

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
ПО ПРОГРАММНОМУ
МОДЕЛИРОВАНИЮ
УЧЕБНОЕ ПОСОБИЕ

Версия 1.1β
5 февраля 2014 г.

Copyright © 2011, 2012, 2013, 2014 Grigory Rechistov, Yulyugin Evgeny.



Текст данного варианта произведения распространяется по лицензии Creative Commons Attribution-NonCommercial-ShareAlike (Атрибуция — Некоммерческое использование — На тех же условиях) 4.0 весь мир (в т.ч. Россия и др.). Чтобы ознакомиться с экземпляром этой лицензии, посетите <http://creativecommons.org/licenses/by-nc-sa/4.0/> или отправьте письмо на адрес Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Все зарегистрированные торговые марки, названия и логотипы, использованные в данных материалах, являются собственностью их владельцев. Представленная точка зрения отражает личное мнение авторов, не выступающих от лица какой-либо организации.

Содержание

Введение	6
1 Компьютерная симуляция	6
2 Предварительные замечания	7
3 Обозначения	8
1 Знакомство с Simics	10
1.1 Цель занятия	10
1.2 Ход работы	10
1.3 Запуск симулятора Simics. Элементы интерфейса . .	17
1.4 Задания	23
2 Концепции симуляции	25
2.1 Цель занятия	25
2.2 Ход работы	25
2.3 Задания	30
3 Связь симуляции с внешним миром	32
3.1 Цель занятия	32
3.2 Диски	32
3.3 Копирование файлов внутрь моделируемой системы	34
3.4 Ход работы	37
4 Использование симуляции для изучения системы	45
4.1 Цель занятия	45
4.2 Ход работы	45
4.3 Задания	55
4.4 Вопросы для самостоятельного изучения	55
A Дополнительная информация по работе с Simics	59
A.1 Обновление workspace	59
A.2 Список часто используемых команд Simics	61
A.3 Лицензия Simics	62
B Скрипт для отладки	63

C	Код целевого скрипта <code>practicum.simics</code>	65
D	Программа <code>debug_example.c</code>	68

DRAFT

Предисловие

В настоящем практикуме описываются лабораторные работы по курсу «Основы программного моделирования ЭВМ», проводившиеся в Московском физико-техническом институте на базе вычислительного кластера лаборатории суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур iSCALARE.

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.

Данное пособие подготовили сотрудники лаборатории суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур МФТИ и образовательной лаборатории МФТИ-Интел: Юлюгин Е.А., Речистов Г.С., Щелкунов Н.Н., Гаврилов Д.А.

Введение

1. Компьютерная симуляция

Использование компьютерного моделирования в процессе проектирования позволяет заметно сократить среднее 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, что позволяет совмещать многие процессы параллельно во времени, так как часть из них может быть начата гораздо раньше, чем это было возможно ранее, и эффективно сокращать цикл разработки (рис. 1).

Программное обеспечение для имитационного моделирования используется для тестирования и исследования производительности, функциональных 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.

Задача данного цикла лабораторных работ — познакомить слушателей с новейшими достижениями в области компьютерной симуляции, связанными с эффективным созданием моделей, максимально точно представляющих аппаратные средства и при этом имеющих приемлемую скорость работы, получаемую благодаря эффективному задействованию имеющихся вычислительных ресурсов. Изучение проводится на программном продукте Wind River® Simics (в дальнейшем просто 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, почти все рассматриваемые вопросы

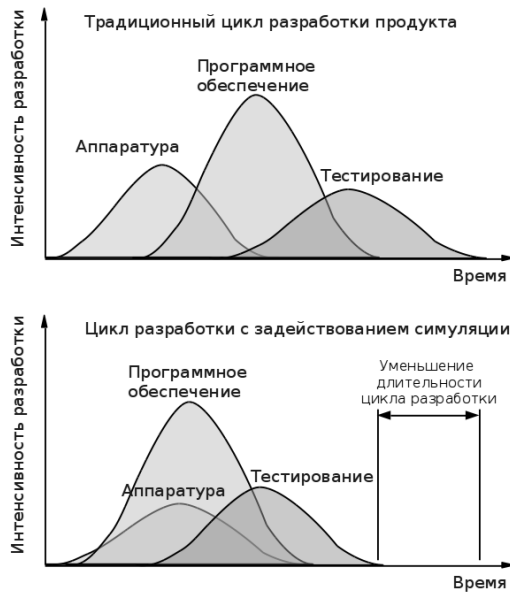


Рис. 1. Сдвиг влево — возможность совместить моменты начала отдельных стадий проектирования новых цифровых систем, таким образом сокращая цикл разработки и уменьшая время вывода их на рынок

включают в себя идеи и концепции, общие для разных симуляторов.

2. Предварительные замечания

Для максимально эффективного усвоения материала данного пособия читателю рекомендуется иметь начальные знания по архитектуре ЭВМ. Желательно понимание общих принципов работы операционных систем, а также знакомство как минимум с одним языком программирования.

3. Обозначения

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.
- *Наклонный 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 Simics, имеют префикс **simics>**:

```
simics> list-objects
```

- Команды, которые необходимо вводить в строку приглашения Bash, имеют префикс **\$** для обычного пользователя или **#** для команд, выполняемых из-под аккаунта суперпользователя:

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

- Обязательные аргументы команд указаны в угловых скобках, необязательные — в квадратных, например:

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

При использовании терминов, заимствованных из английского языка и не имеющих известных авторам общепринятых пере-

водов на русский язык, в скобках после них будут указываться оригинальные термины.

DRAFT

1. Знакомство с Simics

1.1. Цель занятия

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

- Доступ на удалённую Unix систему с помощью SSH/VNC.
- Запуск симулятора. Элементы интерфейса.
- Сценарии работы. Гостевые системы.
- Процесс загрузки ОС Linux внутри симулятора.
- Базовые операции: остановка модели, инспектирование состояния, сохранение на диск.

1.2. Ход работы

1.2.1. Доступ на удалённую систему через SSH

SSH (*англ.* Secure SHell — «безопасная оболочка») [2] — сетевой протокол прикладного уровня, позволяющий производить удаленное управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). SSH-клиенты и SSH-серверы доступны для большинства сетевых операционных систем.

Ниже будут описаны способы использования SSH из-под операционных систем Linux и Windows.

Linux

Откройте терминал и наберите команду, заменив слово **user** на свой логин:

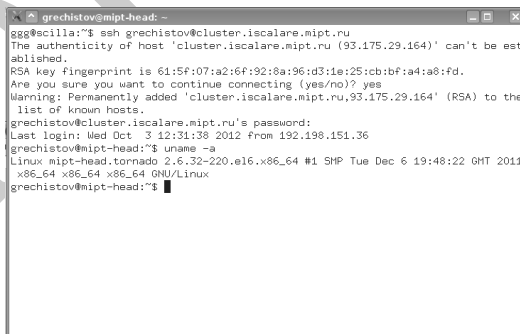
```
$ ssh user@cluster.iscalare.mipt.ru
```

После этого система спросит у вас пароль. Введите его. Внимание! Вводимые символы не отображаются на экране. При удачном вводе пароля вы увидите сообщения приветствия командной строки (рис. 1.1).

Windows

Для того чтобы подключаться к удаленному серверу из системы Windows, необходимо воспользоваться SSH-клиентом, например, PuTTY [4] (мы рекомендуем использовать именно его):

1. Запустите PuTTY.
2. В главном окне программы в поле **Host Name (or IP address)** введите строку `user@cluster.iscalare.mipt.ru`, заменив `user` на свой логин (рис. 1.2).
3. Порт оставьте равным 22.
4. Сохраните сессию под некоторым именем, чтобы избежать необходимости повторного ввода в дальнейшем.
5. После открытия окна вам будет предложено ввести пароль. Введите его и нажмите **Enter**. Внимание: вводимые символы не отображаются на экране!
6. При удачном входе вы получите приглашение командной строки (рис. 1.1).



```
grechistov@mipt-head: ~
ssh@gcilla:~$ ssh grechistov@cluster.iscalare.mipt.ru
The authenticity of host 'cluster.iscalare.mipt.ru (93.175.29.164)' can't be est
ablished.
RSA key fingerprint is 61:5f:07:a2:6f:92:8a:96:d3:1e:25:cb:bf:a4:a8:fd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'cluster.iscalare.mipt.ru,93.175.29.164' (RSA) to the
list of known hosts.
grechistov@cluster.iscalare.mipt.ru's password:
Last login: Wed Oct 3 12:31:38 2012 from 192.198.151.36
grechistov@mipt-head:~$ uname -a
Linux mipt-head.tornado 2.6.32-220.el6.x86_64 #1 SMP Tue Dec 6 19:48:22 GMT 2011
x86_64 x86_64 x86_64 GNU/Linux
grechistov@mipt-head:~$
```

Рис. 1.1. Открытие сессии SSH

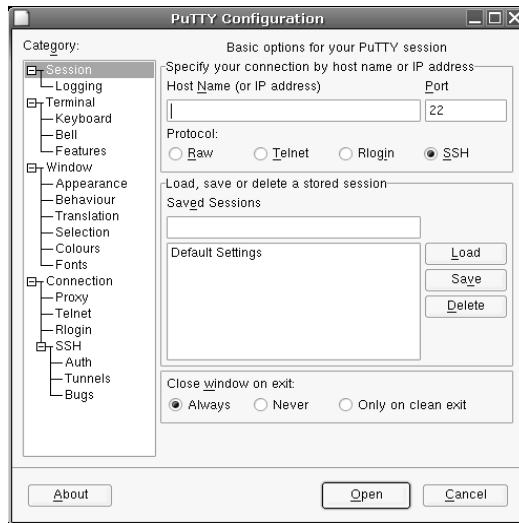


Рис. 1.2. Окно PuTTY

1.2.2. Создание VNC-сессии

VNC (*англ.* Virtual Network Computing) [1] — система удалённого доступа к рабочему столу компьютера, использующая протокол RFB (*англ.* Remote Frame Buffer). VNC состоит из двух частей: клиента и сервера. Сервер — программа, предоставляющая доступ к экрану компьютера, на котором она запущена. Клиент (*англ.* viewer) — программа, получающая изображение экрана с сервера и взаимодействующая с ним по протоколу RFB. Управление осуществляется путём передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть. Система VNC платформонезависима: VNC-клиент, запущенный на одной операционной системе, может подключаться к VNC-серверу, работающему на любой другой ОС. Существуют реализации клиентской и серверной части практически для всех операционных систем, в том числе и для Java (включая мобильную платформу J2ME). К одному VNC-серверу одновременно могут подключаться множественные клиенты. Наиболее популярные способы

использования VNC — удалённая техническая поддержка и доступ к рабочему компьютеру из дома.

Использование VNC для доступа к удалённому серверу

Для создани VNC-сессии необходимо выполнить следующую последовательность действий:

1. Зайдите на удалённый сервер, как описано в секции 1.2.1.
2. Запустите команду для создания графического сервера с необходимым вам разрешением экрана:

```
$ vncserver -geometry 1280x1024
```

Рекомендуется выбирать разрешение, равное или меньшее физического разрешения экрана вашей системы.

3. При необходимости (при первом запуске) задайте пароль (он не обязан совпадать с SSH-паролем). Пароль можно изменить впоследствии с помощью команды `vncpasswd`. Если вы забыли его, то удалить его можно, удалив файл `$HOME/.vnc/passwd`.
4. Запомните номер сессии (число после двоеточия в выводе команды). Например, из вывода команды ниже видно, что только что созданный номер сессии равен 4.

```
$ vncserver
New 'mipt-head.tornado:4 (user)' desktop is mipt-head.
tornado:4
Starting applications specified in /home/user/.vnc/
xstartup
Log file is /home/user/.vnc/mipt-head.tornado:4.log
```

Примечание. Допускается иметь более одной VNC-сессии, при этом каждая будет иметь свой идентификатор (номер порта после двоеточия). Однако имейте в виду, что каждая из них потребляет небольшое количество общих системных ресурсов. Файл журнала соответствующей сессии, например `mipt-head.tornado:4.log`, полезен при анализе проблем при запуске, например, когда открываемый впоследствии графический экран пустой или VNC-сессия вообще не запустилась.

Замечание о необходимости туннелирования VNC

В настоящее время на `cluster.iscalare.mipt.ru` закрыты все внешние порты, кроме SSH (22). Поэтому для VNC-соединения, использующего порты в диапазоне 5901–5999, необходимо создавать т.н. **SSH-туннель**. Теоретически для локального конца туннеля можно выбирать любой порт (из непривилегированного диапазона 1001–65535), однако для избежания путаницы рекомендуется выбирать его равным номеру удалённого конца; в последующих примерах мы придерживаемся этого правила.

Отключение VNC-сессии

Обычно закрывать сессию VNC не требуется — достаточно отключиться от неё. Затем можно возобновить работу, используя VNC-клиент. При необходимости уничтожение десктопа и всех запущенных приложений может быть выполнено командой

```
$ vncserver -kill :4
```

Если необходимо уничтожить все свои VNC-сессии, используйте команду

```
$ killall Xvnc
```

Linux

Для начала нам необходимо установить VNC-клиент на наш рабочий компьютер (если он еще не установлен). Для установки клиента на персональную ЭВМ необходимо воспользоваться пакетным менеджером. Пример для Ubuntu/Debian:

```
$ sudo apt-get install xtightvncviewer
```

Кроме того, допускается использовать любой клиент, поддерживающий протокол VNC: RealVNC, TighVNC, Remmina, Vinagre, Krdp...

Затем используйте следующие опции **ssh** для создания туннеля (в примере ниже — для порта 5904, имени пользователя **user**)

```
$ ssh -L 5904:127.0.0.1:5904 -l user cluster.iscalare.mipt.ru
```

Если вы используете TightVNC, то для запуска VNC-сессии вам необходимо воспользоваться командой, указав после двоеточия номер вашей VNC-сессии (в данном примере номер VNC-сессии равен 4).

```
$ vncviewer localhost:4
```

После этого система спросит у вас пароль. Введите его. Внимание! Вводимые символы не отображаются на экране. При удачном вводе пароля откроется окно VNC-сессии, готовое к работе.

Windows

Для Windows мы рекомендуем использовать RealVNC. Инструкцию по установке и дополнительную информацию можно найти в [5].

Допускается использовать любой клиент, поддерживающий протокол VNC: RealVNC, TightVNC, UltraVNC...

При открытии SSH из Putty дополнительно на вкладке Tunnels создаём проброс порта вашей VNC-сессии. В данном примере (рис. 1.3) это экран VNC :4, что соответствует TCP-порту 5904.

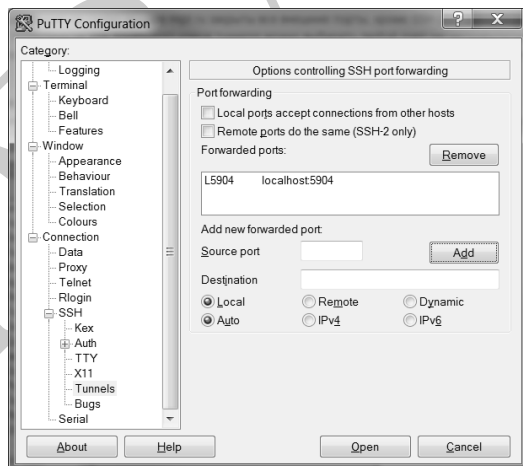


Рис. 1.3. Создание ssh туннеля с помощью PuTTY

Для подключения к удаленному рабочему столу необходимо воспользоваться VNC-клиентом (в нашем примере это RealVNC). Выполните следующую последовательность действий:

1. Откройте VNC viewer.
2. В поле VNC Server укажите **localhost:4** (рис. 1.4), заменив число 4 после двоеточия на номер вашей VNC-сессии.

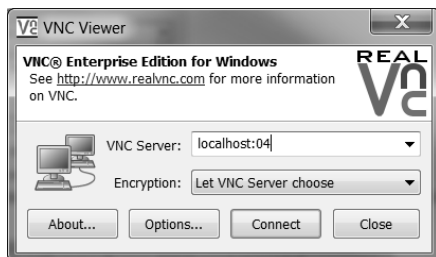


Рис. 1.4. Окно RealVNC

3. Нажмите **Connect**.
4. Система запросит ваш пароль от VNC-сессии. Введите его.
5. При удачном вводе пароля откроется окно VNC-сессии готовое к работе.

1.2.3. Базовые команды Linux

В этой секции перечислены команды Linux, которые понадобятся для выполнения данной лабораторной работы:

- Команда **man** предназначена для формирования и вывода справочной информации о команде `<command>`:

```
$ man <command>
```
- Команда **ls** выводит информацию о `<file>` (по умолчанию выводится информация о текущей директории).

```
$ ls <file>
```
- Команда **cd** меняет текущую рабочую директорию на `<directory>`.


```
$ cd <directory>
```

- Команда `mkdir` создает директорию `<directory>` в текущей, если она до этого не существовала.

```
$ mkdir <directory>
```

- Команда `cat` направляет `<file>` на стандартный вывод. Чаще всего это означает распечатывание его содержимого на экране.

```
$ cat <file>
```

- Команда `lspci` показывает список всех PCI-устройств.
- Команда `mount [node] <folder>` монтирует файловую систему; вызванная без аргументов, она перечисляет уже смонтированные системы.
- Команда `umount <folder>` предназначена для отмонтирования файловой системы.

Больше информации о командах Linux вы можете найти в мануалах, воспользовавшись командой **man**, описанной выше.

1.3. Запуск симулятора Simics. Элементы интерфейса

Wind River® Simics [3] — это быстрый, функционально точный полноплатформенный симулятор. Simics позволяет создать виртуальное окружение, в котором любая электронная система, начиная с одной платы и заканчивая целыми многопроцессорными и многоядерными системами, может быть определена, разработана и запущена.

1.3.1. Создание и обновление Simics workspace

Инсталляция Simics может быть использована совместно множеством пользователей и предпочтительно должна быть доступна только для чтения. Рабочее окружение (далее «workspace») — это персональная «копия» установки, в которой Simics хранит ваши собственные настройки среды и моделей и в которой

вы можете держать ваши рабочие файлы. Каждый пользователь может иметь более одного workspace, и все они могут использоваться одновременно и независимо друг от друга.

Создание workspace

Для создания нового workspace вам необходимо выполнить следующую последовательность действий:

1. Создайте директорию, в которой будет находиться ваш персональный workspace (рекомендуется использовать локацию внутри домашней директории пользователя) и перейдите в неё:

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

2. Вызовите команду

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

Новый workspace создан! Можно приступать к работе с симулятором.

1.3.2. 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** (рис. 1.5). 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. 1.6). Выберите **Tools → Command Line Window** для того, чтобы открыть его.

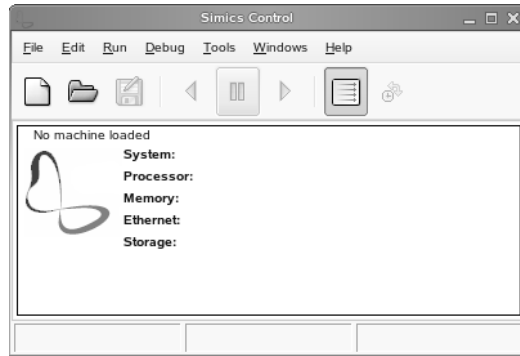


Рис. 1.5. Окно Simics Control

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, предупреждения, сообщения об ошибках. Всё, что вы можете сделать с помощью окна **Simics Control**, вы также можете сделать и с помощью командной строки. Большинство действий в дальнейшем будет производиться именно в ней.

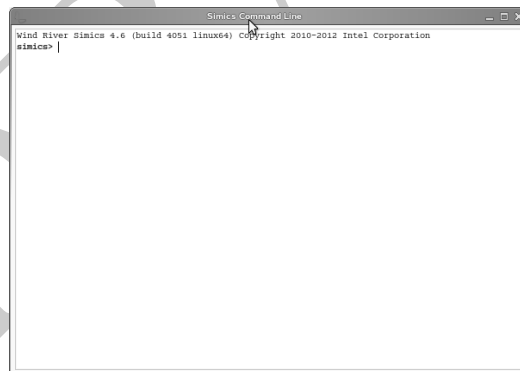


Рис. 1.6. Командная строка Simics

В данной работе изучается система (вычислительная платформа), основанная на процессоре Intel® Core™ i7. Linux используется в качестве операционной системы. В качестве среды поль-

зователя выступает пакет под названием BusyBox [6], часто используемый во встраиваемых (*англ.* embedded) системах.

Для подключения 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`.

Спустя некоторое время окно *Simics Control* покажет вам суммарную информацию о моделируемой системе, которую вы только что загрузили (рис. 1.7). Также должно появиться новое окно **Serial Console on viper.mb.sb.com[0]** (рис. 1.8).

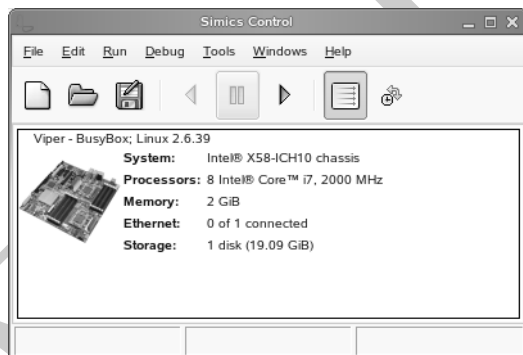


Рис. 1.7. Окно Simics Control после загрузки модели

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



Рис. 1.8. Окно **Serial Console on viper.mb.sb.com[0]**

1.3.3. Запуск моделирования

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

Рассмотрим следующий пример:

```
simics> continue
... the simulation is running ...
running> stop
[viper.mb.cpu0.core[0][0]] cs:0xfffffffff8100944d p:0
    x00100944d MWait state
... the simulation is now stopped ...
simics>
```

После запуска виртуальное время начинает увеличиваться, гостевые инструкции начинают исполняться, и сообщения от программного обеспечения выводятся в окно консоли. Если вы позволите модели поработать некоторое время, то увидите сообщения загрузки в окне **Serial Console on viper.mb.sb.com[0]** (рис. 1.9). Симуляция может быть остановлена с помощью команды **stop** или нажатия **Stop** на панели инструментов.

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

```
busybox login:
```

1. Знакомство с Simics

```
Serial Console on viper.mb.sb.com[0]
[ 2.264101] sit0: Features changed: 0x0007800 -> 0x0000780
[ 2.266290] NET: Registered protocol family 17
[ 2.267464] Registering the dns_resolver key type
[ 2.269171] registered taskstats version 1
[ 2.274107] Magic number: 12:155:908
[ 2.275082] new port: hash matches
[ 2.718997] ata2: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
[ 2.720551] ata2.00: ATAPI: Simics Turbo CD-ROM, ALPHAL, max UDMA/33
[ 2.722120] ata2.00: applying bridge limits
[ 2.732383] ata2.00: configured for UDMA/33
[ 2.724386] scsi 11:0:0:0: CD-ROM          VTA8      Turbo CD-ROM    RO  PQ
0: ANSI: 5
[ 2.726747] sr0: scsi3-mmc drive: 4x/4x cd/rw xa/form2 odda tray
[ 2.728234] cdrom: Uniform CD-ROM driver Revision: 3.20
[ 2.730544] sr 11:0:0:0: Attached scsi generic sg1 type 5
[ 3.036997] ata3: SATA link down (SStatus 0 SControl 300)
[ 3.041878] Freeing unused kernel memory: 640k freed
[ 3.043173] Write protecting the kernel read-only data: 10240k
[ 3.050762] Freeing unused kernel memory: 624k freed
[ 3.070435] Freeing unused kernel memory: 1892k freed

root@viper.mb.sb.com[0] input: PS/2 Generic Mouse as /devices/platform/i8042/serio/i
nput/input1
```

Рис. 1.9. Загруженная операционная система Linux

Введите в него **root**, нажмите **Enter**. Теперь операционная система загружена, и вы можете начать взаимодействовать с ней, например, вводить команды 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 `cpuinfo`:

```
# cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 10
model name    : Intel(R) Core(TM) i7 CPU           @
               2.00GHz
stepping      : 8
cpu MHz       : 1999.991
cache size    : 8192 KB
physical id   : 0
siblings      : 1
core id       : 2
cpu cores     : 2
```

Для того чтобы закончить работу симулятора, необходимо выполнить команду `quit`, или `q`, или `exit`.

1.4. Задания

1. Загрузить модель *Viper* с операционной системой *BusyBox* [6]. Для этого необходимо из вашего `workspace` выполнить команду

```
./simics targets/x86-x58-ich10/viper-busybox.simics
```
2. Открыть окно **Simics Command Line** и запустить симуляцию.
3. Сравнить содержимое файла `/proc/cpuinfo` симулируемой и реальной машины.
4. Найти способы определения факта нахождения внутри симулятора (имена устройств будут выдавать факт виртуализации).

Список литературы к занятию

1. Getting started with VNC. — RealVNC Ltd. — URL: <http://www.realvnc.com/support/getting-started.html>.
2. OpenSSH Manual. — URL: <http://www.openssh.com/ru/manual.html>.
3. Simics Getting Started Guide 4.6 / Wind River. — 2011.
4. *Tatham S.* PuTTY User Manual. — URL: <http://the.earth.li/~sgtatham/putty/0.62/puttydoc.txt>.
5. VNC. Access to your computers, when you need it. — RealVNC Ltd. — URL: <http://www.realvnc.com/products/vnc/>.
6. Официальный сайт BusyBox. — URL: <http://www.busybox.net/>.

2. Концепции симуляции

2.1. Цель занятия

В данной работе продолжается изучение принципов работы симулятора Simics. Центральная тема занятия — определение способов представления сущностей реальных устройств и систем в программных моделях.

2.1.1. Базовая единица симуляции

Аналогично тому, что физические системы строятся из узлов, программные платформы состоят из моделей отдельных узлов. При этом, как и в реальности, они образуют иерархическую структуру.

2.1.2. Диагностические сообщения

Одно из назначений симуляторов — помогать в разработке аппаратуры и программ для неё. При этом разработчикам необходимо понимать, что происходит в системе в определённые моменты времени. Для этого всем моделям позволено создавать диагностические сообщения, которые могут выводиться на консоль управления Simics, а также в файл. Поскольку излишне частые и подробные сообщения могут засорить журнал, Simics предоставляет несколько механизмов для их фильтрации.

2.2. Ход работы

Запустите симулятор с моделируемой системой внутри:

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

2.2.1. Иерархия устройств

Корневой элемент созданной модели компьютера имеет имя, хранящееся в переменной `$system`. В нашем случае это `viper` — модель шасси, на котором «крепятся» все остальные устройства. Этот факт выражается 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`, процессор на ней — `viper.mb.cpu0`, а первое ядро в этом процессоре — `viper.mb.cpu0.core[0][0]`.

Для просмотра списка всех устройств используйте команду `list-objects -recursive`. Альтернативно можно использовать окно **Object Browser** (рис. 2.1).

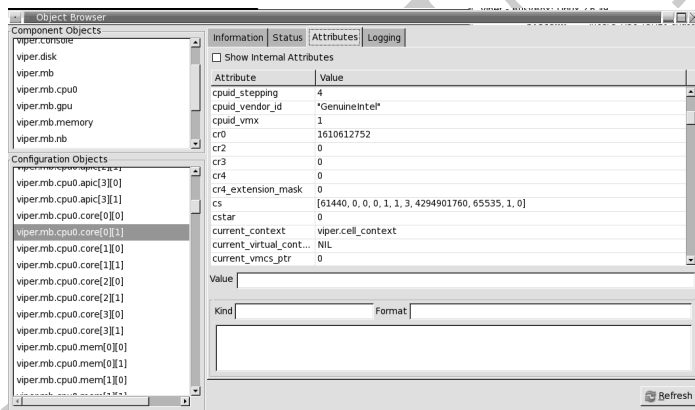


Рис. 2.1. Окно для просмотра объектов и их атрибутов

2.2.2. Команды

Каждый класс моделей может предоставлять несколько команд, которые позволяют инспектировать или изменять состояние объектов данного класса. Такие команды приписываются к иерархическому имени модели.

Самые часто реализуемые внутри классов команды — это `status` и `info`.

Так же все устройства модели памяти имеют команды `get` и `set`, которые позволяют читать и записывать данные в симулируемую память. Например:

```
simics> viper.mb.cpu0.mem[0][0].get address = 0xffff0000

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

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
```

2.2.3. Атрибуты

Атрибуты являются аналогом полей классов в 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 «->»:

Например, для просмотра 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],  
[0, 0], [0, 0]]
```

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

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

Типы атрибутов

По аналогии с переменными в языках программирования атрибуты имеют типы. При присвоении атрибуту нового значения предварительно производится проверка, что тип выражения с правой стороны соответствует его заявленному типу.

Типы атрибутов специфицируются с помощью строки, символы которой определяют допустимые варианты входных значений.

1. Скалярные. Имеют типы *i*, *s*, *b*.
2. Объекты. Их тип — *o*.
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*.
5. Допустимо иметь атрибут, тип которого выбирается 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 архитектурному состоянию, а используются для удобства представления информации об устройстве или же могут быть выведены из содержимого других атрибутов.

2.2.4. Интерфейсы

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. Стороннее 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 данной работе мы не будем уделять много внимания деталям работы с интерфейсами. Отметим лишь, что все команды любого устройства внутри себя написаны с использованием только его атрибутов и интерфейсов, т.к. не существует другого способа взаимодействий с ним. Однако устройства должны взаимодействовать друг с другом только через интерфейсы, но не атрибуты.

2.2.5. Модули устройств

С точки зрения организации хранения в Simics любая модель предоставляется ровно одним *модулем* — разделяемой библиотекой, загружаемой на этапе инициализации или позже. При этом модуль может предоставлять более одного класса моделей.

Для просмотра текущих загруженных модулей используйте команду:

```
simics> list-modules -l
```

Для загрузки некоторого модуля вручную используется команда `load-module`.

Кроме того, некоторые найденные модули по той или иной причине могут быть отвергнуты при загрузке. Увидеть их список можно с помощью команды `list-failed-modules`.

2.2.6. Последовательность создания 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, наличие всех обязательных атрибутов. Устройства представлены так называемыми предобъектами (*англ.* preobj).
2. *Инстанцирование устройств* с помощью команды `instantiate-components`. На этом этапе производятся все проверки на корректность атрибутов, наличие необходимых интерфейсов. Если не найдено ошибок, то предобъекты преобразуются в полноценные объекты Simics, которые могут участвовать в симуляции.

Отметим, что при необходимости добавить новые объекты в процессе симуляции описанные фазы могут быть повторены.

2.3. Задания

В этой работе мы будем создавать небольшую симуляцию, состоящую из простой модели процессора класса `sample-risc` и модели оперативной памяти. Для этого необходимо будет создать и настроить индивидуальные объекты. Исходный скрипт находится в файле `script.py` (см. приложение В). Однако он содержит несколько ошибок, которые не позволяют это сделать сразу. Задание состоит в том, чтобы разобрать сообщения симулятора, внести модификации в исходный скрипт и запустить его.

Следующие замечания должны помочь в решении поставленной задачи.

- Данный скрипт написан на языке Python и поэтому требует специального флага при указании его имени в командной строке.

- Для выяснения списка опций используйте команду `simics -h`.
- Переменные в Simics объявляются с помощью оператора «=». Имена переменных начинаются со знака доллар, например `$system`.
- Перед выполнением второй фазы инициализации объектов все обязательные атрибуты предобъектов должны быть настроены.
- Обращайте внимание на вывод сообщения об ошибках — часто они содержат имя файла и номер строки.
- Комментарии начинаются со знака «дизель» `#`.

3. Связь симуляции с внешним миром

3.1. Цель занятия

Необходимость коммуникации — доставка данных.

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

3.2. 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 сохране-

ние состояния между запусками симуляции нарушает принцип её воспроизводимости и повторяемости.

3.2.1. 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 9660¹.

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

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

3.2.2. 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.

3.3. Копирование файлов внутри 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 прямо-

го доступа к хозяйской операционной системе из моделируемой, называемой *SimicsFS*.

3.3.1. 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
```

3.3.2. Подключение симулятора к реальной сети

Подключение симулятора к реальной сети открывает много новых возможностей. Например, позволяет упростить процесс за-

грузки 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.

3.3.3. 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 появляется 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 3.1 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 существует команда, которая принимает объект типа Ethernet link.

3.4. Ход работы

3.4.1. SimicsFS

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

3. Связь симуляции с внешним миром

```
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.

3.4.2. 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
```

3. Связь симуляции с внешним миром

```
<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. Подключите 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 создаваться после выполнения команды `connect-real-network`:

3. Связь симуляции с внешним миром

```
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 TS0  
[ 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>
```

3. Связь симуляции с внешним миром

```
<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 3.1

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. — 2011.
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. — 2011.
4. Virtual Machine Disk Format (VMDK). — VMware, 2012. — URL: <http://www.vmware.com/technical-resources/interfaces/vmdk.html> (дата обр. 22.10.2012).

4. Использование симуляции для изучения системы

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].

4.1. Цель занятия

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

4.2. Ход работы

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

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

4.2.1. Подготовка исследуемой программы

Исходный код программы `debug_example.c`, содержащей ошибку, находится в приложении D. 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
```

4.2.2. 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
```

4. Использование симуляции для изучения системы

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
```

4.2.3. 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. 4.1 и 4.2 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 1 0 0 0 0 0 1 1 0 = 0  
    x00000206
```


4. Использование симуляции для изучения системы

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 O O F F F F F F F F F F F
P F P P
L L

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 = 0x82
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 = 0x0000000000040c930
cr3 = 0x0000000007c01b000

cr4 = 0 — 1 1 0 1 1 1 1 0 0 0 0 = 0x000006f0
V O O 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

4. Использование симуляции для изучения системы

```
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>

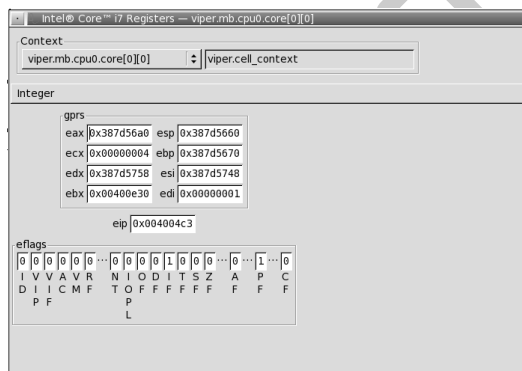


Рис. 4.1. Окно просмотра регистров. Вкладка с регистрами общего назначения

Память системы

Для просмотра различных пространств памяти, присутствующих в моделируемой системе, используется окно **Memory Contents**, вызываемое также командой `win-memory` (рис. 4.3).

4.2.4. Начало отладки

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

4. Использование симуляции для изучения системы

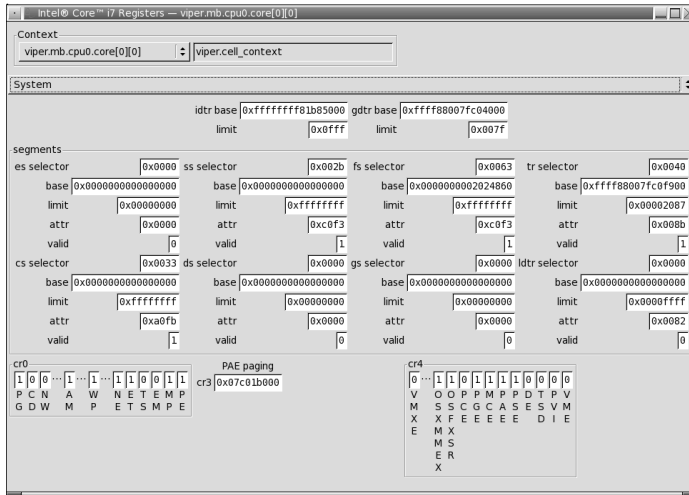


Рис. 4.2. Окно просмотра регистров. Вкладка с системными регистрами

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

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. 4.4) заполнено. В противном случае загрузите в окно исходный файл кнопкой **Find**.

4. Использование симуляции для изучения системы

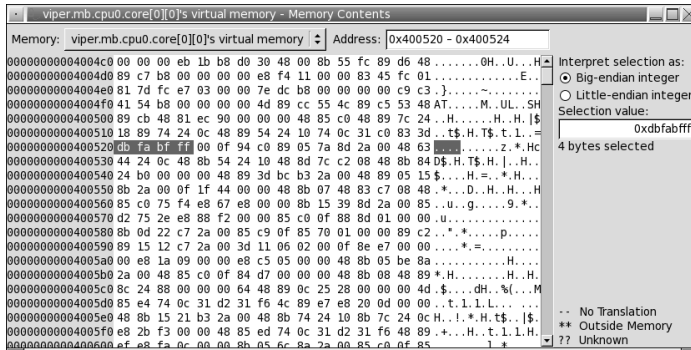


Рис. 4.3. Окно просмотра содержимого памяти системы

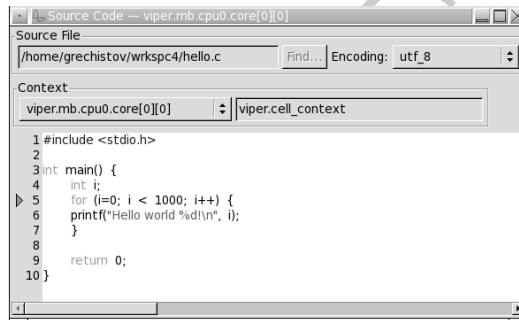


Рис. 4.4. Окно просмотра исходного кода

4.2.5. 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. 4.4), дизассемблера (рис. 4.5) и стека (рис. 4.6). Информация, содержащаяся в них, также может быть получена с помощью команд `list <имя функции>`, `disassemble <адрес>` `<количество>` и `stack-trace`.

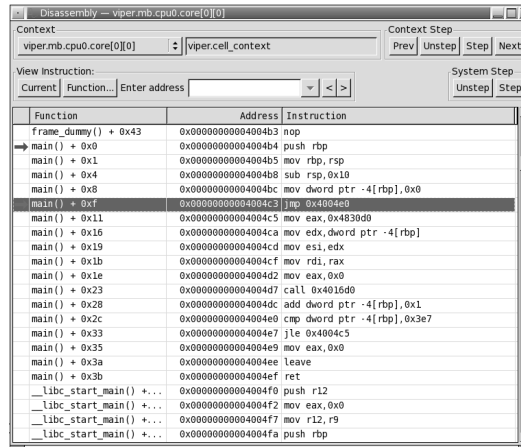


Рис. 4.5. Окно дизассемблера

4.2.6. 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.

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 выполнение подпроцедур, если они вызываются.
4. **pos** — узнать адрес функции или строки кода:

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

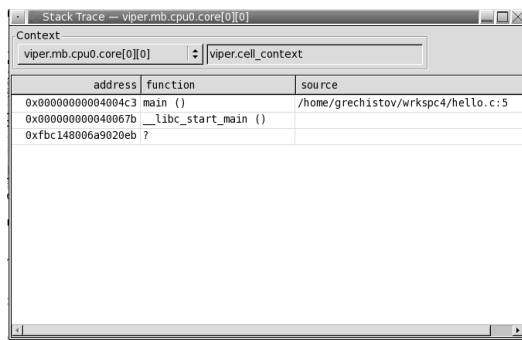


Рис. 4.6. Окно просмотра стека

Кроме задания этих команд, управление исполнением может осуществляться с помощью кнопок **Step** и **Next** окна Disassembly (рис. 4.5).

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:

4. Использование симуляции для изучения системы

```
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`.

4.3. 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.

4.4. 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 использован 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. — 2011.
4. Simics Hindsight User Guide 4.6 / Wind River. — 2011.

Приложения

А. 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.

А.1. Обновление workspace

Для получения последних исправлений ошибок в моделях необходимо использовать самую свежую версию базового пакета Simics из установленных на кластере. Номер версии определить по файлам, установленным в папке `/share/simics/simics-4.6/`. В тексте данной работы последней версией будет считаться **4.6.32**, при этом 4.6 — это основная версия, а последняя цифра — номер минорной версии обновления. Он будет использован позже.

Каждая копия workspace характеризуется версиями пакетов, в ней используемых. Номер пакета Simics Base определяет настройки версий остальных пакетов, установленных одновременно с ним. Для того чтобы увидеть список пакетов с их версиями, используйте команду:

```
$ ./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)
```

В примере сверху базовый пакет имеет версию 4.6.32. Новые пакеты будут периодически ставиться на кластере для исправления ошибок в базовых моделях. Для обновления своего workspace используйте команду **workspace-setup**, находящуюся внутри новой версии базового пакета (версии 4.6.<minor>), выполненную внутри workspace, который вы хотите обновить.

```
$ /share/simics/simics-4.6/simics-4.6.<minor>/bin/workspace-  
setup
```

Также версию Simics можно узнать прямо изнутри симуляции:

```
simics> version
```

А.2. Список часто используемых команд Simics

Команда	Синонимы	Выполняемая функция
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
help <command>	h, man	Вывести справку о команде или понятии
win-help		Открыть окно индексированной справки
win-control		Открыть окно Simics Control
%<register name>	read-reg, write-reg	Прочитать или записать содержимое регистра текущего процессора
output-radix <10 16>		Изменить основание используемой для вывода чисел системы счисления
delete [id]		Удалить точку останова по её номеру

А.3. Лицензия Simics

Получение лицензии и настройка лицензионного сервера Simics

TODO Написать

«Wind River Simics Installation Guide», глава 10.

А.3.1. Получение файла лицензии

TODO Написать Пример содержимого файла: **TODO** Написать

А.3.2. Настройка лицензионного сервера

TODO Написать

А.3.3. Решение проблем с конфигурацией

TODO Написать

В. Скрипт для отладки

```
name_prefix = cli.simenv.host_name
if name_prefix != '':
    name_prefix = name_prefix + '_'

sample_risc0 = pre_conf_object(name_prefix + "sample_risc0",
    "sample-risc")
sample_risc0.queue = sample_risc0

ram_image0 = pre_conf_object(name_prefix + "ram_image0", "
    image")
ram_image0.queue = sample_risc0
ram_image0.size = 0x800000

ram0 = pre_conf_object(name_prefix + "ram0", "ram")
ram0.image = ram_image0

phys_mem0 = pre_conf_object(name_prefix + "phys_mem0", "
    memory-space")
phys_mem0.queue = sample_risc0
phys_mem0.map = [[0x0, ram0, 0, 0, 0
    x800000],
    ]

ctx0 = pre_conf_object(name_prefix + "ctx0", "context")
ctx0.queue = sample_risc0

sample_core0 = pre_conf_object(name_prefix + "sample_core0",
    "sample-risc-core")
sample_core0.queue = sample_risc0
sample_core0.sample_risc = sample_risc0
sample_core0.physical_memory_space = phys_mem0
sample_core0.current_context = ctx0

#cosim_cell = pre_conf_object(name_prefix + "cosim_cell", "
    cell")
#cosim_cell.current_processor = sample_core0
```

```
#cosim_cell.current_step_obj = sample_risc0  
#cosim_cell.current_cycle_obj = sample_risc0  
#cosim_cell.scheduled_object = sample_risc0  
#sample_risc0.cell = cosim_cell
```

```
SIM_add_configuration([sample_risc0, ctx0, ram_image0, ram0,  
    phys_mem0, sample_core0, sample_core1], None)
```

DRAFT

C. 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    = TRUE
$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"
}
```

D. 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;  
}
```

DRAFT

Список TODO

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

- Исправить форматирование таблиц.
- Описать установку лицензии Simics в приложении.
- Отформатировать блоки кода: <http://mydebianblog.blogspot.ru/2012/12/latex.html>.

Платформы для симуляции

- DCPU-16 (Mojang)
- CHIP8, CHIP16 (game)
- Z-machine (Zork)
- MIX/MMIX (Knuth)