Детерминированное воспроизведение процесса выполнения программ в виртуальной машине[1]

Павел Довгалюк
Pavel.Dovgaluk@ispras.ru

Аннотация. В статье описывается разработка технологии, позволяющей записывать и воспроизводить сценарии выполнения программ в виртуальной машине. Данная технология позволяет выполнять детерминированную отладку приложений, а также должна в дальнейшем лечь в основу реализации различных механизмов динамического анализа программ (в том числе снятия трассы с выполняющейся программы) и реверсивной отладки.

1. Введение

Зачастую программисты в своей работе сталкиваются с ошибками, которые трудно выявить традиционными методами. Например, проявления ошибок могут пропадать в режиме отладки, возникать от случая к случаю, быть результатом сложного взаимодействия потоков в многопоточных приложениях, или требовать сложной настройки окружения, выполняемой перед каждым запуском. Во всех этих случаях процесс отладки мог бы быть значительно упрощен, если можно было бы автоматически записать сценарий, в котором ошибка проявляется, а лишь затем приступить к отладке, многократно проигрывая данный сценарий в симуляторе.

Это становится возможным при использовании механизмов записи недетерминированных событий, происходящих в отлаживаемой системе и последующем их детерминированном воспроизведении.

2. Методы детерминированного воспроизведения программ

Большинство их существующих на настоящий момент технологий детерминированного воспроизведения процесса выполнения программ ограничивают сферу своего применения поддержкой определенной операционной системы, записью только одного процесса, слабой поддержкой

[1] Работа поддержана грантом РФФИ (11-07-00353)

многопоточности, небольшим подмножеством записываемых системных вызовов [[1]], [[2]], [[3]], [[4]].

Из известных решений только одно основано на записи журнала недетерминированных событий в процессе симуляции виртуальной машины – это симулятор VMWare [[5]]. Для этого симулятора реализованы механизмы реверсивной отладки, однако, т.к. он является программой с закрытым исходным кодом, добавить в него связанную с динамическим анализом кода функциональность не представляется возможным. Кроме того, он симулирует только одну аппаратную платформу – i386.

3. Описание предлагаемого подхода

Для реализации технологии детерминированного воспроизведения сценариев выполнения программ был выбран симулятор Qemu. Данный симулятор поддерживает множество архитектур симулируемых машин (i386, ARM, MIPS, PowerPC и т.д.), используется в составе SDK для платформы Android, а также имеет открытый код, что позволяет вносить в него изменения, необходимые для динамического анализа программ.

3.1. Структура симулятора

Симулятор Qemu имеет структуру, показанную на рис. 1Рис. 1.

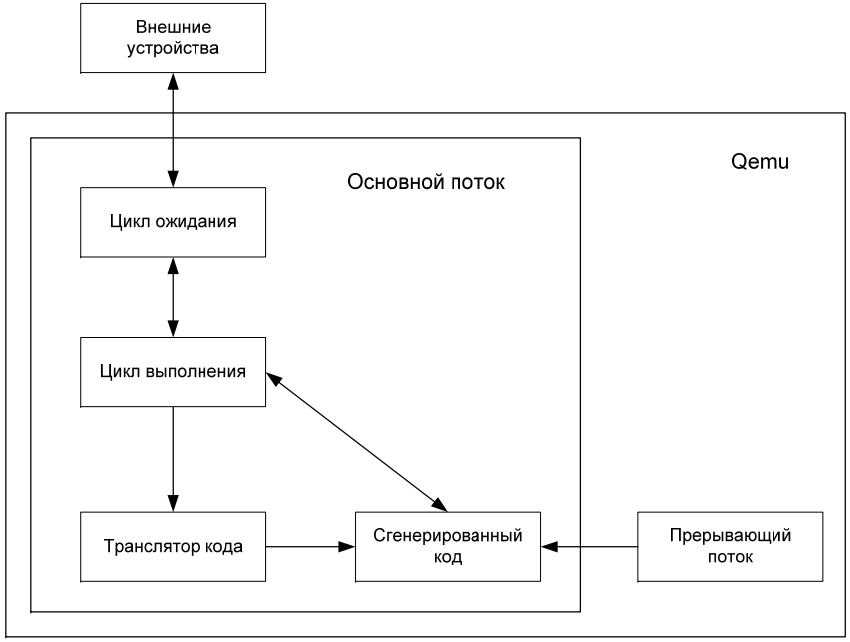

Рис. 1. Структура симулятора Qemu

Цикл выполнения в Qemu производит трансляцию кода симулируемой машины и исполнение сгенерированного кода, обрабатывая небольшую его часть в каждой итерации.

Параллельно с этим работает прерывающий поток, останавливающий выполнение кода с определенной периодичностью. При этом цикл выполнения прерывается и происходит возврат в цикл ожидания.

В цикле ожидания симулятор сначала проверяет состояние внешних устройств и интерфейсов управления симулятором, а затем возвращает управление в цикл выполнения.

Так как прерывающий поток не синхронизирован с основным, детерминированное воспроизведение моментов возникновения прерываний таймера невозможно без переработки этого механизма.

Кроме того, Qemu взаимодействует с внешними устройствами, включая мышь, клавиатуру, сетевую карту, аппаратные таймеры. Через эти внешние устройства в Qemu могут приходить сообщения, которые не являются детерминированными и, следовательно, их обработчики также должны быть доработаны.

3.2. Запись журнала событий

Для того чтобы воспроизводить процесс выполнения программ в виртуальной машине, была реализована запись всех недетерминированных событий в специальный журнал.

События, записываемые в журнал, делятся на синхронные и асинхронные. Синхронные события вызываются действиями, выполняемыми основным потоком (выполнение инструкции, итерации цикла ожидания, чтение часов). Асинхронные события приходят в симулятор извне в произвольный момент времени (прерывание таймерного потока, нажатие клавиши на клавиатуре, движение мыши, приход сетевого пакета).

3.2.1. Выполнение очередной инструкции

Выполнение инструкции является внутренним детерминированным событием. Однако, для того, чтобы корректно воспроизводить моменты возникновения недетерминированных событий (внутренний таймерный поток, события от внешних устройств), необходимо учитывать количество выполненных инструкций.

Для подсчета инструкций был изменен модуль, отвечающий за трансляцию симулируемого кода в код симулирующей машины. После кода, соответствующего каждой инструкции, выполняется код, отвечающий за увеличение счетчика инструкций.

3.2.2. События от часов

В процессе своей работы симулятор осуществляет считывание показаний часов реального времени как для своей работы, так и для передачи в виртуальную машину.

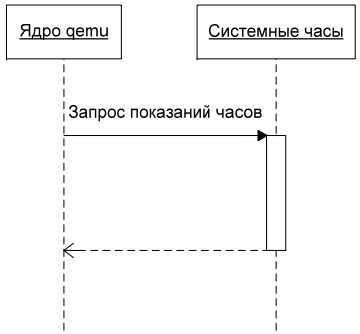

Рис. 2. Получение показаний часов в исходной версии Qemu

Чтобы показания часов реального времени, передаваемые внутрь симулируемой системы, не изменились при воспроизведении поведения системы, в модуль, осуществляющий работу с аппаратными часами, были внесены изменения, позволяющие записывать считанные показания часов в журнал.

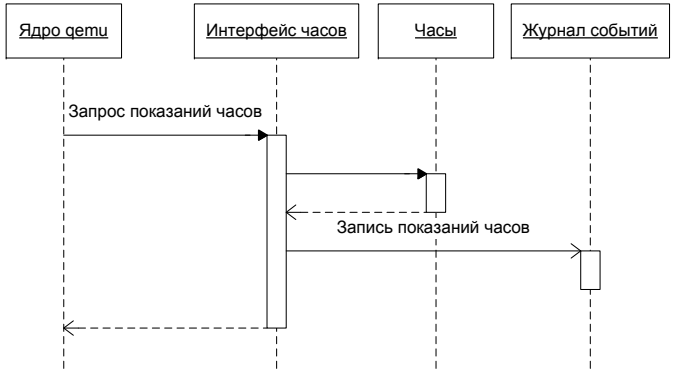

Рис. 3. Запись показаний часов в журнал

При воспроизведении журнала событий вместо считывания показаний аппаратных часов, выполняется чтение их из журнала. Так как воспроизведение является детерминированным, операция чтения происходит в тот же момент (относительно позиции в журнале), что и при записи.

Однако, в случае выхода симулятора в цикл ожидания из цикла выполнения (например, при отладке), могут выполняться операции чтения времени, которых не было в исходном потоке событий. Для того, чтобы они завершались успешно, при воспроизведении журнала выполняется кэширование показаний часов. Таким образом, в цикле ожидания будут считываться одни и те же показания, пока выполнение симулируемого кода не продолжится.

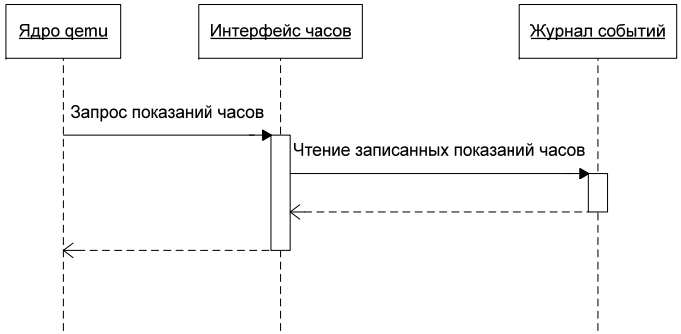

Рис. 4. Чтение показаний часов из журнала

3.2.3. Прерывание выполнения симулируемого кода с помощью таймера

Для решения проблемы с синхронизацией потоков внутри Qemu, было решено зафиксировать точки в коде основного потока, в которых он может взаимодействовать с прерывающим потоком. Таким образом, в процессе воспроизведения журнала событий появится возможность точно восстанавливать моменты этих взаимодействий.

Несколько точек возможных прерываний были размещены в циклах ожидания и выполнения, а также был модифицирован код транслятора для добавления таких точек перед выполнением каждой инструкции. Таким образом становится возможным детерминированное воспроизведение поведения системы даже при пошаговом выполнении кода.

3.2.4. События от периферии

Запись и воспроизведение событий от мыши, клавиатуры и сетевой карты отличается от предыдущих видов событий тем, что они инициируются извне. Поэтому запись информации о них в журнал производится непосредственно в момент возникновения соответствующего события.

Так как основные действия в Qemu, а также запись в журнал, выполняются в одном потоке, сообщения от периферийных устройств будут обрабатываться строго между какими-либо другими событиями, не пересекаясь с ними по времени. Поэтому при воспроизведении журнала (при поиске очередного синхронного события) может встретиться такое асинхронное событие (например, нажатие клавиши), которое может быть тут же обработано.

3.3. Детерминированное воспроизведение журнала событий

Технология детерминированного воспроизведения журнала событий заключается в выполнении кода симулятора и считывании необходимых для воспроизведения программы данных из журнала:

- Считывание показаний часов из журнала, когда их запрашивает какой-либо код. Если соответствующих показаний нет – возвращается кэшированное значение.
- Передача сообщений от клавиатуры, мыши, звуковой и сетевой карт в тот момент, когда они считываются из журнала в соответствующие обработчики.
- Обновление внутреннего счетчика команд, когда происходит выполнение очередной инструкции.

Чтобы воспроизведение журнала было детерминированным, оно должно начинаться с того же самого состояния симулируемой системы, что и запись. Состояние системы включает в себя состояния всех симулируемых устройств, включая образы используемых дисков.

Остальные устройства инициализируется при старте симулятора и, поэтому, их состояния не отличаются при записи и воспроизведении.

3.4. Фрагмент журнала событий

События в журнал записываются последовательно одно за другим. Какой-либо индексации не производится, так как для целей воспроизведения журнала необходимо лишь последовательное чтение и исполнение недетерминированных событий, записанных в журнал.

Ниже приведен пример нескольких событий, записываемых с начала журнала:

1. Системное время
2. Часы 1
3. Часы 0
4. Часы 2

5. Таймер
6. Цикл ожидания 1
7. Цикл ожидания 0
8. Цикл ожидания 2
9. Часы 2
10. Часы 1
11. Цикл ожидания 1
12. Таймер
13. Выполнение инструкции
14. Цикл ожидания 0
15. Цикл ожидания 2
16. Часы 2
17. Часы 1
18. Цикл ожидания 1
19. Выполнение инструкции
20. Выполнение инструкции
21. Выполнение инструкции
22. Выполнение инструкции
23. Выполнение инструкции

В данный фрагмент журнала попали несколько показаний различных часов, а также значения системного времени. Кроме того, происходило выполнение цикла ожидания. В нем расставлено 3 контрольных точки, позволяющих управлять воспроизведением фрагментов цикла (инициирование прерываний и обработка внешних событий) в том же порядке, в котором они выполнялись при записи.

Также в журнале присутствует несколько событий «выполнение инструкции». Здесь они показаны по отдельности, однако запись их в журнал производится группой – одна запись соответствует целой последовательности выполненных инструкций.

3.5. Контроль корректности воспроизведения журнала

Для того чтобы контролировать, является ли процесс воспроизведения детерминированным, при считывании событий из файла журнала проверяется следующее ограничение: в момент, когда ожидается наступление события «выполнение инструкции» не должно возникать ни события от часов, ни события «цикл ожидания».

Если это ограничение нарушается, это означает, что ход симуляции был нарушен. В этом случае выдается сообщение об ошибке и процесс воспроизведения останавливается. Это позволяет контролировать отсутствие

ошибок в механизме воспроизведения и выполнять отладку при их возникновении.

3.6. Замедление работы симулятора при записи журнала событий

Так как при симуляции с включенной записью (и воспроизведением) журнала выполняются дополнительные действия (например, запись в журнал) по сравнению с нормальным режимом работы, происходит уменьшение скорости симуляции.

Для измерения степени замедления рассматривался сценарий загрузки Windows XP до появления графического интерфейса.

- Загрузка Windows XP без записи журнала – 80 секунд.
- Загрузка Windows XP с записью журнала – 520 секунд.
- Загрузка Windows XP при воспроизведении журнала – 606 секунд.

Такое замедление (менее чем в 10 раз) не является критичным для большинства приложений, что позволяет использовать для них данный механизм.

Объем файла с записанным журналом этого сценария составил 18 Мегабайт.

4. Детерминированная отладка с использованием журнала событий

Детерминированная отладка – это способ поиска ошибок в недетерминированных приложениях, при котором недетерминированность устраняется с помощью записи сценария работы системы (или программы) в журнал. Разработанная технология позволяет выполнять детерминированную отладку недетерминированных приложений следующим образом:

1. Тестировщик записывает сценарий при выполнении которого проявляется дефект в тестируемой программе в журнал, а затем передает этот журнал вместе с образами дисков системы разработчику. Здесь сценарий выступает не только в роли исходных данных для отладки, но и в роли описания способа воспроизведения дефекта.
2. Разработчик может неоднократно проигрывать полученный сценарий в симуляторе, анализируя причины появления дефекта. Разработчику не нужно настраивать сложное окружение системы так же, как это делал тестировщик, так как все особенности взаимодействия с этим окружением уже записаны в журнал.

Таким образом, отладка недетерминированных приложений с применением разработанной технологии становится детерминированной, что позволяет

сократить время, затрачиваемое разработчиками на локализацию дефектов в программе, а тестировщиками – на описание процесса их воспроизведения.

5. Заключение

Таким образом, были получены следующие результаты:

1. Разработана технология детерминированного воспроизведения, реализованы механизмы записи и воспроизведения журнала недетерминированных событий в симуляторе Qemu.
2. Данная технология была реализована для платформ x86 и x86_64, что позволяет выполнять отладку приложений, работающих под ОС для этих платформ (в частности, Windows и Linux), а также вновь создаваемых операционных систем.
3. Запись событий производится для системы целиком, включая пользовательские процессы и внутренние механизмы операционной системы, установленной в виртуальной машине, что позволяет учесть все межпроцессные взаимодействия.
4. В журнал событий записываются сетевые пакеты, полученные через внешние интерфейсы. Таким образом, можно анализировать и отлаживать приложения, работающие с сетью.
5. Реализована возможность многократной отладки сложных сценариев воспроизведения ошибок с помощью их детерминированного воспроизведения.
6. Также технология предоставляет возможности для реализации других механизмов динамического анализа программ (в частности, снятия трассы их выполнения для последующего анализа) и реверсивной отладки.
7. Детерминированное воспроизведение было протестировано как для изолированной виртуальной машины, так и работающей в сетевом окружении. При тестировании использовались виртуальные машины с установленными Windows XP и Linux. Для большинства сетевых приложений вносимое механизмом сохранения журнала замедление является незначительным, поэтому разработанная технология может быть использована для отладки и анализа таких приложений.

Дальнейшее развитие технологии детерминированного воспроизведения планируется в следующих направлениях:

1. Реализация технологии детерминированного воспроизведения для платформ, отличных от i386 (в частности, ARM), что позволит выполнять детерминированное воспроизведение сценариев работы, записанных на симуляторах мобильных устройств.
2. Одно из других важных направлений – разработка и реализация технологии, позволяющей выполнять реверсивную отладку

приложений в симуляторе с использованием детерминированного воспроизведения недетерминированных событий. Средства реверсивной отладки позволяют отладчику возвращаться от места проявления ошибки к месту в программе, где эта ошибка возникает на самом деле (например, запись некорректного значения в переменную или чтение неинициализированной ячейки памяти).
3. Также будет проводиться работа как по уменьшению объема журнала событий, так и по уменьшению накладных расходов при его создании, так как для некоторых высокопроизводительных сетевых приложений создаваемое замедление оказывается критичным.
4. Четвертым направлением является расширение поддержки внешних устройств при записи журнала. В частности, работа с внешними устройствами, подключаемыми через USB.

Список литературы

[1] J. Choi and H. Srinivasan, “Deterministic replay of java multithreaded applications”, In Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools, Aug. 1998, pages 48-59.

[2] H. Patil, C. Pereira, M. Stallcup, G. Lueck, J. Cownie, “PinPlay: a framework for deterministic replay and reproducible analysis of parallel programs”, Proceedings of the 8th annual IEEEACM international symposium on Code generation and optimization (2010), pages 2-11.

[3] S. M. Srinivasan, S. Kandula, C. R. Andrews, and Y. Zhou, “Flashback: a lightweight extension for rollback and deterministic replay for software debugging”, USENIX 2004 Annual Technical Conference, Pp. 29–44 of the Proceedings

[4] ReplayEngine. Record and replay debugging race conditions and deadlocks in Linux applications, 2011. http://www.roguewave.com/products/totalview-family/replayengine.aspx

[5] Better Software Development with Replay Debugging, 2009. http://www.replaydebugging.com/