Об особенностях детерминированного воспроизведения при минимальном наборе устройств

В.Ю.Ефимов <real@ispras.ru>
К.А. Батузов <batuzovk@ispras.ru>
В.А.Падарян <vartan@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Технология (детерминированного) воспроизведения вычислительного процесса в виртуальных вычислительных машинах используется для отладки, повышения отказоустойчивости, а также в различных исследованиях программного кода, в том числе, в области информационной безопасности при обратной инженерия вредоносных программ. В данной работе описывается реализация технологии воспроизведения для гостевых машин на базе Intel Architecture 32-bit в программном эмуляторе QEMU, предлагающая минимизацию перечня воспроизводимых устройств. Подробно рассмотрено устройство эмулятора QEMU и обоснованы технические приемы, использованные при реализации. Приводятся экспериментальные оценки ключевых характеристик: объем записываемого журнала недетерминированных событий и замедление.

Ключевые слова: виртуальная машина; детерминированное воспроизведение; эмулятор; QEMU.

DOI: 10.15514/ISPRAS-2015-27(2)-5

Для цитирования: Ефимов В.Ю., Батузов К.А., Падарян В.А. Об особенностях детерминированного воспроизведения при минимальном наборе устройств. Труды ИСП РАН, том 27, вып. 2, 2015 г., стр. 65-92. DOI: 10.15514/ISPRAS-2015-27(2)-5.

1. Введение

В различных исследованиях машинного кода используется воспроизведение вычислительного процесса в виртуальной вычислительной машине (ВМ) (также известное как «детерминированное воспроизведение», «deterministic replay»). При воспроизведении гарантируется точное повторение последовательности состояний основных элементов ВМ и, как следствие, пути выполнения машинного кода. Имея такую возможность, можно выполнять

анализ уже на повторном выполнении, при этом факт анализа никак не повлияет на выполнение исследуемого кода: оно уже зафиксировано. Единственное оказываемое влияние — замедление от записи необходимой для воспроизведения информации при оригинальном выполнении, но для большинства известных реализаций технологии величина замедления не превышает десятков процентов. Такое замедление является пренебрежимо малым, особенно если сравнивать с замедлением от применения анализа непосредственно к оригинальному выполнению, которое составляет несколько порядков при трассировке или до нескольких секунд на инструкцию при отладке. Последнее делает невозможным успешное применение подобного анализа к коду чувствительному к задержкам (без технологии воспроизведения). Низкое замедление обусловлено малым объёмом данных, которые приходится записать, чтобы успешно воспроизвести первоначальный запуск. Большинство этих данных составляют взаимодействия ВМ с внешним миром.

Перечислим основные применения воспроизведения:

- реверсивная отладка позволяет проходить путь выполнения в обратном направлении: это даёт возможность искать место возникновения ошибки, начиная с места её проявления и двигаясь назад, а не использовать итеративный запуск программ с самого начала, как при классической отладке;
- устранение трудновоспроизводимых ошибок (в основном это ошибки, возникающие в параллельных программах: состояние гонок за данные, взаимоблокировки и т. д.);
- анализ компьютерных инцидентов (т. е. действий злоумышленника);
- анализ кода, чувствительного к замедлению (примерами такого кода могут служить сетевой сервер, сервер-приманка, защищенный от анализа вредоносный код, узел сети зараженных компьютеров: в последних трёх случаях модель нарушителя включает использование методов противодействия анализу, основанных на контроле замедления относительно внешнего мира).

Рассмотрим особенности воспроизведения вычислительного процесса, протекающего в ВМ на примере эмулятора QEMU [1] версии 2.1, работающего в режиме полносистемной эмуляции с использованием динамической двоичной трансляции. QEMU может эмулировать пользовательское окружение Linux или эмулировать систему с использованием аппаратной виртуализации, но в данной работе рассматривается исключительно вышеупомянутый режим. Фиксирование конкретной ВМ важно, так как наиболее интересные особенности воспроизведения скрыты именно в реализации ВМ, в то время как общая модель воспроизводимой системы включает в себя только высокоуровневые абстрактные правила. Воспроизвести вычислительный процесс — значит совершить ещё один, но таким образом, чтобы он был равен оригинальному

по некоторому заданному критерию (с заданными требованиями к точности). Точность воспроизведения определяется прикладными задачами. Например, иногда достаточно детерминированности выхода [2], что существенно ослабляет требования к методу. В данной же работе применяются следующие требования точности:

- совпадение последовательностей инструкций (коды операций, значения операндов, адреса инструкций в памяти);
- совпадение последовательностей состояний виртуального процессора и оперативного запоминающего устройства (ОЗУ) между инструкциями.

Заметим, что в состояние виртуального процессора также входят запросы прерывания (interrupt request — IRQ), а в состояние ОЗУ: таблица векторов прерываний, таблица виртуальной памяти и другие данные. Т.о., приведённые требования точности достаточны для выполнения анализа кода как на предмет наличия ошибок, так и на предмет уязвимостей и закладок [3].

При разработке технологии воспроизведения выделяют следующие понятия: детерминированная область, журнал недетерминированных событий, детерминированный таймер. Детерминированная область — это подсистема ВМ, поведение которой определяется взаимодействием с остальной частью ВМ. Т.е. если воспроизвести для детерминированной области все её взаимодействия с окружением, то последовательность её состояний также будет воспроизведена. Очевидно, что способ задания детерминированной области не единственен. В случае, когда ВМ — программный эмулятор, детерминированная область является частью данных и кода эмулятора, взаимодействующей с операционной системой (ОС), а также другими частями данных и кодом эмулятора. Если эмулятор работает параллельно, то следует учитывать как относительный порядок получения процессорного времени и взаимодействия между нитями внутри детерминированной области, так и обращения (относительный порядок и записываемые данные) внешних нитей к данным внутри области. Все взаимодействия с окружением (определяющие поведение детерминированной области) называются недетерминированными событиями. Они сохраняются в специальный журнал недетерминированных событий. Поскольку часть событий происходят асинхронно из внешней среды, их нужно повторять искусственно (т.к. внешняя среда недетерминированная и не гарантирует их повторение при каждом воспроизведении) и в правильные моменты относительно детерминированной области. Для последнего в детерминированной области выделяют детерминированный таймер, показывающий течение времени внутри области. В данной работе используется счётчик количества выполненных инструкций. В ряде работ [4, 5] предлагается использовать тройку: указатель инструкций (EIP), регистр-счетчик (ECX) и счётчик количества ветвлений (подразумевается IA-32).

Схематично процесс записи и воспроизведения показан на рис. 1. Он состоит из двух этапов: записи оригинального вычислительного процесса и его

воспроизведения. При записи все недетерминированные события сохраняются в журнал с соответствующими показаниями таймера. При воспроизведении все возникающие события скрываются от детерминированной области, но в соответствующие моменты искусственно подаются события из журнала.

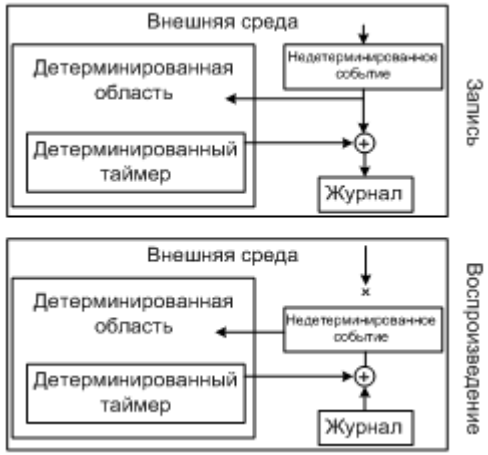

Рис. 1. Запись и воспроизведение.

Как уже отмечалось, детерминированная область не единственна, что создаёт возможность применения разных подходов [6]. В работе [7] был предложен и реализован альтернативный подход, отличающийся принципом выбора детерминированной области. А именно: в детерминированную область было включено большинство устройств ВМ. В предлагаемом подходе, наоборот, в область включены только ЦП и ОЗУ. Будем именовать подходы «Max ВМ» и «Min ВМ», исходя из принципов выбора детерминированной области. Сравнение подходов приведено в табл. 1.

Табл. 1. Сравнение подходов к обеспечению воспроизведения QEMU.

Характеристика	Max BM	Min BM
Перечень воспроизводимых состояний устройств	**Вся система**: процессор, ОЗУ, шины, контроллеры шин, таймеры, устройства ввода/вывода (жесткий диск, сетевой интерфейс, ...) и т. д. вплоть до взаимодействия с ресурсами основной ОС.	**Процессор**, **ОЗУ**, видео адаптер (как следствие воспроизведения выходных данных из процессора в его сторону).
Замедление	**Выше**. Требуется синхронизация работы всей системы. Задачи, выполняемые параллельно, нужно сериализовать.	**Ниже**. Наибольшая часть событий происходит из одной нити. Количество точек синхронизации нитей минимально.
Размер журнала	**Меньше**. Можно не писать некоторые события, исходя из их происхождения, если известно, что они детерминированы. Напр., образ жесткого диска.	**Больше**. Сложно отличить, какая информация пришла из детерминированных источников. Следовательно, нужно писать всё.
Наглядность событий	События непосредственно связаны с источником, напр., нажатие клавиши, движение мыши, приход сетевого пакета, срабатывание таймера и т. д. Наглядность **выше**.	У событий есть только адреса и разрозненные отрезки байт, которые уже прошли обработку в устройствах. Как следствие, наглядность **ниже**.
Сложность поддержки	Периферийные устройства в настоящее время меняются (добавляются) активно, что приводит к **более высокой** сложности поддержки.	Нужно только поддерживать изменения в интерфейсах взаимодействия с ближней периферией процессора, которые уже устоялись и почти не меняются, что приводит к **более низкой** сложности.

Характеристика	Max BM	Min BM
Сложность реализации	*QEMU версии до 0.13.0*. Сводится к перехвату необходимого минимума взаимодействий: их много. Сложность **средняя**. *QEMU версии от 1.0.1*. Взаимодействия с внешним миром разнесены в разные нити: дополнительно требуется обеспечить их синхронизацию. Сложность **выше**.	*Независимо от версии QEMU*. Необходимый минимум перехваливаемых событий невелик. Действия, выполняемые в других нитях, находятся за пределами области. Главная проблема — прямой доступ к памяти (DMA), выполняемый из другой нити (в случае версии от 1.0.1): требуется также организовать синхронизацию. Сложность **ниже**.
Расширяемость	Если добавляемое устройство взаимодействует с внешним миром (как чаще всего и есть), оно должно сохранять все свои взаимодействия с оным. Т. е. новое устройство — новый (часто уникальный) тип события. Расширяемость **сложнее**.	Процессор иногда находится через несколько интерфейсов до добавляемых устройств. Вплоть до того, что нет разницы, что именно находится с другого конца. Чаще всего добавление в систему нового устройства вообще не требует модификаций механизма воспроизведения. Расширяемость **легче**.

Объем недетерминированных событий в единицу времени не выходит за рамки пропускной способности жесткого диска компьютера. Высоконагруженные системы, обрабатывающие потоки сетевых данных, не развертываются на эмуляторах с двоичной трансляцией, а основными источниками недетерминизма в ВМ являются: загрузка данных с виртуального диска, пользовательский ввод и входящие сетевые пакеты. Таким образом, сброс на диск журнала перехваченных событий существенно не влияет на замедление ВМ. Помимо того, запись журнала можно буферизировать и выполнять в параллельной нити.

2. Эмулятор QEMU

В качестве ВМ для реализации воспроизведения был выбран QEMU по следующим причинам. Во-первых, необходима открытость исходного кода. Во-вторых, QEMU поддерживает наиболее распространённые архитектуры процессоров (IA-32, AMD64, ARM, MIPS, Power PC, др.). Это позволит позже расширить реализацию на другие архитектуры. В-третьих, QEMU переносим на уровне исходного кода: может быть скомпилирован для Linux и Windows.

Далее рассматриваются особенности работы QEMU, относящиеся к реализации воспроизведения. Эмулятор должен выполнять код, предназначенный для некоторой вычислительной машины (ВМ) (гостевой), опираясь на возможности основной машины (в которой выполняется сам эмулятор). Схема работы эмулятора и его связь с прообразом моделируемой ЭВМ изображены на рис. 2. Практика подразумевает использование упрощённой модели ЭВМ, с подробностью достаточной для выполнения машинного кода. В памяти эмулятора (т.е. памяти основной машины) создаются структуры, описывающие модели процессора, памяти и устройств гостевой машины, а также присутствует служебный код, определяющий взаимодействие между ними. Под выполнением гостевого кода понимается такая его обработка, при которой состояние гостевой машины (модели) изменяется таким образом, как изменилось бы состояние соответствующей реальной ВМ после выполнения этого кода. В QEMU этот процесс осуществляется с помощью динамической двоичной трансляции. Транслятор QEMU называется «Tiny Code Generator» (TCG). Двоичная трансляция происходит следующим образом. Сначала гостевой код транслируется в архитектурно независимое промежуточное представление, а затем оно компилируется в машинный код основной машины. Одной гостевой инструкции обычно соответствует несколько инструкций промежуточного представления. Также на уровне промежуточного представления в код вставляется служебная логика, необходимая для работы эмулятора. Служебная логика прозрачна для модели гостевой ВМ. Если эмуляция гостевой инструкции подразумевает сложную логику, то в эмуляторе реализуется специальная вспомогательная функция, а в промежуточное представление встраивается её вызов. Вспомогательные функции компилируются вместе с эмулятором. С точки зрения воспроизведения, среди вспомогательных функций важно отметить те, которые эмулируют инструкции времени, обращения к периферии и т.п. (рассматриваются ниже). Например, RDTSC из IA-32, возвращающая время. Трансляция кода выполнятся линейными участками (называемыми блоками трансляции) при первом обращении. Эмулятор буферизирует транслированный блок, чтобы не транслировать его каждый раз. Выполнение блока гостевого кода — это выполнение соответствующего транслированного кода.

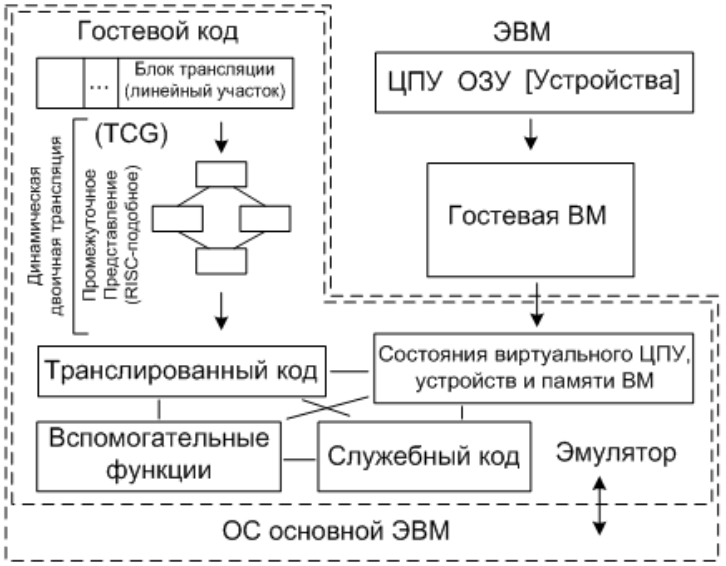

Рис. 2. Эмуляция ВМ в QEMU.

Выполнение в QEMU происходит в отдельной нити TCG в цикле эмуляции процессора. Итерация цикла изображена на рис. 3. Она состоит из выполнения гостевого кода, обработки исключений и запросов прерывания (IRQ), если последние имеются. Выполнение гостевого кода состоит из его получения (либо трансляцией, либо из буфера) и выполнения соответствующего транслированного кода. Если известно, что два блока трансляции всегда следуют один после другого (например, последняя гостевая инструкция в блоке есть переход по константному адресу или смещению), то они сцепляются. Сцепление осуществляется вставкой инструкции прямого перехода в транслированный код предыдущего блока на точку входа в транслированный код следующего блока.

Очевидно, что пока не получен IRQ, выполнение гостевого кода образует вложенный цикл в цикле эмуляции процессора (этот «горячий» путь выделен на рисунке толстой линией). Однако иногда эмулятору требуется по служебным нуждам приостановить эмуляцию. При этом для выхода из вложенного цикла используется специальный флаг — «CPU Exit ReQuest» (CPUERQ). Его опрос встречается дважды в итерации, т.к. получение транслированного кода может быть длительным (если происходит трансляция). Однако если из-за сцепления образовался длинный путь (или цикл) в транслированном коде, то CPUERQ не достаточно. В последнем случае применяется ещё один служебный флаг — «TCG Exit ReQuest» (TCGERQ). Проверка его установки встраивается в начало блока на уровне промежуточного представления при трансляции. Наконец, есть случаи, когда

необходимо прервать выполнение гостевого кода немедленно (не дожидаясь конца текущего блока). Например, для эмуляции исключений. Тогда выход происходит с помощью вызова стандартной функции siglongjmp.

Рассмотрим обработку IRQ, для выполнения которой эмулятор использует унифицированный механизм. В большинстве случаев IRQ соответствует моделируемому IRQ гостевой машины. Но также, например, может соответствовать запросу внешнего отладчика на остановку гостевого процессора. Если IRQ гостевое, то при его обработке, например, может быть изменено состояние контроллера прерывания, или могут быть совершены другие промежуточные действия. Т.е. происходит взаимодействие с устройствами. Признак наличия IRQ реализован в виде битового поля, где за каждым типом IRQ закреплён соответствующий бит-флаг. Кроме того, поведение обработчика прерывания является архитектурно зависимой частью итерации и, как следствие, его поведение может определяться другими переменными модели виртуального ЦПУ (или даже ВМ). В частности, гостевому прерыванию соответствует только один бит, а гостевой номер прерывания на этом этапе извлекается из контроллера прерывания.

Исключения, как и асинхронные прерывания, бывают гостевыми или служебными. Пример служебного исключения — запрос выхода из цикла эмуляции виртуального ЦПУ, например, для передачи управления другому процессору гостевой системы (многоядерные процессоры в QEMU моделируются несколькими виртуальными ЦПУ: по одному на ядро), для освобождения глобальной блокировки, для целей отладки гостя и т.д. Источниками гостевых исключений являются:

- синхронные исключения процессора (промах страницы, деление на ноль и т.д.);
- программные прерывания-ловушки (инструкция int N в архитектуре IA-32 и т.п.);
- в отдельных случаях обработка асинхронных прерываний гостевой ВМ завершается выдачей эмулятором служебного исключения, которое из-за особенностей его контекста следует рассматривать как гостевое.

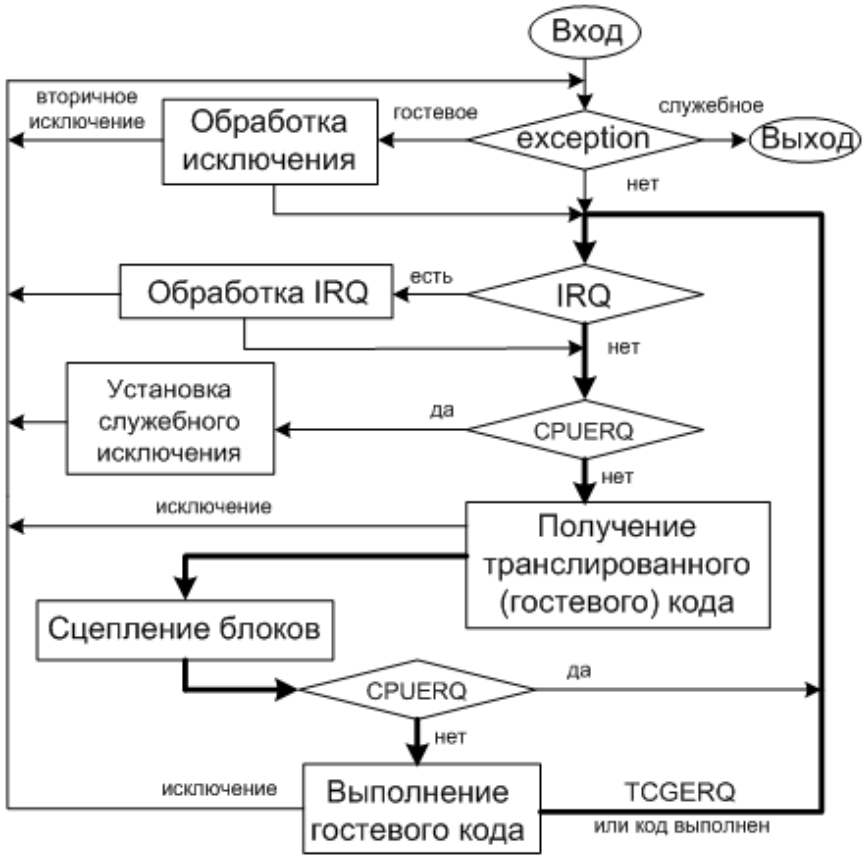

Рис. 3. Итерация цикла эмуляции виртуального ЦПУ.

Рассмотрим механизм обмена данными между виртуальным ЦПУ и устройствами. Он может происходить посредством отображения регистров устройств на физическое адресное пространство (Memory Mapped Input/Output — MMIO), с помощью DMA, а также через программируемый ввод/вывод (Port Mapped Input/Output — PMIO), предусмотренный в архитектуре IA-32. В случае MMIO процессор выполняет инструкцию чтения или запись из памяти по адресу, соответствующему устройству. В случае PMIO процессор выполняет инструкцию, указывая номер порта. В случае DMA процессор сначала задаёт команду устройству одним из вышеупомянутых способов, затем устройство самостоятельно копирует данные в память. Сначала рассмотрим работу MMIO и PMIO.

При моделировании физического адресного пространства используется интерфейс QEMU MMIO API (Application Programming Interface), ключевым

понятием которого является абстракция «регион памяти». Возможны следующие типы регионов памяти:

- ROM (Read Only Memory), RAM (Random Access Memory) — за регионом памяти закреплён буфер в памяти основной машины (например, ОЗУ, BIOS (Basic Input.Output System), видеобуфер, др.), все обращения к региону суть обращения к соответствующему буферу;
- IO (Input/Output) — для региона указаны функции-обработчики чтения и записи различных размеров, функции-обработчики вызываются при обращении к ним виртуального процессора: если выполняется запись в память, то функция принимает записываемое значение, если чтение, то она должна вернуть некоторое значение; кроме того, сам факт обращения может иметь значение для устройства, назначившего функцию-обработчик (в общем случае, логика работы может быть произвольной); примерами являются MMIO, PMIO, отслеживание самомодифицирующегося кода и др.;
- alias (псевдоним) — регион, перенаправляющий все обращения к себе на другой регион со смещением;
- контейнер — регион, содержащий в себе другие регионы (подрегионы) со смещением.

Таким образом, адресное пространство гостевой ВМ моделируется деревом регионов, где рёбро соответствует отношению «контейнер-подрегион». MMIO организуется с помощью регионов типа IO. Для своего регистра (или группы регистров) устройство назначает регион и функции-обработчики, и уже они определяют, как связаны соответствующие участки адресного пространства с состоянием и поведением устройства. PMIO реализовано аналогично MMIO: создано отдельное дерево регионов. Адресами в дереве являются номера портов. Взаимодействие гостевого кода с устройствами через MMIO и PMIO изображено на рис. 4.

Для гостевой инструкции, обращающейся к памяти или порту, генерируется вызов вспомогательной функции. Вспомогательная функция находит регион, соответствующий адресу (номеру порта), и вызывает для него унифицированный обработчик, который в свою очередь передаёт управление обработчику региона.

Важно отметить следующие особенности работы обработчиков:

- специальный обработчик может обратиться к ОЗУ гостя (например, контроллер прерываний APIC (Advanced Programmable Interrupt Controller) хранит в ОЗУ часть своего состояния и синхронизирует его при каждом обращении к своему регистру через MMIO);
- унифицированный обработчик может рекуррентно вызвать сам себя для фрагментации доступа (если произошло обращение длины

большей чем та, для которой устройство назначило обработчик, то обращение будет разбито на серию обращений подходящей длины).

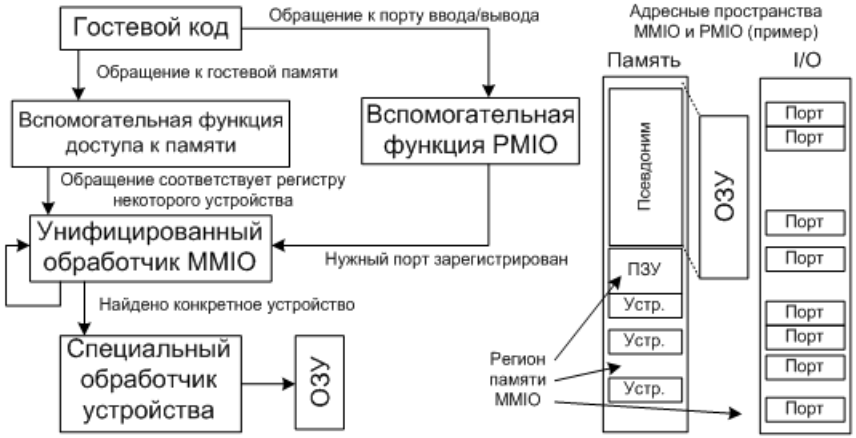

Рис. 4. Взаимодействие виртуального процессора с устройствами через MMIO и PMIO.

Таким образом, был освещен процесс выполнения гостевого кода и его взаимодействие с периферией. Однако для выполнения гостевого кода эмулятору также необходимо эмулировать работу устройств, которые, вообще говоря, работают независимо от ЦПУ. Наконец, эмулятор должен обеспечивать графический интерфейс пользователя и мониторы команд управления. На рис. 5 приведена схема организации QEMU в части распределения работы между нитями, показаны компоненты эмулируемой ВМ (выделены серой заливкой), служебные механизмы и данные. Обращения к ОЗУ и APIC происходят во всех нитях, поэтому на рисунке они вынесены за их пределы.

Нить TCG обеспечивает рассмотренное выше выполнение гостевого кода. Нить ввода-вывода выполняет в основном служебные задачи. А именно, она содержит главный цикл, который обрабатывает команды от пользователя как через графический интерфейс, так и через мониторы управления; обеспечивает интеграцию с ОС и т.д. Кроме того, главный цикл эмулирует работу таймеров: устройство может назначить обратный вызов на определённый момент времени относительно гостя. С точки зрения воспроизведения, важным в работе нити ввода-вывода является следующее:

- нить производит предварительную инициализацию ВМ, включая запись в ОЗУ;
- обратные вызовы таймеров, как правило, изменяют состояние IRQ.

В QEMU для параллельного обращения к файлам-образам ПЗУ ВМ (образа ROM, образа жестких дисков и др.) используются нити-рабочие. Это позволяет не блокировать другие нити на время выполнения обращения. Таким способом эмулируется DMA: нить-рабочий читает (пишет) данные из файла-образа в память основной машины, соответствующую памяти гостевой машины, в которую назначена транзакция DMA.

Описанные потоки данных обозначены на рис. 5 линиями со стрелками, соответствующими возможным направлениям копирования данных.

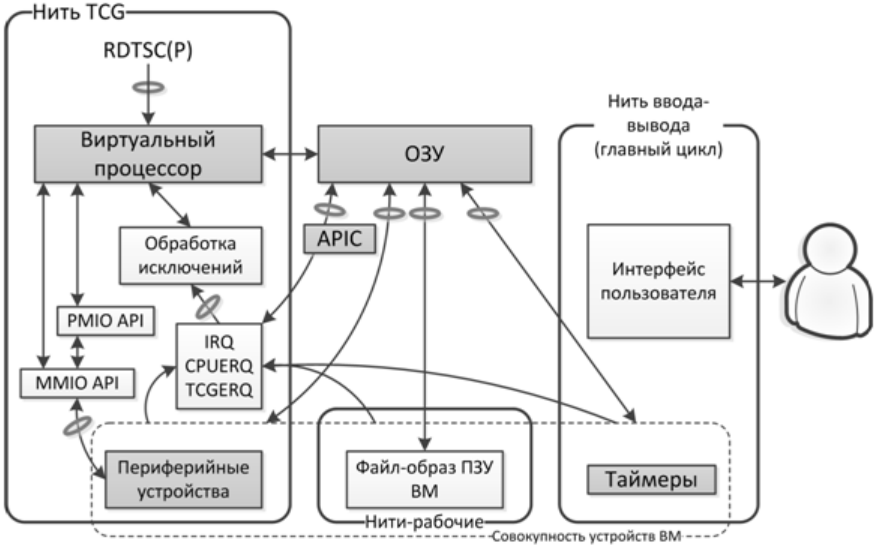

Рис. 5. Схема организации нитей выполнения в QEMU. Места перехвата и последующего вброса недетерминированных событий выделены овалами

3. Запись и воспроизведение

Перейдём к рассмотрению реализации предлагаемого подхода к записи и воспроизведению. Напомним, что одной из частей реализации воспроизведения является детерминированный таймер. В качестве детерминированного таймера используются счётчик выполненных виртуальным ЦПУ инструкций и счётчик недетерминированных событий.

Счётчик инструкций необходим для привязки ко времени IRQ и обращений к ОЗУ из нитей-рабочих и нити ввода-вывода. Подсчёт инструкций реализован на уровне промежуточного представления. В начало каждого блока трансляции вставляется инструкция, увеличивающая счётчик на размер блока в гостевых инструкциях. Кроме того, дополнительно учитываются случаи преждевременного выхода из блока через вызов функции siglongjmp (когда

количество действительно выполненных инструкций меньше, чем всего инструкций в блоке). К таким случаям относятся:

- инструкция, вызвавшая исключение или программное прерывание;
- обработчик MMIO/PMIO, произведший немедленный выход из цикла выполнения гостевого кода по какой-либо причине.

Когда инструкция вызывает исключение, то считается, что она не выполнена. Т.е. предполагается, что гость устранит причину исключения, выполнив другой код, и в следующий раз выполнение этой инструкции не вызовет исключение. Инструкции программного прерывания, напротив, следует считать выполненными. Подобное также справедливо для некоторых особенных инструкций, прерывающих поток управления. Для IA-32 это: `HLT` (останавливающая процессор до следующего прерывания), `INT` и др. Компонента TCG, генерирующая из гостевого кода промежуточное представление, обеспечивает, чтобы подобные инструкции были последними в блоке, как и инструкции перехода. Таким образом, достаточно вычесть из счётчика единицу (или не вычитать: в зависимости от типа инструкции). Кроме того, обращения к MMIO/PMIO могут прерывать выполнение блока в произвольном месте. Это нужно в случае, если после выполнения обращения выполнение последующего кода будет некорректным. Например, через MMIO API осуществляется контроль целостности гостевого кода для поддержки самомодифицирующегося кода. Если код изменил страницу, в которой находится сам, то нельзя считать, что последующие инструкции транслированного кода соответствуют текущему содержимому памяти (в QEMU применяется постраничная гранулярность).В таких случаях используется процедура восстановления регистра счётчика команд, изображенная на рис. 6. Блоки трансляции обозначены на рисунке «БТ», а вспомогательные функции – «helper».

Проблема преждевременного выхода из блока обусловлена оптимизациями, реализованными в QEMU, а именно ленивым вычислением флагов и регистров гостевого процессора. Т.е. они вычисляются только тогда, когда их значения необходимы. В частности, PC (program counter) не устанавливается после выполнения каждой инструкции на адрес следующей. Это не нужно, т.к. инструкции разбиты на блоки и известно, какая будет следующей в пределах блока трансляции. Необходимо только знать, какой блок следует выполнять следующим если они не сцеплены. Т.е. PC имеет корректное значение только между блоками трансляции или при явном чтении его значения. Если произошел преждевременный выход из блока, то PC указывает на его начало, а не на инструкцию, вызвавшую выход. В то же время нужно продолжить выполнение с этой инструкции. Следовательно, нужно узнать её гостевой адрес.

Восстановление правильного гостевого PC происходит следующим образом. Известно, что каждой инструкции гостевого кода соответствует диапазон инструкций кода основной машины. Преждевременный выход происходит из

вспомогательной функции (helper), адрес возврата (в памяти основной машины) из которой известен (TCG передаёт одним из параметров вспомогательной функции адрес возврата из неё). Т. о. можно определить, к какому диапазону он относится. При определении PC блок транслируется повторно, но сопровождается информацией об адресах, порядковых номерах в блоке и отображении гостевых инструкций на диапазоны памяти основной машины. PC определяется по этой информации. Поскольку в ходе восстановления PC известно количество реально выполненных инструкций, нетрудно скорректировать счетчик.

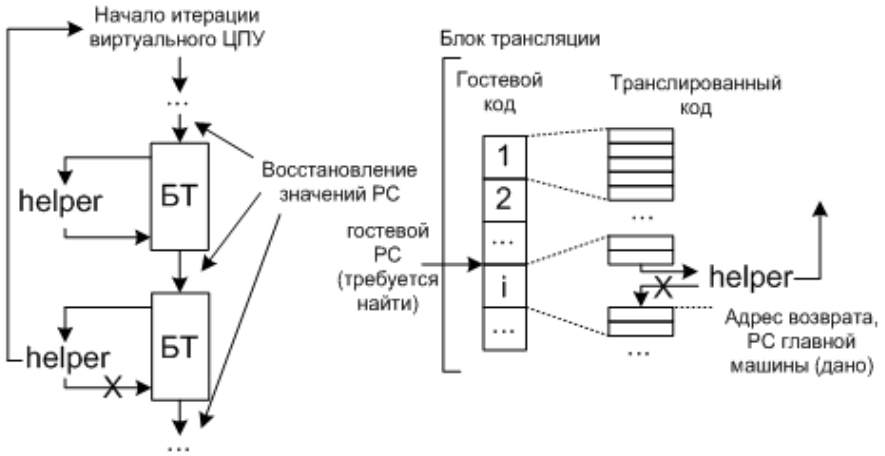

Рис. 6. Восстановление регистра счётчика команд (PC).

Восстановление PC происходит непосредственно перед выходом из гостевого кода. Но после восстановления не обязательно произойдёт выход (например, в случае APIC). Т.е. будет выполнено восстановление, но блок трансляции всё равно будет выполнен до конца. Очевидно, что в этом случае корректировать счётчик инструкций не нужно. Для обработки подобных случаев в состоянии виртуального ЦПУ было добавлено специальное поле, хранящее результат последнего восстановления. Он применяется, если только действительно произошел преждевременный выход. Однако, если преждевременный выход после восстановления всё-таки произошел, но из другой инструкции, которая не предусматривает восстановление PC (например, исключение или программное прерывание, которые явно знают адрес следующей инструкции посредством обращения к TCG), то содержимое этого специального поля будет некорректно применено. Следовательно, нужно обнулять это специальное поле после выполнения специального обработчика (если он вернул управление в универсальный обработчик, то он не совершил преждевременного выхода из блока, а значит, результат восстановления точно не пригодится).

В итоге счётчик инструкций имеет корректное значение между блоками. Этого достаточно, так как:

- IRQ обрабатываются между блоками трансляции;
- параллельные обращения к ОЗУ из других нитей дополнительно блокируются на время выполнения гостевого кода, и, следовательно, выполняются между блоками трансляции.

Однако внутри блока счётчик инструкций имеет некорректное значение. В то же время, доставка некоторых событий (в частности MMIO и PMIO) требует привязки к конкретной инструкции. Для решения этой проблемы используется счётчик событий. Каждое обращение к MMIO вызывает увеличение счётчика, даже если событие детерминировано (например, запись из виртуального процессора в устройство) и не требует записи в журнал. Это порождает в журнале *виртуальные* события, требующие особой обработки при воспроизведении. Подробнее об обработке MMIO будет указано ниже.

Обратим внимание, что в ранних версиях использовался подсчёт инструкций поштучно, т.е. перед каждой инструкцией вставлялся инкремент счётчика. Подобное решение вносило замедление 4 — 11% (в зависимости от характера выполняемого кода). Текущий подход вносит замедление 2 — 6%, соответственно.

Далее рассматриваются места и способы перехвата недетерминированных событий. На рис. 5 эти места обозначены овалами вокруг стрелок. Очевидно, что в соответствии с подходом в детерминированную область входят виртуальный процессор и ОЗУ, но не APIC.

3.1. Инструкции RDTSC и RDTSCP

Инструкция `RDTSC` (и её вариант `RDTSCP`) возвращает гостевой показатель времени виртуального ЦПУ. Возвращаемое значение зависит от времени основной машины, поэтому является недетерминированным. Хотя есть режим работы QEMU «icount», при котором течение виртуального времени выравнивается по числу выполненных инструкций; но даже в этом режиме время корректируется по реальному времени в случае простоя процессора, когда инструкции не выполняются (например, если процессор ожидает прерывание). Инструкция `RDTSC` реализована вспомогательной функцией. При записи возвращаемое значение инкапсулируется в событие и записывается в журнал, а при воспроизведении берётся из него и подменяет текущее.

3.2. MMIO, PMIO и синхронный доступ к ОЗУ

Чтение данных виртуальным процессором из устройств через MMIO и PMIO эмулируется вызовом функции, возвращаемое значение которой — прочитанные данные. Эти данные являются недетерминированными и сохраняются в журнал при записи. При воспроизведении ими подменяются текущие данные. Очевидно, что данные, записываемые процессором в

устройство, во-первых, являются детерминированными, во-вторых, не влияют на поведение детерминированной области: их записывать не нужно. При отладке, тем не менее, они записывались и использовались для проверки корректности воспроизведения.

Основной проблемой является недетерминированность специального обработчика и потенциальная неограниченность его возможных действий. Известно следующее поведение (важное для воспроизведения):

- рекуррентное обращение к MMIO/PMIO,
- обращение к ОЗУ,
- модификация исполняемого в данный момент гостевого кода (особый случай обращения к ОЗУ).

Как параметры, так и сам факт выполнения вышеперечисленных действий являются недетерминированными.

В случае рекуррентного обращения важными являются только данные, возвращенные корневым вызовом. Промежуточные данные возвращаются специальному обработчику, а не виртуальному ЦПУ, поэтому их записывать не нужно. Для их отсева отслеживается откуда обработчик был вызван.

Обращение к ОЗУ из устройства является недетерминированным событием и должно быть записано. Очевидно, что нужно воспроизвести данные, которые записываются в память. Кроме того, факт обращения зависит от недетерминированного состояния устройства. Т.е. при воспроизведении нужно искусственно сделать вызов, если он не был сделан устройством.

Особым случаем обращения к ОЗУ является модификация текущего кода. Единственным известным устройством, выполняющим это, является APIC. Последовательность действий его специального обработчика следующая:

- восстановление PC (чтобы потом продолжить выполнение с того же адреса);
- изменение гостевого кода;
- перетрансляция изменённого кода;
- выход из текущего блока, т.к. он больше не соответствует гостевому коду в этом месте ОЗУ.

Восстановление PC нужно сделать именно до модификации кода, т.к. алгоритм восстановления (среди прочего) требует неизменность кода. Это справедливо и при воспроизведении. Изменение гостевого кода здесь — запись в ОЗУ, которая обрабатывается так же, как и обычное обращение к ОЗУ. В частности, APIC изменяет код инструкции, вызвавшей обращение к нему: вместо неё вставляется вызов функции в гостевой ОЗУ с параметром равным тому, который был передан специальному обработчику APIC. Вышеописанные действия нужно воспроизводить в таком же порядке относительно друг друга. При этом для воспроизведения восстановления PC и перетрансляции с выходом из блока потребовалось ввести специальный тип событий (изменение гостевого кода описывается событиями записи в ОЗУ).

Для обработки событий доступа к ОЗУ, MMIO и PMIO нужно задать:

- порядок среди соответствующих инструкций (внутри блока трансляции),
- порядок в дереве вызовов обработчика каждой конкретной инструкции.

При этом значение счетчика выполненных инструкций внутри блока некорректно, и не может быть использовано. Для решения данной проблемы реализовано следующее:

- подсчёт всех инструкций MMIO/PMIO как событий (вторая составляющая детерминированного таймера, описанная выше), для определения нужной (по порядку) инструкции в блоке трансляции;
- сохранение стека вызовов (путь в дереве вызова обработчика инструкции MMIO/PMIO) для соответствующих событий, чтобы их вставить искусственно, если они были пропущены специальным обработчиком.

Ввиду недетерминированности устройств, при воспроизведении они могут находиться в состоянии, при котором обращение к ним в неподходящий момент или с неудачными данными может привести к отказу эмулятора. Но, между тем, есть и устройства, обращаться к которым можно/нужно. Например, для удобства пользователя желательно воспроизводить состояние графического буфера видеоадаптера. Решение проблемы — при воспроизведении выборочно разрешать обращения к конкретным устройствам.

Помимо того, возможности MMIO API шире, чем эмуляция регистров устройств и портов ввода/вывода. В частности, эмулятор использует его для поддержки самомодифицирующегося кода, простановки точек останова по обращению к данным (watchpoint) и др. Т.е. в адресное пространство добавляются регионы памяти со специальными функциями-обработчиками. Наличие и поведение таких регионов MMIO, во-первых, не влияет на состояние гостевой ВМ, во-вторых, недетерминировано. Т.о. эти события не нужно учитывать, поскольку это может сбить счетчик событий внутри блока, что приведёт к доставке событий MMIO в неправильные обработчики. Для решения последних проблем структуры данных, описывающие регионы памяти, были снабжены полями, регулирующими обработку перехваченных обращений к MMIO API:

- учитывать ли как событие;
- (если учитывать) запрещать ли выполнение специального обработчика чтения/записи при воспроизведении.

Кроме обработчиков MMIO, синхронно к ОЗУ может обращаться APIC в момент получения номера прерывания при обработке IRQ (в теле цикла эмуляции виртуального процессора). Присутствие данного события является

недетерминированным, т.е. не каждое получение номера прерывания сопровождается обращением к ОЗУ. Ввиду аналогии с обращением к ОЗУ из MMIO, было принято решение использовать тот же механизм обработки: место вызова функции, возвращающей номер прерывания, окружено кодом, эмулирующим для последующих вызовов то, что они находятся в дереве вызовов MMIO. Т.о. требуемое обращение к ОЗУ будет записано как обращение к ОЗУ из обработчика MMIO. При воспроизведении такие события выравниваются по счётчику событий как и обычный доступ к ОЗУ из MMIO. Сам же факт получения номера прерывания воспроизводится как запрос прерывания.

3.3. Запросы прерывания

IRQ могут выставляться устройствами в произвольные моменты времени из любой нити. Однако для воспроизведения работы процессора важен момент обработки IRQ относительно инструкций и его недетерминированные данные. Т.о. при записи события сохраняются:

- номер инструкции,
- битовое поле, кодирующее IRQ, и дополнительные архитектурно-зависимые параметры.

При воспроизведении все возникающие IRQ игнорируются, а записанные IRQ встраиваются искусственно. Особенность заключается в том, что разбиение гостевого кода на блоки трансляции является недетерминированным: например, точка остановки, заданная пользователем в отладчике, разобьет блок трансляции на две части. Т.е. нельзя гарантировать, что при записи и каждом воспроизведении место вставки IRQ будет само собой попадать на разрыв между блоками. Поэтому задача вставки IRQ между конкретными инструкциями при воспроизведении требует внесения следующих трех изменений в алгоритм обработки гостевого кода.

- Сцепления блоков не происходит. Сцепление нужно для того, чтобы не возвращать управление эмулятору лишний раз. При записи, когда возникает IRQ, асинхронно (из другой нити) устанавливается флаг TCGERQ, что приводит к возврату управления перед выполнением очередного блока. При воспроизведении этот способ не приемлем, т.к. нельзя гарантировать точную доставку TCGERQ. Отключение сцепления позволяет получать управление после каждого блока и вставлять IRQ. Это замедляет воспроизведение.
- Длина (в гостевых инструкциях) блока трансляции ограничивается таким образом, чтобы его последняя инструкция была не позже, чем следующий записанный IRQ.
- Код повторно транслируется, если длина блока достаточно велика, и его последняя инструкция будет выполняться позже следующего IRQ. Этот случай возможен, т. к. при трансляции блока количество

инструкций до ближайшего IRQ могло быть больше, чем при последующих его выполнениях. Фактически происходит уничтожение блока и генерация нового по приведенному выше (второму) правилу.

Описанные изменения также необходимы и для воспроизведения событий асинхронного обращения к ОЗУ. При реализации этих изменений события асинхронного доступа к ОЗУ становятся частью потока событий IRQ.

3.4. Асинхронный доступ к ОЗУ

Под асинхронным доступом к ОЗУ здесь понимается любой доступ к ОЗУ, который происходит не из потока управления нити TCG. Примерами асинхронного доступа к ОЗУ являются:

- инициализация памяти из нити ввода-вывода при начальной загрузке ВМ;
- изменение состояния APIC (его часть, которая хранится в ОЗУ) при изменении статуса IRQ устройством по таймеру (из нити ввода-вывода) или другой причине;
- выполнение DMA нитью-рабочим и др.

Основная проблема асинхронного доступа к ОЗУ — это состояние гонок между нитью TCG и другими нитями, влияющее на путь выполнения гостевого кода. Решение о выделении процессорного времени нити принимает ОС, а эмулятор может на это повлиять посредством примитивов синхронизации. В текущей реализации записи ОЗУ защищена двоичным семафором. Он принадлежит нити TCG, пока выполняется гостевой код. Между блоками семафор освобождается и может быть захвачен другой нитью. При записи обращение сохраняется с привязкой к счётчику выполненных инструкций (при выполнении нескольких обращений порядок между ними задаётся по счётчику событий). Напомним, что между блоками трансляции счетчик инструкций содержит корректное значение.

При воспроизведении обращение к ОЗУ из других нитей не выполняется. Все записанные обращения встраиваются искусственно и синхронно (из нити TCG). Особенности вставки этих событий между конкретными инструкциями такие же, как и при доставке IRQ, и описаны выше.

Особый случай асинхронного доступа к ОЗУ представляет эмуляции DMA из нитей-рабочих. Основная проблема в том, что копирование происходит между жестким диском и ОЗУ. Обращение к диску сравнительно длительная операция, и захват семафора на время её выполнения вызвал бы неоправданное замедление при записи. Поэтому соответствующий код был модифицирован так, чтобы при записи сначала параллельно с нитью TCG производилось копирование данных между диском и специально выделенным

буфером в ОЗУ, а затем захватывался семафор, и производилось копирование между буфером и первоначальным местом в ОЗУ.

3.5. Запись журнала

Поток событий обладает следующими свойствами:

- количество байт журнала производимых за единицу времени может отличаться на несколько порядков в разные моменты времени и может превысить пропускную способность распространённых жестких дисков;
- усреднённое по времени количество байт журнала производимых за единицу времени не превышает пропускную способность жесткого диска (за исключением случая, когда ВМ активно работает с виртуальным диском, чей образ располагается на диске основной машины);
- количество событий на единицу времени может отличаться на порядки;
- основным источником событий является нить TCG, поэтому низкая скорость записи журнала вызовет замедление виртуального ЦПУ.

С учётом этих особенностей запись журнала была организована по схеме, представленной на рис. 7.

Во-первых, события буферизуются, что позволяет сгладить замедление при большом количестве событий в единицу времени, однако не решает проблему замедления от копирования данных на диск при заполнении буфера. Но количество системных вызовов уменьшается на порядки.

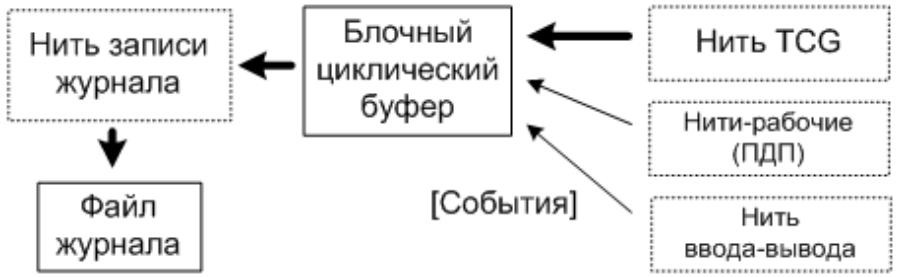

Рис. 7. Схема записи журнала.

Во-вторых, для записи журнала на диск создаётся специальная нить, работающая по схеме поставщик-потребитель. В памяти присутствует несколько буферов. Они упорядочены циклически: когда один заполняется, для записи выбирается следующий. Когда нить записи журнала замечает очередной заполненный буфер, она записывает его на диск. Такая схема позволяет исключить замедление записи журнала от относительно низкой пропускной способности диска. Т.е. при условии, что размер буфера

достаточно велик, замедление гостевой системы от записи будет определяться лишь пропускной способностью ОЗУ. Практически, всегда можно назначить буфер достаточного размера: для большинства гостевых ОС достаточно 100МБ.

4. Результаты

Для оценки ключевых свойств реализаций детерминированного воспроизведения были проведены численные эксперименты. Конфигурация основной машины следующая: процессор Intel Core i7-3770, 8ГБ ОЗУ, ОС Linux Ubuntu 14.04 64-х битная версия. Для сравнения реализаций оценивались:

- загрузка ОС Microsoft Windows XP, Service Pack 3;
- загрузка ОС Microsoft Windows 7, Service Pack 1;
- результаты бенчмарка nbench [8] на базе ОС GNU/Linux, версия ядра 3.12;
- результаты бенчмарка iperf [9] и утилиты ping на базе ОС Microsoft Windows XP, Service Pack 3.

В табл. 2 представлены результаты сравнения свойств реализаций подходов на загрузке гостевых ОС. Напомним, что основными характеристиками реализации, является относительное замедление записи и размер журнала. С точки зрения практического применения, реализация не должна сильно замедлять вычислительный процесс и при воспроизведении: хотя замедление при воспроизведении не скажется на точности, оно может сказаться на времени выполнения последующего анализа. Измерение времени загрузки Windows производилось 5 раз, в таблице приведены средние арифметические значения.

Табл. 2. Экспериментальное сравнение свойств реализаций подходов на загрузке гостевых ОС.

Тест	QEMU 2.1.0	Max BM	Min BM
ОС Windows XP, SP 3			
Время загрузки	23,6 сек	37,0 сек (+57%)	33,6 сек (+42%)
Размер журнала	-	11 МБ	476 МБ
Время воспроизведения	-	~96 сек замедление ~4 раза	~88 сек замедление ~3,76 раза
ОС Windows 7, SP 1			
Время загрузки	105,6 сек	189,3 сек (+79%)	150,0 сек (+42%)
Размер журнала	-	155 МБ	1036 МБ
Время воспроизведения	-	~17 мин замедление ~9,66 раза	~15 мин замедление ~ 8,52 раза

В табл. 3 представлена оценка замедления различных подходов с использованием утилиты nbench. Запуск nbench производился 10 раз и усреднялся. Приведённые значения индексов есть отношение количества выполненных операций в единицу времени к некоторому эталонному значению, соответствующему скорости работы определённого реального процессора. Т.е. чем больше индекс, тем быстрее работает ВМ. Поскольку вырабатываемый бенчмарком результат – индекс производительности, его измерение осмыслено только на первом проходе, когда пишется журнал событий, а измерение течении времени выполнятся по реальным часам.

Табл. 3. Экспериментальное сравнение свойств реализаций подходов с использованием nbench.

Тест	QEMU 2.1.0	Max BM	Min BM
nbench на базе ОС Linux, ядро 3.12			
Memory Index	3,553	1,003 (-72%)	3,208 (-9%)
Integer Index	3,881	1,040 (-73%)	2,289 (-41%)
Float-point Index	1,735	1,441 (-17%)	1,635 (-6%)
Размер журнала	-	26 МБ	28 МБ

Результаты, приведенные в таблицах 2 и 3, позволяют сделать следующие выводы. Во-первых, наблюдается значительная разница размера журнала для загрузки ОС. Большой размер журнала подхода «Min BM» обусловлен записью всех данных, считанных с диска. Напротив, реализация подхода «Max BM» считает образ диска детерминированным и не записывает считанные из него данные. С другой стороны, для воспроизведения по предлагаемому подходу достаточно только журнала, в то время как для воспроизведения по подходу «Max BM» требуется неизменный образ диска. Это особенно важно в случае распределённой работы с журналом, т.к. обычно размер образа диска значительно больше журнала (даже для подхода «Min BM»). Для случая записи приложения (nbench) размеры журналов почти не отличается. Имеющуюся разницу теоретически можно объяснить записью в журнал загрузки приложения с диска (в случае подхода «Min BM»).

Во-вторых, по результатам nbench можно сказать, что замедление зависит от характера выполняемых инструкций. Сравнительно низкая разница замедления инструкций математического сопроцессора (float-point index) обусловлена тем, что в QEMU инструкции сопроцессора эмулируются вспомогательными функциями. Последнее приводит к тому, что на одну инструкцию сопроцессора приходится значительно больше инструкций процессора основной машины, чем на одну инструкцию арифметико-логического устройства процессора. Т.е. замедление непосредственно от записи журнала при активном использовании сопроцессора составляет меньшую долю от общего времени.

В-третьих, в целом предлагаемый подход вносит меньшее замедление. Замедление в подходе «Max BM» может быть обусловлено сериализацией всех нитей QEMU для обеспечения корректной записи.

В-четвёртых, время воспроизведения в реализации предлагаемого подхода, как и в другом подходе, значительно больше времени записи. В подходе «Min BM» основными причинами этого являются:

- отсутствие сцепления блоков и их дополнительная перетрансляция (ради своевременной доставки IRQ);
- при чтении журнала не используются те же оптимизации, как при записи (распараллеливание, буферизация).

На рис. 8 приведён график роста журнала для ОС Windows XP SP3. Ниже приводятся действия, которые производились с целью выявления влияния на них записи и выявления их влияния на скорость роста журнала. Номерами на рис. 8 обозначены соответствующие этапы.

1. *Появление рабочего стола*. Этот момент соответствует окончанию измерения времени загрузки из табл. 2. ОС дано некоторое время на завершение загрузки. Во время загрузки и некоторое время после появления интерфейса пользователя наблюдается хаотичность скорости роста журнала. Это связано с загрузкой драйверов и опросом соответствующих устройств, загрузкой графических элементов пользовательского интерфейса, автозапуском приложений и др. В самом начале загрузки скорость достигает ~100 МБайт/сек, что можно объяснить загрузкой основных компонентов ОС.
2. *Активные движения указателем мыши и ввод текста в текстовой редактор*. Очевидно, что действия пользователя вносят вклад в журнал, т.к. являются недетерминированными. Кроме того, в течение данного этапа был запущен текстовой редактор, т.е. происходило чтение данных с диска.
3. *Работа без внешних воздействий.* Скорость роста журнала около 60 Кбайт/сек.
4. *Запуск утилиты iperf* [9] в режиме клиента для тестирования пропускной способности соединения с основной машиной через виртуальное сетевое окружение, реализуемое QEMU (SLIRP). На основной машине iperf была запущена в режиме сервера. Тестирование выполнялось с использованием протокола TCP. Достигнутая пропускная способность соединения: 41,3 Мбит/сек (при аналогичных измерениях, но без записи журнала — 54,2 Мбит/сек: замедление ~24%). Измерение проводилось 5 раз, приведены средние арифметические значения. Обмен информацией по сети также вызвал рост журнала с соответствующей скоростью.
5. *Запущено штатное завершение работы ОС*.

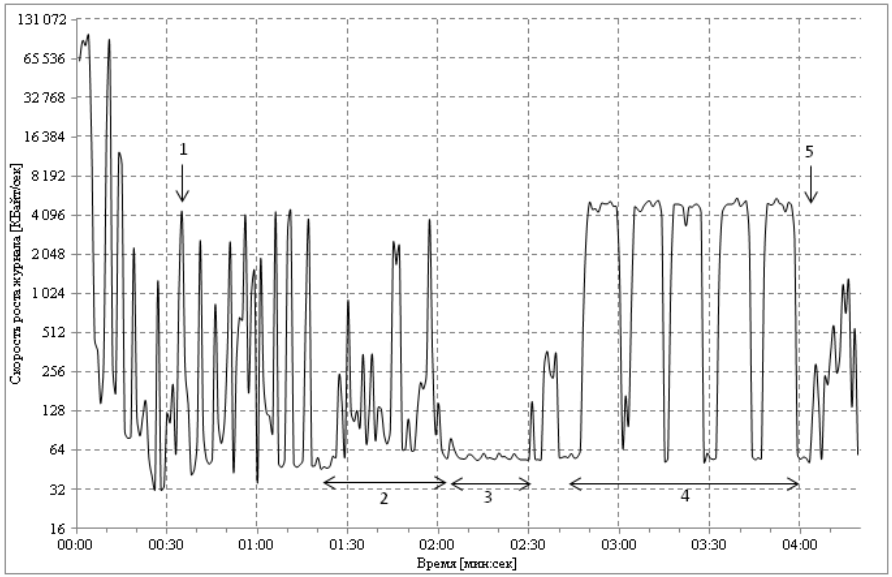

Рис. 8. Скорость роста журнала для ОС Windows XP SP3

Отдельно измерялось влияние записи на время отклика по сети с использованием утилиты ping. Для измерения было запущено две ВМ, сопряженные в одну сеть, и время отклика измерялось между ними (вторая ВМ нужна, т.к. SLIRP не поддерживает протокол ICMP). Запись производилась только на той ВМ, где была запущена ping. Как при записи, так и без неё первые 2-4 ответа приходили с задержкой менее 5мс, а последующие пакеты приходили с задержкой не более 1 мс. Т.о. запись не вызывает заметного увеличения времени отклика. Сравнительно высокая задержка первых пакетов может быть объяснена трансляцией соответствующего гостевого кода.

Все записанные действия были воспроизведены, отклонений выявлено не было. Время воспроизведения вычислительного процесса, соответствующего рис. 8, составило 8 мин 56 сек (+106%).

5. Заключение и дальнейшая работа

Таким образом, существуют два подхода к реализации воспроизведения ВМ QEMU. В работе был исследован подход «Min ВМ» к воспроизведению гостевой среды QEMU, основывающийся на минимизации детерминированной области. Было проведено теоретическое сравнение данного подхода с альтернативным подходом «Max ВМ», основывающимся на максимизации детерминированной области. Реализация предлагаемого подхода поддерживает воспроизведение однопроцессорной ВМ на базе IA-32.

Она была протестирована на популярных ОС: Microsoft Windows XP, Microsoft Windows 7 и GNU/Linux 3.12.

Свойства реализации предлагаемого подхода соответствуют практическим требованиям, не уступают свойствам альтернативного подхода, а в части замедления (как при записи журнала, так и при воспроизведении) показываются лучшие результаты. Вносимое предложенным методом замедление составляет от 6 до 42% в зависимости от вида гостевого кода. Снижение пропускной способности сетевого взаимодействия гостевой системы составляет порядка 24% при несущественном увеличении времени отклика. Во время воспроизведения вычислительный процесс замедляется не более чем на десятичный порядок. Скорость роста журнала непосредственно зависит от интенсивности обмена информацией с внешним миром (включая трафик с виртуальным диском). При работе без взаимодействий с внешним миром скорость роста журнала имеет порядок десятков КБайт в секунду, что близко к росту журнала в подходе «Max ВМ».

В текущей реализации используется один счётчик инструкций. В итоге корректная работа ВМ с несколькими процессорам (многоядерный ЦПУ) не гарантируется. Очевидное решение: использовать разные счётчики инструкций и разные потоки событий для разных ЦПУ. Данная проблема ещё подлежит исследованию.

Список литературы

[1]. QEMU open source processor emulator — http://wiki.qemu.org/Main_Page
[2]. G.Altekar, I.Stoica. ODR: Output-Deterministic Replay for Multicore Debugging, UC Berkley, October 2009.
[3]. А.Ю.Тихонов, А.И. Аветисян. Развитие taint-анализа для решения задачи поиска программных закладок. Труды Института системного программирования РАН, том 20, 2011 г., стр. 9-24.
[4]. P. Colp, S. Dadizadeh, M. Nanavati. Deterministic Replay for Xen. Department of Computer Science. University of British Columbia. Vancouver, BC, Canada
[5]. 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.
[6]. К. Батузов, П. Довгалюк, В. Кошелев, В. Падарян. Два способа организации механизма полносистемного детерминированного воспроизведения в симуляторе QEMU. Труды Института системного программирования РАН, том 22, 2012г., стр. 77-94.
[7]. Довгалюк П. Детерминированное воспроизведение процесса выполнения программ в виртуальной машине. Труды Института системного программирования РАН, том 21, 2011г., с. 123-132.
[8]. NBench benchmark port to Linux/Unix — http://www.tux.org/~mayer/linux/bmark.html
[9]. Iperf — The TCP/UDP Bandwidth Measurement Tool — https://iperf.fr

Deterministic Replay Specifics in Case of Minimal Device Set

V.Y. Efimov <real@ispras.ru>
K.A. Batuzov <batuzovk@ispras.ru>
V.A. Padaryan <vartan@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, Russia, 109004

Abstract. The deterministic replay technique can be used for debugging, improving reliability and robustness, software development and incident investigation (including reverse engineering of malware). The paper describes the implementation of deterministic replay of IA-32 based boards in QEMU. Another implementation of this technique in QEMU had been previously published, but it uses a significantly different approach. Deterministic replay implementation details and features substantially depend on deterministic area — the part of virtual machine which execution is being replayed. For replay to be deterministic the implementation must ensure that (1) all information flows across deterministic area borders should be logged and then replayed, and (2) there is no non-determinism inside deterministic area. The proposed approach is called «Min VM» because it's based on the minimal deterministic area while the former one should be called «Max VM» as it attempts to stretch deterministic area to cover whole virtual machine. The proposed approach shows the advantages of lower time overhead for logging phase and easier support (because it is much easier to ensure determinism of small deterministic area). On the other side the shortcoming is larger log size mostly because deterministic area doesn't include hard disks so all data flows from disks are being logged. It makes the self-sufficient replay log: image of the original HDD is not needed to replay the execution. The implementation has been tested on popular operating systems: Windows XP, Windows 7 and GNU/Linux 3.12. The current implementation shows 6 – 42% slowdown depending on application code that exceeds previous approach slowdown (17 – 79%).

Keywords: deterministic replay, emulator, QEMU, virtual machine.

DOI: 10.15514/ISPRAS-2015-27(2)-5

For citation: Efimov V.Y., Batuzov K.A., Padaryan V.A. Deterministic Replay Specifics in Case of Minimal Device Set. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 2, 2015, pp. 65-92 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-5.

References

[1]. QEMU open source processor emulator — http://wiki.qemu.org/Main_Page
[2]. G.Altekar, I.Stoica. ODR: Output-Deterministic Replay for Multicore Debugging, UC Berkley, October 2009.

[3]. A.Ju.Tihonov, A.I. Avetisjan. Razvitie taint-analiza dlja reshenija zadachi poiska programmnyx zakladok [Development of taint analysis to search for software backdoors]. Trudy ISP RAN [The Proceedings of ISP RAS], volume 20, 2011. p. 9-24. (In Russian)
[4]. P. Colp, S. Dadizadeh, M. Nanavati. Deterministic Replay for Xen. Department of Computer Science. University of British Columbia. Vancouver, BC, Canada
[5]. 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.
[6]. K. Batuzov, P. Dovgaljuk, V. Koshelev, V. Padarjan. Dva sposoba organizacii mexanizma polnosistemnogo determinirovannogo vosproizvedenija v simuljatore QEMU [Two approaches of deterministic replay development for QEMU system emulator]. Trudy ISP RAN [The Proceedings of ISP RAS], volume 22, 2012 г. p. 77-94. (In Russian)
[7]. Dovgaljuk P. Determinirovannoe vosproizvedenie processa vypolnenija programm v virtualqnoj mashine [A deterministic replay of software execution in virtual machine]. Trudy ISP RAN [The Proceedings of ISP RAS]. volume 21, 2011, p. 123-132. (In Russian)
[8]. NBench benchmark port to Linux/Unix — http://www.tux.org/~mayer/linux/bmark.html
[9]. Iperf — The TCP/UDP Bandwidth Measurement Tool — https://iperf.fr