Лабораторный практикум по программному моделированию

Учебное пособие

Версия 1.6β
25 августа 2014 г.

Содержание

Предисловие

В настоящем практикуме описываются лабораторные и практические работы по курсу «Основы программного моделирования ЭВМ», проводящиеся в Московском физико-техническом институте. Эта книга дополняет практическими аспектами программирования и использования технологий теоретические основы, изложенные в учебнике:

Основы программного моделирования ЭВМ: Учебное пособие / Г. Речистов, Е. Юлюгин, А. Иванов, П. Шишпор, Н. Щелкунов, Д. Гаврилов. — 2-е изд., испр. и доп. — Издательство МФТИ, окт. 2013. — ISBN 978-5-7417-0444-8.

Авторы прикладывают усилия для того, чтобы поддерживать все учебные материалы в актуальном состоянии. Самую свежую версию данного документа вы можете получить на сайте `http://atakua.doesntexist.org/wordpress/simulation-course-russian/`.

Если вы обнаружили опечатку, стилистическую, фактическую ошибку, которые, более чем вероятно, встречаются в тексте, имеете замечания по содержанию или предложения по тому, как можно улучшить данный материал, то просим сообщить об этом по e-mail `grigory.rechistov@phystech.edu` — нам очень важно ваше мнение!

Отметим, что текст данной работы постоянно обновляется, и поэтому в версиях, имеющих в своём номере пометку «бета» (β), могут присутствовать незаконченные места, которые обозначаются символом TODO .

Введение

> Essentially, all models are wrong, but some are useful[1].
>
> *(George E. P. Box, Norman R. Draper. Empirical Model-Building and Response Surfaces)*

1. Компьютерная симуляция

Использование компьютерного моделирования в процессе проектирования цифровых систем позволяет заметно сократить время, проходящее от момента предложения концепции новой системы до поступления на рынок первых образцов готовой продукции. Это происходит благодаря т. н. «сдвигу влево» (*англ.* shift left) всей существующей методологии создания продукции, что позволяет выполнять ключевые процессы параллельно во времени, и значительноая часть из них может быть начата гораздо раньше, чем это было возможно ранее. Всё это эффективно сокращает длину цикла разработки нового продукта и увеличивает его конкурентноспособность (рис. 1).

Программное обеспечение для имитационного моделирования используется для тестирования функциональности, исследования производительности, оценки энергопотребления и иных свойств вычислительных систем на стадиях их раннего проектирования, когда реальные образцы соответствующей аппаратуры ещё недоступны. Кроме того, оно позволяет писать приложения для таких систем заранее и выпускать аппаратуру, готовую для использования конечным потребителем, не ожидая, пока все необходимые программы будут адаптированы.

[1] В сущности, все модели неправильны; но некоторые из них приносят пользу.

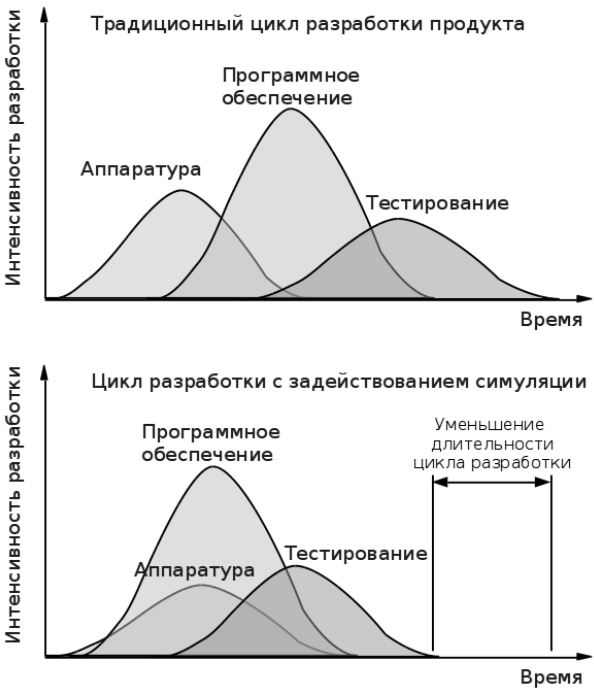

Рис. 1. Сдвиг влево — возможность совместить моменты начала отдельных стадий проектирования новых цифровых систем, таким образом сокращая цикл разработки и уменьшая время вывода их на рынок

Задача цикла лабораторных и практических работ, описанных в это книге, — познакомить слушателей с новейшими достижениями в области компьютерной симуляции, связанными с эффективным созданием моделей, максимально точно представляющих аппаратные средства и при этом имеющих высокую скорость работы, получаемую благодаря эффективному задействованию имеющихся вычислительных ресурсов. Изучение проводится на программном продукте Wind River® Simics (в дальнейшем сокращённо называемого Simics), который в настоящее время является одним из самых современных инструментов разработки, тестирования и исследования цифровых компьютерных систем

и используется как в промышленности, так и в научной среде. Несмотря на это, все рассматриваемые в книге вопросы основаны на концепциях, общих для многих других программных симуляторов, как коммерческих, так и исследовательских. В приложениях в конце книги дана информация о том, как подготовить компьютерный класс для выполнения лабораторных работ на Simics.

Для максимально эффективного усвоения материала данного пособия читателю рекомендуется иметь начальные знания по архитектуре ЭВМ. Желательно иметь понимание общих принципов работы операционных систем, а также знакомство с языками программирования высокого уровня и ассемблера. Базовой операционной системой для запуска приложений в практических работах служит GNU/Linux; для более чёткого понимания используемых в работах операций читатель должен быть знаком с работой интерпретатора командной строки Unix.

2. Обозначения

При первом использовании в тексте терминов, заимствованных из английского языка и не имеющих известных авторам общепринятых переводов на русский язык, в скобках после них будут указываться оригинальные выражения.

Всюду в тексте данной работы будут использованы следующие шрифтовые выделения и обозначения.

- Обычный текст используется для основного материала.
- `Моноширинный текст` вводится для исходных текстов программ на различных (псевдо) языках программирования и их ключевых слов, имён регистров устройств, листингов машинного кода, результатов работы операторов командной строки.
- *Курсивный текст* используется для выделения новых понятий.
- **Полужирный текст** используется для обозначения элементов графического интерфейса: имён окон, пунктов меню и т.п.

- Числа в шестнадцатеричной системе счисления имеют префикс **0x** (например, 0x12345abcd), в двоичной системе счисления — суффикс **b** (например, 10010011b).
- Команды, которые необходимо вводить в строку приглашения запущенного Simics, имеют префикс `simics>`:

  ```
  simics> list-objects
  ```

  Ответный вывод команд, если он есть, приводится без каких-либо предваряющих префиксов. Если вывод очень длинен, то часть его заменяется многоточием.
- Команды, которые необходимо вводить в строку приглашения интерпретатора (в данной книге используется стандартный `/bin/sh`), имеют префикс `$` для обычного пользователя или `#` для команд, выполняемых суперпользователем root:

  ```
  $ ./simics targets/x86-x58-ich10/viper.simics
  # mount /dev/sdb /mnt/disk
  ```

- При описании синтаксиса команд их обязательные аргументы команд указываются в угловых скобках, необязательные — в квадратных:

  ```
  $ command <mandatory argument> [optional argument]
  ```

  Если команда принимает несколько однотипных аргументов подряд, спользуется многоточие `...` для второго и последующих параметров.
- Имена функций вне блоков кода записываются со скобками в конце имени, например: `main()`, `printf()`. Это сделано для того, чтобы отличать их от имён переменных.

1. Первое знакомство с Simics

1.1. Цель занятия

Ознакомиться с базовыми операциями, которые можно выполнять в рабочем окружении (*англ.* workspace) симулятора Simics.

- Запуск симулятора. Элементы его интерфейса.
- Сценарии работы. Гостевые системы.
- Процесс загрузки гостевой операционной системы внутри симулятора.
- Базовые операции: остановка модели, сохранение и восстановление симуляции с диска и на диск.

1.2. Запуск симулятора Simics. Элементы интерфейса

Для запуска Simics на физическом *хозяйском* компьютере архитектуры IA-32 должны быть установлены следующие программы.

- Вариант операционной системы GNU/Linux. Simics может работать практически во всех современных дистрибутивах, включая Debian, Ubuntu, Fedora, OpenSUSE и др. Настоятельно рекомендуются 64-битные варианты ОС.
- Графическая оболочка, любая из поддерживаемых выбранным дистрибутивом: KDE, Gnome, Fluxbox и др. Несмотря на то, что Simics может работать из чистой командной строки, далее всюду в тексте книги будет подразумеваться, что работа ведётся в графическом режиме.

- Собственно копия Simics. По-умолчанию программа устанавливается в подкаталог `/opt/simics/simics-4.6`, который будет использоваться всюду в тексте.

Подробнее об аппаратных требованиях к используемым компьютерам сказано в [2]. Особенности централизованной установки Simics в компьютерном классе описаны в приложении D.

1.2.1. Создание Simics workspace

Одна инсталляция Simics может быть использована совместно множеством пользователей; по этой причине её файлы должны оставаться неизменными. Рабочее окружение (*англ.* «workspace»[1]) — это персональная «копия» общей установки, в которой Simics хранит ваши собственные настройки симуляционной среды, сценарии для конфигурации гостевых систем, двоичный и исходный код моделей и прочие данные. Каждый пользователь может иметь несколько независимых workspace, все они могут использоваться одновременно и независимо друг от друга и содержать различные окружения.

Создание workspace

Для создания нового workspace необходимо выполнить следующую последовательность действий.

1. Создайте директорию, в которой будет находиться новый workspace (рекомендуется использовать локации внутри домашней директории `$HOME`) и перейдите в неё:

   ```
   $ mkdir <folder_name>
   $ cd <folder_name>
   ```

2. Вызовите программу workspace-setup из инсталляции Simics. Учтите, что путь к инсталляции и версия Simics может отличаться.

[1]Начиная с Simics версии 4.8 название workspace было сменено на project, однако мы будем использовать традиционную терминологию.

```
$ /opt/simics/simics-4.6/simics-4.6.32/bin/workspace-
    setup
Workspace created successfully
```

Новый workspace создан! Текущая директория теперь содержит несколько файлов и поддиректорий:

```
$ ls -1
bin
compiler.mk
config.mk
doc-cache
GNUmakefile
modules
simics
simics-gui
targets
```

1.2.2. Запуск Simics

После создания workspace вы можете начинать работу с Simics. Запустить симулятор вы можете с помощью команды:

```
./simics
```

Должно открыться окно управления **Simics Control**[1] (рис. 1.1). Оно включает в себя иконки панели инструментов и меню, позволяющее контролировать состояния симулятора и текущей гостевой системы. Для выполнения данных лабораторных работ нам понадобится дополнительное окно с командной строкой **Simics Command Line** (рис. 1.2). Выберите **Tools** → **Command Line Window** для того, чтобы открыть его.

Simics Command Line позволяет вам управлять симуляцией с помощью ввода текстовых команд. Во время своей работы Simics будет выводить в него диагностическую информацию: события симулятора и моделей, сообщения об ошибках и прочее. Всё, что

[1]Последние версии Simics используют в качестве базового графического окружения среду Eclipse [1]. Поддержка предыдущего, классического интерфейса была сохранена в полной мере. В этой книге мы всюду будем использовать только его.

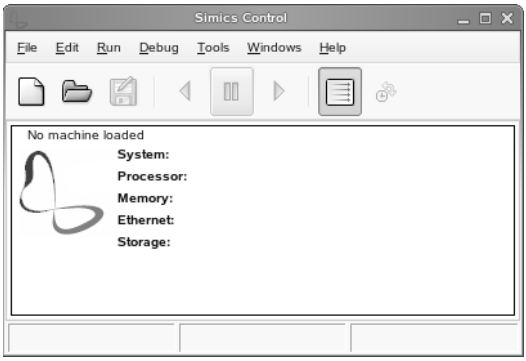

Рис. 1.1. Окно Simics Control

можно сделать с помощью окна **Simics Control**, вы также можете сделать и с помощью командной строки. Большинство действий в дальнейшем будет производиться именно в ней.

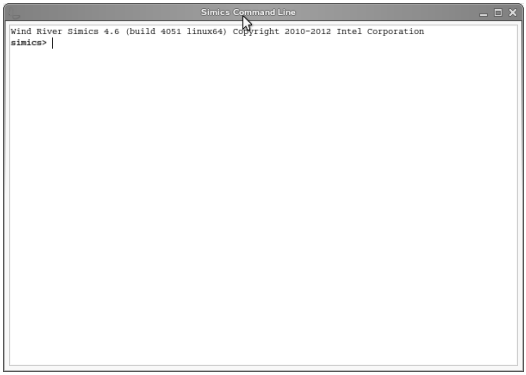

Рис. 1.2. Командная строка Simcs

В данной главе в качестве моделируемой — *гостевой* — используется вычислительная платформа, основанная на процессоре Intel® Core™ i7. GNU/Linux используется в качестве операционной системы. В качестве системной среды пользователя выступает пакет под названием BusyBox [3], часто используемый во встраиваемых (*англ.* embedded) системах.

Для загрузки конфигурации модели воспользуйтесь окном **Simics Command Line** и введите команду:

```
simics> run-command-file targets/x86-x58-ich10/viper-
    firststeps.simics
```

То же самое можно было сделать с помощью окна **Simics Control**, выбрав **File** → **New session from script** и открыв файл `viper-firststeps.simics`, лежащий в директории `x86-x58-ich10`.

Автодополнение команд. Если нажать несколько раз клавишу **Tab** при неполностью набранной строке команд, Simics автоматически дополнит её или предложит допустимые варианты для её завершения. Используйте это полезное свойство для ускорения набора и изучения списка понимаемых системой команд.

Спустя некоторое время окно **Simics Control** покажет суммарную информацию о симулируемой системе (рис. 1.3): тип процессора, объём памяти, диска. Также должно открыться новое окно **Serial Console on viper.mb.sb.com[0]** (рис. 1.4).

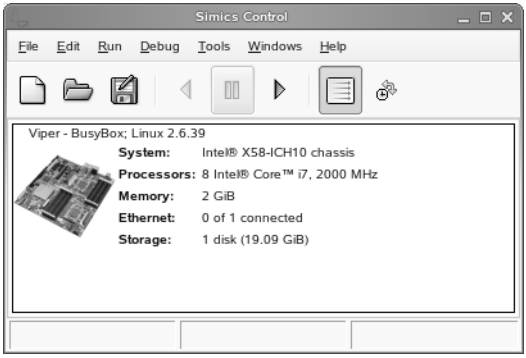

Рис. 1.3. Окно Simics Control после загрузки модели

Это новое окно является частью симуляции. Оно соединено с последовательным портом моделируемой материнской платы. Вывод сообщений гостевого ПО будет отображаться в нём. Кроме того, через него вы сможете взаимодействовать с моделиру-

Рис. 1.4. Окно **Serial Console on viper.mb.sb.com[0]**

емым программным обеспечением, тогда как **Simics Command Line** используется для управления симуляцией.

1.2.3. Запуск моделирования

Теперь Simics ожидает ваших команд, вводимых через панель инструментов или через командную строку. Вы можете запустить симуляцию с помощью нажатия кнопки ▷ **Run Forward** на панели инструментов или ввода команды `continue` в командной строке.

Рассмотрим следующий пример:

```
simics> continue
```

Для паузы симуляции в любой момент наберите команду `stop`.

```
running> stop
[viper.mb.cpu0.core[0][0]] cs:0xffffffff8100944d p:0
    x00100944d  MWait state
simics>
```

После запуска симуляции виртуальное время (указываемое в нижней части **Simics Control**) начинает увеличиваться, инструкции гостевого процессора исполняются, и сообщения от программного обеспечения выводятся в окно консоли. Если вы позволите модели поработать некоторое время, то увидите сообщения от загрузки гостевой ОС в окне **Serial Console on viper.mb.sb.com[0]** (рис. 1.5). Симуляция может быть остановлена с помощью команды `stop` или нажатия **Stop** на панели инструментов.

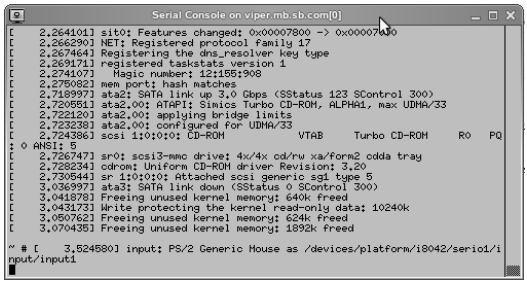

Рис. 1.5. Загруженная гостевая операционная система Linux

После полной загрузки ОС в гостевой консоли будет выведено приглашение для логина пользователя:

```
busybox login:
```

Введите в него `root`, нажмите **Enter**. Теперь гостевая операционная система загружена, и вы можете начать взаимодействовать с ней — вводить инструкции в командной строке.

Получить листинг корневой директории. Пример вывода команды `ls`:

```
# ls /
bin      etc      host     linuxrc  root     sys
dev      home     init     proc     sbin     var
```

Увидеть информацию о моделируемом процессоре. Пример содержимого псевдофайла `/proc/cpuinfo` гостевой системы:

```
~ # cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 26
model name      : Intel(R) Core(TM) i7 CPU     @ 2.00GHz
stepping        : 8
cpu MHz         : 1999.991
cache size      : 8192 KB
fpu             : yes
fpu_exception   : yes
```

```
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic
    sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
    fxsr sse sse2 ss tm pbe syscall nx rdtscp lm
    constant_tsc up arch_perfmon pebs bts rep_good nopl
    xtopology nonstop_tsc aperf mperf pni dtes64 monitor
    ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2
    popcnt lahf_lm ida dts tpr_shadow ept
bogomips        : 3999.98
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Для того чтобы закончить работу симулятора, необходимо выполнить команду `quit`. Более подробно о командах Simics рассказывается в последующих главах, а также в приложении А.

1.3. Сохранение и восстановление состояния симуляции

Так как в любой момент исполнения симуляции известны состояния всех моделируемых внутри неё устройств, имеется возможность записать их в постоянное хранилище, например, в файл. В дальнейшем эту точку сохранения (*англ.* checkpoint) можно загрузить и продолжить симуляцию с той точки, в которой она была сохранена.

В Simics для создания точки сохранения используется команда `write-configuration <name>`.

- Запустите симуляцию. Позвольте ей исполняться в течение нескольких виртуальных секунд:

```
$ ./simics targets/x86-x58-ich10/viper-firststeps.
    simics
simics> continue-seconds 3
simcis> write-configuration new-chkpt
```

- Завершите сессию симуляции с помощью команды `quit`.

- Проверьте, что в текущем workspace была создана новая директория с именем `new-chkpt`.

```
$ ls -1
bin
compiler.mk
config.mk
doc-cache
GNUmakefile
modules
new-chkpt
simics
simics-gui
targets
```

  В ней сохранено состояние всей симуляции на момент выполнения команды сохранения.
- Запустите пустую сессию Simics, без загрузки каких-либо сценариев:

```
$ ./simics
```

- Загрузите точку сохранения с помощью команды `read-configuration`:

```
simcis> read-configuration new-chkpt
```

  В результате состояния симуляции будет эквивалентно тому, что она имела на момент создания точки.
- Продолжите исполнение симуляции с помощью `continue` и убедитесь, что загрузка системы продолжается корректно.

1.4. Задания и контрольные вопросы к главе

1. Определите все отличия в параметрах аппаратных средств гостевой и хозяйской систем, используемых в данной работе. Какие средства могут быть использованы для обнаружения различий?
2. Сравните скорость течения виртуального времени, сообщаемого симуляцией, и настоящего времени. Чем, по-вашему, вызваны наблюдаемые различия?

3. Попытайтесь удалить *все* файлы внутри симуляции. Каким образом это действие скажется на хозяйской системе? Перезапустите симуляцию. Что произошло с удалёнными файлами?
4. Придумайте сценарии применения точек сохранения симуляции а практике разработки ПО и аппаратных средств с помощью симуляторов.

Список литературы к занятию

1. Eclipse - The Eclipse Foundation open source community website. — Eclipse Foundation, Inc., 2014. — URL: `http://eclipse.org/home/`.
2. Simics Installation User Guide 4.8 / Wind River. — 2013.
3. Официальный сайт BusyBox. — URL: `http://www.busybox.net/`.

2. Концепции симуляции

2.1. Цель занятия

В данной лабораторной работе продолжается изучение принципов работы симулятора Simics. В ней будут определены основные понятия, используемые при работе с симуляцией и при написании моделей устройств.

Запустите симулятор со сценарием моделируемой системы `viper-busybox`:

```
$ ./simics -e '$cpu_class=core-i7-single' targets/x86-x58-
    ich10/viper-busybox.simics
```

2.1.1. Единицы симуляции

Аналогично тому, что физические системы собираются из устройств, программные платформы состоят из моделей отдельных узлов. При этом, как и в реальности, они образуют иерархическую структуру. Выбираемый уровень детализации зависит от целей симуляции и от возможностей симулятора.

Рассматриваемая в качестве примера в данной работе модель системы Viper соответствует физической системе, основанной на материнской плате формата Intel X58, набора системной логики ICH10 и центрального процессора Intel Core i7, кодовое имя микроархитектуры Nehalem. Вернеуровневая (т.н. компонентная) структура модели приведена на рис. 2.1. Порядок соединения отдельных устройств друг с другом именуется *конфигурацией* гостевой системы.

2.2. Устройства и классы Simics

Базовая единица симуляции в Simics — это объект. Объекты одного типа образуют класс. Классы в Simics имеют уникальные имена; встроенная справка по имени класса выдаёт всю информацию о свойствах объектов. Самые часто встречающие классы в Simics — это `memory-space`, используемый для организации доступа к периферийным устройствам, размещённым в пространстве физической памяти, `ram` — для моделирования ОЗУ и ПЗУ, и `image` — для представления всех сущностей, хранящих группы адресуемых байт: ОЗУ, диски, кэши и т.п.

Самое важное назначение объектов Simics состоит в том, что большинство из них являются устройствами, эволюция которых во времени наблюдается при симуляции. Согласно принципам объектно-ориентированного программирования у них есть методы (группируемые в *интерфейсы*) и поля (называемые в Simics *атрибутами*).

2.3. Компоненты

Для удобства создания больших конфигураций виртуальных систем некоторые объекты, называемые *компонентами*, выполняют лишь ограниченную связующую роль и практически не изменяются после того, как все участвующие в симуляции устройства соединены друг с другом. Задача компонент — обеспечить на этапе создания конфигурации проверку правильности соединения устройств друг с другом. Например, очень часто используются две следующих компоненты: процессорный блок (*англ.* package) и разъём на материнской плате для него (*англ.* socket). Процессорный блок может содержать на себе несколько устройств-ядер процессора, каждое из которых надо соединить с ОЗУ и другими устройствами, находящимися в составе модели материнской платы. Модель разъёма на плате устроена так, что автоматически создаёт все необходимые соединения. В случае несовместимости компонент симулятор сообщает об ошибке,

предотвращая ситуации, в которых конфигурация была создана лишь частично.

Отличие объектов-компонент от объектов-устройств состоит в том, что они очень редко модифицируются в ходе симуляции. Фактически они нужны только один раз, на этапе её создания. Их код необязательно должен быть быстрым — достаточно обеспечить корректность. Поэтому практически всегда компоненты Simics пишутся на языке Python, процедуры которого, в отличие от языков, используемых для написания устройств — Си, C++, DML, — работают медленно. Однако скорость разработки компонент на Python ввиду его динамической природы значительно выше, а скорость финального кода практически не влияет на работу модели в целом.

2.3.1. Иерархия устройств

Корневой элемент создаваемой модели компьютера имеет имя, традиционно хранящееся сценариях конфигураций в переменной `$system`. В нашем случае она содержит строку `''viper''`. Это имя используется для именования компонента шасси, на котором «крепятся» все остальные устройства. Это проявляется в том, что иерархические имена подкомпонент образованы присоединением к имени надкомпоненты своего имени через символ точки. Так, материнская плата данной системы имеет имя `viper.mb`, первый процессор на ней — `viper.mb.cpu0`, а первое ядро в этом процессоре — `viper.mb.cpu0.core[0][0]`. Следует отметить, что подобная иерархическая организация имён диктуется именно использованием механизма компонент; если устройства соединять «вручную», то их имена могут и не образовывать иерархии.

Для просмотра списка всех устройств используйте команду `list-objects -recursive`. Альтернативно можно использовать окно **Object Browser** (рис. 2.2).

2.4. Команды

Каждый класс моделей может предоставлять несколько команд, которые позволяют инспектировать или изменять состояние объектов данного класса. Самые часто реализуемые внутри классов команды — это `status` и `info`. Для того, чтобы вызвать команду для некоторого устройства, следует указать её имя после имени устройства и символа «точка». Например, все устройства модели памяти имеют команды `get` и `set`, которые позволяют читать и записывать данные в симулируемую память:

```
simics> viper.mb.cpu0.mem[0][0].get address = 0xffff0000
0x00
simics> viper.mb.cpu0.mem[0][0].set address = 0xffff0000
    value = 0xaabbccdd
```

Аргументы любой команды можно узнать в справке к ней (команда `help <устройство>.<команда>`) или с помощью автодополнения (нажав клавишу **Tab**).

2.4.1. Глобальные команды

Кроме команд, специфичных для устройств, существуют так называемые глобальные команды, эффект которых состоит или в доступе к состоянию всей симуляции в целом, или же к текущему «устройству» некоторого класса, что позволяет сэкономить время на наборе иерархических имён. Например, команда `ptime` выводит значения симулируемого времени для текущего процессора, тогда как `ptime -all` позволяет увидеть аналогичную информацию о всех процессорах в системе.

```
simics> ptime
processor                  steps   cycles   time[s]
viper.mb.cpu0.core[0][0]       0        0         0
```

2.5. Диагностические сообщения

Одно из назначений симуляторов — помогать в разработке аппаратуры и программ для неё. При этом разработчикам необходимо понимать, какие события в каком порядке происходят во

время симуляции и контролировать их корректность. Для этого все объекты Simics могут выводить диагностические сообщения в консоль управления Simics или в файл.

2.5.1. Формат сообщений

Пример сообщений при загрузке некоторой гостевой системы.

```
simics>
[viper.mb.gpu.vga info] writing character   (0x20) at (30,
    203) Attrib 0x7
[viper.mb.nb.core_scratch_gpio spec-viol] 4 byte read access
     at offset 0x100 in pci_config (addr 0xa1100) outside
    registers or misaligned access
[viper.mb.nb.core_control_status spec-viol] 4 byte read
    access at offset 0x100 in pci_config (addr 0xa2100)
    outside registers or misaligned access
[viper.mb.conf unimpl] QEMU_CFG_SMBIOS_ENTRIES unimplemented
[viper.mb.sb.ehci[1] spec-viol] Write to read-only field
    usb_regs.usbsts.hchalted (value written = 0, contents =
    0x1).
```

В целом формат напоминает текст пьесы, в которой участвующие в ней персонажи по очереди произносят свои реплики. Формат сообщения по умолчанию имеет вид `[device] type message`[1]. Здесь

- `device` — имя устройства;
- `type` — одна из следующих строк: «info» для информационных сообщений, «unimpl» для обозначения неполной точности в поведении модели, «spec-viol» для ситуаций, в которых моделируемое устройство вводится в режим, противоречащий спецификации модели, «undef» — ситуации неопределённого поведения устройств, и «error» — для обозначения ошибок в работе модели или симуляции в целом, после которых правильность работы не гарантируется;
- `message` — собственно текст сообщения.

[1] С помощью команды `log-setup` в него можно добавить вывод значения виртуального времени на момент выдачи сообщения.

2.5.2. Фильтрация диагностических сообщений

Поскольку излишне частые и подробные сообщения могут засорить журнал, Simics предоставляет несколько механизмов для их фильтрации. С помощью команды `log-type` определяется, какие из типов сообщений будут выводиться, а какие будут игнорироваться. При этом можно установить фильтрацию как для отдельных объектов, так и для всей симуляции в целом.

Кроме того каждое сообщение, кроме типа error (они выводятся всегда), имеют уровень важности, выражаемый числом от 1 до 4. Сообщения уровня 1 самые важные, тогда как на четвёртом уровне обычно содержится информация, необходимая для отладки. При старте симуляции по умолчанию показываются только важные сообщения (первого уровня). Для управления тем, начиная с какого уровня они будут выводится, доступна команда `log-level`. Она может использована как глобальная, так и быть указанной для отдельных объектов:

```
simics> log-level 4
New global log level: 4
simics> viper.mb.rom.log-level 3
[viper.mb.rom] Changing log level: 4 -> 3
```

2.6. Атрибуты

Атрибуты являются аналогом полей в объектах из парадигме ООП. В симуляции основная их задача состоит в хранении архитектурного состояния моделей. Обратиться к ним можно по их имени, указываемому после имени объекта, с помощью оператора стрелки «`->`». Например, для просмотра содержимого регистров центрального процессора можно использовать соответствующие атрибуты:

```
simics> viper.mb.cpu0.core[0][0]->rax
0
simics> viper.mb.cpu0.core[0][0]->rip
65520
simics> viper.mb.cpu0.core[0][0]->rdx
67233
```

```
simics> viper.mb.cpu0.core[0][0]->xmm
[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
     0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
     0], [0, 0]]
```

Для изменения значения, хранимого в атрибуте, используется оператор присваивания «=»:

```
simics> viper.mb.cpu0.core[0][0]->rax = 123
```

Естественно, что изменение атрибутов необходимо проводить очень аккуратно, так как их содержимое напрямую влияет на симуляцию. Например, если изменить значение атрибута `rip` и связанного с ним регистра `RIP`, то дальнейшее исполнение кода начнётся с нового места.

2.6.1. Типы атрибутов

По аналогии с переменными в языках программирования атрибуты имеют типы. При присвоении атрибуту нового значения с помощью присваивания производится проверка, что тип выражения с правой стороны соответствует его заявленному типу, и, если это не так, операция прерывается.

Типа каждого атрибута специфицируется при написании модели с помощью строки, символы которой определяют допустимые варианты входных значений.

1. Скалярные. Имеют типы `''i''` (целое число), `''s''` (строка), `''b''` (булевый тип).
2. Объекты. Их тип — `''o''`. С помощью таких атрибутов одни устройства могут вызывать методы других.
3. Списки. Они могут быть как однородными `''[iiii]''`, так и содержать элементы разных типов `''[oios]''`. Кроме того, их длина может быть переменной: `''[i*]''`.
4. Пустой тип `''n''`. В консоли он задаётся с помощью NIL или нуля.
5. Вариативный тип, значение которого может быть одним из нескольких ранее описанных: `''i|s|o''`.

Кроме того, некоторые атрибуты при регистрации могут быть помечены как псевдоатрибуты или атрибуты только для чтения.

Почти всегда они не соответствуют «настоящему» архитектурному состоянию, а используются для более удобного представления информации об устройстве или для нужд отладки. Более подробно о них будет сказано в последующих главах.

2.7. Задания

1. Найдите команду, перечисляющую все атрибуты заданного объекта. Перечислите все атрибуты объектов центральных процессоров `viper.mb.cpu0.core[0][0]`.
2. Найдите команду, перечисляющую все объекты текущей симуляции. Какие флаги она имеет и каково их назначение?
3. Изучите атрибуты и команды объекта `sim`. Какому устройству гостевой симуляции он соответствует?

2.8. Контрольные вопросы

1. Перечислите известные вам способы получения справки по командам и моделям Simics.
2. Какие языки программирования чаще всего используются для написания моделей устройств Simics? Почему?
3. Чем, по-вашему, отличается объект с моделью устройства от объекта компонента?

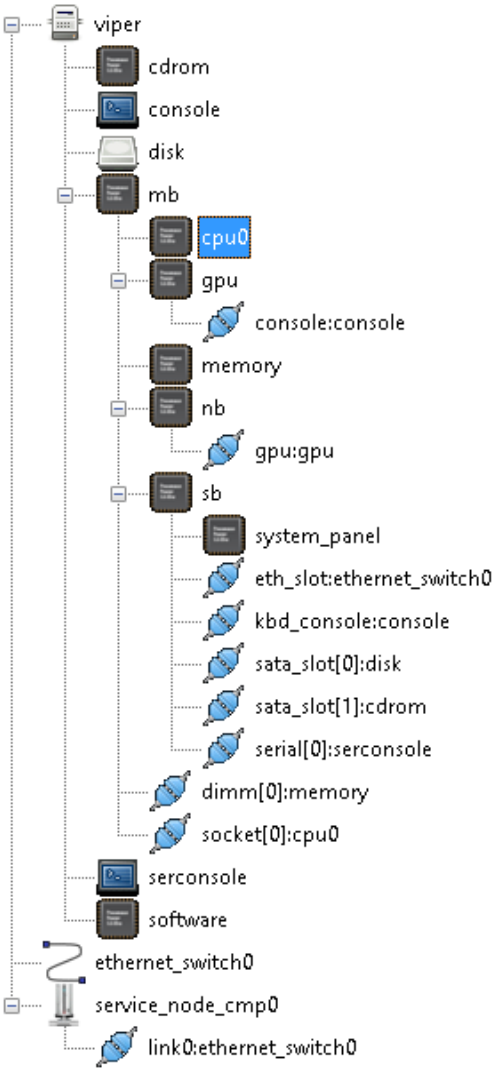

Рис. 2.1. Конфигурация системы Viper

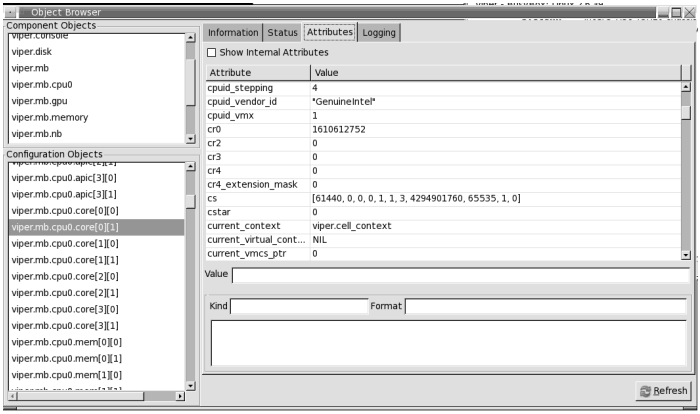

Рис. 2.2. Окно для просмотра объектов и их атбрибутов

3. Создание программных моделей

3.1. Цель занятия

В данной лабораторной работе продолжается изучение принципов работы симулятора Simics. В ней будут определены основные понятия, используемые при работе с симуляцией и при написании моделей устройств. В качестве основного языка программирования моделей будет использоваться Си.

Кроме использования готовых моделей устройств из поставляемых в составе Simics, имеется возможность создавать свои собственные модели различных устройств. Для этого симулятор предоставляет среду сборки, API для взаимодействия с симулятором и специальный язык DML для быстрой разработки периферийных устройств[1]. Подробное описание принципов написания моделей периферийных устройств дано в [3]; особенности интеграции моделей центральных процессоров описываются в [2].

3.2. Модули Simics

Базовая единица загрузки нового кода Simics — *модуль*, разделяемая библиотека (в Linux это файлы с расширением .so или .pymod), использующая API Simics. Каждый класс объектов должен содержаться максимум в одном модуле. При этом один модуль может содержать в себе более одного класса моделей.

Для просмотра текущих загруженных в симуляцию модулей используйте команду `list-modules -l`. Например, в «пустой»

[1]В данной работе DML не будет использоваться; однако Simics API доступен из всех поддерживаемых языков, включая Python, Си и C++.

симуляции этот список выглядит так (флаг `-v` позволяет увидеть больше информации о том, откуда загружены модули):

```
simics> list-modules -v -l
Name                    Status   ABI  Build ID  Thread-safe
Path
-------------------------------------------------------------
breakpoint-manager      Loaded  4052      4145  Yes
/opt/simics/simics-4.6/simics-4.6.103/linux64/lib/breakpoint
    -manager.so
software-tracker        Loaded  4052      4145  Yes
/opt/simics/simics-4.6/simics-4.6.103/linux64/lib/software-
    tracker.pymod
software-tracker-iface  Loaded  4052      4145  Yes
/opt/simics/simics-4.6/simics-4.6.103/linux64/lib/software-
    tracker-iface.so
```

Поиск стандартных модулей, доступных для загрузки, проводится при старте Simics в директориях инсталляции базового и других пакетов. Пользовательские пакеты ищутся в текущем workspace в подпапке `linux64/lib`. Они помещаются туда в результате успешной компиляции из исходных кодов.

Для загрузки некоторого модуля из командной строки вручную используется команда `load-module <name>`. После этого становятся доступны все классы и команды, определённые в нём, т.е. можно создавать новые объекты для моделей устройств новых классов.

3.3. Исходный код и сборка моделей устройств

Код всех модулей дожен быть размещён в текущем workspace, в поддиректории `modules`. По общим соглашениям имя директории должно совпадать с именем модуля. Система сбоки модулей Simics довольно сложна и требует строгого соблюдения процедуры. Она основана на GNU Make [1] и использует компилятор GCC на всех поддерживаемых хозяйских платформах.

3.3.1. Генерация шаблонного устройства

Для начала необходимо сгенерировать «скелет» нового устройства — минимально необходимый набор файлов. Это позволит начать работать с уже компилирующимся примером.

Из workspace выполните команду:

```
$ ./bin/workspace-setup --copy-device sample-device-c
```

Теперь в директории `modules` появился новый элемент `sample-device-c` с исходными файлами:

```
$ ls -R modules/
modules/:
sample-device-c

modules/sample-device-c:
Makefile  commands.py  sample-device.c  test

modules/sample-device-c/test:
SUITEINFO  s-sample-c.py
```

3.3.2. Сборка с помощью Make

Для запуска сборки достаточно использовать программу `make` с именем цели, совпадающей с именем модуля:

```
make sample-device-c
=== Environment Check ===
'/home/simics/workspace' is up-to date
gcc version 4.6.2
=== Building module ''sample-device-c'' ===
        module_id.c
DEP     module_id.d
DEP     sample-device.d
CC      sample-device.o
CC      module_id.o
CCLD    sample-device-c.so
        mod_sample_device_c_commands.pyc
```

Далее мы будем работать с файлами внутри директории `sample-device-c`.

3.3.3. Структура Makefile

В целом синтаксис языка Makefile позволяет писать достаточно сложные и запутанные правила для сборки приложений. В Simics используется собственная система соглашений для упрощения определения модулей. В простейшем случае достаточно перечислить все требуемые файлы с исходными кодами в переменной `SRC_FILES`. Кроме того, имя нового класса должно указываться в переменной `MODULE_CLASSES` в `Makefile`.

3.4. Регистрация нового класса

Перейдём теперь к деталям того, как использовать Simics API для создания нового класса устройств. При загрузке любого модуля, написанного на Си, Simics исполняет из него единственную функцию с именем `init_local()`. В ней должна быть выполнена регистрация нового класса объектов с указанием всех его свойств.

Для этого используется функция `SIM_register_class()`. В примере `sample-device-c`:

```
conf_class_t *class = SIM_register_class("sample-device-c",
    &funcs);
```

Первый аргумент этой функции — имя класса, второй — структура, задающая функцию-конструктор новых копий объекта, а также строки с описанием.

В результате успешного завершения возвращается ссылка на `class`. Только что созданный класс пока что почти пуст — он не экспортирует наружу никаких свойств моделируемых объектов. Перейдём к добавлению двух важнейших аспектов любого полноценного класса — регистрации атрибутов и интерфейсов.

3.5. Атрибуты

Атрибуты позволяют видеть и изменять состояние объекта из командной строки Simics. Кроме того, они необходимы для того, чтобы механизм точек сохранения мог корректно записывать состояние на диск и впоследствии загружать его.

Структура объектов `sample-device-c` задана следующей структурой:

```
typedef struct {
        /* Simics configuration object */
        conf_object_t obj;

        /* device specific data */
        unsigned value;

} sample_device_t;
```

Здесь `conf_object_t obj` — «базовый класс» иерархии объектов Simics, он должен присутствовать в любом объекте. Собственно устройство имеет только один регистр, названный `value`. Для того, чтобы иметь возможность читать и писать его значение, с этим полем ассоциируется атрибут, также названный value:

```
SIM_register_typed_attribute(
        class, ''value'',
        get_value_attribute, NULL, set_value_attribute, NULL
    ,
        Sim_Attr_Optional, ''i'', NULL,
        ''The <i>value</i> field.'');
```

Здесь `class` — ранее созданный класс устройства, `get_value_attribute()` и `set_value_attribute()` — функции для чтения и записи[1] значения, `''i''` — тип атрибута (в данном случае это целое число), строка `''The <i>value</i> field.''` — справка о назначении атрибута. Необязательные аргументы функции `SIM_register_typed_attribute()` равны `NULL`.

Особое внимание следует обратить на то, как устроены getter и setter. Значения атрибутов, которые могут иметь довольно сложные типы, упакованы в специальный класс `attr_value_t`. Для получения значений используется семейство функций `SIM_attr_*`.

[1] Т.н. getter и setter.

3.6. Интерфейсы

Назначение методов, группируемых в интерфейсы, состоит в изменении состояния устройств, а также чтения значений. В отличе от атрибутов, способных делать то же самое, интерфейсы служат для представления архитектурных возможностей устройств. Атрибуты не имеют выражения в реальной аппаратуре, тогда как интерфейсы напрямую отображаются на шины, сигналы, протоколы и т.п.

Каждый интерфейс имеет уникальное имя и фиксированный набор методов, объединённых общей целью. Модель, желающая предоставлять некоторый интерфейс для других устройств, обязана реализовать один или несколько его методов и затем объявить его доступным. Устройство, имеющее ссылку на объект, может получить по нему заявленные интерфейсы и вызывать включённые в него методы. Таким образом, в Simics интерфейсы предоставляют объектно-ориентированную парадигму для взаимодействия отдельных моделей.

Например, один из атрибутов процессора, настраиваемый на этапе инициализации модели, — это `physical_memory`, его тип `o`, т.е. «объект». Допустим, что `cpu->physical_memory = mem`. По указателю на объект `mem` процессор может извлечь из него реализацию интерфейса `memory-space`, который содержит методы `read`, `write`, `access` и др. для работы с пространствами памяти.

В случае `sample-device-t` регистрируются два интерфейса — `sample_interface` и `io_memory`. Пример регистрации первого из них:

```
static const sample_interface_t sample_iface = {
        .simple_method = simple_method
};
SIM_register_interface(class, SAMPLE_INTERFACE, &
    sample_iface);
```

Здесь `sample_iface` — структура из указателей на функции, которая передаётся функции `SIM_register_interface()`. В данном случае в ней содержится только один указатель — `simple_method`. `class` — тот же самый класс, что был получен при ре-

гистрации класса, `SAMPLE_INTERFACE` — строка[1] с именем интерфейса.

3.7. Ход работы

- Соберите модуль `sample-device-c`.
- Измените код внутри модуля так, чтобы поменять имя класса на `mydevice`, а имя модуля — на `mymodule`.
- Добавьте атрибут `counter`, содержащий значение целого типа.
- Измените логику работы `simple_method()` так, чтобы она изменяла значение `counter`, например, монотонно увеличивала его.
- Загрузите Simics, затем подгрузите модуль `mymodule`.
- С помощью команды `help mydevice` проверьте, что новый класс действительно предоставляет все интерфейсы и атрибуты.
- Создайте объект класса `mydevice` с именем `dev0`[2].
- Вызовите метод simple_method из устройства с помощью команды

```
simics> @conf.dev0.iface.sample_iface.simple_method(3)
```

- Проверьте, что значение атрибута `counter` изменилось.

3.8. Контрольные вопросы

1. Чем отличается вывод команды `list-modules` с флагом `-l` и без него?
2. Некоторые найденные модули по той или иной причине могут быть отвергнуты при загрузке. Увидеть их список можно с помощью команды `list-failed-modules`. Загрузите модель Viper и определите причины, по которым список, выдаваемый командой `list-failed-modules`, не пуст.

[1] По соглашениям строки с названиями интерфейсов хранятся в #define'ах, названных из заглавных букв имени, т.е. `SAMPLE_INTERFACE` эквивалентно `''sample_interface''`.

[2] О создании конфигураций рассказывается в следующей главе

3. По каким причинам не следует использовать манипуляцию атрибутами одного устройства из другого?
4. Объясните, для чего служит атрибут `add_log`, регистрируемый в конце `init_local()`.

Список литературы к занятию

1. GNU ‘make’. — Free Software Foundation, 2014. — URL: `http://www.gnu.org/software/make/manual/make.html`.
2. Processor Model Integration Guide 4.6 / Wind River. — 2014.
3. Simics Model Builder User Guide 4.6 / Wind River. — 2014.

4. Язык сценариев Simics

4.1. Цель занятия

В данной лабораторной работе продолжается изучение принципов работы симулятора Simics. В ней будут даны основы языка сценариев конфигурации Simics, а также использование языка Python. Подробная информация может быть также найдена в [1].

4.2. Файлы сценариев, интерактивный ввод и опции командной строки

Во время своей работы окружение Simics, кроме объектов моделей устройств, предоставляет динамическую среду программирования с переменными, ветвлениями, циклами и другими возможностями. Пользователям, знакомым с Unix Shell, Windows CMD.EXE и другими REPL[1]-системами, она окажется достаточно привычной.

Все команды, понимаемые Simics, можно вводить тремя способами.

1. Через *файл сценария*. Традиционно такие файлы получают расширение `.simics`. Имя файла передаётся как опция командной строки Simics, после чего все его строки интерпретируются как команды и исполняются в неинтерактивном режиме, т.е. без возможности пользователя повлиять на ход исполнения. Если указано несколько имён файлов, то все они будут исполнены последовательно.
   Файлы сценариев чаще всего используются для конструирования моделируемой системы из отдельных компонент. При

[1]От *англ.* read-eval-print loop.

этом в изначально пустой симуляции возникает готовая к запуску виртуальная система с заданными устройствами и фиксированными настройками, с которой пользователь затем проводит эксперименты.

Кроме того, имеется возможность создавать полностью неинтерактивные сценарии, которые протекают от начала до конца без участия человека. Для этого последней строкой файла должна быть команда `quit`, завершающая работу симулятора.

Например, если в корне текущего workspace есть файл `1.simics` со следующим содержимым:

```
# 1.simics – a short script
# This is comment
if not defined a {$a = 2}
echo ”Variable a is ” + $a
echo ”About to exit”
quit
```

То его исполнение приведёт к следующему выводу:

```
$ ./simics 1.simics
Wind River Simics 4.8 (build 4561 linux64) Copyright
    2010–2014 Intel Corporation

Use of this software is subject to appropriate license.
Type ’copyright’ for details on copyright and ’help’
    for on–line documentation.

Variable a is 2
About to exit
$
```

2. После исполнения последней строки из последнего файла сценария симулятор переходит в *интерактивный режим*, и пользователь может вводит новые команды через строку приглашения (*англ.* command prompt). Вывод каждой команды, если он есть, будет распечатан сразу после её завершения и перед возвращением управления пользователю.
3. Наконец, команды можно указывать прямо *в опциях командной строки* Simics, с ключом `-e`. При этом, если в команде есть специальные символы, интерпретируемые самой

командной оболочкой операционной системы, то их приходится «прятать» с помощью кавычек. Для Unix Shell среди них содержатся символы пробела, доллара, одинарные и двойные кавычки, обратные слеши.

Тот же самый файл сценария `1.simics` выдаст другое значение для переменной `a`, если её значение переопределить в командной строке:

```
$ ./simics -e '$a="_changed_"' 1.simics
Wind River Simics 4.8 (build 4561 linux64) Copyright
    2010-2014 Intel Corporation

Use of this software is subject to appropriate license.
Type 'copyright' for details on copyright and 'help'
    for on-line documentation.

Variable a is _changed_
About to exit
$
```

Заметим, что, в отличие от многих утилит Linux, порядок аргументов командной строки Simics важен — он определяет порядок их исполнения. Все принимаемые флаги можно увидеть, исполнив `simics -help`.

4.3. Переменные окружения

В процессе работы симуляции пользовательские данные можно хранить в переменных. В Simics переменные имеют имена, начинающиеся со знака доллара. Они могут хранить числа, строки, булевы значения и списки. Как и во многих других скриптовых языках, тип переменных определяется динамически.

Пример операций с переменными:

```
simics> $a
No CLI variable "a"
simics> $a = 2
simics> $a
2
simics> $a = "text value"
simics> $a
"text value"
```

```
simics> $a =(3+4)/5
simics> $a
1
simics> $a =(3+4)/5.0
simics> $a
1.4
simics> $a = [1,2,3]
simics> $a
[1, 2, 3]
simics> $a = TRUE
simics> $a
TRUE
simics> defined a
TRUE
simics> defined b
FALSE
```

Очень часто переменные используются как элементы конфигурации гостевой платформы при её создании. Фиксированного соглашения на их именование нет; однако, традиционно `$cpu_class` содержит имя процесссора, `$system` — имя создаваемого компонента верхнего уровня (т.е. материнской платы), `$disk` — путь к образу диска и т.д.

Для того, чтобы увидеть все определённые переменные, используется команда `list-variables`. Например, для скрипта `viper-busybox.simics` определены десятки переменных, используемых для первоначальной конфигурации:

```
simics> list-variables
$apic_freq_mhz           = 133
$bios                    = ''seabios-simics-x58-ich10-sata
    -1.6.3-20131111.bin''
$break_on_reboot         = TRUE
$cdrom                   = ''viper.cdrom''
$connect_real_network    = TRUE
$connect_usb_tablet      = FALSE
$console                 = ''viper.console''
$cpi                     = 1
$cpu                     = [''viper.mb.cpu0'']
$cpu_class               = ''core-i7''
$create_network          = TRUE
$create_processor        = ''create-processor-core-i7''
$dhcp_pool_ip            = ''10.10.0.100''
$dhcp_pool_size          = 100
```

```
$dimm                    = ''viper.mb.memory''
$disk                    = ''viper.disk''
$disk_size               = 20496236544
$efi                     = FALSE
$enter_in_boot_menu      = TRUE
$eth_cnt                 = ''eth_slot''
$eth_comp                = ''viper.mb.sb''
$eth_link                = ''ethernet_switch0''
$freq_mhz                = 2000
...
```

4.4. Управляющие конструкции

В языке сценариев Simics используются классические команды для управлением исполнения, такие как `if (cond) then {op1} else {op2}`, `while cond {op}` и `foreach $var in $list {op}` . Также доступна конструкция для ловли исключений `try {op1} except {op2}`.

4.5. Python

Кроме использования собственного языка сценариев Simics позволяет использовать отдельные команды и даже целые файлы, написанные на Python. Для отдельных команд необходимо использовать знак «эт» в начале строки для того, чтобы интерпретатор понял, что синтаксис последующего текста следует разбирать как Python-строку:

```
simics> @print ''Hello''
Hello
```

Для запуска целого файла, написанного на Python, следует использовать команду `run-python-file <file>` или указать его с флагом `-p` командной строки Simics:

```
$ ./simics –p 2.py
```

4.6. Ход работы

TODO

4.7. Последовательность конструирования модели системы

При создании симуляции она и все участвующие в ней объекты проходят две фазы, чётко отделённые друг от друга во времени, тогда как внутри каждой из них порядок инициализации компонент неопределён.

1. *Объявление* устройств и соединение их с помощью инициализации их атрибутов. На этом этапе ещё не производится проверок на соответствие типов, определённость значений всех обязательных атрибутов. Устройства представлены так называемыми предобъектами (*англ.* preobj).
2. *Инстанциирование* устройств с помощью команды `instantiate-components` или с помощью вызова `SIM_add_configuration()`, например, из Python. На этом этапе производятся все проверки на корректность атрибутов, наличие необходимых интерфейсов и т.п. Если не найдено ошибок, то предобъекты преобразуются в полноценные объекты Simics, которые могут участвовать в симуляции. Если обнаружены ошибки, то уже созданные объекты удаляются, а последующие не создаются.
3. *Пост-инициализация.* Некоторые устройства требуют дополнительного шага. Желательно, чтобы объём действий, выполняемых в этой фазе, был минимален.

При необходимости добавить новые объекты в процессе симуляции описанные фазы могут быть повторены.

4.8. Задания

1. Из документации Hindsight [1] выясните, каким образов на Python записать следующие конструкции: 1) присваивание переменной Simics `$var` значения `''test''`; 2) чте-

ние атрибута `classname` устройства с именем `dev0`; 3) Вызов метода `fun()` интерфейса `sample_interface` устройства `board.nb.cb[0][0]`.

2. Найдите способ программно узнать, какую версию Python использует Simics.

4.9. Контрольные вопросы

1. Перечислите три способа ввода команд в Simics.
2. Для каких нужд могут понадобится полностью неинтерактивные сценарии симуляции? Приведите несколько примеров.
3. Найдите, откуда появляются значения всех переменных из сценария `viper-busybox` при условии, что их не задали в командной строке или вручную.
4. Придумайте способ прочитать значение некоторой переменной из модели устройства. Почему не существует прямого API для выполнения подобной операции?

Список литературы к занятию

1. Simics Hindsight User Guide 4.6 / Wind River. — 2014.

5. Моделирование платформы с архитектурой CHIP16

Данная практическая работа посвящена реализации модели компьютерной платформы, основанной на спецификации CHIP16 [4] — полностью виртуальной системы, предназначенной для запуска простых видеоигр и демо. Модель строится на основе API Simics и оформляется как набор модулей и сценариев для данного симулятора.

5.1. Исходные спецификации CHIP16

CHIP16 — это компьютер с RISC-подобным процессором архитектуры фон-Неймана, работающий на частоте 1 МГц, имеющий 64 кбайт ОЗУ, со спрайтовой 16-цветной графикой разрешением 320×240, двумя джойстиками с 8 кнопками каждый и одноканальной звуковой картой.

5.1.1. Существующие приложения

- Референсный симулятор — MASH16 [3].
- Описание устройств, набора инструкций и периферии [2].
- Готовые образы памяти с приложениями, собранными для этой архитектуры [1].

5.2. Структурная схема платформы

Базовые узлы системы показаны на рис. 5.1. Каждый обозначенный блок представляет собой один класс Simics, имя которого стоит после двоеточия.

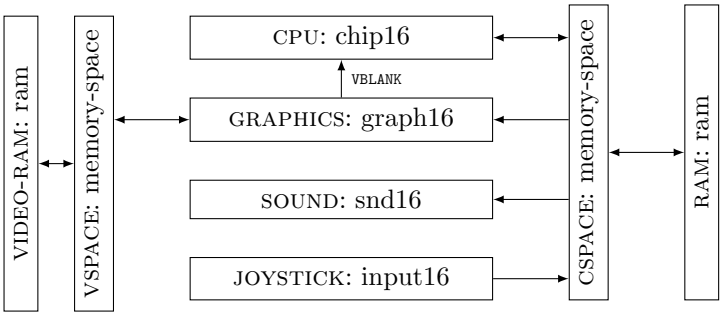

Рис. 5.1. Схема соединения блоков

5.3. Модификации спецификации для полноплатформенной модели

Оригинальная документация не описывает некоторые важные для практического построения детали взаимодействия узлов. Поэтому для нужд задания вводятся уточнения по принципам работы ряда узлов.

VBLANK. Инструкция VBLANK введена для синхронизации ЦПУ с видеопроцессором. В данной работе её семантика изменена. VBLANK вызывает остановку процессора до поступления следующего прерывания.

Прерывания. Для обеспечения работы VBLANK используется прерывание от видеокарты до процессора. Последний реагирует на прерывание пробуждением из режима остановки.

Инструкции DRW, PAL, SNG, CLS, BGC, SPR, FLIP, SNDx, SNP . Данные инструкции предназначены для работы с периферией (видео и звуком). Для их реализации необходимо иметь однонаправленный канал для передачи сообщений. Для этих нужд используется часть резервированного диапазона адресов I/O. Карта памяти выглядит следующим образом.

TODO Описать форматы пакетов аудио- и видеосообщений.

Диапазон адресов	Длина	Назначение
`0x0000 - 0xfdef`	65008	ОЗУ
`0xfdf0 - 0xffef`	512	Стек
`0xfff0 - 0xfff1`	2	Джойстик 1
`0xfff2 - 0xfff3`	2	Джойстик 2
`0xfff4 - 0xfff5`	2	Звуковая карта
`0xfff6 - 0xfff9`	4	Видеокарта
`0xfffa - 0xffff`	6	Зарезервировано

5.4. Ход работы

В данной секции разобраны общие вопросы организации разработки.

5.4.1. Технология разработки

- Хранение кода — в Git `https://github.com/yulyugin/ilab-simics`. Лицензия кода — закрытая, согласно договору предоставления Intel Academic SLA 1.0 (см. пункт 6.2 соглашения).
- Документация — в вики `https://github.com/yulyugin/ilab-simics/wiki`.
- Распределение задач — по одному модулю Simics на одного–двух выполняющих.
- Промежуточная проверка качества — юнит тесты для отдельных устройств.
- Финальный продукт работы — набор моделей и связывающий их скрипт (построенный на основе стандартной цели cosim)

5.4.2. ЦПУ

Основа для модели — модифицированный `sample-risc`. TODO Подготовить код.

5.4.3. Видеоконтроллер

Основа для модели — `sample-device-c`. Отрисовка изображения производится через SDL. Пример работы с экраном и клавиатурой из SDL: `http://www.aaroncox.net/tutorials/2dtutorials/sdlkeyboard.html`.
Для 2.0: `http://www.sdltutorials.com/sdl-tutorial-basics`, `https://wiki.libsdl.org/MigrationGuide`.

Пример на русском языке: `http://habrahabr.ru/post/134936/`

5.4.4. Джойстик

Основа для модели — `sample-device-c`. Ввод с хозяйской клавиатуры производится через SDL.

5.5. Минимум и максимум цели проекта

В зависимости от полноты выполнения студентами подзадач проекта выделяются следующие вехи, обозначающие достижение следующей ступени к симуляции, по сравнению с референсной моделью MASH16.

1. Модель способна исполнять образ памяти, написанный участниками проекта.

2. Модель способна исполнять образ памяти из директории Demo (графическое приложение без звука и ввода).

3. Модель способна исполнять образ памяти из директории Demo (графическое приложение со звуком и без ввода).

4. Модель способна исполнять образ памяти из директории Games (графическое приложение со звуком и вводом с джойстика).

5.6. Порядок занятий и заданий

Занятие	Задание
Системы контроля версий Git	Установить Linux, скачать репозиторий проекта, изменить файл, зафиксировать изменения в репозитории
Сборка проекта: модули, модели, workspace	Собрать заглушки модулей, создать недостающие устройства (распределение устройств по владельцам)
Структура кода моделей Simics: атрибуты, интерфейсы, функция init_local	Начать заполнять устройства архитектурным состоянием
Архитектура CHIP16	Продолжать работу над устройствами
Моделирование ЦПУ через интерпретацию; моделирование графики	Реализовать первые инструкции в процессоре; подключить библиотеку SDL к сборке проекта
Моделирование ввода с клавиатуры. Интерактивность симуляции. MMIO. Прерывания	Продолжать писать модели устройств
Тестирование устройств. Среда Simics для конфигурации соединения устройств симуляции.	Запустить скрипт targets/chip16; написать свои скрипты для отдельных устройств

5.7. Контрольные вопросы

1.
2.

3.

Список литературы к занятию

1. *Kelsall T.* Chip16 Games & co. — URL: http://www.doc.ic.ac.uk/~tk2010/chip16/games/ (дата обр. 13.04.2014).
2. *Kelsall T.* Machine Specification·tykel/chip16 Wiki. — 2014. — URL: https://github.com/tykel/chip16/wiki/Machine-Specification (дата обр. 13.04.2014).
3. *Kelsall T.* Reference Chip16 Emulator. — 2014. — URL: https://github.com/tykel/mash16 (дата обр. 13.04.2014).
4. *Shendo* Codename: CHIP16 (prev. CHIP9)|NGEmu. — 2010. — URL: http://ngemu.com/threads/codename-chip16-prev-chip9.138170/ (дата обр. 13.04.2014).

6. Моделирование процессора архитектуры OpenRISC 1000

Данная практическая работа посвящена реализации модели компьютерной платформы, основанной на спецификации OpenRISC 1000 [1]. Модель строится на основе API Simics и оформляется как набор модулей и сценариев для данного симулятора.

6.1. Спецификации OpenRISC 1000

OpenRISC 1000 — дизайн процессора, построенный по философии Open Source [3] как для программного кода, так и для аппаратуры. Спецификация определяет 32- или 64-битный RISC-процессор общего назначения, с пятью стадиями конвеера, необязательной поддержкой MMU (*англ.* memory management unit), кэшей, управлением энергопотреблением и др. Важная для данной лабораторной работы черат спецификации OpenRISC 1000 — это опциональность многих частей функциональности, что позволяет ограничиваться только минимально необходимыми для конкретной реализации элементами.

Существуют реализации OpenRISC 1000 в виде программных моделей, спецификаций для FPGA и описаний RTL на языке Verilog. Программная поддержка существует со стороны компиляторов GCC и LLVM, адаптированных библиотек LibC и GNU toolchain.

6.1.1. Этапы проекта

Для создания рабочей функциональной модели ЦПУ требуется спецификация на следующие его элементы: архитектурное со-

стояние (число и типы регистров), набор инструкций (кодировка и семантика), а также интерфейсы к внешним элементам системы (память и периферийные устройства). Детали, относящиеся к особенностям организации конвеера, длительностям работы отдельных инструкций, устройству кэшей и т.п. не являются необходимыми на данном уровне точности моделирования.

6.2. Краткое введение в архитектуру OpenRISC 1000

6.2.1. Набор регистров

Классификация регистров OpenRISC 1000 (рис. 6.1), требующих реализации в ходе работы.

- 32 регистра общего назначения шириной в 64 бита, доступные из пользовательского и супервизорного режимов: `R0` – `R31`. Регистр R0 всегда должен содержать ноль.

- Специфичные для модулей регистры (*англ.* special purpose registers, SPR). В случае, если некоторая опциональная часть спецификации реализована, с ней могут идти дополнительные регистры для индикации статуса и управления состоянием. Каждый из них имеет собственное имя и функциональность. SPR разделены по группам. Группа 0 — регистры супервизора шириной 32 бита. Некоторые из них могут быть доступны на чтение из пользовательского режима.

6.2.2. Набор инструкций

Набор инструкций состоит из нескольких классов (рис. 6.2), соответствующих различным задачам и типам обрабатываемых данных.

- ORBIS32 — 32-битные команды общего назначения; единственный класс, обязательный для реализации;

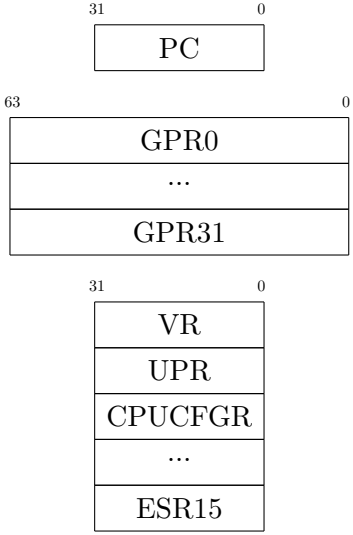

Рис. 6.1. Регистры OpenRISC 1000

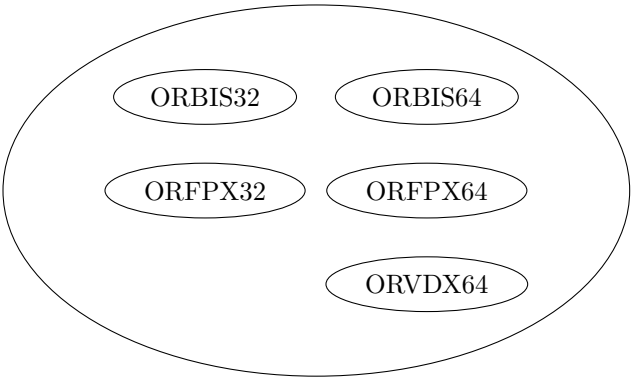

Рис. 6.2. Классы инструкций набора команд OpenRISC 1000

- ORBIS64 — 64-битные команды общего назначения;
- ORFPX32 — 32-битные команды для работы над числами с плавающей запятой;
- ORFPX64 — 64-битные команды для работы над числами с плавающей запятой;
- ORVDX64 — команды для операции над векторами данных с элементами шириной 8, 16, 32 или 64 бита;
- Частные команды, специфичные для конкретной модели.

Ширина отдельных инструкций фиксирована и равна 32 битам. ORBIS32 состоит из следующих подклассов команд:

- арифметические операции;
- базовые инструкции для обработки сигналов (*англ.* digital signal processing, DSP);
- загрузка и запись данных из памяти;
- управление порядком исполнения;
- прочие.

6.3. Кодовая база Simics для создания моделей процессоров

Основа модели ЦПУ — код `sample-risc`, модифицированный для нужд учебного проекта:

- убрана многоядерность;
- декодер упрощён до одной инструкции.

Конфигурация платформы (ЦПУ, память, опционально таймер) — основана на сценарии `cosim`.

6.4. Кодовая база Simics для создания моделей периферийных устройств

Основа для моделей периферийных устройств — код `sample-device-c` и `sample-timer-device`, переименованный в `tick-device`.

6.5. Совместная симуляция процессоров и периферийных устройств

Следующий пример работы с сессией Simics иллюстрирует совместную работу исполняющих (процессоры) и неисполняющих устройств в симуляции.

```
simics> phys_mem0.map
   Base   Object       Fn   Offset    Length
-------------------------------------------------
0x00000   ram0          0      0x0   0x10000
    width 8192 bytes
0x00100   joy0          0      0x0      0x10
    width 8 bytes
0x00110   joy1          0      0x0      0x10
    width 8 bytes
0x08000   test0         0      0x0    0x8000
0x10000   tick0:regs           0x0       0x8
    width 2 bytes

simics> tick0->irq_dev
[''chip0'', ''EXTERNAL_INTERRUPT'']
```

tick0 — это периодический таймер, chip0 - центральный процессор. Они соединены через порт EXTERNAL_INTERRUPT, кроме того, регистры tick0 отображены в физическую память, начиная с адреса 0x10000.

```
simics> log-level 4
New global log level: 4
simics> phys_mem0.write address = 0x10000 value = 0 -b size
    = 2
[tick0 info] Write to register regs.counter (addr 0x10000)
    <- 0
```

```
simics> phys_mem0.write address = 0x10002 value = 0 -b size
    = 2
[tick0 info] Write to register regs.reference (addr 0x10002)
     <- 0
simics> phys_mem0.write address = 0x10002 value = 100 -b
    size = 2
[tick0 info] Write to register regs.reference (addr 0x10002)
     <- 0x64
simics> peq
   Step  Object      Description

  Cycle  Object      Description
      0  sim         Time Quantum End
 500001  cosim_cell  sync_report
1000000  cosim_cell  sync_block
```

Очередь событий не содержит архитектурных событий, только псевдособытия симулятора.

```
simics> phys_mem0.write address = 0x10004 value = 2 -b size
    = 2
[tick0 info] Write to register regs.step (addr 0x10004) <- 0
    x2
[tick0 info] step updated
[tick0 info] reposting event to 200 cycles ahead
simics> peq
   Step  Object      Description

  Cycle  Object      Description
      0  sim         Time Quantum End
    200  tick0       reference_reached
 500001  cosim_cell  sync_report
1000000  cosim_cell  sync_block
```

После настройки регистров в очереди событий появилось событие таймера reference_reached.

```
simics> log-level 1
New global log level: 1
simics> continue-cycles 199
[chip0] v:0x031c p:0x031c  nop
simics> peq
  Step  Object      Description

 Cycle  Object      Description
     1  tick0       reference_reached
```

```
499802  cosim_cell  sync_report
999801  sim         Time Quantum End
999801  cosim_cell  sync_block
```

Расстояние до события таймера уменьшается.

```
simics> continue-cycles 10
[tick0 info] reference_reached is fired!
[chip0 info] EXTERNAL_INTERRUPT raised.
[chip0 info] EXTERNAL_INTERRUPT lowered.
[chip0] v:0x0344 p:0x0344  nop
simics> peq
  Step  Object      Description

 Cycle  Object      Description
499792  cosim_cell  sync_report
999791  sim         Time Quantum End
999791  cosim_cell  sync_block
```

Таймер отработал, а процессор получил прерывание.

6.6. Тестирование моделей

Для тестирования корректности моделирования отдельных инструкций применяется юнит-тестирование на основе фреймворка `test-runner`. Для проверки последовательностей команд гостевой код может быть написан вручную в машинных кодах OpenRISC 1000 или же с помощью GNU toolchain [2].

6.7. Порядок работы

1. Подготовить workspace Simics.
2. Собрать код sample-risc.
3. Реализовать набор регистров базового архитектурного состояния.
4. Реализовать декодер команд.
5. Интегрировать модель openrisc в сценарий платформы.

6. Реализовать модель таймера.
7. Интегрировать модель таймера в сценарий платформы.

6.7.1. Дополнительные шаги

- Реализовать набор инструкций ORBIS64.
- Реализовать функциональность TLB и/или MMU.
- Реализовать устройство PIC.
- Реализовать модель кэша данных.
- Реализовать набор инструций ORFPX32.

Список литературы к занятию

1. *Open Cores* OpenRISC 1000 Architecture Manual. Architecture Version 1.0 Document Revision 0. — 2012. — URL: `http://opencores.org/websvn,filedetails?repname=openrisc&path=/openrisc/trunk/docs/openrisc-arch-1.0-rev0.pdf`.
2. *Open Cores* OpenRISC GNU tool chain. — 2014. — URL: `http://opencores.org/or1k/OpenRISC_GNU_tool_chain`.
3. The Open Source Definition (Annotated). — Open Source Initiative. — URL: `http://opensource.org/osd-annotated`.

7. Тестирование моделей

7.1. Цель занятия

TODO

Подробнее о написании юнит-тестов для моделей Simics говорится в [1].

7.2. Контрольные вопросы

1.
2.
3.

Список литературы к занятию

1. Testing Simics Models / Wind River. — 2014.

8. Связь симуляции с внешним миром

8.1. Цель занятия

Необходимость коммуникации — доставка данных.

1. Образы дисков, форматы raw, craff, vmdk, iso.
2. Паравиртуализация: hostfs, magic instructions.
3. Сетевое взаимодействие: NAT, port forwarding, bridging.

8.2. Диски

Под дисками мы будем подразумевать устройства хранения данных на жёстких магнитных дисках (стандарты SATA, IDE, FireWire, SCSI), твердотельных накопителях (USB-флешки, SSD), а также оптические диски (CD, DVD, Blu-ray) и теряющие актуальность гибкие диски (*англ.* floppy disks).

С моделированием дисковой подсистемы связано несколько специфичных вопросов.

- Обеспечение высокой скорости симуляции. Объём передаваемых данных для ряда приложений может быть большим, как и связанное с этим замедление модели.
- Обеспечение непосредственного хранения массивов данных. Ёмкость моделируемых дисков может достигать десятков терабайт.
- Постоянство хранилища модели. В отличие от ОЗУ и регистров устройств, жёсткие диски не теряют данные при выключении или перезагрузке компьютера. Однако сохранение состояния между запусками симуляции нарушает принцип её воспроизводимости и повторяемости.

8.2.1. Форматы хранения

Поскольку диск представляет собой устройство с произвольным доступом, естественная форма хранения его данных — это файл в хозяйской системе. В простом случае он содержит собой просто копию байт-в-байт всего содержимого реального диска, т.н. «сырой» (*англ.* raw) образ диска. Преимущество такого формата — его простота и универсальность; практически все симуляторы поддерживают его. Основной недостаток — нерациональное использование хозяйского дискового пространства. Например, симуляция установки ОС может занять 1 Гбайт места на образе диска в 100 Гбайт; результирующий образ диска будет занимать 100 Гбайт, при этом 99% его будут потеряны для хозяйской системы впустую.

Многие симуляторы поддерживают более компактные способы хранения, в которых в файл записываются только изменённые секторы диска; заголовочная часть файла содержит список этих секторов и их местоположение. Как правило, каждая программа имеет свой формат и иногда поддерживает другие или позволяет конвертировать их друг в друга. Примеры: Qcow2 [2] (Qemu), VDI (Oracle Virtualbox), VMDK [4] (VMware ESX), VHD (Microsoft VirtualPC), HDD (Parallels Desktop), CRAFF (Wind River Simics).

Некоторые форматы поддерживают прозрачное сжатие записанных данных.

Для образов оптических дисков, которые в большинстве случаев являются носителями с данными только для чтения, используются сырые образы. Чаще всего они именуются ISO-образами по имени стандарта используемой на них файловой системы ISO 9660[1].

Из-за своего небольшого размера (меньше 3 Мбайт) образы гибких дисков хранятся в raw-формате.

[1]Хотя это не единственный стандарт для оптических дисков; альтернативой является UDF (universal disk format), призванный обойти многие ограничения ISO 9660.

8.2.2. Сохранение состояния дисков

Зачастую нежелательно модифицировать исходный файл образа диска: экспериментальное ПО/вирусы/ошибки пользователя внутри симулятора может сделать его неработоспособным, или же желательно впоследствии запускать симуляцию из первоначального состояния.

Для таких целей в большинстве симуляторов существует опция: все изменённые секторы сохранять не в оригинальном, а в дополнительном **разностном** файле (также называемом **дельтой**). Для моделируемого приложения указанная схема абсолютно прозрачна — внутри симуляции изменения видны там, где и должны быть. Однако после выключения симуляции допустимо отбросить дельту и использовать оригинальный образ. В случае появления желания зафиксировать внесённые в результате последней симуляционной сессии правки — следует воспользоваться утилитами слияния оригинального образа и дельты.

Развивая эту идею, можно вообразить себе схему с несколькими дельтами, полученными на разных этапах симуляции (и даже «дельты к дельтам»), одновременно наложенными на диск. Таким образом, можно иметь множество снимков состояний дискового хранилища, суммарно занимающие места меньше, чем занимали бы отдельные полные копии.

8.3. Копирование файлов внутрь моделируемой системы

Simics — полноплатформенный симулятор, а это значит, что вся целевая система моделируется, включая диски, на которых установлено программное обеспечение. Часто может возникнуть необходимость транспортировки файлов из хозяйской системы в симулируемую и наоборот. Simics предоставляет способ прямого доступа к хозяйской операционной системе из моделируемой, называемой *SimicsFS*.

8.3.1. SimicsFS

SimicsFS — это модуль ядра ОС, доступный в Linux и Solaris, для поддержки специальной файловой системы, делающий доступной файловую систему хозяина внутри гостевой системы. В рассматриваемой нами системе *Viper* поддержка *SimicsFS* уже установлена (описание процесса установки *SimicsFS* можно найти в [3]), поэтому для доступа к хозяйской файловой системе достаточно воспользоваться командой:

```
# mount /host
[   25.060559] [simicsfs] mounted
```

После этого вы можете просматривать, скачивать и удалять файлы с хозяйской ЭВМ. Например, вы можете посмотреть содержимое корневой директории хозяйской операционной системы (Для Linux)

```
# ls /host
bin     dev   ipathfs  lost+found  opt   sbin
    share_debian  tmp
boot    etc   lib      media       proc  selinux  srv
        usr
cgroup  home  lib64    mnt         root  share    sys
        var
```

Вы можете сравнить вывод команды `ls /host` внутри моделируемой системы с выводом команды `ls /` внутри хозяйской системы и убедиться, что он совпадает.

Для того чтобы прекратить работу с *SimicsFS*, достаточно просто отмонтировать хозяйскую файловую систему следующей командой:

```
# umount /host
```

8.3.2. Подключение симулятора к реальной сети

Подключение симулятора к реальной сети открывает много новых возможностей. Например, позволяет упростить процесс загрузки файлов на симулируемую машину с помощью FTP (*англ.*

File Transfer Protocol); данная возможность может также использоваться для доступа к виртуальной машине удаленно с помощью Telnet.

8.3.3. Типы соединений

Simics предоставляет 4 способа подключения к реальной сети, которые описаны в [1]. В данной секции описано, как они работают, их преимущества и недостатки.

Перенаправление портов (*англ.* Port forwarding)

Перенаправление портов — это самый простой тип соединений. Для установления соединения он не требует ни прав администратора, ни какой-либо другой конфигурации хозяйской машины.

Однако перенаправление портов ограничено только TCP- и UDP-трафиком. Другой трафик, например ping пакеты, которые использует ICMP-протокол, не пройдет через такое соединение. Нельзя использовать порты, которые уже используются на хозяйской машине, а также нельзя использовать порты меньше 1024 без прав администратора.

Каждый TCP- и UDP-порт требует отдельного правила перенаправления. По этой причине конфигурация приложений, которые используют множество портов или порты, определяемые произвольным образом (*англ.* random ports), может оказаться довольно обременительной.

Перенаправление портов разрешает коммуникации между гостевой и хозяйской машинами, а также любыми узлами реальной сети.

Соединение через сетевой мост (*англ.* Ethernet bridging connection)

С помощью соединения через сетевой мост у симулируемой машины появляеться возможность непосредственно подключаться к реальной сети. Данное соединение позволяет использовать трафик любого типа. Обычно симулируемый узел использует IP-адреса подсети, так как в данном случае не требуется менять

конфигурацию хозяина. Тем не менее при использовании такого соединения хозяйская машина остается не доступной из гостевой.

Для использования соединения через сетевой мост на хозяйской машине необходимо иметь права на доступ к TAP.

Соединение с хостом (*англ.* Host connection)

С помощью соединения с хостом хозяин подключается к моделируемой сети, позволяющей использование трафика любого типа между моделируемым и реальным узлами.

Соединение с хостом также поддерживает и перенаправление IP. Когда используется перенаправление IP, хозяйская операционная система маршрутизирует IP-трафик между реальной и симулируемой сетью. Из вышесказанного следует, что маршрутизация должна быть настроена между симулируемой и реальной сетью для того, чтобы данный способ работал.

Для использования соединения через сетевой мост на хозяйской машине необходимо иметь права на доступ к TAP.

Таблица 8.1 резюмирует преимущества и недостатки каждого типа соединений. Для простых TCP-сервисов, таких как FTP, HTTP или Telnet, лучше всего подходит перенаправление портов. В нашем случае этого достаточно, поэтому другие способы соединения с реальной сетью мы подробно рассматривать не будем.

Для каждого типа соединений с реальной сетью существет команда, которая принимает объект типа Ethernet link.

8.4. Ход работы

8.4.1. SimicsFS

1. Загрузите Simics со стартовым скриптом `viper-busybox.simics` в конфигурации с процессором класса `core-i7-single`:

```
$ ./simics -e '$cpu_class=core-i7-single' targets/x86-
    x58-ich10/viper-busybox.simics
```

2. Запустите симуляцию:

```
simics> continue
running>
```

3. После того как гостевая система загрузится, авторизуйтесь и примонтируйте хозяйскую файловую систему:

```
busybox login: root
[guest]# mount /host
```

4. Выведите содержимое корневой директории хозяйской операционной системы и сравните его с выводом содержимого папки /host внутри симулируемой машины:

```
[host]$ ls /
[guest]# ls /host
```

5. Создайте файл в своей рабочей директории и убедитесь, что он также стал виден из симулируемой машины:

```
[host]$ cd /share_debian/workspaces/students/ivanov
[host]$ touch test
[guest]# ls /host/share_debian/workspaces/students/
    ivanov
```

6. Удалите созданный файл из симулируемой машины и убедитесь, что в реальной системе он тоже был удален:

```
[guest]# rm test
[host]$ ls /share_debian/workspaces/students/ivanov
```

7. Попробуйте из моделируемой системы удалить какой-либо системный файл хозяйской ОС, например, /bin/sh:

```
[host]$ rm /bin/sh
rm: cannot remove ‘/bin/sh’: Permission denied
[guest]# rm /host/bin/sh
rm: cannot remove ‘/host/bin/sh’: Permission denied
```

Видно, что даже наличие прав администратора внутри симулируемого узла не позволяет нам удалять файлы, недоступные для записи пользователю, который запустил симуляцию.

8.4.2. Подключение к реальной сети

1. Прежде чем подключать симулируемую систему к реальной сети, нам необходимо убедиться, что у хозяйской машины есть подключение к Интернету. В противном случае симулируемая система также не сможет подключиться к сети. Проверим, что соединение с Интернетом есть у реальной системы. Протестируем Telnet соединение с веб-сервером, например google.com, на 80 порту, с помощью ввода команды `GET /`. Эта команда должна вернуть HTML-содержимое стартовой страницы сервера

```
$ telnet www.google.com 80
Trying 74.125.232.243...
Connected to www.google.com.
Escape character is '^]'.
GET /
HTTP/1.0 302 Found
Location: http://www.google.ru/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=8ebf0b0d9fda87c2:FF=0:TM
    =1358767271:LM=1358767271:S=i7ZpIQ-KLg2H1VXZ;
expires=Wed, 21-Jan-2015 11:21:11 GMT;
path=/; domain=.google.com
Set-Cookie: NID=67=
    ogXTGYQQ5wlyfWupN8EmmGRdfL_7FQ5CNhU3yf2lArXX-04...
expires=Tue, 23-Jul-2013 11:21:11 GMT;
path=/; domain=.google.com; HttpOnly
P3P: CP="This is not a P3P policy! See http://www.
    google.com/support/accounts/bin/answer.py?hl=en&
    answer=151657 for more info."
Date: Mon, 21 Jan 2013 11:21:11 GMT
Server: gws
Content-Length: 218
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="
    text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.ru/">here</A>.
```

```
</BODY></HTML>
Connection closed by foreign host.
```

2. Загрузите Simics со стартовым скриптом `practicum.simics` refchap:target-script:

```
$ ./simics targets/practicum.simics
```

3. На холостом ходу наша симулируемая машина может бежать быстрее реального времени, поэтому время ожидания соединения может пройти быстрее, чем требуется. Для того чтобы избежать этого, необходимо включить режим реального времени.

```
simics> enable-real-time-mode
simics>
```

4. Подкулючите моделируемую систему к реальной сети с помощью команды connect-real-network

```
simics> connect-real-network 192.168.1.100
No ethernet link found, created default_eth_switch0.
Connected practicum.mb.sb.eth_slot to
    default_eth_switch0
Created instantiated 'std-service-node' component '
    default_service_node0'
Connecting 'default_service_node0' to '
    default_eth_switch0' as 192.168.1.1
NAPT enabled with gateway 192.168.1.1/24 on link
    default_eth_switch0.link.
NAPT enabled with gateway fe80::2220:20ff:fe20:2000/64
    on link default_eth_switch0.link.
Host TCP port 4021 -> 192.168.1.100:21
Host TCP port 4022 -> 192.168.1.100:22
Host TCP port 4023 -> 192.168.1.100:23
Host TCP port 4080 -> 192.168.1.100:80
Real DNS enabled at 192.168.1.1/24 on link
    default_eth_switch0.link.
Real DNS enabled at fe80::2220:20ff:fe20:2000/64 on
    link default_eth_switch0.link.
simics>
```

5. Необходимо включить DHCP pool. Переменная `service_node` должна создаться после выполнения команды `connect-real-network`:

```
simics> default_service_node0.dhcp-add-pool pool-size =
    50 ip = 192.168.1.150
simics>
```

6. Запустите симуляцию:

```
simics> continue
running>
```

7. После загрузки симулируемого узла для доступа в Интернет необходимо настроить DHCP.

```
user@master0:~$ sudo dhclient -v eth1
Internet Systems Consortium DHCP Client 4.1.1-P1
Copyright 2004-2010 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/
    dhcp/

[  121.257375] ADDRCONF(NETDEV_UP): eth1: link is not
    ready
[  121.261351] e1000e: eth1 NIC Link is Up 10 Mbps Full
     Duplex, Flow Control: No
ne
[  121.261764] 0000:00:19.0: eth1: 10/100 speed:
    disabling TSO
[  121.262438] ADDRCONF(NETDEV_CHANGE): eth1: link
    becomes ready
Listening on LPF/eth1/00:19:a0:e1:1c:9f
Sending on   LPF/eth1/00:19:a0:e1:1c:9f
Sending on   Socket/fallback
DHCPDISCOVER on eth1 to 255.255.255.255 port 67
    interval 8
DHCPOFFER from 192.168.1.1
DHCPREQUEST on eth1 to 255.255.255.255 port 67
DHCPACK from 192.168.1.1
bound to 192.168.1.150 -- renewal in 1517 seconds.
user@master0:~$
```

8.
```
user@master0:~$ telnet gnu.org 80
Trying 208.118.235.148...
Connected to gnu.org.
Escape character is '^]'.
GET /
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
```

```
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href=''http://savannah.
    nongnu.org/?''>here</a>.</p>
<hr>
<address>Apache/2.2.14 Server at www.nongnu.org Port
    80</address>
</body></html>
Connection closed by foreign host.
```

9. Также с помощью Telnet вы можете посмотреть ASCII-версию «Звездных войн»

```
user@master0:~$ telnet towel.blinkenlights.nl
```

Таблица 8.1

Способы соединения с реальной сетью

	П. Портов	Мост Т	Мост НТ	Хост
Права администратора для конфигурации	нет	да	да	да
Права администратора для запуска	нет	нет	да	нет
Реальный IP	нет	да	да	нет
Поддержка UDP/TCP	да	да	да	да
Ограничение UDP/TCP-портами	да	нет	нет	нет
Поддержка всего IPv4	нет	да	да	да
Поддержка всего Ethernet	нет	нет	да	да
Соединение с хозяином	да	нет	нет	да
Конфигурация хозяина	нет	нет	да	нет

П. Портов: перенаправление портов
Мост Т: создание сетевого моста с трансляцией MAC-адресов
Мост НТ: создание сетевого моста без трансляции MAC-адресов
Хост: соединение с хостом

Список литературы к занятию

1. Ethernet Networking User Guide 4.6 / Wind River. — 2014.
2. *McLoughlin M.* The QCOW2 Image Format. — 2008. — URL: `http://people.gnome.org/~markmc/qcow-image-format.html` (дата обр. 22.10.2012).
3. Simics Hindsight User Guide 4.6 / Wind River. — 2014.
4. Virtual Machine Disk Format (VMDK). — VMware, 2012. — URL: `http://www.vmware.com/technical-resources/interfaces/vmdk.html` (дата обр. 22.10.2012).

9. Использование симуляции для отладки приложений

В ходе данной лабораторной работы слушатели ознакомятся со способами изучения состояния симулируемой системы, отладки приложений, в том числе с символьной информацией, запущенных под управлением гостевой ОС.

Хотя для отладки приложений непосредственно на хозяйской системе существует достаточно широкий набор инструментов, например GDB [1], WinDbg [2] и KB, в ряде сценариев, таких как отладка системного кода BIOS или ОС, симуляция обеспечивает определённые удобства и преимущества, например, отсутствие необходимости использовать отдельную физическую машину для запуска отладчика. На ранних стадиях загрузки системы, когда никакой отладчик ещё не может быть подключен, моделирование является единственным решением.

Подробная информация о поддерживаемых методах отладки в Simics содержится в [3, 4].

9.1. Цель занятия

- Научиться подготавливать модель к символьной отладке приложений.
- Изучить команды инспектирования состояния гостевой системы.

9.2. Ход работы

В приведённом ниже эксперименте мы будет изучать поведение гостевого приложения средствами инспектирования состоя-

ния и символической отладки, присутствующими в Simics. В нашем примере имя этого приложения — `debug_example`.

9.2.1. Подготовка исследуемой программы

Исходный код программы `debug_example.c`, содержащей ошибку, находится в приложении С. Скомпилируйте `debug_example.c` для архитектуры x86, используя флаг `-m32`, и с включением отладочной информации в исполняемый файл, флаг `-g`.

```
gcc -m32 -g -static debug_example.c -o debug_example
```

9.2.2. Подготовка гостевой системы

1. Загрузите Simics со стартовым скриптом `viper-busybox.simics` в конфигурации с процессором класса `core-i7-single`:

   ```
   $ ./simics -e '$cpu_class=core-i7-single' targets/x86-
       x58-ich10/viper-busybox.simics
   ```

2. Включите режим обратного исполнения. Затем запустите симуляцию:

   ```
   simics> enable-reverse-execution
   simics> continue
   ```

3. После того как гостевая система загрузится, авторизуйтесь и примонтируйте хозяйскую файловую систему и скопируйте файл изучаемого приложения внутрь гостя

   ```
   busybox login: root
   ~ # mount /host
   ~ # cp /host/home/user/debug_example ./
   ~ # chmod +x debug_example
   ```

4. Проверьте настройки отладчика запуском симуляции и командой:

   ```
   simics> viper.software.list
   ```

Вывод в консоль должен содержать список запущенных процессов на гостевой системе.

```
Process Binary PID TID
kthreadd 2
migration/0 3
ksoftirqd/0 4
watchdog/0 5
migration/1 6
ksoftirqd/1 7
watchdog/1 8
events/0 9
events/1 10
khelper 11
async/mgr 14
sync_supers 98
bdi-default 100
kblockd/0 102
kblockd/1 103
kseriod 113
rpciod/0 132
rpciod/1 133
khungtaskd 159
kswapd0 160
aio/0 208
aio/1 209
nfsiod 217
crypto/0 223
crypto/1 224
flush-1:0 967
kjournald 957
init 1 1
sh 974 974
httpd 973 973
telnetd 971 971
```

5. Для того чтобы сконфигурировать отладчик, создайте объект символьной таблицы.

```
simics> new-symtable debug_example
Created symbol table 'debug_example'
debug_example set for context viper.cell_context
```

6. Затем загрузите символьную информацию из исполняемого файла в созданный объект.

```
simics> viper.software.track node = debug symtable =
    debug_example
Context debug0 will start tracking debug when it starts
simics> debug_example.load-symbols debug_example
```

9.2.3. Инспектирование состояния гостевой системы

Окно просмотра регистров

Основное состояние центрального процессора хранится в его регистрах. Для просмотра регистров в Simics используется окно **CPU registers**, вызываемое через меню главного окна или через команду консоли `win-cpu-registers`. На рис. 9.1 и 9.2 показаны примеры содержимого двух вкладок этого окна. Также регистры можно просмотреть с помощью команды `pregs`:

```
simics> pregs -all

64-bit mode
rax = 0x00007fff387d56a0            r8  = 0
    x000000000000000b
rcx = 0x0000000000000004            r9  = 0
    x000000000000000f
rdx = 0x00007fff387d5758            r10 = 0
    x000000000000000b
rbx = 0x0000000000400e30            r11 = 0
    x0000000000000008
rsp = 0x00007fff387d5660            r12 = 0
    x0000000000000000
rbp = 0x00007fff387d5670            r13 = 0
    x0000000000000000
rsi = 0x00007fff387d5748            r14 = 0
    x0000000000000000
rdi = 0x0000000000000001            r15 = 0
    x0000000000000000

rip = 0x00000000004004c3, linear = 0x00000000004004c3

eflags = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 = 0
    x00000206
         I V V A V R - N I I O D I T S Z - A - P - C
         D I I C M F   T 0 0 F F F F F F   F   F   F
           P F           P P
```

```
es   = 0x0000, base = 0x00000000, limit = 0x0, attr = 0x0
cs   = 0x0033, base = 0x00000000, limit = 0xffffffff, attr =
     0xa0fb
ss   = 0x002b, base = 0x00000000, limit = 0xffffffff, attr =
     0xc0f3
ds   = 0x0000, base = 0x00000000, limit = 0x0, attr = 0x0
fs   = 0x0063, base = 0x02024860, limit = 0xffffffff, attr =
     0xc0f3
gs   = 0x0000, base = 0x00000000, limit = 0x0, attr = 0x0
tr   = 0x0040, base = 0xffff88007fc0f900, limit = 0x2087,
    attr = 0x8b
ldtr   = 0x0000, base = 0x00000000, limit = 0xffff, attr = 0
    x82
idtr: base = 0xffffffff81b85000, limit = 00fff
gdtr: base = 0xffff88007fc04000, limit = 0007f

efer = 1 1 - 1 -- 1 = 0x00000d01
       N L   L    S
       X M   M    C
       E A   E    E

cr0 = 1 0 0 -- 1 - 1 -- 1 1 0 0 1 1 = 0x80050033
      P C N    A   W    N E T E M P
      G D W    M   P    E T S M P E

cr2 = 0x000000000040c930
cr3 = 0x000000007c01b000

cr4 = 0 - 1 1 0 1 1 1 1 0 0 0 0 = 0x000006f0
      V   0 0 P P M P P D T P V
      M   S S C G C A S E S V M
      X   X F E E E E E   D I E
      E   M X
          M S
          E R
          X
          C
          P
          T

dr0 = 0x0000000000000000 disabled
dr1 = 0x0000000000000000 disabled
dr2 = 0x0000000000000000 disabled
```

```
dr3 = 0x0000000000000000 disabled

dr6 = 0 0 0 —— 0 0 0 0 = 0xffff0ff0
      B B B    B B B B
      T S D    3 2 1 0

dr7 = 00000400Вывод

< опущен>
```

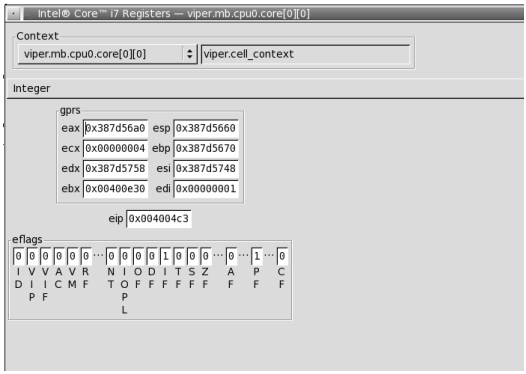

Рис. 9.1. Окно просмотра регистров. Вкладка с регистрами общего назначения

Память системы

Для просмотра различных пространств памяти, присутствующих в моделируемой системе, используется окно **Memory Contents**, вызываемое также командой `win-memory` (рис. 9.3).

9.2.4. Начало отладки

Поставьте точку останова на функции `main()` и запустите симуляцию.

```
simics> break (pos main) -x
Breakpoint 92 set on address 0x80485c6 in 'viper.
    cell_context' with access mode 'x'
simics> continue
```

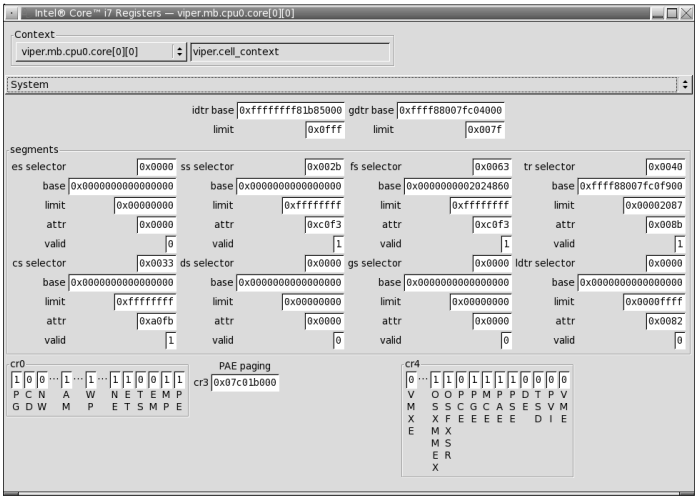

Рис. 9.2. Окно просмотра регистров. Вкладка с системными регистрами

В гостевой ОС запустите исполняемый файл `debug_example`. Симуляция должна остановиться с выводом в консоль:

```
Breakpoint 922 on instruction fetch from 0x80485c6 in viper.
    cell_context.
[viper.mb.cpu0.core[0][1]] cs:0x00000000080485c6 p:0
    x07c17e5c6  lea ecx,4[esp]
Setting new inspection cpu: viper.mb.cpu0.core[0][1]
main (argc=, argv=) at /nfs/ims/home/mvchurik/working_folder
    /debug_example.c:53
53      (file /nfs/ims/home/mvchurik/working_folder/
    debug_example.c not found)

simics> win-source-view
```

Убедитесь, что поле **Source File** окна **Source Code** (рис. 9.4) заполнено. В противном случае загрузите в окно исходный файл кнопкой **Find**.

9.2.5. Окна с информацией для символической отладки

Так как мы предварительно загрузили символьную информацию о приложении, то для его символьной отладки могут быть использованы дополнительные окна, такие как окно исходного кода

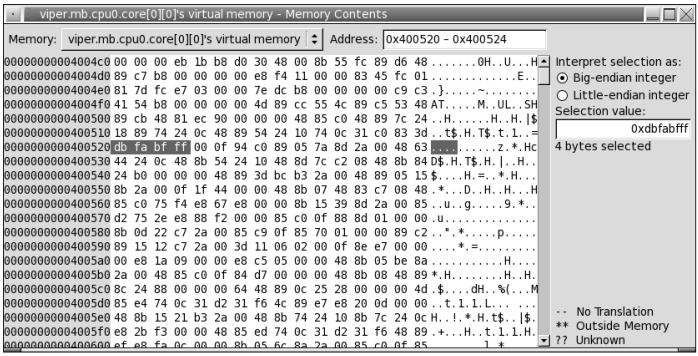

Рис. 9.3. Окно просмотра содержимого памяти системы

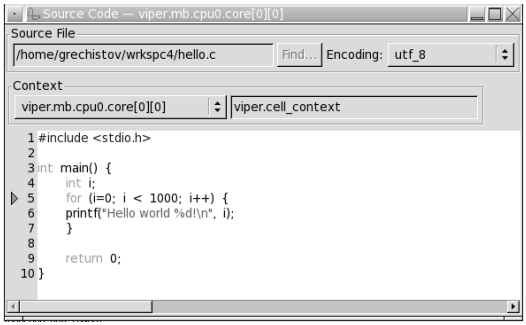

Рис. 9.4. Окно просмотра исходного кода

(рис. 9.4), дизассемблера (рис. 9.5) и стека (рис. 9.6). Информация, содержащаяся в них, также может быть получена с помощью команд `list <имя функции>`, `disassemble <адрес> <количество>` и `stack-trace`.

9.2.6. Управление исполнением программы

Теперь, когда симуляция находится внутри интересующей нас программы, отладчик должен позволять инспектировать состояние её переменных, положение указателя текущей инструкции, а также управлять пошаговым исполнением её операций. Для нас окажутся полезными следующие команды Simics.

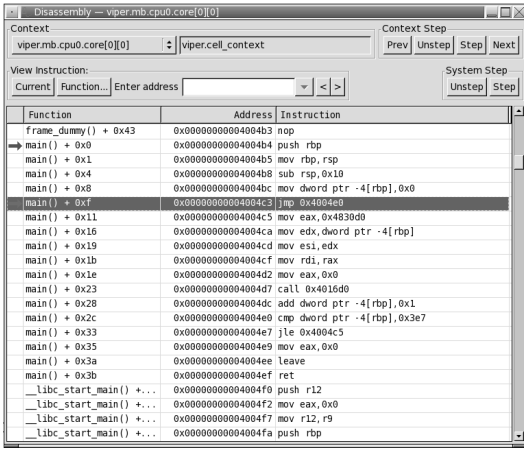

Рис. 9.5. Окно дизассемблера

1. `sym` — получить значение переменной, определённой в текущем контексте гостевой программы.

   ```
   simics> sym argc
   1
   ```

2. `step-line` — выполнить одну строку исходного кода отлаживаемой программы и остановиться.
3. `next-line` — выполнить одну строку исходного кода отлаживаемой программы и остановиться, при этом пропустив

Рис. 9.6. Окно просмотра стека

выполнение подпроцедур, если они вызываются.

4. `pos` — узнать адрес функции или строки кода:

```
simics> pos main
0x804832e
simics> pos debug_example.c:5
0x8048260
```

Кроме задания этих команд, управление исполнением может осуществляться с помощью кнопок **Step** и **Next** окна Disassembly (рис. 9.5).

Использование точек останова

Доступы в память. Самый простой и часто используемый тип — это точка останова по виртуальному адресу исполняемой инструкции:

```
simics> break 0x4004b4
Breakpoint 1 set on address 0x4004b4 in 'viper.cell_context'
     with access mode 'x'
```

Кроме инструкций, точки останова могут быть созданы для регионов данных, при этом попытка гостевой программы обратиться к такому региону вызовет остановку симуляции. Формат команды при этом включает дополнительные флаги `-r` и `-w` для указания, должна ли она реагировать на чтение или запись памяти:

```
simics> break 0x7fffdc2b3d7c -r -w
Breakpoint 2 set on address 0x7fffdc2b3d7c  in 'viper.
    cell_context' with access mode 'rw'
```

Полный формат команды `break` позволяет выбрать любую комбинацию флагов, а также указать длину наблюдаемого диапазона в байтах:

```
break <address> [length] [-r] [-w] [-x]
```

Для того чтобы увидеть данные обо всех установленных точках останова, используется команда `list-breakpoints`.

Исключения. Другой класс событий, который может наблюдаться в отладчике, — это архитектурные исключения. Для установки таких точек используется команда:

```
break-exception name = Page_Fault_Exception
```

Полный список допустимых событий достаточно длинен; нас будут интересовать следующие из них: `Page_Fault_Exception`, `General_Protection_Exception`, `Invalid_Opcode_Exception`.

Точка останова прерывает исполнение симуляции. Если вместо этого желательно просто наблюдать за происходящими событиями, то следует использовать команду `trace-exception`, синтаксис который аналогичен `break-exception`.

9.3. Задания

1. Начать симуляцию и передать исследуемую программу в гостевую систему.
2. Охарактеризовать тип проблемы, возникающий при работе программы.
3. Отладить программу с помощью Simics.

9.4. Вопросы для самостоятельного изучения

1. Для того чтобы любой отладчик, в том числе встроенный в Simics, мог иметь информацию об исходном коде исследуемой программы, информация о нём должна быть доступна ему на этапе отладки. В свою очередь программа должна быть скомпилирована с использованием особенных флагов компиляции. Выясните, какие опции должны быть использованы в случае использования компилятора GCC.
2. Для того чтобы программа, скомпилированная на хозяйской системе, могла быть успешно запущена внутри гостя, требуется соблюсти несколько условий. Одно из них — использование т.н. статической линковки, в случае GCC обозначаемой флагом `-static`. Выясните, зачем был использо-

ван этот флаг. При каких условиях на гостевую и хозяйскую системы можно использовать динамическую линковку?

Список литературы к занятию

1. Debugging with GDB / Free Software Foundation. — 2013. — URL: `http://sourceware.org/gdb/current/onlinedocs/gdb/` (дата обр. 17.01.2013) ; 7.5.50.20130117.
2. *Kuster R.* WinDbg. From A to Z! — Дек. 2007. — URL: `http://windbg.info` (дата обр. 19.01.2013).
3. Simics Analyzer User Guide 4.6 / Wind River. — 2014.
4. Simics Hindsight User Guide 4.6 / Wind River. — 2014.

Приложения

A. Дополнительная информация по работе с Simics

В данное приложение включены сведения о различных приёмах, используемых при ежедневном использовании Simics, не описанные в главах, посвящённых индивидуальным лабораторным работам. Данный материал не заменяет необходимость ознакомления с официальной документацией Simics, а лишь подчёркивает ключевые моменты в ней.

A.1. Обновление workspace

Для получения последних исправлений ошибок в моделях необходимо использовать самую свежую версию базового пакета Simics из установленных на системе. Номера доступных версий можно определить по именам существующих директорий (по умолчанию в `/opt/simics`). В дальнейших примерах последней версией будет считаться **4.6.32**, при этом 4.6 — это основная версия, а последняя цифра — номер минорной версии обновления, который будет использован ниже.

Каждая копия workspace характеризуется *комбинацией* версий пакетов, в ней используемых. Версия пакета Simics Base (№1000) определяет настройки версий остальных пакетов, установленных одновременно с ним. Для того, чтобы увидеть список установленных пакетов и их версии, используйте ключ `-v` при запуске Simics:

```
$ ./simics -v
Simics Base                                              1000
    4.6.32    (4051)
Model Library: Intel Core i7 with X58 and ICH10          2075
    4.6.21    (4051)
```

```
Model Builder                                          1010
    4.6.14     (4051)
Extension Builder                                      1012
    4.6.6      (4042)
```

Также версию Simics можно узнать из командной строки любой уже запущенной симуляции с помощью команды `version`:

```
simics> version
Installed Products:

Model Builder
Extension Builder
Model Library: Intel Core i7 with X58 and ICH10

Installed Packages:

Package                    Nbr  Version  Build  Sources
=======================================================
Simics-Base               1000  4.6.32    4145       No
Model-Builder             1010  4.6.49    4146       No
Extension-Builder         1012  4.6.19    4141       No
x86-Core-i7-X58-ICH10     2075  4.6.59    4146       No
```

В примере сверху базовый пакет имеет версию 4.6.32. Обновления пакетов могут периодически устанавливаться в вашей системе для исправления ошибок в предоставляемых моделей. Однако уже созданные workspace будут по-прежнему использовать старые версии, если для них не выполнить процедуру обновления.

Для обновления workspace, как и для его создания, используется программа `workspace-setup`, находящаяся внутри новой версии базового пакета (версии 4.6.<**minor**>).

```
$ /opt/simics/simics-4.6/simics-4.6.<minor>/bin/workspace-
    setup
Workspace updated successfully
```

A.2. Список часто используемых команд Simics

Меню и кнопки отвлекают, создавая иллюзию простоты творческого процесса.

(А.Б. Шипунов и др. Наглядная статистика. Используем R!)

Команда	**Синонимы**	**Выполняемая функция**
`help <topic>`	`man`	Справка по команде, классу или слову topic
`win-help`		Открыть окно индексируемой справки
`continue`	`c, r, run`	Начать или продолжить симуляцию
`stop`		Остановить симуляцию
`step-cycle [count]`	`sc`	Исполнить count циклов, печатаю следующую инструкцию
`exit`	`quit, q`	Выйти из симулятора
`run-command-file <script.simics>`		Выполнить скрипт Simics
`pregs [-all]`		Распечатать содержимое регистров текущего процессора
`print-time [-all]`	`ptime`	Вывести значение виртуального времени процессора
`win-control`		Открыть окно **Simics Control**
`%<register name>`	`read-reg`	Прочитать содержимое регистра текущего процессора
`%<register name> = <val>`	`write-reg`	Записать значение в регистр текущего процессора
`output-radix <10|16>`		Изменить основание используемой для вывода чисел системы счисления
`break <address>`		Поставить точку останова по адресу
`delete [id]`		Удалить точку останова по её номеру

B. Код целевого скрипта practicum.simics

```
# Script for mipt practicum
load-module pci-components
load-module std-components
load-module x86-comp
load-module x86-nehalem-comp
load-module x58-ich10-comp
load-module memory-comp

add-directory "%simics%/targets/x86-x58-ich10/images/"

$disk_image          = "/share_debian/hpc-images/debian-
    master-2012-05-12.craff"
$cpu_class           = core-i7-single
$freq_mhz            = 3300
$cpi                 = 1
$disk_size           = 20496236544
$rtc_time            = "2008-06-05 23:52:01 UTC"
$memory_megs         = 2048
$text_console        = TURE
$use_acpi            = TRUE
$gpu                 = "accel-vga"
$bios                = "seabios-simics-x58-ich10
    -0.6.0-20110324.bin"
$break_on_reboot     = TRUE
$apic_freq_mhz       = 133
$use_vmp             = TRUE
$spi_flash           = "spi-flash.bin"
$mac_address         = "00:19:A0:E1:1C:9F"
$host_name           = "practicum"

$system = (create-x86-chassis name = $host_name)

### motherboard
$motherboard = (create-motherboard-x58-ich10 $system.mb
```

```
        rtc_time = $rtc_time
        acpi = $use_acpi
        break_on_reboot = $break_on_reboot
        bios = $bios
            mac_address = $mac_address
        spi_flash = $spi_flash)
$southbridge = $motherboard.sb
$northbridge = $motherboard.nb

### processor
$create_processor = ''create-processor-'' + $cpu_class
$create_processor_command = (
        $create_processor
        + '' $motherboard.cpu0''
        + '' freq_mhz = $freq_mhz''
        + '' apic_freq_mhz = $apic_freq_mhz''
        + '' use_vmp = $use_vmp''
        + '' cpi = $cpi'')
$cpu0 = (exec $create_processor_command)
connect $motherboard.socket[0] $cpu0.socket

### memory
$dimm = (create-simple-memory-module $motherboard.memory
                                     memory_megs =
    $memory_megs)
connect $motherboard.dimm[0] $dimm.mem_bus

### GPU
$vga = (create-pci-accel-vga $motherboard.gpu)
connect $northbridge.gpu $vga.connector_pci_bus

### consoles
$console = (create-std-text-graphics-console $system.console
    )
$console.connect keyboard $southbridge
$console.connect $vga

### disk
if not (lookup-file $disk_image) {
    interrupt-script ''Disk image file not found: '' +
    $disk_image
}
$disk = (create-std-ide-disk $system.disk size = $disk_size
    file = $disk_image)
$southbridge.connect ''ide_slot[0]'' $disk
```

```
instantiate-components

#SimicsFS support (add SimicsFS pseudo device)
$hostfs = python ''SIM_create_object('hostfs', 'hfs0', [])''
practicum.mb.phys_mem.add-map $hostfs 0xfed2_0000 16

try {
    win-command-line
} except { echo ''Failed to create GUI''}

script-branch { # Automatize GRUB and login
    local $con = $host_name.console.con
    $con.wait-for-string ''automatically in 5s''
    $con.input ''\n''
    $con.wait-for-string ''login:''
    $con.input ''user\n''
    $con.wait-for-string ''Password:''
    $con.input ''user\n''
}
```

C. Программа debug_example.c

```
/*
 * This program  reads input and converts it to uppercase.
 * It has an intentional bug included that makes it crash on
     certain inputs.
 *
 * Usage: stdin - input string.
 * Compile with gcc -static -g debug_example.c -o
    debug_example
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void read_input(char* in) {
    char symbol;
    in[0] = 0; // initialize the string with zero length
    while((symbol = getchar()) != EOF) {
        *in++ = symbol;
    }
    *in = 0; // close the string
}

void convert_to_uppercase(char * in, char *out) {
    int i;
    for (i =0; i <= strlen(in); i++) {
        if (isalpha(in[i]))
            out[i] = toupper(in[i]);
        else
            out[i] = in[i];
    }
}

int main(int argc, char** argv) {
    char input[32], *output;
```

```
    read_input(input);
    convert_to_uppercase(input, output);

    printf("%s\n", output);
    return 0;
}
```

D. Установка и лицензирование Simics

В данное приложение включена информация по лицензированию и установке Simics в учебной лаборатории. Наиболее полная информация по данному вопросу содержится в документе «Simics Installation Guide» [3], который идёт в поставке со всеми пакетами Simics (файл `installation-guide.pdf`).

Приводимые ниже инструкции были собраны для Simics версии 4.6 для хозяйской системы Linux 64 бит, рекомендуемой для всех пользователей. Для ОС Linux 32 бит инструкции изменяются незначительно; для ОС Windows они применимы после учёта особенностей графического процесса инсталляции.

D.1. Академическая программа Wind River Simics

Компания Intel предлагает Simics бесплатно для некоммерческих исследований и обучения в выбранных университетах через академическую программу Intel Simics Academic Program. Для включения нового университета в эту программу необходимо согласие на поддержку начинания одного ментора — сотрудника Intel.

Дополнительная информация об истории и статусе академической программы Simics [1].

D.1.1. Условия использования

Использование Simics по академической программе должно строго соответствовать условиям соглашения, т.е. быть ограничено учебной и/или некоммерческой научно-исследовательской

деятельностью. В случае возникновения необходимости проведения коммерческих исследований или разработок необходимо обратиться к представителям Wind River для получения нового соглашения и другого типа лицензии.

Держатель лицензии от участвующего университета обязан донести эту информацию до всех пользователей инсталляции и контролировать выполнение ими условий соглашения, в том числе с помощью административных и технических мер.

Подробные детали об условиях и ограничениях академической программы содержатся в документе «Intel Academic SLA», поставляемом с копией Simics для университетов.

D.2. Установка файлов и запрос лицензии

D.2.1. Пакеты

Simics распространяется в формате пакетов — набора файлов, реализующих одну или несколько типов моделируемых систем или функциональность самого симулятора. Каждый пакет имеет свой фиксированный номер. Пакет №1000 — это Simics Base, содержащий базовую функциональность симулятора. Все остальные пакеты являются дополнениями к нему.

Дистрибутив пакета — это файл с именем вида `simics-pkg-1000-4.6.34-linux64.tar`. Здесь 1000 — номер пакета, 4.6.34 — версия пакета, linux64 — архитектура хозяйской системы. Каждый дистрибутив каждого пакета зашифрован собственным ключом, состоящим из 32 символов. Дистрибутивы и их ключи получите у спонсора вашей академической программы.

Для установки всех необходимых файлов выполняется следующая процедура. Часть команд может потребовать наличия прав администратора.

1. Разархивируйте все пакеты *.tar:

   ```
   $ for f in simics-pkg-*.tar; do tar xf $f; done
   ```

2. В созданной директории `simics-4.6-install` запустите скрипт установки:

```
$ cd simics-4.6-install
# ./install-simics.pl
```

3. Введите ключи шифрования для каждого номера пакета, который планируется установить:

```
-> Looking for Simics packages in current directory...
Enter a decryption key for package-1000-4.6.34-linux64.
    tar.gz.tf,
or Enter to [Abort]: введите или скопируйте ключ
```

4. На вопрос, какие из пакетов требуется установить, ответьте «All packages»:

```
install-simics can install the following packages:
 Number  Name          Type    Version  Host
    Package
   1     Simics-Base   simics  4.6.34   linux64
    package-1000
   2     Eclipse       addon   4.6.16   linux64
    package-1001
   3     All packages
Please enter the numbers of the packages you want to
    install, as in "1 4 3"
Package numbers, or Enter to [Abort]:  3
```

5. На вопрос о директории назначения введите абсолютный путь или оставьте значение по умолчанию:

```
Enter a destination directory for installation, or
    Enter
for [/opt/simics]:  Путь[ установки или Enter]
```

6. Подтвердите начало установки, выбрав «y».

7. При введении правильных ключей дистрибутивы будут расшифрованы и установлены в указанную при установке директорию — в ней должны появиться подпапки с файлами из пакетов Simics.

```
-> Decrypting package-1000-4.6.34-linux64.tar.gz.tf
-> Testing package-1000-4.6.34-linux64.tar.gz
-> Installing package-1000-4.6.34-linux64.tar.gz
```

```
-> Decrypting package-1001-4.6.16-linux64.tar.gz.tf
-> Testing package-1001-4.6.16-linux64.tar.gz
-> Installing package-1001-4.6.16-linux64.tar.gz

==============================

install-simics has finished installing the packages and
     will now
configure them.

No previous Simics installation was found. If you wish
    to configure
the newly installed Simics from a previous installation
     not found by
this script, you can do so by running the 'addon-
    manager' script in
the Simics installation with the option --upgrade-from:
    ./bin/addon-manager --upgrade-from /previous/
    install/

install-simics has installed the following add-on
    package:
   Eclipse  4.6.16  /opt/simics/simics-eclipse-4.8.26
```

8. На вопрос о регистрации расширений (*англ.* add-on) ответьте «y»:

```
Do you wish to make these add-on packages available in
Simics-Base 4.6.34? (y, n) [y]:  y
```

После успешного завершения файлы Simics были скопированы на ваш диск. Следующий шаг — получение лицензии для их запуска. Он описывается далее.

D.2.2. Получение lmhostid

Для получения файла лицензии необходимо сгенерировать и передать число, так называемый lmhostid.

Об именовании сетевых интерфейсов. На момент написания данного материала утилиты из состава Simics не поддерживали получение корректного lmhostid на системах, использующих

схему «стабильного именования» сетевых интерфейсов. Вместо традиционных для Linux имён `eth0`, `eth1` и т.д. сетевым картам выдаются имена, зависящие от производителя и физического расположения в системе, например, `enp6s0`, `enp11s0`. Методы решения данной проблемы описаны в [2].

1. Установите пакет `lsb-core` на системе. Для Debian и Ubuntu это выполняется командой:

   ```
   # apt-get install lsb-core
   ```

2. Для получения lmhostid на сервере, *который будет использоваться для запуска демона лицензий*, выполните команду:

   ```
   $ /opt/simics/simics-4.6.34/flexnet/linux64/bin/lmutil
       lmhostid
   lmutil - Copyright (c) 1989-2011 Flexera Software, Inc.
        All Rights Reserved.
   The FlexNet host ID of this machine is ""602fe934a369
       422fe934a36c ""
   Only use ONE from the list of hostids.
   ```

   Выданное число (в примере выше «602fe934a369») — это lmhostid. Если чисел выдано несколько, то используйте только одно из них.

D.2.3. Заполнение заявки

Для получения пакетов, ключей к ним, а также подписания лицензионного соглашения об участии в академической программе Wind River Simics обратитесь к вашему спонсору или пройдите процедуру, описанную в [1].

D.3. Настройка сервера лицензий

Сервер лицензий — отдельная программа, запущенная на постоянно включенном компьютере и определяющая, какие модели и в каком количестве будут доступны в компьютерном классе.

D.3.1. Файл лицензии

Получаемый от производителя файл лицензии — это текстовый документ, содержащий информацию о сроке действия, ограничениях количества одновременно запускаемых копий и поддерживаемых расширениях приложения. Пример содержимого для начала этого файла:

```
# Simics 4.6 licence for the Simics Academic Program
#
# University:        Moscow Institute of Physics and
    Technology
# Contact:           academic.contact@university.edu
# Sponsor:           sponsor.contact@sponsor.com
# Licensing Contact: licencing.contact@licencer.com
#
SERVER  lic.university.edu lmhostid
VENDOR simics /opt/simics/simics-4.6/simics-4.6.100/flexnet/
    linux64/bin
#
FEATURE simics simics 4.6 28-feb-2014 50 BD47D265FA68 \
VENDOR_STRING=intel;academic HOSTID=ANY BORROW TS_OK \
SIGN=''0441 6AFA 450C BDBE E4D7 E125 1042 EEFF 04B5 767A ABCD
     \
        5088 80DB D912 292E 4FD5 22DD 22D0 D55F 5B25 4818''
<...>
```

Не изменяйте никаких строк этого файла, кроме имени сервера лицензий (строка с `SERVER`) и пути к каталогу с файлом вендор-демона (строка `VENDOR`, должна указывать на положение файлов с именами `lmgrd` и `simics`). Сохраните копию файла в надёжном месте. Запишите дату окончания действия лицензии для последующей своевременной инициации процедуры её обновления.

D.3.2. Запуск сервера

Для запуска серевера лицензий используется программа `lmgrd`, поставляемая с базовым пакетом[1]. Её расположение: `<simics-base>/flexnet/linux64/bin/lmgrd`.

[1]Варианты этой программы, полученные из других источников, не рекомендуются и не поддерживаются.

Пример последовательности команд для ручного запуска `lmgrd`:

```
$ cd /opt/simics/simics-4.6/simics-4.6.100/flexnet/linux64/
    bin/
$ ./lmgrd -c /opt/simics/simics-4.6/simics-4.6.100/licenses/
    simics.lic
```

В данном случае процесс остаётся в консоли (не уходит в фоновый режим) и печатает диагностику в консоль. Для остановки достаточно послать ему сигнал с помощью клавиш Ctrl-C.

Для автоматического запуска и остановки процесса `lmgrd` при включении и выключении системы рекомендуется использовать init-скрипт в стиле инициализации SysV. Пример такого скрипта: `https://gist.github.com/grigory-rechistov/11142235`, также он приведён в секции D.5.

D.3.3. Проверка работоспособности

Запустите демон лицензии. Затем запустите копию Simics из любого workspace на этой же системе. В случае успеха приложение успешно откроет своё окно или покажет приглашение командной строки `simics>`.

D.4. Расположение файлов и сервера лицензий при подключении по сети

По умолчанию все файлы Simics размещаются в директории `/opt/simics`. Если необходимо обеспечить запуск симулятора на нескольких компьютерах, подсоединённых по сети, рекомендуется разместить эти файлы установки в файловой системе, доступной по сети, например, по протоколам NFS или CIFS. Таким образом, клиентские машины смогут переиспользовать структуру инсталляции без необходимости её копирования на локальные диски, что упростит её поддержку и выполнение обновлений. Настройка сетевой файловой системы выходит за рамки данного руководства; необходимую информацию можно найти, например, в [4].

D.4.1. Финальный вид инсталляции

На рис. D.1 приведена рекомендуемая схема соединения систем и расположения служб для работы Simics на всех компьютерах учебного класса или лаборатории. В данном примере сервер для запуска демона лицензий отделён от сервера общих файлов; на практике они могут быть одной и той же системой.

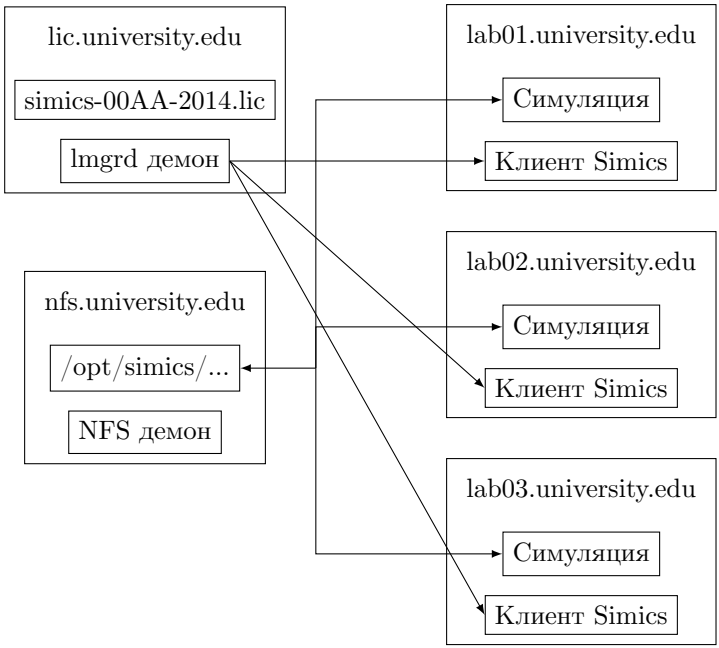

Рис. D.1. Расположение и функции узлов инсталляции Simics

D.4.2. Решение возникших проблем

Следует отметить, что процесс `lmgrd` рекомендуется запускать из-под непривилегированного пользователя, т.е. не root. Кроме того, он должен быть в состоянии найти файл с т.н. программой vendor-daemon, которая для Simics называется `simics` и находится в той же директории, что и `lmgrd`. Поэтому запуск должен

демона происходить из этой же директории, или же она должна быть внесена в переменную окружения $PATH.

Ниже описаны некоторые часто встречающиеся неполадки и способы их устранить.

Невозможно стартовать lmgrd. Проверьте флаги файла `lmgrd` на признак исполняемости; при необходимости выставите его с помощью `chmod +x lmgrd`.

lmgrd выходит сразу после запуска. Возможные причины: 1) одна копия `lmgrd` уже запущена; 2) не найден файл лицензии; 3) не найден файл с вендор-демоном `simics`; 4) запуск на системе с неправильным lmhostid; 5) Файл блокироки /tmp/locksimics существует и недоступен на запись. Прочитайте вывод об ошибке, оставшийся после выхода демона, и исправьте указанную в нём причину.

Нет подключения к демону лицензии локально. При этом Simics на некоторое время зависает, а затем сообщает код ошибки подключения к серверу лицензий. Проверьте, что настройки Simics указывают на правильный файл.

Simics зависает в самом начале загрузки. Проверьте, что переменная окружения $DISPLAY или настроена правильно, или неопределена (при запусках без графического интерфейса).

Нет подключения к демону лицензии по сети. Проверьте, что между машинами с работающим `lmgrd` и запускаемым Simics нет фаервола. Откройте порты 28000 и 28001 в фаерволе, если он присутствует и необходим.

Демон лицензий не работает после перезагрузки сервера. Проверьте, что процесс демона запущен. Проверьте, что файл /tmp/locksimics не существует. Настройте корректное завершения процесса `lmgrd` при остановке ОС сервера лицензий.

Исчерпано число лицензий. Выключите излишние, ненужные запущенные симуляции.

D.5. init скрипт для старта и остановки демона лицензий

Данный скрипт `lmgrd-simics` должен быть размещён в /etc/inid.d с правом исполнения, затем для Debian-систем должен быть включён с помощью команды:

```
# update-rc.d lmgrd-simics defaults
```

Он доступен по ссылке `https://gist.github.com/grigory-rechistov/11142235`.

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          lmgrd-simics
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Control Flexera lmgrd license daemon
    for Simics installation
# Description:       Control start/stop of lmgrd entry for
    Simics
#
### END INIT INFO

# Author: Grigory Rechistov (<grigory.rechistov@phystech.edu
    >)
#

# Do NOT ''set -e''

# PATH should only include /usr/* if it runs after the
    mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC=''lmgrd for Simics''
SIMICSDIR=/opt/simics/simics-4.6/simics-4.6.100
LICENSEFILE=/opt/simics/simics-4.6/simics-4.6.100/licenses/
    simics.lic # change to your license file
NAME=lmgrd
VENDORDAEMON=simics
DAEMONDIR=$SIMICSDIR/flexnet/linux64/bin
SCRIPTNAME=/etc/init.d/$NAME
DAEMON=$DAEMONDIR/$NAME
```

```
PIDFILE=/var/tmp/$NAME.pid
LOCKFILE=/var/tmp/locksimics
LOGFILE=/var/tmp/lmgrd-simics.log

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.2-14) to ensure that this file is
    present
# and status_of_proc is working.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
  # Return
  #   0 if daemon has been started
  #   1 if daemon was already running
  #   2 if daemon could not be started
  start-stop-daemon --start -c daemon:daemon --make-pidfile
    --pidfile $PIDFILE -d $DAEMONDIR --exec $DAEMON -- -c
    $LICENSEFILE -l +$LOGFILE || return 2
  pidof $NAME > $PIDFILE # This is lame; but lmgrd about
    itself does not create anything.
}

#
# Function that stops the daemon/service
#
do_stop()
{
  # Return
  #   0 if daemon has been stopped
  #   1 if daemon was already stopped
  #   2 if daemon could not be stopped
  #   other if a failure occurred
```

```
  start-stop-daemon --stop --retry=TERM/30/KILL/5 --pidfile
    $PIDFILE --name $NAME
  RETVAL="$?"
  [ "$RETVAL" = 2 ] && return 2
  # Many daemons don't delete their pidfiles when they exit.
  rm -f $PIDFILE
        rm -f $LOCKFILE
  return "$RETVAL"
}

#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
  #
  # If the daemon can reload its configuration without
  # restarting (for example, when it is sent a SIGHUP),
  # then implement that here.
  #
  start-stop-daemon --stop --signal 1 --quiet --pidfile
    $PIDFILE --name $NAME
  return 0
}

case "$1" in
  start)
  [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "
    $NAME"
  do_start
  case "$?" in
    0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
    2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
  esac
  ;;
  stop)
  [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "
    $NAME"
  do_stop
  case "$?" in
    0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
    2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
  esac
  ;;
  status)
       status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
```

```
      ;;
  #reload|force-reload)
  #
  # If do_reload() is not implemented then leave this
    commented out
  # and leave 'force-reload' as an alias for 'restart'.
  #
  #log_daemon_msg "Reloading $DESC" "$NAME"
  #do_reload
  #log_end_msg $?
  #;;
  restart|force-reload)
  #
  # If the "reload" option is implemented then remove the
  # 'force-reload' alias
  #
  log_daemon_msg "Restarting $DESC" "$NAME"
  do_stop
  case "$?" in
    0|1)
    do_start
    case "$?" in
      0) log_end_msg 0 ;;
      1) log_end_msg 1 ;; # Old process is still running
      *) log_end_msg 1 ;; # Failed to start
    esac
    ;;
    *)
      # Failed to stop
    log_end_msg 1
    ;;
  esac
  ;;
  *)
  #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force
    -reload}" >&2
  echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-
    reload}" >&2
  exit 3
  ;;
esac
:
```

Литература

1. *Engblom J.* Academic Simics. — 2010. — URL: http://blogs.windriver.com/engblom/2010/07/academic-simics.html (дата обр. 02.03.2014).
2. Predictable Network Interface Names. — Freedesktop.org, 2014. — URL: http://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/.
3. Simics Installation User Guide 4.8 / Wind River. — 2013.
4. *Сгибнев М.* Настройка NFS сервера и клиента в Debian Lenny. — 2009. — URL: http://www.opennet.ru/tips/info/2061.shtml (дата обр. 22.03.2014).

Список TODO

Даннаяя секция предназначена для напоминания авторам, какие задачи по улучшению содержимого книги необходимо выполнить. Всем остальным просьба не обращать внимания.