, SMSW поддерживается с давних времен исключительно для совместимости с процессорами 8086 и 80286 и в современных приложениях встречаться не должна, но береженого бог бережет, и при создании надежного гипервизора об этой инструкции следует помнить.

Следующей проблемой являются инструкции SGDT, SIDT и SLDT, считывающие адреса таблиц GDT (глобальные дескрипторы сегментов), LDT (локальные дескрипторы сегментов) и IDT (дескрипторы прерываний) из соответствующих регистров. Изменение этих адресов требует привилегий кольца 0, но вот считать их может любой процесс. Как следствие, гостевая ОС может считать адреса таблиц, относящиеся совсем к другой ВМ или даже к основной системе, что с немалой вероятностью приведет к краху как минимум приложения, а то и всей ВМ.

Есть в процессорах Intel и инструкции, обладающие определенным «интеллектом» и выполняющие только те действия, которые дозволены текущему уровню привилегий. Поскольку для разных уровней список таких действий может быть различен (например, некоторые системные флаги можно выставлять только из кольца 0), то инструкции ведут себя по-разному, будучи вызванными в разных кольцах. Никаких аппаратных исключений при этом не возникает. С одной стороны, такой дизайн в ряде случаев удобен для про-

граммистов, с другой, при построении гипервизора такое поведение является серьезной проблемой. К подобным инструкциям относятся LAR, LSL, VERR и VERW, выполняющие проверку соответствия уровней привилегий.

Наконец, «опасными» являются инструкции перехода (непосредственно JMP, вызов прерывания INT n, а также инструкции вызова процедуры и возврата из нее – CALL и RET), точнее, некоторые варианты их вызова, затрагивающие проверку или изменение уровня привилегий.

Виртуализация на платформе x86 имеет богатую историю, обусловленную изначальной неприспособленностью этой архитектуры для создания гипервизоров

Более детальный разбор инструкций архитектуры x86, препятствующих реализации схемы «Trap and Emulate», можно найти, например, здесь: https://www.usenix.org/legacy/events/sec2000/robin.html.

Перечень же основных инструкций таков:

- MOV
- PUSH, POP
- PUSHF, POPF
- SGDT, SIDT, SLDT
- SMSW
- STR
- CALL, JMP, INT n, RET
- LAR, LSL, VERR, VERW

Помимо основных инструкций, необходимо помнить и об их разновидностях – например, IRET и IRETD, PUSHFD и POPFD и так далее.

Динамическая бинарная трансляция

Первопроходцем в области эффективной виртуализации на платформе x86 стала компания VMware, реализовавшая динамическую трансляцию бинарных инструкций виртуальной машины. В предложенном компанией подходе гипервизор анализирует все инструкции, которые пытается выполнить гостевая ОС. Привилегированные инструкции, а также «нехорошие» непривилегированные команды заменяются «на лету» на обращения к гипервизору. «Хорошие» непривилегированные инструкции передаются на выполнение непосредственно процессору.

Трудно назвать революционной саму идею этого подхода, но при ее реализации на практике остро встает вопрос производительности – процесс перехвата и анализа инструкций не должен быть слишком ресурсоемким. В конце концов техника бинарной трансляции применялась и до VMware (и применяется до сих пор в ситуациях, когда требуется эмулировать аппаратную архитектуру на процессоре с несовместимым набором инструкций), но вот производительность ее оставляла желать лучшего. В частности, традиционно при бинарной трансляции код сначала компилируется в инструкции целевого процессора, а потом исполняется; такой подход влечет серьезные задержки в работе процессов гостевых ОС.

VMware неплохо справилась с этой технической задачей и даже запатентовала свои технологии. В частности, инженеры компании смогли реализовать эффективную трансляцию кода в динамике без необходимости надолго останавливать выполнение кода гостевых систем. Единственное ограничение – процессор, на котором работает гипервизор, должен иметь ту же архитектуру, что и эмулируемый. Единственной поддерживаемой архитектурой для VMware стала x86, впрочем, эта платформа уже тогда занимала львиную долю рынка.

Между тем не спали и конкуренты – практически в то же время появился VirtualPC от Connectix для Mac OS на платформе PowerPC, впоследствии портированный на Windows и купленный вездесущей Microsoft. Также заметную долю рынка заняли решения Parallels и открытый VirtualBox, нынче поддерживаемый Oracle.

Продукты, основанные на бинарной трансляции, выполняют эмуляцию всех необходимых устройств для каждой ВМ. Однако – дело хлопотное, и эмулировать что-то серьезное (например, современную видеокарту) практически нереально. Такая проблема решается установкой в гостевую ОС дополнений (таких как VMware Tools или VirtualBox Guest Additions), включающих в себя специализированные драйверы. Эти драйверы знают о том, что ОС работает внутри ВМ, а не на реальной машине, и вместо попытки обращения к реальным устройствам обращаются напрямую к гипервизору. Такой подход упрощает схему работы (гипервизору не надо перехватывать и анализировать инструкции), повышает производительность и дает возможность предоставить виртуальной машине нечто большее, чем видеокарта, поддерживающая режим VESA в разрешении 800x600. Впрочем, эти драйверы уже фактически относятся к другой технологии – паравиртуализации, речь о которой пойдет ниже.

Изначально решения наподобие VirtualBox и VirtualPC предназначались в первую очередь для домашних пользователей – благо персональные компьютеры достигли достаточных мощностей, чтобы запускать на них несколько ВМ для тех или иных целей. Однако с финансовой точки зрения этот рынок не самый привлекательный – гораздо интереснее выйти на уровень предприятий и серверов, требующих работы большого количества ВМ. Это прекрасно осознавали в VMware и в придачу к средствам виртуализации отдельных машин разработали инструменты для построения и управления большим набором виртуальных сред, то есть все для развертывания облачных инфраструктур.

Но на этом поприще у компании появились конкуренты, которые были не очень известны среди домашних пользователей, поскольку использовали другие технологии, не особо подходящие для применения.

Паравиртуализация

Одним из таких подходов явилась паравиртуализация, основанная на внесении модификаций в код гостевых ОС. Цель таких модификаций – замена инструкций, которые необходимо эмулировать, на прямые обращения к гипервизору, так что последнему не надо ничего перехватывать – ОС сама отдает ему управление в случае появления «неправильной»

инструкции. Именно так работают упомянутые выше специализированные драйверы в средах, использующих бинарную трансляцию, поэтому корректно их называть паравиртуализированными драйверами. Точно такие же драйверы используются и в системах с паравиртуализированным ядром.

Но при разработке систем паравиртуализации встает вопрос поддержки оборудования, ведь гипервизор должен уметь работать со всем спектром устройств, доступных для платформы x86. А для многих устройств требуются специальные драйверы, реализованные только для конкретных ОС. Как решается этот вопрос? Здесь показателен пример Xen, пионера паравиртуализации в промышленных масштабах для x86. Xen является компактным гипервизором, основанном на микроядре Nemesis. Конечно, производители оборудования драйверы для Nemesis делать не спешили, поэтому разработчики гипервизора решили использовать возможности существующих ОС. Для этого одной из ВМ, контролируемых Xen, предоставляются особые привилегии, а именно прямой доступ к оборудованию. Эта ВМ (в терминах Xen – домен 0, dom0) запускается первой, и функции работающей в ней ОС используются Xen при общении с аппаратным обеспечением. Остальные ВМ (пользовательские домены, domU) работают в полностью виртуальном окружении. Проводя аналогии с другими средствами виртуализации, ОС в домене 0 можно условно назвать хост-системой, а все остальные – гостевыми.

Основным недостатком паравиртуализации является необходимость модификации ОС – как основной, так и гостевых. Как следствие, решения на основе паравиртуализации стали популярными только в открытых ОС. В частности, большое внимание Xen уделила компания Red Hat, включившая в свой коммерческий дистрибутив Red Hat Enterprise Linux (RHEL) ядра, модифицированные для работы с Xen. Затем подтянулись другие дистрибутивы Linux, а также родственные системы – NetBSD, OpenSolaris, Minix и прочие. Но отсутствие поддержки популярных закрытых ОС вкупе с относительной сложностью настройки ограничили применение Xen и схожих продуктов серверными решениями, домашние же пользователи с этими технологиями были малознакомы. Правда, в Microsoft Research в рамках исследовательского проекта добавили поддержку Xen в Windows XP, но это было скорее демонстрацией возможности, в реальную систему эти наработки не вошли.

Виртуализация на уровне ОС

Легковесной альтернативой бинарной трансляции и паравиртуализации является виртуализация на уровне ядра ОС, когда одно и то же ядро разделяется несколькими виртуальными машинами. Поскольку в кольце 0 работает только ядро, то необходимости перехвата инструкций в этом случае нет, проблемы работы с устройствами также не возникает. Однако надо научить ядро поддерживать параллельное выполнение не просто процессов, но полноценных ВМ (со своими системными библиотеками, окружением и так далее), то есть следует превратить ядро в гипервизор, не снимая при этом с него других обязанностей. Такие возможности присутствуют в ядрах открытых ОС – Solaris (Solaris Containers), FreeBSD (FreeBSD Jails) и Linux (Linux Containers, LXC). Также популярны решения от Parallels – OpenVZ для Linux и Virtuozzo для Windows [1].

Серьезным преимуществом такого легковесного подхода являются малые накладные расходы на запуск дополнительных ВМ. Однако есть и обратная сторона медали – запускать можно только ОС, способные работать с используемым ядром. То есть можно запустить несколько копий Linux (при условии, что версия ядра им подходит) либо несколько копий одной и той же версии Windows, но никак не Windows и Linux одновременно. Поскольку у домашних пользователей необходимость запустить несколько копий одной и той же ОС возникает редко, то у них подобные решения отклика не нашли. Зато они пользуются большой популярностью на серверах, где необходимо иметь именно много копий одной и той же ОС – например, в сервисах, предоставляющих веб-хостинг.

Технологии виртуализации от Intel (VT-x) и AMD (AMD-V) подразумевают дополнительное разграничение привилегий на аппаратном уровне

Аппаратная поддержка виртуализации

Наблюдая всевозрастающий интерес к виртуализации, основные разработчики процессоров x86 – Intel и AMD– решили пойти навстречу программистам и доработать архитектуры своих продуктов, существенно упростив создание гипервизоров.

Технологии виртуализации от Intel (VT-x) и AMD (AMD-V) подразумевают дополнительное разграничение привилегий на аппаратном уровне, а именно в процессорах вводятся два новых режима (фактически еще одно разграничение привилегий): корневой (root) и некорневой (nonroot). При этом традиционные кольца защиты x86 располагаются внутри некорневого уровня. Гостевые ОС, работающие в этом режиме, не заметят никакой разницы по сравнению с работой на процессоре без поддержки виртуализации, в частности, они по-прежнему будут работать в кольце 0.

Корневой режим предназначен специально для размещения гипервизора. При возникновении ситуаций, определяемых посредством соответствующих управляющих структур (например, при вызове тех или иных инструкций), работающий в некорневом режиме процесс останавливается, а управление передается гипервизору. Тот, в свою очередь, выполняет необходимые действия по эмуляции инструкций и возвращает управление в некорневой режим.

Первые реализации аппаратной поддержки виртуализации были сосредоточены на перехвате инструкций, то есть они нацеливались прежде всего на виртуализацию процессора. Однако со временем производители добавили средства работы с памятью и с устройствами, предоставив возможность создавать гипервизоры, контролирующие практически все активности системы и даже предоставляющие виртуальным машинам доступ к реальным устройствам.

Более того, в рамках рабочей группы PCI SIG IOV (PCI Special Interest Group, I/O Virtualization) разработаны специ-

фикации для устройств PCI Express, описывающие поддержку работы с несколькими ВМ на уровне самих устройств.

Отметим, что предложенные Intel и AMD решения для краткости часто называют «аппаратной виртуализацией» (как в иностранных, так и в русскоязычных источниках). Однако это словосочетание не совсем корректно и нередко вызывает неверное представление, что якобы уже внутри процессора присутствует некоторый гипервизор. Но это заблуждение не имеет ничего общего с реальностью – производители процессоров всего лишь добавили в свои изделия возможности, позволяющие реализовать гипервизор по классической схеме «Trap and Emulate». Встроенных гипервизоров в процессорах нет, их реализация – забота поставщиков решений виртуализации, причем задача эта очень даже непростая. Мы немного поговорим о ней в следующем выпуске «Пятой пары».

Не удивительно, что в первую очередь новые технологии нашли применение в уже существующих продуктах. Так, аппаратная поддержка виртуализации позволила решить одну из основных проблем производителей «паравиртуализированных» решений, а именно необходимость внесения изменений в гостевые системы. Ведь основная цель этих изменений заключалась в перехвате «нехороших» непривилегированных инструкций, а теперь эту задачу стало возможно переложить на сам процессор. Надо отметить, что такая возможность была реализована очень оперативно – так, уже через несколько месяцев после представления процессоров с аппаратной поддержкой виртуализации появилась новая версия Xen – 3.0, – позволившая запускать немодифицированные гостевые системы (в частности, MS Windows).

С большим энтузиазмом восприняли новшества и разработчики средств динамической трансляции. Например, именно задействование новых возможностей аппаратуры позволило реализовать запуск 64-битных гостевых ОС в VirtualBox. В целом поддержка со стороны «железа» позволяет избавиться от громоздкого и сложного кода, присущего бинарным трансляторам, нацеленным на высокую производительность и поэтому использующим всевозможные оптимизации. Показательна история QEMU – свободного эмулятора, изначально использовавшего бинарную трансляцию, но в плане производительности сильно уступавшего тому же VMware. При работе в 32-битных Linux-системах для QEMU был доступен модуль KQEMU, позволявший получить серьезный выигрыш в скорости работы. Однако, по словам разработчиков, код этого модуля со временем стал слишком сложным и запутанным, и они с удовольствием выкинули его, как только появилась возможность использовать аппаратуру.

Конечно, сразу же выкидывать старый код спешат далеко не все – в конце концов пока что в мире хватает машин и без аппаратной виртуализации, да и зачем менять хорошо работающий код (если его состояние не столь плачевно, как в случае с KQEMU)? К тому же при промышленном применении важную роль играет не только сам гипервизор, но и инструменты управления виртуальными средами, и здесь новые возможности процессоров вряд ли чем-то помогут.

Тем временем по мере совершенствования аппаратной поддержки виртуализации на рынке появились и новые игроки и продукты. В частности, классическая схема реализации гипервизора лежит в основе Microsoft Hyper-V, доступного для 64-битных редакций Windows Server начиная с версии 2008. В мире Linux аналогом является KVM (Kernel-based Virtual Machine) – модуль, добавляющий функционал гипервизора в ядро Linux и имеющийся практически во всех современных дистрибутивах. В частности, внимание Red Hat переключилось с Xen на KVM, и именно KVM пришел на смену упомянутому выше модулю KQEMU. Фактически KVM и QEMU образовали симбиоз, в котором первому ся перехвата и эмуляции инструкций процессора, а второй отвечает за эмуляцию различных устройств – жесткого диска, сетевой карты и так далее. При этом для каждой гостевой ОС запускается отдельный процесс QEMU, в рамках которого и выполняются коды ОС и ее приложений. Так что планирование ресурсов гостевой системы в случае KVM сводится к управлению ресурсами обычных процессов и полностью возлагается на ядро Linux.

Завершая тему аппаратной поддержки виртуализации, хотелось бы отметить, что сфера применения решений, предложенных Intel и AMD, может быть существенно шире создания виртуальных машин. Фактически имеющиеся технологии позволяют запускать на физической машине программу, привилегии которой выше, чем привилегии ОС. Такая программа может останавливать работу системы в случае возникновения определенных событий, совершать произвольные манипуляции с адресным пространством ОС и ее компонентов и возвращать ей управление.

Этот факт дал толчок новому направлению исследований в области информационной безопасности, ведь ПО, работающее в корневом режиме, невидимо для ОС и выполняющихся в ней программ. Для злоумышленников очень привлекательна перспектива запустить в корневом режиме вредоносное ПО, а для противоположной стороны было бы неплохо запустить инструменты обнаружения подобных программ. Исследования в этой области активно идут уже несколь-ко лет, и есть достаточно интересные практические на-работки.

***

Итак, виртуализация на платформе x86 имеет богатую историю, обусловленную изначальной неприспособленностью этой архитектуры для создания гипервизоров. За более чем десятилетний период было предложено несколько достаточно эффективных способов обхода ограничений дизайна процессоров, но затем свое слово сказали производители аппаратуры, устранив большинство проблем и упростив жизнь разработчикам ПО.

Аппаратная виртуализация не привела к исчезновению продуктов, созданных до ее появления. Скорее можно сказать, что она органично вписалась в эти продукты, позволив избавиться от наиболее неэффективных либо сложных в поддержке решений. Тем не менее развитие средств поддержки виртуализации на уровне аппаратуры способствовало бурному росту облачных технологий, которые сегодня находят применение не только на предприятиях, но и среди домашних пользователей. EOF

1. Штомпель И., Павел Емельянов: «Разработка ядра Linux – это общение в «клубе по интересам». //«Системный администратор», № 7-8, 2013 г. – С 62-66 (http://samag.ru/archive/article/2487).

Ключевые слова: виртуализация, процессоры, платформа x86, облачные технологии.