УДК 004.416.2

Применение рекурсивного спуска для анализа машинного кода в полноплатформенном симуляторе

Р.С. Проскин, Г.С. Речистов

Московский физико-технический институт (государственный университет)
АО «Интел А/О»

В процессе разработки программного обеспечения нередко приходится работать с исполняемыми файлами, исходный код которых недоступен или не существует. Это могут быть коммерческие приложения с закрытым кодом, или устаревшие версии программ, отдельные элементы которых необходимо воспроизвести. При работе с функциональными симуляторами гораздо чаще сталкиваются с отсутствием отладочной информации или исходного кода. В таком случае пользуются различными методами анализа машинного кода. Поскольку симуляция значительно ускоряет время выхода товара на рынок, наличие полноценного, удобного в использовании отладочного инструмента в составе симулятора является необходимым средством для успешной разработки программного продукта.

Алгоритмы обратной разработки машинного кода находят своё применение в различных приложениях (рис. 1): статические дизассемблеры (objdump [1], Radare2 [2], IDA [1]), динамические отладчики (gdb [3], OllyDbg [4]). Перечисленные программы используются под управлением операционной системы на реальном оборудовании. Однако аналогичного результата можно добиться и в симуляторах, где основным подводным камнем анализа машинного кода становятся всевозможные процессы, происходящие в симулируемой платформе. К ним можно отнести обработку исключений, наличие страничной системы памяти и необходимость трансляции виртуальных адресов, код гостевой операционной системы, а также присутствие инструкций дальних (в том числе и межстраничных) переходов. К тому же не стоит забывать, что платформа может содержать различное количество процессоров, которые, в свою очередь, могут работать в различных режимах: то есть формат используемых машинных команд может отличаться.

Классическим методом статического анализа кода является линейная развёртка [1]. Подобным механизмом, выдающим ассемблерный код по стандарту архитектуры Intel [5], обладает симулятор Wind River® Simics [6] (далее – Simics). Основным недостатком данного подхода является сложность восприятия полученного результата в связи с описанными выше причинами. Разработчику необходимо потратить значительное количество времени на определение подлежащего отладке участка кода, отделение его от подпрограмм операционной системы и обработчика исключений. Даже при возможности непосредственного доступа ко всему состоянию платформы это представляется тяжёлым испытанием.

В отличие от линейной развёртки, дизассемблирование с помощью рекурсивного спуска [1] основывается на анализе потока управления [7]: в ходе анализа машинного кода обрабатываются лишь те сегменты кода, на которые ссылаются предыдущие инструкции. Используя данный принцип, можно не только улучшить точность разбора кода, но и кардинально облегчить процесс отладки в полноплатформенном симуляторе путём визуализации переходов между соседними блоками машинных инструкций.

Поскольку обозначенная на рис. 1 ниша остаётся свободной, в настоящей работе, как отдельный модуль Simics, был разработан прототип инструментария для анализа машинного кода без отладочной информации, основанный на методе рекурсивного спуска. После первоначального анализа данные об инструкциях помещаются в базу данных. Контроллер приложения следит за изменениями в оперативной памяти и поддерживает состояние базы актуальным, по мере надобности снова вызывая дизассемблер. Представление данных происходит в текстовом псевдографическом режиме. Также возможно отображение результата в графическом виде с помощью элементарных базовых блоков и переходов между ними. Схема анализирующего модуля представлена на рис. 2.

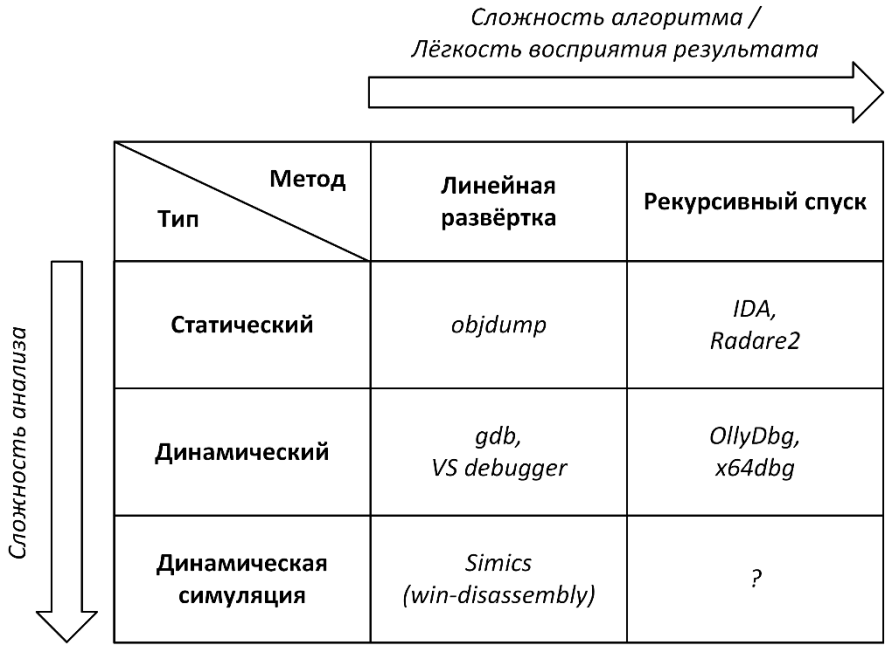

Рис. 1. Классификация приложений для обратной разработки в зависимости от метода анализа

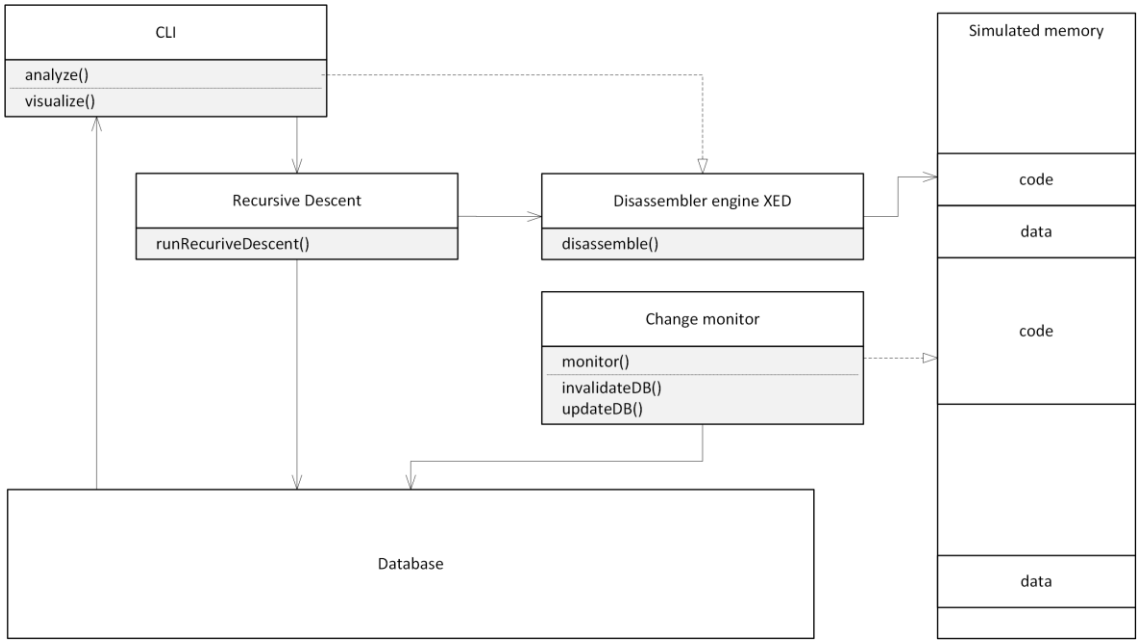

Рис. 2. Схема разработанного инструмента для отладки машинного кода в симуляторе Simics

Литература

1. *Eagle C.* The IDA Pro Book, 2nd edition. – San Francisco, CA, USA: No Starch Press, Inc., 2011. 672 pp.
2. Radare 2 Book / ed. by maijin [et al.]: GitBook. URL: https://radare.gitbooks.io/radare2book/content/
3. *Free Software Foundation.* The GNU Project Debugger. URL: https://www.gnu.org/software/gdb/
4. *Oleh Yuschuk*. OllyDbg. URL: http://www.ollydbg.de/
5. *Intel Corporation*. Intel®64 and IA-32 Architectures Software Developer's Manual. V. 2A – 2C, 3A – 3D.
6. *Peter S. Magnusson, Magnus Christensson, Jesper Eskilson [et al.]* Simics: A Full System Simulation Platform // Computer. 2002. V. 35. P. 50-58. URL: http://dl.acm.org/citation.cfm?id=619072.621909
7. *Muchnick S.S.* Advanced Compiler Design & Implementation. – San Francisco, CA, USA: Morgan Kaufmann Publishers, 1997. 856 pp.