Мультиплатформенный метод обратной отладки виртуальных машин

Довгалюк П. М., Климушенкова М. А., Дмитриев Д. С., Макаров В. А.
Новгородский государственный университет им. Ярослава Мудрого
В. Новгород, ул. Б. С.-Петербургская, 41
Emails: {pavel.dovgaluk, maria.klimushenkova, denis.dmitriev, vladimir.makarov}@ispras.ru

Аннотация—Отладка и прототипирование новых аппаратных платформ, периферийных устройств и операционных систем осложняется недетерминированностью их поведения, воздействием отладчика на систему, длительностью процесса воспроизведения ошибок, отказами всей системы и другими подобными факторами. В работе представлен метод детерминированного воспроизведения работы виртуальных машин и показано как он позволяет бороться со сложностями, возникающими при отладке. На основе разработанного подхода авторами создан мультиплатформенный полносистемный обратный отладчик. С помощью этого отладчика становится возможным изучение сбоев без воздействия на их проявление. Можно воспроизводить и многократно изучать состояние всей виртуальной машины и всех ее периферийных устройств. В работе представлен обратный отладчик, поддерживающий целевые платформы i386, x86-64, MIPS, PowerPC и ARM. Он протестирован для гостевых ОС Windows и GNU/Linux. Отладчик можно использовать при моделировании периферийных устройств и отладке пользовательского и системного кода. Влияние на процесс отладки ограничено лишь вносимым при записи работы системы замедлением. Это замедление незначительно влияет на работу гостевой системы и позволяет отлаживать критичные ко времени приложения.

1. Введение[1]

Прототипирование новых аппаратных платформ, периферийных устройств, операционных систем и драйверов устройств требуют значительных затрат на симуляцию, тестирование и отладку [1]. Эти процессы можно упростить и ускорить с помощью применения технологий виртуализации [2].

Отладку усложняют ошибки в ядре ОС, состояния гонки и сбои оборудования — каждый раз, когда происходит критическая ошибка, разработчик должен перенастроить окружение и перезапустить отлаживаемую систему. Это приводит к потерям времени, а кроме того, в результате сбоев могут быть повреждены данные в системе. Сбои могут проявляться каждый раз по-разному или не проявляться вообще [3].

1. Работа была выполнена при частичной поддержке РФФИ, проект 14-07-00411 а

Если используется интерактивный отладчик, то задержки при остановке программы могут вызвать сбои из-за таймаутов оборудования или удаленных систем. Также отладчик может повлиять на работу программы из-за состояний гонки, неинициализированной памяти и т. п.

Традиционная итеративная отладка требует, чтобы программа при каждом запуске работала одинаково. Но при взаимодействии с внешней аппаратурой становится сложнее обеспечить это условие, так как начальное состояние устройств должно быть одним и тем же. Разработчику приходится затрачивать дополнительное время для инициализации и настройки устройств и окружения.

Обратная отладка позволяет уменьшить влияние подобных проблем на процесс разработки драйверов и моделей оборудования. Вместо повторного выполнения программы можно записать ее работу и изучать ошибку многократно с помощью воспроизведения этой записи.

Обратная отладка нацелена на изучение прошедших состояний системы, включая значения ячеек памяти и переменных в программе [4]. Например, можно установить точку останова и «выполнить» программу назад, чтобы узнать, когда это точка срабатывала ранее. Основное преимущество обратной отладки — это возможность отслеживания источников значений данных. Кроме того, можно отделить этап анализа или отладки программы от этапа ее выполнения. Таким образом динамический анализ не будет влиять на работу программы [5].

Наша цель — это упрощение отладки операционных систем, драйверов, моделей и прототипов периферийных устройств. Для этого используется обратная отладка на основе виртуальных машин. Воспроизведение работы всей виртуальной машины позволяет изучать код ядра операционной системы, состояние всех виртуальных устройств и взаимодействие с внешними устройствами. Многоплатформенный механизм воспроизведения позволяет упростить протопипирование виртуальных устройств, подключаемых к различным платформам, переносимых операционных систем, а также использовать платформо-независимые методы динамического анализа программ. Многоплатформенность позволяет многократно использовать код для анали-

за программ без затрат времени на его портирование под разные платформы, если это потребуется.

Те немногие существующие разработки, которые доступны для использования или описаны в публикациях, обычно поддерживают только одну или две гостевые платформы. Simics, единственное по-настоящему многоплатформенное средство для обратной отладки, работает слишком медленно, чтобы можно было отлаживать критичные по времени программы (например, сетевые приложения) или взаимодействие с периферией [6].

В настоящей работе представлен подход к созданию мультиплатформенного средства обратной отладки. Это средство может быть использовано для отладки пользовательского и системного кода, а также моделей аппаратных устройств. В работе представлены следующие результаты:

- Метод воспроизведения работы системы, который не зависит от аппаратной и программной гостевых платформ. Новизна метода воспроизведения заключается в использовании платформо-независимых средств для внедрения недетерминированных событий.
- Реализация обратной отладки на основе симулятора QEMU и отладчика GDB. Отладчик работает с современными аппаратными платформами и операционными системами, и не требует их модификации.
- Оценка производительности симулятора со включенным механизмом воспроизведения программ и сравнение его с другими существующими реализациями.

Работа состоит из следующих разделов. В разделах 2 и 3 описаны особенности предлагаемого метода и его реализации. В разделе 4 представлена реализация обратной отладки на основе симулятора QEMU, совместимая с интерфейсом отладчика GDB. В раздел 5 приведены результаты измерения производительности симулятора при записи и воспроизведении работы виртуальной машины. Раздел 6 содержит обзор других исследований в области обратной отладки виртуальных машин и их сравнение с предлагаемым подходом. Разделы 7 и 8 содержат заключение и описание направлений продолжения исследований.

2. Платформо-независимая обратная отладка

Необходимость создания нового реверсивного отладчика продиктована тем, что не существует других средств обратной отладки, которые могли бы работать со всеми современными платформами, отлаживать взаимодействие с внешними устройствами и были бы достаточно быстрыми для отладки критичных ко времени приложений.

Для полносистемной отладки и поддержки различных платформ отладчик может быть реализован на основе монитора виртуальных машин. Поэтому при его реализации мы использовали симулятор QEMU. Он поддерживает множество гостевых систем, например x86, x86-64, ARM, MIPS и PowerPC [7]. QEMU транслирует гостевой бинарный код в код хозяйской машины, а затем исполняет уже его. За счет кэширования ранее транслированных блоков QEMU работает быстрее, чем программные интерпретаторы.

Для создания механизма обратной отладки мы использовали идею детерминированного воспроизведения, когда во время выполнения сценария с исследуемой ошибкой записываются все недетерминированные данные в системе. Такие данные могут поступать от пользователя, удаленных компьютеров, часов реального времени и внешних устройств. Аппаратные прерывания также должны быть записаны, так как они возникают в случайные моменты времени.

Наш подход основывается на воспроизведении состояния центрального процессора, памяти и всех виртуальных периферийных устройств, подключенных к системе. Для этого перехватываются все входы к этим компонентам системы. Данные, поступающие на эти входы, записываются (а затем и воспроизводятся) в журнал недетерминированных событий, как показано на рисунке 1.

Новизна нашего подхода заключается в создании аппаратно-независимого слоя внутри симулятора для захвата недетерминированных данных. Мы используем платформо-независимый счетчик инструкций процессора, в то время как в других подходах для этого используются платформо-зависимые регистры [8], [9]. Эти свойства позволяют нам создать средство обратной отладки, поддерживающее множество аппаратных платформ. Добавляемые в симулятор платформы автоматически получают возможность использования обратной отладки без их модификации.

На основе этого подхода мы реализовали платформо-независимые запись и воспроизведение работы виртуальной машины, а также обратную отладку для нее. Все функции, связанные с воспроизведением, работают с абстрактными данными и событиями. Можно записывать работу всей виртуальной машины, а затем воспроизводить все программы и состояния виртуальных устройств или внешних моделей устройств (например, моделей SystemC [1]) с целью отладки или анализа их работы.

3. Реализация детерминированного воспроизведения

Наш механизм записи и воспроизведения основан на сохранении недетерминированных событий (например, ввода с клавиатуры) и симуляции детерминированных (например, чтения жесткого диска или

Рис. 1. Мультиплатформенное воспроизведение работы виртуальных машин на основе QEMU.

гостевой памяти) [10]. Сохранение только высокоуровневых событий значительно уменьшает объем журнала событий по сравнению с реализациями, сохраняющими обращения процессора к портам ввода-вывода (например, [8]), что в конечном итоге ускоряет процесс симуляции и позволяет использовать обратную отладку для критичных по времени приложений.

В журнал сохраняются следующие данные от периферийных устройств: ввод с клавиатуры и мыши, сетевые пакеты, ввод от аудио контроллера, пакеты USB, ввод с последовательного порта и показания аппаратных часов реального времени. Ввод из виртуальных устройств, гостевой памяти, программные прерывания и выполнение отдельных инструкций не сохраняются в журнал, потому что они детерминированные и могут быть воспроизведены с помощью симуляции работы виртуальной машины, если начинать ее с одного и того же стартового состояния.

Для реализации воспроизведения работы виртуальной машины нам нужно было решить три задачи: запись недетерминированных событий, воспроизведение событий и проверка того, что поведение виртуальной машины при воспроизведении не изменилось.

Для этого необходимо было привязать внешние события к моменту выполнения гостевой программы, чтобы воспроизводить их в нужное время. Мы задаем эти моменты времени с помощью подсчета инструкций, выполнившихся между каждой парой последовательных событий.

Чтобы убедиться, что работа виртуальной машины соответствует работе во время записи выполнения, мы проверяем, что внутренние события машины (обращения к аппаратным часам, прерывания, исключительные ситуации) считываются из журнала в правильном порядке. Если прочитано событие, которое не может произойти в текущем состоянии симулятора, то воспроизведение останавливается с сообщением об ошибке.

3.1. Подсчет инструкций

Для подсчета количества инструкций между внешними событиями мы используем встроенный в QEMU независимый от гостевой платформы счетчик icount. Этот счетчик работает в отдельном режиме работы icount. Он позволяет детерминированно выполнять программу в отсутствие внешних воздействий. Мы расширили этот режим, чтобы была возможна запись и воспроизведение работы виртуальной машины со всеми внешними взаимодействиями.

QEMU использует динамическую трансляцию гостевого кода. Каждая гостевая инструкция преобразуется в последовательность команд хозяйской машины. Инструкции объединяются в блоки трансляции — непрерывные последовательности инструкций. Блоки трансляции всегда выполняются от начала до конца целиком, за исключением случаев, когда возникают исключительные ситуации.

С помощью icount можно узнавать точное количество выполненных инструкций. Но счетчик инструкций увеличивается не после каждой инструкции, а лишь в начале каждого блока трансляции. Если же возникает исключительная ситуация, то блок транслируется повторно для того, чтобы вычислить сколько инструкций успело выполниться с начала блока до возникновения исключения.

Мы используем icount, чтобы управлять восстановлением последовательности недетерминированных событий. Число инструкций между событиями записывается в журнал. А при воспроизведении счетчик icount останавливается на том месте, где должно быть загружено очередное событие.

Использование icount позволяет контролировать выполнение недетерминированных событий, но при этом уменьшает производительность симулятора. Код для подсчета инструкций добавляется к коду блока трансляции, как показано на рисунке 2. Знаком “+” помечены операции, выполняющие подсчет инструкций. Оценка влияния этого кода на производительность приведена в разделе 5.

Выполняемый блок трансляции

```
0x000fd1fe:  mov    %eax,%gs
0x000fd200:  mov    %ecx,%eax
0x000fd202:  jmp    *%edx
```

Промежуточное представление блока трансляции

```
  ld_i32 tmp12,env,$0xfffffff4
  movi_i32 tmp13,$0x0
  brcond_i32 tmp12,tmp13,ne,$0x0
+ ld_i32 loc14,env,$0xffffffe8
+ movi_i32 tmp12,$0x3
+ sub_i32 loc14,loc14,tmp12
+ movi_i32 tmp12,$0x0
+ brcond_i32 loc14,tmp12,lt,$0x1
+ st16_i32 loc14,env,$0xffffffe8
  ---- 0xfd1fe
  mov_i32 tmp0,eax
  movi_i32 tmp3,$0xfd1fe
  st_i32 tmp3,env,$0x20
  mov_i32 tmp6,tmp0
  movi_i32 tmp12,$0x5
  call load_seg,$0x0,$0,env,tmp12,tmp6
  ---- 0xfd200
  mov_i32 tmp0,ecx
  mov_i32 eax,tmp0
  ---- 0xfd202
  mov_i32 tmp0,edx
  st_i32 tmp0,env,$0x20
  ----
  exit_tb $0x0
  set_label $0x0
  exit_tb $0x5c8033b
+ set_label $0x1
+ exit_tb $0x5c0045a
```

Рис. 2. Трансляция кода в режиме icount.

3.2. Часы реального времени

QEMU использует часы реального времени хозяйской машины (real time clock, RTC) для использования в гостевой машине или для внутренних таймеров. С помощью таймеров функции обратного вызова для различных подсистем QEMU вызываются в заданные моменты времени. Есть несколько типов часов для таймеров:

- Часы реального времени (real time). Используется для функций, которые не меняют состояния гостевой системы. Поэтому эти часы не влияют на работу записи и воспроизведения.
- Виртуальные часы (virtual). Эти часы идут только во время эмуляции. В режиме icount их показания вычисляются на основе счетчика инструкций. В результате они становятся детерминированными и их показания не нужно записывать в журнал событий.
- Хозяйские часы (host). Они используются моделями устройств, которые эмулируют источники реального времени (например, микросхему часов). Поэтому они являются одним из источников недетерминизма и их показания записываются в журнал при считывании времени.
- Часы реального времени для icount. Это те же самые часы, что и первые, но они используются для увеличения показаний виртуальных часов, когда гостевой процессор не работает. Эти изменения влияют на работу гостевой системы и поэтому записываются в журнал событий.

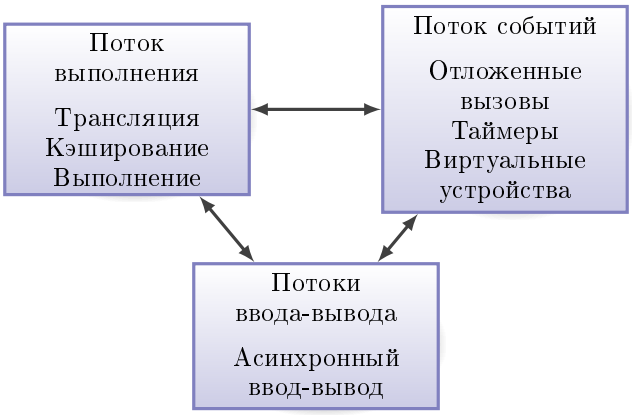

Рис. 3. Структура потоков управления QEMU.

Для тех часов, чьи показания записываются в журнал, мы создали функции-обертки вокруг обращений к ним. В режиме записи эти функции записывают возвращаемое время в журнал. В режиме воспроизведения реального обращения к часам не происходит, а выполняется чтение из журнала.

3.3. Контрольные точки симуляции

Работа с часами или периферией в большинстве случаев выполняется синхронно с основным процессором, так как эти события инициируются гостевыми инструкциями, которые все работают в одном потоке выполнения. Но кроме этого потока в QEMU работает еще несколько потоков для обеспечения асинхронного ввода-вывода.

Симулятор выполняется в нескольких потоках, как изображено на рисунке 3. Поток выполнения работает в цикле и вызывает модуль исполнения гостевого кода. Модуль исполнения выполняет трансляцию очередного блока или берет его из кэша, а затем выполняет. В потоках событий и ввода-вывода выполняются отложенные или асинхронные задания. Таким образом чтение данных с диска может производиться параллельно с выполнением кода. Кроме того, несмотря на то, что недетерминированные показания часов и записываются в журнал, обращения к этим часам могут происходить в произвольные моменты времени, когда симулятор проверяет не истекли ли таймеры.

Такая асинхронная архитектура усложняет детерминированное воспроизведение работы гостевой машины. Кроме недетерминированного ввода от периферийных устройств, появляется еще недетерминированное межпотоковое взаимодействие и обработка таймеров.

Так как мы хотим поддерживать состояние всех устройств в корректном состоянии как во время записи, так и во время воспроизведения, нужно поддерживать относительный порядок таймерных вызовов друг для друга, если они привязаны к одному моменту в потоке инструкций гостевой системы. Для этого мы добавили в симулятор контрольные точки. Каждый раз, когда выполняется очередная контрольная точка, в журнал событий записывается информация об этом. Контрольные точки позволяют обрабатывать таймеры в том же порядке при воспроизведении, в котором они выполнялись при записи.

Контрольые точки также позволяют корректно управлять изменением счетчика времени, когда процессор виртуальной машины не выполняет код. За это отвечает функция qemu_clock_warp, вызовы которой привязываются к контрольной точке. Тогда эта функция вызывается в одни и те же моменты времени выполнения гостевой программы и при записи, и при воспроизведении ее работы.

3.4. Начальное состояние системы

Перед воспроизведением записанного сценария работы системы нужно привести ее в начальное состояние, соответствующее начальному состоянию при записи работы. И так как виртуальная машина выполняет тот же код, а все недетерминированные события записаны в журнал, ее состояния будут при воспроизведении меняться так же, как и менялись при записи.

3.5. Асинхронный ввод-вывод

Дисковый ввод и вывод оперирует одними и теми же данными и при записи, и при воспроизведении сценария работы системы. Но из-за того, что операции чтения и записи работают в отдельных потоках, может произойти рассинхронизация с работой процессора.

Для того, чтобы сделать ввод-вывод детерминированным, мы синхронизировали эти операции с контрольными точками симулятора. Когда создается задание на ввод-вывод, оно добавляется в очередь. При прохождении контрольной точки все задания из очереди выполняются, а их список сохраняется в журнале. Когда сценарий работы воспроизводится, содержимое очереди сопоставляется с содержимым журнала. Если в журнале нужного задания нет, то оно ожидает следующей контольной точки.

3.6. Внешние устройства

QEMU поддерживает подключение реальных USB-устройств к виртуальной машине. С помощью этого можно прототипировать и отлаживать программное обеспечение, работающее с USB. QEMU работает с USB версий 1.1–3.0. Наш механизм записи и воспроизведения работает в абстрактном слое оборудования QEMU, поэтому он поддерживает воспроизведение работы с реальными USB-устройствами, подключаемыми к симулятору.

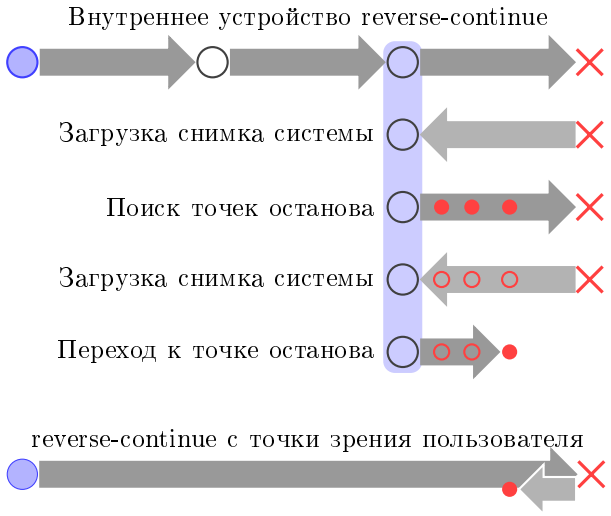

Рис. 4. Обратная отладка с помощью детерминированного воспроизведения.

4. Обратная отладка через интерфейс GDB

Обратная отладка — это прохождение процесса выполнения программы в обратном направлении с целью изучения ее состояния в разные моменты времени [11]. Обратная отладка позволяет использовать точки останова и находить те из них, которые должны были бы сработать в прошлом, если бы были установлены.

QEMU поддерживает обычную отладку через интерфейс GDB для удаленной отладки. Этот интерфейс поддерживает точки останова, точки наблюдения, пошаговое выполнение и другие традиционные возможности отладки. Но так как обратная отладка в QEMU реализована не была, нам пришлось расширить QEMU, чтобы сделать обратную отладку доступной через интерфейс GDB. Мы добавили поддержку операций «шаг назад» (reverse step) и «продолжить назад» (reverse continue). Выполнение этих операций становится возможным в режиме воспроизведения работы системы. «Шаг назад» переходит к предыдущему записанному шагу (инструкции), а «продолжить назад» находит самую позднюю из предшествующих точек останова.

Для обеих этих команд используются заранее сделанные снимки состояния системы, как показано на рисунке 4. Снимки делаются во время записи сценария работы системы. Первый снимок делается при старте, а остальные через каждые N секунд (где N — опция командной строки).

После загрузки состояния из сохраненного снимка симулятор воспроизводит сценарий работы, чтобы найти заданную точку (для шага назад) или для поиска сработавшей точки останова (для команды продолжить назад). После этого для продолжения назад нужно сделать еще один дополнительный проход. После достижения точки откуда команда была подана, нужно снова вернуться назад и перейти на найденную точку останова.

Пользователю для отладки в обратном направлении нужно лишь подать команду симулятору через отладчик. Мы реализовали эти команды так, что все действия, связанные с загрузкой состояний системы и несколькими проходами воспроизведения, выполняются автоматически.

5. Оценка производительности записи и воспроизведения

В этом разделе приводятся оценки замедления симулятора, возникающего при записи и воспроизведении работы виртуальной машины по отношению к обычной симуляции. Для этого мы использовали несколько тестов. В первом тесте мы загружали Windows XP на виртуальной машине с процессором x86. В остальных тестах загружалась операционная система Debian Wheezy в системах на x86, ARM, PowerPC и MIPS. У всех виртуальных машин было по 128 Мб памяти.

Симулятор запускался на компьютере, оснащенном процессором Intel Core i7 с 8 ядрами на частоте 3.40GHz и 8Gb ОЗУ под управлением 64-битной версии Ubuntu 14.04.

Каждый тест выполнялся в 4 режимах:

- regular Симуляция в QEMU без использования механизма icount для подсчета инструкций. Это недетерминированный режим и приводится для сравнения с режимом детерминированного выполнения.
- icount Включен подсчет инструкций с помощью опции -icount shift=7. Этот режим необходим для детерминированного выполнения инструкций и хода виртуальных часов. Внешние воздействия остаются недетерминированными.
- icount+record Режим записи работы системы. Все внешние устройства подключены к виртуальной машине, а поступающие от них данные записываются в журнал событий.
- icount+replay Работа системы воспроизводится по ранее записанному журналу. Внешние устройства не подключены, а данные поступают из журнала.

Мы измеряли время выполнения каждого теста в каждом из режимов. Результаты представлены в таблице 1. Кроме времени в секундах в таблице приведены два нормализованнх значения для каждого теста. Первое нормализованное значение показывает увеличение времени работы симулятора по отношению к режиму обычного выполнения. Второе нормализованное значение позволяет оценить время, затрачиваемое на запись журнала событий и его воспроизведение.

Таблица 1. Производительность детерминированного воспроизведения

Режим работы	Время выполнения, с	Отношение к обычному времени	Отношение ко времени с icount
Загрузка Windows (x86)			
regular	75	1.00	—
icount	89	1.19	1.00
icount+record	98	1.31	1.10
icount+replay	228	3.04	2.56
Загрузка Debian Wheezy (x86)			
regular	130	1.00	—
icount	177	1.36	1.00
icount+record	183	1.41	1.03
icount+replay	418	3.22	2.36
Загрузка Debian Wheezy (ARM)			
regular	124	1.00	—
icount	157	1.27	1.00
icount+record	164	1.32	1.04
icount+replay	457	3.69	2.91
Загрузка Debian Wheezy (PowerPC)			
regular	74	1.00	—
icount	71	0.96	1.00
icount+record	153	2.07	2.15
icount+replay	очень долго	—	—
Загрузка Debian Wheezy (MIPS)			
regular	160	1.00	—
icount	179	1.12	1.00
icount+record	182	1.14	1.02
icount+replay	428	2.68	2.39

Включение icount делает работу системы детерминированной. Виртуальное время в системе вычисляется на основе числа выполненных инструкций. Поэтому скорость симуляции не будет влиять на работу системы. Виртуальная машина может обнаружить замедление лишь при взаимодействии с внешним миром (сетью или USB-устройствами). Если замедление будет слишком велико, это может повлиять на удобство использования системы во время записи сценария ее работы. В нашем случае замедление из-за подсчета инструкций невелико и составляет от 12 до 36%

Замедление из-за записи журнала также может исказить работу системы. Чтобы оценить это замедление, мы разделили время выполнения с записью сценария ко времени работы со включенным icount. Полученные значения представлены в четвертой колонке таблицы 1. В наших тестах замедление записи находится в пределах от 3 до 10% (за исключением PowerPC).

Замедление при воспроизведении не влияет на работу гостевой машины, а только на удобство пользователя. Замедление в наших тестах незначительно влияет на удобство работы и находится в пределах

Таблица 2. Скорость роста журнала для разных платформ.

Платформа	Размер журнала, байт	Выполнено инструкций	Байт на 1000 инструкций
x86	139M	6346M	21.9
ARM	80M	4469M	17.9
PowerPC	2258M	4790M	471.4
MIPS	257M	3408M	75.4

от 204 до 269%.

Кроме замедления мы также измерили объем дискового пространства, используемого для хранения журнала событий. Журнал растет на 10 Кб/с, когда ОС находится в режиме ожидания, до 715 Кб/с, если происходит много прерываний и дисковых операций. Скорость роста журнала достаточно небольшая для того, чтобы записывать длительные сценарии работы, вплоть до нескольких дней. В таблице 2 приведена скорость роста журнала для случаев загрузки Debian на разных платформах.

Симулятор платформ на процессоре PowerPC показал заметно отличающиеся от остальных результаты. Симулятор со включенным подсчетом инструкций работает быстрее, чем с выключенным подсчетом, а воспроизведение записанного сценария работает на порядки медленнее, чем обычное его выполнение. Скорость роста журнала также значительно больше, чем для других платформ. В дальнейшем мы планируем разобраться с этим случаем, чтобы исправить работу записи и воспроизведения для PowerPC.

6. Обзор полносистемной обратной отладки

Существует множество работ, описывающих подходы и реализации полносистемной обратной отладки [5], [8], [9], [12], [13]. Некоторые из них направлены на работу только с одной гостевой платформой, другие слишком медленные или не поддерживают работу с периферийными устройствами. Мы также не будем рассматривать здесь подходы, направленные на отладку отлдельных программ, так как сфера их применения отличается от полносистемных отладчиков.

PANDA — симулятор на основе QEMU, поддерживающий запись и воспроизведение для нескольких аппаратных архитектур [8]. Во время воспроизведения PANDA восстанавливает только состояние ОЗУ и центрального процессора. В журнал записываются читаемые процессором данные из портов, произошедшие прерывания и DMA-транзакции.

PANDA использует платформо-зависимый способ определения того, когда нужно воспроизводить очередное событий из журнала. Для x86 моменты возникновения событий определяются с помощью счетчика инструкций (EIP), количества выполненных инструкций и значения неявной переменной цикла (регистра ECX).

PANDA не может продолжить нормальное выполнение после воспроизведения записанного журнала, так как восстанавливаются только состояния ЦП и ОЗУ. Поэтому пользователь не может изучать состояние виртуальных устройств при их моделировании. Наша реализация записи и воспроизведения позволяет продолжать работу в любой момент воспроизведения сценария работы, потому что все виртуальные устройства всегда находятся в согласованном состоянии. Замедление записи работы для PANDA составляет 85%, а воспроизведения — 257%.

Simics — это мультиплатформенный симулятор от компании Wind River [6]. Поддержка обратной отладки в нем появилась в 2005 году. Обратная отладка возможна для целой виртуальной машины и даже для нескольких машин одновременно. Simics симулирует множество платформ (включая x86-64, ARM, MIPS и PowerPC) и операционных систем. Все это делает Simics одним из лучших коммерческих обратных отладчиков. С другой стороны, из-за такого обширного набора возможностей, стоимость Simics очень высока, что делает его недоступным для большинства разработчиков.

В то же время Simics работает слишком медленно для комфортной отладки приложений, взаимодействующих с пользователем и реальным оборудованием. В работе [14] приведены данные о том, что Simics выполняется в 40 раз медленнее, чем QEMU.

7. Заключение

В работе представлен полносистемный легковесный платформо-независимый метод записи и воспроизведения работы виртуальных машин. Наш метод был реализован на основе мульти-платформенного симулятора QEMU, а исходные тексты были опубликованы на github[2]. Изменения в симулятор вносились только платформо-независимые модули, поэтому запись и воспроизведение автоматически становятся доступными для всех аппаратных платформ, поддерживаемых в QEMU. Мы протестировали запись и воспроизведение для платформ i386, x86-64, MIPS, PowerPC и ARM.

Механизм записи поддерживает работу с сетевыми картами, аудио адаптером, последовательным портом и USB-устройствами. Его работа была протестирована с рядом современных USB-устройств, работающих на разных скоростях и в разных режимах.

Обратная отладка, добавленная в QEMU, позволяет изучать состояние всей системы, возвращаясь в любой момент времени записанного сценария. Механизм обратной отладки поддерживает переход на шаг назад и поиск точек останова назад во времени. Отлаживать работу с реальными устройствами можно без их подключения, записав однажды сценарий работы.

Реализованный механизм записи и воспроизведения работы программ можно использовать для от-

2. https://github.com/Dovgalyuk/qemu/tree/rr-17

ладки драйверов, кода операционных систем и пользовательских приложений, не затрачивая времени на повторное воспроизведение ошибок и настройку системы. Воспроизведение виртуальной машины можно использовать при прототипировании новых платформ и устройств. Разработанную модель устройства, не существующего «в железе», можно изучать в режиме «медленной пермотки» благодаря детерминированному воспроизведению работы системы.

8. Направления будущей работы

Разработанный механизм детерминированного воспроизведения выложен в открытый доступ. Сейчас мы дорабатываем его для включения в основную ветку симулятора QEMU. Это сделает обратную отладку доступной для всех разработчиков, использующих QEMU. Также мы планируем доработать запись и воспроизведение для ускорения эмуляции платформ на PowerPC в этих режимах. Детерминированное воспроизведение может быть полезным и при использовании других методов анализа программ помимо обратной отладки. Анализ уже записанного сценария может быть сколь угодно «тяжелым» и при этом не влиять на работу приложения. Например, анализ помеченных данных может добавлять замедление до 600% [15]. Перенос этого замедления с фазы выполнения программы на фазу ее воспроизведения может сделать результаты анализа более достоверными. А сам анализ можно ускорить за счет распараллеливания на нескольких компьютерах [14].

Список литературы

[1] M. Monton, A. Portero, M. Moreno, B. Martinez, and J. Carrabina, "Mixed sw/systemc soc emulation framework," in Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on, June 2007, pp. 2338–2341.

[2] M. Baklashov, "An on-line memory state validation using shadow memory cloning," in Proceedings of the 2011 IEEE 17th International On-Line Testing Symposium, ser. IOLTS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 186–189. [Online]. Available: http://dx.doi.org/10.1109/IOLTS.2011.5993837

[3] J. Gray, "Why do computers stop and what can be done about it?" 1985.

[4] J. Engblom, "A review of reverse debugging," in in S4D, 2012.

[5] J. Chow, T. Garfinkel, and P. M. Chen, "Decoupling dynamic program analysis from execution in virtual environments," in USENIX 2008 Annual Technical Conference on Annual Technical Conference, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404015

[6] J. Engblom, D. Aarno, and B. Werner, "Full-system simulation from embedded to high-performance systems," in Processor and System-on-Chip Simulation, R. Leupers and O. Temam, Eds. Springer US, 2010, pp. 25–45. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-6175-4_3

[7] F. Bellard, "Qemu, a fast and portable dynamic translator," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247401

[8] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, "Repeatable reverse engineering for the greater good with panda," Oct. 2014.

[9] H. Liu, H. Jin, X. Liao, and Z. Pan, "Xenlr: Xen-based logging for deterministic replay," in Proceedings of the 2008 Japan-China Joint Workshop on Frontier of Computer Science and Technology, ser. FCST '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 149–154. [Online]. Available: http://dx.doi.org/10.1109/FCST.2008.31

[10] P. Dovgalyuk, "Deterministic replay of system's execution with multi-target qemu simulator for dynamic analysis and reverse debugging," in Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, ser. CSMR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 553–556.

[11] B. Boothe, "Efficient algorithms for bidirectional debugging," SIGPLAN Not., vol. 35, no. 5, pp. 299–310, May 2000. [Online]. Available: http://doi.acm.org/10.1145/358438.349339

[12] S. S. Chia-Wei Hsu, "Free: A fine-grain replaying executions by using emulation," ser. The 20th Cryptology and Information Security Conference (CISC 2010), 2010.

[13] D. Jacobowitz and B. P., "Reversible debugging," ser. GCC Developer's Summit, 2007.

[14] M. Rittinghaus, K. Miller, M. Hillenbrand, and F. Bellosa, "Simuboost: Scalable parallelization of functional system simulation," in Proceedings of the 11th International Workshop on Dynamic Analysis (WODA 2013), Houston, Texas, Mar. 16 2013.

[15] A. Henderson, A. Prakash, L. K. Yan, X. Hu, X. Wang, R. Zhou, and H. Yin, "Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform," in Proceedings of the 2014 International Symposium on Software Testing and Analysis, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 248–258. [Online]. Available: http://doi.acm.org/10.1145/2610384.2610407