Два способа организации механизма полносистемного детерминированного воспроизведения в симуляторе QEMU

К. Батузов, П. Довгалюк, В. Кошелев, В. Падарян
{batuzovk, Pavel.Dovgaluk, vedun, vartan}@ispras.ru

Аннотация. В работе представлены два подхода к реализации полносистемного детерминированного воспроизведения в симуляторе QEMU, которые отличаются тем, какие компоненты виртуальной машины включаются в воспроизведение. Оба предложенных подхода позволяют воспроизводить выполнение гостевой ОС и прикладных программ без модификации их кода. Они не предполагают никаких ограничений на аппаратное окружение виртуальной машины. В частности, поддерживаются устройства, использующие в своей работе DMA. Оцениваются достоинства и недостатки этих подходов, оценивается уровень накладных расходов при записи журнала невоспроизводимых событий.

Ключевые слова: QEMU, детерминированное воспроизведение, программный симулятор

1. Введение

За последнее десятилетие виртуальные машины получили широкое распространение среди пользователей, как Linux, так и Windows. Основным сценарием использования можно считать развертывание виртуальной платформы (virtual appliance) на которой устанавливается некоторое ПО, требующее определенное окружение, отличное от имеющегося на физической машине, либо намеренная изоляция установленного ПО от внешнего мира. Массовое использование виртуальных машин началось благодаря тому, что их производительность достигла достаточно высокого уровня. Такие техники, как бинарная трансляция и аппаратная виртуализация, позволяют получать накладные расходы в пределах нескольких десятков процентов. Помимо того, развитый графический интерфейс и устойчивость работы бесплатных виртуальных машин (VMware Player [1], Virtual Box [2], QEMU [3, 4]) значительно упростили их эксплуатацию, рядовой пользователь имеет возможность начать эксплуатировать машину без длительного предварительного обучения, непосредственно после скачивания дистрибутива с сайта разработчика.

Но виртуальные машины вошли не только в обиход обычных пользователей, но и самих разработчиков. Отладка ядерного кода еще в начале нулевых годов начала мигрировать с обычных программных отладчиков, использующих возможности уровня ОС и аппаратуры, на отладчики, работающие с виртуальными машинами. Такой подход к отладке необходим для поиска и устранения сложных ошибок, проявление которых зависит, в том числе и от влияния отладчика на изучаемую систему. В этом случае связка отладчик/виртуальная машина успешно применяется для отладки Гейзенбагов, ошибок в драйверах устройств, межпроцессном и межтредовом взаимодействии, отладки ПО при кроссплатформенной разработке. Последнее становится все более актуальным, поскольку последние годы стремительно растет рынок мобильных устройств. Ведущие производители предлагают разработчикам набор программных инструментов, в состав которых обязательно входит виртуальная машина, используемая для отладки.

Еще одним востребованным сценарием использования виртуальных машин является расследование инцидентов нарушения информационной безопасности и криминалистическая экспертиза. Широкое распространение информационных технологий привело к необходимости соблюдения определенных политик информационной безопасности при работе с персональными данными и секретной информацией. Виртуальные машины могут применяться как для ограничения действий, так и для фиксации происходящего в информационной системе с целью последующего анализа. Применяемые ранее механизмы журналирования действий пользователя привели к идее развития этого подхода, когда начальное состояние системы и журнал действий позволяют воспроизвести работу всей эмулируемой системы внутри виртуальной машины с высокой точностью.

Данная статья посвящена вопросу организации детерминированного воспроизведения работы программного симулятора (Рис. 1). Существуют закрытые коммерческие симуляторы [5], в которых данный механизм присутствует и используется для расширения возможностей отладки. Однако среди программ с открытым исходным кодом еще не было представлено соответствующей разработки, пригодной для промышленной эксплуатации. В статье рассматриваются два подхода к реализации детерминированного воспроизведения в симуляторе QEMU, оцениваются перспективы их развития.

Дальнейший материал организован следующим образом. В разделе 2 приводится сравнение известных механизмов детерминированного воспроизведения в программных симуляторах (виртуальных машинах). Раздел 3 описывает структуру симулятора QEMU. В разделе 4 описываются различные подходы, реализующие детерминированное воспроизведение в рамках симулятора QEMU. В разделе 5 рассматриваются детали реализации этих подходов в QEMU. В разделе 6 приводятся результаты численных экспериментов. В последнем разделе делаются заключительные выводы.

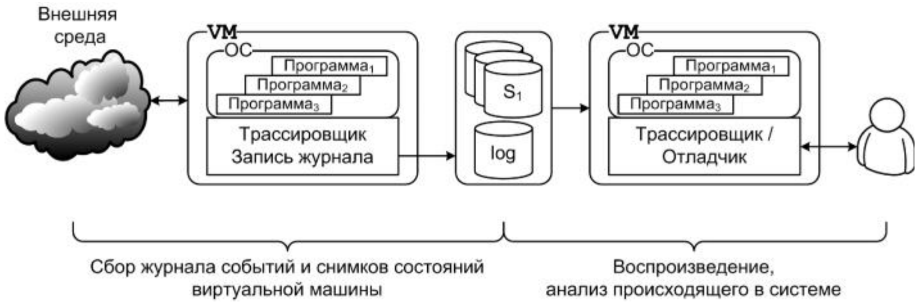
Внешняя среда
VM
ОС
Программа$_1$
Программа$_2$
Программа$_3$
Трассировщик Запись журнала
S$_1$
log
VM
ОС
Программа$_1$
Программа$_2$
Программа$_3$
Трассировщик / Отладчик
Сбор журнала событий и снимков состояний виртуальной машины
Воспроизведение, анализ происходящего в системе

Рис. 1. Детерминированное воспроизведение работы виртуальной машины.

2. Обзор средств трассировки

Известно несколько проектов, в которых детерминированное воспроизведение работы виртуальной машины используется для достижения перечисленных ранее целей.

В 2002 году был опубликован один из первых результатов: проект ReVirt [6], проводившийся в Университете Мичигана. Детерминированное воспроизведение на основе виртуальной машины было успешно применено для анализа работы зловредного кода.

В качестве основы была взята виртуальная машина UMLinux [7], записывается и воспроизводится состояние виртуальной аппаратуры, на которой работает гостевая операционная система. Виртуальная машина представляет собой процесс хостовой системы, для доставки в виртуальную машину прерываний от периферийных устройств и исключений используются сигналы. Реализованный механизм записи и воспроизведения использует аппаратные счетчики производительности, специфические для архитектуры x86, и требует доработки ядра хостовой операционной системы. Начальное состояние представляет собой образ диска в момент перед запуском системы, промежуточные состояния сохранятся как обычные контрольные точки, синхронизированные со счетчиком выполнившихся команд. Недетерминированные события: асинхронные аппаратные прерывания, пользовательский ввод и входящие сетевые пакеты. Помимо того, определенные машинные команды, вносящие недетерминированность (rdtsc, rdpmc), эмулируются виртуальной машиной, результаты их работы заносятся в журнал. В данной работе был достигнут малый уровень накладных расходов на саму виртуализацию (от 0% до 50%), накладные расходы на снятие журнала составляют примерно 5% по сравнению с обычной работой виртуальной машины. Размер журнала (после сжатия gzip) составляет порядка 200МБ в сутки, не считая сетевой трафик.

Сотрудниками компании VMware в 2007 году была представлена публикация [8], где описывался ReTrace, механизм детерминированного воспроизведения работы всей вычислительной системы: в работе определялся набор событий в виртуальной машине, который позволял для заданного начального состояния воспроизводить работу всей вычислительной системы с точностью до машинной инструкции. Поток записываемых в журнал событий достаточно компактен, авторами работы был достигнут результат в 5 байт на тысячу инструкций, замедление работы виртуальной машины составило примерно 5-10% при включенном сборе журнала. Полная загрузка и выключение виртуальной машины под управлением ОС Windows уместилась в сжатом журнале размером 776 KB. Прототипная реализация ReTrace использовала на втором этапе симулятор QEMU. Использовалась крайне упрощенная виртуальная машина, события доставлялись в систему состоящую только из центрального процессора и виртуальной памяти.

Вскоре была опубликована еще одна работа [9] с описанием системы динамического анализа бинарного кода Aftersight, построенной на основе механизма ReTrace, но непосредственного доступа к описанной системе предоставлено не было.

В результате приведенных исследований, в коммерческий продукт VMware Workstation 6 была включена технология Replay Debugging, предоставляющая пользователю интерактивный отладчик во время воспроизведения. Базовая технология ReTrace была доведена до промышленного уровня, но возможность получать детальную трассу временно присутствовала только в отладочной версии Workstation 6, и впоследствии была отключена.

Поддержка данной технологии была остановлена при переходе виртуальных машин VMware на аппаратную виртуализацию. Можно предположить, что причиной этому стала высокая сложность переноса технологии, особенно в части перехвата асинхронных событий в периферии, т.е. отслеживание непосредственных операций чтения/записи данных ой работы устройств с памятью.

Проект XenLR [10] (2008 г., Huazhong University of Science & Technology) реализует на основе Xen 3.10 крайне ограниченную виртуальную машину, в ней присутствует только два источника недетерминированных событий: клавиатура и таймер. Работает в таком окружении модельная операционная система MiniOS. Накладные расходы на снятие журнала фактически нулевые, объем сохраняемых данных составляет 1.4МБ в сутки. Данную работу следует рассматривать исключительно как подтверждение концепции, практическое применение из-за скудности виртуальной машины едва ли возможно.

Проект ExecRecorder [11] использует виртуальную машину Bochs для полносистемного воспроизведения работы. Создание промежуточных контрольных точек происходит через системный вызов fork, последующее переключение на контрольную точку происходит как активация соответствующего остановленного процесса. Предложенная система не поддерживает многоядерные системы и DMA-устройства, помимо того, сам симулятор Bochs вносит высокие накладные расходы на виртуализацию. Размер записываемого журнала составляет около 5.5ГБ в сутки.

Наиболее близкой работой является проект FREE [12], ведущийся в Национальном Университете Чиао Тун, Тайвань. В числе организаций, поддерживающих проект, присутствует известный центр информационной безопасности TRUST [13], расположенный в США.

В рамках данного проекта детерминированное воспроизведение реализуется в симуляторе с открытым исходным кодом QEMU. К числу недетерминированных событий относят аппаратные прерывания, чтение данных из устройств (как через порты, так и через отображение в память), DMA-транзакции. Причем вопрос поддержки последнего типа событий решен не был, в виртуальной системе не должно быть устройств, осуществляющих непосредственную запись данных в память.

При сборе журнала поддерживается счетчик инструкций, отражающий количество выполненных виртуальной машинной команд. Записываемые в журнал события снабжаются номером шага, что позволяет их доставлять при воспроизведении в соответствующие моменты времени. Состояние периферийных устройств при воспроизведении не восстанавливается, что позволяет сделать заключение о границе, по которой прошло детерминированное воспроизведение: как и в случае с ReTrace воспроизведение затрагивает только центральный процессор и оперативную память.

Несмотря на ограничения, гостевой системой может являться полноценная ОС, такая как Linux или Windows. Размер журнала относительно невелик и составляет около 500МБ в сутки. Накладные расходы на трассировку в публикации не приводятся.

Помимо рассмотренных работ нельзя не отметить соответствую возможность в коммерческом программном симуляторе SimNOW [5]. В состав симулятора входит «устройство» XTR, которое сохраняет начальное состояние виртуальной машины (дамп физической памяти, состояние процессора), а затем записывает в журнал все обращения к памяти, включая «DMA-транзакции», проходящие через северный мост. Сохраненные данные впоследствии могут быть использованы для воспроизведения работы системы и анализа ее состояния средствами плагинов-анализаторов, подключаемых к виртуальной машине через штатное API. К недостаткам следует отнести крайне медленную работу записи журнала, замедление составляет более пяти порядков, по сравнению с реальной системой, и значительный размер журнала. Положительной стороной является высокая достоверность симуляции.

Исходя из результатов рассмотренных работ, можно заключить следующее. Механизм полносистемного детерминированного воспроизведения, неоднократно разрабатывался и включался в состав различных виртуальных машин. Наилучшие результаты были получены в коммерческом симуляторе VMware Workstation, но на данных момент поддержка этого механизма остановлена. Промышленная эксплуатация детерминированного воспроизведения поддерживается коммерческим симулятором SimNOW.

Известно несколько открытых работ, предлагающих данный механизм, но перспективы доведения этого механизма до соответствующего технологического уровня (промышленной эксплуатации) не ясны.

Одна из работ [12] предлагает использовать в качестве основы для реализации полносистемного детерминированного воспроизведения программный симулятор QEMU, но в ней не решен ряд важных вопросов, в частности поддержка «DMA»-устройств, воспроизведение состояния периферийных устройств, и, главное, интеграция предложенного механизма с направлением развития симулятора.

Исходя из характеристик рассмотренных систем, можно сделать вывод о приемлемых характеристиках детерминированного воспроизведения: размер журнала составляет несколько ГБ в сутки, а накладные расходы при сборе журнала должны не превышать нескольких десятков процентов.

3. Структура симулятора QEMU

QEMU — эмулятор вычислительных систем, использующий в своей работе динамическую двоичную трансляцию. Основной цикл работы QEMU состоит из 4 этапов (рис. 2):

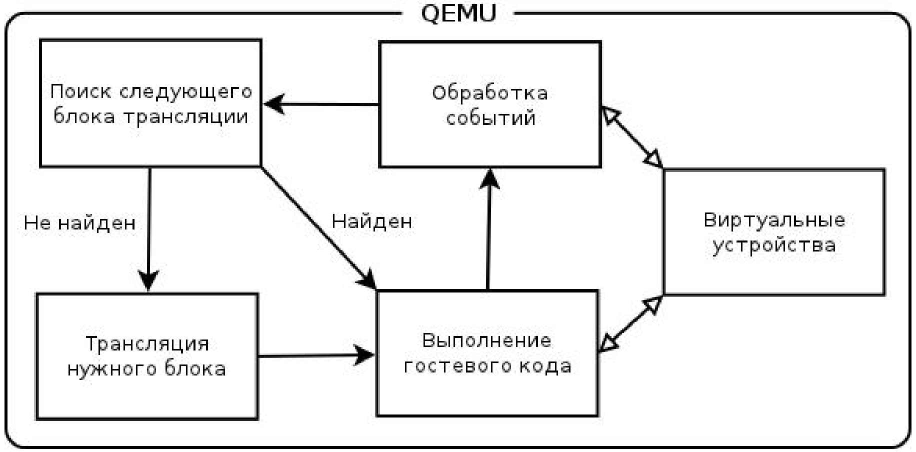

Рисунок 2 – Структура симулятора QEMU.

- выполнение оттранслированного гостевого кода,
- обработка событий, произошедших в системе,

- поиск следующего участка кода в кэше трансляций,
- трансляция нужного участка, если он не был найден в кэше.

Также QEMU создает виртуальные устройства, которые эмулируют работу реальной аппаратуры. Это позволяет ему выполнять код операционных систем и прикладных программ без каких-либо модификаций.

Теперь рассмотрим возможные источники недетерминизма в этой системе. Их можно разбить на две категории: внешние, возникающие из-за недетерминированности поступающих извне через виртуальные устройства данных, и внутренние, возникающие в результате недетерминированных взаимодействий внутри самого QEMU.

Дело в том, что в своей работе современные версии QEMU используют несколько потоков. Потоки могут создавать виртуальные устройства для того, чтобы асинхронно выполнять длительные синхронные операции. Начиная с версии 0.10.50 обработка событий и выполнение гостевого кода могут быть разнесены в различные потоки. Начиная с версии 1.0 такое поведение является единственным возможным и не может быть отключено. Потоки при взаимодействии используют различные механизмы, включая сигналы. Момент прихода сигнала является недетерминированным событием и повлиять на него из пользовательского приложения без модификации операционной системы невозможно.

Более того, сейчас ведутся исследования, целью которых является параллельное выполнение кода на разных ядрах эмулируемого процессора в различных потоках в QEMU [14]. Соответствующие изменения пока не включены в основную ветку разработки QEMU и в данной статье они рассматриваться не будут.

4. Сравнение двух подходов к детерминированному воспроизведению в QEMU

Детерминированное повторное воспроизведение программы основывается на записи всех «внешних» недетерминированных событий и обеспечении детерминированности всех «внутренних» процессов. Два рассматриваемых в данной статье подхода различаются тем, где проходит граница между «внутренними» процессами и «внешними» событиями.

Оба подхода, рассматриваемые в данной работе, воспроизводят поведение всей операционной системы с помощью эмулятора виртуальных машин QEMU. QEMU эмулирует центральный процессор гостевой системы, а также все периферийные устройства. Для передачи информации между устройствами и процессором используются виртуальные аналоги системной шины, прерываний и портов ввода-вывода. Некоторым устройствам, таким как сетевой адаптер или контроллер USB, может быть нужен доступ к внешним, по отношению к виртуальной машине, ресурсам. Для каждого типа таких устройств QEMU предоставляет интерфейс для взаимодействия с основной ОС: сетевой интерфейс, позволяющий виртуальной машине получать доступ к реальным сетям, проброс USB для доступа к физическим USB устройствам и так далее.

На рисунке 3 схематично изображены некоторые устройства, поддерживаемые QEMU и их взаимодействие. Общее количество устройств достаточно велико и не может быть изображено на одной схеме. Пунктирные линии 1 и 2 на данной схеме ограничивают ту часть виртуальной машины, которая будет детерминировано воспроизводиться в первом и втором подходах соответственно.

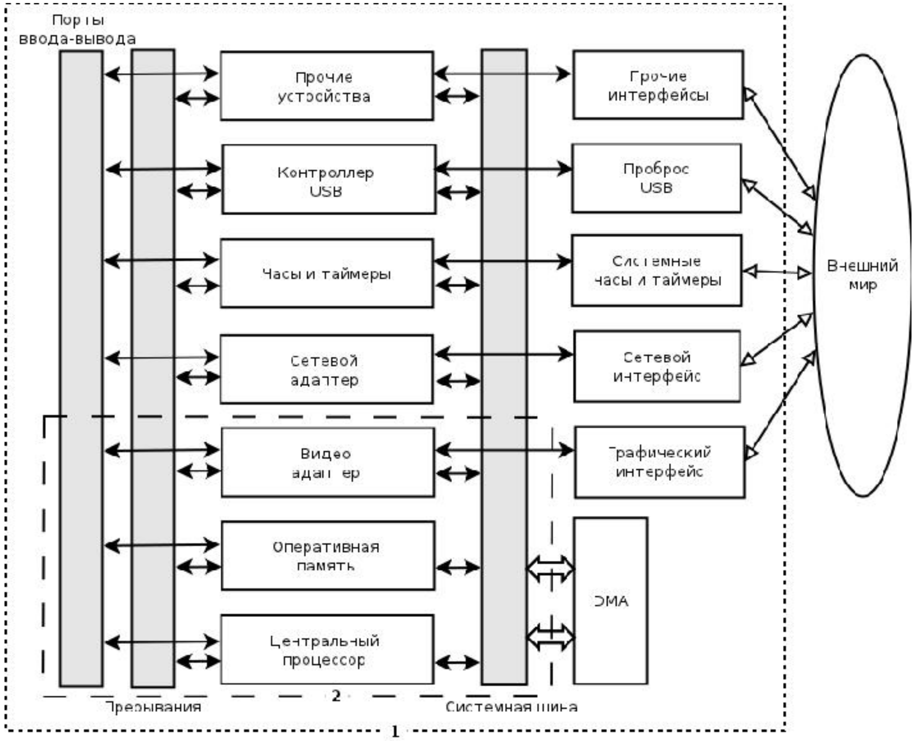

Рис. 3 – Взаимодействие различных частей QEMU.

В первом подходе предлагается детерминировано воспроизводить работу всей виртуальной машины. Граница между «внешними» событиями и «внутренними» процессами при этом проходит по интерфейсам QEMU для общения с основной ОС. В каждом из них необходимо реализовать запись и воспроизведение всех «внешних» событий, проходящих через данный интерфейс. Во «внутренней» части QEMU необходимо реализовать счетчик инструкций (для обеспечения связи времени прихода недетерминированного события и момента исполнения программы) и модифицировать аппарат прерываний так, чтобы во время воспроизведения прерывания приходили ровно в тот же момент выполнения программы, когда они пришли во время записи журнала событий.

Подход показал свою состоятельность на версии QEMU 0.13.0, однако он имеет сложности с переносом на более новые версии.

Преимуществами такого подхода являются:

- возможность воспроизводить состояние всей виртуальной машины, включая периферийные устройства,
- наглядность записываемых событий — каждое событие отображается на понятное пользователю событие, такое как приход сетевого пакета или нажатие кнопки на клавиатуре,
- очень низкая зависимость от конфигурации гостевой системы — достаточно только реализовать подсчет выполненных инструкций.

Однако он также обладает некоторыми недостатки:

- интерфейсов взаимодействия с основной ОС в QEMU достаточно много, они меняются и часть из них, например, проброс USB-устройств в гостевую систему, существенно зависит от основной ОС,
- появление нового интерфейса, например, позволяющего подсоединить виртуальную машину как USB устройство к основной ОС, потребует его доработки для использования с детерминированным воспроизведением,
- «внутренняя» часть очень большая и обеспечить ее детерминированность сложно.

Последние версии QEMU существенно используют многопоточность. В QEMU версии 1.0.1 выполнение гостевого кода и обработка событий разнесены в отдельные потоки, также отдельные потоки заводятся для реализации асинхронного ввода-вывода через синхронные средства. Различные потоки взаимодействуют между собой с помощью сигналов. Обеспечение детерминированности такой системы является задачей, сопоставимой по сложности с исходной. Таким образом последний недостаток является ключевым.

Второй подход предлагает минимизировать количество устройств, поведение которых воспроизводится. К ним относятся:

- центральный процессор — именно он выполняет гостевую программу и обеспечивает ее детерминированное исполнение,
- оперативную память, чтобы не увеличивать размер журнала излишне сильно за счет чтений и записей в нее гостевой программой,
- видео адаптер, чтобы во время воспроизведения пользователь мог следить за «ходом» исполнения программы.

Второй подход решает проблему недетерминизма «внутренних» процессов, вынося наиболее сложные части во «вне». Все процессы, являющиеся внутренними для второго подхода, выполняются в одном потоке. Также достоинством данного подхода является его независимость от значительной части периферийных устройств и их интерфейсов взаимодействия с основной ОС. Устройства, которые не используют прямую запись в оперативную память, могут быть использованы с детерминированным повторным воспроизведением без необходимости их дополнительной модификации.

Недостаткам данного метода является более сильная зависимость от конфигурации гостевой машины. Основная работа делается в процессоре, также необходимо модифицировать все устройства, использующие DMA транзакции - прямую запись в оперативную память гостевой системы. Вторым ограничением является плохая наглядность записанных в журнале событий. Она решается сохранением соответствия между записанными в журнале событиями и событиями, наблюдаемыми пользователем. Третьим ограничением является то, что при воспроизведении поддерживается только состояние процессора и оперативной памяти, состояние периферийных устройств выпадет из рассмотрения. Помимо того, возможно, потребуется переработка или написание собственного механизма сохранения состояний системы, т.к. конфигурация виртуальной машины будет отличаться при записи и воспроизведении и встроенным механизмом снимков состояния нельзя будет воспользоваться.

5. Детали реализации детерминированного воспроизведения в QEMU

Реализация первого подхода была подробно описана в статье [15], поэтому данный раздел будет посвящен второму подходу.

Второй подход рассматривает взаимодействие программного кода операционной системы с аппаратным окружением как недетерминированное внешнее событие. При этом все внутренние процессы, в случае однопроцессорной системы, будут детерминированными. Для однозначного определения момента возникновения недетерминированного события будем включать в информацию о взаимодействии количество выполненных инструкций с момента начала работы системы.

Реализация метода строится на том, что различных способов взаимодействия с аппаратным окружением у ОС фиксированное и очень небольшое количество. Полный набор этих способов является архитектурно-зависимым. В частности, для аппаратной платформы i386\x86-64 имеют место взаимодействия через:

- контроллер памяти (MMU),
- порты ввода\вывода (PIO),
- прерывания,
- машинно-зависимые регистры (MSR),

- специальные инструкции, такие как RDTSC, CPUID.

Нужно отметить, что хотя полный набор и является архитектурно-зависимым, некоторые его составляющие, а именно контроллер памяти и прерывания, на которые приходится большая часть взаимодействий, могут быть реализованы архитектурно-независимым способом.

5.1. Подсчёт количества выполненных инструкций

В своей работе QEMU использует динамическую двоичную трансляцию. Команды целевой архитектуры транслируются во внутреннее представление, из которого в последствии генерируется машинный код для реальной архитектуры. Для подсчета количества выполненных инструкций предлагается завести глобальную переменную внутреннего представления для хранения соответствующего значения. Во время трансляции во внутреннее представление необходимо в него добавлять код, который увеличивает этот счетчик на 1 при каждом выполнении каждой инструкции. Таким образом получается достаточно легковесный счетчик инструкций.

5.2. Машинно-зависимые регистры и специальные инструкции

Так как чтение и запись машинно-зависимых регистров производится с помощью специальных инструкций (RDMSR и WRMSR для архитектуры x86), то машинно-зависимые регистры и специальные инструкции будут рассмотрены вместе. Обращение к машинно-зависимым регистрам и выполнение специальных инструкций в QEMU транслируется в вызовы вспомогательных функций. Вспомогательные функции реализуют операции, выражение которых на языке внутреннего представления будет слишком громоздким и запутанным. Они написаны на языке Си и являются частью исходных кодов QEMU. Предлагается модифицировать функции, отвечающие за машинно-зависимые регистры и специальные инструкции таким образом, чтобы значения, полученные на первом проходе, сохранялись в журнал недетерминированных событий, а во время воспроизведения считывались из него.

5.3. Контроллер памяти и порты ввода\вывода

В QEMU за контроллер памяти и порты ввода\вывода отвечает одна и та же подсистема работы с памятью. В ней вся память рассматривается как набор областей. Области памяти бывают четырёх типов: контейнер, псевдоним, RAM и MMIO. Каждая область памяти имеет размер и поддерживает операцию адресации, семантика которой зависит от типа области. Рассмотрим подробнее каждый из типов.

- Область памяти типа контейнер включает в себя другие области по различным смещением. При адресации к такой области происходит переадресация к области, содержащейся по соответствующему адресу. Например, если область типа контейнер «C» содержит в себе область «A» по смешению «OFFSET», то при адресации к области «C», по адресу «ADDR», такому что «OFFSET» <= «ADDR» < «OFFSET» + «Длина A», произойдёт переадресация к области A по адресу «ADDR» - «OFFSET».
- Область типа псевдоним является ссылается на часть другой области по определённому смещению. Адресация для такой области работает следующим образом. Если область «A» является псевдонимом области «B» по смещению «OFFSET», то при адресации к области «A» по адресу «ADDR» произойдёт переадресация к области «B» по адресу «OFFSET» + «ADDR».
- Область памяти типа RAM ассоциирована с некоторым указателем из адресного пространства эмулятора QEMU. При адресации к области памяти типа RAM происходит разыменование указателя по соответствующему адресу.
- Область памяти типа MMIO ассоциирована с обработчиком доступа, который вызывается при адресации к этому типу области памяти.

Изначально эмулятор QEMU инициализирует только две области памяти типа контейнер: SYSTEM и IO. SYSTEM представляет собой область памяти, непосредственно доступную гостевой операционной системе при операциях работы с памятью, а IO - область памяти, доступную посредством инструкций доступа к портам ввода\вывода. Области памяти создаются, добавляются и удаляются динамически, по ходу работы гостевой операционной системы. Например, процесс добавления оперативной памяти к гостевой операционной системе выглядит следующим образом.

1. Создаётся новая RAM область памяти, размер области равен размеру оперативной памяти.
2. Выделяется память для оперативной памяти виртуальной машины.
3. Задаётся соответствие между новой областью и выделенной памятью.
4. Новая область добавляется в область SYSTEM по соответствующему смещению.

Теперь, при обращении гостевой операционной системы по адресам со «смещение» по «смещение + размер оперативной памяти» будет происходить обращение к ранее выделенной памяти.

Виртуальные устройства создают как RAM, так и IO области памяти, причём часть областей создаётся и добавляется при инициализации виртуального устройства эмулятором QEMU, а часть и при его инициализацией гостевой операционной системой. Поэтому необходимо сохранять информацию обо всех созданиях и добавлениях\удалениях областей памяти. Кроме того, в том случае, если RAM область была создана с заранее инициализированной памятью (например, это ROM-память видеокарты), необходимо сохранять данные, которыми она была инициализирована. В случае же IO областей, необходимо сохранять результаты всех операций.

Самым распространённым способом передачи большого объёма данных между устройством и операционной системой является прямой доступ к памяти. Поэтому виртуальные устройства QEMU имеют возможность писать в память гостевой операционной системы напрямую по указателю, никак не прерывая при этом работу гостевой операционной системы. Для сохранения данных, записанных напрямую, предлагается инструментировать каждое виртуальное устройство отдельно.

5.4. Прерывания

В эмуляторе QEMU аппаратные прерывания обрабатываются в основном цикле QEMU. При этом, хотя сам цикл и является аппаратно-независимым, код для обработки прерываний различается в зависимости от аппаратуры. За вызов соответствующих обработчиков прерываний отвечает переменная interrupt_request. Предлагается сохранять значение переменной interrupt_request перед условным оператором, определяющим наличие необработанного прерывания, в том случае, если её значение изменилось по сравнению с предыдущим. Кроме этого, необходимо также сохранять номер аппаратных прерываний. Для этого предлагается инструментировать код обработчика аппаратных прерываний, которой также находится в основном цикле QEMU.

5.5. Воспроизведение журнала

Для воспроизведения выполнения по журналу событий предлагается создать в QEMU собственную конфигурацию виртуальной машины RETRACER. В отличии от обычных конфигураций, она не создает периферийных устройств, зато воссоздает структуру памяти из журнала первого прохода. Чтобы гарантировать синхронность выполнения, размер блока трансляции будет установлен равным одной инструкции, а основной цикл будет инструментирован таким образом, что после каждой выполненной инструкции происходил вызов обработчика виртуальной машины RETRACER. Данный обработчик проверяет наличие асинхронных событий типа «создание области памяти», «прямая запись в память», «доставка прерывания» и производит необходимые действия. При такой реализации «асинхронные события», с точки зрения гостевой операционной системы будут происходить тогда же, когда и при первом проходе.

6. Результаты численных экспериментов

Тестировался симулятор QEMU, в котором детерминированное воспроизведение было реализовано первым способом, т.е. высокоуровневые события доставлялись в полную виртуальную машину.

Эксперименты проводились на персональном компьютере Intel® Core™ i5-2500 CPU @ 3.30GHz (4 CPUs), 8192MB RAM под управлением 64-разрядной ОС Windows 7 Enterprise.

В ходе экспериментов оценивались две наиболее важные характеристики детерминированного воспроизведения: накладные расходы, возникающие при снятии журнала и скорость роста журнала.

На Рис. 4 показаны накладные расходы, возникающие при записи журнала. Измерения проводились на четырех примерах, для каждого примера сравнивается время работы двух режимов работы симулятора. Первый режим (Базовая версия, светло серый цвет столбцов) предполагает, что доставка прерываний может происходить в произвольном месте, второй режим допускает доставку прерываний только на границах блоков трансляции (темно серый цвет столбцов). Сплошная заливка столбцов соответствует обычному выполнению примера, заливка с градиентом – снятию журнала.

Возникающее при трассировке замедление сильно варьируется как в первом, так и во втором режиме. Однако доставка прерываний только на границах стабильно улучшает скорость. Наибольшее замедление стабильно показывает трассировка ICQ, тогда как загрузка ОС происходит даже несколько быстрее, при включенном журнале. Объяснения для этого эффекта пока не найдены. В целом, замедление варьируется в интервалах от 10 до 110% для первого режима и от 0 до 90% для второго.

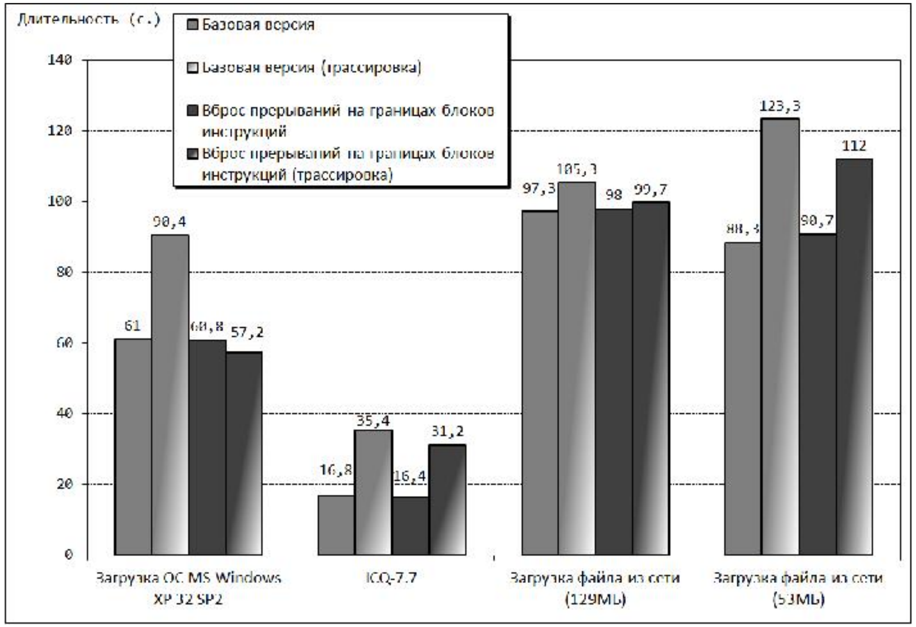

Рис. 4. Сравнение накладных расходов, возникающих при снятии журнала.

На рис. 5 и 6 показаны графики со скоростью роста журнала для гостевых систем Windows и Linux соответственно. Характер графиков одинаков, после начального этапа загрузки ОС, когда объем записываемых в журнал данных достаточно велик (в отдельные моменты времени достигает нескольких МБ в секунду), прирост размера журнала стабилизуется: и в том и в другом случае в секунду пишется порядка 50КБ данных. Большой объем данных в начальные моменты объясняются тем, что загружаемая ОС активно опрашивает шины в поисках доступного оборудования, а затем проводит инициализацию найденных устройств. Если для Windows выход на стабильную скорость роста происходит примерно через две минуты, то для Linux этот момент наступает несколько позже (шестая минута). Поскольку измерения делались для гостевых систем находящихся в состоянии покоя, пользователь не совершал никаких действий, соединение с сетью отключено, полученная скорость роста должна рассматриваться как нижняя граница. Дополнительный вклад будут вносить действия пользователя (клавиатура и мышь), а также входящий сетевой трафик. Если первым можно пренебречь, то второе неизбежно увеличит размер журнала. Однако достигнутая скорость для состояния покоя достаточно мала и сравнима с показателями аналогичных систем: 50КБ/с за сутки создают журнал размером около 4ГБ.

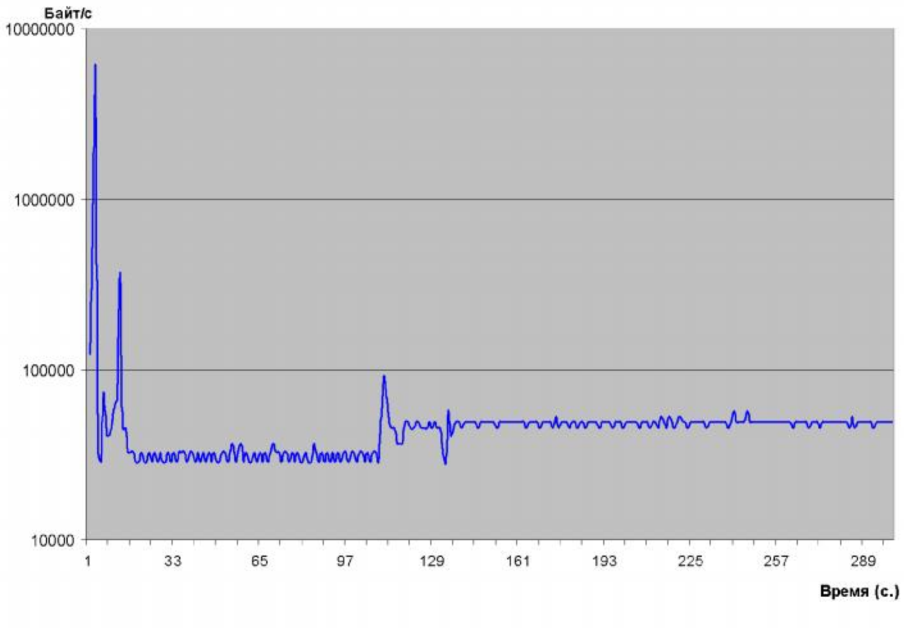

Рис. 5. Скорость роста журнала при загрузке Windows XP.

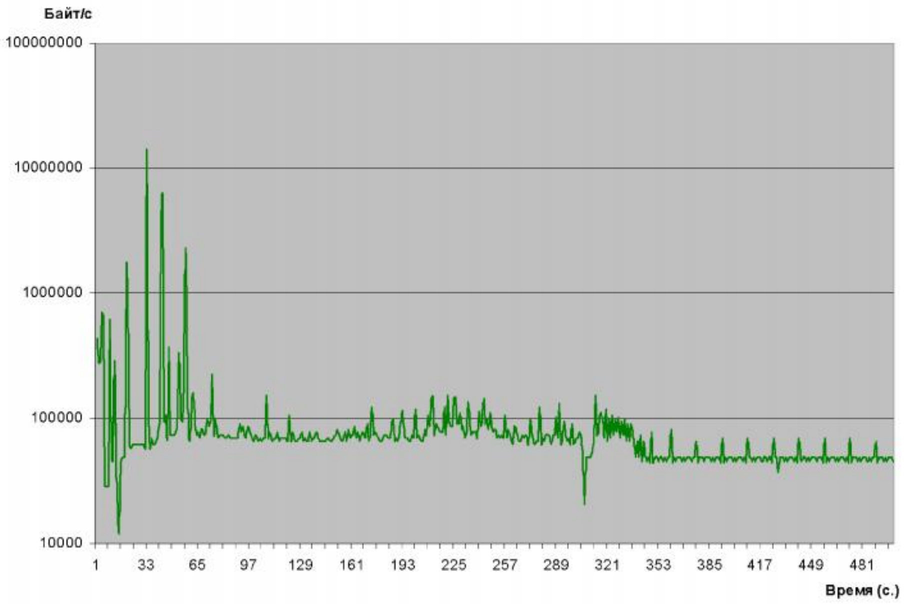

Рис. 6. Скорость роста журнала при загрузке Kubuntu v9.04.

7. Заключение

В работе был рассмотрена внутренняя организация программного симулятора QEMU и предложены два подхода к реализации в рамках данного симулятора механизма детерминированного воспроизведения.

Оба подхода обладают соответствующими достоинствами и недостатками. Накладные расходы у подхода, предполагающего доставку событий в полную виртуальную машину, показывают приемлемые значения. Оба подхода позволяют включать в состав конфигурации виртуальной машины устройства, использующие DMA.

Воспроизведение работы всей системы позволяет реализовать полносистемную отладку с возможность перемещения обратно во времени, а также интегрироваться в процесс анализа бинарного кода.

Список литературы

[1] VMware Player http://www.vmware.com/products/player/ дата обращения 2 апреля 2012

[2] Virtual Box https://www.virtualbox.org/ дата обращения 2 апреля 2012

[3] F. Bellard. QEMU, a fast and portable dynamic translator. // In USENIX 2005 Annual Technical Conf. pages 41–46, Apr. 2005.

[4] QEMU – Open Source Processor Emulator. http://wiki.qemu.org/Main_Page дата обращения 2 апреля 2012

[5] R. Bedicheck. SimNow: Fast platform simulation purely in software. In Hot Chips 16, Aug. 2004.

[6] Dunlap, George W. and King, Samuel T. and Cinar, Sukru and Basrai, Murtaza A. and Chen, Peter M. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. // ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, vol. 36, 2002, pp. 211-224.

[7] K. Buchacker, V. Sieh. Framework for Testing the Fault-Tolerance of Systems Including OS and Network Aspects. // Proc. of Sixth IEEE International Symposium on High Assurance Systems Engineering (HASE'01), 2001 pp.0095

[8] Min Xu, Vyacheslav Malyugin, Jeffrey Sheldon, Ganesh Venkitachalam, Boris Weissman. ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay. // Workshop on Modeling, Benchmarking and Simulation (MoBS), June 2007.

[9] Jim Chow, Tal Garfinkel, Peter M. Chen. Decoupling dynamic program analysis from execution in virtual environments. // Proceedings of the 2008 Annual USENIX Technical Conference, June 2008. pp. 1-14.

[10] Haikun Liu, Hai Jin, Xiaofei Liao, Zhengqiu Pan. XenLR: Xen-based Logging for Deterministic Replay. // In proc. of Japan-China Joint Workshop on Frontier of Computer Science and Technology (2008). pp. 149-154.

[11] Daniela A. S. de Oliveira, Jedidiah R. Crandall, Gary Wassermann, S. Felix Wu, Zhendong Su, and Frederic T.Chong. ExecRecorder: VM-based full-system replay for attack analysis and system recovery. // Proc. of the 1st workshop on Architectural and system support for improving software dependability (ASID '06), 2006. pp. 66-71

[12] Chia-Wei Hsu, Shiuhpyng Shieh. FREE: A Fine-grain Replaying Executions by Using Emulation. // The 20th Cryptology and Information Security Conference (CISC 2010), Taiwan, 2010.

[13] The Team for Research in Ubiquitous Secure Technology (TRUST). http://www.truststc.org/ дата обращения 2 апреля 2012

[14] Jiun-Hung Ding, Po-Chun Chang, Wei-Chung Hsu, Yeh-Ching Chung. PQEMU: A Parallel System Emulator Based on QEMU. // IEEE 17th International Conference on Parallel and Distributed Systems, 2011.

[15] Pavel Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. // Proc. of 16th European Conference on Software Maintenance and Reengineering, 2012.

Two approaches to full-system deterministic replay in QEMU

K. Batuzov, P. Dovgaluk, V.Koshelev, V. Padaryan
{batuzovk, Pavel.Dovgaluk, vedun, vartan}@ispras.ru

Abstract. In the paper we evaluate two approaches to full-system deterministic replay. Both of them allow replaying guest OS and applications without modifications. Deterministic replay is intended for debugging and dynamic analysis of system core, multithreaded and non-deterministic applications, cross-platform applications, and devices' drivers.

Presented approaches differ by boundary line dividing non-deterministic "outer world" and deterministic part of the simulator. All inputs from the "outer world" are written into the log to allow latter replaying of deterministic part. The common thing in both approaches is that deterministic part includes CPU, RAM, and video adapter. This allows debugging, tracing, and analyzing of the replayed code.

In the first approach "outer world" is presented by inputs – keyboard input, USB devices, mouse, network adapter, and microphone. All virtual peripheral devices should be deterministic in this approach. In the second approach all emulator parts except CPU, RAM, and video adapter are considered external. This means that all interactions between CPU and virtual peripheral devices (including IO, MMIO and DMA transactions) are written into the replay log.

The first approach has the following pros: one can replay whole virtual machine for devices' drivers debugging; relatively low number of changes within the simulator; low usage of storage for replay log. The drawback of this approach is the need to support for all external interfaces of the virtual machine. The second approach is completely opposite – it requires changes only in the several interfaces, but every virtual device with DMA interface should be modified. It also generates bigger replay logs.

We evaluated time and space overheads for the first approach. Slowdown is very low – it is lower than 1% in loading Windows XP scenario and is about 25% for network operations. Replay log growth speed for basic guest OS execution is about 50kb per second.

Keywords*:* QEMU, deterministic replay, full-system emulator

References

[1]. VMware Player http://www.vmware.com/products/player/
[2]. Virtual Box https://www.virtualbox.org/
[3]. F. Bellard. QEMU, a fast and portable dynamic translator. In USENIX 2005 Annual Technical Conf. pages 41–46, Apr. 2005.
[4]. QEMU – Open Source Processor Emulator. http://wiki.qemu.org/Main_Page

[5]. R. Bedicheck. SimNow: Fast platform simulation purely in software. In Hot Chips 16, Aug. 2004.
[6]. Dunlap, George W. and King, Samuel T. and Cinar, Sukru and Basrai, Murtaza A. and Chen, Peter M. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, vol. 36, 2002, pp. 211-224. doi: 10.1145/844128.844148
[7]. K. Buchacker, V. Sieh. Framework for Testing the Fault-Tolerance of Systems Including OS and Network Aspects. Proc. of Sixth IEEE International Symposium on High Assurance Systems Engineering (HASE'01), 2001 pp.0095
[8]. Min Xu, Vyacheslav Malyugin, Jeffrey Sheldon, Ganesh Venkitachalam, Boris Weissman. ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay. Workshop on Modeling, Benchmarking and Simulation (MoBS), June 2007.
[9]. Jim Chow, Tal Garfinkel, Peter M. Chen. Decoupling dynamic program analysis from execution in virtual environments. Proceedings of the 2008 Annual USENIX Technical Conference, June 2008. pp. 1-14. doi: 10.1109/FCST.2009.55
[10]. Haikun Liu, Hai Jin, Xiaofei Liao, Zhengqiu Pan. XenLR: Xen-based Logging for Deterministic Replay. In proc. of Japan-China Joint Workshop on Frontier of Computer Science and Technology (2008). pp. 149-154. doi: 10.1109/FCST.2008.31
[11]. Daniela A. S. de Oliveira, Jedidiah R. Crandall, Gary Wassermann, S. Felix Wu, Zhendong Su, and Frederic T.Chong. ExecRecorder: VM-based full-system replay for attack analysis and system recovery. Proc. of the 1st workshop on Architectural and system support for improving software dependability (ASID '06), 2006. pp. 66-71. doi: 10.1145/1181309.1181320
[12]. Chia-Wei Hsu, Shiuhpyng Shieh. FREE: A Fine-grain Replaying Executions by Using Emulation. The 20th Cryptology and Information Security Conference (CISC 2010), Taiwan, 2010.
[13]. The Team for Research in Ubiquitous Secure Technology (TRUST). http://www.truststc.org/
[14]. Jiun-Hung Ding, Po-Chun Chang, Wei-Chung Hsu, Yeh-Ching Chung. PQEMU: A Parallel System Emulator Based on QEMU. IEEE 17th International Conference on Parallel and Distributed Systems, 2011. doi: 10.1109/ICPADS.2011.102
[15]. Pavel Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. Proc. of 16th European Conference on Software Maintenance and Reengineering, 2012. doi: 10.1109/CSMR.2012.74