Открытые технологии

Материалы десятой международной конференции
Linux Vacation / Eastern Europe 2014

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов десятой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2014

(г. Гродно, 22–24 августа 2014 г.)

Брест
«Альтернатива»
2014

УДК 004.45(082)
ББК 32.973.26-018.2я43
О-83

Под общей редакцией **Д.А. Костюка**
Редакционная коллегия: **Д.А. Костюк, Н.Н. Маржан, Д.А. Пынькин, А.О. Шадура, В.В. Шахов**

Рецензенты: кандидат технических наук, доцент, декан факультета компьютерных систем и сетей Белорусского государственного университета информатики и радиоэлектроники **А.В. Прытков**
кандидат технических наук, доцент, заведующий кафедрой электронных вычислительных машин и систем Брестского государственного технического университета **С.С. Дереченник**

О-83 **Открытые технологии** : сборник материалов Десятой Международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2014, Гродно, 22–24 августа 2014 г. / под общей ред. Д.А. Костюка. – Брест : Альтернатива. – 188 с.

ISBN 978-985-521-446-6.

В сборник вошли материалы, представленные авторами на десятую международную конференцию разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2014, включая выездную зимнюю сессию конференции LVEE Winter 2014. Материалы докладов представлены на сайте конференции http://lvee.org и распространяются под лицензией Creative Commons Attribution-ShareAlike 3.0. Статьи посвящены новым технологиям и разработкам в сфере свободного (открытого) программного обеспечения и затрагивают широкий спектр платформСборник может быть интересен специалистам в области информационных технологий, занимающимся разработкой и сопровождением свободного программного обеспечения, а также подготовкой кадров высшей квалификации по профильным специальностям.

УДК 004.45(082)
ББК 32.973.26-018.2я43

ISBN 978-985-521-446-6

Содержание

О методах динамического встраивания в ядро операционной системы (на примере Linux)

Илья Матвейчиков, Москва, РФ*

The article presents an overview of methods for dynamic integration into the Linux kernel to modify (add, change) its functionality. Both traditional methods of integration based on changing kernel's code, such as patching, and methods based on using other capabilities are considered. Special attention is paid to bypassing integrity mechanisms while doing the interception. Data invalidation method is proposed.

Постановка задачи

Под встраиванием в программную систему понимается процесс внедрения в неё дополнительных (сторонних) программных элементов, осуществляемый таким образом, чтобы с одной стороны сохранялось её функционирование, а с другой — расширялись или изменялись её функциональные возможности.

Говоря о встраивании, будем рассматривать следующую практическую задачу. Пусть есть некоторый целевой компонент программной системы, функционирующий в соответствии с заданным (базовым) алгоритмом. Необходимо осуществить модификацию работы данного компонента таким образом, чтобы иметь возможность вносить конкретные изменения в этот базовый алгоритм.

Таким образом, основной целью встраивания является получение возможности контроля, модификации и расширения функций компонентов программной системы, тогда как основной задачей встраивания является обеспечение внедрения в существующую программную систему при сохранении её работоспособности. Успешно выполненное встраивание характеризуется сохранением функционирования программной системы при наличии в её составе нового компонента.

*matveychikov@gmail.com, http://lvee.org/en/abstracts/123

Терминология

При рассмотрении программной системы как совокупности взаимосвязанных программных модулей (компонент), образующих общую вычислительную систему, ее центральной частью становится ядро ОС. Оно выполняет функции посредника между приложениями и устройствами, осуществляющими обработку данных на аппаратном уровне. При этом, основной его задачей является эффективное управление ресурсами.

Рассматривая компоненты программных систем в качестве разного рода прикладных и системных программ, выполняющихся на соответствующем оборудовании, можно установить связь между встраиванием и получением возможности перехвата управления в ходе выполнения участков этих программ. При этом программный код, который обрабатывает подобные ситуации, называется кодом-перехватчиком или проще — хуком (англ., hooking). Сами же термины перехват (управления) и встраивание считаются схожими и, если это не оговаривается отдельно, используются для обозначения одного и того же. Однако следует иметь в виду существующее различие между ними: встраивание представляет собой процесс внедрения в общем смысле, тогда как перехват скорее указывает на конкретный методический приём.

Отличительной чертой методов динамического встраивания является отсутствие необходимости перезагрузки целевой системы для того, чтобы ожидаемые изменения вступили в силу. Как правило, объектами перехвата являются функции — элементы кода ядра, реализующие тот или иной алгоритм. Реже встраивание происходит в такие системные механизмы как обработчики исключений и, в частности, диспетчер системных вызовов.

Специфика ядра Linux

По своей архитектуре Linux представляет собой монолитное ядро с поддержкой возможности расширения функциональности за счёт модулей, по необходимости загружаемых в процессе работы. Учитывая данную особенность, для ядра Linux существует возможность разрабатывать расширения, которые, фактически являясь частью ядра, могут переопределять/дополнять различные его функции, т.е. изменять порядок работы его подсистем.

Перехват функций ядра является базовым методом, позволяющим переопределять/дополнять различные его механизмы. Исходя из того, что ядро Linux почти полностью написано на языке C, за исключением небольших архитектурно-зависимых частей, можно утверждать, что для осуществления встраивания в большинство компонентов ядра достаточно иметь возможность перехвата соответствующих функций.

Обработка исключений лежит в основе функционирования множества системных механизмов. Вследствие этого перехват обработчиков исключений ядра Linux позволяет повысить степень контроля над системой, а перехват диспетчера системных вызовов даёт возможность осуществлять регуляцию запросов прикладного ПО к сервисам ядра Linux.

Техника патчинга

В основе традиционных методов встраивания лежит патчинг (англ., patching) — техника внесения изменений в код или данные, позволяющая модифицировать поведение целевого алгоритма требуемым образом. Технически, результатом патчинга является изменение содержимого ячеек в оперативной памяти. Однако модификация кода, в отличие от модификации данных, имеет свои особенности, связанные прежде всего с фундаментальным отличием кода от данных.

Реализация перехвата с использованием техники патчинга требует квалификации, а также понимания принципов работы не только самого ядра, но и особенностей используемой аппаратной платформы. При модификации кода стоит особо обращать внимание на корректность встраивания в случае многопроцессорных систем, ведь в результате изменений не должна нарушаться когерентность. Кроме того следует учитывать необходимость обхода механизмов защиты кода ядра от модификации, а также особенности поиска и использования скрытых и не экспортируемых символов. Так или иначе, в большинстве случаев патчинг позволяет решить задачу встраивания.

Платформо-зависимые подходы

Недостатком патчинга можно считать необходимое нарушение целостности компонентов целевой системы. Внесение изменений в

код может быть легко обнаружено, что в некоторых контекстах является принципиальным ограничением. В этом случае следует использовать методы, лишённые такого рода ограничений. Как правило, часть таких методов использует аппарантые возможности платформы (например, аппаратные точки останова), что в принципе не может являться универсальным, учитывая хотя бы ограничения на число устанавливаемых перехватов. С другой стороны, всегда остаётся возможность использования разного рода виртуальных функций и прочих динамически заменяемых указателей, позволяющих переопределять в известных пределах поведение системы. Последнее, в частности, распространено для перехвата операций, осуществляемых в рамках виртуальной файловой системы (VFS), когда для операций с объектами используются таблицы виртуальных методов, замена которых может быть выполнена без модификации кода. Однако данный подход также имеет ряд ограничений, главное из которых заключается в том, что нет возможности контролировать то, контроль чего архитектурно не предусмотрен.

Метод инвалидации данных

В ходе исследования вопросов осуществления встраивания без модификации кода была отмечена возможность управления обработкой исключений в ядре Linux. На базе этого был разработан метод встраивания, получивший название метода инвалидации данных, суть которого заключается в том, что модификации (инвалидации) подвергается внутренняя переменная, используемая в коде целевой подсистемы. Вследствие того, что значение этой переменной инвалидируется, создаются условия для возникновения исключения при доступе к ней частей алгоритма. Обработка таких ситуаций позволяет получить управление, необходимое для исправления ошибки, что в свою очередь используется для перехвата управления, а следовательно и встраивания.

Примером применения метода инвалидации данных является осуществление встраивания в ключевые механизмы ядра Linux, контролируемые с использованием фреймворка LSM (Linux Security Modules) для архитекруры x86_64. При этом операция инвалидации данных заключается в изменении одного единственного бита в ключевой для LSM переменной — security_ops.

Заключение

Таким образом, для встраивания в ядро ОС существуют способы, использование которых применительно к конкретной задаче является более или менее целесообразным. Патчинг является базовым методом встраивания и может быть применим, если отсутствуют ограничения на сохранение целостности кода. В противном случае, в зависимости от ситуации, могут применяться специфичные для архитектуры решения (такие, как использование аппаратных точек останова), перегрузка виртуальных функций, а также метод инвалидации данных, который является в достаточной степени универсальным и может быть реализован для широкого круга систем.

Безупречная история в Git или Mercurial

Алексей Хлебников, Осло, Норвегия*

History of development saved in version control systems (VCS) is very important. It simplifies investigation of problems, reversion of regressions, picking specific changes for specific customers or releases, learning code for new developers in a team, generally keeping control over the code, assigning blame, etc. However, after long development of a complex software product, its VCS history is often hard to read. The talk shows ways to remedy the problem by consistent use of branching, rebasing and squashing, with detailed examples for Git and Mercurial.

Введение

Существуют различные способы использования VCS в процессе разработки. Некоторые команды просто коммитят всё в основную ветку одного репозитория. Иные используют ветвление (branching), слияние (merging), несколько репозиториев (cloning). Ниже предлагается вариант, который удобен разработчикам и в то же время оптимизирован для улучшения читаемости истории VCS. Иными словами, как правильно бранчить, сквошить и ребэйсить код, используя команды Git и Mercurial.

Важные приёмы процесса разработки

Ветвление (branching)

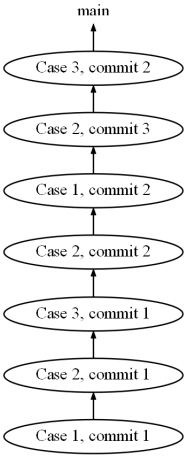
Ветвление имеет следующие преимущества:

- Работая в отдельной ветке над отдельным кейсом (case), вы можете свободно экспериментировать, не боясь сломать mainline. Это важно не только технически, но и психологически: над разработчиком не довлеет груз ответственности и он может позволить себе большую свободу действий.
- Соответственно, коммиты других участников проекта на других ветках не смогут ничего сломать на вашей собственной ветке.

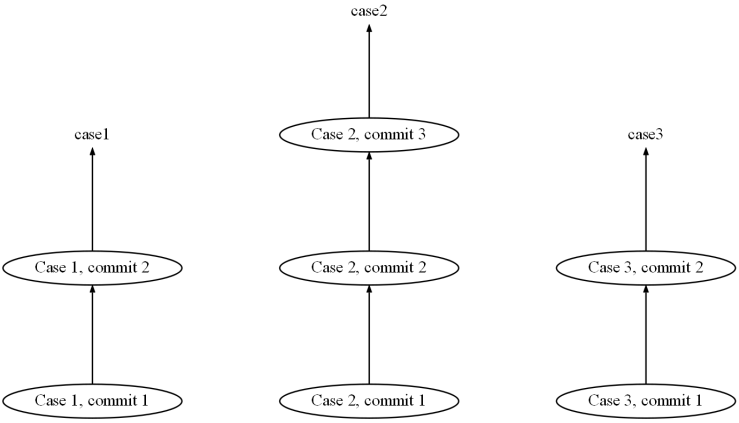
*alexei.khlebnikov@gmail.com, http://lvee.org/en/abstracts/125

– Все коммиты, относящиеся к данному кейсу, сгруппированы. Они идут по порядку, в отличие от ситуации, когда все участники разработки используют только одну ветку. Это облегчает понимание кода данного кейса и улучшает историю в VCS.
– Пока код данной ветки не доставлен в mainline, можно редактировать историю (остановимся на этом позже).

Без ветвления все коммиты идут вперемешку на mainline:



При ветвлении коммиты группируются по кейсам:



Rebasing

Rebasing может применяться в процессе работы над кейсом, а также для доставки коммитов в mainline.

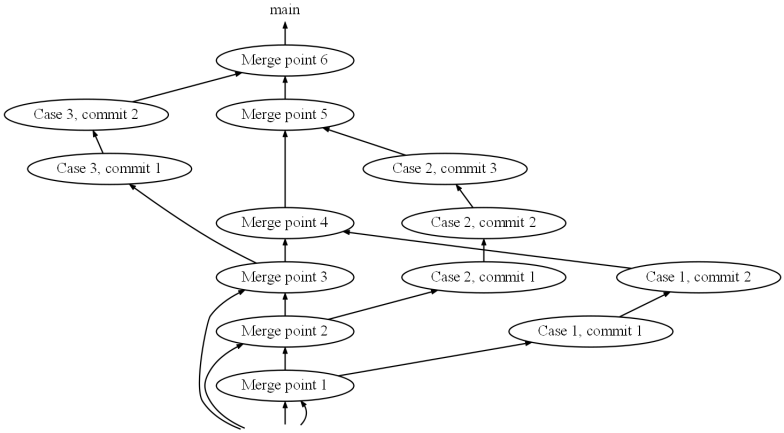
Rebasing в процессе работы позволяет:

- Получить ответвление от обновлённого mainline.
- Разрешать merge-conflicts небольшими порциями в ходе работы, а не одним большим куском в самом конце.
- Тестировать код своего кейса относительно нового mainline без его доставки в этот самый mainline.

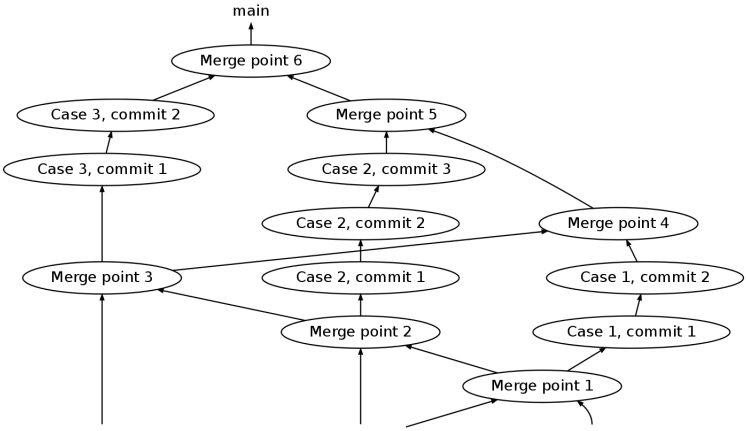
Rebase вместо Merge как метод доставки кода в mainline позволяет получить:

- Линейную историю в VCS. Линия проще и наглядней графа.
- Меньше проблем с blame, bisect и revert. Эти команды работают лучше на коммитах с одним родителем, чем с несколькими.
- Возможность удалять очень старые ветки (и их коммиты) из репозитория. Это повышает быстродействие репозитория. А если эти ветки всё-таки нужны — их можно хранить в архивном репозитории или в бэкапе. Тем, кто считает замедление репозитория при увеличении количества коммитов надуманной проблемой, есть смысл ознакомиться с исследованием Facebook [1].

При доставке кода в mainline через Merge, mainline состоит в основном из merge-коммитов:

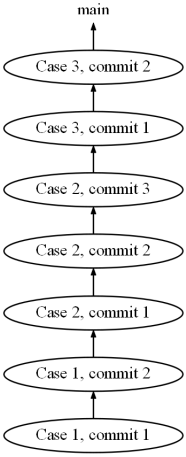


Тот же самый граф, но без ярко выраженного вертикального mainline:



Как видно, в такой истории довольно трудно разобраться, особенно по прошествии долгого времени, или если разбирающийся — новый человек на проекте. В этом графе даже mainline трудно найти.

А вот история, которая получается при Rebase:



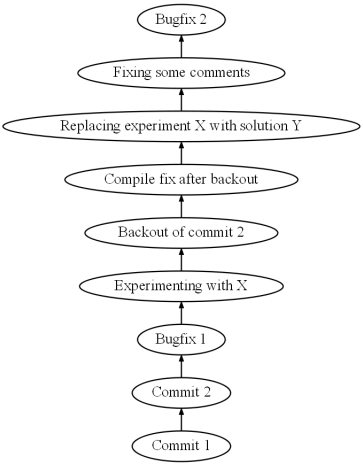
Как видно, история линейна, что сильно улучшает её читаемость. Но можно сделать ещё лучше — уменьшить количество коммитов. И в этом поможет squashing.

Squashing

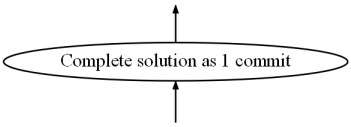
Squashing — это слияние нескольких коммитов в один. Squashing позволяет получить:

- Компактную историю. Это тем важнее, чем дольше идёт разработка и чем больше коммитов собралось в репозитории.
- Меньше мусора в истории.
- Более лёгкий откат изменений.

Если после работы над кейсом ветка содержит много мусора в истории...



...squashing позволит избавиться от этого мусора, слив все комиты в один:



	Git	**Mercurial**
Ответвление от mainline	git checkout --b case4	hg book case4
Работа над кейсом	git commit -m "Commit 1" git commit -m "Commit 2"...	hg commit -m "Commit 1" hg commit -m "Commit 2"...
Rebase и squash	git checkout --b *case4-2* git rebase --interactive main	(нужны расширения rebase и histedit) hg rebase --keep --dest main hg histedit main hg book *case4-2*

Конкретные команды, шаг за шагом

Приведем примеры команд для Git и Mercurial в виде таблицы:

Отдельно коснёмся «табу на rebase после push». Существует довольно распространённый миф, что если ветка запушена (push) на сервер — все возможности ребэйса (rebase) для неё потеряны, потому что если ветку пробебэйсить и запушить на сервер опять (что возможно только с ключом –force), то это создаст несоответствие между репозиторием на сервере и репозиториями других разработчиков. В результате эти разработчики при попытке подтянуть эту ветку с сервера получат сломанную ветку.

На самом деле rebase возможен, если выполнять его правильно. На примере команд, приведённых в таблице, видно, что при ребэйсинге создаётся новая ветка, **case4-2**. Это принципиальный момент. Первоначальная ветка, **case4**, так и остаётся на своём месте, и получается аналог copy-on-write. Таким образом консистентность репозитория не нарушается, и ветка case4 не ломается — она просто устаревает. Теперь про неё можно забыть, а дальнейшую разработку, если она ещё продолжается, вести на ветке case4-2.

Также следует обратить внимание на ключ –interactive для Git и команду histedit для Mercurial. В результате их использования Git

или Mercurial вызывают текстовый редактор, в котором разработчик может редактировать историю своей ветки: помечать коммиты для правки комментария, сливать несколько коммитов вместе, менять коммиты местами, удалять ненужные коммиты.

В сущности, многие коммиты — просто мусор в истории VCS: неудавшиеся эксперименты, багфиксы, фиксы компиляции, чистка неиспользуемых переменных, исправления опечаток в комментариях. Подобный материал в истории VCS не представляет ровным счётом никакого интереса. Как правило, от разработчика требуется имплементация фичи X или исправление бага Y, и желательно одним куском (то есть, как правило (хоть и не всегда), одним коммитом). А детали того, через что разработчик прошёл в процессе разработки, никого не интересуют. По этой причине мелкие правящие коммиты всегда имеет смысл объединить с «главными» коммитами, которые они дополняют. Это же относится и к фиксам в результате code review.

Делать слишком много «главных» коммитов для одного кейса тоже не имеет смысла. Наоборот, для большинства кейсов перед доставкой кода в mainline лучше слить все коммиты в один и использовать название кейса как комментарий этого единственного коммита. Если имел место рефакторинг без изменения функциональности — его имеет смысл выделить в отдельный коммит. Если имели место фиксы багов mainline'а, которые проявились при работе над данным кейсом — их тоже имеет смысл выделить в отдельные коммиты, и поместить эти коммиты перед основной разработкой. Других коммитов не нужно. Это и есть squashing — слияние нескольких коммитов в один.

Кроме чистки мусора в истории squashing также помогает убрать коммиты, которые не компилируются, путём слияния с фиксом компиляции. Это важно, если используется bisect, а также в случае отката изменений.

Если вам захочется оставить в истории VCS свои чаяния и веяния по поводу данного кейса из ностальгических соображений — есть возможность сохранить их только для себя на той старой ветке case4, которая была любезно сохранена при copy-on-write-rebase. Не перетаскивая свой мусор в mainline, разработчик заодно не создает аналогичного искушения для товарищей по команде.

Git	Mercurial
git push . case4-2:main	(hg update case4-2) hg book main # «book» does fast-forward # in this case

Доставка кода в mainline

Итак, работа над кейсом завершена, код кейса оттестирован с новейшим mainline. После финальных rebase и squash на ветке должна находиться краткая и красивая история кейса, а сама ветка основана на верхушке mainline. Это и есть подходящий момент для доставки кода кейса в mainline. Для этого надо всего лишь переместить указатель mainline вперёд по ветке case4-2 — сделать fast-forward. Это можно сделать несколькими способами; автор предпочитает такие:

Важно обратить внимание на точку после git push. Она означает «текущий репозиторий». Однако можно пушить и сразу в origin:

git push origin case4-2:main

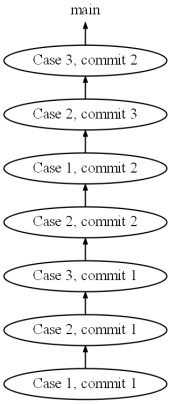
При этом не следует опасаться поломки origin/main, т.к. если предлагаемый push не fast-forward, Git не позволит выполнить push без ключа --force.

Выводы

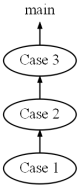
В результате использования предложенного подхода мы получаем:

- Удобство разработки на отдельных ветвях.
- Возможность всегда работать и тестировать свои изменения относительно новейшего mainline.
- Линейную историю.
- Группировку коммитов по кейсам.
- Безупречную и компактную историю без мусора.

Before: chaos



After: order



Литература

[1] `http://thread.gmane.org/gmane.comp.version-control.git/189776`

Коммерциализация СПО под GPL лицензией

Александр Рябиков, Сергей Середа, Москва, РФ*

In accordance to main goals of ADempiere Foundation our task was to find a way to get return on investments in Free Software modification/enhancement for a SMB developer company without dual licensing and without accompanying the product with proprietary add-ons/plugins or additional services or merchandise. Also there were a need to provide a mechanism of costs' sharing between several interested SMB companies.

After initial economic analysis we've concluded that the only way to get return on investments in free software development is to create a time lag between the moment of software product provision to a user and the moment of rights provision for this software according to GPL. Otherwise, demand and supply law just will not work with free software.

Also we have studied the legislation and FSF Comprehensive FAQ about the GNU Licenses. As a result, there were constructed two schemes able to create such time lag without GPL violation.

First is based mostly on GNU FAQ, second is based on the provisions of contract law.

СПО под GPL лицензией

Как известно, «свободные» лицензии дают конечным пользователям полную свободу, в том числе и свободу распространять конечный программный продукт наравне с разработчиком. А это означает, что каждый раз, когда разработчик продаёт копию ПО под «свободной» лицензией, он создает себе конкурента. Принято считать, что невозможно зарабатывать деньги непосредственно на доработке и распространении свободного программного обеспечения, распространяемого на условиях GPL или совместимых с нею лицензий. Описываемые в литературе и применяемые на практике бизнес-модели практически безусловно предполагают бесплатное (или за символическую плату) распространение таких программ. В

*rsashka@mail.ru, http://lvee.org/en/abstracts/121

противоположность этому, наличие «традиционной» (на жаргоне называемой «проприетарной») лицензии позволяет зарабатывать непосредственно на распространении программного обеспечения.

Обладателю исключительных прав на программу ещё доступна модель двойного лицензирования, подразумевающая «несвободную» коммерческую лицензию на ПО для бизнес-заказчиков и GPL-совместимую лицензию для представителей сообщества. Для всех же вторичных проектов (так называемых «форков») подобная возможность исключена. Любые доработки кода для ПО, выпущенного под GPL, могут быть выпущены только под этой же или совместимой «вирусной» лицензией, а замена GPL на несовместимую лицензию допускается только с согласия всех обладателей исключительных прав на программу.

С учетом этого, считается, что зарабатывать на Free Software, не являясь обладателем прав на оригинальный продукт, можно либо оказывая сервисные услуги (сопровождение, документирование, обучение и т.п.), либо компонуя его с собственными разработками с закрытым исходным кодом, способом, который допускается используемой свободной лицензией.

Описываемый ниже способ коммерциализации доработок свободного ПО позволит в определенной мере изменить сложившуюся ситуацию. Предлагаемая бизнес-модель предназначена, в основном для b2b-разработчиков, развивающих уже существующие свободные программные продукты, но не владеющих исключительным правом на сам продукт (т.е. наиболее часто встречающаяся ситуация).

Суть бизнес-модели сводится к созданию условий, аналогичных «традиционному» модели распространения программного обеспечения, т.е. к «продаже» доработанного продукта сразу нескольким покупателям, что даст возможность окупить понесенные затраты на доработку СПО за счет их многократной продажи. Такой эффект достигается за счет создания временного лага между началом продаж доработанного программного продукта и моментом его размещения в свободном доступе для бесплатной загрузки, что позволяет разработчику стать, на какое-то время, единственным продавцом, у кого эти доработки можно будет приобрести.

Особенность предлагаемой модели заключается в том, что при этом не происходит ни изменения, ни нарушения условий «вирусного» лицензионного соглашения — эффективное ограничение распространения продукта определяется договором между хозяйству-

ющими субъектами и вытекает из процесса выполнения заказных программных разработок.

1. Решение, «подсказанное» Free Software Foundation (FSF)

Временной лаг создаётся за счёт превращения пользователя в (со)разработчика, подписания с ним соглашения о нераспространении экземпляров продукта и предоставления ему программного обеспечения для тестирования на продуктивных данных.

Часто забываемые факты о GPL

- GPL допускает платное распространение программ [1].
- GPL не требует от разработчика делать исходные коды своих разработок доступными широкой публике [2].
- Разработчик не вправе запрещать Заказчику дальнейшее распространение ПО [3, 4].
- GPL позволяет Заказчику запретить разработчику распространять результаты заказной разработки [5].

Сценарий по «подсказке» FSF

Исходная ситуация: Разработчиком доработан GPL-продукт, есть компания-пользователь, желающая внедрить доработанный продукт, но не готовая оплатить полную стоимость доработки, Разработчику необходимо обеспечить возврат инвестиций в доработку, но есть риск, что компания-пользователь опубликует доработки, как только получит к ним доступ.

Предлагаемая схема взаимодействия к компанией-пользователем:

- Разработчик продает компании-пользователю исходный продукт без своих доработок.
- Разработчик нанимает компанию-пользователя для выполнения работ, связанных с новым кодом, в частности, для тестирования.
- В соответствии с комментариями FSF [5] существует законная возможность ограничить распространение тестируемого кода на время действия соответствующего договора.

– Время действия этого договора и будет определять временной лаг между началом распространения доработок GPL продукта и моментом его появления в свободном доступе.
– Разработчик может одновременно работать по такой схеме сразу с несколькими заинтересованными компаниями.

2. Решение на основе договорного права

Временной лаг создаётся за счёт того, что права на созданный по заказу код и возможность его дальнейшего распространения возникают только после полного исполнения обязательств сторонами договора на доработку ПО.

Сценарий на основе договорного права

Исходная ситуация: Разработчиком доработан GPL-продукт (разработчик готов его доработать), есть компания-пользователь, желающая внедрить доработанный продукт, но не готовая оплатить полную стоимость доработки, Разработчику необходимо обеспечить возврат инвестиций в доработку, но есть риск, что компания-пользователь опубликует доработки, как только получит к ним доступ.

Предлагаемая схема взаимодействия к компанией-пользователем:

– Разработчик заключает с компанией-пользователем договор на доработку исходного GPL продукта.
– В соответствии с действующим законодательством РФ (Статья 712. Право подрядчика на удержание, Статья 1296. Программы для ЭВМ и базы данных, созданные по заказу), в этом случае существует законная возможность ограничить распространение тестируемого кода на время действия соответствующего договора.
– Время действия этого договора и будет определять временной лаг между началом распространения доработок продукта под GPL лицензией и моментом его появления в свободном доступе.
– Разработчик может одновременно работать по такой схеме сразу с несколькими заинтересованными компаниями.

Описанная конструкция не является каким-то искусственным построением. Наоборот, она является описанием вполне реального

процесса заказной разработки корпоративного ПО, который завершается только после проведения опытной или опытно-промышленной эксплуатации разработок (с их полным сопровождением все это время) и передачи их в промышленную эксплуатацию.

По мнению авторов, регулируя длительность временного лага между началом продаж доработанного программного продукта и моментом его размещения в свободном доступе для бесплатной загрузки, в зависимости от типа, сложности, популярности и иных характеристик дорабатываемого программного продукта, можно обеспечить возврат инвестиций в его развитие, достаточный для поддержания коммерческого предприятия. При этом, за счет заключения подобных договоров с несколькими (в идеале — со многими) пользователями-заказчиками будет также обеспечено и разделение затрат между ними (в отличие от схемы bounty source, в рамках которой доработка целиком оплачивается одним заказчиком).

Конечно, наложение любых ограничений на распространение СПО может быть воспринято «в штыки» сообществом и, возможно, потребуются определенные усилия, что бы сторонники двух диаметрально противоположных точек зрения на способы распространения ПО смогли оценить положительные и отрицательные стороны от такого «нововведения». Тем не менее, описанная бизнес-модель полностью основана на законе и не нарушает ни одного положения о защите интеллектуальной собственности. Более того, она, по сути, является просто специализированной модификацией широко применяемой в корпоративном секторе стандартной модели заказной разработки ПО.

Мы обсудили с Ричардом Столлманом эти способы и он сделал важные замечания к описываемым моделям. Второй способ на основе контрактного права соответствует GPLv2, но нарушает GPLv3, где в явном виде прописано, что лицензия имеет приоритет над контрактными обязательствами. С этим мнением не согласны наши юристы, но кто из них прав может решить только судебная практика. Насчет первого способа, Ричард конечно не в восторге, но нарушение GPL будет в том случае, если работа пользователя будет фиктивной. Другими словами, пользователь должен по настоящему работать в соответствии с контрактными обязательствами и эта работа должна реально оплачиваться.

Конечно, используя описанные выше принципы, первое время код будет отсутствовать в свободном доступе, но через какое то время он все равно станет доступен для всех. И в результате, и

инвесторы вернут вложенные средства и сообщество получит ощутимый вклад в развитие СПО.

Данная методика коммерциализации СПО с лицензией GPL была представлена на Russian Open Source Summit 2014. Статья по материалам доклада доступна на сайте PCWEEK [6].

Литература

[1] http://www.gnu.org/licenses/gpl-faq.en.html#DoesTheGPLAllowMoney

[2] http://www.gnu.org/licenses/gpl-faq.en.html#DoesTheGPLRequireAvailabilityToPublic

[3] http://www.gnu.org/licenses/gpl-faq.en.html#DoesTheGPLAllowNDA

[4] http://www.gnu.org/licenses/gpl-faq.en.html#DoesTheGPLAllowModNDA

[5] http://www.gnu.org/licenses/gpl-faq.en.html#DevelopChangesUnderNDA

[6] http://www.pcweek.ru/foss/article/detail.php?ID=164583

Text parsing with python and PLY

Daniil baturin, Tomsk, Russia*

Language parsing is a common software development problem. Python is widely used in both production software development and rapid prototyping; a number of lexer and parser generators were written for it. In this talk we discuss using one of them, PLY, which is a pure python implementation of classic lex and YACC tools, for parsing a made-up configuration file grammar.

Formal language parsing is a common problem in software development. Standardized formats with already available parsing libraries, such as XML and JSON, have simplified the problem, but did not completely remove it. Python is widely used as a general purpose programming language as well as one for rapid prototyping, so Python programmers face this task as well.

There is a number of parser generators for Python, using different algorithms and APIs. In this talk we discuss PLY that is a pure Python implementation of classic UNIX tools: lex scanner generator, and YACC parser generator. It is maintained by David Beazley and distributed under thee-clause BSD license.

Lex and YACC are there for decades and have been reimplemented not once. This makes PLY easy to learn for UNIX programmers and relatively easy to convert the Python code into C/C++ if needed.

Parsing basics

There are three typical approaches to make a parser:

1. write an ad hoc parser manually,
2. use a parser generator,
3. write an advanced parsing algorithm manually.

Ad hoc parsers are suitable for simple grammars, but tend to quickly become hard to maintain and modify. Coding an advanced parsing algorithm like an LL parser is usually redundant and only feasible for very complex grammars or in case of special requirements. For the

*daniil@baturin.org, http://lvee.org/en/abstracts/122

majority of grammars a parser produced with some parser generator is usually the best option.

Most parser implementations do not work directly with the character stream but use a *token* stream produced by a *scanner*. Using a separate scanner (also known as lexer) to break the stream into tokens simplifies the task as it allows to write grammar rules in terms of token names rather than literal strings, e.g. all arithmetic operators can be referred to as OPSIGN token in a rule for arithmetic expressions. Scanner code is usually generated automatically from a set of regular expressions as well.

The most common algorithm used by parser generators is LALR (1) that reads tokens until a sequence matching the entire grammar rule is found.

Using PLY

PLY consists of two modules, `ply.lex` and `ply.yacc`. Actions for tokens and grammar rules are defined as functions, while token regular expressions and grammar rules are defined in function docstrings. Tokens that need no actions can be defined as variables.

A simple parser for breaking into parts a line consisting of letters and digits groups separated by semicolon will look like:

```
import re
import ply.lex as lex
import ply.yacc as yacc

## Lexer part
tokens = ( 'LETTER', 'DIGIT', 'SEMI' )

t_LETTER = r'[a-z]'

def t_DIGIT(t):
    r'[0-9]'
    t.value = int(t.value)
    return t

t_SEMI = r';'
t_ignore = r' '
```

```
lexer = lex.lex(debug=1)

## Parser part
start = 'start'

def p_letter_digit_pair(p):
    ''' pair : LETTER DIGIT SEMI '''
    p[0] = (p[1], p[2])

def p_pair_group(p):
    ''' pair_group : pair_group pair
                   | pair
    '''
    print p[0], p[1]
    p[0] = [p[1]] if len(p) == 2 else p[1] + [p[2]]

start = 'pair_group'

parser = yacc.yacc(debug=1)

## Test it
s = "a 0; b 1; c 2;"
print(parser.parse(s))
```

A more elaborate example discussed in the conference talk can be found at `https://github.com/dmbaturin/ply-example`

References

[1] PLY website. `http://www.dabeaz.com/ply/`

[2] Pete Jinks, The Implementation and Power of Programming Languages. `http://www.cs.man.ac.uk/~pjj/cs212/ho/ho.html`

Bootloader and Linux kernel debugging on ARM board with OpenOCD

Vladimir Zapolskiy, Espoo, Finland*

The system software development in area of embedded systems is complicated by proximity to the physical world, incomplete knowledge of hardware errors and sophisticated existing software. Fortunately it is possible to reuse advantages of a debugger in development of operating systems and bootloaders with the aid of a bridge from hardware and running system software to the GDB session on a developer's workstation, which is provided by JTAG interface, a cheap JTAG-USB adapter and connective software from open source OpenOCD project.

Consider a task of a new device development from system programmer's point of view, and to immerse in the details let the device has any purpose but it complies with the following requirements:

- the device is powered by ARMv4 or higher application processor, possibly multicore one,
- the ARM application processor executes bootloader, hypervisor or operating system kernel, and the developer has access to its source code, for instance U-boot bootloader and Linux kernel.

A system programmer of ARM powered embedded systems unavoidably meets a number of tasks and problems during the low-level development stage:

- port a bootloader or Linux kernel to a new ARM SoC,
- port a bootloader or Linux kernel to a new device governed by some ARM SoC,
- add new features of arbitrary nature into a bootloader or Linux kernel,
- fix a bug in a bootloader or Linux kernel code,
- thoroughly perceive the work of a subcomponent in a bootloader or Linux kernel during runtime.

Arguably not all aforementioned tasks are related to debugging, but for simplicity let's call them debugging tasks, and let's find a

*vz@mleia.com, http://lvee.org/en/abstracts/126

handy debugger, which helps to resolve the problems listed above. One well-known by developers, favourable and powerful debugger is GNU Debugger or GDB [1], and fortunately there is a way to utilize the same debugger for the introduced new challenges in the domain of low-level system programming, so GDB helps to debug not only applications, but bootloaders and operating systems.

One of the most popular methods of system code debugging is the control of application processor cores in runtime by means of boundary scan checking of integrated circuits according to JTAG specification (IEEE Std. 1149.1)[2], and the considered ARM SoCs and most of development PCBs have such interface. There are commercial brand JTAG adapters from Lauterbach and Abatron, but due to many benefits (convenience and flexibility in use, price, advantages of open source software etc.) it is reasonable to use simple cheap JTAG-USB adaptors and the attendant supplementary managing code from the open source, constantly developing Open On-Chip Debugger project or OpenOCD[3].

Functionally OpenOCD software paired with e.g. FT2232 powered JTAG-USB adapter successfully compete with notably more expensive brand alternatives. OpenOCD allows to write sophisticated Tcl scripts operating with the target CPU and board (for example, automated firmware upload, writing to a flash drive on board reset, etc.), scripts handling and post-processing events from cores, reading and modifying board RAM content and core registers, setting execution breakpoints and data read/write watchpoints, and also OpenOCD presents two user interaction interfaces, among which one is for low-level and manual operations via telnet protocol and another one serves as a GDB server. The latter familiar GDB server interface is conveniently utilized by a developer, who applies her/his ready practical knowledge of debugging with GDB and uses GDB open documentation, as well as add-ons – for example, user's own GDB scripts or GDB frontends and IDEs like DDD, Emacs or Eclipse.

One of the powerful functions of GDB server from OpenOCD is the ability to debug in runtime the code execution on multiple CPU cores simultaneously by means of a GDB protocol extension. From the user's point of view it looks similarly to debugging a multithread application, but the role of an application is performed here by e.g. the Linux kernel.

For embedded software developers OpenOCD facilitates essentially the execution of any tasks mentioned in the beginning, and debugging a complex source code beneath the application level becomes easy and

comprehensible, making OpenOCD a helpful tool and adding it to the system developer's must-have toolbox.

References

[1] The GNU Project Debugger, https://www.gnu.org/software/gdb/

[2] IEEE 1149.1-2013, http://standards.ieee.org/findstds/standard/1149.1-2013.html

[3] Open On-Chip Debugger, http://openocd.sourceforge.net/doc/html/index.html

Язык программирования Go: использовать нельзя игнорировать

Алексей Романов, Минск, Беларусь*

This paper gives a short introduction into Go programming language. It is a statically-typed language with syntax loosely derived from that of C, adding garbage collection, type safety, some dynamic-typing capabilities, additional built-in types such as variable-length arrays and key-value maps, and large standard library. Main advantages and disadvantegs of Go language are reviewed. Go would be extremely helpful for creating scalable and robust network services with great performance.

Go, часто именуемый так же как Golang — компилируемый, многопоточный язык программирования, разработанный компанией Google. Язык довольно молодой, разработка его началась в недрах компании Google в 2007 году, в 2009 он был анонсирован публике, а мартом 2011 датирована первая стабильная версия — r56. Язык активно развивается и в настоящее время, текущая стабильная версия языка — 1.3. Создателями языка являются небезызвестные товарищи Роб Пайк и Кен Томпсон, а также Роберт Гризмер.

Ключевыми особенностями языка Go являются:

- многопоточность и конкурентность встроена в язык;
- автоматическое управление памятью (garbage collection);
- высокая производительность(сравнима с C/C++);
- мощная стандартная библиотека;
- частичная поддержка ООП;
- статическая и строгая типизация;
- C-подобный простой синтаксис;
- открытый исходный код;
- большое и открытое сообщество разработчиков (522 контрибьютера);
- серьёзная поддержка от Google и других вендоров ПО.

Язык Go умеет распараллеливать написанную программу на все процессоры компьютера, на котором она запускается. Для этого используется механизм сопрограмм, которые называются *горутины*

*drednout.by@gmail.com, http://lvee.org/ru/abstracts/140

(go-routines). *Сопрограммы* — это легковесные потоки, и их количество может достигать десятков и сотен тысяч в одной программе. При этом Go относится довольно бережно относится к ресурсом компьютера и не потребляет много процессора и памяти без необходимости.

Автоматическое управление памятью в языке Go прилично ускоряет скорость разработки, но реализация сборки мусора может вызвать дополнительные задержки в процессе выполнения программы.

Если верить бенчмаркам, Go значительно превосходит в производительности популярные скриптовые языки Python, Ruby, PHP, примерно равен по производительности языку Java и немного отстаёт по этому показателю от C/C++. Потребление памяти у Go довольно скромное по сравнению с Java, Python, Ruby, но C/C++ он проигрывает в этом компоненте.

Язык Go имеет богатую стандартную библиотеку, которая позволяет просто решать многие задачи, не изобретая велосипед. Существуют довольно большое пакетов от сторонних разработчиков, которые можно загрузить через систему пакетов Go. При необходимости, можно использовать C/C++ библиотеки напрямую из Go.

Поддержка ООП в языке Go довольно оригинальная. В нем нет классов, наследования и традиционного полиморфизма. Вместо этого есть структуры, интерфейсы и возможность встраивать их друг в друга. Таким образом, язык Go можно назвать легковесным ООП-языком.

Синтаксис языка довольно простой, и изучить его можно за несколько дней. Статическая и строгая типизация не вызовет никаких проблему у программистов с опытом на C/C++/Java. При переходе на него со скриптовых языков Python/Ruby/Perl замечено, что скорость разработки довольно прилично падает, но при этом возрастает качество полученного на выходе кода.

Существуют 2 основных реализации языка Go: Go Compiler(gc) и проект gccgo. GC — это оригинальный компилятор, разработанный в недрах компании Google и выпущенный под BSD-лицензию с 3 пунктами. gccgo является частью коллекции компиляторов GCC и лицензирован под GPLv3.

Язык активно используют различные коммерческие компании для решения своих производственных задач, таким образом, его можно считать готовым к промышленному использованию. Наиболее известные компании, использующие Go для решения серьёз-

ных инженерных задач — Google, Yandex, Dropbox, Github, Iron.io, Zynga.

К недостаткам Go можно отнести:

- проблемы с написанием обобщенного кода (generics в Java, templates в C++);
- проблемы со использование Go во встраиваемых системах (embedded systems);
- язык довольно специализированный (например, написать приложение с GUI-интерфейсом будет не так просто);
- язык довольно молодой (не так много библиотек, возможны проблемы с производительностью, поиском программистов под проект на этом языке).

Таким образом, Go — это довольно молодой язык с открытым исходным кодом и большим растущим сообществом разработчиков. Основная специализация Go — написание высокопроизводительных сетевых сервисов, которые могут обрабатывать большое количество одновременных запросов. При этом Go эффективно использует основные ресурсы компьютера — процессор и память. Кроме этого, Go является языком общего назначения, позволяющим решать широкий круг различных производственных задач.

Пример кода на Go, запускающего 2 бесконечных сонных горутины:

```
func server(i int) {
   for {
      print(i)
      time.Sleep(10)
   }
}
go server(1)
go server(2)
```

Литература

[1] https://www.openhub.net/p/go
[2] https://code.google.com/p/go-wiki/wiki/GoUsers
[3] http://benchmarksgame.alioth.debian.org/
[4] http://golang.org/doc/devel/release.html
[5] http://golang.org/doc/faq
[6] http://yager.io/programming/go.html

ROOT. A data analysis framework

Andrew Savchenko, NRNU MEPhI, Moscow, Russia*

Modern high energy physics (HEP) demands a high performance large scale data mining toolkit. An introduction to such tool — a ROOT data analysis framework is presented. A brief overview of its ample features is provided. Some performance and architecture details are discussed.

High energy physics (HEP) is well-known not only for fundamental research, but for being an incentive for technology bleeding edge as a byproduct of its challenging demands. WWW was born at CERN [1], Grid technology is nursed in scientific environment, and petabyte scale data processing free tools are bred there.

Today HEP experiments produce petabytes of data and are in demand of a tool to process and physically analyze these data. Such tool is available since 1995 and is known as ROOT [2]. It is licensed by LGPL and is developed by worldwide recognized scientific centers (CERN, FermiLab, BNL, etc). ROOT is an object-oriented C++ framework, designed for large scale data analysis, mining and storing and analyzing petabytes of data in an efficient way [3].

Any instance of a C++ class can be stored into a ROOT file in a machine-independent compressed binary format. In ROOT the TTree object container is optimized for statistical data analysis over very large data sets by using vertical data storage techniques [4]. This container uses buckets for each tree branch, where each bucket is a continuous space in file, allowing to effectively extract data subsets (e. g. if only several values from each event are needed for current analysis) [5]. These containers can span a large number of files on local disks, the web, or a number of different shared file systems.

In order to analyze this data, user can chose from a wide set of mathematical and statistical functions, including linear algebra classes, numerical algorithms such as integration and minimization, and various methods for performing regression analysis (fitting). In particular, the RooFit package [6] allows user to perform complex data modeling and

*bircoph@gmail.com, http://lvee.org/en/abstracts/128

fitting while the RooStats library provides abstractions and implementations for advanced statistical tools. Multivariate classification methods based on machine learning techniques and neural networks are available via the TMVA package [7].

A central part of these analysis tools are the histogram classes which provide binning of one- and multi-dimensional data. Results can be saved in vector formats like Postscript, PDF or LaTeX with Metafont, or in bitmap formats like JPG, PNG or GIF [4].

Users typically create their analysis macros step by step, making use of the interactive C++ interpreter Cling (which is based on Clang and have superseded older CINT project), while running on small data samples. Once the development is finished, they can run these macros at full compiled speed over large data sets, using on-the-fly compilation, or by creating a standalone C++ program using ROOT libraries. Bindings for Python, Ruby as well as integration with R and Mathematica are available.

Finally, if HPC clusters are present, the user can reduce the execution time of intrinsically parallel tasks — e. g. data mining in HEP — by using PROOF, which will take care of optimally distributing the work over all available resources in a transparent way [4]. Grid and AFS [8] support is also available.

Besides High Energy Physics ROOT is also widely used in many other scientific fields like astronomy and biology, and also in finance and medicine [9], so it may be used in any other field requiring spectral analysis or advanced hiincenvestogram facilities.

References

[1] `http://home.web.cern.ch/topics/birth-web`

[2] `http://root.cern.ch`

[3] ROOT: An Object-Oriented Data Analysis Framework // Linux Journal, Issue 51, July 1998. `ftp://root.cern.ch/root/lj.ps.gz`

[4] ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization // Computer Physics Communications; Anniversary Issue; Volume 180, Issue 12, December 2009, Pages 2499-2512. `http://dx.doi.org/10.1016/j.cpc.2009.08.005`

[5] ROOT Users Guide. I/O chapter. `http://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideChapters/InputOutput.pdf`

[6] Roofit quickstart guide. http://root.cern.ch/drupal/sites/default/files/roofit_quickstart_3.00.pdf
[7] TMVA User's Guide. http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf
[8] Andrew File System. http://www.openafs.org/
[9] http://root.cern.ch/drupal/content/about

Test environment configuration with Ansible

Викентий Лапа, Минск, Беларусь*

Integration testing assume frequent changes in test environment such as application installation, configuration for every software build, for defects verification and reproduction. This task is not very complicated but monotonous and it complexity increases with number of hosts in test environment. Manually it can be solved only on one small range of hosts less than ten, because configuration time increases with number of nodes and can't be parallel for only person. As solution for test environment deployment it is suggestion to use Ansible configuration management application. In this presentation we review pros and cons of this product and present basic application functionality with practical examples used in testing.

Сегодня существуют десятки различных систем для управления конфигурациями. Среди них есть как проекты, которые получили широкое распространение и хорошо известны (Chef, Puppet, Salt), так и относительно новые системы. О возможностях системы управления конфигурациями Ansible и пойдет речь в данном докладе.

Системы управления конфигурациями хорошо себя зарекомендовали в областях, где требуется управление группами компьютеров, либо где окружение динамично меняется. Хорошими примерами являются облачные вычисления и кластерные системы. Тестирование кластерных файловых систем — это пример динамичной системы, состоящей из группы компьютеров.

Типичная конфигурация, с которой нам приходится иметь дело, представляет собой 6 компьютеров, а ее время жизни составляет 3–5 дней. Постоянно мы используем 5 систем т.е. всего 30 компьютеров. Поскольку процесс тестирования предполагает частую реконфигурацию систем, либо при выходе новых версий ПО, либо для верификации/воспроизведении дефектов, то необходимо

*nop@tut.by, http://lvee.org/en/abstracts/130

этот процесс автоматизировать. Отметим, что при ручной установке сложность и время установки возрастает с ростом количества хостов, и при этом параллельно можно конфигурировать до 10 узлов. Естественным решением является применение автоматизации — либо разработкой собственного средства на каком-то языке программирования, либо с помощью готового приложения. Так нами было выбрано приложение Ansible. Ее главное отличие от аналогичных систем управления конфигурациями — это отсутствие единого сервера для управления и отсутствие специального агента на хостах. Минимально необходимые требования к хосту — сервер SSH и Python 2.6. Минусом такого подхода является то, что скорость выполнения задач зависит от скорости установления соединения с узлом; однако есть дополнительные режимы для ускорения.

Для описания конфигурации систем используются два типа файлов с описанием хостов (inventory) и файлы задач (playbooks).

Inventory-файл является простым INI-файлом, хранит информацию об одном узле, группе узлов, содержит переменные, объединения узлов и группы по заданному шаблону, параметры соединения (порт, имя пользователя, IP адрес). В нашем случае мы создали 5 групп, по группе на тестовое окружение, и эти группы в свою очередь разделили на группы серверов и клиентов. Например, такое разделение позволяет применить индивидуальные настройки на серверах и общие настройки на всю группу.

Второй важный файл — это сборник заданий (playbook). Это тестовый файл в формате YAML, где описаны действия, которые будут применяться к каждому хосту, а также указана группа хостов для конфигурации. Все действия описываются именем задачи и именем модуля с параметрами. Сборник заданий можно организовать в нескольких файлах, что позволяет использовать задания повторно. В заданиях поддерживаются дополнительные элементы — такие как циклы, условные выражения, переменные.

В Ansible существует около сотни доступных модулей. Вот примеры задач, которые успешно решаются с их помощью:

– установка публичных ключей;
– заполнение файлов конфигурации;
– копирование вспомогательных скриптов для тестирования.

Иногда стандартных модулей недостаточно. Пример — конфигурация сетевых интерфейсов. По скольку в Ansible заложены возможности по дополнению функциональности системы с помощью интерфейса модулей, плагинов, либо скриптов, то можно использо-

вать различные варианты для решения этой задачи. Так изначально использовался написанный ранее bash-скрипт. Позже по мере изучения возможностей приложения эта задача была реализована нами с использованием циклов, модуля templates и переменных узлов. Также эту задачу можно реализовать в виде дополнительного модуля на любом языке программирования. Но в нашем случае предпочтение отдается Python, т.к. это позволяет использовать стандартную библиотеку Ansible, которая содержит функции для написания модулей.

В итоге за счет использования системы освободилось дополнительное время для других задач, а сама система — проста для внедрения, поддерживает достаточно большое число реализованных функций, и в случае необходимости существуют простые способы добавления новых.

Luwrain as a platform for distributing applications for handicapped people

Michael Pozhidaev, Tomsk, Russia*

The paper proposes the new conception of distributing applications for people with low vision or blind ones, which is based on accessible environment Luwrain. The offered approach involves Java and doesn't require any prior experience or practice in accessibility technologies.

Last years we can see the process when various vendors release applications, which typically play a role of clients for services, usually accessed through the Internet. These applications have a version for every popular platform, especially mobile, because it is the simplest way to bring particular service to user and make access easy and comfortable. Meantime, the problem remains actual in a context of accessibility questions. When new application of this type appears, usually there are no promises that it will be understandable by people with low vision or blind ones.

The problem is partially solved by accessibility features incorporated in particular platform but initial UI design anyway is prepared mostly for sighted people and disabled users are continuing to be in handicapped position, because there is still no any technology which could provide the interface completely suitable and understandable for them. Actually, the embedded accessibility features could be useful for people who have the experience of work with a particular platform as sighted user, because they do an assisting in terms (GUI, windows, dialogs, multitouch) of the interface, which the user should be already familiar with. Such prior explanation and introduction to new environment are sometimes difficult to do, especially for seniors or children born blind. Only the interface, which was designed for disabled users from scratch, could guarantee the complete access to services. The author thinks that it is especially important as we can see now the process of spreading the digital government services over all parts of the society as well as making popular various payment systems, which evidently could be

*msp@altlinux.ru, http://lvee.org/en/abstracts/129

very useful for everybody who is restricted in moving or consuming services in normal way.

From this point of view the accessible environment Luwrain can be considered as a platform for distributing accessible applications for blind and visually impaired people. The main advantage it can offer to potential application providers is that they do not need any experience in creating accessibility technologies. They may think of Luwrain as of a Java library which consists of classes representing the behavior of various controls, such as lists, trees, menus, forms etc, but in a way suitable for using by blind consumers. The purpose of every such class is clearly understandable and does not require any special knowledge or prior practice. The set of Luwrain classes to be created should be sufficient for constructing the interface of most applications which do not imply graphical and visual objects. Further, Luwrain library is written on Java SE and that makes possible its using on every platform with proper Java SE support. Besides that, since Java is involved in creating applications for Android, we may think that providers should have a lot of already written code on Java and the preparing of new front-end will be relatively easy and will not require significant amount of new resources for extra work.

Luwrain environment is designed in such way that gives the user the simplest and the most understandable method of interaction. When the user doesn't see a picture on the screen he/she replaces it with the image in the imagination. Luwrain takes care that this artificial picture should be the most simple as it could be, because with these circumstances simplicity means all necessary properties — to be comfortable, elegant and time efficient. The portions of the information, transmitted from Luwrain to user, are also as minimal as it is possible to be sufficient. The information transmitted mostly through speech. The screen also contains the picture with the textual data reflecting the working object, but it plays mostly a supplementary role. In Luwrain there are no any objects or pieces of data, which are inaccessible without involving the monitor.

If potential user hasn't any suitable operating system on the PC, that will not make impossible using the applications created for Luwrain, because Luwrain is distributed as a complete standalone Linux-based OS as well. The license of this ISO image is free and the chosen legal status eliminates any barriers for users to have complete access to desired services. The bootable ISO image being described should be

suited for installation without sighted help and should be shipped with accessible browser.

Finally, we would like to invite everybody who is interesting in developing this conception to discuss the question of creating the central repository or store for the applications designed for Luwrain. Having the single place to get applications from, handicapped consumers could get the generally new approach to Internet services which so far raised a lot of questions of compatibility with accessibility features.

Создание 3D-мультфильма средствами СПО

Виктория Бабахина, Рязань, РФ*

The article gives an overview about using free software in film making industry. It covers all the basic steps of creating an animated film with sample use of Blender, Krita, Gimp, Mypaint, Alchemy, etc.

В последнее время СПО все активнее применяется в киноиндустрии. Не последнюю роль в этом играет студия Blender Foundation, некоммерческая организация, занимающаяся разработкой программного пакета трёхмерного моделирования с открытым исходным кодом под названием Blender. Основателем фонда и главным разработчиком является Тон Розендаль.

Blender Foundation выпустили уже не один мультфильм, среди которых «Big Buck Bunny» и «Sintel». Мультфильмы эти были полностью, от концепта до готового продукта, созданы при помощи СПО.

Кроме Blender Foundation в последнее время все больше и больше мультипликационных студий обращаются к СПО во всех странах мира. В большинстве случаев это небольшие студии, однако есть примеры использования, к примеру Blender, такими гигантами, как Columbia Pictures при съемках фильма «Человек-Паук 2».

Разработка концепции

На этапе разработки концептов неоценимо важна возможность создать много быстрых и выразительных эскизов-идей, некоторые из которых в дальнейшем лягут в основу готового продукта. В этом отношении программа Alchemy может стать хорошим подспорьем. Еще одним полезным графическим редактором является Krita — созданная непосредственно для рисования, для этого в ней есть все необходимое и даже более того: обширный набор кистей, удобные инструменты для построения перспективы и прочие необходимые вещи. Однако, она практически не пригодна для обработки изображения. Для дальнейшей обработки можно воспользоваться GIMP.

*vitorry@gmail.com, http://lvee.org/en/abstracts/134

Раскадровка

После появления первых эскизов и написания сценария, необходима предварительная визуализация будущего мультфильма. Для этого создается раскадровка. Как правило, раскадровка выглядит как небольшие пронумерованные картинки, зачастую довольно схематичные. Однако, благодаря этим картинкам становится понятно чередование планов, тоновой разбор, монтажные стыки кадров, да и в принципе визуальный ряд сюжета. Целесообразно использование Krita или MyPaint. Помимо использования обычных контурных кистей возможно применение широкого спектра кистей, имитирующих натуральные материалы, что позволяет сделать раскадровку более «живой» и подходящей настрою мультфильма.

Аниматик

Аналогичную роль выполняет аниматик, но в отличии от ракадровки, он дает уже более конкретное представление о том, что будет происходить в мультфильме, не просто последовательность действий, но и более подробные движения персонажей. Фактически аниматик — это черновой мультфильм. Как правило, аниматиков рисуется довольно много. В случае трехмерного мультфильма первый аниматик все равно создается рисованный, как черновая последовательность картинок. Для чернового аниматика вполне возможно использовать программу для 2D-анимации sunfig. Или просто покадрово нарисовать основные фазы в Krita и собрать их в том же Blender или в любом другом видео-редакоре. Большим плюсом специальных программ для 2D-анимации является то, что есть возможность прорисовать некоторые фазы более подробно, и просто в одном редакторе, и сразу там же их и собрать.

Следующие варианты аниматика создаются уже в 3D. Выглядят они поначалу достаточно неказисто — для упрощения процесса рендеринга в них не используются текстуры, убрана большая часть мелочей. В первых вариантах аниматика отсутствует модель как таковая, вместо нее упрощенная шарнирная болванка. Анимированных движений еще тоже нет. Затем аниматик начинает прорабатываться подробнее, добавляются персонажи, текстуры, мебель. И так до финального композитинга.

Эта работа проходит уже целиком и полностью в Blender. Для оптимизации процесса все предметы и персонажи, находящиеся в

сцене, расположены в отдельных файлах. В главном документе на них присутствуют только ссылки, что значительно облегчает работу и ускоряет рендер.

Моделирование персонажей

Когда определены первые эскизы персонажей, начинается подготовка референсов для моделирования. На референсах представлены несколько видов будущей модели, а так же, при необходимости, детали костюма. Затем начинается процесс создания модели в Blender. Поскольку процесс создания высокополигональной модели человека очень труден и долог, а время часто поджимает, то имеет смыл воспользоваться программой MakeHuman. Это программа для создания модели человека методом задания параметров, таких как рост, пол, раса, форма носа и так далее. Эта программа удобна для создания простой «Болванки» будущего персонажа, на основе которой уже можно будет создать более сложную модель.

Анимация

Для анимации модели ее необходимо оснастить скелетом, так называемым «ригом». Как правило, процесс заключается в следующем. В модели в нужных местах располагаются «кости», затем они привязываются к мэшу. Для удобства, кости позже заменяют специальными рычажками — контроллерами. Получается этакая марионетка, которой очень просто и удобно управлять.

В Blender есть встроенный аддон для автоматического создания рига — Rigify. Это довольно-таки удобный аддон, которым пользуется сейчас большинство аниматоров. Сначала он автоматически задает арматуру. Ее можно отредактировать в зависимости от вида модели: увеличить, уменьшить, добавить запасную пару рук и так далее. Затем, на основе этой арматуры, автоматически генерируются контроллеры. Опять же, все можно дополнять при необходимости.

Освещение и постобработка

Одним из заключительных этапов работы над мультфильмом является чистовая анимация движений персонажей. На этом же этапе производятся настройки освещения (лайтинг). Лайтинг — это

большой отдельный этап в создании мультфильма, так как освещение — один из основных способов создания атмосферы в фильме.

На этом же этапе происходит запекание симуляций, если таковые имеются. Blender располагает внушительным набором симуляций: одежды, дыма, огня, ветра и т.д.

Мультфильм рендерится, как правило, небольшими фрагментами, которые в последствии монтируются друг с другом согласно аниматику. В финале, после рендеринга, каждая сцена проходит этап композитинга — постобработки. К готовой картинке добавляются необходимые эффекты: виньетирование, размытие, цветокоррекция. В Blender этот процесс производится при помощи нодов — так называемых узлов. Каждый узел в этой системе — какое-то действие, они последовательно соединяются друг с другом, в начале и в конце входные и выходные узлы.

Литература

[1] Хитрук Ф.С. Проффесия — аниматор М.: Гаятри, 2008.

[2] Burne Hogarth Dynamic Wrinkles And Drapery New York: Watson-Guptill Publications, 1995.

[3] Ken A. Priebe The Advanced Art of Stop-Motion Animation New York: Course Technology, a part of Cengage Learning, 2011.

[4] http://disneyfrozen.tumblr.com/tagged/concept-art Дата просмотра: 20.04.2014.

[5] http://www.blendernation.com/ Дата просмотра: 14.04.2014.

[6] http://www.disneyanimation.com/ Дата просмотра: 20.04.2014.

Zotero: аўтаматычная бібліяграфія ў WYSIWYG-рэдактарах

Антон Літвіненка, Кіеў, Украіна*

Automatic bibliography generation, being common for LaTeX users, is irrationally rarely used by scientists that prepare publications via WYSIWYG editing packages. While some linkers of BibTeX system to office packages exist, one may prefer more profound reference systems that automatize also crawling and managing citations as well as regular sorting and pattern formatting. FLOSS example of such system, named Zotero, is discussed in the presentation.

Матывацыя ды патрабаванні

Афармленне вынікаў навуковых даследванняў у выглядзе артыкулаў, манаграфій ды іншых працаў, патрабуе афармлення спісу літаратуры пад пэўныя патрабаванні. Гэтыя патрабаванні, як правіла, ўключаюць нумерацыю спасылак на літаратурныя крыніцы ў парадку іхняга ўзнікнення ў тэксце працы, а таксама фарматаванне тэксту спасылкі паводле пэўнага шаблону, які адрозніваецца ў розных рэдакцыях. Парадак узнікнення можа істотна мяняцца пад час працы над тэкстам, рознасць шаблонаў афармлення замінае ўжытку спасылак з аднаго артыкулу пры напісанні іншага. Ручная перанумероўка ды перафарматаванне спасылак з'яўляецца маруднай працай, якая лёгка прыводзіць да памылак.

У выдавецкіх сістэмах на базе LaTeX сродкі аўтаматычнага фарматавання з'яўляюцца тыповай і неад'ёмнай часткай сістэмы. Уключэнне спасылкі рэалізуецца праз адпаведную каманду працэсара. У выпадку WYSIWYG-працэсараў падобны функцыянал даводзіцца рэалізоўваць праз вонкавую сістэму ды плагіны для ёйнай інтэграцыі з тэкставым працэсарам. З улікам адсутнасці тэкставых камандаў патрэбныя адмысловыя аб'екты для захавання дадзеных спасылкі (у выпадку OpenOffice ужываюцца зноскі (reference marks) альбо закладкі (bookmarks), ў MS Word — палі (fields) альбо закладкі (bookmarks).

*tenebrosus.scriptor@gmail.com, http://lvee.org/en/abstracts/127

Кампаненты Zotero

1. Сістэма імпарту спасылак. Дазваляе дадаць спасылку да бібліяграфічнай базы «адным націскам» са старонкі навуковага часопісу ці сістэмы пошуку навуковай літаратуры, альбо праз зазначэнне DOI, ISBN, Pubmed ID і г.д., а таксама з іншых бібліяграфічных базаў (напрыклад, базы для BibTeX).
2. Арганізатар бібліяграфічнай базы. Дазваляе ствараць, выдаляць, рэдагаваць і сартаваць спасылкі, а таксама арганізуе сінхранізацыю праз анлайн-сховішча.
3. Плагін інтэграцыі з тэкставым працэсарам.
4. Дадатковыя кампаненты.

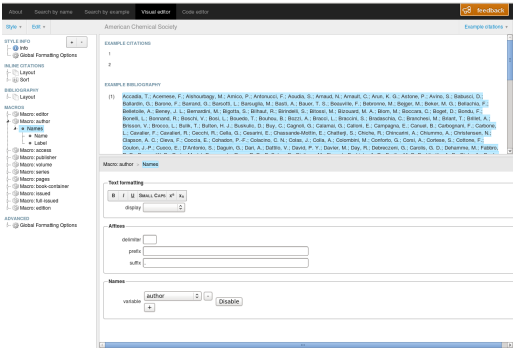
CSL. Рэпазіторыі стыляў

Фарматаванне тэксту спасылкі задаецца шаблонам, апісаным з дапамогай адмысловае мовы CSL (Citation Style Language), што базуецца на мове разметкі XML [1]. Напрыклад, фрагмент апісання стылю Journal of American Chemical Society выглядае наступным чынам:

```
<if type="article-magazine">
         <group delimiter=" ">
           <text variable="container-title" font-style=
"italic" suffix="."/>
           <text macro="edition"/>
           <text macro="publisher"/>
           <text macro="full-issued" suffix=","/>
           <text macro="pages"/>
         </group>
       </if>
       <else-if type="thesis">
         <group delimiter=", ">
           <group delimiter=". ">
             <text variable="title"/>
             <text variable="genre"/>
           </group>
           <text macro="publisher"/>
           <text macro="issued"/>
           <text macro="volume"/>
           <text macro="pages"/>
```

```
    </group>
  </else-if>
```

За час існавання праэкту напрацавана вялікая колькасць стыляў для разнастайных навуковых часопісаў, якія можна знайсці ў адмысловым рэпазіторыі [2]. У выпадку, калі патрэбнага стылю бракуе, яго можна стварыць уласнаруч праз адаптацыю наяўных. Для гэтага распрацаваны візуальны рэдактар (рыс. 1), які дазваляе паводле прыкладу спасылкі знайсці найбольш падобны стыль і змяніць яго пад свае патрэбы, не кранаючы XML-код [3].



Іл. 11.1. Акно візуальнага рэдактара: фарматаванне імёнаў аўтараў

Практычнае выкарыстанне

Zotero рэалізаваны ў выглядзе плагіна для Firefox (найбольш поўнафункцыянальны) ды standalone-праграмы.

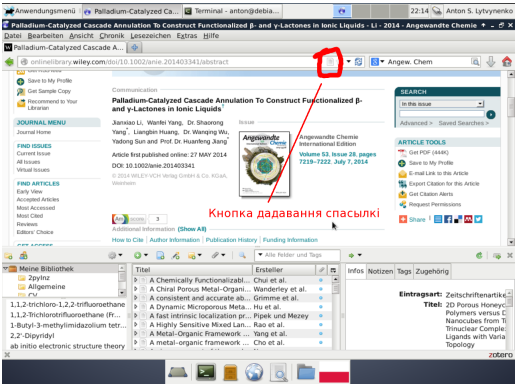
Напаўненне базы бібліяграфіі

Найбольш тыповы і зручны спосаб для артыкулаў: зайсці на старонку з анатацыяй адпаведнага артыкулу на афіцыйным сайце часопісу і націснуць кнопку захавання спасылкі (рыс. 2).

Для кніг: ўвесці ISBN.

Кіраванне базай

Спасылкі сартуюцца па тэках (адна спасылка можа знаходзіцца ў некалькіх тэках). Акно плагіна адлюстроўвае спіс тэкаў, спіс спасылак у абранай тэцы ды падрабязнасці абранай спасылкі (рыс. 1). Магчымы хуткі пошук па зместу спасылак.



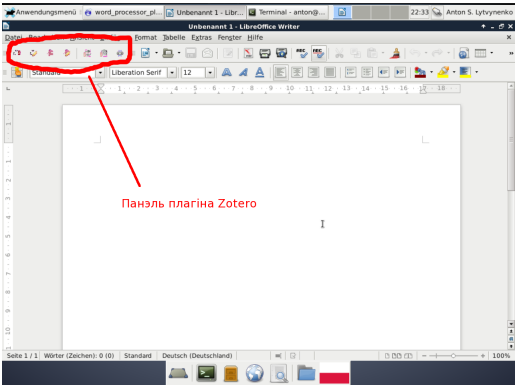
Іл. 11.2. Панэль Firefox-плагіна Zotero і аўтаматычнае дадаванне спасылкі са старонкі часопіса Angewandte Chemie

Дадаванне спасылкі ў тэкст

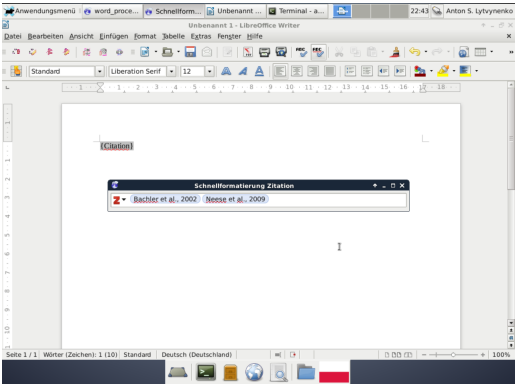
Інтэграцыйны плагін дадае кнопкі ў панэль офіснага працэсара (рыс. 3): дадаць спасылку (пад час першага выкарыстання прапануецца выбраць шаблон фарматавання), рэдагаваць спасылку, дадаць спіс літаратуры, рэдагаваць спіс літаратуры, панавіць спіс, усталяваць опцыі для дакуманту, выдаліць коды палёў (пераўтварае фарматаваныя спасылкі ў звыклы тэкст). Устаўка ды рэдагаванне магчымае праз выбар спасылак са спісаў, а таксама праз хуткі пошук у акенцы дадавання (рыс. 4) Выдаленне спасылкі — праз выдаленне адпаведнага аб'екту ў тэксце.

Палі і закладкі. Copy/Paste/Delete. Абмен Open/Libre Office ды MS

Zotero можа захоўваць дадзеныя пра спасылку ў зносках/палях альбо закладках. Варыянт закладак забяспечвае сумяшчальнасць



Іл. 11.3. Інтэграцыя Zotero ў LibreOffice



Іл. 11.4. Устаўка спасылкі хуткім пошукам

пры супольнай працы ў OpenOffice ды MS Word, але не дазваляе аперацыі капіявання-ўстаўкі над спасылкамі ў тэксце. Больш стандартным рэжымам з'яўляецца выкарыстанне зносак/палёў, якія дазваляюць працу са спасылкамі як са звыклымі элементамі тэксту — капіяванне, ўстаўка, перанос (у тым ліку з файлу ў файл).

Некаторыя брудныя падрабязнасці

Іншы раз Zotero праз тыя ці іншыя прычыны фарматуе спасылку няправільна. Асабліва часта гэта стаецца пры наяўнасці ў

спасылцы верхніх і ніжніх індэксаў, якія адлюстроўваюцца без адпаведнага фарматавання. Магчымасць «рэдагаваць бібліяграфію» дазваляе адфарматаваць тэкст спасылкі ў спісе літаратуры ўручную. Пры гэтым яе тэкст (а таксама нумар, зазначаны ў спісе літаратуры) прыпыняе абнаўляцца аўтаматычна (нумар у асноўным тэксце абнаўляецца) — пажадана не забыцца на гэта ў фінальнай версіі дакумента.

Магчымасць ручнога фарматавання дае таксама магчымасць розных нестандартных спасылак (напрыклад, кампазітнай спасылкі, пры якой пад адным нумарам цытуецца некалькі спасылак (a, b, ...)). Карыстальнік можа спачатку згенераваць праз сістэму індывідуальныя спасылкі, пасля стварыць спасылку на нейкі dummy-аб'ект (напрыклад, нататку), спаслацца на яго і адфарматаваць тэкст уручную, ўставіўшы згенераваны тэкст пасля адпаведнага рэдагавання.

Над дакументам, у які з дапамогай Zotero ўстаўленыя спасылкі, можна працаваць і на кампутарах, дзе Zotero не ўсталявана, ў тым ліку ў версіях рэдактараў, якія не падтрымліваюцца Zotero (напрыклад, MS Word раней за версію 2003), ў тым ліку выконваць аперацыі капіявання, пераносу ці выдалення спасылак у тэксце. Пасля рэдагавання на такой сістэме дастаткова адчыніць дакумант у сістэме з усталяваным Zotero і панавіць бібліяграфію.

Высновы

FLOSS-сістэма Zotero, што інтэгруецца ў папулярныя офісныя тэкставыя працэсары, дазваляе аўтаматычнае фарматаванне і перанумероўку спасылак на літаратурныя крыніцы, што істотна палягшае працу навукоўцаў пры афармленні вынікаў іхняе працы. У адрозненні ад працы BibTeX і заснаваных на ім сістэм, Zotero дапоўненая кампанентамі аўтаматызаванага збору ды арганізацыі бібліяграфічных спасылак, што з'яўляецца мэтазгодным для WYSIWYG-сістэмы і таксама палягшае працу.

Літаратура

[1] http://citationstyles.org/
[2] https://www.zotero.org/styles
[3] http://editor.citationstyles.org/searchByExample/

Применение популярных протоколов и свободного ПО в управлении мобильным роботом

Андрей Дунец, Брест, Беларусь*

We consider the problems of telemetry, positioning and control of the mobile robot for rivers and lakes monitoring. The project uses Bluetooth technology to transmit the telemetry data. RTKLIB library implementation of Real-Time Kinematic algorithm is used for positioning. Current version of the control system is based on the Ardupilot project.

Мониторинг водоемов актуален для решения самых разных задач. Начиная от построения профиля дна водоема для прогнозирования паводков и обеспечения безопасности судоходства и заканчивая контролем состояния воды в интересах рыболовецких хозяйств и служб охраны природных ресурсов. Применение мобильных водоплавающих роботов позволяет автоматизировать процесс мониторинга и значительно снизить его себестоимость.

На данный момент робот и его система управления находится в разработке. Текущие решаемые задачи это телеметрия, позиционирование и управление роботом.

Телеметрия обязательная часть подобного комплекса, так как без неё нормальная разработка ПО и отладка оборудования сильно затруднена. Выбор был сделан с расчетом на доступные популярные решения, так как особенности этих технологий хорошо известны и они недороги в отличии от решений, которые основаны на закрытых протоколах.

Первоначальным вариантом было использование Bluetooth Class 1. Заявленная дальность до 100 метров, дешевизна оборудования, простота настройки позволили поставить несколько экспериментов по передаче телеметрических данных. Использовался профиль SPP (Serial Port Profile). На стороне робота микроконтроллер автопилота связывался с компьютером через Bluetooth модем BTM-222, который полностью реализует весть стек необходимых беспроводных протоколов профиля SPP, предоставляет TTL

*dunets@gmail.com, http://lvee.org/ru/abstracts/143

UART-интерфейс и управляется AT-командами. На стороне компьютера применялся встроенный адаптер Bluetooth. Выяснилось, что приемо-передатчики, кроме заявленных характеристик должны быть оснащены соответствующими антеннами. Удалось использовать на стороне робота антенну от адаптера WiFi (частотный диапазон один и тот же) и практическая полученная дальность составила 30 метров.

Эксперименты показали, что Bluetooth можно использовать для передачи телеметрической информации на небольшие расстояния. Но профиль SPP, будучи простым в использовании, позволяет передавать только один поток данных. На практике нужно как минимум два потока: поток с приемника GPS и поток консоли. Можно использовать протокол, который поддерживает подобное мультиплексирование. Возможны так же экзотические варианты: пробросить TCP трафик через PPP.

Дополнительно выявились проблемы с восстановлением связи при утере соединения: пересопряжение устройств требует поддержки в прикладном ПО, что неудобно. Сейчас как наиболее оптимальное решение для следующего шага рассматривается переход на WiFi. WiFi обеспечит большую скорость передачи данных. Можно использовать более мощные антенны. Штатно доступный стек TCP/IP позволит мультиплексировать данные. При этом возрастет энергопотребление, что является минусом данного решения. Эксперименты покажут, какой вариант наиболее оптимален.

Для позиционирования робота в акватории водоема применяется GPS. Для уточнения координат используется алгоритм RTK (Real-Time Kinematic), который реализован в открытой библиотеке RTKLIB (`rtklib.com`). Для получения данных GPS применяется приемник NEO-6M фирмы u-blox и штатные утилиты из состава RTKLIB. Приемник настраивается с помощью хаков, разработанных сообществом OpenStreetMap. БД корректировок берутся с серверов проекта IAG Reference Frame Sub-Commission for Europe (`http://www.euref.eu/`), базовые станции которого находятся на достаточно близком расстоянии от Беларуси.

Первоначальное решение по управлению, на которое был сделан упор — использовать существующие автопилоты, в которых используется открытое ПО. Одним из таких проектов является APM Autopilot Suite (`http://ardupilot.com`). Его особенности:

– открытые исходные тексты ПО автопилота и базовой станции (лицензия GNU GPL);

– базовые аппаратные решения основаны на открытой платформе Arduino (Atmel Atmega), но более продвинутые используют 32-разрядные контроллеры ARM Cortex и платы с закрытым дизайном;
– активное сообщество пользователей DIY Drones, которые делятся наработками и обмениваются опытом (`diydrones.com`);
– используется открытый протокол MAVLink (Micro Air Vehicle Communication Protocol, `http://qgroundcontrol.org/mavlink/start`);
– реализовано управление бесплотниками с разной кинематикой: несколько видов коптеров, вертолет, самолет, автомобиль, что положительно сказалось на архитектуре ПО: код представляет из себя несколько программ управления для разной кинематики и общую библиотеку алгоритмов;
– в алгоритмах доступно множество полезных решений для создания собственного автопилота: от драйверов устройств (датчики, исполнительные механизмы, телеметрия и т.п.) до математики (ПИД, фильтры Калмана и т.д.).

Был приобретен комплект оборудования APM 2.5 и поставлено несколько экспериментов программированию автопилота. Выяснилось, что хотя код написан довольно качественно, документирован он слабо и отлаживать на микроконтроллере сложную математику весьма трудоемко. Также оказалось, что решение на базе микроконтроллеров не подходит для задач управления с нашими требованиями по позиционированию. Для достижения автономности на борту должен быть 32-разрядный компьютер с тактовой частотой в несколько сотен мегагерц, чтобы вести расчеты позиции по алгоритму RTK. На такой компьютер можно установить полноценную ОС, которая упростит остальные эксперименты. При этом от микроконтроллера для управления вводом/выводом нельзя отказываться, так как есть перечень задач, которые оптимально выполнять именно на нем. Сейчас рассматривается возможность дополнить бортовую вычислительную систему платой микрокомпьютера под управлением Linux. Выбор будет сделан из двух вариантов: Raspberry Pi и Beagle Bone Black. Скорее всего, выбор будет остановлен на Rapberry Pi, так как у этой платы проще подсистема питания, что упрощает стыковку со штатным аккумулятором робота.

Результаты экспериментов доступны на Bitbucket:
`https://bitbucket.org/pondskaterteam/pondskater`

Smart GreenHouse

Дмитрий Ясевич, Василий Слапик, Павет Вервенко,

Дмитрий Огиевич, Минск, Беларусь*

«Java for Farmers»: Greenhouse monitoring and automation, using Java, Raspberry Pi, Linux and multiple sensors. Smart Greenhouse Project is a Oracle IoT winner 2014 in professional category.

История проекта

Ни для кого не секрет, что «умные» программные решения (дома, парники, и т. д.) находят все большее применение в реальном мире. Узнав о существовании Java Embedded для создания «Интернета вещей», мы загорелись идеей попробовать ее в деле. После недолгого обсуждения в качестве объекта для экспериментов была выбрана «умная» теплица.

Причин было несколько. Первая из них — умными домами занимаются широкий круг инженеров и энтузиастов, начиная от студенческих клубов и заканчивая серьезными IT-компаниями, поэтому здесь было тяжело создать что-то действительно новое и интересное.

Вторая, но не менее значимая причина, заключается в том, что Беларусь — это страна, в которой хорошо развит аграрный сектор. Наша команда решила быть патриотичной и создать устройство достаточно простое, но при этом потенциально полезное для использования в сельском хозяйстве. Таким образом выбор пал на теплицу, как точку приложения наших сил.

Java Embedded

Java уже успела зарекомендовать себя в качестве успешной платформы для решения множества задач, включая и «умные» системы. Если охватывать весь спектр техники, то можно насчиать более 10 млрд устройств, использующих Java. При этом подавляющая

*dzmitry_yasevich@epam.com, http://lvee.org/ru/abstracts/131

часть таких устройств так или иначе базируются на *nix платформах.

Почему все-таки стоит использовать Java для встраиваемых систем; ведь на первый взгляд у Java недостатков гораздо больше, чем преимуществ:

- Java является одной из самых популярных платформ для разработки приложений;
- Оптимизирована для Embedded решений;
- Высокопроизводительные, переносимые приложения;
- Свободно распространяемые инструменты разработчика;
- Проверенная модель безопасности.

Как показала практика, для создания «умной» теплицы с помощью Java Embedded достаточно скромных ресурсов Raspberry Pi, работающей под управлением Linux.

Функциональные возможности теплицы

К числу основных особенностей проекта относятся:

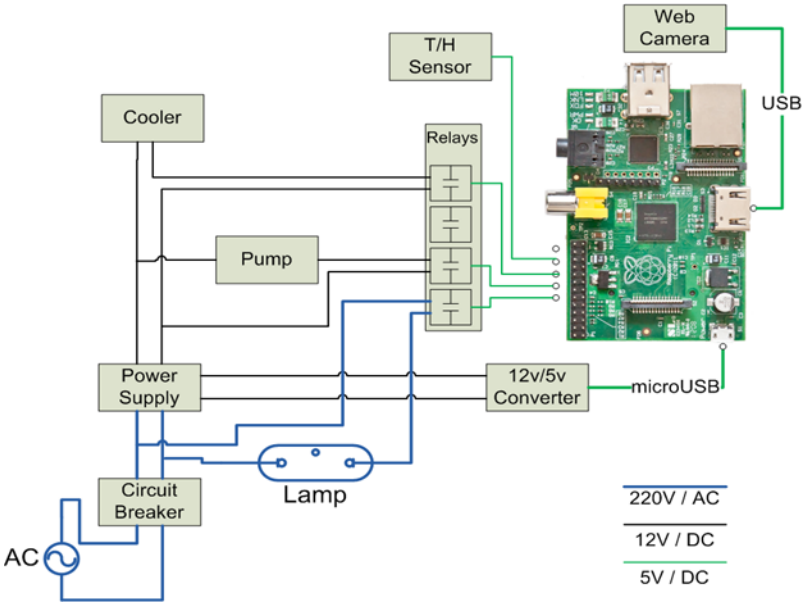
- Контроль и управление светом.
- Контроль полива.
- Контроль температуры и влажности.
- Удаленный мониторинг текущего состояния теплицы.
- Автоматическое управление теплицей.
- Автоматический процесс фотографирования роста растений.
- Низкая потребляемая мощность.
- Защита от коротких замыканий и отключения электричества.

Таким образом, наша разработка на данный момент представляет собой полнофункциональную автоматизированную систему, которая позволяет выращивать комнатные растения, сохраняя душевное спокойствие владельца теплицы. Обеспечивается удаленное управление и мониторинг света, температуры и влажности. Также запланирована возможность дистанционной проверки текущего процесса роста в режиме онлайн.

Реализация

При создании нашего проекта мы старались использовать открытые и свободные компоненты и технологии: Raspberry Pi, Java Embedded, Raspbian, pi4j, Jetty и нескольких сенсоров.

Электрическая схема Smart GreenHouse показана на рисунке.



Raspbian — это операционная система, основанная на Debian и оптимизированная для Raspberry Pi, а pi4j — библиотека для работы с аппаратной частью Raspberry Pi.

Ниже приведен пример кода на Java для датчика влажности и температуры при использовании pi4j:

```
// инициализация
GpioController gpio = GpioFactory.getInstance();
GpioPinDigitalOutput light =
gpio.provisionDigitalOutputPin(
RaspiPin.GPIO_07, "Light", PinState.LOW);
// подключились к пину 7

light.setShutdownOptions(true, PinState.LOW,
PinPullResistance.OFF); /* задали опцию, чтоб на выходе
из приложения этот пин отключался (чтоб свет гас) */

// управление
light.high(); // включить пин
light.low(); // выключить
```

Текущий статус и планы

На данный момент проект все еще развивается — добавляем поддержку разных датчиков, решаем проблемы, возникающие при совместной работе нескольких таких устройств. Также создаем специализированный дистрибутив на базе Yocto Project, содержащий все необходимое для работы автоматизированной теплицы «из коробки».

Полный исходный код управляющей части проекта доступен по адресу `https://bitbucket.org/Temdegon/greenhouse`

clsync progess: security and porting to FreeBSD

Дмитрий Окунев, НИЯУ МИФИ, Москва, РФ*

The report focuses on methods to increase a level of security of "clsync"(free live sync utility). Multiple approaches to reduce risks of attack were used, and practical experience of applying these approaches is given. Used methods include unsharing unnecessary resources, splitting the process to privileged and non-privileged threads and using Linux-specific security-related API. Also problems of porting Linux-based clsync to FreeBSD are described, including variants and problems of FS monitoring in FreeBSD.

Введение

Проект clsync возник как альтернативный высокопроизводительный инструмент для синхронизации контейнеров LXC между узлами HA-кластера и их резервного копирования, изначально ориентированный под внутренние нужды отдела UNIX-технологий НИЯУ МИФИ [1], [2]. Данная статья является отчётом о достижениях по развитию программы «clsync» в вопросах повышения уровня безопасности и портируемости на другие ОС в 2014 году.

Средства защиты

Процесс отслеживания изменений на ФС часто требует привилегий root (или capability «CAP_DAC_READ_SEARCH» [3] в Linux). Однако известные программы [4] для синхронизации на базе inotify [5] имеют достаточно малое сообщество и при этом не имеют никаких встроенных механизмов повышения безопасности (сброс лишних привилегий, namespaces и т.п.). В результате становится маловозможной настройка синхронизации посредством inotify в системах, требующих особой защиты. Далее будут рассмотрены средства для минимизации последствий эксплуатации уязвимостей, реализованные в «clsync» летом 2014-го года.

*xaionaro@gmail.com, http://lvee.org/ru/abstracts/138

Задача

Задача заключается в минимизации последствий обнаружения и эксплуатации уязвимостей в clsync злоумышленником.

Предполагается, что:

- Необходимо предоставить процессу «clsync» полный доступ к некоторому файловому дереву, содержащему объекты (файлы, директории и т.п.) с произвольными наборами прав.
- Злоумышленник имеет возможность управлять содержимым данного файлового дерева, но не имеет других путей для манипуляции над процессом «clsync».
- «clsync» не наблюдает за файлами, чтение которых злоумышленником привело бы к ущербу.
- Процессу «clsync» необходимо запускать внешний процесс (например «rsync») для осуществления синхронизации.

Практическим применением данной задачи является хостинг на базе LXC, требующий синхронизации на другой узел для реализации High Availability [1, 6]. А для синхронизации запускается по одному процессу «clsync» на каждый контейнер. В такой конфигурации содержимым контейнера управляют малодоверенные пользователи, которые могут иметь интерес выйти за пределы предоставленных им пространств имен (namespaces), эксплуатируя уязвимость в «clsync» (который в свою очередь запущен с хост-системы, то есть извне пространств имен контейнера).

Решение

В качестве мер для повышения безопасности предлагается:

- Применить unshare() [7] для отсоединения от пространств имен хост-системы для сети, IPC и других ресурсов.
- Сбросить привилегии с помощью setuid()/setgid(), предварительно сохранив capability «CAP_DAC_READ_SEARCH» для сохранения полного доступа к файловому дереву.
- Производить chroot(), pivot_root() [8] и umount() для запрета доступа к файлам вне наблюдаемой директории.
- Разделять процесс на «большой непривилегированный» и «малый привилегированный».
- Использовать seccomp filter [9] для запрета лишних системных вызовов (syscalls) внутри непривилегированного потока.

unshare(). Применение unshare() позволяет запретить доступ к сети (и сервисам, работающим на localhost), IPC и другим ресурсам. Это необходимо, чтобы предотвратить взлом процессов по цепочке (когда злоумышленник пытается взламывать другие сервисы через уязвимость clsync).

Сброс привилегий. Сброс привилегий — тривиальная и очень распространенная процедура. В «clsync» предлагается производить сброс с помощью setuid() и setgid() с предварительным сохранением capability «CAP_DAC_READ_SEARCH» для доступа к файлам. Однако в рамках сформулированной задачи необходимо решить следующие проблемы:

1. Запускаемому процессу для синхронизации (например, «rsync») аналогично необходим доступ на чтение всего синхронизируемого файлового дерева. И наследовать для этого capability «CAP_DAC_READ_SEARCH» недостаточно, так как кроме этого необходимо его активировать (добавить в effective capabilities bitmask). Но такого рода процессы обычно не имеют поддержки активации capabilities. Поэтому данную проблему необходимо решить возможностями «clsync». Для решения данной проблемы можно либо добавить в связку sudo (или аналог), или делать setuid()/setgid() назад на root. Использование дополнительного ПО существенно снижает уровень безопасности, поэтому setuid()/setgid() является более желательным вариантом. Но сохранение capabilities «CAP_SETUID» и «CAP_SETGID» фактически полностью ликвидирует защиту за счёт сброса привилегий. Чтобы решить данную проблему предложен способ разделения процесса на «большой привилегированный» и «малый непривилегированный».
2. «CAP_DAC_READ_SEARCH» предоставляет доступ на чтение всех файлов, что недопустимо в рамках сформулированной задачи. В качестве решения проблемы предлагается использовать chroot(), pivot_root() и umount().

Namespace файлового дерева. Как уже говорилось, для запрета доступа к файлам вне наблюдаемой директории предлагается использовать pivot_root(), chroot() и umount(). В автоматической форме это делается по следующему алгоритму:

1. Открепление от namespace точек монтирования с помощью unshare().
2. Монтирование chroot-окружения в отдельную директорию с помощью bind (с опцией режима «только на чтение»).
3. Переход в данную директорию.
4. Вызов pivot_root() для последующего отмонтирования rootfs хост-системы.
5. Вызов chroot() на данную директорию.
6. Отмонтирование старого rootfs (что спровоцирует каскадное отмонтирование всех лишних точек монтирования).

Данный подход даёт возможность достаточно надёжно защитить лишние файлы от чтения/изменения процессом «clsync». Однако существует риск, вызванный уязвимостью, описанной в статье [10]. Данная проблема решается за счёт разделения процесса на «большой непривилегированный» и «малый привилегированный».

Splitting. Технически разделение процесса «clsync» на «большой непривилегированный» и «малый привилегированный» предполагает запуск дополнительного потока (thread) для обращения к системным вызовам, требующим привилегий выше чем «nobody». С точки зрения безопасности, это способ многократно снизить площадь для атаки на clsync. Если предположить, что потенциальные уязвимости распределены по всему коду равномерно, то данный подход позволяет ликвидировать последствия взлома почти полностью для более чем 90% уязвимостей, и чем меньше будет код привилегированного потока, тем выше окажется эффективность метода. Однако данный подход требует применения блокировок и передачи сообщений между потоками, что существенно снижает производительность (время выполнения увеличивалось в несколько раз).

Для уменьшения потерь производительности был реализован механизм блокировок, комбинирующий pthread_mutex_* и spinlock, что позволило достичь на многоядерных системах времени выполнения близкого к прежнему (без данного разделения потоков), но со значительно большим расходом ресурсов CPU.

Однако, так как разделение производится при помощи pthread, то у непривилегированного потока есть доступ ко всей памяти привилегированного потока, включая доступ к stack и т.п. Это позволяет без особых сложностей провести атаку на привилегированный поток из непривилегированного. Чтобы решить данную проблему, предлагается использовать seccomp filter для запрета вы-

зова mprotect из непривилегированного потока, а также «забыть» максимальное количество информации о привилегированном потоке. Если первое является реальным средством защиты, то второе — лишь способ усложнить эксплуатацию потенциальной уязвимости.

Seccomp. Seccomp, как уже было сказано выше, предлагается использовать для фильтрации лишних системных вызовов для непривилегированного потока, что позволяет:

- защитить привилегированный поток с помощью mprotect [11].
- дополнительно снизить последствия эксплуатации уязвимости, так как набор действий злоумышленника будет ограничен до конкретного короткого списка системных вызовов.

Портирование на FreeBSD

FreeBSD (как и другие BSD-системы) до сих пор активно используются для реализации различных сервисов высокой доступности, однако в стандартном дереве портов данной ОС не предоставляется никаких production-ready решений для живой синхронизации файлов (без использования специальных файловых систем). Далее будет описаны основные проблемы при портировании программы для живой синхронизации «clsync» под FreeBSD.

Оригинальный «clsync» использует inotify для наблюдения за ФС, однако данный API специфичен для Linux и не предоставляется во FreeBSD. В результате при портировании в качестве альтернатив для наблюдения за событями на ФС были выбраны «kqueue» [12], «bsm» [13] и «dtrace» [14].

Основными проблемами при использовании «kqueue» являются:

- использование большого количества файловых дескрипторов (в сравнении с inotify);
- невозможность получать детали по создаваемым файлами и директориям (что было решено полным пересканированием директории при появлении в ней нового объекта);
- необходимость «вручную» вычислять переносы файлов и директорий (и различать их с созданием жестких ссылок, т.е. hard links);
- множество сложноучитываемых эффектов (например, необходимо учитывать, что нельзя открывать pipes).

Основной проблемой при использовании «bsm» является требование глобальной перенастройки auditd.

А «dtrace», как выяснилось в ходе реализации его поддержки в «clsync», реализован не в соответствии с оригинальным «dtrace» [15]. Причем различия построенным таким образом, что становится очень затруднительным получить полный путь файла, которому соответствует пойманное событие. В результате, реализация «dtrace» во FreeBSD непригодна для наблюдения за событиями ФС.

На данный момент самая стабильная работа clsync во FreeBSD обеспечена с применением библиотеки libinotify-kqueue [16], которая представляет собой реализацию API inotify на базе kqueue.

Литература

[1] Окунев Д.Ю. «clsync — live sync utility», материалы Международная конференция разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe (LVEE Winter 2014), http://lvee.org/en/reports/materials_lvee_winter_2014

[2] «file live sync daemon based on inotify, written in GNU C», https://github.com/xaionaro/clsync

[3] http://linux.die.net/man/7/capabilities

[4] «Manual to Lsyncd 2.1.x», https://github.com/axkibe/lsyncd/wiki/Manual-to-Lsyncd-2.1.x

[5] «inotify – monitoring file system events», http://linux.die.net/man/7/inotify

[6] Окунев Д.Ю. «Опыт внедрения отказоустойчивого web-кластера для портала приёмной комиссии НИЯУ МИФИ», научная сессия НИЯУ МИФИ, 2012, http://www.pandia.ru/text/78/343/297.php

[7] http://linux.die.net/man/2/unshare

[8] http://linux.die.net/man/2/pivot_root

[9] http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/prctl/seccomp_filter.txt

[10] «Выявлена уязвимость, позволяющая выйти за пределы контейнеров Docker», http://www.opennet.ru/opennews/art.shtml?num=40046

[11] «mprotect – set protection on a region of memory», http://linux.die.net/man/2/mprotect

[12] «kqueue, kevent — kernel event notification mechanism», http://www.freebsd.org/cgi/man.cgi?query=kqueue&sektion=2

[13] «auditd — audit log management daemon», http://www.freebsd.org/cgi/man.cgi?query=auditd&sektion=8
[14] «dtrace — DTrace dynamic tracing compiler and tracing utility», http://www.unix.com/man-page/freebsd/1/dtrace/
[15] «DTrace Built-in Variables», Oracle, http://docs.oracle.com/cd/E18752_01/html/819-5488/gcfpz.html
[16] «NetBSD Google Summer of Code 2011 project (#2)», https://github.com/dmatveev/libinotify-kqueue

Особенности коррекции оптических искажений в цифровой фотографии

Алексей Бабахин, Рязань, РФ*

The article talks about the distortion correction in digital images, as well as cases where correction is used to solve applied problems.

Любой объектив вносит искажения в формируемое им изображение. Чем дороже оптика, тем меньше искажений. Но полностью избавиться от них невозможно.

- Виньетирование — затемнение изображения по краям кадра.
- Хроматические аберрации — «расслоение» изображения по цветовым каналам из-за различных углов преломления у света с разной длиной волны. Проявляется в виде цветного ореола вокруг контрастных мест.
- Дисторсия — искривление изображения, вызванное неравномерным линейным увеличением при отклонении от оптической оси. Из-за дисторсии прямые линии на кадре становятся изогнутыми.

С развитием цифровой техники появилась возможность строить математические модели оптических искажений и исправлять их. Помимо общего повышения качества фотографий, расчет и устранение аберраций критически необходимы для решения множества практических задач, таких как компьютерное зрение (CV — Computer Vision), фотограмметрия, объединение нескольких фотографий и создание панорам. Точные расчеты на основании фотографий без коррекции искажений невозможны.

С точки зрения различных расчетов наиболее важным является исправление дисторсии. виньетированием и хроматическими аберрации зачастую можно либо пренебречь, либо они исправляются некоторыми камерами прямо в процессе съемки, если камера знает калибровки для текущего объектива.

Существует несколько математических моделей, описывающих дисторсию. Пожалуй, самой популярной моделью является PTLens, изначально разработанная доктором Хельмутом Дерша (Helmut

*tamerlan311@mail.ru, http://lvee.org/ru/abstracts/136

Dersch) в Panorama Tools. На данный момент эта модель является основной в библиотеке LensFun. В свою очередь эту библиотеку используют множество популярных открытых фоторедакторов — UFRaw, Darktable, Rawstudio, Digikam/Kipi, GimpLensfun, Photivo и оболочка для создания панорам Hugin. Библиотека имеет постоянно пополняемую базу объективов, которая облегчает исправление искажений. Однако есть возможность и самостоятельно создать профиль для своего объектива.

Немного особняком стоит библиотека компьютерного зрения OpenCV, которая использует свою собственную математическую модель, описывающую дисторсию. Ту же самую модель использует и Blender для реконструкции и привязки живого видео к 3D-сцене.

Проблема заключается в том, что LensFun не поддерживает модель дисторсии, используемую в OpenCV. Да и при наличии такой поддержки, прямой конвертации одной модели в другую добиться невозможно. Поэтому на данный момент нет возможности использовать обширную базу объективов и инструменты для профилирования при работе с видео в Blender и в других разработках, использующих библиотеку OpenCV. Перспективной выглядит идея подбора коэффициентов одной модели на основании коэффициентов другой модели методом наименьших квадратов — например, при помощи библиотеки ceres-solver.

Редакторы фотографий, как правило, могут использовать только уже готовые данные калибровки объективов. Сшиватель панорам Hugin или Blender могут подбирать приблизительные коэффициенты для коррекции дисторсии в процессе своей работы. Такое поведение обусловлено работой сразу с несколькими фотографиями (или видео), которые позволяют сопоставлять между собой разные ракурсы. Тем не менее, предварительная аккуратная калибровка объектива специальными мишенями позволяет повысить точность расчетов, качество результата и снизить суммарные трудозатраты.

Гарантированное уничтожение информации

Виталий Балашов, Харьковский НИИ

судебных экспертиз, Харьков, Украина*

Today HDDs help people to save huge volumes of information. People often use already used HDDs to save money. If the information stored in the HDD should not be accessible to a new HDD owner (or other persons), it should be utilized by special methods. Use of a special utilization method is important for information security, because a lot of filesystems actually don't wipe information, but only remove (delete) it. Information that was not wiped, is always at risk of recovery.

Человек издавна старался сохранять информацию. Использовал для этого все возможные способы и всё время их совершенствовал в соответствии со своими потребностями в коммуникации. Рисунок на скале в пещере не поддавался транспортировке и человек придумал использовать различные каменные дощечки, которые уже легко перенести и показать своему собрату. Но дощечки тяжелые и неудобные, поэтому человек решил использовать более легкие варианты — из дерева или аналогичного материала. И весь этот долгий процесс эволюции носителей информации привел нас к широко распространенным на сегодняшний день способам хранения информации с помощью фиксации состояния магнитного поля и дальнейшего его считывания. К сожалению, в таких носителях человек уже не может самостоятельно считывать информацию, обязательным становится использование устройства, умеющего интерпретировать магнитное поле в информацию, воспринимаемую человеком. Но эта жертва простительна, учитывая современные требования к уровню коммуникации.

Мы научились хранить огромные объемы самых разнообразнейших типов информации десятками лет и считывать их практически в режиме реального времени, находясь при этом на другом континенте, а то и за пределами планеты.

*vitaly.balashov@gmail.com, http://lvee.org/ru/abstracts/141

Оперируя носителями информации, человек всегда хотел иметь возможность ее гарантированного уничтожения. И если с краской на скале, березовой берестой, бумагой и другими носителями, хранящими информацию в виде, доступном прямому восприятию человеком, все достаточно просто, то с магнитными и транзисторными накопителями возникает вопрос: а действительно ли информация полностью и безвозвратно уничтожена?

Практически все существующие файловые системы не уничтожают информацию фактически, когда ее уничтожают пользователи штатными средствами. Это позволяет добиться более высоких показателей производительности носителя и продления срока его службы. Информация не уничтожена, а удалена от пользователя, ему более не доступна, результат достигнут. Когда, с технической точки зрения, разрушить эту информацию будет целесообразным, тогда она и будет уничтожена. Бросить что-либо как есть всегда дешевле, чем проводить утилизацию (а если в ней нет критической необходимости, то и вообще не целесообразно).

Само собой, этот эффект имеет и обратную сторону. Информация, которая не должна попасть после ее удаления пользователем в руки третьих лиц, остается доступной для восстановления и использования.

Среди решений данной проблемы различают два основных метода гарантированного уничтожения информации:

1. Физическое уничтожение носителя;
2. Гарантированное уничтожение с сохранением работоспособности носителя и возможностью дальнейшей его эксплуатации.

Первый метод не нуждается в комментариях, носитель просто механически или химически разрушают (например, разбивают диски НЖМД молотком, используют мощный электромагнитный импульс и т. п.).

Рассмотрим подробнее метод №2. При больших количествах носителей, хранящих информацию, подлежащую уничтожению, их разрушение становится экономически накладным (пусть даже оправданным). Стоимость НЖМД позволяет уничтожать их большими объемами разве что некоторым очень хорошо спонсируемым государственным ведомствам, например, военным или правоохранительным, но даже военные ведомства далеко не в каждой стране могут позволить себе подобное расточительство.

Гарантированно уничтожить информацию, не хуже чем при физическом уничтожении, позволяет ее перезапись. На перезаписи базируются все существующие способы уничтожения информации с сохранением работоспособности носителя.

Но даже перезапись иногда может не спасти от полного уничтожения данных. К примеру, если данные были записаны в секторах, которые в последствии были помечены как сбойные, не являясь таковыми по сути, то контроллер НЖМД не допустит его перезаписи, т. к. уже считает сектор сбойным и не использует в работе.

Иным способом восстановления, лишь частично предотвращаемым с помощью перезаписи, является считывание данных между дорожками разметки НЖМД. Запись в этих пространствах появляется за счет рассеивания магнитного поля в зазоре между записывающей головкой и поверхностью диска. Когда ширина поля рассеивания становится больше ширины дорожки разметки, появляется теоретическая возможность считывания сигналов, сохранившихся на поверхности диска между дорожками.

Если же восстановление информации из перемещенных (bad) секторов может дать полезный эффект, в случае, когда этих секторов много и они подлежат считыванию, то восстановление из междорожечного пространства, учитывая плотность размещения дорожек в современных НЖМД, носит более теоретический характер.

Другой фактор, который является больше теорией — подход с использованием остаточной намагниченности магнитных доменов. Суть этого метода заключается в том, что уровень намагниченности не всегда соответствует уровню нуля или уровню единицы. Грубо говоря, если в магнитном домене содержался уровень 0 и был перезаписан уровнем единицы, то по факту уровень намагниченности будет равен, к примеру, 0,75. Или же наоборот, после содержания единицы, перезаписанной нулем, в домене уровень намагниченности будет ближе к 0,25. Контроллером НЖМД такие уровни будут округляться и декодироваться как 0 и 1, но если считать эти уровни в обход контроллера, например, с помощью магнитного силового микроскопа, то можно интерпретировать их по-своему: 0,25 воспринимать как 1, а 0,75 как 0. Теоретически, в результате должна получиться информация, содержащаяся на данном участке накопителя до его перезаписи. Также, теоретически, есть вероятность восстановления сигнала вплоть до нескольких перезаписей, к примеру, если сигналы имели различные характеристики частоты магнитного поля.

На данный момент такой подход имеет достаточное количество как теоретических, так и практических проблем, и остается теорией, имеющей право на жизнь. Реально работающие устройства либо компании, реально гарантирующие восстановление данных после перезаписи, пока неизвестны. Сам же Питер Гутман, разработчик одного из наиболее эффективных методов гарантированного уничтожения данных с помощью перезаписи, утверждает, что у спецслужб такие устройства есть. Спецслужбы в свою очередь не дают подтверждения, но и, как правило, их процедуры безопасности данных считают перезаписанный диск ненадежным. К примеру, в США жесткие диски, на которых хранилась информация с грифом «СОВЕРШЕННО СЕКРЕТНО», подлежат только размагничиванию или физическому уничтожению (что по сути почти одно и тоже).

Питер Гутман предложил метод, охватывающий все возможные НЖМД, а также предусматривающий эффективное уничтожение данных с НЖМД, использующих MFM и RLL кодирование, для которых были предназначены 27 проходов. В силу тех факторов, что пользователь, как правило, не знает, какое магнитное кодирование используется в утилизируемом НЖМД, метод Гутмана предусматривает в сумме 35 проходов перезаписи. Каждый проход записывает различные шаблоны данных в каждый байт каждого сектора, 8 из которых используют в качестве шаблонов случайные последовательности. При точном знании метода магнитного кодирования НЖМД, количество проходов перезаписи можно сократить без потери эффективности уничтожения данных. Также следует учитывать, что метод был предложен в 1996 году, и некоторые методы кодирования уже являются устаревшими, что также позволяет сократить количество проходов без потери качества уничтожения данных.

Существует несколько свободных проектов, реализующих метод Гутмана. Наиболее популярна, по всей видимости, утилита shred, входящая в состав GNU Core Utilities.

Для унификации и оптимизации методов уничтожения информации в различных странах были выпущены различные стандарты, которыми все на сегодняшний день и пользуются.

Министерством Обороны США был выпущен стандарт DoD 5220.22-M. Стандарт имеет различные модификации для различных областей применения. Количество проходов различными шаб-

лонами данных, включая случайные шаблоны, колеблется от 2 до 7.

В Российской Федерации разработан стандарт ГОСТ Р50739-95. Данный государственный стандарт предоставляет свободу в выборе шаблона перезаписи и предусматривает, в зависимости от класса автоматизированной системы, от 1 до 2 циклов перезаписи.

В Германии разработан стандарт VSITR. Стандарт предусматривает 7 циклов перезаписи, но не предусматривает случайных последовательностей в шаблонах перезаписи и использует всего три заранее заданных шаблона.

К сожалению, белорусских нормативных актов, регулирующих процессы уничтожения цифровой информации, найти не удалось, а украинские не описывают методы, которые должны применяться, и лишь указывают на необходимость применения гарантированного уничтожения информации с цифровых носителей.

Как правило, на практике зачастую достаточно одного-трех проходов для вполне успешного уничтожения данных и дальнейшего использования НЖМД без практической возможности восстановления информации, содержащейся на нем ранее.

FlowForwarding Warp: how is JVM running the SDN controller

Dmitry Orekhov, Minsk, Belarus*

How to build a fast, scalable and portable SDN controller? Is JVM an appropriate platform for this? What solutions may Java world suggest for distribute systems and data serialization? And how fast would it be, eventually? These questions make a subject of the presentation.

Introduction: be ready for the Real World

Today Software-defining is a real factor in the Industry. Network Function Virtualization, Service insertion in datacenters and clouds; Dynamic WAN rerouting and interconnecting, Bandwidth on Demand for providers — that's only a short list of SDN use cases. The most well-known usage of SDN is OpenStack — Open Source cloud computing platform, created and supported by free developers in tight collaboration with enterprise vendors.

At considering SDN as an Enterprise technology, new non-functional requirements become actual: stable work under high-load (hundreds and thousands of controllers and switches) and scalability.

For Open Source developers we may add one more requirement, a portability. Enterprise vendors can tune software carefully against certain hardware, but Open Source developers cannot afford this to themselves.

Instead of this, a strategy would be to provide open and portable solutions, so everybody may use them on favorite platforms, improve and customize. Probably JVM is currently one of the best platform for such solutions. Additionally, most interesting Open Source initiatives in SDN OpenDaylight and ONOS are written in Java. Also one can take Hadoop as an example: we have made some experiments using OpenFlow controller Java library to make Hadoop topology more adaptive. So our decision is JVM.

*Dmitry_Orekhov@epam.com, http://lvee.org/ru/abstracts/135

Apache Avro: Fast run-time serialization framework in Java

In the real world, a very desirable feature for SDN (and, particularly, OpenFlow) Controller would be an ability to update itself with new versions on the fly. It dictates us, at first, to separate protocol definition from other code and, secondly, to provide dynamic load of protocol in run-time, as it may be critical for topology to update SDN controller on the fly, without stopping. To fulfill these requirements we chose Apache Avro.

Avro is a data serialization and remote procedure call framework. For us, the most important Avro distinction from other similar solutions like Protobuff or Thrift, is that Avro doesn't demand code generation and may parse protocol and apply any protocol changes in run-time.

When you use Avro, the workflow is:

1. Define protocol in JSON-based Avro language,
2. If you don't need run-time protocol updating, you may compile your protocol, get bunch of classes and get all advantages of static typing
3. If real-time protocol update in Avro is quick enough, then there should be wide use caching and pools of pre-built objects.

Akka

Akka library was developed to simplify development of distributed and concurrent software on JVM. It was inspired with Erlang and implements high-performing Actors model. Millions messages per second, very small footprint and distributability by design make Akka very good for distributed software on JVM.

We have chosen Akka because Actors model fits ideally into the SDN controller architecture. SDN Controller must run multiple independent and stateless sessions, one per connected Switch. No session can harm the other one. Further, according to SDN Controller ideology, it is stateless, and therefore shouldn't store any information about the Switch state. So, we don't need any failover. Eventually, if any Controller session want to crash — we should let it crashing. And it matches ideally Actors model implemented by Akka (and by Erlang before).

Scala

Akka is in Scala, so it brings support of functional programming. Also, Scala is scalable by design. One Scala feature we don't use now is very powerful parsing facility, which we use in our project, dedicated to Domain-specific language for 'binary' protocols, like OpenFlow.

Assemble everything together

Using Akka we have built a pool of Actors, communicating via sending messages. Every Switch is handled by separate Switch Connector. Switch Connectors define only basic functionality, and developer may customize behavior, implementing Event handlers and registering his Actor for specific Event. Currently, we have some custom actors implementing REST API for Warp controller.

Protocol drivers are totally separated from Controller part. Via simple API every Controller actor may build, customize and serialize messages. Moreover, it's possible to use Protocol drivers with any other JVM applications. We made proof of a concept, using Warp OpenFlow driver in Floodlight controller, by the way. Currently, only OpenFlow protocol is implemented as the most widely used protocol in SDN.

High-load testing

Well, we've just started yet. We have performed quick and initial testing of Warp controller running on on 4-core CPU against 600 LINC logical switches running on two 4-core CPUs. We were satisfied, having:

1. About thousand heartbeats from different switches per minute
2. Session establishing (TCP connection, handshake) in 25 seconds
3. Using script we're able to establish and we have about 60-70% of CPU utilization during these tests for all cores

References

1. `http://www.sdncentral.com/sdn-use-cases`
2. `http://flowforwarding.org`
3. `https://github.com/FlowForwarding/warp`
4. `https://github.com/FlowForwarding/LINC-Switch`
5. `http://akka.io`
6. `http://avro.apache.org`

Лицензионный иммунитет СПО. Освобождение проекта на примере Kallithea

Андрей Шадура, Братислава, Словакия*

Free software business models are difficult to design and not as lucrative as proprietary ones. Sometimes, companies make a difficult choice to proprietarise the work of a Free Software community in an effort to gain more revenue. When proprietary models do often produce more revenue, they bear a cost on the community. Fortunately enough, the very idea behind free software, coupled with specifics of some of the free licenses, makes it possible to liberate the project when it's at danger of becoming non-free.

Зачастую авторы успешных свободных программных проектов задумываются над тем, как превратить хобби в способ зарабатывания денег на жизнь. Самым действенным способом зарабатывания денег на свободном коде, оставляя его свободным, является платная поддержка его или продуктов на его основе, либо продажа сервисов, в которых этот продукт используется. Помимо крупных компаний, таких как RedHat и Ubuntu, подобным образом зарабатывают авторы меньших проектов. Одним из известных примеров таких продуктов является веб-сервер nginx: основанная автором компания осуществляет платную поддержку и обновления безопасности, а также предоставляет услуги по настройке специальных конфигураций и внедрению дополнительных возможностей. Менее известный белорусский проект — Ajenti, инструмент веб-администрирования наподобие Webmin, но основанный на современных технологиях. Проект распространяется под лицензией AGPLv3, но предлагает и схему коммерческого лицензирования для компаний, предоставляющих услуги хостинга, или производителей серверного оборудования.

Тем не менее, некоторые разработчики решают перейти на «тёмную сторону силы», уходя от модели open source на модель, базирующуюся исключительно на продаже лицензий по количеству пользователей. Рассмотрим один из таких случаев подробнее на примере

*andrew@shadura.me, http://lvee.org/ru/abstracts/145

недавней борьбы за освобождение кода одного некогда свободного проекта.

В феврале 2010 года польский программист Марцин Кузьминьски начал проект под названием RhodeCode, цель которого была в создании платформы для совместной работы над кодом с использованием системы контроля версий Mercurial, подобной GitHub. На тот момент для Mercurial не было сравнимой по возможностям системы, кроме «облачного» и закрытого Bitbucket. Код RhodeCode, с другой стороны, распространялся на условиях GPLv3, и его можно было поставить на свой собственный сервер, в отличие от «облачных» решений. Проект начал быстро развиваться, к разработке подключилось много разработчиков, в том числе, например, из Unity Technologies. Появилась поддержка pull requests, комментирования коммитов, поддержка Git, авторизации через LDAP и другие возможности.

В июле 2013 года после продолжительной закрытой разработки компания, которую основал Марцин, RhodeCode GmbH, выпустила новую версию, 2.0, на новых условиях — Business Source License. Первоначально лицензия покрывала весь код, требуя покупки лицензий для коммерческих пользователей, но при этом обещалось, что лицензия автоматически превращается в GPLv3. Помимо того, что такая лицензия делает код несвободным (хоть и доступным), подобная смена лицензии без согласия всех, кто вносил свой вклад в кодовую базу, попросту нелегальна. После существенного недовольства пользователей решение было изменено: отныне код на Python и HTML снова под лицензией GPLv3, а всё остальное — под Business Source License.

Что же такое опенсорс? Небольшое отступление насчёт критериев «опенсорсности» лицензий. Первоначально, Ричард Столлман определил свободу ПО как «право пользователя свободно запускать, копировать, распространять, изучать, изменять и улучшать его». В июле 1997 проект Debian опубликовал Debian Free Software Guidelines (DFSG) — набор критериев для определения свободного ПО:

- Свободное распространение: лицензия не ограничивает распространение каким бы то ни было лицам или организациям, не требует денежной компенсации
- Исходные тексты: они должны присутствовать, и лицензия не должна ограничивать их распространение

– Производные работы: лицензия должна разрешать создание и распространение производных работ от данного ПО на тех же условиях, как и оригинал
– Целостность авторских исходных текстов: лицензия может запрещать распространение производных работ от исходных текстов, но в этом случае она должна разрешать свободное распространение патчей для исходного текста
– Запрещается дискриминация людей или групп людей
– Запрещается дискриминации по областям деятельности
– Распространение лицензии: лицензия распространяется на любого, кто получил копию ПО
– Лицензия не должна относиться исключительно к Debian
– Лицензия не должна ограничивать другое ПО
– Примеры лицензий: GNU GPL, BSD, Artistic License — свободные лицензии

Эти самые критерии легли в основу определения open source. Вопреки распространённому мнению, любое программное обеспечение, которое является open source, всегда является и free software. Иначе говоря, разница в определениях free software и open source исключительно идеологическая: FSF и Столлман исходят из этической стороны вопроса, а OSI и Debian — из практической. Кроме того, Столлман не распространяет своё видение этих свобод в полной мере на документацию: лицензия GNU FDL с неизменяемыми секциями признаётся Debian несвободной, а именно под такой лицензией распространяется документация к многим проектам GNU.

Проблемы с лицензией RhodeCode. Легко видеть, что ограничения на количество пользователей, род деятельности (коммерческий, военный и т.д.) переводят лицензию в род несвободных. Именно такой является новая лицензия Business Source RhodeCode.

Следующая проблема с лицензией RhodeCode — раздельное применение GPLv3 и Business Source к разным частям проекта. Как известно, GPL требует, чтобы производные работы распространялись также под GPL, так что по сути GPLv3 применима ко всему проекту, но на часть файлов накладываются ограничения в соответствии с Business Source. А здесь уже вступает в силу раздел §7 GPLv3, в котором говорится, что дополнительные условия могут только давать права пользователям, но не могут их отбирать. Подобные дополнительные ограничения, утверждает этот пункт, не имеют силы, могут игнорироваться и могут быть вовсе устранены.

По мнению Брэдли Куна, президента фонда Software Freedom Conservancy, подобная схема лицензирования, возможно, представляет собой нарушение GPL, и вносит неясность насчёт того, какими же правами обладает пользователь (leaves the redistributor feeling unclear about their rights).

К сожалению, проблемы с RhodeCode неясностью лицензии не ограничились. Американский разработчик, Тревис Бертрам, опубликовал на GitHub четырёхстрочный патч, устраняющий ограничение по количеству пользователей в RhodeCode... после чего с ним связался CEO упомянутого GmbH с угрозами судебного преследования за нарушение условий использования торговой марки. Подобные угрозы получил Джеймс Родс, который месяцами ранее поднял вопрос о легальности смены лицензии. Несмотря на это, RhodeCode продолжал рекламироваться как opensource-решение, таковым по сути являясь весьма условно.

Форк. Как легко догадаться, такая ситуация расстроила немалое количество людей, потому в январе 2014 началась работа по подготовке форка RhodeCode. Очевидным и самым простым способом было бы взять за основу код последней полностью свободной версии, 1.7.2. Таким путём удалось бы избежать борьбы с неясностями лицензии, но кодовая база проекта была бы устаревшей как минимум на год, и в ней отсутствовала бы некоторая весьма востребованная функциональность. Кроме того, версия 2.0 состояла из объединённых двух веток: развивавшейся в закрытых условиях ветки с графическим интерфейсом, и открытой до определённого момента ветки с бэкендом, которая не писалась единолично автором. Весь этот код было бы очень неприятно потерять, поэтому было решено рискнуть и взять максимум того, что получится, из самого свежего кода, выпущенного GmbH.

Форк производился под эгидой вышеупомянутой организации Software Freedom Conservancy, что позволило гарантировать дальнейший свободный характер разработки и снять с разработчиков решение юридических и финансовых вопросов. Группа энтузиастов с помощью юридического отдела SFC провела ревизию всех опубликованных изменений в RhodeCode начиная с последней полностью свободной версии и заканчивая наиболее актуальной на данный момент версией 2.2.5. Был тщательно рассмотрен лицензионный статус каждого из них, и те, которые были признаны полностью свободными, были перенесены в новый проект. Кроме того,

был наведён порядок с лицензиями на JavaScript-библиотеки, использованные в проекте (jQuery, Bootstrap, YUI и другие).

Таким образом, Kallithea получила большую часть кода RhodeCode на Python, но оказалась лишена нового графического интерфейса, который в версии 2.0 был переписан практически с нуля. Адаптация старого интерфейса на новую кодовую базу, ребрендинг, включение в поставку полных исходных кодов JavaScript-зависимостей — всё это потребовало ещё немало времени, но в конце концов проект был выпущен в свет 4 июля 2014 года, в день независимости США. Код Kallithea гарантированно 100% GPLv3, и таковым и останется: поддержание данного статуса является одной из задач Software Freedom Conservancy. Поддержку проекту выразил и автор и главный разработчик Mercurial — Matt Mackall.

Надо сказать, несмотря лицензионную ситуацию, которая привела к форку, Conservancy и группа разработчиков Kallithea приглашают исходных авторов проекта к сотрудничеству, например, по устранению неоднозначностей в лицензии. Тем не менее, реакции от них пока что не последовало вообще никакой.

Как видно, «закрыть» открытый проект — довольно непростое дело в силу специфики свободных лицензий, предоставляющих определённый иммунитет к такого рода атакам. С учётом эффекта Стрейзанд — уже опубликованные версии общедоступны, а сменить лицензию постфактум нельзя — такой разработчик не имеет гарантий успешности своей затеи.

Литература

[1] Why Conservancy's Kallithea Project Exists, by Bradley M. Kuhn, Software Freedom Conservancy. `http://sfconservancy.org/blog/2014/jul/15/why-kallithea/`

Token-based авторизация для сессий прямого соединения в облачной системе

Андрей Романюк, Минск, Беларусь*

Performance and security is the cornerstone of the Cloud based applications. Under this thesis we are going to discuss solution which was proposed by ITS Partner developers team to optimize and improve access to the Cloud data without loosing security level.

Введение

Представленный материал обобщает опыт, полученный командой разработчиков ITS Partner при создании облачного приложения для доступа к хранящемся в облаке данным.

Данное приложение позволяет пользователям получать доступ к данным своих устройств NAS (Network Attached Storage) независимо от их физического местоположения. Облачный сервис, таким образом, имеет свою специфику: каждый пользователь облака непосредственно владеет и управляет собственным хранилищем данных. Кроме того, сервис позволяет обмениваться данными с другими пользователями, за счет предоставления доступа к отдельным ресурсам на своих NAS.

В обмене данными участвуют клиентское приложение, NAS-устройство, а также сервер, используемый в качестве промежуточного звена коммуникации. Обмен между клиентом, сервером и NAS-устройством происходит по протоколу HTTPS и HTTP, причем клиент может использовать для доступа к устройству и управления им как веб-приложение в браузере, так и его десктопную либо мобильную версию.

Среди задач, решавшихся в процессе разработки, наибольший интерес по мнению авторов представляют две связанные между собой задачи: оптимизация сетевого доступа клиента к NAS при появлении возможности их прямого соединения, а также обеспечение безопасности обмена данными. Ниже рассмотрена проблематика и особенности решений, примененных в обоих случаях.

*andreyrom@tut.by, http://lvee.org/en/abstracts/132

Оптимизация доступа

В ходе разработки облачного приложения возникла необходимость решать проблему, связанную с оптимизацией доступа к данным и повышения производительности сервиса.

В качестве одного из способов оптимизации был выбран вариант установки прямого соединения (локальной сессии) между пользователем и его NAS-устройством в случае нахождения их в одной сети. Такое прямое соединение временно исключает сервер из обмена данными между конкретной парой «клиентское приложение — устройство NAS», что позволяет не только увеличить скорость передачи данных с переходом на локальный трафик, но также способствует снижению нагрузки на сервер.

При этом для пользователя переход на работу в локальной сессии должен выполняться автоматически и незаметно, без дополнительных сообщений, диалогов авторизации и прочих атрибутов.

Для определения возможности соединения между клиентом и NAS в локальной сети используется следующая стратегия: клиент запрашивает через сервер локальный IP-адрес NAS-устройства и его идентификатор. После этого отправляется запрос напрямую на устройство по полученному внутреннему IP-адресу. Очевидно, что если клиент и устройство находятся в одной сети, то запрос от клиента будет обработан успешно. NAS возвращает клиенту свой идентификатор (серийный номер), по которому определяется, что это именно то устройство, которое требуется пользователю. Если клиент не получил ответа по локальному адресу, или ответ был неверный, то предполагается, что прямого соединения с устройством хранения данных нет и поэтому коммуникация продолжается через сервер.

Как только браузерное приложение определяет, что есть возможность работать с устройством хранения данных напрямую через локальную сеть, оно инициирует создание локальной сессии, отправляя запрос на сервер.

Обеспечение безопасности

Данный подход, при всей универсальности, имеет риски, связанные с безопасностью локальной сессии. При работе через облачный сервер авторизация и аутентификация пользователей реализуются непосредственно на сервере; однако в случае локальной сессии

устройству хранения данных, равно как и клиентскому приложению, необходимо самостоятельно выполнять часть функций, связанных с контролем доступа пользователей.

Для обеспечения высокого уровня безопасности был выбран подход с использованием одноразовых паролей — токенов. При инициализации локальной сессии клиент и его NAS-устройство получают от сервера секретный ключ. Каждый прямой запрос между пользователем и его устройством подписывается одноразовым токеном, генерируемым на базе этого ключа с добавлением соли — текущего времени в миллисекундах на машине клиента.

Токен представляет собой закодированный в шестнадцатеричном формате MD5-хэш секретного ключа и значение текущего времени в миллисекундах в формате Linux time. Он вставляется в URI следующим образом:

```
uri-prefix/token/timestamp-in-hex/rel-path
```

например:

```
/fsbroker/dee0ed6174a894113d5e8f6c98f0e92b/43eaf9c5/file_t
o_protect.txt
```

Проверка токена на NAS-устройстве происходит с помощью формирования собственного токена на базе секретного ключа и timestamp, полученного от клиента. В случае совпадения полученного и сгенерированного хешей запрос считается авторизованным и передается на выполнение устройству.

Особенности реализации

Благодаря использованию GNU/Linux в качестве программной платформы NAS-устройства оказалось возможным делегировать механизм авторизации веб-серверу Apache, работающему непосредственно на NAS. При этом проверку токенов осуществляет модуль mod_auth_token, специально доработанный для целей интеграции с разрабатываемым решением. Для интеграции модуля авторизации в сервер apache необходимо выполнить:

```
cd ~
wget "http://mod-auth-token.googlecode.com/files/mod_auth\
_token-1.0.6-beta.tar.gz"
```

```
tar xvzf mod_auth_token-1.0.6-beta.tar.gz
cd mod_auth_token-1.0.6-beta /
sudo ./configure
sudo make
sudo make check
sudo make install
sudo service apache2 restart
```

Также для использования модуля авторизации mod_auth_token в конфигурации веб-ресурса под управлением apache прописываются необходимые настройки (обычно эта конфигурация находится в файлах /etc/apache2/sites-enabled/*). Token-авторизация применяется только для директивы Location:

```
<Location /downloads/>
      AuthTokenSecret       "secret string"
      AuthTokenPrefix       /downloads/
      AuthTokenTimeout      60
      AuthTokenLimitByIp    off
</Location>
```

Доработка модуля авторизации

В процессе исследования модуля mod_auth_token было обнаружено, что он позволяет использовать только статический секретный ключ, прописанный в конфигурации:

```
AuthTokenSecret ‘‘secret string’’
```

В то же время для усиления безопасности при коммуникации в локальных сессиях было принято решение использовать динамические секретные ключи. Такой секретный ключ NAS-устройство получает с сервера в момент инициализации новой локальной сессии и сохраняет в конфигурационном файле.

Была выполена доработка модуля для возможности использования помимо статически-заданной конфигурации также и динамических секретных ключей. В итоге модуль авторизации считывает динамический секретный ключ в режиме реального времени из файла конфигурации и использует его для проверки входящих запросов.

В настоящий момент авторы пробуют осуществить включение своих доработок в основной код проекта mod_auth_token.

Что подтолкнуло белорусских пиратов создать филиал Creative Commons?

Михаил Волчек, Минск, Беларусь*

The initiative of Belarussian priates to open the Creative Commons affiliate in Belarus is presented. Covered topics include the analysis of Belarussian copyright law and its incompartibilities with most of open source licenses, as well as proposed steps to initiate its improvement.

Введение

Часто можно услышать фразу «Dura lex sed lex» — «Закон суров, но закон». Это древнеримское выражение предлагает нам подчиниться принуждению закона, который может быть неоправданно суров. Но те, кто знакомился с трактатом Цицерона «О законах», встречались с другой не менее интересной парадигмой римского права: «Salus populi suprema lex esto». Это значит, «благо народа — высший закон». В своём докладе автор видит необходимость рассказать, почему современный белорусский закон «Об авторском праве и смежных правах» (Закон «Об авторском праве», ЗоАП, копирайт) слишком «суров», почему он не действует на «salus populi», и каковы намерения белорусских пиратов в связи с созданием филиала Творческих Общин (Creative Commons).

Запретительная природа копирайта

Копировать, изменять и распространять контент, программы запрещено, если это явно не разрешено правообладателем. Статья 16 ЗоАП гласит, что *«. . . автору или иному правообладателю принадлежит право разрешать или запрещать другим лицам использовать произведение»*, согласно статье 16 пункту 2 — *«переработку произведения для создания производного произведения»*. Теперь посмотрим, что такое «производное произведение». *«Производное произведение — перевод или иная переработка произведения, являющиеся результатом творческого труда, в том числе обработка,*

*fannrm@gmail.com, http://lvee.org/en/abstracts/137

обзор, пересказ, аннотация, резюме, реферат, инсценировка, музыкальная аранжировка» (ст. 4 ЗоАП) по сути устанавливает запретительные ограничения на множество форм творчества.

Замедление культурного и интеллектуального обмена

Страх автора опубликовать какие-то данные в образовательных целях создаёт атмосферу, когда публичная циркуляция знаний перестаёт быть безопасной. Несколько примеров из белорусской Википедии. Например, автор статьи сомневается можно ли разместить автограф в статье [1]. Или знаменитый пример, когда из материала удалялась фотография Национальной Библиотеки, т.к. «произведения архитектуры, градостроительства и садово-паркового искусства» согласно Статья 6 ЗоАП тоже являются объектами авторского права.

Экономика рантье

Копирайт даёт монополию. Например, в Беларуси срок этой монополии составляет жизнь автора плюс 50 лет. Согласно статье 20 ЗоАП «Исключительное право на произведение действует в течение жизни автора и пятидесяти лет после его смерти. . . »

На протяжении этого времени правообладатель не производит ничего нового, просто собирает прибыль из когда-то созданного произведения. Живёт на проценты. Такой подход стимулирует создание перекупщиков «авторских прав», другими словами посредников.

В мире широко известны монополистическим беспределом организации по коллективному управлению авторскими правами (ОКУП), которые пропускают через себя гигантские суммы денег, а также являются лоббистами крупных корпораций-держателей целых портфелей «авторских прав» — такие, как IFPI, RIAA, MPAA, BSA. В Беларуси функциями ОКУП занимается Национальный Центр Интеллектуальной Собственности, которому в принципе не чужды многие проблемы подобных организаций (известен пример 2013 года с ресурсом webmusic.by).

Массивный и непонятный большинству создателей контента

Сферу авторского права регулируют различные документы. Статья 3 ЗоАП отсылает читателя (это может быть, например, художник, или дизайнер) прочитать ещё пару (а их десятки) международных договоров: «если международным договором Республики Беларусь установлены иные правила, чем те, которые содержатся в настоящем Законе, то применяются правила международного договора». Расплывчатые и оценочные формулировки (например, «объективная форма» (ст.1), «оправданный объёмом, целью цитирования, информационной целью» (ст. 32, 33)) ЗоАП запутывают не только обывателя.

Устаревший и вредный

На сегодня ЗоАП не учитывает свойство Интернет, его динамичность, возможности быстрого обмена контентом и выстраивает различные преграды. Чего, например, стоит ограничение белорусского законодательства заключать лицензионные договоры в отличной от письменной формы виде. За бортом закона остаётся набор таких публичных лицензий как Creative Commons (для контента), GNU GPL, BSD, Apache, Mozilla, MIT, CPAL (для софта).

Последняя капля

«Копирайт» — один из самых неэффективных законов. По оценкам Microsoft, 87% белорусов являлись пиратами в 2013 году [2]. 9 из 10 нарушают этот закон. Какие могут быть причины? Закон не соответствует интересам общества? Незнание закона? Финансовая невозможность приобретать дорогие лицензии? Возможность получать гораздо более дешёвые «контрафактные» программы? Неуважение к автору или правообладателю? Разобраться в этом призван филиал Творческих Общин.

Филиал Творческих Общин

В конце 2013 года активистами молодёжной организации Фаланстер началась подготовка к официальному открытию филиала (мероприятие пройдёт 29 августа 2014). На сегодня составлена до-

рожная карта [3], а также запущен проект на площадке Талака [4]. Можно выделить основные три направления на этот период.

Во-первых, это комплексное исследование белорусского закона «Об авторском праве» на предмет возможности имплементации публичного лицензирования для обычных и сетевых произведений.

Во-вторых, это опубликование и открытое обсуждение данных исследования для всех заинтересованных сообществ.

И в-третьих, это создание обновленного закона об авторском праве методом краудсорсинга, а также сбор 1000 подписей в поддержку этих правок через инструменты электронного участия — петицию.

Помимо этого будут проходить лекции, дискуссии по проблемам копирайта, консультации и информационная поддержка проектов использующих свободные лицензии.

Эти шаги с одной стороны наполняют созидательную деятельность пиратов, а с другой — делают возможным через присутствие филиала Creative Commons дополнительно вовлечь белорусское общество в мировой цифровой контекст.

Литература

[1] `http://bit.ly/1sk1bZt`
[2] `http://bit.ly/1sk7uMP`
[3] `http://wiki.creativecommons.org/Belarus`
[4] `http://www.talaka.by/projects/521`

Flume и Morphlines — трансформация потоков данных без строчки кода

Денис Пынькин, Минск, Беларусь*

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. Morphlines is an open source framework that reduces the time and efforts necessary to build and change Hadoop ETL stream processing applications. Combination of these technologies give a powerful tool for data stream transformation on distributed configurations without programming.

Введение

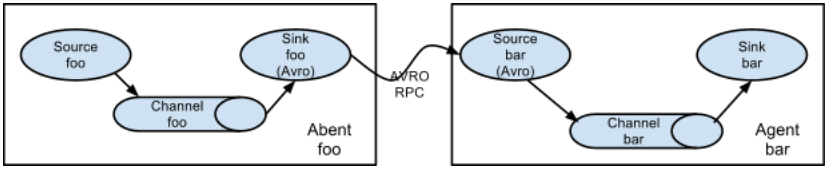
Одним из столпов на которых стоят *nix системы является концепция «конвейеров» (pipeline) для обеспечения совместной работы множества программ. К сожалению возможности конвееров, без использования вспомогательных средств вроде netcat, ограничиваются пределами одного хоста и слабой параллелизацией данных. Даже классические задачи, например, сбор и обработка большого количества логов с фермы серверов становится головной болью администратора.

В то же время адепты Big Data активно развивают концепцию DataFlow, которая по сути является развитием все той же концепции конвейеров, но предназначена для организации управления большими потоками данных.

Apache Flume

Flume [1] — это рапределенный, надежный сервис для эффективного сбора, агрегирования и передачи больших потоков данных. Простой и гибкий в настройке, Flume обладает очень архитектурой, позволяющей настроить буквально любой аспект поведения сервиса. Используется простая и легко расширяемая модель данных, что очень удобно для использования в аналитических сервисах.

*fannrm@gmail.com, http://lvee.org/en/abstracts/133



Каждый агент Flume состоит из 3 составляющих:

– source — источник данных, фактически сервис умеющий преобразовывать входные данные во внутреннее представление;
– channel — очередь сообщений, служащий связующим звеном между источником данных и выходом;
– sink — сервис умеющий преобразовать внутреннее представление данных в необходимое для потребителя.

Особую пикантность придает то, что помимо множества уже существующих «стандартных» компонентов, существует возможность легкого создания и использования любой из составляющих агента.

Каждая составляющая агента может состоять из одного или более компонентов, что превращает Flume в конструктор с возможностью построения любой топологии по передаче потоков данных.

Базовые блоки «конструктора»

На данный момент существует несколько десятков стандартных компонентов и с каждым релизом их становится все больше:

– Source — начиная от запуска любой утилиты, дающей вывод через stdout и заканчивая организацией полноценного сетевого сервиса, работающего с различными сетевыми протоколами: tcp, http, syslog, AVRO, Thrift и др.;
– Memory Channel — как in-memory, для скорости, так и надежная очередь с записью промежуточных данных в файл или даже базу данных;
– Sink — от банальной записи в файл (организация логов) на локальной или кластерной файловой системе, до интеграции с другими агентами Flume или BigData системами.

Синтаксис файла конфигурации, описывающего агента очень прост и приспособлен для использования человеком:

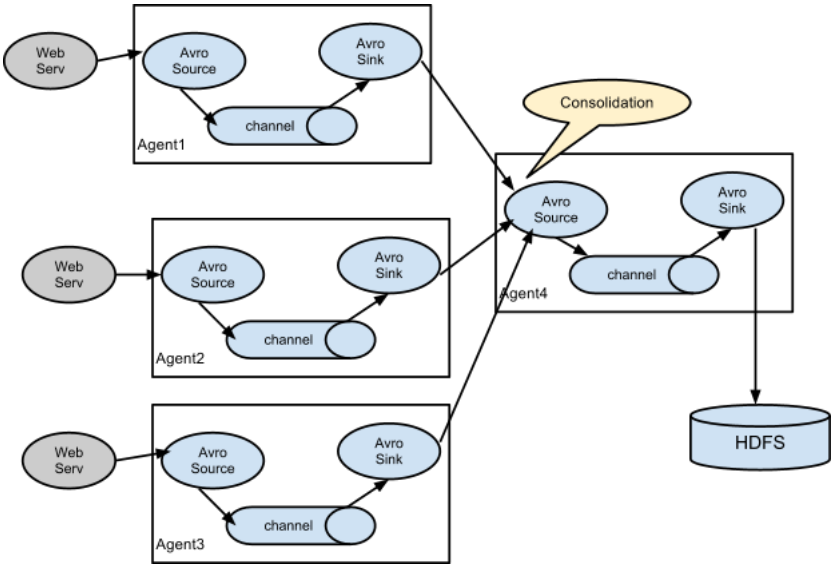
```
# properties for sources
```

```
<Agent>.sources.<Source>.<someProperty> = <someValue>
# properties for channels
<Agent>.channel.<Channel>.<someProperty> = <someValue>
# properties for sinks
<Agent>.sources.<Sink>.<someProperty> = <someValue>
```

Топология

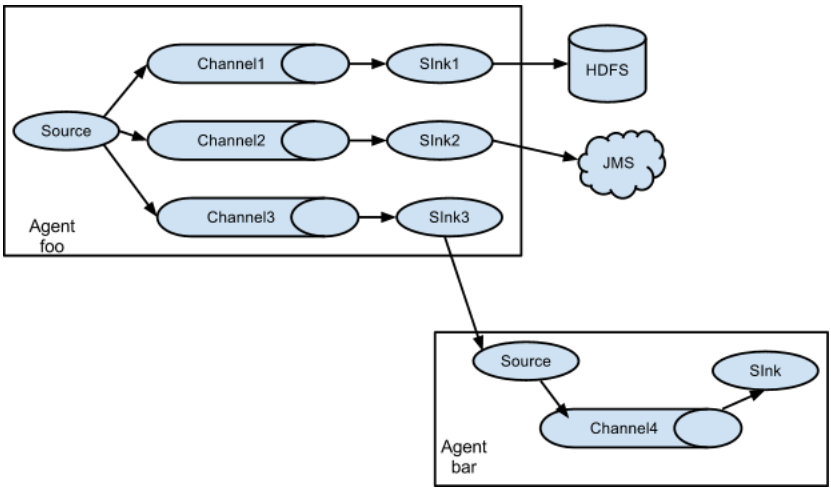
Возможность агентов Flume взаимодействовать между собой позволяет создать действительно распределенную и, при необходимости, надежную сеть передачи потоков данных.

Так, например, с помощью Avro Sink и Avro Source очень легко реализовать агрегирование из нескольких источников в одну систему (топология «fan-in»):



Завязав несколько Sink на одну очередь можно легко добиться дробления одного потока данных на несколько более мелких (топология «fan-out»).

А при использовании нескольких очередей можно добиться мультиплексирования (роутинга) отдельных записей «по условию», как это сделано в системах очередей сообщений:



Чудесное превращение Flume в ETL

ETL [2] (от англ. Extract, Transform, Load — дословно «извлечение, преобразование, загрузка») — один из основных процессов в управлении хранилищами данных, который включает в себя:

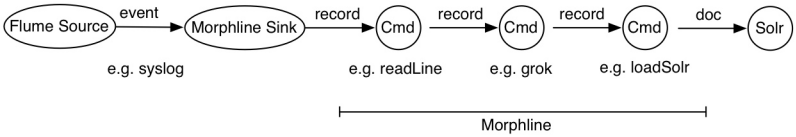
- извлечение данных из внешних источников;
- их трансформация и очистка, чтобы они соответствовали нуждам бизнес-модели;
- и загрузка их в хранилище данных.

Если с Extract и Load у Flume все очевидно, то для Transform у Flume существует замечательный и мощнейщий механизм под названием «Interceptor», который позволяет производить преобразование данных в пределах каждой записи. Как и для других элементов, для «Interceptor» существует несколько базовых механизмов, среди которых особняком стоит механизм Morphlines, позволяющий преобразовывать данные без строчки кода!

Morphlines

Morphlines [3] — это фреймфорк с открытым исходным кодом, разработанный как часть системы Cloudera Search. Основная цель создания Morphlines — быстрая разработка приложений для обработки потоков данных в Hadoop с последующей записью результатов в Apache Solr, HBase, HDFS и прочие системы.

В лучших традициях «правила разделения» приведенным Эриком Реймондом в его бессмертной книге «Искусство программирования в UNIX», в Morphlines была заложена концепция управления с помощью данных — фактически конфигурационного файла, описывающего pipeline последовательных преобразований для записи данных:



Фреймворк получился настолько удачным, что он был выделен в отдельную часть, а также предусмотрены механизмы для встраивания в любое приложение написанное на Java.

Модель данных

Концепция Morphlines предполагает, что данные поступают в виде бесконечного (или хотя бы достаточно большого) потока записей.

Центральной частью Morphlines является запись (Record), которая представляет собой набор именованных полей, причем каждое поле может содержать от одного и более значений. Поля записи могут использоваться как key-value пары, так и выстраиваться в более сложные связи. Для упрощения можно рассматривать работу с записью, как работу с JSON-объектом, поскольку поддерживаются все возможности по организации данных, которые предоставляются этой нотацией.

На самом деле все еще более интересно, поскольку в качестве значения для поля можно использовать любой объект Java.

Конфигурационный файл

Конфигурационный файл представляет собой последовательный список команд для преобразования данных, описанный в формате HOCON (Human Optimized Config Object Notation), кроме того предоставляются возможности бранчевания с помощью команд pipe, if и tryRules.

Список возможных команд преобразования чрезвычайно обширен [4], но не ограничивается только предопределенными командами — достаточно легко можно написать свою реализацию недостающего функционала. Более того, с помощью команды «java» можно встраивать исходный код прямо в конфигурационный файл!

Чрезвычайно мощным инструментом является команда «grok» реализующая разбор любой текстовой строки с помощью регулярных выражений с использованием одноименного синтаксиса из пакета LogStash [5]. Особенно приятно то, что для отладки можно использовать онлайн дебаггер «Grok Debugger» [6].

В онлайн-версии данных тезисов дополнительно рассмотрен подробный синтетический пример использования предлагаемой связки: схема, собирающая статистику скачивания файлов со множества FTP-серверов и сохраняющая ее в файл в формате JSON.

Краткий вывод

При совместном использовании Flume с поддержкой Morphlines можно организовать распределенную систему обработки потоковых данных почти любой сложности, буквально не написав ни строчки кода, что должно особенно понравится администраторам.

Также можно констатировать, что, не смотря на то, что оба инструмента написаны на Java, их архитектура отлично укладывается в принципы, декларируемые философией Unix, а использование близко к «классической» концепции pipeline.

Литература

[1] `http://flume.apache.org/FlumeUserGuide.html`
[2] `https://ru.wikipedia.org/wiki/ETL`
[3] `http://kitesdk.org/docs/current/kite-morphlines/index.html`
[4] `http://kitesdk.org/docs/current/kite-morphlines/morphlinesReferenceGuide.html`
[5] `http://logstash.net/docs/1.4.2/filters/grok`
[6] `http://grokdebug.herokuapp.com/`

Virtualization-based illustrated reviews of the software history

Dmitriy Kostiuk, Pavel Lutsiuk, Sergey Vlasenko,

Vitaliy Zheludok — Brest, Belarus*

Experience of using virtual machines instead of screenshots in a visual timeline of GUI is reviewed. Availability of materials is considered as far as problems of QEMU-based nested virtualization. A solution is proposed for free distribution of F/LOSS virtualized items with a possibility to automatically integrate proprietary ones in case of their presence.

Although technically working with the new information technologies doesn't demands knowing the history of their development, however specialist who formulates or applies modern theory without knowledge of its history, runs the risk of repeating the mistakes of predecessors personally one by one.

In the context of spoken above statement this project lies, providing the user of a local network or a standalone workstation with the set of chronologically HTML documents, each with a description of the specific graphical operating system and its live illustration in the form of built-in frame with the screen of a running virtual machine (thanks to the performance of today's notebooks and desktop PCs, this task is easy enough). The information content of this project is based on the lecture course on the history of graphical user interface, including 40 desktop and 30 mobile operating systems and desktop environments.

Technical infrastructure implementing such informational materials, is close to one reviewed in [1] and includes following components:

- QEMU virtual machine with hardware virtualization support,
- noVNC VNC client, written in JavaScript and HTML5,
- JavaScript framework to display informational materials in one interactive timeline.

Running the scheme shown on the figure 1 is done by a startup script, which scans subdirectories in search for the information elements: pages with the content, virtual machine images and scripts to run them.

*dmitriykostiuk@gmail.com, http://lvee.org/ru/abstracts/144

Script passes VNC port numbers to discovered virtual machines and reconstructs HTML document to include of pages with information materials into timeline. This component based approach allows to divide the information, making free/libre parts publicly available on the network, and taking away from public access those virtualized systems, which can not be redistributed due to the terms of commercial licenses.

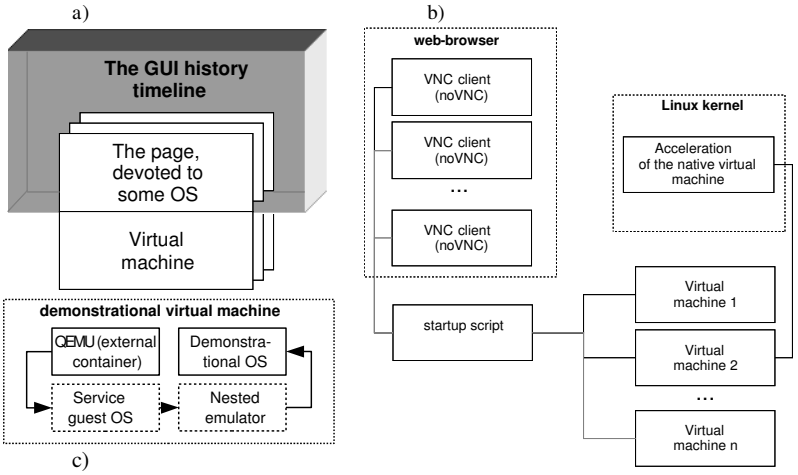


Fig. 22.1. Scheme of the document building (a), and components interaction (b) with nested virtualization (c)

The virtual machine is used as a fully isolated container storing a snapshot of the running OS [2]. Choice of QEMU is caused by the extremely simple transfer of virtual machine images between computers, and also by its ability to emulate not only x86-compatible platforms, but also ones based on SPARC, PowerPC, Motorola 68k, MIPS and ARM processors, which are necessary to run many operating systems from 80s and 90s. However, currently multiplatformness of QEMU is poorly used, because the support of peripheral devices on alternative platforms at this emulator is rarely sufficient to boot ancient operating systems.

At the same time, there are emulators available to support virtually all ancient Intel-incompatible systems — community-developed ones or created as the part of commercial SDK. The first variant is more common in desktop operating systems, so it was possible to enrich the timeline with Xerox Alto, Amiga, RiscOS, Apple Lisa, MacOS of

1.x, 7.x and X versions. Proprietary emulators for desktop operating systems are rare case and usually belong to the manufacturer of the OS, as in the case of Xerox GlobalView emulator. In case of mobile operating systems emulators from the SDK are dominating, such as Psion EPOC16 and EPOC32, PalmOS, Magic Cap, Windows CE, pre-release versions of Android SDK. For now we use only one community-developed mobile OS emulator — Open Einstein project, which allows to run NewtonOS.

However, all these emulators don't support snapshots and the VNC protocol. Therefore a large part of demos is build along with the the nested virtualization scheme (see Fig. 1) where QEMU plays the role of an external container.

Of course many operating systems do not require nested virtualization and so internal emulator is not used for them. That's the case of desktop Windows OS oif 1.x, 2.x, 3.x and 95 versions, as far as IBM OS/2 2.x and 4.x, GEM from Digital Research, GEOS by Berkeley Softworks, as well as a number of mobile operating systems: Pen Windows, Maemo, Android, WebOS (not least due to the fact that QEMU is often included in mobile SDK).

Freely distributable part of the timeline contains free/libre software systems (having such licenses from the very beginning, or opensourced because of the extreme aging or being a free clone of the abandoned commercial system). That's about different Unix-based and Linux-based desktop environments for desktop and mobile computers as well as GEM, Amiga, RiscOS, and HaikuOS.

The timeline material is still at the stage of filling; at this point the review has some missing objects, which have still played an important role in the history of graphical operating systems. Mainly these are new OS versions from Microsoft and Apple. In addition, DOS shell Visi On, and NeXTSTEP are incompatible with the current versions of QEMU. Currently we use Linux-based GNUStep DE as the NeXTSTEP replacement. The problem with the Visi On can be solved using Bochs and VirtualBox, which is undesirable from the portability and resources consumption points of view.

The number of historically important GUI shells without runnable versions appeared to be surprisingly small: currently this row includes multi-window interface of Smalltalk from the late 70s, Xerox Star Document Processor, as well as two mobile systems: PenPoint OS and IBM Simon.

One more component can be seen in the scheme of embedded virtualization — a service guest OS, which is used to start the nested emulator. Its choice is determined by the requirement of the minimum memory consumption, usage of idle CPU cycles, as well as support for USB bus emulation, which allows to emulate pointing devices in absolute coordinates. The latter requirement is important for a comfortable mouse control in a virtual machine [1]. As a service guest operating system in addition to multiple versions of Linux we have used FreeDOS and ReactOS. It should be further noted that ReactOS perfectly meets all three requirements, and thus in our own experience it is the first case of its successful application for some practical needs.

References

[1] Костюк Д.А. Особенности использования виртуализованных окружений, внедренных в презентационные материалы // Восьмая конференция «Свободное программное обеспечение высшей школе»: тез. докл. / Переславль, 26–27 января 2012 года. М.: Альт Линукс, 2012. — С. 83–86.

[2] Костюк Д.А., Дереченник С.С. Построение прозрачных виртуализованных окружений для изоляции уязвимых программных систем // Комплексная защита информации: матер. XVI научно-практич. конф., Гродно, 17–20 мая 2011 г. Гродно, 2011. — С. 209–212.

Удобства и особенности OpenBSD Ports

Вадим Жуков, Москва, РФ*

OpenBSD have acquired Ports framework from FreeBSD many years ago, and since then those frameworks diversed very much. Current OpenBSD Ports infrastructure makes it possible to have many up-to-date ports, including 'heavy' ones like GNOME, LibreOffice and KDE, by relatively few number of active maintainers. This talk is about features of current OpenBSD ports system: bulk package builds, package signing, updating, manual library versions handling, modules framework and other stuff.

Как в любой нормальной современной ОС общего назначения, в OpenBSD есть механизм для установки стороннего ПО. В данном случае — OpenBSD Ports, происходящие, как многие знают, от FreeBSD Ports. Однако, сохранив ряд внешних признаков прародителя, OpenBSD Ports кардинально отличаются от FreeBSD Ports внутри. Проще всего это показать, перечислив то, что сохранилось:

1. Использование Makefile для описания порта.
2. Названия основных операций и ряд переменных, используемых в этом Makefile.

Во всём остальном OpenBSD Ports заметно ушли вперёд — впрочем, разработчики FreeBSD раскачались и стали догонять, что видно на примере свежевышедшей FreeBSD 10.

Ключевые особенности OpenBSD Ports:

1. Сборка пакета — обязательный этап. Пакет сначала собирается, и только потом ПО устанавливается из него в систему.
2. Контроль сборки посредством `systrace`(1), `sudo`(8), fake-фреймворка и `pkg_create`(1). Файлы для пакета помещаются в отдельный каталог, а не прямо на рабочую систему. В ходе «фальшивой установки» отлавливается запись за пределы рабочего каталога (кроме `/tmp` и `/var/tmp`).
3. Поддержание целостности дерева портов посредством непрерывной сборки пакетов с использованием `dpb(1)`.
4. Четыре вида зависимостей: `BUILD_DEPENDS`, `LIB_DEPENDS`, `RUN_DEPENDS`, `TEST_DEPENDS`.

*`zhuk@openbsd.org`, `http://lvee.org/en/abstracts/103`

5. Опции и субпакеты, см. FLAVORS AND MULTI_PACKAGES в `bsd.port.mk`(5).
6. Модули, см. `port-modules`(7).
7. Реально работающий механизм обновления пакетов, с учётом переименований, требований к конкретным версиям, наличия конфликтов и т. д.
8. Ручной контроль версий «публичных» разделяемых библиотек.

Благодаря имеющимся средствам автоматизации большинства рутинных операций и жёстком контролю в виде `dpb`(1), на поддержание высокого качества пакетов уходит сравнительно небольшое количество сил. Как результат, в OpenBSD при сравнительно небольшом количестве мейнтейнеров поддерживается большая часть современного софта. Текущие примеры:

– порты GNOME и XDG-стек поддерживаются всего двумя людьми
– порты LibreOffice и Chromium — одним и тем же человеком
– KDE 3 и 4 — одним и тем же человеком
– все порты Mozilla-приложений — одним и тем же человеком

О субпакетах и опциях:

OpenBSD Ports, как и другие аналогичные системы, позволяет задавать для пакетов опции сборки. Основные отличия OpenBSD в этой области следующие:

– Количество опций OpenBSD старается поддерживать на минимальном уровне. Это позволяет при разумных усилиях тестировать все или почти все возможные варианты пакетов.
– Используемые при сборке опции записываются в имя пакета через дефис, после номера версии. Например:
`kdelibs-4.11.5-debug`
`vim-7.4.45p2-huge-gtk2-python`

Благодаря этому в репозитории пакетов вместе могут одновременно находиться разные варианты пакетов, и не возникает вопросов вроде «какие опции были использованы для пакета?».

Изначально в FreeBSD Ports было чёткое соответствие: одна сборка — один пакет. Когда требовалось собирать несколько разных пакетов на базе одного порта (например: `postgresql-client` и `postgresql-server`), то один порт включал себя директивой `.include` другой, незначительно модифицировал переменные и предоставлял альтернативный plist. Таким образом, сборка этих двух

пакетов должна была бы происходить дважды, либо требовала довольно «грязных» хаков.

В OpenBSD Ports имеется возможность создавать из одного порта несколько пакетов. Строго говоря, корректнее было бы говорить о субпортах, а не субпакетах, так как создаваемые пакеты являются изначально независимыми и могут иметь любые (допустимые) имена. Однако менять название переменной теперь никто не собирается.

Названия самих субпакетов начинаются с дефиса. Это делается для возможности отличать субпакеты от опций в `FULLPKGPATH` — своеобразном «пути», используемом для пакета с портом, из которого он собирается. `FULLPKGPATH` — своего рода идентификатор (суб-)пакета и играет большую роль на всех этапах работы с пакетами.

Наравне с `FULLPKGPATH` имеется `FULLPKGNAME`. Если `PKGNAME` содержит собственно название пакета и его версию, то в `FULLPKGNAME` к этому добавляются номера ревизии порта (`REVISION`) и эпохи порта (`EPOCH`), а также используемые опции. Именно `FULLPKGNAME` и становится именем конечного файла, содержащего собранный пакет (не считая расширения).

Для портов также можно задавать псевдоопции (`PSEUDO_FLAVORS`). Они используются в тех случаях, когда необходимо управлять сборкой порта без влияния на содержимое создаваемых пакетов. Обычно это ситуации, когда нужно избежать лишних зависимостей при сборке, когда какие-то субпакеты являются лишними, или, например, зависимости нужны только для прогона тестов. Псевдоопции не записываются в имени пакета.

Так как отдельные субпакеты могут выпадать и по другим причинам (например, непригодность для какой-то аппаратной архитектуры) посредством переменных вида `BROKEN-subpkg` и `IGNORE-subpkg`, для конечного определения параметров сборки имеется вспомогательный механизм `bsd.port.arch.mk`(5). Хотя внешне он кажется аналогом `bsd.port.pre.mk` из FreeBSD, назначение у них всё же разное: `bsd.port.arch.mk` используется только для двух вещей:

1. Определение аппаратной архитектуры, под которую собирается пакет.
2. Вычисление переменной `BUILD_PACKAGES`, содержащей список подлежащих сборке пакетов. После включения `<bsd.port.arch.mk>` порт может использовать проверки вида

`.if ${BUILD_PACKAGES:M-subpkg}` для определения, нужно ли включать сборку определённого субпакета.

Результаты данных проверок обычно используются для установки специфичных параметров конфигурации и наборов зависимостей.

О модулях:

Модули — Makefile, подключаемые OpenBSD Ports по просьбе порта. Модули могут как модифицировать переменные, так и устанавливать или добавлять обработчики стандартных операций. Для подключения модуля достаточно добавить его имя в переменную `MODULES`. Примеры модулей:

1. Один из самых простых модулей — `converters/libiconv`. Всё, что он делает — добавляет ряд зависимостей для порта и дополняет `WANTLIB` (список используемых портом разделяемых библиотек).
2. Более сложный модуль — `devel/cmake`. Он используется портами, которые, как не трудно догадаться, собираются при помощи CMake. Данный модуль устанавливает ряд переменных, используемых CMake при конфигурировании и сборке, в том числе для: выбора движка сборки (поддерживаются make и Ninja), перекрытия версий разделяемых библиотек, настройки вывода прогресса сборки и т. д.
3. Модуль `lang/ruby` — пожалуй, самый сложный в дереве портов; он даже получил персональную man-страницу: `ruby-module`(5). С его помощью собираются как обычные программы, зависящие от Ruby, так и Ruby gems, с платформозависимым кодом и без. Из возможностей модуля:

– Поддержка использования одним портом различных версий Ruby (включая JRuby и Rubinius) через механизм опций.
– Санация путей к интерпретатору Ruby в скриптах (замена `#!/usr/bin/env` или `#!/usr/bin/ruby` на то, с чем должен работать порт).

Функциональность, подобная описанной выше, присутствует и в других аналогичных модулях: `lang/python`, `lang/tcl`...

О ручном контроле версий разделяемых библиотек:

Речь здесь пойдёт о самих библиотеках как объектах компиляции, а не о дистрибутивах ПО. Также следует сказать, что нижесказанное относится лишь к «публичным» библиотекам, которые используются при компиляции ПО. Всевозможные плагины и им

подобные сущности не используются при компиляции, а загружаются динамически, посредством `dlopen()`, и располагаются в отдельных подкаталогах.

Для разделяемых библиотек существует соглашение о нумерации: `MAJOR.MINOR[.OTHER...]`, со следующими правилами изменения версии:

- Изменение в библиотеке, приводящее к изменению ABI без сохранения обратной совместимости: увеличение MAJOR.
- Изменение в библиотеке, не приводящее к изменению ABI, или же сохраняющее обратную совместимость ABI: увеличение MINOR.

Номер версии записывается в имени файла и/или в SONAME библиотеки. Когда требуется найти подходящую для программы версию библиотеки, поиск осуществляется с учётом вышеуказанного соглашения: MAJOR должна совпадать с тем, что требуется программе, а MINOR — быть не меньше требуемого программой. Казалось бы, всё довольно просто, но:

- Часто разработчики смешивают версию дистрибутива, поставляющего библиотеку, с её внутренним номером. В результате, смена внутреннего номера библиотеки находится в полном отрыве от реальности.
- В других случаях не производится изменение внутреннего номера библиотеки, или изменяется не та часть номера. В результате, после обновления установленной версии библиотеки, использующее её ПО может оказаться неработоспособным.
- Наконец, иногда в самой операционной системе могут происходить изменения, влияющие на ABI. В случае OpenBSD, правда, обычно делается увеличение MAJOR для libc, что автоматически «разделяет» разные поколения пакетов.

В силу вышеперечисленных причин, в OpenBSD введена практика использования собственной нумерации разделяемых библиотек. Ответственность за соблюдение правил ABI в этом случае ложится на того, кто подготавливает порт или его обновление. Это может показаться большим усложнением, однако на самом деле для всех популярных систем сборки в OpenBSD имеется поддержка перекрытия версий библиотек. Внешне эти изменения сводятся к следующему: когда система сборки готовится слинковать конечную версию библиотеки libfoo, то проверяется наличие переменной окружения `LIBfoo_VERSION` и, если такая установлена, использует

её значение вместо заданного в конфигурации сборки разработчиком. Таким образом, максимум, что приходится делать при подготовке (обновления) порта — это второй раз его собрать, если были пропущены какие-то библиотеки; обычно инфраструктура портов сама обнаруживает и уведомляет о таковых при выполнении операции «`make update-plist`».

Версии разделяемых библиотек играют большую роль при обновлении ПО, так как позволяют осуществлять плавную миграцию между версиями использующего библиотеки ПО.

Операционная система Linux как основа для построения высокопроизводительных систем хранения данных

Александр Фахрутдинов, Сызрань, РФ*

Report describes some Linux kernel subsystems which can be used to create a high-performance data storage. Well-known instruments of storage management (RAID, LVM) are explained as far as cutting-edge technologies for multi-level caching and data tiering. Also, kernel-mode virtual target device for remote access to data storage from other hosts is reviewed.

Системы хранения данных (СХД) — это одна из основ современного мира компьютерных технологий. С возникновением облачных сред и повсеместном внедрением виртуализации возникла необходимость в сверхскоростных хранилищах большого объема и повышенной отказоустойчивости. Кроме того, потребовались «умные» системы, которые хранят вместо несколько копий одних и тех же данных только одну, выделяют под данные именно столько реально имеющегося дискового пространства, сколько требуется, а не сколько запросит пользователь, умеют копировать массивы данных внутри устройства без отправки их на сервер и так далее.

Для организации эффективного хранения данных в Linux применяются виртуальные блочные устройства, которые представляют собой прослойку между собственно аппаратным хранилищем данных, например, жестким диском, и приложениями. Классический случай применения таких устройств программные дисковые массивы — RAID. За обслуживание RAID в Linux отвечает подсистема md (multiplie devices), которая позволяет создавать основные типы RAID — простое объединение дисков (JBOD), RAID уровней 0, 1 (зеркало), 4, 5, 6, а также — RAID 10. . RAID обеспечивает объединение дисков и отказоустойчивость в пределах одного сервера, однако он не решет проблему распределения пространства массива. Для управления дисковым пространством в Linux принято использовать менеджер виртуальных томов — LVM. Он позволяет гибко

*ar.fahrutdinov@gmail.com, http://lvee.org/en/abstracts/105

распределять место на дисках между приложениями, поддерживает создание, удаление и изменение размера тома «на лету», а также позволяет создавать собственные массивы типа JBOD, RAID 0 и 1. Кроме того, LVM обеспечивает создание мгновенных снимков (snapshot) томов вне зависимости от того, поддерживает ли снимки файловая система тома, причем снимки доступны не только на чтение, но и на запись. Однако платой за эту универсальность является низкое быстродействие снимков, что ограничивает область их применения.

В основе md и LVM лежит система проецирования устройств — device-mapper (dm). Она предоставляет единый механизм для создания на базе простых блочных устройств более сложных, наделенных дополнительными функциями. Возможности dm расширяются при помощи модулей, называемых «mapping targets». В стандартном ядре Linux, кроме md и LVM, присутствуют модули общего назначения для следующих целей:

- диагностика и тестирование
  - создание фиксированной задержки — dm-delay
  - имитация сбоев — dm-flakey
  - сбор статистики об обращениях к конкретным областям устройства — dm-statistics
  - имитация пустого устройства — dm-zero
  - проверка цифровой подписи тома — dm-verity
- построение RAID 0, 1 — dm-mod
- шифрование тома — dm-crypt
- доступ к подсистеме md для управления массивами через интерфейс device-mapper — dm-raid

Кроме того, в состав ядра входят модули, применяемые в высокопроизводительных СХД уровня предприятия

- доступ к хранилищу через множество путей
  - с простым резервированием — dm-multipath
  - с учетом длины очереди — dm-queue-length
  - с учетом времени доступа — dm-service-time
  - доступ к разным регионам хранилища через разные пути — dm-switch
- кэширование данных на твердотельных накопителях
  - dm-cache (с версии ядра 3.9, релиз 28 апреля 2013)
  - bcache (с версии 3.10, релиз 30 июня 2013)
  - flashcache (разработка Facebook, не включен в ядро)
- не включены в ядро, но могут быть полезны:

– «многослойное» хранилище из разных типов дисков — btier
– RAM-диск с периодическим сохранением информации — eprd Как известно, в Linux кэшируется только доступ к файловой системе, доступ же напрямую к блочным устройствам не кэшируется, поэтому одним из способов повысить быстродействие, особенно в может быть применение указанных выше модулей. В особенности это справедливо для систем виртуализации.

В системах виртуализации принято выделять дисковое пространство не на этапе создания виртуальной машины, а по мере необходимости (thin provisioning). Device mapper имеет подобную функциональность начиная с ядра 3.2 (релиз 4 января 2012). Модуль dm-thin-pool позволяет создавать виртуальные устройства, которые не резервируют весь предназначенный им объем в момент создания, а расходуют выделенное физическое пространство по мере заполнения самого устройства данными. Также dm-thin-pool поддерживает возврат более не используемых блоков в общий пул, если вышележащая файловая система уведомит его об этом. Кроме того, модуль dm-thin-pool позволяет создавать виртуальное устройство на базе шаблона, в роли которого выступает другое устройство, доступное только для чтения.

Несмотря на то, что device-mapper обладает широкой функциональностью, нельзя не заметить, что его назначение — создание виртуальных устройств «высокого уровня», которые служат для управления уже имеющимся дисковым пространством и должны опираться на программный или аппаратный RAID. В то же время device-mapper не обеспечивает доступ к созданному устройству за пределами хоста, например, по протоколам iSCSI и FC. Для этих целей в ядро Linux не так давно была включена инфраструктура LIO-target.

LIO-target — это разработка компании Rising Tide Systems, которая была лицензирована под GPL и включена в ядро Linux, начиная с версии 2.6.38 (релиз 15 января 2011 г.). В 2013 году разработчики LIO-target покинули Rising Tide Systems и основали компанию Datera, целью которой является разработка программного обеспечения для СХД.

LIO-target состоит из высокоскоростного виртуального блочного устройства с поддержкой расширенного набора команд SCSI, драйверов нижнего уровня (бэкэндов), при помощи которых вир-

туальное устройство отображается на реальное и драйверов верхнего уровня (фронтэндов), при помощи которых можно получить доступ к устройству из-за пределов хоста. Поддерживаются следующие бэкэнды:

- FILEIO — обращение к нижележащему блочному устройству как к файлу через слой виртуальной ФС.
- BLOCKIO — обращение к нижележащему блочному устройству при помощи команд SCSI.
- PSCSI — пересылка команд SCSI физическому устройству, например, RAID-контроллеру, без обработки.
- Memory Copy RAMDISK — блочное устройство в оперативной памяти. Фронтэнды обеспечивают подключение к LIO-target других хостов. В настоящий момент поддерживаются интерфейсы iSCSI, FCoE, Fibre Channel, InfiniBand, IBM vSCSI, FireWare и USB. Кроме того, возможна эмуляция блочного устройства на локальной машине, а также передача такого устройства внутрь виртуальной машины под управлением KVM (vHost).

Особенностью LIO-target является поддержка SCSI-команд аппаратного ускорения для систем хранения данных (VAAI). Эти команды используются, в первую очередь, системами виртуализации и призваны разгрузить гипервизор при таких ресурсоемких операциях, как клонирование виртуальной машины. До недавнего времени этот набор команд был реализован только в коммерческих СХД, теперь же он доступен всем пользователям ОС Linux.

Таким образом, ОС Linux предоставляет функциональность, достаточную для построения на базе конкретной аппаратной платформы надежного и высокопроизводительного хранилища, которое может быть использовано как само по себе, так и в качестве узла в распределенной системе хранения данных.

Steganography — coding and intercepting the information from encoded pictures in the absence of any initial information

Monika Kwiatkowska, Lublin, Poland

Łukasz Świerczewski, Łomża, Poland*

The work includes implementation and extraction algorithms capabilities test, without any additional data (starting position, the number of bits used, gap between the amount of data encoded) information from encoded files (mostly images). The software is written using OpenMP standard which allowed them to run on parallel computers. Performance tests were carried out on computers, Blue Gene/P, Blue Gene/Q and the system consisting of four AMD Opteron 6272. Source code is available under GNU GPL v3 license and are available in a repository OLib.

Introduction

Steganography is the science of determining how to conduct communication so that the presence of the message could not be detected by third parties. It differs from cryptography in a way that in cryptography existence of the message is not negated but its content remains implicit. Steganography hides the fact that any communication has been conducted.

Differences between steganography and cryptography is shown in Table 1.

Traditional Steganography

The use of steganography dates back to the time of Herodotus, the fifth century BC. Examples of traditional steganography can be tattooing the scalp (after the hair grew back information remains invisible). One of the best solutions of this kind applied by the Germans during World War II — microdots technique. It is based on minimazing pictures to such scale so that you can paste them into the text as a dot.

*lswierczewski@pwsip.edu.pl, http://lvee.org/en/abstracts/106

	Cryptography	Steganography
Transforming information into a form incomprehensible by third parties	Yes	Yes
Hiding information	No	Yes
Key usage	Yes	Yes
Hiding the fact of communication	No	Yes
Ensuring anonymity of communicating parties	No	Yes
The amount of information transmitted to the encrypted information	Comparable	Much greater

Table 1. Differences between steganography and cryptography.

Digital Steganography

With development of digital technology, steganography has found a new use also in the field of science. Digital steganography bases on making subtle changes to the original medium, and therefore gives a much more possibilities than traditional steganography.

The carrier of concealed information can be virtually any file (one that can be modified without having to worry about damage to its internal structures). However, the most commonly used are multimedia files — these are relatively large in size and difficult to capture the modification of original file.

Digital steganography depending on the type of operation can be divided into the following categories:

- Substitutional
- Transformational
- Spectrum modification
- Spectrum spread

– Distortional
– Statistical
– Carrier generation

Another field of use for steganography is to communicate using VoIP technology. The Polish jargon adopted using in this case the word 'steganphony'. The first such solution had been proposed by two Polish scientists Wojciech Mazurczyk and Krzysztof Szczypiorski in 2008 at a conference in Mexico [11]. It was based on in transferring the hidden content in the delayed packets, which according to a standard communication protocol (operating in real time) are omitted.

LSB — Least Significant Bit — one of the methods of substitution

The article will summarize often used method of substitution — LSB. Mostly it uses the least significant bits to record information. These bits often carry only noise and are insignificant from the point of view images for example. The principle of LSB operation for one and two least significant bits are shown in Figure 1 and Figure 2.

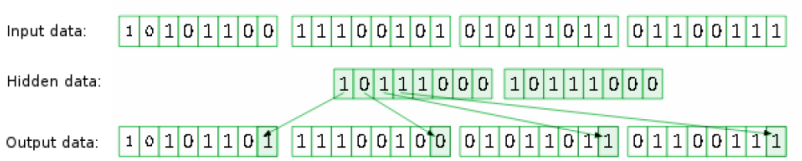


Figure 1. Applying the LSB using one least significant bit.

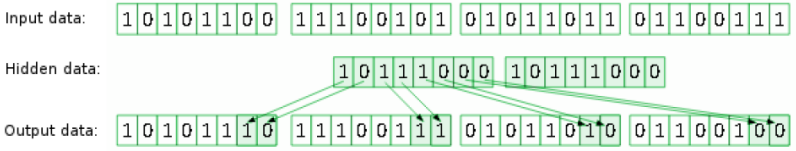


Figure 2. Application of the LSB method using the two least significant bits.

Implementation

Source code had been implemented in pure ANSI C. Sample code for function encrypting text using least significant bits of the image you can see in Listing 1.

```
int crypt_space_1bit(char * buffer, char *open_text,
  long int start_position, long int open_text_size,
  long int space_size)
{
  long int i;
  for (i = 0; i < (open_text_size * 8); i++)
  {
    if (check_bit(open_text[i / 8], i % 8))
    {
      set_bit(buffer[start_position + i * space_size], 0);
    }
    else
    {
     clear_bit(buffer[start_position + i * space_size], 0);
    }
  }
}
```

Listing 1. The function encrypting the data in the image using one least significant bit.

We are passing pointers to two array to the function — the buffer array (image data) and open_text array (array of information to be encoded). In addition, the code uses macros check_bit, set_bit and clear_bit. Their definition is shown in Listing 2.

```
# define check_bit(var,pos) ((var) & (1LL<<(pos)))
# define set_bit(var,pos) ((var) |= (1LL<<(pos)))
# define clear_bit(var,pos) ((var) &= ~{}(1LL<<(pos)))
```

Listing 2. Macros defining operations on bits.

The most important, in the case of this article, however, is a function used to extract encrypted text without knowledge of the initial starting value, or even the length of ciphertext. It has been shown in Listing 3.

```
long int breaker_standard_1bit_unknown_size_openmp(
  char *buffer, long int buffer_size, long int size_down,
  long int size_upper, result_structure *result,
  int number_of_threads)
{
  char *open_text;
  long int start_position;
  long int index_result;
  index_result = 0;
  long int i;
  long int j;
  j = size_down;

  #pragma omp parallel for shared(buffer, buffer_size, \
      size_down, size_upper, result, index_result)     \
      private(i, j, start_position, open_text)         \
      num_threads(number_of_threads)
  for(i=0; i < (buffer_size - j); i++)
  {
    open_text = (char *)malloc((size_upper + 1) *
    sizeof(char));
    start_position = i;
    for (j = size_down; j <= size_upper; j++)
    {
      encrypt_standard_1bit(buffer, open_text,
      start_position, j);
      if (ascii_veryfi(open_text, j) == 1)
      {
        open_text[j] = '\0';
        #pragma omp critical
        {
          result[index_result].start_position = i;
          strcpy(result[index_result].text, open_text);
          index_result++;
        }
      }
    }
    free(open_text);
  }
  return index_result;
```

```
}
```

Listing 3. Function to extract information from the least significant bits without the initial information.

This function searches the least significant bits in the buffer array of size of buffer_size and searches all the possible encrypted patterns of lengths from size_down to size_upper. The result is inserted into the result array. The code uses so-called pragma derived from the OpenMP standard. This pragma aims to parallelize the main loop. Also worth mentioning is separate critical section — it takes care of the correct addition of blocks to the results array. Only one thread can access the critical section at a time. Also a function has been used:

```
int ascii_verify(pointer, size);
```

which verifies whether the starting substring at the pointer of length size is the correct text stored using ASCII code.

Results

Performance results obtained when searching for encoded text inside graphic file with a resolution of 1600×1200 is shown in Table 2. A similar table for resolution 4096×4096 is presented in Table 3. Also, obtained through parallel programming (OpenMP), speedup on IBM Blue Gene/Q and AMD Opteron 6272 is shown in Figure 3.

In the case of a platform consisting of four AMD Opteron 6272 maximum speedup is achieved by using 64 CPU and it reached 36.00. Using computational units used in Blue Gene/Q the speedup reached to 31.915. But it was not obtained, as might be expected, with a maximum (64) number of processors, but only 48. This may be due to the fact that one Blue Gene/Q CPU has only 16 physical cores that implement the execution of 64 threads. It can also be noted that increasing the resolution of the analyzed image from 1600×1200 to 4096×4096 did not increase the runtime of the algorithm even tenfold — for applications executed sequentially on the Blue Gene/Q time increased from 193.11 seconds to 1309.42 seconds (about 6.78 times) and for the Blue Gene/P changed from 560.47 seconds to 4769.00 (about 8.50 times). Analysis has been performed for the speedup depending on the size of the hidden string searched. The results are shown in Table 3 (for AMD Opteron 6272) and in Table 4 (for Blue Gene/Q). Presentation of speedup obtained is presented in Figure 4. Additionally, Figure 5 shows the

Number of threads	Execution time		
	IBM Blue Gene/P	IBM Blue Gene/Q	AMD Opteron 6272
1	4769.00 s	1309.42 s	1282.08 s
2	2482.00 s	651.12 s	652.00 s
4	1311.00 s	308.89 s	401.00 s
6	-	201.24 s	265.00 s
8	-	147.00 s	200.00 s
12	-	97.25 s	132.00 s
16	-	74.19 s	107.00 s
32	-	36.90 s	59.00 s
48	-	25.76 s	43.00 s
64	-	22.49 s	35.00 s

Table 2. Performance results obtained during scan of the image with a resolution of 1600×1200 (searching one least significant bit using length of the search phrase in the range of 10 to 25).

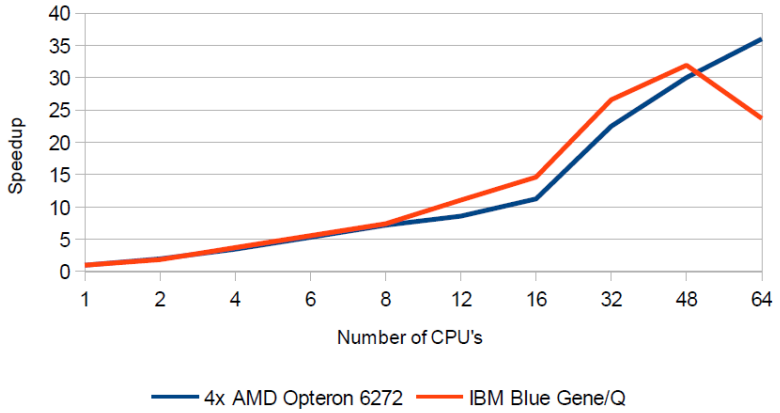


Figure 3. The speedup obtained on platforms AMD Opteron 6272 and IBM Blue Gene/Q while searching an image with a resolution of 1600×1200 (search only one least significant bit of the length of the text to search in the range from 10 to 25).

difference in the runtime of the sequential program execution between two platforms (AMD Opteron 6272 — IBM Blue Gene/Q).

Number of threads	Execution time		
	IBM Blue Gene/P	IBM Blue Gene/Q	AMD Opteron 6272
1	4769.00 s	1309.42 s	1282.08 s
2	2482.00 s	651.12 s	652.00 s
4	1311.00 s	308.89 s	401.00 s
6	-	201.24 s	265.00 s
8	-	147.00 s	200.00 s
12	-	97.25 s	132.00 s
16	-	74.19 s	107.00 s
32	-	36.90 s	59.00 s
48	-	25.76 s	43.00 s
64	-	22.49 s	35.00 s

Table 3. Performance results obtained during scan of the image with a resolution of 4096×4096 (searching one least significant bit using length of the search phrase in the range of 10 to 25).

Number of CPU's	*Down_Size = 10; Upper_Size =*									
	15	20	25	30	35	40	45	50	55	60
1	348.47	757.28	1282.40	1919.38	2668.21	3524.63	4501.62	5631.80	6799.42	8122.23
64	13.00	24.00	34.00	51.00	72.00	99.00	125.00	158.00	193.00	192.00
Speedup	26.805	31.553	37.718	37.635	37.064	35.602	36.013	35.664	35.230	42.303

Table 4. The results of the analysis of performance for different sizes of searched hidden string [Down_Size; Upper_Size] (the ranges of [10, 15] to [10,60]) and platform based on AMD Opteron 6272.

Number of CPU's	*Down_Size = 10; Upper_Size =*									
	15	20	25	30	35	40	45	50	55	60
1	353.18	773.93	1295.46	1949.57	2692.76	3562.74	4545.64	5646.70	6863.10	8222.39
64	9.43	15.64	22.00	31.90	43.01	55.16	67.39	81.99	98.64	116.93
Speedup	37.452	49.484	58.885	61.115	62.608	**64.589**	**67.452**	**68.871**	**69.577**	**70.319**

Table 5. The results of the analysis of performance for different sizes of searched hidden string [Down_Size; Upper_Size] (the ranges of [10, 15] to [10,60]) and platform based on processors installed in the Blue Gene/Q.

The strangest fact can be seen in the analysis of speedup presented in Table 5. According to the measurements on 64 processors for searching

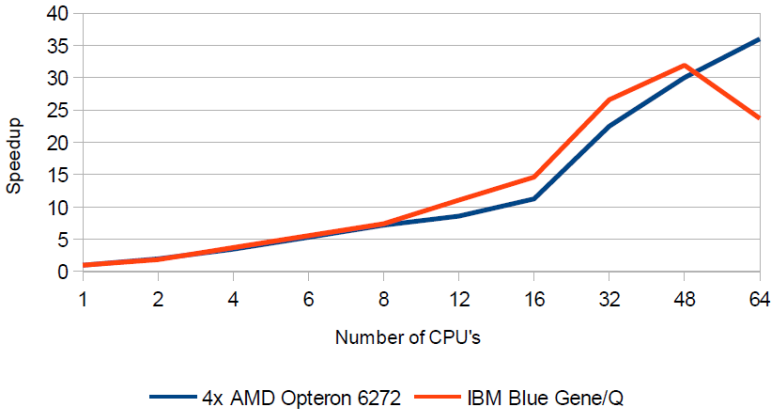


Figure 4. Graph showing the resulting speedup on AMD Opteron 6272 and IBM Blue Gene/Q in the analysis of performance for different sizes of searched hidden string.

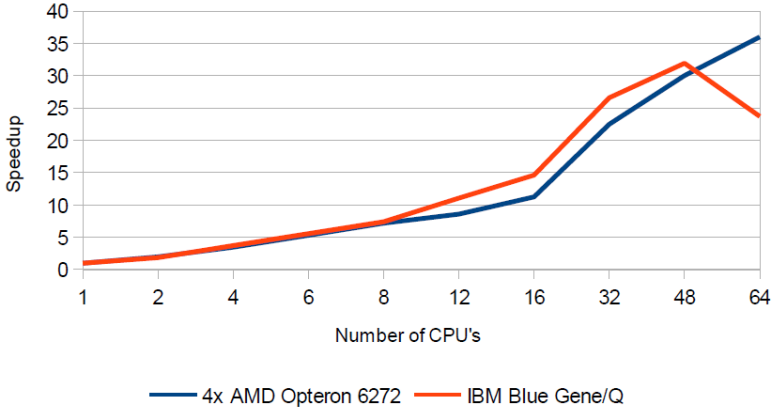


Figure 5. Graph showing the time difference obtained on a platform consisting of the AMD Opteron 6272 and obtained on the Blue Gene/Q in the analysis of performance for different sizes of searched hidden string.

string of a length of 10 to 60 characters, maximum achieved speedup of 70.319, which is impossible and must be the result of the time

measurment error. Problematic data in the table indicated in red. This anomaly is no longer present on the computer with four AMD Opteron 6272. However, in this case, the maximum speedup was only 42.303 and was obtained while searching a string with the length in the range [10, 60].

While in terms of speedup, the platform for the Blue Gene/Q is better but the lower sequential execution runtime of the program is always achieved on a computational node with AMD Opteron 6272 — it is confirmed in Figure 5. Differences in the runtime between these two systems, however, are relatively small and do not exceed 2%.

Conclusions and capabilities of work development

Using the abilities of parallel programming, one can very effectively deal with data processing in the field of steganography. LSB technique is used in most commercial software.

The work can be treated as a base for discussion, because it is very difficult to find a practical application of implemented steganography methods in the modern world. In the so-called 'pure steganography' strenght of encryption is mainly based on lack of knowledge about the techniques used by the individual encrypting the information[7]. So you can not publish this type of algorithms as Open Source. Such an approach, however, does not meet Kerckhoffs principle[10] stating, that a cryptographic system should be secured even if all the details about how it works (except the key) are known, and it is not recommended.

The presented work results concern the simplest variant in which the string is interpreted as ASCII code. Of course you can come up with your own, much more complicated system for the representation of characters.

Figure 6. Logo of the LVEE conference with hidden text 'LVEE – the best conference'.

Acknowledgment

Interdisciplinary Centre for Mathematical and Computational Modeling (ICM), Warsaw University, Poland is acknowledged for providing the computer facilities under the Grant No. G55-11.

Note

The thesis presented on Winter LVEE 2014 Conference in Minsk (Belarus).

References

1. Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." Computational Science & Engineering, IEEE 5.1 (1998): 46-55.
2. Lin, Heshan, et al. "Massively parallel genomic sequence search on the Blue Gene/P architecture." High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for. IEEE, 2008.
3. Haring, Ruud A., et al. "The IBM Blue Gene/Q compute chip." Micro, IEEE 32.2 (2012): 48-60.
4. Keltcher, Chetana N., et al. "The AMD Opteron processor for multiprocessor servers." Micro, IEEE 23.2 (2003): 66-76.
5. OLib Library; http://lib.oproject.info
6. Katzenbeisser S.: Information Hiding Techniques for Steganography and Digital Watermarking, s. 65 – 105, 1999, Artech House
7. Katzenbeisser S.: Information Hiding Techniques for Steganography and Digital Watermarking, s. 20 – 23, 1999, Artech House
8. Katzenbeisser, Stefan, and Fabien AP Petitcolas. Information hiding techniques for steganography and digital watermarking. Vol. 316. Norwood: Artech house, 2000.
9. Kozieł, G. "Przezroczystość danych ukrytych w sygnale audio." Pomiary, Automatyka, Kontrola 58 (2012): 972-974.
10. Auguste Kerckhoffs, "La cryptographie militaire" Journal des sciences militaires, vol. IX, pp. 5–83, January 1883, pp. 161–191, February 1883.
11. Mazurczyk, Wojciech, and Krzysztof Szczypiorski. "Steganography of VoIP streams." On the Move to Meaningful Internet Systems: OTM 2008. Springer Berlin Heidelberg, 2008. 1001-1018.

BOINC — Not only calculations

Łukasz Świerczewski, Łomża, Poland*

Many people participated in the SETI@Home project, which was launched almost 15 years ago — on 17 May 1999. At that time providing ones computing power to the scientists from big American research center was for a common user virtual adventure. Research conducted on shared computers involved (and still do) rather 'popular' subject, searching in the radio waves, signals that may come from foreign civilization. The project has gained popularity and in this respect a comparison to today's Facebook can be quite accurate. One should remember though that this are completely different systems and SETI@Home began operations in 1999, when Internet in Poland was its infancy. However, SETI@-home and BOINC turned out to be a great initiative, which has already nearly two decades and unites people around the distributed computing.

Initially, the BOINC platform has been used only by the above-mentioned project SETI@Home. Today, according to statistics at BOINCstats there are more than 77 projects. A large number is carried out by scientific institutions such as CERN, Oxford University or the University of Washington. But there are also projects directed by individuals.

For the calculations performed, BOINC users are rewarded with 'credits'. They reflect, or rather should reflect person contribution to scientific calculations. However, this isn't an ideal solution. In recent years, administrators of projects have revealed a number of successful or failed attempts to obtain an incorrect (too large) number of credits for a time contributed. Any such attack, however, contributes to the improvement of the gaps in safety and fair 'reward' for the users.

Due to the very large difference in performance, it is difficult to compare the points obtained by the graphics card (GPU, Graphics Processing Unit) with those assigned for work on a regular processor (CPU, Central Processing Unit). For example, one hour of running a job on a video card can get us about 10000 credit points. At the same time, a modern computer without the graphics card supporting calculations, obtained only 200 credit points. This very large difference

*lswierczewski@pwsip.edu.pl, http://lvee.org/en/abstracts/108

is due to the fact that the GPU under specific conditions can actually be up to 50x faster than classical CPU. It's difficult to compare the contribution if we have only one number for reference. One person can exceed 50000 points within one hour, and the other, without the use of graphics cards, after a month. We also need to know that all the calculations can be done on the CPU, but for graphics cards it is not possible to transfer programs for some research projects, because of their architecture. Some projects are therefore limited to processors. The user decides which BOINC projects to join. Sometimes user will have to choose between more interesting projects for science and a little less interesting, but more crediting ones.

BOINC is not just calculations. According to statistics provided by Ohloh website (created by former employees of Microsoft) BOINC is 427 899 lines of source code that has been written by 81 programmers (data from 29 January 2014). When we try to assess the cost of the system, using one of the models for software engineering called COCOMODO, it equals to 113 years of working for one programmer.

Many projects using BOINC also provides their source code of applications or modules. Many users often try to optimize the app-

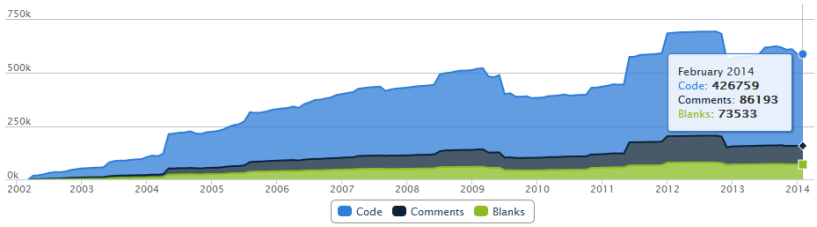


Figure 1. BOINC project – lines of code (data from 5 February 2014). Source: www.ohloh.net.

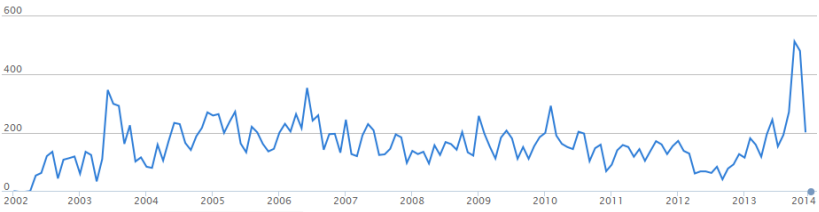


Figure 2. BOINC project – commits per month (data from 5 February 2014). Source: www.ohloh.net.

lication code so that it becomes more afficient . Sometimes such an improved software goes to the administrators of the project, and will be officially added and used as standard on all computers in the system. Also an interesting idea is to announca a competition for the best algorithm and its implementation in a particular programming language. This type of far more 'social' solution was used recently in the project OProject@Home (www.oproject.info) in the subproject Weird Engine, which aims to find the so-called 'weird numbers". After two months and receiving several interesting proposals, it turned out that the best solution has been sent by the Frenchman — Cedric Den Drijver. Basing on the new concept, a new version of the application has been created, which was several hundred times faster than the previous one. This change significantly contributed to the development of the project OProject@Home. With it you can now get far more results.

The only conclusion is that sometimes it is worth to describe a problem and ask for international help in finding a solution. If the issue is interesting, then, despite the fact, that the level of knowledge required to solve it, is very large and beyond knowledge of a statistical student, one can expect a response from some part of the world. For the creators of the BOINC projects, development can move in two directions. The first is calculations performed on thousands of computers and other, working on more and more perfect algorithms, which often are written not only by the official developers. Contribution to the project can be achieved by whole society, which forms around the project.

Краткий обзор базовых лицензий СПО

Ирина Шубина, Минск, Беларусь*

With this report we are going to make a quick and brief overview of the open-source licenses. Starting with the base ideas of the licensing we'll come to the most wirespread and most popular licenses used in the open-source software. Also one of the most important points of the report I would consider the differentiation of the permissive and copyleft licenses.

Введение

Понятие лицензирования

Лицензирование — соглашение сторон, по которому одна сторона предоставляет какие-либо права другой стороне. Лицензирование используется для защиты авторских прав.

Именно правила лицензирования диктуют различные права доступа и использования исходного кода в приложении к программному обеспечению.

Типы лицензирования

Различают пермиссивные и копилефт лицензии.

Пермиссивные лицензии — это лицензии на программное обеспечение, которые практически не ограничивают свободу действий пользователей ПО и разработчиков, работающих с исходным кодом. По своему духу, распространение работы под пермиссивной лицензией схоже с помещением работы в общественное достояние, не требующее отказа от авторского права.

Идея **копилефт** состоит в том, что каждый, кто распространяет программу как с изменениями, так и без них, не вправе ограничивать свободу её дальнейшего распространения либо модификации.

– **«Сильная»** copyleft лицензия разрешает использовать код только программам, созданным под такой же лицензией.

*klasy@tut.by, http://lvee.org/en/abstracts/104

– **«Слабая»** copyleft лицензия разрешает вносить любые изменения в код данной программы. Но она ставит условие, что другая программа, использующая данный код, будет строиться с указанием изначальной в качестве библиотеки. Тогда новая программа может выходить под любой другой лицензией.

В рамках данного типа лицензирования выделяют также полный и частичный копилефт.

Полный копилефт — все части программы (за исключением самой лицензии) могут модифицироваться и распространяться только под лицензией копилефта.

Частичный копилефт — программа может исключать несколько условий копилефт лицензии и при этом включать модификации в рамках какой-то не-копилефт лицензии. Или в некоторых случаях программа, распространяемая под такой лицензией, может следовать не всем принципам копилефта. Например, исключение, сделанное для некоторых программ для GPL связывания.

Краткий обзор лицензий

GPL (General Public License)

Стандартная Общественная Лицензия GNU (GNU General Public License, GNU GPL) — это свободная copyleft лицензия для программного обеспечения (ПО) и других видов произведений[1].

GNU GPL требует распространения с двоичными файлами (в том числе неизменными) исходного кода или письменного обязательства его предоставить (своего или чужого; способы зависят от версии лицензии).

Лицензии, созданные на базе GPL:
– AGPL (Affero General Public License)
– LGPL (Lesser General Public License)

BSD (Berkley Software Distribution)

Существуют две основные версии лицензии BSD, которые необходимо различать: «оригинальная» и так называемая «модифицированная» (вторую в англоязычной литературе часто называют New BSD License). Данная лицензия является пермиссивной.

[1] Взято из Неофициального Перевода GNU GPLv3 (`http://code.google.com/p/gpl3rus/wiki/LatestRelease`)

Лицензия BSD допускает проприетарное коммерческое использование ПО. Для ПО, выпущенного под этой лицензией, допускается встраивание в проприетарные коммерческие продукты. Работы, основанные на таком ПО, даже могут распространяться под проприетарными лицензиями (но всё же обязаны соответствовать требованиям лицензии). Наиболее заметные примеры таких программ — использование сетевого кода BSD в продуктах корпорации Microsoft, а также использование многих компонентов FreeBSD в операционной системе Mac OS X.

Apache Software License

Лицензия Apache даёт пользователю право использовать программное обеспечение для любых целей, свободно распространять, изменять, и распространять изменённые копии, за исключением названия. Данная лицензия не ставит условием неизменность лицензии распространения программного обеспечения, и не настаивает даже на сохранении его открытого статуса. Единственным условием, накладываемым лицензией Apache, является информирование получателя о факте использования исходного кода. В противоположность copyleft-лицензиям, получатель модифицированной версии не обязательно получает все права, изначально предоставляемые лицензией Apache.

В каждом лицензируемом файле должна быть сохранена вся исходная информация о копирайтах или патентах, в каждый изменённый файл должна добавляться информация о проведённых изменениях.

Совместимость лицензий

Совместимость, понятие, возникающее при попытке комбинирования двух и более лицензий, определяет непосредственно возможность комбинации одной лицензии с другими. Совместимость может варьироваться в зависимости от типа и версии лицензии. Различают GPL-совсемстимые и GPL-несовместимые лицензии. Также нужно отметить возможность сочетать закрытые (или проприетарные) лицензии с открытыми. Ссылка на ресурс, где можно посмотреть совместимость лицензий: `http://www.tldrlegal.com/compare`.

Кандалы прогресса: авторское право и научные публикации

Антон Литвиненко, Киев, Украина[†]

Copyright treats scientific publication equal to regular work of art, ignoring its specific nature, that gives some signs of natural monopolies to scientific publishing houses. This severely complicates exchange of scientific information and, so, slows down the worldwide research. Present issues of access to publications together with some present and theoretical methods of their solution are discussed.

Открытость как основа мировой исследовательской деятельности

Во времена Средневековья и более ранних цивилизаций, когда наука в современном виде еще не сформировалась, а объем знаний о мире был невелик, исследователи тщательно скрывали свои результаты, распространяя их максимум в узком круге учеников, зачастую придумывая специальные обозначения и шифры. При экспоненциальном нарастании количества информации в мире вообще и научного знания в частности наука перестала быть делом одиночек-энциклопедистов и вынуждена основываться на тесном сотрудничестве и интенсивном обмене информацией между исследователями и их группами. Каждая научная работа добавляет незначительный фрагмент информации в общую структуру научного знания, общий объем которого давно невозможно изучить, а тем более исследовать одному человеку. При этом, фактически, работа, не обнародованная в общедоступных источниках, не существует для научной общественности.

Доступность для исследователя научных работ его коллег исторически составляла некоторую проблему по ряду технических причин: необходимость физической доставки экземпляров журналов и книг, языковой барьер, сложность поиска, политические аспекты (например, «железный занавес»). Для решения этих проблем

[†]tenebrosus.scriptor@gmail.com, http://lvee.org/en/abstracts/109

с переменным успехом применялся ряд технических подходов (например, реферативные журналы для облегчения поиска), пока они не были фактически решены за счет компьютеризации, четкого доминирования английского языка и краха биполярной политической системы.

Однако, на пути прогресса встали вопросы авторского права.

Особенности авторского права на научные публикации

Фундаментальная информация о природе, получаемая в процессе научного познания, является общественным достоянием. Тем не менее, при работе исследования создается и продукт, который может становиться объектом интеллектуальной собственности. Это касается:

- Разработки новых изобретений; устройств, методов, материалов и прочих результатов, которые могут быть запатентованы;
- Баз данных научной информации, которые, не обладая правами на содержащуюся информацию, обладают правами на ее компиляции и результаты поиска;
- Авторское право на научные публикации.

С точки зрения авторского права, научные публикации являются обычными литературными произведениями.

Принципиальные отличия между научной публикацией и литературным произведением в контексте авторского права

1. Литературное произведение можно опубликовать с помощью любой организации, способной подготовить его к печати (при необходимости) и физически создать нужное количество экземпляров. Именитость издания может иметь некоторое значение для дальнейшей судьбы произведения, но основным является физическое наличие тиража. В то же время, для научного произведения место опубликования имеет критическое значение:
   - Научное произведение должно пройти рецензирование в той или оной форме. Именитость места публикации (издательства, периодического издания, представительства

конференции и т. д.) играет значительную роль и выступает показателем качества произведения (в том числе через систему наукометрических параметров, характеризующих место публикации — например, импакт-фактор журнала).
   - Авторитет места публикации влияет на принятие работы как научной, а также ее добавление в общую структуру научного знания — будут ли другие исследователи читать работу, воспримут ли ее всерьез, будет ли она доступна в научных поисковых системах.
   - В некоторых случаях в качестве приемлемых научных работ воспринимаются только опубликованные в местах из определенного авторизованного списка (например, списки ВАКов).
   - Выход на рынок научных публикаций достаточно затруднен, так как малоизвестному издателю сложно претендовать на поступление для публикации интересных высококачественных работ.
2. Первичные научные работы (по материалам оригинальных исследований), как правило, содержат достаточно уникальную информацию, которая редко где-либо дублируется полностью. Таким образом, необходимость доступа к конкретной публикации (а в процессе текущих работ теоретически может понадобиться доступ к любой ранее опубликованной работе) может быть критической без возможности замены какой-либо другой публикацией.
3. Как правило, исследователь не получает значительного дохода от продажи своих научных публикаций (часто не получает его вообще), мотивы публикации и распространения своих трудов более нематериальны.

Таким образом, рынок научного издательства имеет черты природной монополии (олигополии), в котором интерес издателя и автора существенно различен. Однако, это совершенно игнорируется при юридическом регулировании.

Типичная политика научных издательств

1. Продается подписка на печатный журнал и/или на онлайн-доступ к электронным версиям статей за все время или опре-

деленный период; а также продается онлайн-доступ к отдельным публикациям при отсутствии общей подписки;
2. Подписка стоит дорого. Например, годовая подписка на журнал Angewandte Chemie стоит $11529 (печатный + онлайн) [1]. А таких журналов только по химии десятки самых важных, а по широкому набору дисциплин (например, для научной библиотеки) сотни и тысячи. Доступ же к единичной статье стоит десятки долларов. Так, библиотека Гарварда в 2013 году сделала заявление о непомерных расходах на журнальные подписки, призвав ученых публиковаться в бесплатно распространяемых журналах [2].
3. Подписки могут продаваться сразу на группу журналов. Фактически, часть журналов может продаваться в довесок.
4. Гонорары авторам невелики или вовсе отсутствуют.
5. Рецензенты не получают денег за работу.
6. Условия лицензионных соглашений предусматривают передачу практически всех прав издательству.
7. Высокая стоимость не гарантирует однозначно высокого качества научных публикаций.
8. В последнее время большинство научных журналов было скуплено тремя большими издательскими домами: Wiley, Elsevier, Springer.

Издатель научных публикаций торгует воздухом, пользуясь монопольным положением. Такая ситуация ведет к затруднению научных исследований, повышению порога вхождения в научную деятельность, недоступности результатов оригинальных исследований для широких масс интересующихся людей.

В последние годы это (а также ряд других причин вроде поддержки SOPA и PIPA) вызвало даже попытки «бунта» среди ученых, в том числе и весьма известных, — в частности, бойкот издательства Elsevier [3,4].

Методы борьбы

1. Заграница нам поможет. Ссылки, которые нужно скачать, направляются знакомым «утекшим мозгам» или временно стажирующимся/работающим за бугром коллегам или друзьям. Они скачивают статью, пользуясь местной подпиской.

2. Расширенные версии предыдущих. Сообщества для реквестов вроде жжшного pdf.livejournal.com, межбиблиотечные подписки и т.д.
3. Связка проектов sci-hub.org и libgen.org. Первый представляет собой систему прокси с парсерами и прочими техническими дополнениями для автоматического доступа к журналам через компьютеры в западных университетах. Второй — огромный фонд научных публикаций, ставящий цель собрать большинство научных публикаций в истории мировой науки вообще. Скачанные через sci-hub статьи автоматически попадают в libgen, при повторной скачке предлагается уже готовая копия. При неудачной попытке поиска через libgen предлагается открыть статью через sci-hub.
4. Опция издательства — авторы платят за публикацию, после чего она становится открытой всем.
5. Open-access журналы — аналог предыдущего, но применяется для всего журнала. Как правило, не имеет печатной версии. В то же время, могут финансироваться как авторами, так и из других источников. Некоторые исследователи критикуют низкое качество рецензирования, но проводимые ими исследования не вполне строгие (традиционные журналы страдают теми же проблемами) [3,5], и эти проблемы могут быть объяснены новизной явления. В качестве примера open-access журнала можно привести PLOS Biology, который взнимает с авторов зависящую от страны плату и публикует работу под лицензией Creative Commons CC-BY [6].
6. Тома журналов выкачиваются постатейно ботами и выкладываются на торрентах.
7. Максимально используются оставшиеся у автора права — рассылка небольшого числа авторских копий, публикация препринтов (не все журналы). Публикация препринтов эффективна в интеграции с Google Scholar (рядом со ссылкой на статью позволяет скачать найденный препринт с другого адреса).
8. Использование временного открытия некоторых статей во время рекламных кампаний, доступа по паролям для рецензентов через проприетарные поисковики и прочих лазеек.

Таким образом, большинство имеющихся методов борьбы являются откровенно пиратскими, а существующие легально не решают

целостной проблемы (open access распространяется только на работы, явно опубликованные по этой модели).

Для существенного прогресса в вопросе авторских прав на научные работы следует разрабатывать и вносить изменения в законодательство об авторском праве, выделяя научные публикации в особый вид произведений со специальным регулированием. Например, существенное уменьшение (до нескольких лет) времени перехода в общественное достояние.

Таким образом, несмотря на принципиальную приверженность открытости информации, научному сообществу еще только предстоит пройти путь по либерализации недопустимой ситуации с авторскими правами, который уже успешно проходит общество программистское.

P. S. Во время обсуждения доклада слушателями были названы еще некоторые цифровые хранилища и библиотеки, из которых можно бесплатно или за разумную цену поличить желаемую литературу по определенным областям знаний. В частности, сайт CiteSeerX [7] (спасибо Алексею Чеусову).

Литература

[1] http://ordering.onlinelibrary.wiley.com/subs.asp?ref=1521-3757&doi=10.1002/(ISSN)1521-3757
[2] http://www.vestifinance.ru/articles/22492/print
[3] http://theoryandpractice.ru/posts/8440-znaniya-dlya-vsekh
[4] http://www.svoboda.org/content/article/24892099.html
[5] http://habrahabr.ru/company/cyberleninka/blog/197946/
[6] http://www.plosbiology.org/static/information
[7] http://citeseerx.ist.psu.edu

Опыты над людьми и Octave: FOSS-based GSR measurements

Ольга Карабутова, Минск, Беларусь*

The paper describes practical experience of creating a biometric device to evaluate user's stress level. Signs specific to stress are considered as well as the device prototype and intermediate research results. Free/libre open-source software used to evaluate and analyse raw biometric data is covered.

Вступление

С каждым днем различные устройства, использующие биометрические данные (фитнес-гаджеты, сканеры отпечатков пальцев, умные часы, умные очки, одежда, протезы, управляемые силой мысли, нейрокомпьютерные интерфейсы) находят все более широкое применение, а сами устройства становятся меньше и удобнее для повседневного использования.

Перед нами стояла задача создания биометрического устройства, которое могло бы отслеживать уровень стресса человека в течение дня, недели. Хотя обычно человек сам в состоянии определить, находится сейчас он в стрессовом состоянии или нет, но он далеко не всегда способен объективно оценить стрессовую нагрузку на свой организм, тем более, не всегда очевидны причины стресса.

Стресс

Под стрессом (от англ. stress — нажим, напряжение) понимают общую реакцию организма на физическое или психологическое воздействие, выводящее его из состояния равновесия (нарушающее гомеостаз), а также соответствующее состояние нервной системы и организма в целом. «Изобретатель» стресса канадский эндокринолог Ганс Салье доказал, что стресс ведет к истощению организма и к очень многим нарушениям здоровья.

Реакцию организма на стресс можно выразить фразой «дерись или беги». За большинство физиологических изменений, отвечают

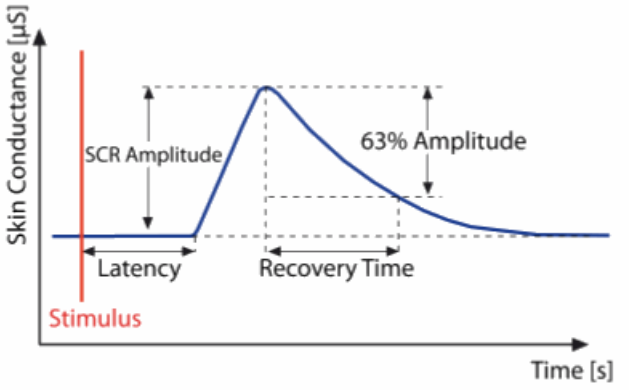
*nadfilka@gmail.com, http://lvee.org/en/abstracts/119

две нейроэндокринные системы, управляемых гипоталамусом: симпатической и парасимпатической. Симпатическая нервная система подготавливает организм к атаке, обеспечивая выброс в кровь глюкозы, увеличение кислорода в крови и артериального давления, сердцебиения, расширяет зрачок, активирует железы и мускулатуру, сокращает ток периферических сосудов, а также подавляет деятельность большинства внутренних органов, такие как пищеварение. Действие парасимпатической системы во многом противоположно. Поэтому можно выделить стандартные признаки, сопутствующие стрессовой ситуации:

– артериальное давление
– содержание сахара и кислорода в крови
– повышение частоты сердцебиения, учащение дыхания
– расширение зрачков
– специфическое сокращение лицевых нервов (мимика)
– уменьшение слюноотделения

С точки зрения использования в конечном устройстве для выявления стресса нас интересуют в первую очередь метрики, пригодные для неинвазивного измерения и обладающие при этом достаточной достоверностью.

Ряд исследований показывает, что для детектирования стресса достаточно одного датчика кожно-гальванической реакции (Galvanic Skin Response или GSR).



GSR — биоэлектрическая реакция, обусловлена деятельностью потовых желез. Если приложить к коже электрическое напряже-

ние, то между двумя участками кожи можно замерить электрическое сопротивление или проводимость. Во время стрессового воздействия потовые железы кожи выделяют микрочастицы пота, в результате чего сопротивление кожи меняется.

На рисунке показана идеальная реакция на стрессовый стимул, обозначенный красной линией с подписью «Stimulus». Можно заметить временную задержку отклика (Latency) и последующее время восстановления (Recovery), т.е. адаптации к новому гомеостазу или релаксации после точечного стресса.

Устройство

Разработка включает две части: носимое устройство в виде браслета и программное обеспечение, устанавливаемое на персональный компьютер или мобильное устройство.

Носимое устройство включает следующие части:

- датчик GSR
- акселерометр
- микропроцессор
- память
- RF-модуль (Bluetooth Low Energy)
- устройство питания (батарея CR2032 или литий-полимерный аккумулятор)

Биометрия считывается с аналоговых датчиков, усиливается, конвертируется в цифровой сигнал. Далее показания агрегируются в памяти, находящейся на устройстве. При портативном ношении данные собираются на устройстве за продолжительный период времени. Также они могут передаваться в режиме реального времени на персональный компьютер или мобильное устройство для обработки — как по интерфейсу USB, так и через беспроводной интерфейс Bluetooth Low Energy, который широко используется в медицинских устройствах.

Для обработки данных на PC использован GNU Octave.

Методика тестирования

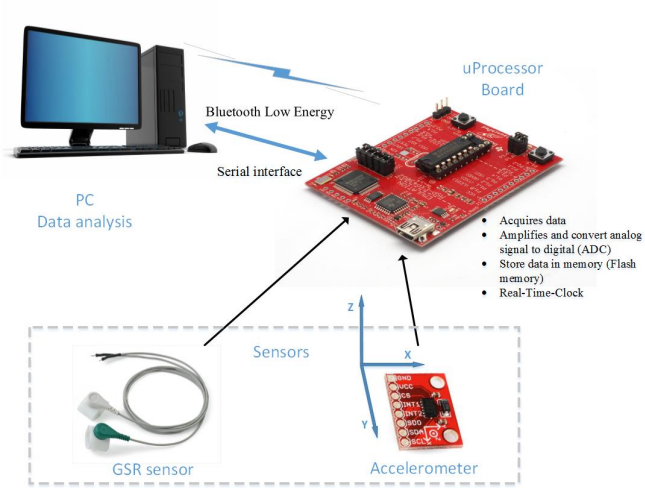
Для проверки работоспособности прибора использовалась серия тестов следующего вида. После закрепления датчиков подопытному давался адаптационный период не менее 15 минут. Далее испытуемый подвергается стрессовым воздействиям, которые следовали

одно за другим в случайном порядке. Воздействия проводились с перерывом в 20—30 минут. Между последним стрессовым воздействием и завершением предусматривалась пауза не менее 10—15 минут. Общая продолжительность одной сессии составляет 1—3 часа.

Датчики GSR крепятся на двух пальцах одной руки или на запястье подопытного, но должны быть разнесены минимум на 1 сантиметр расстояния между ними, с обеспечением плотного контакта. Плотность контакта на пальцах обеспечивается заводским креплением на «липучке», а на запястье — креплением на резинке с регулируемой длиной браслета. Дополнительно на теле человека закрепляется акселерометр.

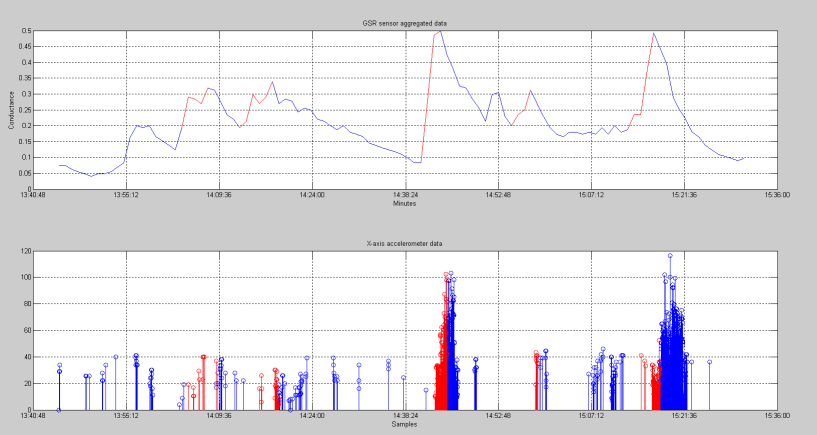
В качестве типов стрессовой нагрузки применялись следующие:

Физическое воздействие	Физические упражнения Изменение температуры в помещении
Когнитивная нагрузка	Решение интересных, но не тривиальных задач



	Прохождение/выполнение задач вне компетенции на время Соревновательные игры
Эмоциональная нагрузка	Видеоролики или материалы, вызывающие эмоциональный отклик Обсуждение событий, вызывающих радость, положительные эмоции Требование отчета вышестоящим начальником (этот тест показал слабую эффективность) Ситуация экзамена

Данные с датчиков сохранялись в cvs-файл и затем обрабатывались в Octave. Там же обработанные данные пропускались через алгоритм, выдающий на выходе, когда у человека был стресс (по показаниям GSR), и был ли он вызван физической активностью или нет (по показаниям акселерометра).



Параллельно все происходящее во время опытов тщательно документировалось наблюдателем. Пример журнала:

13.57	Испытуемый разговаривает по телефону Виден пик повышения GSR, но как стресс программа это не выделила (не достаточно существенное отклонение от общего уровня GSR)
14.05	Игра «Морской бой» с ограничением по времени
14.09	Окончание игры
14.10	Обсуждение игры и тактик. (Испытуемый выиграл)
14.44	Физическое упражнение «бег на месте» Стресс, вызванный физической активностью испытуемого, легко отличить по показаниям акселерометра
14.45	Завершение физических упражнений
15.00	Включение обогревателя в комнате
15.16	Выполнение разнообразных физических упражнений
15.21	Завершение физических упражнений

Результаты испытаний

Тестирование показало, что стресс, вызванный физической нагрузкой, детектируется 100%. Что касается прочих причин, было сложно иногда создать однозначно стрессовую ситуацию для подопытного. Например, ситуация отчета перед начальством не всегда вызывала у подопытного волнение (подопытному было известно, что он в ситуации «теста» и ситуация создана искусственно). Тем не менее, у разработчика, который также иногда выступал в роли подопытного, сообщение о визите вышестоящего начальства, как и сам визит, однозначно повышало регистрируемый уровень стресса.

Игровые ситуации дают однозначный результат, как и во время игры, так и при последующих обсуждениях результатов и стратегий. Аналогично дело обстоит и с просмотром видеоматериалов, вызывающих с высокой долей вероятности эмоциональный отклик у реципиентов.

Octave

GNU Octave — свободная система для математических вычислений, использующая совместимый с MATLAB язык высокого уровня. По сути, Octave — это свободный клон MATLAB, благодаря чему и динамически подгружаемые модули (*.m-файлы), и пользовательские функции можно очень легко портировать из одной среды в другую.

В среде профессиональных исследователей MATLAB занимает лидирующие позиции (благодаря крайне обширному набору вычислительных функций практически для всех областей применения) а к его свободным клонам часто относятся с предубеждением. Однако по нашему опыту GNU Octave оказался чрезвычайно мощным математическим инструментом, с помощью которого можно экспериментировать, моделировать устройства и обрабатывать цифровые данные.

Не в последнюю очередь это достигается благодаря тому, что Octave позволяет подключать сторонние утилиты, а также общие или предметно-ориентированные библиотеки функций для вычислений. Например, для построения графиков по умолчанию Octave использует Gnuplot, хотя также есть на выбор не менее 5 альтернативных графических пакетов.

Сторонняя библиотека Octave-forge включает множество пакетов, среди которых для решения класса задач анализа биометрических данных нами были отобраны следующие:

Для получения данных из подключаемых устройств:

- **instrument-control** позволяет взаимодействовать с внешними устройствами через последовательный и параллельный порты, I^2C, TCP/IP, интерфейсы GPIB, VXI11 и USBTMC.
- **communications** — передача цифровых данных, кодов коррекции ошибок ECC, модуляции и поля Галуа.

Для цифровой обработки сигнала:

– **signal** — наборы функций для обработки сигналов (а также обработки изображений, проектирования цифровых фильтров и систем связи – БПФ, фильтры различных порядков).
– **ltfat** — библиотека частотно-временного анализа сигналов. Позволяет производить большое количество линейных преобразований, включая преобразование Габора и вейвлет-преобразование, а так же шаблоны для конструирования окон (моделирование фильтров) и утилиты для манипулирования коэффициентами.

Для статистики, анализа данных и машиного обучения:

– **nan** — набор утилит для статистики и машинного обучения, работает как с Octave, так и с Matlab, умеет обрабатывать как непрерывные данные, так и данные с пропущенными значениями.
– **mvn** — кластеризация многомерного нормального распределения
– **fuzzy-logic-toolkit** позволят создавать экспертные системы на основе нечеткой логики, проводить кластеризацию нечеткими алгоритмами, а также проектировать нечеткие нейронные сети.
– **fl-core** содержит базовые функции для нечеткой логики.
– **queueing** содержит набор функций в том числе и для анализа цепей Маркова

В итоге для решения нашей задачи был задействован меньший набор пакетов Octave для построения эффективной обработки данных и ее статистического анализа (преимущественно использовались пакеты signal). Определенно мог быть задействован пакет instrument-control для приема данных на ПК. Но Octave — поистине мощный математический инструмент, с помощью которого можно экспериментировать, моделировать устройства и обрабатывать цифровые данные.

Реализация UEFI SecureBoot в ALT Linux

Михаил Шигорин, Киев, Украина*

There are many misunderstandings and myths related to UEFI in general and SecureBoot extension in particular; I've implemented both within ALT Linux distribution and would like to help sort things out.

Вводная

Следует оговориться, что задокументированная реализация построена поверх предшествовавшей работы по поддержке загрузки в режиме UEFI, о чём было бы полезно написать отдельное HOWTO к тому моменту, когда мы этим занялись (конец 2012 года). Поскольку на данный момент основные дистрибутивы скорее умеют такую загрузку, было решено ограничиться пользовательской страничкой на вики и документированием средств сборки.

Ситуация с SecureBoot отличалась: по состоянию на 2013 год загружаться без отключения этой «ручки» и лишних проблем умели только Fedora, openSUSE и Ubuntu, причём сведения о реализации оказывались либо устаревшими, либо лоскутными, либо вместе.

Проблему можно охарактеризовать в нескольких частях, некоторые из которых являются (по крайней мере пока) одноразовыми, а другие — повторяющимися либо имеющими шанс свалиться на голову.

Одноразовая морока

Как и описывал Matthew Garrett, для авторизации на sysdev.microsoft.com пришлось обеспечить наличие IE/Windows, сертификата Symantec и логина live.com.

Сертификат Verisign/Symantec понадобится единственный раз, чтобы аутентифицировать компанию при регистрации на портале UEFI CA, а затем каждый раз, когда вы готовитесь загрузить

*mike@altlinux.org, http://lvee.org/en/abstracts/117

очередной объект на рассмотрение и подпись. Это сертификат Authenticode class 3; его можно получить с одноразовой скидкой через Windows Dev Center. Для получения сертификата понадобится IE7+ с нестандартными настройками ActiveX — чтобы сначала его сгенерировать и импортировать, а затем экспортировать в файл.

Та же Windows-система (реальная или виртуальная) пригодится для заведения акаунта на sysdev.microsoft.com. Предлагаемые шаги:

1. завести акаунт live.com (sysdev.microsoft.com требует его для работы);
2. привязать/аутентифицировать компанию к этому акаунту (однократная процедура, заключается в подписывании тестового бинарника сертификатом Symantec/Verisign);
3. прочитать и принять лицензионные соглашения Microsoft.

Вам понадобится создать базу данных NSS с импортированными в нее сертификатами Symantec, скачать winqualexe.zip с сайта sysdev.microsoft.com, распаковать и подписать его (используя pesign), а затем закачать подписанный winqual.exe обратно. На момент осени 2013 г. требуется подписать лицензионные соглашения «Windows Logo Program Testing Agreement V3» и «UEFI Firmware Agreement».

Многократная морока

Получение подписанного бинарника shim — последующая многократная процедура, которая выполняется каждый раз, когда вы собираете новую версию оного в пакет для выпуска. В процессе будет задействован все тот же Windows-хост или виртуалка с установленным Silverlight, подготовленный shim.efi с публичной частью вашего cacert, сертификат Symantec (чтобы подписать заявку), а также акаунт sysdev с правами подписи UEFI. Процедура занимает от нескольких дней до нескольких недель.

Процесс включает следующие шаги:

1. подготовка shim.efi;
2. заливка shim.cab на sysdev.microsoft.com;
3. отправка запроса на sysdev@microsoft.com;
4. ожидание и периодическая проверка входящих и спам-бокса вашего почтового акаунта;
5. скачивание подписанного бинарника, если/когда процесс наконец будет завершен.

По большому счету, есть два варианта EFI shim: старше версии 0.5 либо новее (сама версия 0.5 поломана). Начиная с версии 0.5, реализованы дополнительные ограничения по вторичному загрузчику, что потребует подписывать еще и образы ядра. Версия 0.4 лишена этого функционала, но нет гарантии, что вам удастся ее подписать (как это удалось сделать нам).

При подаче заявки с shim.cab вы получите номер заявки — submission ID, который потом будет использоваться в переписке с sysdev@microsoft.com для идентификации вашей заявки. Переписка может включать типовой вопросник от sysdev@microsoft.com на предмет дополнительных подробностей о вашем shim. По-видимому, в sysdev проверяют очередь заявок раз или два в неделю. Чтобы ускорить процедуру, можно заранее выслать ответы на ссылкой на полученный submission ID в теме письма.

Собирая все вместе

После получения подписанного бинарника shim понадобится аппаратный или виртуальный стенд со включенным SecureBoot, поддержка UEFI boot/install, а также терпение для внесения доработок в уже работающие части. Вы должны убедиться в наличии верифицированной загрузочной цепочки при включенном SecureBoot как для установочного/загрузочного образа, так и для установленной ОС:

1. bootx64.efi или shim.efi (shim);
2. grubx64.efi (первичный бутменеджер или загрузчик);
3. vmlinuz (ядро) либо вторичный загрузчик;

цепочка может варьироваться после shim, но основные принципы сохраняются:

1. shim верифицируется ключем, который предоставляется для firmware с помощью KEK/DB, а затем верифицирует бинарник загрузчика с помощью встроенного сертификата, MOK или firmware;
2. последующие загрузчики могут обмениваться информацией с shim чтобы использовать информацию о вашем сертификате, встроенном в него, и MOK, добавленных пользователем конкретной машины.

ALT way

Ваша реализация может сильно отличаться; ALT Linux сейчас использует следующую:

shim → refind → elilo → vmlinuz для install/live/rescue media

shim → grub2 → vmlinuz для установленной системы

Мы используем refind в качестве бут-менеджера, поскольку некоторые реализации UEFI поставляются с кривыми реализациями выбора цели загрузки или не имеют таковой вовсе; elilo работает в качестве фильтра, позволяющего загрузить ядро Linux — нам это необходимо, т. к. мы хотим дать пользователям возможность загрузки неподписанных ядер и при этом не оставлять дыру, позволяющую загрузить что попало.

Как пропатчить KDE4 под OpenBSD

Вадим Жуков, Москва, РФ*

For a long time OpenBSD did not ship modern KDE versions. But during two last years situation changed. The talk is about what happened behind the scenes to make the working KDE4 platform usable by OpenBSD users. There will be covered such things as tweaking CMake modules, authentication support and so on. Also, there was a success in making KDE 3 and 4 coexist that involved solving assorted technical problems. The experience gained during process could be useful for porting other software on OpenBSD as well as for porting in general.

KDE — крупный открытый проект, включающий в себя множество различных компонентов. Определённое множество этих компонентов составляет KDE Software Compilation (KDE SC):

- основные библиотеки, например: kdelibs, kdepimlibs;
- служебные приложения, например: lnusertemp, kded, kreadconfig;
- основные пользовательские приложения, например: Konqueror, Dolphin, KMail;
- компоненты среды KDE, например: kde-workspace (включает, среди прочего, KDM и KWin), kde-artwork;
- дополнительные компоненты: игры, обучающие приложения, всевозможные полезные утилиты.

За пределами KDE SC находится ряд связанных проектов, которые можно условно разделить на две группы:

- KDE Support: полусамостоятельные проекты, необходимые для сборки и работы KDE, например: сервер Akonadi, libattica, набор модулей для CMake.
- базирующиеся на KDE проекты, как-то: Calligra Suite, Digikam SC, KMyMoney, Amarok и т.д.

Портирование KDE состоит из следующих этапов:

1. Определение и портирование недостающих зависимостей. В случае OpenBSD это были весь KDE Support (включая сервер Akonadi), Virtuoso и ряд небольших библиотек. Моменты, которые хочется отметить:

*zhuk@openbsd.org, http://lvee.org/en/abstracts/111

- Akonadi: ни с одним бэкендом до сих пор не наблюдается 100% стабильной работы. В лучшем случае очередь запросов к серверу забивается из-за плохого планирования. Патчи, внесённые разработчиками Akonadi в SQLite-бэкенд, выглядят не совсем корректными, поэтому в OpenBSD добавлена возможность использовать штатный, из состава Qt4.
- Virtuoso: ряд встроенных в дистрибутив Virtuoso тестов на регрессии до сих пор не проходит, по различным причинам. Однако, во-первых, имеющейся функциональности вполне хватает для нужд Nepomuk — единственного пользователя Virtuoso; а во-вторых, в KDE в данный момент ведётся работа над следующим поколением Nepomuk под кодовым названием Baloo, в которой значительно переделана и упрощена внутренняя архитектура данной подсистемы, в результате чего хранилище RDF-данных вроде Virtuoso становится не нужным.
- Модули CMake: часть этих модулей дублировала имеющиеся в поставке CMake (в портах OpenBSD силами dcoppa@ всегда есть актуальная версия), некоторые потребовали полного переписывания. А в случае с FindGettext.cmake пришлось писать обвязку для KDE4-портов, которая исправляет на лету файлы CMakeLists.txt, в которых производится компиляция файлов локализация.

2. Портирование kdelibs и других базовых компонентов. Этот процесс в OpenBSD, на самом деле, продолжается до сих пор. Все известные критичные проблемы устранены, для ряда подсистем обновлены или написаны «с нуля» специфичные для ОС реализации. На данный момент под OpenBSD в KDE недоступны следующие возможности:

   - Solid: отсутствует поддержка OpenBSD, поэтому KDE не умеет получать данные об устройствах в системе.
   - KDM: отсутствует поддержка multi-seat X (т.н. «быстрое переключение пользователей») из-за ограничений базовой ОС.
   - Незначительно ограничены возможности управления звуковой подсистемой из-за привязки некоторых компонентов Phonon к ALSA. Учитывая сравнительно малую проблемность звуковой подсистемы в OpenBSD (в большин-

стве случаев не требуется делать абсолютно ничего), данный пункт упомянут скорее для полноты картины.

3. Портирование остальных частей KDE SC. Выполнено практически полностью, со следующими оговорками:
   - Отсутствует поддержка Web-камер и Jingle/GoogleTalk в Kopete.
   - Порт Kalzium отмечен как BROKEN из-за проблем с зависимостями. Проблема обходится при использовании официального репозитория WIP-портов, в котором имеются обновлённые, но не до конца проработанные порты (https://github.com/jasperla/openbsd-wip/).
   - Не собираются биндинги к libattica. Проблема скорее в самом фреймворке Smoke, но, поскольку эти биндинги не требуются ни для одного портированного приложения, изучение проблемы отложено до лучших времён.
4. Обеспечение совместной установки приложений KDE 3 и 4. В случае с OpenBSD для этого была переименована часть каталогов, в которых хранятся данные приложений KDE 3. Благодаря централизованности управления списками каталогов с данными в KDE, данное решение удалось разработать и внедрить в течение всего двух месяцев, включая отладку и тестирование. Единственный побочный эффект — некоторое разрастание пользовательского профиля KDE, из-за того, что приложение может считать данные из одного локального конфигурационного файла, а записать данные уже по другому пути, внутри того же профиля — это связано с тем, что иерархия «системных» и «пользовательских» каталогов в KDE едина.

   Следует отметить, что, в отличие от других немногочисленных ОС, предоставлявших возможность параллельной установки KDE 3 и 4, в OpenBSD это сделано без ущерба для целостности системной структуры каталогов: никаких /opt или /usr/local/kde3.
5. Обеспечение совместной работы приложений KDE 3 и 4. Это потребовало внесения изменений в пакеты kdelibs, kde-runtime и kde-workspaces, чтобы приложения KDE 4 использовали альтернативные пути к различным хранилищам временных файлов: KDE создаёт каталоги вида «/var/tmp/kdecache-username», а для доступа к ним использует символические ссылки в каталоге профиля пользователя. Данное изменение, в отли-

чие от предыдущего, было внесено в KDE 4 с целью избежать ненужных проблем у пользователей KDE 3.

6. Обеспечение совместной сборки KDE 3 и 4. Пользователям, не собирающим пакеты самостоятельно, эта проблема совершенно не очевидна и не видна. Однако для менйтейнеров она составляла заметную головную боль: невозможность собрать KDE 3 при установленном KDE 4 и наоборот ставила крест на используемой в OpenBSD технике непрерывной сборки пакетов. Для решения этой проблемы пришлось убедить в её серьёзности Марка Эспи, главного разработчика инфраструктуры портов и, в прошлом, мейнтейнера портов KDE. Это было сделано на EuroBSDCon 2013, а уже через несколько недель появился патч для dpb(1), позволявший разграничивать сборку портов по так называемым тегам. Фактически, это самый крупный «костыль», который был добавлен в OpenBSD в связи с портированием KDE.

   В последние пару месяцев KDE 4 были подготовлены и частично внесены в дерево портов патчи, позволяющие KDE 3 собираться в присутствии KDE 4 и наоборот. Однако уже подготовленное полное решение в OpenBSD 5.5 не попадёт из-за близости момента заморозки дерева, которая должна произойти буквально в дни проведения нынешней конференции.
7. Портирование приложений за пределами KDE SC. На текущий момент подгтовлены в WIP-репозитории и по большей части отлажены порты для Calligra Suite, Digikam SC, K3b, Kdenlive, Kile, KMyMoney, KTorrent, Tellico и Yakuake. Их импортирование так же ожидается на следующей итерации цикла разработки OpenBSD. Желающие же могут уже сейчас подключить WIP-репозитории и собрать интересующие их порты самостоятельно.

SDN сегодня

Дмитрий Орехов, Минск, Беларусь*

Today Software-Defined Networking is still a cutting-edge rather than a common production technology. In spite of this, SDN technologies are evolving actively, as well as it's enthusiasts community and engineer's culture. It holds out hope that common usage of SDN in world-wide networks is a matter of the closest future. And you're able to be a part of this! We made a review of current SDN state and most valuable solutions which are available right now to build and manage SDN, it's platforms and technology stacks. Wherein we paid a special attention on Open Source software which enables SDN, so everyone can contribute in the future of networking.

Программно-управляемые сети

Концепция SDN (Software-defined networking) основана на идее разделения уровня передачи данных и уровня управления правилами, по которым данные передаются. Обычно говорят, что на уровне управления действует Контроллер, а на уровне передачи данных — Свич. Контроллер при этом не только устанавливает правила для Свича, но и принимает от него сообщения о событиях, происходящих в сети, обеспечивая обратную связь.

OpenFlow

Ясно, что одним из ключевых элементов SDN является протокол взаимодействия между Свичом и Контроллером. В настоящий момент таким протоколом в основном является OpenFlow. Ключевое понятие этого протокола — Flow или правило, по которому обрабатываются пактеы внутри Свича. Flows объединены в таблицу Flow Table, которые, в свою очередь, объединены в конвейер.

Фактически, Flows представляют собой микрокоманды для программирования сети. Как средство управления сетью, OpenFlow прошел большой путь от примитивной концепции, включающей в

*Dmitry_Orekhov@epam.com, http://lvee.org/ru/abstracts/113

себя одну Flow Table и весьма ограниченный набор критериев, в версии 1.0, до версии 1.3 (готовится к выпуску версия 1.4), с конвейером таблиц, улучшенной обратной связью между Свичом и Контроллером, поддержкой множества критериев сравнения как то MPLS, IPv6 и т. д., и возможностью писать весьма сложные «программы» для управления сетью.

Открытые лицензии для открытого стандарта

Уже сейчас наиболее интересные и полные решения, позволяющие строить OpenFlow топологии, опубликованы под различными Open Source лицензиями. Интересно то, что большая часть из них поддерживается крупными корпорациями — поставщиками сетевого обурудования и программного обеспечения. Решения эти используют очень разнообразные стеки технологий. Ниже представлен краткий список наиболее интересных, на мой взгляд, сообществ и их решений.

- Project Floodlight (бывший OpenFlowHub.org) Представляют контроллер OpenFlow Floodlight, базирующиеся на нём решения для управления сетью, Java API, REST API, а также средства для тестирования OpenFlow устройств. Основные языки программирования — Java, Python, C
- CPqD Представляют OpenFlow свич и контроллер, OpenFlow драйвер с API для создания кастомных контроллеров, поддерживают дистрибутив OpenWRT с поддержкой OpenFlow. Основные языки — C/C++ и Python
- Ryu SDN framework Очень функциональный фреймворк для создания контроллеров, написанный на Питоне. Оттестирован с большим числом свичей, поддерживается OpenStack'ом
- FlowForwarding.org Представляет решения с использованием двух различных стеков:
  - Erlang VM: LINC Switch, Loom controller, Tapestry — анализатор сети
  - Java VM: Warp OpenFlow драйвер с API для создания кастомных контроллеров и Warp Controller на базе фреймворка Akka

Высокоуровневое программирование

И все же OpenFlow остается низкоуровневым «языком программирования»; энтузаистами SDN был запущен проект Frenetic, ставящий своей целью создание и развитие высокоуровневого языка программирования для сетей. С запуском же проекта Pyretic (Python + Frenetic) эта инициатива приобрела черты вполне реального domain-specific language.

Прогулки с монстрами

Усилиями SDN-сообщества, The Linux Foundation и корпораций-доноров был запущен весьма амбициозный проект OpenDaylight, представляющий собой SDN-стек и фреймворк для создания SDN-сетей.

Поскольку технологии SDN прекрасно подходят для создания виртуальных сетей, они широко используются в проекте OpenStack

Оптимизм

Будущее SDN, еще недавно весьма туманное, теперь вызывает сдержанный оптимизм. Множество решений опубликованных под свободными лицензиями позволяют привлечь энтузиастов и стимулируют активное развитие области.

Свободное программное обеспечение на службе у психолога

Олег Кондрашов, Алексей Городилов,

Александра Кононова, Москва, РФ*

Psychology is a purely humanitarian science. Nevertheless, specialists more often prefer solutions of application tasks to be done by PC. Automation of diagnostic processes gives more accuracy when choosing healing approach. Free software offers very useful tools for it for a psychologist or a coach.

Одной из самых экзотических областей нашей жизни, где может найти применение программное обеспечение (причем не обязательно свободное), является психология. Ведь как только мы пытаемся найти что-нибудь, разработанное специально для профессиональных психологов или коучей, мы определенно терпим неудачу. Другими словами, почти ничего специализированного для этого пласта наук не написано.

На мой взгляд, это обуславливается двумя факторами. Самый значительный заключается в серьезной разрозненности и несистемности многих современных теорий, что затрудняет формулирование «запроса на ПО» со стороны их приверженцев, в сторону программистов. Только за последние сорок лет появилось около тысячи новых течений, так или иначе перекликающихся друг с другом и имеющих чисто практическую, но не научную подоплеку. Соответственно, совершенно не ясно, какие процессы можно автоматизировать и для чего может использоваться софт при работе непосредственно с людьми.

Другим фактором является то, что многие эмпирические, эзотерические и игровые выводы некоторых течений в практической психологии принципиально не поддаются формализации и автоматизации.

В частности, так называемый «Дизайн человека». Системы его построения сильно разнятся, хотя при анализе используют одни и

*illinc@bk.ru, http://lvee.org/ru/abstracts/114

те же входные данные. Это происходит потому, что каждый специалист по Дизайну человека понимает его по-разному, и нет общепринятой схемы для построения.

Однако, для решения локальных, сиюминутных задач, зачастую встающих перед психологами и коучами, вполне применимы существующие программы, применяемые порой для совершенно других целей. Примеры таких задач:

- Определение психологических травм человека по внешности
- Вычисление диаграммы уровней мышления человека
- Определение архетипа человека
- Выведение клиента из состояния паники, шока
- Помощь в принятии решений
- Имитация работы подсознания

Все эти задачи, конечно, не могут быть в полной мере решены с помощью компьютера, но компьютерное моделирование и вычисление может повысить точность диагностики, а следовательно эффективность рекомендаций психолога.

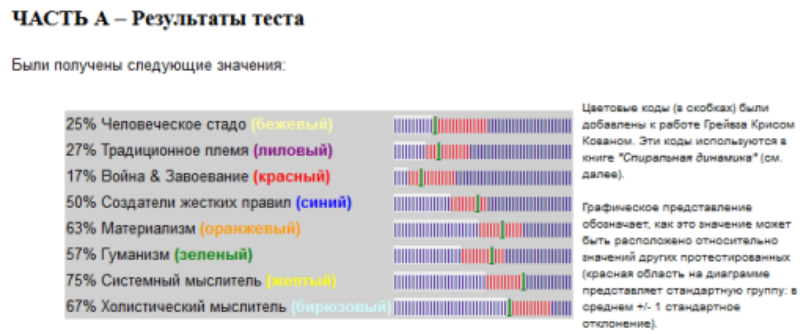
Распознавание лиц с помощью OpenCV

Эта библиотека имеет сравнительно большие возможности для идентификации лиц на фотографиях, очертаний и фигур. Согласно теории Л. Бурбо, форма лица и форма тела человека напрямую демонстрируют нам те или иные психологические травмы. Необходимо научить искусственную нейронную сеть распознавать форму лица, а затем идентифицировать травму по фотографии. Например, лицо человека, страдающего травмой «Униженного», можно представить в виде силуэта:

Программа, найдя на фотографии пациента похожее положение глаз, размер подбородка и др., выдаст нам результат о наличии или отсутствии травмы «Униженного». То же подходит и для других травм, только зависит от качества фотографий. Тело человека может содержать и несколько травм. Это потребует большего числа циклов обучения.

Вычисление диаграммы уровней мышления

Согласно теории американского ученого К. Грейвза, все живые системы в мире развиваются по спирали. По спирали также эволюционирует и мышление человека, приобретая четкие, характерные черты для каждого уровня мышления. Посредством онлайн теста можно построить для себя «диаграмму мышления» по типу jobEQ ASQ, ответив на некоторые вопросы:



Осуществляется такой тест на свободном движке, например, testMaker 3.3.

Лаборатория экспериментов PEBL

PEBL — Psychology Experiment Building Language. Это простой язык программирования для создания различных экспериментов на основе шаблонов. С помощью его реализации PEBL 0.13 можно запускать написанные на PEBL скрипты. Хорошей демонстрацией возможностей языка является скрипт, генерирующий случайные числа для выявления проблем с памятью.

Обработка звука в Audacity

Для выведения человека из состояния шока широко используются специальные звуковые дорожки, в которых правильно подобраны частоты сигнала. С помощью Audacity легко подготовить звуковые дорожки, например для следующих целей:

– Устранение шумов и изоляция лишних звуков из многослойной дорожки, чтобы научить клиента фокусировать внимание на отдельных фракциях звука (обучение концентрации внимания).

– Волны человеческого голоса имеют разную громкость на разном уровне частот. Разделив дорожку собственного голоса клиента на части, можно продемонстрировать ему, буквально «на каких частотах» он разговаривает в обычной жизни. Это позволит задуматься, что именно стоит изменить в поведении и общении.
– Для активизации правого полушария мозга человека психологи часто применяют вдохновляющую музыку. Компиляции из нескольких композиций удобно делаются в Audacity простыми инструментами монтажа.

Поскольку большинство программ используется психологами для прикладных целей, то можно сделать вывод: свободное ПО даже на сегодняшний день предлагает весьма солидный инструментарий и совершенно не уступает собственническому ПО. Тем не менее, вопрос о разработке централизованного ПО для психологов и коучей, которое собрало бы в себя новые течения, было бы высококачественным и при этом свободным, остается на сегодняшний день открытым.

Краткий словарь

Коучинг — персональный систематический психологический тренинг, в рамках которого тренер путем совместного с клиентом поиска решений, обеспечивает улучшение качества жизни, самообучение и личностный рост клиента. (перевод определения Association for Coaching)

Архетип — (от греч. arche — начало и typos — образ) Прообраз, первоначальный образ, идея. Был введен К. Г. Юнгом, как древнейшие общечеловеческие символы, лежащие в основе мифов, фольклора и культуры. В данном контексте под архетипами подразумеваются совокупности поведенческих моделей, характерных для того или иного человека.

«Дизайн человека» — одна из современных психологических теорий, в основе которой лежит предположение, что момент рождения человека четко определяет возможные ветви его развития. Знание о конкретном моменте позволяет выстроить довольно сложную «карту» эволюции личности. Интересна тем, что было предпринято множество попыток автоматизировать процесс построения Дизайна человека.

Обзор GNURadio

Юрий Адамов, Минск, Беларусь*

Gnuradio is a software framework intended in particular to building software defined radio. New functions and new external devices are added easily by writing plugins. It also has graphical front end to rapidly create software signal processors. I will demonstrate creating software radio receivers with GNUradio and commodity components. Also I will discuss using gnuradio as a general signal processing tool and writing plugins for gnuradio, using processing of electroencephalogram (EEG) as an example.

От обычного радио к программно определяемому радио

Изначально все компоненты радиоприемников и радиопередатчиков создавались из дискретных компонент. Из отдельных катушек, резисторов, конденсаторов, транзисторов и т.п. собирались устройства, которые применяли определенные математические преобразования к исходному сигналу, принятому в антенне, дабы получить какой-то полезный сигнал на выходе. Например, типичный супергетеродинный приемник осуществляет предварительное усиление сигнала (умножение на константу в идеале), преобразование частоты (перемножение исходного сигнала и сигнала внутреннего генератора — гетеродина), фильтрацию получившегося сигнала полосовым фильтром, детектирование сигнала.

Возьмем более сложную систему — телевизор. Он также может быть описан как совокупность сравнительно простых блоков, осуществляющих простые математические преобразования сигнала: перенос телевизионного сигнала на промежуточную частоту (гетеродин — смеситель), усиление полного сигнала на промежуточной частоте (фильтр, усилитель), выделение несущих звука и изображения (фильтры), детектирование звука и изображения (AM детектор, FM детектор), усилители, выделение сигналов цветности (фильтры), детектирование сигналов цветности (FM детекторы),

*begemotv2718@tut.by, http://lvee.org/ru/abstracts/115

линии задержки (фазовый преобразователь), сумматоры (восстановление R, G, B), схемы синхронизации (пороговый детектор).

Несмотря на чисто аналоговую элементную базу, проектирование сложного радиоэлектронного прибора подобно проектированию программной системы — мы разбиваем систему на слабосвязанные функциональные блоки, блоки разбиваем на подблоки, пока не дойдем до элементарных операций (усиления, фильтрации, детектирования, порогового детектирования и т.д.).

Сейчас, когда частоты цифровой электроники уже довольно глубоко в радиодиапазоне, стало возможным сначала преобразовать сигнал из антенны в цифровую форму и затем делать многие из перечисленных выше операций уже на цифровых процессорах. При этом проектирование системы можно устраивать как средствами обычных языков программирования, так и с помощью старых добрых блок-схем.

Программно-определяемые радиосистемы, собственно, и предназначены для такой задачи. Gnuradio — как раз представитель подобного класса систем.

Рассмотрим чуть подробнее возможности современного процессора по обработке радиосигнала. Если мы возьмем диапазон длинных волн (30-300 кГц, километровые волны), то за один период этого сигнала успевает пройти более 10000 тактов процессора. Этого более чем достаточно, чтобы сделать с сигналом все что угодно (например, принять одновременно все длинноволновые станции и записывать каждую из них в отдельный файл). При этом процессор справится даже если мы добавим к ДВ существенный кусок средневолнового диапазона (300 кГц – 3 МГц). Однако выполнить то же самое с наиболее интересными коротковолновым, УКВ и СВЧ диапазонами так просто не получится: в запасе остается слишком мало тактов. Здесь приходится использовать компромисс: с помощью аналоговой электроники часть представляющего интерес диапазона можно перенести по частоте в диапазон от 0 до нескольких мегагерц, а потом уже этот сигнал оцифровывать и обрабатывать с помощью программно-определяемого радио.

Практически те же самые средства, как программные так и аппаратные, работают и в обратную сторону — для подготовки к передаче сигнала в эфир.

Предварительное преобразование и оцифровку сигнала делает специальная аппаратура Universal software radio peripheral (USRP, универсальное внешнее устройство для программного радио). Она

содержит в себе предварительный усилитель, преобразователь частоты и АЦП. До недавнего времени это были специализированные и потому сравнительно дорогие устройства (например, USRP фирмы Ettus research стоит порядка \$700). Однако несколько лет назад были открыты свойства USRP у сравнительно дешевых устройств для приема DVB-T телевидения на базе чипов Elonics E4000 и Raphael Micro R820T. Эти устройства стоят порядка \$20, имеют довольно широкий диапазон частот настройки (54—2200 МГц у E4000, 24—1700 МГц у R820T), большую частоту выборки АЦП (2.5 Ms/s) и приемлемую чувствительность. Недостатки таких приемников DVB-T — закрытость и невозможность передачи сигнала. Подобных недостатков лишен проект HackRF — свободное аппаратное обеспечение с возможностью приема и передачи. Однако, HackRF дороже и может вызвать претензии у служб радиоконтроля и таможни (поскольку является передатчиком).

Архитектура Gnuradio

Все схемы в GNUradio строятся из *блоков*. Блок — это элементарная единица обработки сигнала, «черный ящик» с несколькими входами и выходами. Выходы одних блоков можно соединять с однотипными входами других блоков, строя блок-схемы. Блоки можно соединять как программно (внутри программ на Python или C++) так и в графическом редакторе (GNURadio companion). Алгоритм обработки сигнала внутри блока реализуется на C++ (с ассемблерными вставками), для всех блоков существует обвязка на Python.

Блоки делятся на источники (sources), потребители (sinks), промежуточные и т.д. Также имеется большая библиотека графических виджетов, оформленных в виде блоков (поддерживаются Wx- и QT-виджеты).

Источники не имеют входов, зато имеют выходы. К источникам относятся интерфейсы USRP-приемников, программные генераторы сигналов, интерфейсы к микрофонам звуковой карточки, файлы с записями сигналов... Например, описанные в предыдущем разделе приемники DVB-T обрабатываются блоком RTL-SDR.

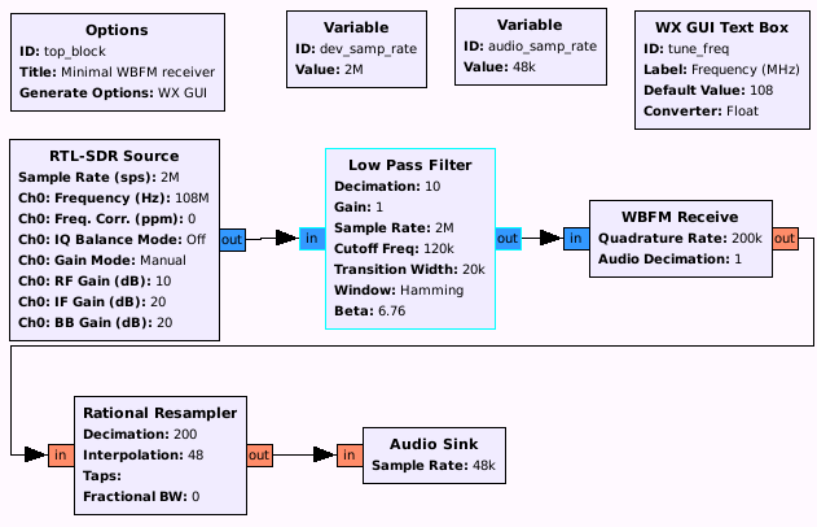
Потребители (sinks) не имеют выходов, но имеют один или несколько входов. К потребителям относят интерфейсы USRP-передатчиков, выходы звуковой карточки, выводы в файл. Некоторые

графические виджеты также относятся к потребителям (например анализатор спектра, осциллограф).

Имеются потребители и источники, конвертирующие сигнал в/из TCP или UDP-поток данных. С помощью таких блоков можно разделить обработку сигнала на несколько машин.

Типичный представитель промежуточных блоков — фильтр. Имеется полный набор стандартных FIR или IIR-фильтров. Также фильтры могут конвертировать частоту выборки сигнала (децимация). Еще один класс промежуточных блоков — детекторы.

Имеются также вспомогательные блоки — например, представляющие глобальные переменные или виджеты для настройки глобальных переменных.



Для примера разберем, как сделать работающий FM-приемник из стандартных компонент GNUradio. Блок-схема самого простого приемника показана на рисунке. Он состоит из источника (RTL-SDR), фильтра нижних частот (селектор сигнала одной станции), FM-детектора, еще ресэмплера и потребителя — аудиокарточки. Частота настройки и частота выборки задаются глобальными переменными. Для удобства работы добавим виджеты настройки и красивую картинку для спектра сигнала. Чтобы превратить эту схему в приемник AM-сигнала, достаточно заменить детектор. Можно

устроить одновременный прием двух FM станций, если они помещаются в полосу частот 1 МГц. Для этого можно использовать блок сдвига частоты frequency XLATE.

Нестандартные применения GNUradio и написание своих блоков

Хотя проект GNUradio изначально предназначен для задач обработки радиосигналов, он предоставляет набор стандартных хорошо оптимизированных компонент, пригодных для любой цифровой обработки сигналов. В качестве еще одного примера рассмотрим, как можно обработать данные энцефалограммы человека (запись мозговой активности). Для анализа в энцефалограмме выделяют несколько ритмов, они лежат в разных диапазонах частот (альфа 8—13 Гц, бета 13—30 Гц, гамма $>$ 30 Гц, тета и дельта 1—7 Гц). Для определения активности в этих каналах нужно отфильтровать соответствующие диапазоны частот, по возможности сдвинуть частоты к нулю, продетектировать и усреднить сигнал. Однако данные от энцефалографа поступают в виде csv-файла, который стандартные средства GNUradio читать не умеют. С помощью gr_modtool мы создаем шаблон нового компонента

```
% gr_modtool newmod csvfile
```

Добавляем источник с типом выходных данных float (f).

```
% cd gr_csvfile
% gr_modtool add -t source csvfile_f
```

Это создает структуру директорий:

```
% ls gr_csvfile
apps cmake CMakeLists.txt docs examples
grc include lib python swig
```

Нам в первую очередь потребуется отредактировать файлы **csvfile_f_impl.cc** и **csvfile_f_impl.h**. В качестве образца возьмем стандартный компонент чтения из wav-файла, который есть в исходных кодах GNUradio. Нужно также отредактировать xml-файл описания компонента в директории grc. Далее, после компиляции с помощью cmake, получаем нужный компонент.

clsync — live sync utility

Дмитрий Окунев, НИЯУ МИФИ, Москва, РФ*

The report focuses on a live syncing utility "clsync"developed by UNIX-tech department of NRNU MEPhI for an LXC-based infrastructure. "clsync"is free and open solution, that appears as replacement of "lsyncd"in fine-tuned systems. A practical experience of applying the utility to setup an LXC HA cluster, a backuping system and a configuration files syncing through an HPC cluster is given.

Поддержка синхронности файлов между узлами — это типовая задача при реализации большинства кластеров. Для решеиня разных задач формулируются разные требования к синхронности файлов между узлами, но в рамках данной работы в первую очередь рассматриваются системы высокой доступности (от англ. «high availability», далее — «HA»). В HA-системах типовыми требованиями являются:

- высокая производительность (сравнимая с производительностью локальной файловой системы на локальной дисковой подсистеме);
- высокая доступность (отказ сервиса не более нескольких секунд);
- высокая надёжность (не создавать дополнительных отказов сервиса за счёт использования данного решения);
- универсальность (применимость для широкого спектра разных решений, разворачиваемых на данной системе).

В общем случае, данная задача на данный момент остаётся нерешённой. Однако создано большое множество решений, которые в разных приближениях решают эту задачу, определяя свои порядоки приоритетов данных требований. И основными подходами на данный момент являются:

- **файловые системы только для чтения** — это простое и надёжное решение, однако имеет очень узкий спект применимости;

*xaionaro@gmail.com, http://lvee.org/ru/abstracts/118

– **единые общие файловые хранилища** — простое и универсальное решение, но создаёт единую точку отказа;
– **блочная репликация** — надёжное и универсальное решение, но создаёт большие потери производительности и очень чувствительно к качеству интерконнекта;
– **файловая репликация** — компромиссное решение, которое не выполняет ни одно требование полностью;
– комбинации вышеперечисленных подходов.

В НИЯУ МИФИ кроме прочих задач необходимо поддерживать HA-инфраструктуру с резервным копированием для быстрого создания VPS, в которых допускается работать малоопытным web-программистам [1]. Опыт показал, что использование блочной репликации для в данной ситуации приводит к недопустимой потере производительности. Поэтому был использован lsyncd [2] для синхронизации LXC-контейнеров между узлами кластера и их резервного копирования.

Однако lsyncd оказался сложным в настройке для ряда экзотических ситуаций, малопроизводительным, а также недостаточно надёжным, простым в отладке, гибким и переносимым. В результате был написана альтернативная реализация под внутренние нужды отдела UNIX-технологий НИЯУ МИФИ — clsync [3]. Далее clsync был документирован и опубликован через репозитории пакетов Debian.

Clsync написан на GNU C99 с использованием inotify [4] и адаптирован для реализации LXC HA-кластера (в связке с rsync [5]), создания системы резервного копирования (аналогично) и синхронизации конфигурационных файлов на HPC-кластере (в связке с pdcp [6]).

Для реализации LXC HA-кластера в clsync начата реализация подсистемы уведомления других инстанций посредством multicast. Однако данная подсистема перестала быть актуальной после перенастройки кластера для запуска отдельной инстанции для каждого LXC контейнера. На данный момент в процессе реализации API и менеджер для централизованного управления инстанциями clsync.

Литература

[1] Окунев Д.Ю. Опыт внедрения отказоустойчивого web-кластера для портала приёмной комиссии НИЯУ МИФИ. Научная сессия

НИЯУ МИФИ, 2012, http://www.pandia.ru/text/78/343/297.php
[2] «Manual to Lsyncd 2.1.x», https://github.com/axkibe/lsyncd/wiki/Manual-to-Lsyncd-2.1.x
[3] «file live sync daemon based on inotify, written in GNU C», https://github.com/xaionaro/clsync
[4] «inotify — monitoring file system events», http://linux.die.net/man/7/inotify
[5] «rsync — a fast, versatile, remote (and local) file-copying tool», http://linux.die.net/man/1/rsync
[6] «pdcp(1) – Linux man page», http://linux.die.net/man/1/pdcp

Обзор свободного фреймворка ROS — операционной системы для роботов

Валерий Касьяник, Брест, Беларусь*

ROS (Robotics Operation System) is an open-source framework designed to ease developing software for robots. Report covers it's main advantages, use cases, existing shortcomings and future development roadmap.

Введение

Робототехника сейчас одна из самых активных областей развития технологий. Множество компаний и энтузиастов собирают своих собственных роботов, разрабатывают новые аппаратные и программные решения, накапливая критическую массу для возниконовения новых технологий. Так программное обеспечение роботов эволюционирует огромными темпами от простейших прошивок контроллеров устройств до сложнейших программных систем управления для решения широкого круга задач в робототехнике. И как в любой сложной системе здесь возникают архитектурные проблемы, проблемы повторного использования кода, разработки и отладки, коллективной разработки.

Для решения этих проблем был предложен фреймворк ROS (Robotics Operation System)

Фреймворк ROS как операционная система

Фреймворк ROS — это набор утилит, библиотек, соглашений, который упрощает разработку сложного программного обеспечения для различных робототехнических платформ. ROS работает поверх классической операционной системы, обеспечивая службы мета-операционной системы робота: аппаратную абстракцию различных устройств робота, низкоуровневый контроль этих устройств, реализацию часто используемых функций и задач, передачу сообщений между процессами, и управление пакетами. Можно сказать,

*val.tut@gmail.com, http://lvee.org/ru/abstracts/116

что ROS управляет взаимодействием вычислительных устройств с реальным миром.

Как и другие ОС, ROS состоит из двух частей: непосредственно ядра фреймворка **ros**, и **ros-pkg**, набора поддерживаемых сообществом пакетов, которые реализуют различные функции робототехники: позиционирование, планирование, восприятие, моделирование и др.

ROS выпускается в соответствии с условиями BSD- лицензии и с открытым исходным кодом. Кроме того, ROS бесплатен для использования не только в исследовательских, но и в коммерческих целях. Пакеты из **ros-pkg** распространяются на условиях различных открытых лицензий и авторы сами решают какую лицензию использовать.

Ключевые архитектурные особенности ROS

Основными концепциями ROS являются узлы, сообщения, темы, сервисы. Архитектурно все вычислительные задачи выполняются в узлах ROS, которые обмениваются между собой информацией посредством сообщений. Эти сообщения узлами публикуются в темах, которые разделяют эти сообщения на группы интересов. Когда некоторому узлу необходимо получать сообщения с определенными данными, этот узел подписывается на определенную тему. Для реализации синхронной передачи сообщений, которая необходима в определенных случаях, ROS определяет сервисы — механизм, который работает по принципу вопрос-ответ.

Основные способы применения ROS

Как фреймворк ROS подразумевает наличие методологии его применения для решения задач робототехники.Рассмотрим его основные сценарии применения.

Тестирование одной задачи. Этот способ применения хорошо подходит для исследовательских экспериментов. Обычно даже простая исследовательская задача требует большого количества вспомогательного кода, который работает с оборудованием или реализует связанные задачи. ROS преоставляет разработчику возможность сосредоточиться только на своем коде.

Логирование и воспроизведение данных. ROS обеспечивает логирование и воспроизведение записанных результатов. Механизм реализован через единый канал **rosout**.

Обмен разработками и повторное использование кода. Также ROS реализует свой собственный пакетный менеджер, который предоставляет пользователю возможность использовать уже отлаженные и готовые алгоритмы, искать уже реализованные идеи, удобно хранить их и использовать в своих проектах. Для этого существуют утилиты **rospack**, **rosbash**, которые реализуют функциональность пакетного менеджера unix операционных систем для быстрой навигации и работы с пакетами ROS. Кроме того, такой подход позволяет эффективно организовать совместную разработку и перенос стороннего ПО в пакеты ROS.

Тестирование и отладка. Для тестирования и отладки в ROS имеются специальные утилиты **rviz**, **rxplot**. Первая позволяет динамически отображать граф связей запущенных узлов, публикуемые темы, сервисы. Вторая утилита предоставляет инструмент для анализа передаваемых сообщений, отображения временных данных, показаний датчиков и т.д.

Управление сложностью. Как отмечалось выше, архитектура ROS состоит из узлов, которые реализуют некую функциональность робота. Так как сложные задачи навигации или управления манипулятором требуют множества вычислений, то они как правило реализуются множеством связанных узлов, которые могут выполнять одни и те же действия для разных роботов. ROS решает проблему дублирования функциональности с помощью кластеров. Так например, собрав и отладив узлы управления манипулятором, разработчик может запустить несколько копий такого кластера для управления несколькими манипуляторами, а координировать эти кластеры будет отдельный высокоуровневый узел.

Преобразование данных. Потоки данных между подсистемами робота разнородные и требуют соответственно различных преобразований. Для упрощения разработки и отладки информационных потоков в ROS был предложен отдельный модуль **tf**, который отвечает за все информационные преобразования между узлами и сервисами. Таким образом, при написании узла разработчик не заботится о согласовании своего модуля с другими, а выносит эту функциональность в модуль **tf**.

Недостатки и проблемы ROS

Основной проблемой ROS является ее главное достоинство. и продуманая архитектура является проблемой для новичков, только начинающих знакомство с системой или для исследователей, плохо знакомых с разработкой ПО. На решение этой проблемы сообщество выделяет значительные ресурсы: ведется портал с документацией по каждой версии ROS, выпускается серия книг, множество туториалов по использованию ROS с другими популярными инструментами.

Вторым недостатком считается отсутствие поддержки ОС Windows, т. к. многие исследователи и разработчики по-прежнему продолжают использовать эту ОС. Над этим недостатком также ведется работа сообществом, и есть предварительные результаты.

Развитие ROS

ROS продолжает активно развиваться во многих направлениях, что обусловлено философией коллективной разработки. Активно добавляются новое аппаратное обеспечение, сделаны драйвера для множества популярных устройств: роботов-пылесосов iRobot, Microsoft Kinect, Arduino, Raspberry Pi, Leaf Maple и многих других. Сейчас одним из популярных направлений является портирование основных инструментов ROS на ARM, чтобы у разработчиков появилась возможность запускать узлы на самых разных устройствах. Также расширяется выбор инструментов разработки: C++, Python, Matlab.

Выводы

В результате всего вышеперечисленного на ум приходит аналогия с разработкой Linux, когда ядро Линуса Торвальдса объединило и дало новый толчок разработке свободного программного обеспечения. Похоже, что бурно растущая область робототехники тоже получила тот центральный элемент, который способен объединить многих разработчиков на пути достижения мечтаний Азимова.

Автору также представляется, что фреймворк ROS в будущем, благодаря своим достоинствам, может превратиться из мета-операционной системы роботов в операционную систему «умных вещей», которая объединит в себе роботов, умные дома, бытовую технику и т. д. Все возможности для этого в ней уже заложены.

Raspberry Pi Tank

Павел Бондарь, Минск, Беларусь*

Raspberry Pi Tank is an open source project implementing Wi-Fi remote control for a toy tank with Web and console iterfaces. Project includes software tools developed for remote control, such as daemon for controlling tank over GPIO, WebUI for transmitting commands and receiving video, and others. Working prototype is included as a part of presentation.

На сегодняшний день существуют сотни различных применений Raspberry Pi. Он может быть домашним медиацентром, роутером, устройством для автономного скачивания торрентов, а так же многим другим. Благодаря достаточному числу GPIO-портов и компактным размерам он хорошо подходит для взаимодействия с множеством других устройств, идеален для разработки в домашних условиях, а кроме того поддерживает установку практически любого дистрибутива Linux.

Raspberry Pi Tank — ещё одно нестандартное применение для Raspberry Pi, нацеленное на удаленное управление по Wi-Fi автономным роботом с телеметрией и использующее в качестве мобильной платформы массово выпускаемый и потому чрезвычайно дешевый игрушечный танк.

С точки зрения аппаратной составляющей Raspberry Pi Tank включает следующие части:

1. Raspberry Pi.
2. Собственно игрушечный радиоуправляемый танк.
3. Батарея с USB-выходом.
4. Веб-камера.
5. USB-совместимый адаптер Wi-Fi.

В качестве мобильной платформы танк содержит три двигателя: два для независимого вращения гусениц и один для вращения башни. Каждый двигатель может вращаться в прямом и обратном направлении, а потому на каждый двигатель используется по два управляющих контакта. С точки зрения схемотехники управление

*pasha117@gmail.com, http://lvee.org/ru/abstracts/113

двигателем организовано по мостовой схеме. Исходно танк поддерживает управление по радио интерфейсу, но в связке с Raspberry Pi используется прямое подключение выводов GPIO к управляющим контактам на плате.

Радиоуправление активно использовалось нами во время реверс-инжиниринга платы для выявления связи между контактами на плате и их функциональным назначением. В результате была получена распиновка внутренних контактов электроники танка и их функционального назначения. Благодаря тому, что напряжение управляющих сигналов двигателей оказалось 3.3В, не понадобились дополнительные схемотехнические решения для согласования напряжений с GPIO-портами Raspberry Pi (там также используется напряжение 3.3В).

Как упоминалось, для питания Raspberry Pi используется батарея с выходом USB. Такие батареи обычно используются для зарядки телефонов и планшетов и выдают выходной ток до 2 А. Ёмкость использованной в проекте батареи 23 Вт·ч, и при типичном энергопотреблении Raspberry Pi (700 мА) этого должно быть достаточно на 23Вт·ч/(0,7А $\times$ 5В) $\approx$ 6,5 часов работы.

В плане программного обеспечения Raspberry Pi Tank состоит из следующих составных частей: Arch Linux, mjpg-streamer, rpi-gpiod.pl, rpi-keyboard.pl, rpi-tank-rack.

В качестве дистрибутива для Raspberry Pi используется Arch Linux. Благодаря своей легковесности и гибкости, он отлично себя проявил на процессоре ARM с его ограниченными ресурсами. Из полезных особенностей хотелось бы отметить rolling release (что обеспечивает всегда свежие версии пакетов), быстрый менеджер пакетов pacman, асинхронную инициализацию на основе systemd, а также netctl (systemd style network manager).

Для передачи видео используется mjpg-streamer. Изображение передаётся в виде обновляющегося jpg-файла. При таком подходе не используется межкадровое сжатие, и в итоге, при высоком фреймрейте и разрешении, генерируется значительный по размерам видеопоток (3 — 10 Мбит).

Плюсом такого решения является минимизация нагрузки на процессор. Даже в при высоком фреймрейте и разрешении нагрузка на процессор не превышает 20%. Такое решение хорошо подходит для управления по Wi-Fi, но для управления через Интернет это не самый оптимальный вариант.

Ещё одним плюсом стриминга с помощью mjpg-streamer является встроенная поддержка M-JPEG практически всеми известными веб-браузерами (была протестирована возможность стримига видео в браузере Chrome на Android 4 и в Safari на iOS). Это позволяет сделать полноценный веб-интерфейс для удалённого управления и наблюдения практически с любого современного устройства.

Для Raspberry Pi Tank были специально разработаны несколько программ. Для непосредственного управления GPIO-портами Raspberry Pi был написан демон rpi-gpiod.pl, который запускается как сервис systemd при запуске системы. С помощью Perl-модуля Device::BCM2835 команды отправляются на GPIO. Второй задачей демона является прослушивание порта TCP/IP и преобразование полученных команд в соответствующие сигналы для GPIO. Протокол управления является текстовым, в виде, приближенном к CLI-style. Это сделано для того, чтобы упростить разработку различных фронтэндов. В принципе, в качестве фронтэнда может выступать даже telnet: в ответ на команду help выдается список поддерживаемых команд с описанием их назначения (последнее пока реализовано лишь для некоторых команд). Таким образом можно получить информацию о командах, а потом поочередно экспериментировать с каждой их них.

Для непосредственного управления танком с клавиатуры можно использовать rpi-keyboard.pl. Скрипт устанавливает локальное или удалённое TCP/IP-соединение с демоном rpi-gpiod.pl. При запуске в консоли скрипт считывает нажатия клавиш. Управление движением осуществляется клавишами «WASD»: вперёд, влево, вправо, назад. Управление вращением башни осуществляют клавиши «[» и «]» соответственно выполняя поворот влево и вправо.

Веб-интерфейс реализован через rpi-tank-rack (автором которого является Artem Sheremet). Приложение rpi-tank-rack написано на ruby, при старте запускается легковесный веб-сервер rack. Для взаимодействия клиента и сервера используется WebSocket. Управление с веб-страницы осуществляется как с помощью кнопок, так и с помощью клавиатуры по тому же принципу, что и в rpi-keyboard.pl.

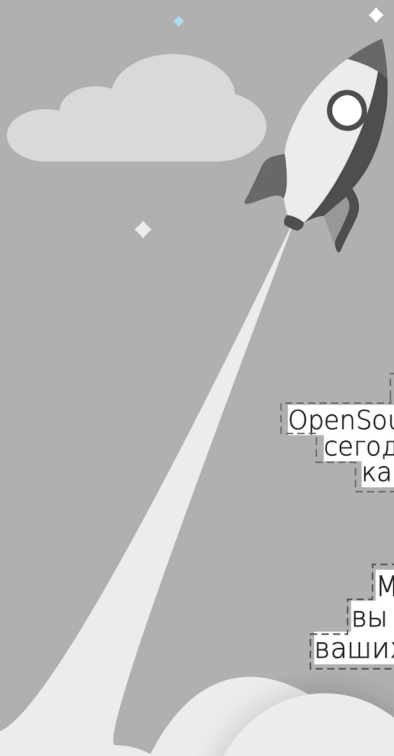
Существует возможность настройки параметров воспроизведения потокового видео (частота кадров, разрешение) прямо из браузера. Весь код проекта располагается на GitHub:

`https://github.com/bondar-pavel/rpi-tank`, `https://github.com/dotdoom/rpi-tank-rack`

MyCloud

Jelastic от MyCloud.by – это облачная хостинговая платформа нового поколения для Java, PHP, Ruby и Python-приложений.

Год назад мы выбрали данную платформу для продвижения в Беларуси и совсем недавно запустили его в эксплуатацию. На базе дата-центра "Datahata" мы развернули инновационный хостинговый сервис, пользующийся широкой популярностью во всем мире, но аналогов которого в Беларуси еще нет.



MyCloud.by всегда стремится к поддержанию OpenSource-инициатив и обмену опытом. Поэтому мы сегодня принимаем участие в конференции LVEE в качестве спонсора и докладчика, и видим в ней перспективное сотрудничество.

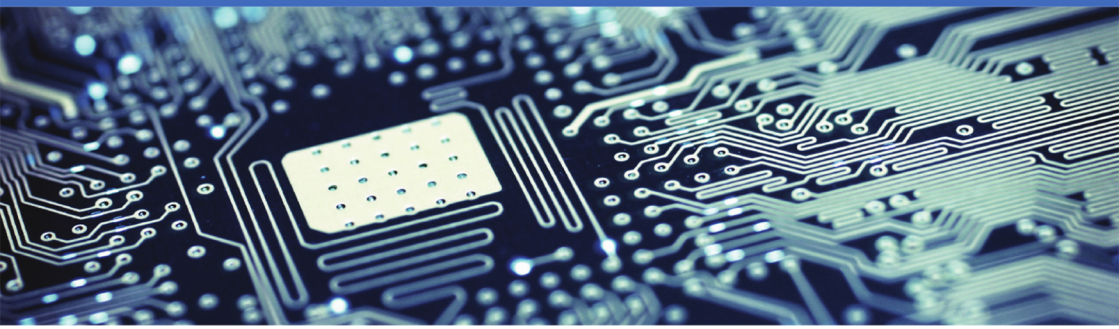
Мы искренне верим, что попробовав Jelastic, вы сможете перевести качество разработки и ваших сервисов на существенно новый уровень.

Подробнее на MyCloud.by

sam solutions
Value Of Talent. Delivered

SaM Solutions GmbH
Am Bahnhof 4a
82205, Gilching
Germany

Tel.: +49 81 057 7890
Fax: +49 81 057 78920
info@sam-solutions.com
www.sam-solutions.com



Profile:

In operation since 1993	50+ active clients
Headquartered in Munich, Germany	600+ employees
Development centers in Eastern Europe	Regional offices in the Netherlands, UK, USA

Experience:

- A broad range of project types involving embedded software development
- Solutions for a variety of platforms and architectures (RISC, CISC, SHARC, x86, ARM, DSP, AVR8/32, PLCPIC, etc.)
- Considerable know-how accumulated over the years, own components, tried and tested code base
- Network protocols experience
- Industry-standard development methodologies (source code/documentation control, change management, defect tracking and continuous integration)
- Cross-platform developent tools
- BSP (adaptation and extension)
- Databases, including small footprint databases
- Porting of kernels, bootloaders, and BSPs to new architectures
- Creation of device drivers
- Cross-architecture migration, e.g. from RTOS's to embedded Linux environment

sam solutions
Value Of Talent. Delivered

SaM Solutions GmbH
Am Bahnhof 4a
82205, Gilching
Germany

Tel.: +49 81 057 7890
Fax: +49 81 057 78920
info@sam-solutions.com
www.sam-solutions.com

Embedded Software Development For:

Network equipment (NAS, routers, Wi-Fi APs, mobile communications equipment)

Consumer electronics (e-books, set-top boxes, GPS navigation systems, media boxes)

Automotive systems (onboard equipment and infotainment systems)

Smartphones and tablets

Medical equipment

POS and warehouse equipment

Experimental and scientific equipment

Development boards and kits

Голос спонсора: Conjur. Authorization in the cloud

Conjur is a cloud-native platform for directory services, authorization, and audit for development and operations teams and their entire infrastructure.

With 100% API coverage and a scalable, easily deployable, high-availability architecture, Conjur reduces the time, cost, and complexity associated with building authorization management via complex home-grown scripts and configuration management tools. Sometimes referred to as "Active Directory for the cloud", Conjur runs in either a virtual machine or container, and works alongside a wide range identity and access management (IAM) solutions to solve access and authorization challenges: machine-to-machine permissions, deployment and access rights to sensitive systems, and the auditing required to meet compliance requirements.

"Conjur is more than just an outstanding platform for managing authorization as a service: they are a partner that we can innovate with, and an essential part of achieving our vision and solving the hard problems that arise as we move our infrastructure into the cloud technologies that will drive our business forward." - Mike Kail, VP of Operations, Netflix

Why Conjur?

Built with system administrators, DevOps professionals, and cloud architects in mind, Conjur natively supports a wide range of potential use cases. Some of the most common reasons why organizations select Conjur over building in-house solutions include:

- **Compliance**: auditable enforcement of both organizational and regulatory policies and rules, via existing reporting systems (e.g., SIEM platforms, SumoLogic, Splunk)
- **Risk Management**: reduction of the attack surface for sensitive data (credentials, SSL/SSH keys and certificates, secrets, etc.) by means of Conjur's policy and governance platform
- **DevOps Optimization**: integration of security & controls in upstream development and operations work, ensuring consistency across all production systems.
- **Access Intelligence**: unified control of identity (human & machine) and permissions across entire infrastructure (bare metal, private, public, cloud) helps prevent failures, without relying on legacy systems for policy governance and enforcement

What is Conjur?

Conjur is designed with two primary goals: (1) to provide an easy-to-use, well documented, extensible authorization system, and (2) to allow for non-disruptive integration into organizational workflows.

This approach has resulted in a series of implementation benefits:

- **Ease of Integration**: Conjur is built with a DevOps optimized UX (CLI-driven), with 100% API access from a variety of languages, and can be deployed as a Linux virtual appliance, or into bare-metal systems
- **No Vendor Lock-In**: Conjur has been used in conjunction with all leading configuration management and DevOps tools (Puppet, Chef, Salt, Docker, etc.)
- **HA Design**: Conjur's system has been designed with high availability and fault tolerance, with a distributed configuration architecture and full fail-over/redundancy capabilities
- **Minimal Dependencies**: Conjur is self-contained, requiring no external database or server
- **Secure**: All communication is fully encrypted, following industry best practices for managing data in use and at rest; additionally, Conjur has invested in a rigorous third-party security analysis (results expected to be published Q4 FY14)

Learn more at `http://conjur.net`

МЫ - АРМИЯ WARGAMING

Мы – компания, прорубившая окно к мировому успеху на рынке многопользовательских игр. Наши продукты часто называют легендарными, а достижения уже причислены к рекордам. За 15 лет нам удалось заставить солнце не заходить над нашими офисами – представительства компании раскинулись от Сиднея до Сиэтла. Мы предлагаем вам присоединиться к команде, нацеленной на достижение новых побед, и стать неотъемлемой частью будущего Wargaming!

МЫ ГАРАНТИРУЕМ

- работу над сложными интересными проектами;
- комфортные условия, гибкий рабочий график;
- конкурентную заработную плату, премии, годовые бонусы;
- емкий социальный пакет.

СВЯЗАТЬСЯ СО ШТАБОМ

www.wargaming.com/ru/careers/
job_Minsk@wargaming.net

WARGAMING.NET
LET'S BATTLE

Голос спонсора: EPAM Systems

Отдел разработки встраиваемого программного обеспечения EPAM Systems занимается разработкой сложных компьютерных систем в следующих областях: IT, медицина, автомобильная промышленность, бытовая техника, строительство, спорт, индустрия развлечений и многих других.

В последние годы EPAM активно участвует в различных открытых проектах, как с целью решения задач для наших заказчиков, так и с целью развития Open Source в целом и помощи сообществу: BigData: Trends Analyzer and Lambda framework, OpenFlow for managing SDN topologies (EPAM участвует в SDN community, наряду с Cisco, IBM, Juniper и другими), драйвер датчика для PI4J, проекты Yocto, Nfstrace, Libdt, Procedural generation for simulations и множество других.

Наша команда **Epam LLPD** включает (но не ограничивается) в себя экспертов в следующих областях:

1. BigData, включая Storm, Cassandra, Hadoop and Hbase;
2. Разработка встроенного программного обеспечения;
3. Настройка/доработка и оптимизация Linux-платформ, разработка и программирование различных устройств (appliances).
4. Широкий спектр возможностей автоматизации компьютерных сетей;
5. Распределенные системы хранения больших объемов данных;
6. Интернет Вещей (Internet of Things).

Software Outsourcing Company

www.itspartner.net

«ИТС Партнер» (ITS Partner) - белорусская компания, резидент Парка высоких технологий (ПВТ), занимается разработкой встроенного (embedded) и web - ориентированного программного обеспечения и облачных решений.

ИТС Партнер имеет ключевую компетенцию в реализации облачных решений в сфере разработки ПО для систем хранения данных (Data Storage Systems). И предлагает свои услуги аутсорсинга в этой сфере, а также в разработке web-ориентированного ПО, приложений для мобильных устройств.
Наша компания участвует в совместных проектах с такими заказчиками мирового уровня, как Netgear (США), ChronoTrack Systems (США), Codetto (Сингапур).

Компания создана в 2010 году, но за это время реализовано более 20 проектов в различных областях IT-технологий. Наиболее интересные из них:

- Ready NAS Remote - клиент-серверное приложение для создания виртуальных сетей и приложение iOS и Android(Netgear, США)
- Ready SHARE Cloud - клиент-серверное firmware приложение для удалённого доступа и управления маршрутизатором (Netgear, США)
- социальная сеть для спортсменов Athlinks.com (ChronoTrack Systems, США)
- ARLO – система домашнего видео-наблюдения и мониторинга. (Netgear, США)

Кроме того, большое внимание уделяется созданию и развитию собственных продуктов, таких как EasyWifi и .Cash на базе использования QR-кода и др.

В компании работает 25 разработчиков, специалистов в различных областях IT. Как динамично-развивающаяся компания, ИТС Партнер постоянно расширяет штат своих сотрудников. К 2015 году мы планируем привлечь еще как минимум 25 человек. Ищем активных и думающих ребят, как опытных, так и начинающих (студентов)! Мы предоставляем комфортные условия работы и совместного корпоративного отдыха, различные бонусы, возможность карьерного роста и развития.

Присоединяйтесь к нашей дружной команде! У вас будет возможность работать как над проектами мирового уровня, так и над собственными разработками компании! Список вакансий вы можете найти на нашем сайте.

Интервью с участниками

По традиции в сборник материалов входят интервью, взятые представителями оргкомитета во время зимней сессии конференции. Датой LVEE Winter 2014 оказался день 14 февраля, и это подвигло организаторов на романтическую идею: выбрать из программы конференции трех представительниц прекрасной половины человечества и узнать их мнение о свободном ПО, роли и месте GNU/Linux, и собственно о LVEE.

1 Ирина Шубина — senior software developer, EPAM Systems, Минск, Беларусь

LVEE: Скажи, помнишь ли ты тот момент, когда ты первый раз услышала о свободном программном обеспечении?
И: Услышать ... даже не так: я его первый раз даже увидела. Я о нем не слышала до этого. Для меня это было естественно новостью. После третьего курса я искала работу — очень долго, целое лето, активно.
L: А курс это...
И: БГУ. Гуманитарный факультет. Специальность «Информатика, web-дизайн и компьютерная графика» на тот момент. И вот я искала работу, после того как прошла курсы тестировщиков — меня почему-то все лето не брали в тестировщики, но зато в конце лета взяли в программисты.
L: На чем писать?
И: Да в общем-то я и не знала, на чем :) Как оказалось, на C++ и Python.
L: «Потом я узнала, что это C++, C и Python...»
И: Да, потом я узнала. А еще, когда я в первый раз пришла на работу, меня посадили за компьютер, там был двойной bootloader, и меня сразу загрузили в Linux и сказали: «Знакомься, это Archlinux». Да, как-то так... Я посмотрела и сказала: «О...».
L: Это была любовь с первого взгляда? Или не была?
И: Ну, на тот момент я еще два года работала и в Windows, и в Linux, а позже перешла полностью на Linux.

L: А почему у вас, собственно, Linux-то использовался? Особенно Archlinux — не очень простой в настройке дистрибутив, и не самый стабильный, из-за постоянного обновления пакетов.

И: Archlinux использовался собственно на моем только компьютере...

L: Кто-то очень тебя любил :)

И: На тот момент я, как вы сами понимаете, не могла его установить с нуля — это же Arch!

L: Да, тогда у него еще был инсталлятор, но все равно книжка по установке была большая :)

И: Вот. Книжку по установке я тоже никогда не видела — до сих пор...

L: А она есть!

И: Верю :)

L: Хорошо. И значит, Archlinux запустился, открылся какой-то Gnome или KDE... И какие были первые впечатления?

И: Он был на KDE. Мне сказали: «Вот здесь есть тоже большая кнопка. Она, правда, вверху... И нужно вот нажать вот здесь, и будет тебе меню».

L: Поскольку девушка-дизайнер собиралась работать программистом, наверное это было не самое большое потрясение на общем фоне.

И: Ну, я не собиралась работать дизайнером, это точно, поскольку я искала работу тестировщика. Но в общем-то мечтая, где-то там на бэкграунде, работать программистом — но думала, что надо начинать с чего-то попроще, попроще...

L: По поводу Linux. Потихонечку ты начала им пользоваться больше...

И: Да, потихонечку. В итоге даже установила на домашнем компьютере. Пришел момент, когда я решила снести свою XP...

L: А что повлияло?

И: Не знаю, в какой-то момент захотелось, чтобы это было стилем жизни. Вот, наверное так.

L: Кто-то из домашних пугался?

И: Да у меня домашние до сих пор пугаются. Это основной момент, когда меня что-то спрашивают и сильно донимают этим, я всем угрожаю, что я приеду и установлю Ubuntu, страшную и ужасную.

Да, это главная угроза — по отношению, по крайней мере, к младшей сестре.

L: Все-таки не Archlinux?

И: Нет, домашний все-таки Ubuntu, на данный момент основной Linux на работе это Fedora у меня...

L: А дома Ubuntu — как у Дональда Кнута...

И: Только не с конфигурационным файлом 30-летней давности :)

L: Хорошо. Как в последнее время, на твой взгляд, меняется отношение к Linux и вообще к свободному программному обеспечению? Заметно что-нибудь?

И: На самом деле очень заметно. Очень заметна положительная тенденция, потому что становится больше обычных простых пользователей.

L: Не разработчиков.

И: Да, не разработчиков, которые не знают о разработке вообще ничего. То есть у меня есть знакомая, которая работает в магазине продавщицей, и не знаю как у нее появился Linux...

L: Где она его взяла...

И: Где она его взяла, да, но откуда-то он у нее появился, и мне, как специалисту, задавали некоторые вопросы по чистому банальному пользованию.

L: Это здорово.

И: Это здорово, это приятно, и это хорошо.

2 Александра Кононова — доцент кафедры информатики и программного обеспечения вычислительных систем, МИЭТ, г. Зеленоград, Москва, РФ

Александра Кононова: Я по отношению к свободному ПО скорее пользователь, для начала. Хотя мне приходится иногда подгонять что-то под свои нужды...

L: Ты говоришь, что ты пользователь. А что побуждает тебя приезжать на LVEE (уже второй раз), и выступать с докладами?

А: Любопытство. А кроме того, мне белая майка не к лицу, а у вас, вот, докладчикам выдают черные :)

L: Хороший стимул :) Так. Ты помнишь тот момент, когда ты впервые услышала про свободное программное обеспечение, или про Linux, или еще про что-то в этом роде?

A: К сожалению, память у меня достаточно скверная :) Поэтому когда первый раз — не знаю...

L: Но если примерно?

A: Может, когда в своё время на малом мехмате МГУ я увидела, как верстался математический журнал... В TEX, кажется, и да, под Linux. И меня поразило, насколько это было красиво.

L: Что-то изменилось в твоей жизни после этого? Linux, во всяком случае, теперь приходится пользоваться, да?

A: Ну не то, чтобы приходится. Оказалось, что это удобно и действительно красиво. Поэтому немножко не то слово выбрано.

L: То есть это собственное желание, которое удается претворять в жизнь.

A: Ну поскольку эта система — логичная и интуитивно-понятная, и к которой можно подобрать удобный интерфейс, а не унифицированный для среднестатистического пользователя, то да, безусловно.

L: А в учебном процессе появляется какое-то свободное программное обеспечение у вас?

A: Поскольку на ВЦ стоит во всех аудиториях Windows официально, поскольку однотипное ПО проще администрировать — то только прикладное: Octave, Scilab, среды разработки...

L: То есть много математического ПО?

A: Да... А еще OpenOffice, как же без него.

L: А студенты различают вообще как-то: «Вот это свободная программа, а это — несвободная, но мы ей пользуемся»?

A: Студенты все разные. Но надо признать, что большинство на вопрос о лицензиях отвечают «Плевали мы на эту лицензию, все равно есть все взломанное».

L: То есть борьба за чистоту лицензий это все-таки, до конечного пользователя когда доходит, носит довольно условный характер, да? В принципе, российские вузы — они пока не очень сильно озабочены этим вопросом, или в вузе все-таки приходится...

A: Это, честно говоря, зависит от ВЦ. Потому что где-то программное обеспечение если не свободное, то купленное, а где-то — обычно это дорогостоящие мощные комплексы — где-то оно... разное :)

L: По-настоящему тактичный ответ. А тот же Linux удается использовать или на собственном компьютере, или на кафедральных как-то?

A: На собственных и кафедральных безусловно стоит у нескольких человек: у кого-то как учебное пособие, а у кого-то — как основная система. Потом есть отдельный учебный курс, «Основы Unix», и курс «Операционные системы», но там используется виртуальная машина, что, наверное, не улучшает впечатление, и в виртуальной машине он: Redhat, Ubuntu.

L: Получается, что преподавательский состав его использует активнее, чем студенты?

A: Опять-таки, и студенты и преподаватели разные, поэтому многие используют, а многие нет.

L: Если соотношение попытаться вывести — где можно чаще услышать об этом, от студента и ли от преподавателя?

A: Слово — ни от кого: кто пользуется, тот пользуется и молчит :)

L: Значит, персональное личное дело каждого. Как вегетарианство :)

A: Но есть ресурс `unix.miet.ru`, где находятся зеркала основных дистрибутивов Linux, BSD. Так что, судя по всему, эта тема хоть и не проговаривается, но поддерживается.

L: То есть, судя по всему, ваш вычислительный центр как раз достаточно активно его использует. Наверняка же ВЦ администрирует эти зеркала. А кстати, по сложности эксплуатации свободных программ какая-то разница ощущается?

A: Проще.

L: Проще?

A: Во-первых, установка проще, и не требуется искать... лицензию, и отвечать на многие кнопочки. А во-вторых, поскольку свободное ПО пишут в основном энтузиасты и для себя, то можно обычно найти программу, которая соответствует твоей логике.

L: И последний вопрос, традиционно: как ты вообще узнала про LVEE?

A: В основном из чувства противоречия. Ибо нам на кафедру сыплются конференции от Microsoft — есть рассылка, и в ней периодически спрашивают: не хотите поучаствовать? Ну и поскольку возник естественный вопрос, а нет ли конференций...

L: ... где все наоборот...

A: Да, где можно обсудить свободное ПО, то стали искать и нашли несколько завершившихся. Через стандартный поиск, google. И эта показалась наиболее вменяемой по времени — когда надо было ждать не год, а несколько месяцев.

L: И как впечатление?

A: Очень приятное.

L: На что это вообще похоже? LVEE все-таки во многом отличается от академических конференций...

A: На капустник. Но весело :) А доклады — ну, я не знаю, с чем их сравнить... На доклады. Можно узнать много новых слов. Много интересного.

L: Спасибо :)

A: Они, наверное, более понятны, чем многие академические, здесь больше практики. И видно, что человек этим занимается, а не просто наскреб на доклад.

3 Ольга Карабутова — senior software engineer, R&D, EPAM Systems, Минск, Беларусь

L: Ты сегодня выступала с докладом. Расскажи об ощущениях — как это страшно, как это волнительно, какие чувства приносит такое публичное выступление?

Ольга Карабутова: Честно говоря, первое что я хочу сказать — что это было мое первое публичное выступление.

L: Кстати, у тебя очень хорошо получилось.

O: Ура. К примеру, магистерскую я не смогла нормально защитить, потому что растерялась, потому что чувствовала, что я что-то не доделала... Диплом у меня был чем-то, что я действительно сделала — аппаратный проект, уже внедренное, работающее устройство. Да, с таким проектом я звезда была по сравнению со всеми остальными, мальчиками, девочками, дневниками, вечерниками. А магистерская была дана преподавателем, магистратура у меня была заочная, все это было недоработано...

L: По специальности?..

O: Радиотехника. Включая радиолокацию, радионавигацию и телевидение. В общем, магистратура была заочной, и я чувствовала, что недоработана тема. Но поблажки мне за это не было... С тех

пор я успела сменить много работ. Сейчас я работаю в EPAM. Там замечательно :)

L: Что не устраивало в предыдущих работах?

O: Ну если взять всю мою эволюцию, как работника, то бывало что что-то серьезно не устраивало, но с последнего места работы я ушла не потому, что что-то не устраивало — было некоторое стечение обстоятельств, и здесь мне предложили и более интересный проект, и более денежный, но в первую очередь все-таки более интересный. И я, как человек общительный, уже знала достаточно много людей, с которыми я теперь работаю... В том числе, благодаря LVEE.

L: А кстати, как ты на LVEE попала впервые?

O: Мне его рекламировали несколько человек. То есть фактически, это была внутрикорпоративная среда, где мне знакомые разработчики рассказывали, что есть такая замечательная конференция. Это было в 2009 или в 2010 году.

L: Это была просто судьба. И какое было первое впечатление?

O: Во-первых, очень интересные доклады. Опять же, формат летнего LVEE располагает к отдыху, общению и новым профессиональным знакомствам и не только, к дружеским, в том числе. То есть я увидела большую-большую сборную из разных интересных людей, и теперь я постоянный посетитель.

L: Может быть это как-то повлияло на карьеру?

O: Конечно же повлияло. Дело в том, что каждый раз, когда вы идете к какому-то изменению, есть много факторов. И нельзя сказать, что первое: мой интерес к Linux, и поэтому LVEE, или LVEE и поэтому мой усилившийся интерес к Linux.

L: А вообще, первое знакомство с Linux — оно как произошло? Это наш вечный вопрос.

O: Одно дело личное знакомство, а другое — почти знакомство. Услышала я в университете, от одногруппников, и даже когда-то, когда я попросила помощи, человек мне сразу поставил и Linux, и Windows на одну систему... Поставил и сказал: Linux — резервная система, если какие-то будут неполадки, то через Linux можно решить. И был еще Linux на работе, после, я тогда работала в публичных библиотеках города Минска.

L: А в публичных библиотеках города Минска есть Linux?

O: Да он давным-давно был. И я думаю, что не только в публичных библиотеках города Минска. Но в библиотеке я им не занималась — там была отдельная девушка выделена, системный администратор,

которая занималась Linux. В общем, эта тема просто была на слуху. А ближе познакомилась я чуть позже. Не помню точный момент...

L: То есть это просто система, которую ты выбрала для себя за какие-то ее достоинства?

O: Да, для себя, для тех экспериментов, которые можно проводить дома — с этим же интересно поиграться. Ну конечно же, я работала на Linux на работе, но я не помню, чтобы это было требованием. Где-то были встраиваемые системы, где-то еще что-то, т.е. с Linux у меня есть опыт работы, но прямо сказать, что я работала разработчиком под Linux — такого, кажется, не было.

L: А какое-то такое ощущение, что свободное программное обеспечение — это есть какой-то отдельной разновидность или отдельная группа программ, чем-то отличается...

O: Есть же своя философия Linux, это известный факт, и это красиво и стройно. Я вообще склонна к философии, поэтому это мне особенно понравилось. В этом есть какая-то гармония. Мне, как инженеру по образованию, вполне понятно, что система может быть красивой, устройство может быть красивым, и обычно это говорит о его лаконичности, эффективности, без всего лишнего. И в этом плане я очень быстро разочаровалась в Windows, как только попробовала Linux. Ну, как есть так есть, что тут сделаешь :) Вот так :)

Научное издание

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов десятой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2014

(Гродно, 22–24 августа 2014 г.)

Фото на обложке Арсения Случевского

Ответственный редактор	Д.А. Костюк
Компьютерная верстка	А.О. Шадура
	А.Ю. Гузик

Подписано в печать 26.08.2014.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная.
Ризография. Усл. печ. л. 10,3. Уч.-изд. л. 6,5.
Тираж 220 экз. Заказ 2391.

Издатель и полиграфическое исполнение:
частное производственно-торговое унитарное предприятие
«Издательство “Альтернатива”».
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий
№ 1/193 от 19.02.2014.
№ 2/47 от 20.02.2014.
Пр. Машерова, 75/1, к. 312, 224013, Брест.