диске, у нас получается стоп-кадр работающей машины. Если программа устраивает полный кавардак на все еще работающей виртуальной машине, можно просто сделать откат к стоп-кадру и продолжить работу как ни в чем не бывало.

Наиболее простой способ создать стоп-кадр — это скопировать все, включая всю файловую систему. Но копирование диска в несколько терабайт может занять уйму времени, даже если это быстрый диск. К тому же приостанавливать работу машины на длительное время, пока проделывается все необходимое, нежелательно. Решение заключается в использовании технологий **копирования при записи** (copy on write), чтобы данные копировались только в случае крайней необходимости.

Создание стоп-кадров работает неплохо, но все же вызывает ряд вопросов. Что делать, если машина взаимодействует с удаленным компьютером? Мы можем сделать стоп-кадр системы и вернуть ее в прежнее состояние на более поздней стадии, но та часть, которая относится к обмену данными, уйдет в прошлое. Понятно, что эту проблему решить невозможно.

## 7.12. Изучение конкретных примеров: VMWARE

С 1999 года VMware, Inc. стала ведущим коммерческим поставщиком решений по виртуализации, предлагая продукты для настольных компьютеров, серверов, облаков, а теперь даже и сотовых телефонов. Компания поставляет не только гипервизоры, но и программы, управляющие виртуальными машинами в больших масштабах.

Изучение этого конкретного примера мы начнем с краткой истории становления компании. Затем будет дано описание VMware Workstation, гипервизора второго типа и первого продукта компании, всех сложностей в его конструкции и ключевых элементов этого решения. Затем будет дано описание происходившего в течение нескольких лет развития VMware Workstation. А в завершение будет дано описание ESX Server, гипервизора первого типа компании VMware.

### 7.12.1. Ранняя история VMware

Хотя идея использования виртуальных машин в 1960–1970-х годах была популярна как в компьютерной промышленности, так и в академических исследованиях, после 1980-х годов интерес к виртуализации был полностью утрачен, и на подъеме было производство персональных компьютеров. Только подразделение универсальных машин компании IBM все еще занималось виртуализацией. Действительно, компьютерные архитектуры, разработанные в то время, в частности архитектура x86 компании Intel, не предоставляла архитектурную поддержку виртуализации (например, они не отвечали критериям, выработанным Попеком и Голдбергом). Это весьма печальный факт, ведь центральный процессор 386, являвшийся полной переработкой процессора 286, был изготовлен спустя 10 лет после выхода статьи Попека и Голдберга и разработчики должны были лучше разбираться в поднятых в ней вопросах.

В 1997 году в Стэнфорде три будущих основателя компании VMware создали прототип гипервизора под названием Disco (Bugnion et al., 1997) с целью запуска товарных операционных систем (в частности, UNIX) на сверхбольшом микропроцессоре, разработанном в Стэнфорде, — на FLASH-машине. В ходе разработки этого проекта авторы поняли, что использование виртуальных машин может простым и элегантным

способом решить сразу несколько трудных проблем системного программного обеспечения: вместо попыток решения этих проблем внутри существующей операционной системы можно применить новое техническое решение на уровне, расположенном **ниже** существующей операционной системы. Работая над Disco, они пришли к ключевому выводу о том, что высокая сложность современных операционных систем затрудняет внедрение инноваций, а относительная простота монитора виртуальной машины и его положение в стеке программного обеспечения предоставляют мощную платформу для преодоления ограничений операционных систем. Хотя Disco был предназначен для очень больших серверов и разработан для MIPS-архитектуры, авторы поняли, что такой же подход может быть применен и к рынку, ориентированному на семейство процессоров x86, и оказаться коммерчески целесообразным.

Вследствие этого в 1998 году была основана компания VMware, Inc., имевшая цель привнесения виртуализации в архитектуру x86 и в индустрию персональных компьютеров. Первый продукт компании VMware (VMware Workstation) стал первым решением виртуализации, доступным на 32-разрядной платформе на основе архитектуры x86. Первый выпуск продукта состоялся в 1999 году в двух вариантах: **VMware Workstation for Linux**, представлявшем собой гипервизор второго типа, запускавшийся поверх основной операционной системы Linux, и **VMware Workstation for Windows**, который запускался поверх Windows NT. Оба варианта обладали одинаковой функциональностью: пользователь мог создавать несколько виртуальных машин путем предварительного указания характеристик виртуального оборудования (например, сколько памяти дать виртуальной машине или каким определить размер виртуального диска) с последующей возможностью установки операционной системы по их выбору на виртуальную машину, обычно с (виртуального) компакт-диска.

Продукт VMware Workstation был предназначен главным образом для разработчиков и IT-профессионалов. До внедрения виртуализации на столе у разработчика обычно стояли два компьютера, один со стабильными характеристиками, предназначенный для разработки, а второй с возможностью переустановки в случае необходимости системного программного обеспечения. При использовании виртуализации второй тестовой системой становилась виртуальная машина.

Вскоре компания VMware приступила к разработке второго, более сложного продукта, который был выпущен как ESX Server в 2001 году. В ESX Server использовался тот же механизм виртуализации, что и в VMware Workstation, но в пакет он входил в качестве части гипервизора первого типа. Иными словами, ESX Server запускался непосредственно на оборудовании, не требуя основной операционной системы. Гипервизор ESX был разработан для интенсивной консолидации рабочей нагрузки и содержал множество оптимизаций с целью обеспечения эффективного и справедливого распределения ресурсов (центрального процессора, памяти и ввода-вывода) среди виртуальных машин. Например, в этом продукте впервые была представлена концепция раздувания (ballooning) для перераспределения памяти между виртуальными машинами (Waldspurger, 2002).

ESX Server был нацелен на объединенный серверный рынок. До внедрения виртуализации IT-администраторы должны были, как правило, купить, установить и сконфигурировать новый сервер для каждой новой задачи или приложения, который приходилось запускать в дата-центре. В результате инфраструктура использовалась крайне неэффективно: серверы в то время обычно использовались на 10 % своих возможностей (в пиковые моменты нагрузки). С появлением ESX Server IT-администраторы могли

объединить множество независимых виртуальных машин на одном сервере, экономя время, деньги, пространство под компьютерные стойки и электроэнергию.

В 2002 году компания VMware представила свое первое управленческое решение для ESX Server, изначально называвшееся Virtual Center, а теперь имеющее название vSphere. Оно предоставляло единую точку управления для кластера серверов с запущенными виртуальными машинами: IT-администратор теперь мог просто войти в приложение Virtual Center и управлять тысячами виртуальных машин, запущенными на предприятии, отслеживая их работу и предоставляя новые виртуальные машины. С Virtual Center было предложено еще одно важное нововведение, **VMotion** (Nelson et al., 2005), позволяющее проводить живые миграции работающих виртуальных машин по сети. Впервые IT-администратор получил возможность переместить работающий компьютер с одного места на другое без необходимости перезагрузки операционной системы, перезапуска приложения и даже без потери сетевых подключений.

## 7.12.2. VMware Workstation

VMware Workstation стал первым продуктом виртуализации для 32-разрядных компьютеров семейства x86. Последующее внедрение виртуализации оказало значительное влияние на отрасль и научное компьютерное сообщество: в 2009 году Ассоциация по вычислительной технике (ACM) присудила его авторам престижную награду **ACM Software System Award** за VMware Workstation 1.0 для Linux. Исходный продукт VMware Workstation подробно описан в технической статье (Bugnion et al., 2012). Здесь будет приведено краткое изложение этой статьи.

Идея состояла в том, что уровень виртуализации может пригодиться на торговых платформах, созданных из центральных процессоров семейства x86 и первоначально работавших под управлением операционных систем Microsoft Windows (известных также как платформа **WinTel**). Преимущества от виртуализации могли помочь в принятии мер по отношению к ряду известных ограничений платформы WinTel, например к совместимости приложений, миграции операционной системы, надежности и безопасности. Кроме того, виртуализация может легко позволить сосуществование альтернативных операционных систем, в частности Linux.

Хотя целые десятилетия были потрачены на исследования и коммерческое развитие технологий виртуализации на универсальных компьютерах, вычислительная среда в семействе x86 имела существенные отличия, что потребовало новых исследований. Например, универсальные машины были **вертикально интегрированными**, что означало, что один и тот же производитель разработал оборудование, гипервизор, операционную систему и большинство приложений.

В отличие от этого индустрия x86 была (и продолжает быть) разделена как минимум на четыре категории:

- Intel и AMD делают процессоры;
- Microsoft предлагает Windows, сообщество разработчиков программ с открытым кодом предлагает Linux;
- третья группа компаний создает устройства ввода-вывода и периферийные устройства, а также соответствующие драйверы устройств;
- четвертая группа системных интеграторов, в числе которых HP и Dell, собирают компьютерные системы для розничной продажи.

Для платформы x86 виртуализация сначала должна быть внедрена без поддержки любого из этих игроков в мире индустрии.

Поскольку это разделение было суровой действительностью, продукт VMware Workstation отличался от классических мониторов виртуальных машин, разработанных как часть архитектуры одного производителя с явной поддержкой виртуализации. Вместо этого VMware Workstation был разработан для архитектуры x86 и того, что было создано компьютерной индустрией вокруг этой архитектуры. VMware Workstation справилась с новыми сложностями, объединив хорошо известные технологии виртуализации, технологии из других областей и новые технологии в единое решение.

А теперь мы рассмотрим конкретные технические сложности, возникшие при создании VMware Workstation.

## 7.12.3. Сложности внедрения виртуализации в архитектуру x86

Вспомним определение гипервизоров и виртуальных машин: гипервизоры применяют широко известный принцип **добавления уровня косвенного обращения** (adding a level of indirection) к области компьютерного оборудования. Они предоставляют абстракцию **виртуальных машин**: нескольких копий основного оборудования, на каждой из которых запущен независимый экземпляр операционной системы. Виртуальные машины изолированы от других виртуальных машин, каждая из них появляется в виде дубликата основного оборудования и в идеале работает с той же скоростью, что и реальная машина. VMware адаптировала эти основные атрибуты виртуальной машины к целевой платформе на базе x86 следующим образом:

- **Совместимость**. Понятие «практически идентичная среда» означает, что любую операционную систему под x86 и все ее приложения можно будет запускать в качестве виртуальной машины без модификаций. Гипервизор необходим для обеспечения достаточной совместимости на уровне оборудования таким образом, чтобы пользователи могли работать на любой операционной системе (вплоть до обновленной и исправленной версии), которую они пожелали установить на конкретной виртуальной машине, без каких-либо ограничений.
- **Производительность**. Издержки от применения гипервизора должны быть довольно низкими, чтобы виртуальную машину можно было использовать в качестве первичной рабочей среды. Разработчики VMware поставили себе цель добиться работы с большой нагрузкой практически на обычных скоростях, а в худшем случае запускать программы на самых последних процессорах с производительностью, аналогичной той, с которой они выполнялись на обычном оборудовании на ближайшем предыдущем поколении процессоров. Такая постановка задачи строилась на наблюдении, что подавляющая часть программного обеспечения под x86 не разрабатывалась для работы только на самых последних поколениях центральных процессоров.
- **Изолированность**. Гипервизор должен обеспечить изолированность виртуальной машины, не выстраивая никаких предположений насчет запускаемых внутри нее программ. То есть гипервизор должен иметь полный контроль над ресурсами. Программное обеспечение, работающее внутри виртуальных машин, должно быть лишено возможности любого доступа, позволяющего ему вмешиваться в работу гипервизора. Кроме того, гипервизор должен гарантировать закрытость всех дан-

ных, не принадлежащих виртуальной машине. Гипервизор должен предполагать, что гостевая операционная система может быть инфицирована неизвестным вредоносным кодом (что сегодня вызывает намного большие опасения, чем во времена универсальных машин).

Между этими тремя требованиями возникало неизбежное противоречие. Например, полная совместимость в конкретных областях могла привести к чрезмерному влиянию на производительность, в таких случаях разработчики компании VMware вынуждены были идти на компромисс. Но при этом ими исключались любые компромиссы, способные поставить под угрозу изолированность или сделать гипервизор уязвимым для атак вредоносного кода гостевой операционной системы. В целом, возникли четыре основные проблемы:

1. **Архитектура x86 была невиртуализируемой**. Она содержала чувствительные к виртуализации непривилегированные инструкции, которые нарушали выработанные Попеком и Голдбергом критерии для строгой виртуализации. Например, инструкция POPF имеет разную (а также неперехватываемую) семантику в зависимости от того, разрешено текущей программе выключать прерывания или нет. Это исключает традиционный подход к виртуализации, использующий перехват и эмуляцию. Даже инженеры из компании Intel были убеждены, что их процессоры практически невозможно виртуализировать.
2. **Архитектура x86 была слишком сложна**. Архитектура x86 была широко известной своей сложностью CISC-архитектурой, включая унаследованную поддержку на многие десятилетия обратной совместимости. На протяжении многих лет были внедрены четыре основных режима операций (реальный, защищенный, v8086 и управления системой), каждый из которых по-разному включал модель аппаратной сегментации, механизмы страничной организации памяти, защитные кольца и функции безопасности (например, шлюзы вызова).
3. **У машин x86 были разные периферийные устройства**. Хотя у процессоров x86 было всего два основных производителя, персональные компьютеры в течение всего времени выпуска могли содержать огромное разнообразие плат расширения и устройств и у каждого были собственные драйверы устройств от конкретных производителей. Виртуализировать все эти периферийные устройства было невозможно. Последствия носили двойственный характер: они относились как к внешнему интерфейсу (виртуальному оборудованию, видимому в виртуальных машинах), так и к внутреннему интерфейсу (реальному оборудованию, которым гипервизор должен был иметь возможность управлять) периферийных устройств.
4. **Нужна была модель, не требующая особого пользовательского опыта**. Классические гипервизоры устанавливались на производстве аналогично прошивкам, имеющимся в современных компьютерах. Поскольку VMware была новой компанией, ее пользователям приходилось добавлять гипервизоры к уже существующим системам. Чтобы стимулировать внедрение своего продукта, компании VMware понадобилась модель доставки программного обеспечения, не требующая особого опыта для установки.

## 7.12.4. VMware Workstation: обзор решения

В данном разделе дается описание на высоком уровне способов решения в VMware Workstation тех сложностей, которые были перечислены в предыдущем разделе.

VMware Workstation является гипервизором второго типа, состоящим из различных модулей. Одним из важных модулей является VMM, отвечающий за выполнение инструкций виртуальной машины. Вторым важным модулем является VMX, который взаимодействует с основной операционной системой.

Сначала в разделе будет рассмотрено, как VMM решает проблему неприспособленности x86-архитектуры к виртуализации. Затем будет дано описание стратегии, сконцентрированной на операционной системе и используемой разработчиками в течение всей стадии разработки. После этого будет рассмотрена конструкция платформы виртуального аппаратного обеспечения, которая берет на себя половину всех сложностей, связанных с разнообразием периферийных устройств. И наконец, будет рассмотрена роль в VMware Workstation основной операционной системы, в частности взаимодействие между компонентами VMM и VMX.

### Виртуализация x86-й архитектуры

VMM запускает текущую виртуальную машину, позволяя ей двигаться дальше. VMM, который создан для виртуализированной архитектуры, использует технологию, известную как перехват и эмуляция, для непосредственного выполнения последовательности инструкций виртуальной машины, но безопасным образом, на оборудовании. При невозможности подобных действий одним из подходов было указание виртуализируемого поднабора процессорной архитектуры и портирование гостевой операционной системы на эту заново определенную платформу. Эта технология называется паравиртуализацией (Barham et al., 2003; Whitaker et al., 2002) и требует модификации операционной системы на уровне исходного кода. Точнее говоря, при паравиртуализации гостевая операционная система модифицируется во избежание выполнения тех действий, которые не могут быть обработаны гипервизором. В VMware паравиртуализация была невозможна из-за требований обеспечения совместимости и необходимости запуска операционных систем, чей исходный код был недоступен, в частности Windows.

Нужно было воспользоваться альтернативным подходом и провести полную эмуляцию. При этом инструкции виртуальных машин вместо непосредственного выполнения на оборудовании эмулировались VMM. При этом можно было добиться достаточной эффективности. Предыдущий опыт с машинным симулятором SimOS (Rosenblum et al., 1997) показал, что использование таких технологий, как динамическая двоичная трансляция, запущенных в виде программы, выполняемой в пользовательском режиме, может ограничить издержки от полной эмуляции пятикратным замедлением. При всей своей эффективности и несомненной пригодности в целях моделирования пятикратное замедление было абсолютно неприемлемым и не могло соответствовать желаемым требованиям производительности.

Решение данной проблемы было в сочетании двух основных идей. Во-первых, хотя для виртуализации всей архитектуры x86 технология непосредственного выполнения инструкций после их перехвата и эмуляции не всегда подходила, в отдельные моменты ее можно было применить. В частности, она могла использоваться в ходе выполнения тех прикладных программ, на долю которых приходится большая часть времени при соответствующих рабочих нагрузках. Дело в том, что инструкции, чувствительные к виртуализации, являются таковыми не всегда, а только при определенных обстоятельствах. Например, инструкция POPF чувствительна к виртуализации, когда от программного обеспечения ожидается возможность блокировки прерываний (например, при запуске операционной системы), но она нечувствительна к виртуализации,

когда программное обеспечение не может заблокировать прерывания (что случается при выполнении почти всех приложений на уровне пользователя).

На рис. 7.7 показаны модульные строительные блоки исходного монитора VMware VMM. Видно, что он состоит из подсистемы непосредственного выполнения, подсистемы двоичной трансляции и алгоритма решения, определяющего, какая из подсистем должна использоваться. Обе подсистемы зависят от ряда совместно используемых модулей, например для виртуализации памяти посредством теневых таблиц страниц или эмулирования устройств ввода-вывода.

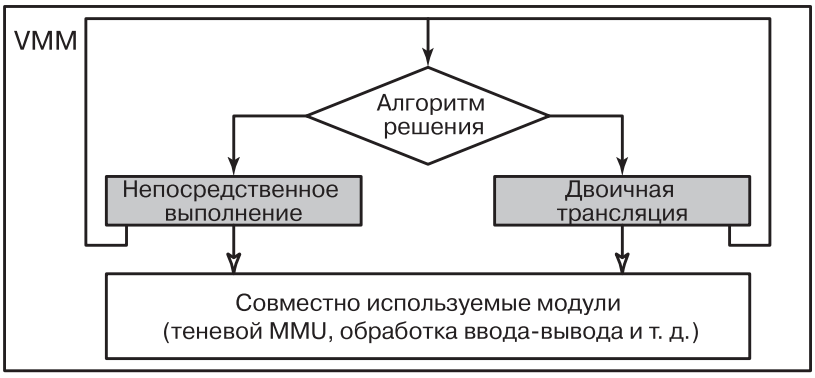


**Рис. 7.7.** Высокоуровневые компоненты монитора виртуальных машин VMware (за исключением аппаратной поддержки)

Предпочтительнее, конечно, использование подсистемы непосредственного выполнения, а подсистема динамической двоичной трансляции предоставляет резервный механизм, когда непосредственное выполнение невозможно. Такой случай представляется, к примеру, когда виртуальная машина находится в таком состоянии, при котором она может выдать чувствительную к виртуализации (служебную) инструкцию. Таким образом, каждая подсистема постоянно переоценивает алгоритм решения, чтобы определить возможность переключения подсистем (с двоичной трансляции на непосредственное выполнение) или необходимость такого переключения (с непосредственного выполнения на двоичную трансляцию). Этот алгоритм имеет ряд входных параметров, таких как текущее кольцо выполнения виртуальной машины, возможность включения прерываний на этом уровне и состояние сегментов. Например, двоичная трансляция должна использоваться при возникновении следующих обстоятельств:

- виртуальная машина в данный момент запущена в режиме ядра (кольцо 0 в архитектуре x86);
- виртуальная машина может заблокировать прерывания и выдать инструкции ввода-вывода (в архитектуре x86, когда уровень привилегий ввода-вывода установлен на уровне кольца);
- виртуальная машина в данный момент запущена в реальном режиме, кроме всего прочего, для BIOS используется устаревший 16-разрядный режим выполнения.

Фактический алгоритм решения содержит несколько дополнительных условий. Подробности можно найти в работе Bugnion et al. (2012). Интересно, что алгоритм не зависит от инструкций, которые сохранены в памяти и могут быть выполнены, он зависит только от значения нескольких виртуальных регистров, таким образом, он весьма эффективно может быть вычислен всего лишь за несколько инструкций.

Второй ключ к пониманию заключался в том, что при надлежащей конфигурации оборудования, особенно при осмотрительном использовании имеющегося в x86 механизма защиты сегментов, системный код при динамической двоичной трансляции также может выполняться на близких к исходным скоростях. Это сильно отличается от пятикратного замедления, обычно ожидаемого от машинных симуляторов.

Разницу можно объяснить путем сравнения того, как динамическая двоичная трансляция преобразует простую инструкцию обращения к памяти. Чтобы эмулировать эту инструкцию в программе, классическому двоичному транслятору, эмулирующему полный набор инструкций архитектуры x86, придется сначала проверить, попадает ли эффективный адрес в диапазон сегмента данных, затем преобразовать адрес в физический адрес и, наконец, скопировать слово, на которое была ссылка, в симулируемый регистр. Разумеется, все эти действия могут быть оптимизированы путем кэширования способом, весьма похожим на тот, которым процессор кэширует отображение в таблицах страниц в буфере ассоциативной трансляции. Но даже такая оптимизация приведет к расширению отдельных инструкций в последовательность инструкций.

Двоичный транслятор в программах подобных действий не совершает. Вместо этого он конфигурирует оборудование таким образом, чтобы эти простые инструкции могли быть заново выданы в виде идентичных инструкций. Это возможно только потому, что VMware VMM (компонентом которого является двоичный транслятор) имеет заранее настроенное под конкретную спецификацию виртуальной машины оборудование:

- VMM использует теневые таблицы страниц, гарантирующие, что блок управления памятью может использоваться напрямую (а не эмулироваться);
- VMM использует такой же подход с теневыми таблицами для таблиц дескрипторов сегментов (играющих большую роль в 16- и 32-разрядном программном обеспечении на более старых операционных системах для x86).

Разумеется, без сложностей и тонкостей не обошлось. Одним из важных аспектов конструкции является обеспечение целостности в песочнице виртуализации, то есть обеспечение того, что никакая программа, запущенная внутри виртуальной машины (включая и вредоносную программу), не сможет вмешиваться в работу VMM. Эта проблема обычно называется изоляцией сбоя программы и, если решение реализовано в программе, добавляет к каждому обращению к памяти издержки времени выполнения. Здесь также VMware VMM использует другой, основанный на оборудовании подход. Он разбивает адресное пространство на две разделенные зоны. Верхние 4 Мбайт адресного пространства VMM резервирует под собственные нужды. Тем самым для использования виртуальной машиной освобождается все остальное пространство (то есть 4 Гбайт – 4 Мбайт, если речь идет о 32-разрядной архитектуре). Затем VMM конфигурирует аппаратную часть, занимающуюся сегментацией, таким образом, чтобы никакие инструкции виртуальной машины (включая и те, что сгенерированы двоичным транслятором) никогда не получали доступ к верхней 4-мегабайтной области адресного пространства.

### Стратегия, сконцентрированная на гостевой операционной системе

В идеале VMM должна быть разработана так, чтобы не приходилось беспокоиться за запущенную на виртуальной машине гостевую операционную систему или за то, как эта система конфигурирует оборудование. Идея, положенная в основу виртуализации, заключается в создании интерфейса виртуальной машины, идентичного аппаратному интерфейсу, чтобы все программное обеспечение, запускаемое непосредственно на

оборудовании, могло запускаться также на виртуальной машине. К сожалению, этот подход реализуем на практике только при наличии виртуализируемой и простой архитектуры. В случае с x86 явной проблемой стала чрезвычайная сложность архитектуры.

Инженеры VMware упростили эту проблему, сфокусировавшись только на выборе поддерживаемых гостевых операционных систем. В первом выпуске VMware Workstation официально в качестве гостевых операционных систем поддерживались только Linux, Windows 3.1, Windows 95/98 и Windows NT. С годами с каждым пересмотром программного обеспечения к списку добавлялись новые операционные системы. Тем не менее эмуляция была вполне подходящей для запуска некоторых весьма неожиданных операционных систем, например MINIX 3, причем взятой прямо из коробки.

Такое упрощение не изменило общую конструкцию — VMM по-прежнему обеспечивал точную копию основного оборудования, но это помогло направить процесс разработки в нужное русло. В частности, инженерам пришлось позаботиться только о сочетаниях тех свойств, которые реально использовались поддерживаемыми операционными системами.

Например, архитектура x86 в защищенном режиме содержит четыре кольца привилегий (от 0 до 3), но ни одна из операционных систем практически не использует кольца 1 и 2 (за исключением давно изжившей себя операционной системы OS/2 от IBM). Следовательно, вместо того чтобы выяснять, как правильно виртуализировать кольца 1 и 2, VMware VMM просто содержит код для обнаружения попыток вхождения гостевой операционной системы в кольцо 1 или 2, и в таком случае монитор прекращает выполнение кода виртуальной машины. Таким образом не только был удален ненужный код, но, что более важно, VMware VMM получил возможность полагать, что кольца 1 и 2 никогда не будут использоваться виртуальной машиной, поэтому монитор может воспользоваться ими для собственных нужд. Фактически для виртуализации кода в кольце 0 входящий в состав VMware VMM двоичный транслятор работает в кольце 1.

### Платформа виртуального оборудования

До сих пор разговор в основном шел о проблеме, связанной с виртуализацией процессора x86. Но компьютер на основе семейства x86 состоит не только из процессора. В нем имеются также микропроцессорный набор, ряд прошивок и набор периферийных устройств ввода-вывода для управления дисками, сетевыми картами, приводами компакт-дисков, клавиатурой и т. д.

Большое разнообразие периферийных устройств ввода-вывода в персональных компьютерах x86 сделало невозможным соответствие виртуального оборудования реальному, основному оборудованию. В то время как моделей процессоров на рынке было не много и их возможности на уровне набора инструкций имели незначительные вариации, устройств ввода-вывода было несколько тысяч и большинство из них не имело общедоступной документации на интерфейс или функциональные возможности. Основным замыслом специалистов VMware был не отказ от попытки добиться соответствия виртуального оборудования конкретному основному оборудованию, а достижение постоянного соответствия определенной конфигурации, составленной из отобранных канонических устройств ввода-вывода. Затем гостевые операционные системы использовали собственные встроенные механизмы для обнаружения и работы с этими (виртуальными) устройствами.

Платформа виртуализации состояла из комбинации мультиплексированных и эмулированных компонентов. Мультиплексирование означало конфигурирование оборудования таким образом, чтобы оно могло непосредственно использоваться виртуальной машиной и совместно использоваться (в пространстве или времени) несколькими виртуальными машинами. Эмулирование означало экспортирование программной симуляции отобранных канонических компонентов оборудования виртуальной машине. В табл. 7.2 показано, что в VMware Workstation мультиплексирование использовалось для процессора и памяти, а эмулирование — для всего остального.

**Таблица 7.2.** Варианты конфигурации виртуального оборудования, характерные для раннего образца VMware Workstation, ca. 2000

| **Оборудование** | **Виртуальное оборудование (внешний интерфейс)** | **Внутренний интерфейс** |
|---|---|---|
| Мультиплексированное | Один виртуальный центральный процессор x86 CPU с одними и теми же расширениями набора инструкций, что и у центрального процессора основного оборудования | Работа регламентировалась основной операционной системой либо на однопроцессорной, либо на мультипроцессорной хост-машине |
| | До 512 Мбайт непрерывной динамической оперативной памяти | Распределялась и управлялась основной операционной системой (постранично) |
| Эмулированное | Шина PCI Bus | Полностью эмулируемая совместимая PCI bus |
| | 4x IDE-диска<br>7x Buslogic SCSI-диска | Виртуальные диски (хранящиеся в виде файлов) или непосредственный доступ к заданному простому устройству |
| | 1x IDE-привод компакт-дисков | ISO-образ или эмулируемый доступ к реальному компакт-диску |
| | 2x 1,44-мегабайтного привода гибких дисков | Физический привод гибких дисков или образ гибкого диска |
| | 1x VMware графическая карта с поддержкой VGA и SVGA | Запускалась в окне и в полноэкранном режиме. Для SVGA требовался гостевой драйвер VMware SVGA |
| | 2x последовательных порта COM1 и COM2 | Подключались к последовательному порту хост-машины или к файлу |
| | 1x принтер (LPT) | Мог подключаться к LPT-порту хост-машины |
| | 1x клавиатура (104 клавиши) | Полностью эмулировалась; события кода клавиши генерировались после их получения приложением VMware |
| | 1x мышь PS-2 | Аналогично клавиатуре |
| | 3x Ethernet-карты AMD Lance | Режим моста и режимы только для хост-машины |
| | 1x Soundblaster (звуковая карта) | Полностью эмулировалась |

Для мультиплексированного оборудования у каждой виртуальной машины имелась иллюзия наличия выделенного центрального процессора и конфигурируемого, но фиксированного количества непрерывной оперативной памяти, начиная с физического адреса 0.

Архитектурно эмуляция каждого виртуального устройства была разбита на компонент внешнего интерфейса, видимый виртуальной машине, и компонент внутреннего интерфейса, взаимодействующий с основной операционной системой (Waldspurger and Rosenblum, 2012). Внешний интерфейс по своей сути был программной моделью аппаратного устройства, которое могло управляться немодифицированными драйверами устройств, запущенными внутри виртуальной машины. Вне зависимости от конкретного физического оборудования хост-машины, внешний интерфейс всегда показывал одну и ту же модель устройства.

Например, первым внешним интерфейсом Ethernet-устройства была микросхема AMD PCnet «Lance», когда-то популярная карта расширения персонального компьютера со скоростью передачи данных 10 Мбит/с, а внутренний интерфейс обеспечивал сетевое подключение к физической сети хост-машины. По иронии судьбы, VMware долго сохраняла поддержку устройства PCnet и после того, как из обихода исчезли физические карты расширения Lance, и на самом деле достигалась скорость ввода-вывода на порядок выше 10 Мбит/с (Sugerman et al., 2001). Для устройств хранения информации исходными внешними интерфейсами были IDE-контроллер и Buslogic Controller, а внутренним интерфейсом был обычно либо файл в основной файловой системе, например виртуальный диск или образ ISO 9660, или же простой ресурс, такой как раздел диска или физический компакт-диск.

У разделения внешних и внутренних интерфейсов есть еще одно преимущество: виртуальная машина VMware может быть скопирована с одного компьютера на другой, возможно, с другими аппаратными устройствами. К тому же, поскольку виртуальная машина взаимодействует только с компонентами внешнего интерфейса, ей не придется устанавливать новые драйверы устройств. Это свойство, называемое **аппаратно-независимой инкапсуляцией** (hardware-independent encapsulation), дает сегодня огромные преимущества в серверных средах и облачных вычислениях. Оно позволяет вводить последующие инновации, такие как приостановка-возобновление, расстановка контрольных точек и незаметная миграция работающих виртуальных машин через физические границы (Nelson et al., 2005). В облаке оно позволяет клиентам развертывать их виртуальные машины на любом доступном сервере, не вникая в детали основного оборудования.

### Роль основной операционной системы

Завершающим важным конструкторским решением в VMware Workstation стало развертывание этой системы поверх существующей операционной системы. Такое решение классифицируется как гипервизор второго типа. У этого выбора было два основных преимущества.

Во-первых, благодаря ему решалась вторая часть тех сложностей, которые были связаны с разнообразием периферийных устройств. VMware реализовала эмуляцию внешнего интерфейса различных устройств, но с опорой на драйверы устройств основной операционной системы для внутреннего интерфейса. Например, VMware Workstation будет вести чтение или запись файла в основной файловой системе, чтобы эмулиро-

вать устройство виртуального диска, или выводить графику в окно рабочего стола хост-машины, чтобы эмулировать видеокарту. Пока у основной операционной системы есть соответствующие драйверы, VMware Workstation может запускать виртуальные машины поверх нее.

Во-вторых, программный продукт может устанавливаться и восприниматься пользователем как обычное приложение, упрощая тем самым его освоение. Как и любое другое приложение, установщик VMware Workstation просто записывает файлы своих компонентов в существующую основную файловую систему, не вмешиваясь в конфигурацию оборудования (не требуя переформатирования диска, создания нового раздела диска или внесения изменений в настройки BIOS). Фактически система VMware Workstation может устанавливаться и приступать к запуску виртуальных машин, не требуя никаких перезагрузок основной операционной системы, по крайней мере там, где в ее роли выступает Linux.

Но обычное приложение не имеет необходимых методов, и для того, чтобы гипервизор мультиплексировал ресурсы центрального процессора и памяти, нужны API-функции, обеспечивающие близкий к обычному уровень производительности. В частности, рассмотренная ранее основа технологии виртуализации машин семейства x86 работает только в том случае, если VMM запущен в режиме ядра и способен контролировать все аспекты процессора без всяких ограничений, включая возможность изменять адресное пространство (создавать теневые таблицы страниц), таблицы сегментов и все обработчики прерываний и исключений.

У драйвера устройства имеется более прямой доступ к оборудованию, особенно если он запущен в режиме ядра. Хотя он способен (теоретически) выдавать любые привилегированные инструкции, на практике от драйвера устройства ожидается взаимодействие с его операционной системой с использованием четко определенных API-функций при абсолютной невозможности произвольной переконфигурации оборудования. А поскольку гипервизоры предназначены для основательной переконфигурации оборудования (включая все адресное пространство, таблицы сегментов, обработчики исключений и прерываний), запуск гипервизора в виде драйвера устройства также был нереальным вариантом.

Поскольку основными операционными системами ни одно из этих предположений не поддерживается, запуск гипервизора как драйвера устройства (в режиме ядра) также не подходил.

Эти строгие требования привели к разработке размещаемой архитектуры — VMware Hosted Architecture. В ней программное обеспечение разбито на три отдельных явно выраженных компонента (рис. 7.8).

Каждый из этих компонентов выполняет различные функции и работает независимо от других компонентов:

- Программа, запускаемая в пространстве пользователя (**VMX**), которую пользователь воспринимает как программу VMware. VMX выполняет все функции пользовательского интерфейса, запускает виртуальную машину, а затем выполняет основную часть эмуляции устройств (внешний интерфейс) и совершает обычные системные вызовы основной операционной системы для взаимодействий во внутреннем интерфейсе. Обычно это один многопоточный VMX-процесс для каждой виртуальной машины.

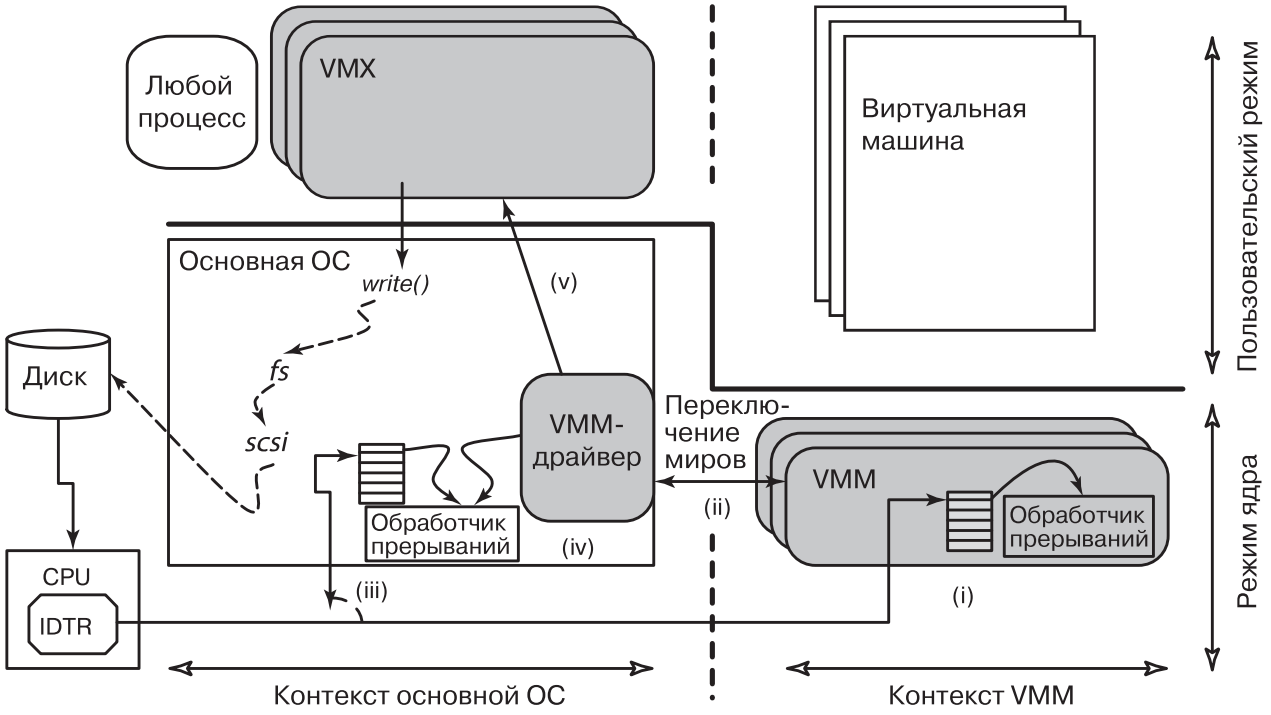


**Рис. 7.8.** VMware Hosted Architecture и три ее компонента: VMX, VMM-драйвер и VMM

- Небольшой драйвер устройства, выполняемый в режиме ядра (**VMX-драйвер**), устанавливаемый в основную операционную систему. Он используется в основном для получения возможности запуска VMM путем временной приостановки всей основной операционной системы. В операционную систему обычно во время начальной загрузки устанавливается один VMX-драйвер.
- VMM, включающий все программное обеспечение, необходимое для мультиплексирования центрального процессора и памяти, в том числе обработчики исключений, обработчики перехватов и эмуляции, двоичный транслятор и модуль теневой страничной организации памяти. VMM запускается в режиме ядра, но не работает в контексте основной операционной системы. Иными словами, он не может напрямую полагаться на службы, предлагаемые основной операционной системой, но он также не связан какими-либо правилами или соглашениями, декларируемыми основной операционной системой. Для каждой виртуальной машины имеется один экземпляр VMM, создаваемый при запуске виртуальной машины.

VMware Workstation выглядит так, будто запускается поверх существующей операционной системы, и, фактически ее компонент VMX запускается как процесс этой операционной системы. Но VMM работает на системном уровне, имея полный контроль над оборудованием и не испытывая никакой зависимости от основной операционной системы. На рис. 7.8 показаны взаимоотношения между объектами: два контекста (основной операционной системы и VMM) являются равноправными по отношению друг к другу, и у каждого есть компонент пользовательского уровня и компонент ядра. VMM при запуске (в правой половине рисунка) проводит переконфигурацию оборудования, обрабатывает все прерывания и исключения ввода-вывода и поэтому безопасным способом временно удаляет основную операционную систему из своей виртуальной памяти. Например, размещение таблицы прерываний настраивается внутри VMM путем назначения регистру IDTR нового адреса. И наоборот, когда работает основная

операционная система (левая половина рисунка), VMM и его виртуальная машина точно так же удаляются из ее виртуальной памяти.

Переход между этими двумя совершенно независимыми контекстами системного уровня называется **переключением миров** (world switch). Самим названием подчеркивается, что во время переключения миров все касающееся программного обеспечения изменяется в отличие от обычного переключения контекстов, реализуемого операционной системой. На рис. 7.9 показана разница между двумя видами переключений. Обычное переключение контекстов между процессами *А* и *Б* меняет местами пользовательскую часть адресного пространства и регистры двух процессов, но ряд важных системных ресурсов оставляет без изменений. Например, часть адресного пространства, принадлежащая ядру, идентична для всех процессов, также не изменяются обработчики исключений. В отличие от этого при переключении миров изменяется все: все адресное пространство, все обработчики исключений, привилегированные регистры и т. д. В частности, адресное пространство ядра основной операционной системы отображается только при работе в контексте основной операционной системы. После переключения миров в контекст VMM оно удаляется из адресного пространства целиком, освобождая пространство для работы как VMM, так и виртуальной машины. Хотя это может показаться довольно сложным процессом, его можно реализовать весьма эффективно и занять для его выполнения всего лишь 45 инструкций на языке машины x86.


**Рис. 7.9.** Разница между обычным переключением контекста и переключением миров

Внимательный читатель может удивиться: а как насчет адресного пространства ядра гостевой операционной системы? Ответ простой: оно является частью адресного пространства виртуальной машины и присутствует при работе в контексте VMM. Поэтому гостевая операционная система может использовать все адресное пространство, в частности те же места в виртуальной памяти, что и основная операционная система. При совпадении основной и гостевой операционных систем (например, обе Linux) именно так все и происходит. Разумеется, все это «просто работает», потому что есть два независимых контекста и между ними происходит переключение миров.

Затем тот же внимательный читатель может вновь удивиться: а как насчет области VMM, находящейся на верхушке адресного пространства? Как уже говорилось, это пространство резервируется самим VMM, и эта часть адресного пространства не может быть использована виртуальной машиной напрямую. К тому же эта небольшая

4-мегабайтная часть используется гостевой операционной системой крайне редко, так как каждое обращение к ней должно быть отдельно эмулировано и влечет за собой заметные программные издержки.

Вернемся к рис. 7.8: на нем показаны различные этапы того, что происходит в случае прерывания от диска, возникшего во время работы VMM (этап I). Разумеется, VMM не может обработать прерывание, потому что у него нет драйвера устройства из внутреннего интерфейса. На следующем этапе (II) VMM осуществляет переключение миров с возвращением основной операционной системы, а именно: код переключения миров возвращает управление VMware-драйверу, который на этапе III эмулирует аналогичное прерывание, выданное диском. Таким образом, на этапе IV обработчик прерывания, принадлежащий основной операционной системе, проходит всю свою логическую цепочку, как будто прерывание от диска произошло во время работы VMware-драйвера (а не VMM!). И наконец, на этапе V VMware-драйвер возвращает управление приложению VMX. К этому моменту основная операционная система может сделать выбор в пользу диспетчеризации другого процесса или же продолжить выполнение процесса VMware VMX. При продолжении выполнения процесса VMX он после всего этого возобновит работу виртуальной машины путем выдачи специального вызова в драйвер устройства, который сгенерирует переключение миров для возвращения в контекст VMM. Как видите, это весьма ловкий прием, скрывающий весь VMM и виртуальную машину от основной операционной системы. А что более важно, он предоставляет VMM полную свободу по перепрограммированию оборудования согласно его предпочтениям.

### 7.12.5. Развитие VMware Workstation

В те десять лет, что последовали за разработкой исходного VMware Virtual Machine Monitor, технологические перспективы сильно изменились.

Архитектура, применяющая основную операционную систему, используется и по сей день для таких самых последних интерактивных гипервизоров, как VMware Workstation, VMware Player и VMware Fusion (продукт, предназначенный для основных операционных систем Apple OS X), и даже для продуктов компании VMware, предназначенных для сотовых телефонов (Barr et al., 2010). Переключатель миров и его способность отделять контекст основной операционной системы от контекста VMM остался основным механизмом сегодняшних продуктов VMware, применяющих основные операционные системы. Например, несмотря на то что с годами реализация переключателя миров получила развитие, для поддержки 64-разрядных систем до сих пор остается в силе основная идея полного разделения адресных пространств для основной операционной системы и VMM.

В отличие от этого с появлением средств аппаратного содействия виртуализации довольно резко изменился подход к виртуализации архитектуры x86. Аппаратные средства содействия виртуализации, такие как Intel VT-x и AMD-v, были представлены в двух фазах. Первая фаза, стартовавшая в 2005 году, была разработана с явной целью обойтись либо без паравиртуализации, либо без двоичной трансляции (Uhlig et al., 2005). Вторая фаза, стартовавшая в 2007 году, предоставляла аппаратную поддержку в MMU в форме вложенных таблиц страниц. Тем самым исключалась необходимость в обслуживании программным способом теневых таблиц страниц. Сегодня, когда процессор поддерживает как виртуализацию, так и вложенные таблицы страниц, гипервизоры VMware главным образом применяют подход, основанный на использовании оборудования с перехватом и эмуляцией (в формулировке Попека и Голдберга сорокалетней давности).

Появление аппаратной поддержки, содействующей виртуализации, оказало существенное влияние на принятую в VMware стратегию, сконцентрированную на гостевой операционной системе. В исходной VMware Workstation эта стратегия использовалась для резкого снижения сложностей реализации за счет совместимости с полной архитектурой. Сегодня же полная архитектурная совместимость предвидится благодаря аппаратной поддержке. Текущий взгляд VMware на концентрацию на гостевой операционной системе свелся к оптимизации производительности для выбранных гостевых операционных систем.

## 7.12.6. ESX-сервер: гипервизор первого типа компании VMware

В 2001 году компания VMware выпустила совершенно другой продукт, названный ESX Server и нацеленный на рынок серверных машин. Здесь инженеры VMware применили иной подход: вместо создания решения второго типа, запускаемого поверх основной операционной системы, они решили создать решение первого типа, которое бы работало непосредственно на оборудовании.

На рис. 7.10 показана высокоуровневая архитектура ESX Server. В ней сочетаются уже существующий компонент, VMM, и настоящий гипервизор, запускаемый непосредственно на голом оборудовании. VMM выполняет те же функции, что и в VMware Workstation, состоящие в запуске виртуальной машины в изолированной среде, представляющей собой дубликат архитектуры x86. Собственно говоря, VMM-мониторы, используемые в двух продуктах, имели один и тот же базовый исходный код и во многом были похожи друг на друга. ESX-гипервизор заменял основную операционную систему. Но вместо реализации полной функциональности, ожидаемой от операционной системы, его цель заключалась в запуске различных экземпляров VMM и эффективном управлении физическими ресурсами машины. Поэтому ESX Server содержал обычные подсистемы, характерные для операционных систем, такие как планировщик задач центрального процессора и подсистема ввода-вывода, и при этом каждая подсистема была оптимизирована для запуска виртуальных машин.

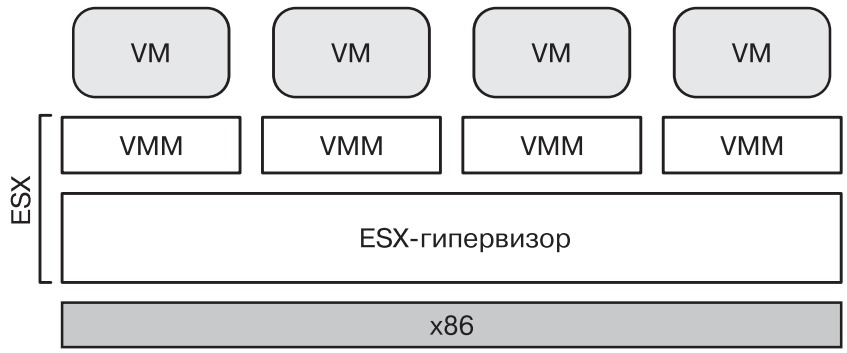


**Рис. 7.10.** ESX Server: гипервизор первого типа от компании VMware

Отсутствие основной операционной системы потребовало от VMware непосредственного решения сформулированных ранее проблем разнообразия периферийных устройств и наличия опыта у пользователей. Для решения проблемы разнообразия периферийных устройств компания VMware ввела следующее ограничение: запускать ESX Server можно было только на широко известных и сертифицированных серверных платформах, для которых у нее имелись драйверы устройств. Что же касается наличия опыта у пользователей, ESX Server (в отличие от VMware Workstation) требовал от пользователей устанавливать образ новой системы в загрузочный раздел диска.

Несмотря на недостатки, компромисс имел смысл для специализированного развертывания виртуализации в дата-центрах, состоящих из сотен или тысяч физических серверов и зачастую из многих тысяч виртуальных машин. Такие развертывания в наши дни иногда называют закрытыми облаками. Здесь архитектура ESX Server предоставляет солидные преимущества с точки зрения производительности, масштабируемости, управляемости и функциональных возможностей, например:

- Планировщик заданий центрального процессора обеспечивает каждой виртуальной машине получение справедливой доли времени центрального процессора (во избежание зависания). Он сконструирован таким образом, что одновременно планируется работа разных виртуальных центральных процессоров данного мультипроцессора виртуальной машины.
- Диспетчер памяти оптимизирован под масштабируемость, в частности под эффективную работу виртуальных машин, даже если им нужно больше памяти, чем фактически доступно на компьютере. Для достижения такого результата в ESX Server было введено понятие раздувания (ballooning) и прямого совместного использования страниц для виртуальных машин (Waldspurger, 2002).
- Подсистема ввода-вывода оптимизирована под высокую производительность. Хотя VMware Workstation и ESX Server зачастую совместно используют одни и те же эмулируемые компоненты внешнего интерфейса, внутренние интерфейсы у них совершенно разные. В VMware Workstation весь поток ввода-вывода проходит через основную операционную систему и ее API, что часто приводит к дополнительным издержкам. Особенно явно это проявляется в случае с устройствами сетевого обмена и устройствами хранения данных. В ESX Server эти драйверы устройств запускаются непосредственно в ESX-гипервизоре, не требуя переключения миров.
- Внутренние интерфейсы обычно также полагались на абстракции, предоставляемые основной операционной системой. Например, VMware Workstation сохраняет образы виртуальных машин в виде обычных (но очень больших) файлов на основной файловой системе. В отличие от этого у ESX Server имеется VMFS (Vaghani, 2010) — файловая система, специально оптимизированная для хранения образов виртуальных машин и обеспечения высокой пропускной способности системы ввода-вывода. Это позволяет достичь экстремальных уровней производительности. Например, компания VMware еще в 2011 году демонстрировала, что один ESX Server может выдать 1 млн операций с диском в секунду (VMware, 2011).
- ESX Server упрощает введение новых возможностей, требующих жесткой координации и специальной конфигурации нескольких компонентов компьютера. Например, в ESX Server было представлено средство VMotion — первое виртуализационное решение, способное осуществить миграцию работающей виртуальной машины, не останавливая ее, с одной машины, на которой запущен ESX Server, на другую машину с запущенным ESX Server. Успех такой миграции требовал координации диспетчера памяти, планировщика задач центрального процессора и сетевого стека.

С годами к ESX Server добавлялись новые свойства. ESX Server превратился в ESXi, компактную альтернативу, небольшой размер которой позволял выполнять ее предустановку в прошивке серверов. Сегодня ESXi является наиболее важным продуктом компании VMware и служит основой набора vSphere.