Открытые технологии

Материалы одиннадцатой международной конференции Linux Vacation / Eastern Europe 2015

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов одиннадцатой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2015

(г. Гродно, 25–28 июня 2015 г.)

Брест
«Альтернатива»
2015

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 государствен-ного университета информатики и радиоэлектроники
В.А. 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 одиннадцатой международной конференции разработчиков и пользователей сво-бодного программного обеспечения Linux Vacation / Eastern Europe 2015, Гродно, 25–28 июня 2015 г. / под общей ред. Д.А. Костюка. – Брест : Альтернатива. – 120 с.

ISBN 978-985-521-495-4.

В сборник вошли материалы, представленные авторами на одиннадцатую международную конференцию разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2015. 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 платформ – от рабочих станций и серверов, включая решения на базе виртуализации, до встраиваемых систем и мобильных устройств. Тематические направления включают разработку программного обеспечения, администрирование систем, построенных на основе свободных программ, правовые и организационные особенности их использования, создание аппаратного обеспечения под свободными лицензиями. Сборник 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 978-985-521-495-4

Содержание

Надёжные ссылки

Юрий Андреев, Санкт-Петербург, РФ*

The rights to study and modify free software imply access to its source code. The rights to verify scientific result imply access to its background and previous results. That could be a problem if an access was given by a (url-)link. The problem will be considered in the present article.

Обзор

Свободное Программное Обеспечение (СПО) предполагает доступность исходного кода. Это отражает научный подход разработки ПО, при котором любой результат должен быть верифицируемым и открытым для критики. В этом смысле программный код является аналогом научной публикации. Что же касается самих научных публикаций и публикации на тему СПО, то они предполагают доступность своих источников *a priori*. Однако, с появлением сети Интернет, источники всё чаще указываются в виде URL-ссылок, которые могут стать недоступными чисто технически. И было бы обидно «засекретить» результат работы таким образом.

Каждый день в глобальной сети появляются миллионы новых ссылок. И каждый день миллионы ссылок становятся битыми — то есть указывающими на недоступный или на неверный ресурс. Эти изменения — простое следствие динамичности сети. И наличие таких временных окон доступности ресурсов нужно учитывать при проектировании ПО, подборке используемых библиотек, внедрении удаленных сервисов и контента; также тема битых ссылок постоянно обсуждается в связи с вопросами поисковой оптимизации (SEO). Учитываться это может совершенно по разному. К примеру, появился «социальный» сервис, `snapchat.com`, принципиально не сохраняющий контент и ссылок на него — и сразу нашел себе пользователя. Нас будет интересовать противоположная ситуация, и ситуация эта будет рассмотрена в контексте научных и научно-технических публикаций. Действительно, в этом случае желательно, чтобы ссылки указывали на верный материал когда бы статья

*andreev.yurij@gmail.com, http://lvee.org/ru/abstracts/150

ни была прочитана. Потеря же ссылки на предыдущий результат в научной публикации ставит под сомнение верность текущего результата (тут можно вспомнить про Великую Теорему Ферма, которая более трёхсот лет оставалась гипотезой, из-за «недоступности» оригинального доказательства /если оно было/).

А ссылки появляются везде. Автор лично присутствовал на серьезной математической конференции, где один из докладчиков в своем выступлении приводил переписку из ночного чата.

Доступность материалов в библиотеке, вообще говоря, кажется более стабильной, чем в интернете. В библиотеке найдется всё, если... к примеру, не сгорит (однажды в Библиотеке Академии Наук [2] автору довелось читать статью из сборника, который, судя по надписям и штампам, сначала тонул в наводнении 1824 года, а потом горел в пожаре 1988 года). Более того, существуют системы обязательного хранения. С момента своего основания, Национальная Библиотека Беларуси стала получать два обязательных бесплатных экземпляра белорусских изданий и один обязательный бесплатный экземпляр изданий, выходивших в СССР (РФ) до марта 1995 г, библиотека также является депозитарием материалов ООН и других международных организаций [3]. То же можно сказать про Российскую Государственную Библиотеку [4]. Важно понять, что найдя нужную книгу, мы сразу можем начать читать, то есть получаем доступ к её содержанию.

С появлением компьютеров возник вопрос физической доступности — нельзя считывать информацию напрямую. А носители информации меняются очень быстро, и с магнитной пленки, да и с дискеты, информацию теперь прочитать будет крайне затруднительно [1]. С появлением интернета вопрос сохранения копированием стал менее актуален. Теперь вопрос сохранности информации предстает в другом разрезе — в проблеме контроля за её доступностью.

Терминология

Уточним терминологию. Мы будем рассматривать URL-*ссылки*, но в более общем смысле — как ссылки на внешние, по отношению к материалу, ресурсы (ссылки в тексте, из разделов «ссылки», «биб-

[1] Однако, развитие идёт по спирали — утрату большой части античного письменного наследия связывают также со сменой технологий, так как множество текстов никогда не переписывалось с папирусов на пергаментные кодексы [5].

лиография», «литература»). Ссылка указывает на ресурс. *Ресурс* — это может быть какой-либо контент, сервис, или печатная статья, книга. *Надёжность ссылки* — способность указывать на верный, доступный ресурс.

Классификация

Классифицируем ссылки по степени надёжности (с допуском на то, что это достаточно индивидуальный вопрос, особенно для внутренних ссылок). Перечислим возможные варианты по степени убывания надёжности.

- Ссылки на части в оригинальном материале кажутся надёжными по определению, если это не документ, страницы из которого могут «выпасть».
- Разумеется, стабильнее всего ссылки, принадлежащие организациям, отвечающим за их распределение и обслуживающим интернет: `icann.org, w3.org, example.com,...`
- Сайты сообществ, таких как `gnu.org, ctan.org`, специальные архивы `arxiv.org`, университетские сайты, `<site>.edu`
- Правительственные сайты, сайты больших компаний (правда бывает... в 1999 году компания Microsoft забыла продлить домен `passport.com`, что привело к недоступности Hotmail для многих пользователей; видимо, этот случай не послужил Microsoft уроком, поскольку в 2003 году компания снова забыла продлить важный домен, на этот раз `hotmail.co.uk`[1]).
- Нужно учитывать различные характеристики: есть ли архив у новостного сайта, сколько подписчиков и форков у проекта на github и т. п. Вообще говоря, доступность материала по ссылке — это также вопрос доступности сервиса, на котором этот материал расположен. Одно дело, если проект расположен на github, другое дело — на каком-либо частном ресурсе, который может закрыться. Пример — закрывшаяся почта @london.com (кстати, там был и платный сервис). Автор, указавший ее в контактах, станет недоступен.
- При ссылке на Википедию нужно учитывать, что материал может измениться. Понятно, что это не всегда желательно. При ссылке на github обычно важнее само наличие контента, и в большинстве случаев он, наоборот, должен изменяться.
- Блоги. Иногда достаточно постоянные, зависит от автора. Социальные сервисы в этом плане намного динамичнее, доступ-

ность гораздо хуже (не говоря о том, что доступность может просто регулироваться закрытостью для незарегистрированных пользователей). На отдельном авторском блоге материалы могут представлять большую ценность для автора, он заинтересован в сохранении доступности. Хотя всегда есть вероятность, что имя не будет продлено.

Графически степень надёжности ссылок в пределах одного сайта можно изобразить в виде треугольной диаграммы (рис. 1). Обычно, над корнем находятся основные ссылки (1), выше — контент сайта (2), еще выше — пользовательский контент (3).

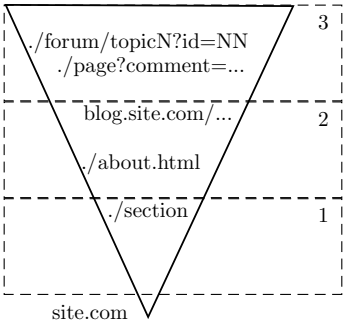


рис. 1

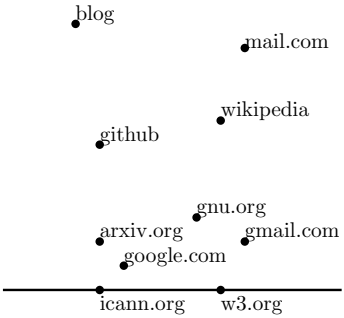


рис. 2

Выделим «центр тяжести» и составим общую диаграмму надёжности, соответствующую нашей классификации (рис. 2). Самые надёжные ссылки располагаются внизу. Ссылки с верхних слоёв легко сдуваются ветром перемен.

Структура ссылок и алгоритм поиска

При выборе ссылки важно учитывать её структуру. Вкратце остановимся на этом. Рассмотрим ссылку вида:

`http://<site-name>.tld/<section>/<page-name>?<query>`

Если ссылка становится недоступной, то, куда мы попадём, зависит от организации редиректа (перенаправления) на сайте. Возможные варианты: это будет страница с ошибкой, главная станица или страница вышележащего раздела; лучший вариант — редирект на актуальный адрес. Обычный алгоритм поиска таков:

1. Ищем информацию на доступных ссылках сайта (двигаясь вниз по рис.1)
   - `http://<site-name>.tld/<section>/<page-name>`
   - `http://<site-name>.tld/<section>/`
   - `http://<site-name>.tld`
2. Через «поиск»
   - поиск `<section>, <page-name>` по сайту, если возможно
   - google `<site-name>, <section>, <page-name>`
   - поиск исходя из контекста, в котором ссылка была указана.
3. Поиск «следов» информации в сети: сервис `archive.org`, кэши поисковых систем [2].

Плохо, если `<site-name>, <section>, <page-name>` не несут информации (это, кстати, правило SEO).

К примеру, `http://www.red-bean.com/kfogel/948.html` [3] (заметьте, данный пример остается актуальным в рамках этой статьи даже если ссылка станет недоступной). Поэтому следует приводить такие ссылки с поясняющей информацией.

Примеры ссылок из сборника

Приведем несколько интересных ссылок из сборника [5]:
- классическая ссылка: The GNU Project Debugger, `http://www.gnu.org/software/gdb/`
- ссылка с датой просмотра: `http://www.blendernation.com/` Дата просмотра: 20.04.2014
- сокращенная ссылка: `http://bit.ly/1sk1bZt` Сервис bit.ly позволяет сокращать ссылки, делая из длинной непонятной короткую непонятную. Хранит он ссылки вечно, но не стоит забывать, что это всего лишь ссылка на ссылку. Она увеличивает читабельность, но не надёжность.

Правила подбора ссылок

Сформулируем правила подбора надёжных ссылок.
- Иногда следует добавлять информацию к ссылке, она может помочь, как в поиске информации, так и в понимании её актуальности.

[2] Отдельная тема. Наличие в кэше зависит от частоты индексации, настроек robots.txt, самой поисковой системы и др. Время снимков `archive.org` не связано с изменениями на сайте, о чём они сразу честно предупреждают.

[3] На самом деле, это контрпример, там написано как раз об этом.

– Иногда следует включить материал в саму статью. К примеру, когда материал — это чья-та цитата, размещенная на блоге (в истории про Microsoft, описанной выше, непонятно, какую ссылку давать, но «гуглится» хорошо).
– Из различных представлений одной ссылки выбираем более читабельное. Для нечитабельных можно воспользоваться сервисами преобразования ссылок, при условии надёжности последних. Иногда лучше давать *структурную* ссылку (см. пример [3]).
– Если есть выбор из нескольких ссылок, указывающих на один ресурс, выбираем наиболее надёжную.
– Если есть выбор между ссылкой на печатное издание или электронное, выбираем обе.
– Нужно исходить из актуальности основного материала и здравого смысла. Однако, значимость материала может недооцениваться автором, поэтому, если есть возможность, лучше давать более надёжную ссылку.
– Хорошо само существование раздела ссылок. Сам по себе, этот раздел помогает понимать общий контекст материала в предметной области.

Литература

[1] http://google.com
[2] https://ru.wikipedia.org/wiki/Библиотека_Российской_ академии_наук
[3] Национальная Библиотека Беларуси, http://nlb.by, главная → информационные ресурсы → фонды и коллекции библиотеки
[4] https://ru.wikipedia.org/wiki/Российская_государственная_библиотека, см. также федеральный закон РФ «Об обязательном экземпляре документов» от 29.12.1994 № 77-ФЗ
[5] https://ru.wikipedia.org/wiki/Книжные_утраты_в_Поздней_античности_и_«Тёмных_веках»
[6] Материалы международной конференции Linux Vacation / Eastern Europe (LVEE 2014), http://lvee.org/ru/reports/materials_lvee_2014

Гибкая графика LaTeX

Юрий Андреев, Санкт-Петербург, РФ[§]

Classification of the different types of graphics elements will be given. Mixed methods to work with graphics within LaTeX system are examined. A brief overview of the appropriate tools and tips are given as well.

Введение

Система вёрстки LaTeX[1] теоретически обладает неограниченными возможностями по набору сложных документов. Подключение соответствующих пакетов позволяет отображать такие сложные объекты как ноты и структурные химические формулы. Наиболее востребованной является возможность быстрого набора сложных математических формул (типа $\oint_\gamma f(z)dz = 2\pi i \sum_k Res(f, a_k)$ [1] и сложнее). О графике LaTeX можно написать книгу[2], в данной же статье хотелось бы заострить внимание на следующей проблеме.

Графика — область чрезвычайно обширная. Многие графические элементы, типа перечисленных выше, создаются по чётко определённым правилам и поэтому могут быть эффективно описаны на каком-либо языке. С другой стороны, есть элементы, требующие повышенной гибкости редактирования, правила создания которых заранее не известны. И в таких случаях применение программного подхода становится практически [2] невозможным. Далее мы дадим классификацию элементов по типу редактирования и кратко опишем несколько практических приёмов работы с графикой [3] в рамках системы LaTeX.

[§] andreev.yurij@gmail.com, http://lvee.org/ru/abstracts/151

[1] формула вычетов, называется среди самых красивых математических формул. Первое место в подобных опросах обычно занимает тождество Эйлера $e^{i\pi} = -1$.

[2] надо понимать не как оборот речи, а в прямом смысле. Теоретическая возможность остаётся.

[3] описанные далее WYSIWYG (What You See Is What You Get) редакторы являются кроссплатформенными

Графика по правилам и без

1. Генерируемая графика

Когда графический элемент может быть сгенерирован, он обычно описывается на языке одного из прикладных пакетов: R, GNUPlot, Graphviz, ROOT, Maple, Sage и др. Если нам нужно будет использовать полученные результаты в latex-документе, то их всегда можно экспортировать в форматы `ps, pdf` и затем вставить в latex-документ с помощью пакета `graphics`. Этот способ можно назвать *обобщением*, так как экспорт производится в формат более широкого назначения. Но при его использовании теряется информация (читаемость). Поэтому, для возможности последующего редактирования и более удобного взаимодействия, для сохранения исходного кода, для поддержки единой стилистики, пакеты обычно предоставляют способы интеграции с LaTeX – *конвертацию* (преобразование кода одного языка в другой) и *внедрение* (использование кода одного языка в другом).

Это может быть экспорт в latex-формат или преобразование в формат одного из latex-пакетов. Например, для вывода из GNUPlot в latex-формат нужно указать соответствующий обработчик[4] `set term latex` и файл вывода `set output "plot.tex"`. Доступны различные варианты, `set term pstricks`, `set term epslatex` и др.

При использовании `epslatex` возможно внедрение latex надписей внутри gnuplot-файла командой `set label`. Это позволяет получить на выходе (`output`) два файла plot.eps и включающий его plot.tex, который уже при latex компиляции выдаёт графику с точно позиционированными поверх неё latex-надписями.[5]

Напротив, существует практика внедрения результатов работы прикладного пакета в latex-документ с помощью подключения специального latex-пакета. Для GNUPlot есть latex-пакет `gnuplottex`, для Sage – `sagetex`. Также, нельзя не упомянуть про `psfrag` и другие latex-пакеты для работы с EPS-форматом.

В заключении сделаем два замечания. Во-первых, при любом внедрении усложняется процесс компиляции. В случае с `gnuplottex` нужно использовать ключ компиляции: `pdflatex -shell-escape`,

[4] ориг. термин: *terminal*

[5] WYSIWYG-добавление latex-надписей обсуждается в третьей части этой секции.

в случае `sagetex` компиляция осуществляется в два прохода. Во-вторых, ограничиваются возможности внедряемого языка.

2. Когда нужна корректировка. Ограничения против гибкости

Иногда возможно генерировать основу графического элемента, но при этом будет необходимо последующее добавление или корректировка деталей. Также, при создании некоторых элементов, становится трудно искать и редактировать их детали – недостатки текстового набора тут начинают перевешивать преимущества.

В качестве примера можно привести работу с таблицами, которую лучше выполнять в WYSIWYG-редакторе. Потом таблицу можно будет преобразовать в latex-формат напрямую, с помощью соответствующего конвертера, или через формат с разделителями значений (его можно обработать latex-пакетом `pgfplotstable`).

Ещё одним примером является пакет `grafviz` (язык `dot`). Он работает по *принципу ограничивающих условий*, когда элементы описываются рядом ограничений по характеристикам, а реальные значения этих характеристик вычисляются в момент генерации. Поэтому полученный результат может нуждаться в корректировке [3]. Пакет используется для отображения различных зависимостей (в программном обеспечении от Puppet до Trac).

Надо заметить, что для пакета возможны те же пути внедрения, что и описанные в предыдущем разделе. Если мы для dot-файла применим конвертер dot2tex с опцией math: `dot2tex -tmath -autosize example.dot > example.tex`, то latex-выражения, написанные в dot-файле будут корректно обработаны. Будет получен tex-файл (с хорошо поясняющими комментариями) в pstricks- и tikz-формате. (Наряду с latex-выражениями, в dot-файле можно передавать pstricks- и tikz-инструкции внутри поддерживаемого dot атрибута style.) Если же мы подсоединим один из latex-пакетов `dot2texi`, `graphviz`, то сможем изъясняться на языке dot внутри latex-документа (не забываем про ключ `pdflatex -shell-escape` при компиляции).

С непосредственным WYSIWYG-редактированием dot-файлов есть проблемы, так как происходит значительная перекомпоновка при внесении практически любого изменения. Для получение более подробной информации о компоновке, можно получить расширенный dot-вывод, применив конвертер dot2tex. Также, можно

переконвертировать dot-файл в файл svg или в файл графических пакетов pstricks, tikz и редактировать его уже средствами из следующего раздела.

3. WYSIWYG подход

Если примером задачи из первого раздела могла бы быть задача построения параболы, когда нужно отдать на выполнение только её формулу и, возможно, границы области вывода, примером для второго раздела могла бы быть кривая Безье, вычисление которой происходит на основе опорных точек, координаты которых и нужно сообщить (и делать это лучше в графическом редакторе), то, задача этого раздела – построение произвольного графика. Это полностью WYSIWYG-задача.

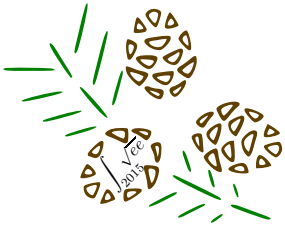
Перечислим несколько редакторов, от общих к специализированным, слева на право: Inkscape, GeoGebra, TexMacs, LatexDraw, jPicEdt[4]. Экспортировать редактируемое изображение в формат latex-пакета `pstricks` умеет такой известный редактор векторной графики, как Inkscape. Другие, помимо `pstricks`, умеют экспортировать в форматы latex-пакетов, таких как `tikz`, `eepic` или в обычный `latex`. (Не очевидно, но чтобы сохранить файл в нужном формате в jPicEdt нужно прежде определить для документа тип: Edit → Content Type. Тут необходимо сказать ещё про установку jPicEdt. Возможно, нужно будет заменить `javaw` на `java` в файле запуска: `sudo nano /usr/bin/jpicedt`.) Поэтому созданный в программе векторный рисунок можно без труда преобразовать в нужный формат и использовать в latex-документе.

Сложности начинаются, когда необходимо редактировать код одного из latex-пакетов. Внутреннем форматом редакторов являются форматы на основе XML, и на данный момент нет полноценной реализации импорта latex-кода. Казалось бы, что импорт реализован в редакторе jPicEdt, но редактор сохраняет в комментариях свой формат внутри сгенерированного latex-файла и, видимо, использует его при импорте. (Поддержка импорта произвольного кода заявлена на сайте программы[4], выдаёт ошибки парсинга) Это полезный приём, которым можно пользоваться, для того, чтобы объединять весь необходимый для редактирования код в одном latex-документе. Для пользователей Emacs плагин Auctex предоставляет удобные функции: комментирование и обратное ему действие над выделенными строками реализовано в виде комбинации

`C-c ;` подобные действия над параграфом осуществляет комбинация `C-c %` Что касается LatexDraw, то в нем есть функция вставки pstricks-кода. Но пока LatexDraw распознаёт подмножество из тех pstricks-команд, которые умеет генерировать сам.

Разумеется, возможности у редакторов различны, как в области обычной графики, так и в поддержке latex-кода. Редактор GeoGebra отображает формулы сразу в стиле `displaystyle` (которым отображаются формулы на отдельной строке), в LatexDraw эту директиву нужно включать в текст. Возможность поворачивать latex-изображение в GeoGebra осуществляется в командной строке редактора, командой `RotateText`, в то время как в LatexDraw его можно поворачивать как обычный векторный элемент.

Также дело усложняется, если нам нужно вставить в редактируемый документ изображение из другого файла. Это не всегда возможно, а если возможно, то при дальнейшем экспорте изображение будет включено в код как отдельный файл, или не включено вообще. Если нас устраивает eps-файл, то тут стоит опять напомнить про пакеты типа `psfrag`. Выполнить графику одним pstricks-кодом всё же можно, но стандартной задачей это не является:



Выводы

В рамках системы LaTeX возможно совместное использование текстового формата и WYSIWYG-управления, что позволяет добиться необходимой гибкости в редактировании графических элементов разной природы. Такая гибкость усложняет межпрограммное взаимодействие. Накладными расходами является также необходимость поиска подходящих конвертеров и редакторов (можно

воспользоваться [5]) и общих соглашений по работе над конкретной задачей.

Литература

[1] Comprehensive TEX archive network, http://ctan.org

[2] «The LATEX Graphics Companion», Michel Goossens, Sebastian Rahtz, Frank Mittelbach. Русскоязычная версия была выпущена издательством «Мир» в 2002 году под названием «Путеводитель по пакету LATEX и его графическим расширениям».

[3] https://en.wikipedia.org/wiki/DOT_(graph_description_language), раздел limitations

[4] LatexDraw – http://latexdraw.sourceforge.net/, jPicEdt http://jpicedt.sourceforge.net/

[5] http://stackoverflow.com

Уменьшение простоя (downtime) при обновлении сетевого приложения

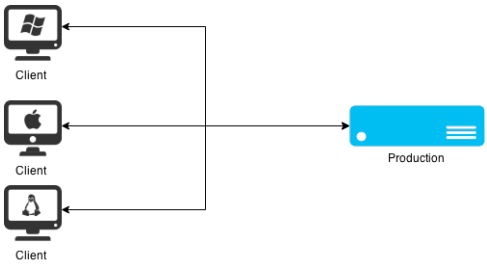
Aliaksandr Kharkevich, Gomel, Belarus[‖]

Continuous availability is an absolute priority for the network applications. However, there are cases when the application is unavailable to users (eg. software update). This publication offers one of possible solutions.

В связи с релизом nginx 1.9.0, в котором была добавлена возможность балансировки любых приложений работающих через TCP, дальнейший материал справедлив не только к веб-ресурсам, но и к любым другим, работающим поверх TCP (СУБД, системам аутентификации, каталогам LDAP, VoIP-системам и т. п.).

При обновлениях, да и при некоторых других работах, простой (downtime) для сетевого приложения является неизбежным злом. Но с ним нужно и можно бороться.

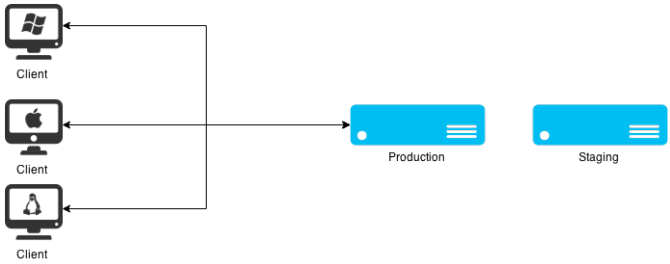
Для примера рассмотрим простейший случай, как на рисунке ниже:



Имеется некое клиент-серверное приложение, и периодически возникает задача его обновлять. При обновлениях возникает время простоя, пользователи выражают недовольство и перестают пользоваться данным приложением (заходить на сайт, оставлять заявки с службу технической поддержки и т. п.).

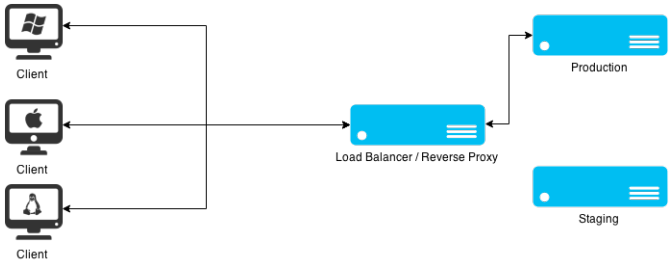
[‖] other.bigmouse@gmail.com, http://lvee.org/ru/abstracts/147

Первым этапом в улучшении приложения стала модификация сетевой инфраструктуры — добавился сервер для тестирования обновлений, так называемый *staging-сервер*:



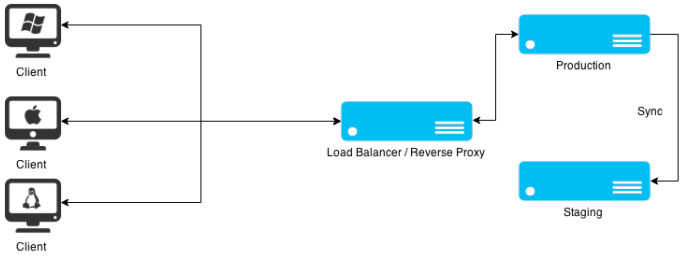
Стало лучше, возможность успешной установки обновлений предварительно проверялась, проверялась и работоспособность основных функций приложения после обновления. Время простоя уменьшилось (как и недовольство пользователей), но не исчезло.

Следующим этапом в улучшении стало добавление обратного прокси в структуру приложения:

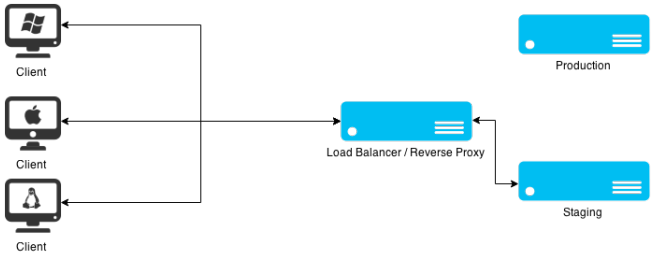


Данная структура обеспечила возможность горизонтального масштабирования системы и позволила гибко переключаться между серверами при необходимости (ликвидировать простои на время обслуживания).

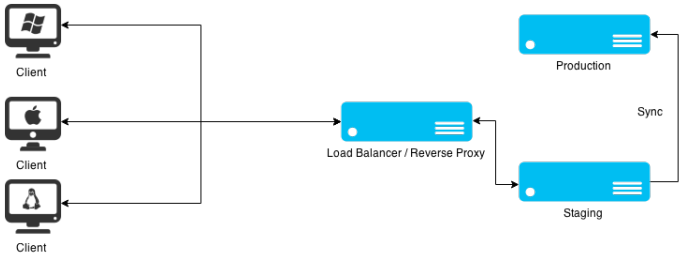
Изначально было решено поступить следующим образом: при подготовке сервера к обновлению происходит синхронизация данных на staging-сервер, затем на нем происходит обновление приложения и основная проверка работоспособности:



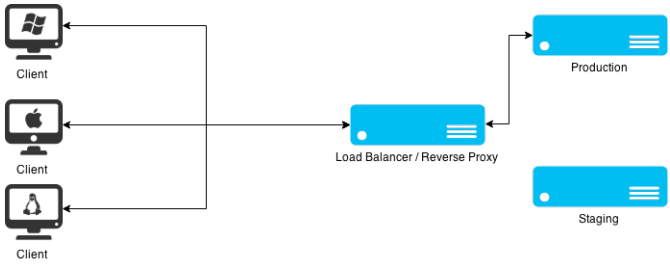
Затем происходит переключение обратного прокси на staging-сервер, переключение длится несколько секунд (из серьезных недостатков — мы теряем все активные сессии пользователей). В остальном — для конечного пользователя вообще ничего не меняется.



После переключения нагрузки на staging-сервер, на *production-сервере* обновляется приложение. После успешного обновления и проверки его жизнеспособности начинается синхронизация данных со staging (теоретически конечные пользователи уже внесли изменения в данных приложения).

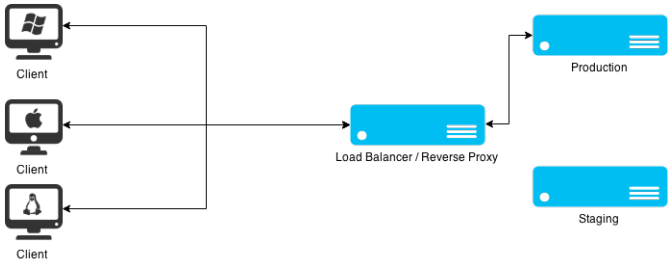


После окончания синхронизации происходит переключение обратного прокси-сервера обратно на production-сервер.

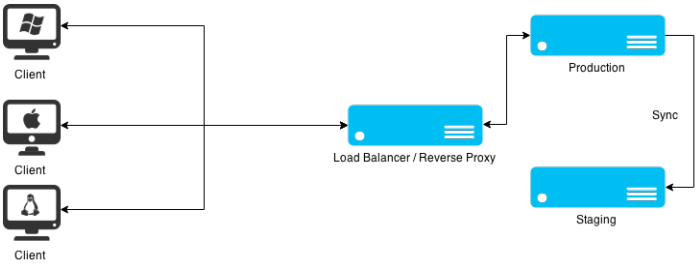


Но при тестировании на реальных данных нашелся небольшой процент пользователей который изменял данные в момент после синхронизации данных и до переключения между серверами. В связи с этим, было решено изменить механизм работы приложения.

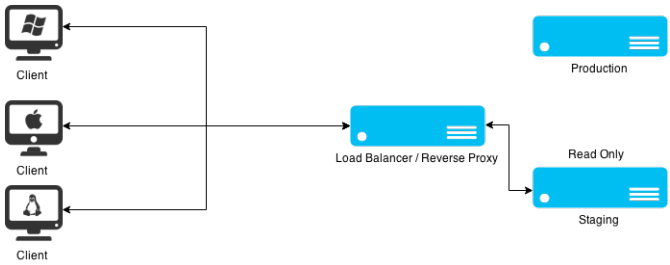
Был разработан механизм, переводивший приложение в режим «только чтение» и уведомлявший пользователей. Перед обновлением production-сервер переводился в режим «только чтение».



После этого происходила синхронизация на staging-сервер. Приложение на staging-сервере также в режиме «только чтение».



После этих манипуляций происходит переключение production-прокси на staging, и при этом мы убрали вероятность потери данных в момент синхронизации-переключения. Пользователи получили доступ к приложению и его данным только на чтение на момент обновления, количество негативных отзывов стало минимальным за все время (все отнеслись с пониманием к надписи, предупреждающей о технических работах). Далее — переключение приложения на staging, обновление production-сервера, переключение обратно, и лишь затем — отключение режима «только чтение».



Вместо заключения

Мы добились высокой доступности приложения. При этом, синхронизация данных максимально упрощена, необходимости в изменении работы логики приложения нет (что в свою очередь позволяет ее применять к уже существующим решениям без добавления и/или изменения логики работы).

Параўнальны аналіз сродкаў кросплатформеннага праграмавання

Г. Злобін, А. Чмихало, Львів, Украіна*

The article provides a comparative analysis of crossplatform programming tools. Crossplatform programming tools are divided into three groups: crossplatform compiled languages (Table 1), crossplatform programming languages at the execution level (Table 2) and crossplatform interpreters (Table 3). The information in Tables 1–3 is ordered by number of supported operating systems (descending). Table 4 provides information about standardized libraries and frameworks used in crossplatform programming. To substantiate the analysis results, the index of popularity of programming languages TIOBE is used.

Кросплатформеннасць — гэта здольнасць праграмнага забеспячэння працаваць больш чым на адной платформе або аперацыйнай сістэме. Кросплатформннасць праграмнага забеспячэння набыла асаблівае значэнне пасля завяршэння эры практычна непадзельнага панавання платформы Wintel (x86 + Microsoft Windows). Як вынікае з [1], колькасць працоўных месцаў з не-Wintel платформай у 2012 перавысіла 65% і працягвае павялічвацца. Гэта зрабіла эканамічна прывабным кросплатформеннае праграмаванне ў галіне распрацоўкі прыкладнога праграмнага забеспячэння.

Мовы праграмавання, якія можна выкарыстоўваць для кросплатформеннай распрацоўкі праграм, дзеляцца на тры групы:

- кросплатформавыя мовы праграмавання на ўзроўні кампіляцыі — для гэтых моў існуюць кампілятары для розных платформаў (C, C++, Pascal, Fortran, Ада і г.д.);
- кросплатформавыя мовы на ўзроўні выканання (Java і C#) — вынікам працы кампілятара ў гэтых мовах ёсць байт-код, які можна запускаць на розных платформах з дапамогай віртуальных машын (Java VM для Java і CLR для C#);
- кросплатформавыя інтэрпрэтатары — для гэтых моў з'яўляюцца інтэрпрэтатары (PHP, Perl, Python, Tcl, Ruby і т.д.) для розных платформаў.

*zlobingg@gmail.com, http://lvee.org/en/abstracts/148

Таблица 1.

Інструментальная абалонка	Падтрымліваюцца кампіляторы/колькасць моў праграмавання	Падтрымліваюцца АС / іх колькасць
Qt Creator	GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW / 8	Linux, OS X, Windows, Unix, iOS, Android, Blackberry 10, WinRT, Embed. Linux, QNX / 10
Eclipse	C/C++, Fortran/3	AIX, FreeBSD, HP-UX, Linux, Mac OS X, OpenSolaris, Solaris, QNX, Windows, Android (AR) / 10
Free Pascal	Free Pascal Compiler, Object Pascal, част. GNU Pascal, ISO Extended Pascal / 4	DOS, FreeBSD, Linux, Mac OS X, Windows, Sun Solaris, Haiku / 7
Lazarus	Free Pascal Compiler / 1	Linux, FreeBSD, Mac OS X, Windows, Android / 5
Code:: Blocks	MinGW, GCC C/C++, GNU GCC (PowerPC, ARM, AVR), SDCC, Digital Mars (C/C++, D), Visual C++, Borland C++, Watcom, Intel C++, GNU Fortran, GNU ARM, GNU GDC / 15	Windows, Linux, Mac OS X, Unix / 4
NetBeans IDE	C, C++, Ада / 3	Windows, Linux, FreeBSD, Solaris / 4
Embarcadero RAD Studio XE7	Delphi, C, C++ / 3	Windows, Mac OS X, iOS, Android / 4

Разгледзім кароткія характарыстыкі кросплатформавых моў праграмавання на ўзроўні кампіляцыі ў табліцы 1.

У табліцы 2 прадстаўлены кароткія характарыстыкі кросплатформавых моў на ўзроўні выканання[1].

У табліцы 3 прадстаўлены кароткія характарыстыкі кросплатформавых інтэрпрэтатараў.

На жаль, аўтарам невядомыя даследаванні папулярнасці сродкаў кросплатформеннага праграмавання, таму скарыстаемся індэксам TIOBE [2], па якім ацэньваюць папулярнасць моў праграмавання. Як вынікае з [2], найбольшай папулярнасцю больш за 10 гадоў карыстаюцца мовы праграмавання C (на 1.2015 г. 16,703%) і Java (15,528%), якія шырока выкарыстоўваюць для кросплатформеннага праграмавання. Менш папулярныя C++ (6,705%), C# (5,045%), PHP (3,784%), Python (2,613%), Perl (2,256%), Delphi / Object Pascal (0,837%). Яшчэ адным фактарам адбору можа быць колькасць аперацыйных сістэм і колькасць апаратных платформаў, для якіх можна выкарыстоўваць сродкі кросплатформеннага распрацоўкі. Не менш важнымі для кросплатформеннага праграмавання з'яўляецца стандартызаваныя бібліятэкі часу выканання. У прыватнасці, стандартам стала бібліятэка мовы Ci. Свои стандартныя бібліятэкі маюць C++, Java, Python, Ruby, якія падаюцца разам са сродкамі распрацоўкі і даступныя на падтрымоўваных платформах. Варта адзначыць таксама некаторыя вялікія кросплатформавыя бібліятэкі — такія як Qt, GTK+, FLTK, STL, Boost, OpenGL, SDL, OpenAL, OpenCL. У табліцы 4 стандартныя бібліятэкі часу выканання падзеленыя на бібліятэкі з адкрытым кодам і бібліятэкі з зачыненым кодам.

Высновы

1. Па папулярнасці моў праграмавання сярод кампілятараў першае месца займае мова праграмавання C, сярод інтэрпрэтатараў — мова Java.
2. Па індэксе Tiobe ў студзені 2015 мова праграмавання Delpi / Object Pascal займае 20 месца.
3. ППа колькасці аперацыйных сістэм, у якіх можна скарыстацца згаданымі ў аглядзе сродкамі распрацоўкі, яны размешчаны ў табліцах 1–3. Апошняя радок у табліцы 1 (кросплатфор-

[1]Праз вялікую колькасць інструментальных сродкаў для Java іх пералік няпоўны

Таблица 2.

Інструментальная абалонка	Падтрымліваюцца кампілятары/колькасць моў праграмавання	Падтрымліваюцца АС / іх колькасць
Eclipse[1]	Java	AIX, FreeBSD, HP-UX, Linux, Mac OS X, OpenSolaris, Solaris, QNX, Microsoft Windows, Android (ARM) / 10
NetBeans IDE[1]	Java	Windows, Linux, FreeBSD, Solaris / 4
IntelliJ IDEA[1]	Java	Linux, Mac OS X, Windows / 3
AIDE[1]	Java	Android
Microsoft Visual Studio Code	C#, Java/2	Linux, Mac OS X / 2
.NET Core 5	C#	Linux, Mac OS X / 2
Mono	C#	Linux, MacOS / 2

Таблица 3.

Інструментальная абаолонка	Падтрымліваюцца кампілятары / колькасць моў праграмавання	Падтрымліваюцца АС / іх колькасць
Eclipse	Perl, PHP, JavaScript, Python, Ruby/5	AIX, FreeBSD, HP-UX, Linux, Mac OS X, OpenSolaris, Solaris, QNX, Microsoft Windows, Android (ARM) / 10
NetBeans IDE	Java, JavaFX, PHP, JavaScript, HTML5, Python, Groovy / 7	Windows, Linux, FreeBSD, Solaris / 4
Embarcadero RAD Studio XE7	HTML5	Windows, Mac OS X, iOS, Android / 4
Xojo IDE Real	Basic	OS X, Windows, Linux, iOS / 4
Komodo IDE/Komodo Edit	Perl, PHP, Python, Ruby, Tcl. JavaScript, CSS3, HTML5, XML, XSLT / 10	Linux, Mac OS X, Windows / 3
PyCharm	Python, JavaScript, HTML / 3	Windows, Linux, Mac OS X / 3
Aptana Studio 3	JavaScript, PHP, Ruby, Python / 4	Windows, Linux, Mac OS X / 3
Microsoft Visual Studio Code	Python, JavaScript, PHP, JSON, XML, HTML, CSS / 7	Mac OS X, Linux / 2

Таблица 4. Стандартныя бібліятэкі і праграмныя каркасы з адкрытым і зачыненым кодам

Бібліятэкі і праграмныя каркасы з адкрытым кодам / адкрытыя стандарты	Бібліятэкі і праграмныя каркасы з зачыненым кодам
Boost, GIMP ToolKit, GTK+, FLTK, Kivy, OpenCV, OpenCL, OpenGL, SDL, Apache Cordova, Tk	
OpenAL (раннія версіі)	OpenAL (пазнейшыя версіі)
Qt	Qt
Simple DirectMedia Layer	Unity3D

меннага мовы праграмавання на ўзроўні кампіляцыі) з двума падтрымоўванымі АС займае Microsoft Visual Studio Code, ў табліцы 2 (кросплатформенныя мовы праграмавання на ўзроўні выканання) тры апошнія радкі з двума падтрыманымі АС займаюць Microsoft Visual Studio Code, .NET Core 5, Mono, у табліцы 3 (кросплатформавыя інтэрпрэтатары) апошні радок з двума падтрыманымі АС займае Microsoft Visual Studio Code.

4. Як і варта было чакаць колькасць стандартных бібліятэк з адкрытым кодам амаль у чатыры разы перавышае колькасць стандартных бібліятэк з зачыненым кодам.

Інфармацыйныя сродкі

1. D. Thompson. The 11 Most Fascinating Charts From Mary Meeker's Epic Slideshow of Internet Trends. `http://tinyurl.com/ozwck2k`
2. TIOBE Index for May 2015. `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

О том как маленький opensource-проект меняет жизнь большой компании

Павел Емельянов, Москва, РФ[†]

CRIU is the open source checkpoint-restore project of the Odin (former Parallels) company. It provides basis for containers live migration, seamless kernel update and a set of other features. In this talk I will present the current state of the project, describe the community that has grown around it and show how the open development model of a small project affected the life of the whole company.

OpenVZ — это открытый проект компании Odin, дающий пользователям простую, но надёжную контейнерную платформу. Неотъемлемой частью проекта с самого его начала является возможность живой миграции контейнеров, для чего используется технология «снятия контрольных точек» (checkpoint) и восстановления из них (restore). В процессе продвижения контейнерной технологии в массы инженеры компании были вынуждены переписать C/R подсистему практически с нуля и в другой парадигме — вместо ядерного модуля checkpoint-restore теперь делается силами процесса с использованием открытых ядерных интерфейсов. Вместе со сменой «адресного пространства» кода был изменён и подход к разработке — CRIU это 100% открытый проект без скрытых компонент и без диктатуры инженеров Odin при принятии архитектурных и технических решений.

За 4 года своего существования CRIU разросся до 100 тысяч строк кода и, что ещё важнее, достиг определённых успехов в социальной сфере.

Во-первых, проект завоевал признание в сообществе Linux kernel, куда изначально предлагалась реализация технологии, и теперь достаточным поводом для начала обсуждения ядерных патчей может служить простая фраза: «это надо для CRIU».

Во-вторых, CRIU «подружился» с другими проектами, например Docker и LXC, так что теперь идеи и улучшения мы получаем не только от клиентов Odin-а.

[†]xemul@openvz.org, http://lvee.org/en/abstracts/149

В-третьих, CRIU оброс небольшим сообществом, которое уже принесло свои плоды — портирование на архитектуры AArch64 и Power, интеграция с LXC и Docker и много другого было сделано не нами, но и для нас в том числе.

И, наконец, проект оказал сильное влияние на весь процесс разработки компании Odin. Недавно начатая новая жизнь OpenVZ планировалась с учетом приобретённого в CRIU опыта ведения открытых проектов.

Рекурсивное наблюдение за файловой системой на примере librnotify

Андрей Змушко, Минск, Беларусь*

One of base questions that appear in data synchronisation, is getting file notifications from the directory. Common solution for this is using a Linux kernel subsystem such as inotify or fanotify. Since fanotify doesn't notify file deletions, file renames or file moves, it is difficult to use fanotify for such applications. But of course fanotify gives as pid of process which have caused the event to happen. This can be very useful for some special cases, for example we should ignore events created by themselves. But in synchronisation it is very important to provide good support of move folders, in fact it is common problem for most synchronisation apps. In this subject we will review inotify just deeper. Known problems of inotify are that it provides no way of recursive monitoring and no map between watch descriptor and real path. Both thess questions have been solved in librnotify.

Предположим, стоит задача синхронизировать данные, хранящиеся в некоторых директориях, доступ к которым не ограничен. Очевидно, что мы будем вынуждены периодически рекурсивно просматривать директорию, чтобы находить различия или подписаться на файловые нотификации, которые поставляются такими подсистемами ядра Linux, как inotify и fanotify.

fanotify имеет серьезный недостаток: не поддерживаются нотификации удаления и перемещения. Поэтому применение его в синхронизации данных усложняется.

Рассмотрим inotify. Основных недостатков в нем, пожалуй, три: первый — это то, что нотификация сопровождается только лишь дескриптором, и ничего не известно о пути сработавшего элемента ФС, если об этом не позаботиться заранее.

Второй — это ограничения (лимиты) в ядре на количество запущенных нотификаторов и длину очереди нотификаций. Лимиты можно и нужно менять, и скорее ваше приложение упрется в ограничения, связанные с производительностью, чем в лимит.

*andrey.zmushko@gmail.com, http://lvee.org/en/abstracts/152

И последний недостаток — это то, что совсем не поддерживается рекурсивная нотификация папок. То есть когда, допустим, мы перемещаем папку внутрь нотифицируемой папки, пока интерфейс просигнализирует о новой папке и мы включим эту новую папку в нотификации, несколько файлов «проскочат» мимо, и мы получим рассинхронизацию данных; это случается практически постоянно. Одним из известных решений по устранению этой проблемы является проект lsync. Действительно, он полностью решает две проблемы: маппинг путей и рекурсивные нотификации. Но он громоздок и непроизводителен в качестве подсистемы.

В одном из наших проектов мы использовали lsync как сервер нотификаций, но если нужно синхронизировать десяток разных корневых папок с сотнями тысяч файлов и папок внутри, решение с lsinc оказывается дорогим в плане потребления системных ресурсов. В итоге мы заменили lsync на библиотеку librnotify.

Подробно опишем интерфейс этой библиотеки. Он прост и содержит только три вызова:

```
Notify* initNotify(const char* path, const uint32_t mask,
                   const char* exclude);
int waitNotify(Notify* ntf, char** const path, uint32_t*
               mask, const int timeout, uint32_t* cookie);
void freeNotify(Notify* ntf);
```

Объект Notify является контейнером состояния нотификатора, и с нашей точки зрения нам не интересен. initNotify подписывает некую папку по пути path на список нотификаций, задаваемый в mask (список тот же, что и в man inotify). Естественно, файл sys/inotify.h также должен присутствовать. exclude — это регулярное выражение, которое задает маску для элементов ФС, которые мы не хотим нотифицировать — это удобно, если, например, мы не хотим нотифицировать .git или просто .*.

Вызов waitNotify ждет нотификации timeout секунд (или микросекунд), а при значении 0 ждет вечно. Он возвращает path и битовое поле (mask), где указанно, что именно случилось. Cookie — это возвращаемое число, которое позволяет связать нотификации перемещения; оно будет одинаковым для пар операций MOVE_FROM и MOVE_TO. Наконец, freeNotify удаляет объект нотификатора.

Библиотека librnotify доступна по адресу `https://github.com/zmushko/librnotify`

Программирование на C для PostgreSQL

Александр Коротков, Москва, РФ*

PostgreSQL offers great extendability. Users can add literally everything on their own: data types, functions, operators, index types, procedural languages and so on. But in order to use the full power of these features one should write C code for PostgreSQL. Traditionally it's assumed that barrier to entry of C programming for PostgreSQL is very high. That's why extendability of PostgreSQL is no as demanded as it could be. The goal of present talk is to overcome this circumstances.

PostgreSQL обладает отличной расширяемостью, пользователи могут добавлять сами буквально всё: типы данных, функции, операторы, типы индексов, языки хранимых процедур и т.д. Но для того, чтобы использовать многие из этих возможностей, нужно уметь программировать под PostgreSQL на C.

Традиционно считается, что это сложно и порог вхождения очень велик. Из-за этого вся мощь расширяемости PostgreSQL оказывается не так востребована, как могла бы быть. Мне хотелось бы это обстоятельство постепенно преодолевать. Действительно, как и в любом большом проекте, написанном на языке C, программирование под PostgreSQL имеет свои особенности.

В PostgreSQL используется свой calling convention, благодаря которому функции, написанные как на C, так и на процедурных языках, могут быть видны из SQL. Есть универсальный тип данных — Datum, к которому может быть приведено значение любого типа. Параметры и результат передаются как Datum. Ниже приведён пример функции, использующей PostgreSQL calling convention.

```
PG_FUNCTION_INFO_V1(increment);

Datum
increment(PG_FUNCTION_ARGS)
{
    int32   arg = PG_GETARG_INT32(0);
```

*aekorotkov@gmail.com, http://lvee.org/en/abstracts/153

```
    PG_RETURN_INT32(arg + 1);
}
```

В PostgreSQL есть свой менеджер памяти: любое выделение памяти осуществляется в рамках некоторого контекста памяти. Контексты памяти, в свою очередь, образуют иерархическую структуру. Ниже приведён шаблон функции, которая возвращает набор строк (set-returning function). Такая функция вызывается для каждой отдельной строки. Память, которую эта функция выделяет, будут освобождена перед следующим вызовом, но также доступен контекст памяти, которых сохраняется между вызовами.

```
Datum
my_set_returning_function(PG_FUNCTION_ARGS)
{
    FuncCallContext  *funcctx;
    Datum             result;

    if (SRF_IS_FIRSTCALL())
    {
        MemoryContext oldcontext;

        funcctx = SRF_FIRSTCALL_INIT();
        oldcontext = MemoryContextSwitchTo(funcctx->
multi_call_memory_ctx);
        /* Инициализация структур памяти, которая
выполняется только один раз */
        пользовательский код
        MemoryContextSwitchTo(oldcontext);
    }

    /* Инициализация структур памяти, которая выполняется
каждый раз */
    пользовательский код
    funcctx = SRF_PERCALL_SETUP();
    пользовательский код

   /* Нужно ли вернуть ещё одну строку или все строки уже
возвращены? */
    if (funcctx->call_cntr < funcctx->max_calls)
```

```
    {
        /* Возврат следующей строки результата (result) */
        пользовательский код
        SRF_RETURN_NEXT(funcctx, result);
    }
    else
    {
        /* Все строки уже были возвращены, выполняется
освобождение использованных ресурсов, если нужно */
        пользовательский код
        SRF_RETURN_DONE(funcctx);
    }
}
```

Благодаря тому, что любой контекст памяти можно очистить в любой момент времени, во многих случаях можно не освобождать отдельно каждый участок памяти. При этом накладные расходы несравнимо малы по сравнению с применением сборщика мусора.

Посылать к БД SQL-запросы можно напрямую в тот же backend, из которого вызвана функция. Для этого есть специальный интерфейс — SPI. Ниже приведён пример функции, который выполняет запрос и выводит его результаты в виде INFO сообщений. Её аргументами являются текст запроса в виде text, и максимальное число выводимых строк ответа.

```
int
execq(text *sql, int cnt)
{
    char *command;
    int ret;
    int proc;

    /* Преобразуем text в C-строку */
    command = text_to_cstring(sql);

    SPI_connect();

    ret = SPI_exec(command, cnt);

    proc = SPI_processed;
    /*
```

```
     * Если удалось получить строки,
     * то выводим их через elog(INFO).
     */
    if (ret > 0 && SPI_tuptable != NULL)
    {
        TupleDesc tupdesc = SPI_tuptable->tupdesc;
        SPITupleTable *tuptable = SPI_tuptable;
        char buf[8192];
        int i, j;

        /* Цикл по строкам ответа */
        for (j = 0; j < proc; j++)
        {
            HeapTuple tuple = tuptable->vals[j];

            /* Цикл по полям строки ответа */
            for (i=1, buf[0] = 0; i <= tupdesc->natts; i++)
                snprintf(buf + strlen (buf), sizeof(buf) -
strlen(buf), " %s%s",
                         SPI_getvalue(tuple, tupdesc, i),
                         (i == tupdesc->natts) ? " " : " |");
            elog(INFO, "EXECQ: %s", buf);
        }
    }

    SPI_finish();
    pfree(command);

    return (proc);
}
```

Но если немного привыкнуть, то для того, кто уже имеет опыт на C, программировать под PostgreSQL не так уж и сложно. А благодаря тому, что уже есть готовые удобные макросы, функции, структуры данных, это может быть даже проще, чем на чистом C.

Мифы и легенды о проекте OpenVZ

Сергей Бронников, Москва, РФ*

In 1999, the company SWsoft (Parallels) was born, as far as the concept of container virtualization. We have formulated three major components that define the container: a set of processes with insulation namespaces, file system for the separation of code and memory and resource isolation. In 2000, the company's employees have prepared a concept of a commercial product Virtuozzo, which allows to create isolated Linux environments (containers). In 2002 the company released a public version of Virtuozzo and in the same year first commercial users have appeared. In 2005 the company have launched OpenVZ project to develop open implementation of Linux containers. This project team is developing a Linux kernel with containers, an utility for container management and other tools. Over the past 10 years, the project gained popularity. OpenVZ is used not only as a platform for hosting, but also for infrastructure applications requiring isolation. During the existence of the project some myths and misconceptions about the project have emerged, which could not dispel the answers on forums, blogs and e-mail. This report is intended to debunk these OpenVZ-related myths.

В 1999 году в компании SWsoft (Parallels) родилась концепция контейнерной виртуализации. Мы сформулировали три главных компонента, составляющих контейнеры: набор процессов с изоляцией namespaces, файловая система для разделения кода и памяти и изоляция ресурсов. В 2000 году сотрудники компании подготовили концепт коммерческого продукта Virtuozzo, который позволял создавать в ОС Linux изолированные окружения (контейнеры). В 2002 году компания выпускает публичную версию Virtuozzo и в том же году появляются первые коммерческие пользователи. В 2005 году компания Parallels запускает проект OpenVZ для разработки открытой реализации Linux контейнеров. В рамках этого проекта команда проекта разрабатывает Linux ядро с поддержкой контейнеров, утилиту для управления контейнерами и другие вспомогательные утилиты. За прошедшие 10 лет проект приобрёл популярность.

*sergeyb@openvz.org, http://lvee.org/en/abstracts/154

OpenVZ используют не только как платформу для хостинга, но и для инфраструктурных задач, требующих изоляции приложений. За время существования проекта появились мифы и заблуждения о проекте, которые не получилось развенчать ответами на форумах, в блогах и почтовых переписках. Одни и те же вопросы возникают снова и снова. Этот доклад призван окончательно развенчать мифы о проекте OpenVZ.

Из-за сложившейся недостатка новостей о проекте и неправильной интерпретации тех новостей, которые появлялись на новостных сайтах, возник миф о том, что проект OpenVZ умер и его развитие прекращено.

Процесс разработки технологии контейнеров в Linux ядре в компании Parallels имеет свои особенности, из-за чего появился второй миф о том, что проект OpenVZ устарел и использует устаревшую версию ядра Linux (RHEL6, ветка 2.6.32), тогда как последний релиз ванильного ядра имеет версию 4.0.4. На самом деле ядра RHEL сильно отличаются от ванильных ядер той же версии (к примеру, RHEL6 и ванильное ядро 2.6.32). Ядра RedHat — это компромисс между стабильностью, безопасностью и функциональностью. Они могут не содержать все последние наработки, но наиболее важные изменения и исправления RedHat активно бэкпортирует в свои ядра.

Вследствие взрывного роста популярности контейнерной виртуализации появилось множество проектов, которые так или иначе используют эту технологию, и большой объем информации в Интернет. Вдобавок появились предпосылки для изменения парадигмы использования контейнеров. Это привело к тому, что инженеры стали в своем выборе отталкиваться от популярности проекта, а не от сценариев использования. Однако на самом деле, несмотря на лавинообразный рост популярности и стремление использовать Docker в везде где только можно, эта технология имеет свои ограничения и применима не ко всем сценариям. Точно также, проект LXC ни в коем случае не является конкурентом OpenVZ.

В силу ограничений инструментов разработки проект OpenVZ долгое время не имел открытого репозитория для работы с кодом, хотя архивы с исходном кодом выкладывались в открытый доступ. Этот факт затруднял возможность сделать вклад в проект. Два месяца назад мы запустили в строй репозиторий с исходным кодом ядра и пользовательских утилит. Теперь факт о закрытости разработки OpenVZ превратился в миф.

Основной сценарий использования коммерческого продукта Virtuozzo — это хостинг-провайдеры, поэтому OpenVZ, как похожий по функциональности проект, пользуется успехом среди хостеров, которые по тем или иным причинам не хотят использовать коммерческую версию. Однако OpenVZ имеет среди своих пользователей и компании, никак не связанные с хостингом (разработка ПО, киностудии, научные центры и т. д.).

Кроссплатформенная разработка. Опыт Opera Software.

Алексей Хлебников, Осло, Норвегия*

Cross-platform development gives you more users, less development costs, immunity to vendor lock-in and other goodies. But how to do it properly? Below the experience of Opera Software with its famous Opera browser is presented, including problems that arised and how they were solved.

Зачем нужна кроссплатформенность

Преимущества кроссплатформенной разработки очевидны. Если программа написана под несколько платформ — у неё будет больше пользователей. Затраты на разработку будут гораздо меньше, чем если бы под каждую платформу программа писалась бы с нуля. Разработчик не привязан к единственной платформе, а поэтому приобретает иммунитет к vendor lock-in. Разработка кроссплатформенного приложения заставляет задуматься о хорошем дизайне программы, избегать привязки к «нестандартностям» и недокументированным функциям платформы, что приводит к меньшему числу багов. Есть и рекурсивный аргумент: если программа написана «в кроссплатформенной манере» — добавление поддержки ещё одной платформы происходит гораздо легче.

Платформы, которые поддерживал браузер Opera

Примером ультра-кроссплатформенной программы можно считать (старый) веб-браузер Opera на движке Presto. Благодаря высокой кроссплатформенности этого движка, браузер Opera поддерживал следующие платформы:

- Desktop, в том числе FreeBSD, Solaris, OS/2, BeOS и QNX
- Планшеты
- Smartphone, включая Symbian/Series60, Maemo/Meego, Bada, BlackBerry, Windows CE/Mobile

*alexei.khlebnikov@gmail.com, http://lvee.org/en/abstracts/155

– Feature phone, в том числе BREW, P2K, EPOC, Psion, Java ME
– Игровые консоли, в том числе Playstation 3, Nintendo Wii и Nintendo DS
– Smart и не-smart TV, TV-приставки
– Автомобильные компьютеры
– Панель управления башенного крана

Пожалуй, Opera кроссплатформеннее ядра Linux.

Возникающие проблемы

Конечно, при кроссплатформенной разработке встречаются и трудности. Разные платформы — значит разные устройства, с разными характеристиками:

– Разные скорости CPU и объёмы памяти
– Разные размеры экранов и их количество
– Разные устройства управления
– Разные дополнительные устройства (камера, GPS, etc)
– Разные компиляторы, в том числе и капризные (ADS)
– Особые методы запуска приложений (BREW, P2K)
– Разные API у разных операционных систем

Требования Opera 8.5x/Presto 1.0 (2005˜2011)

Некоторые производители, например Motorola, поставляли телефоны с браузером Opera Mobile 8.54 вплоть до 2011 года.

Системные требования для Opera 8.5x на движке Presto 1.0 составляли:

– 32-битный процессор
– 8 МБ оперативной памяти, доступной для использования Оперой
– C++-подобный компилятор для платформы

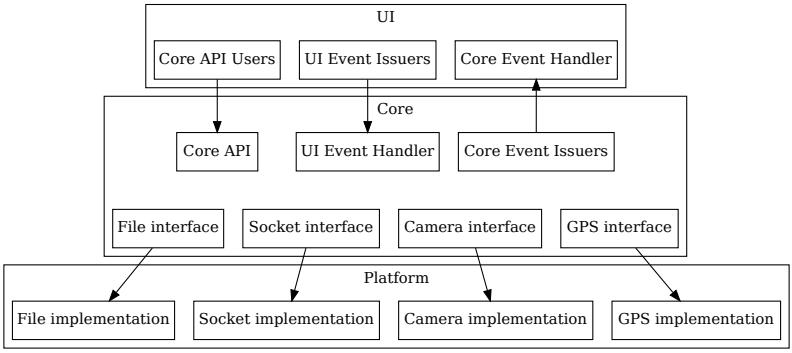
Остальные возможности платформы, даже устройства ввода и вывода, сеть и файловая система, являлись, строго говоря, необязательными. Например, Presto успешно обходилось без устройств ввода-вывода на серверах Opera Mini.

В списке не случайно упомянут «C++-подобный» компилятор. Opera могла быть собрана практически любым компилятором, даже самым капризным.

Как же Опере удавались эти чудеса? Секрет в хорошем проектировании программы.

Модель кроссплатформенного приложения

Архитектура «старой Оперы» представлена на следующей схеме:



Приложение состоит из 3 частей:

- UI — реализация UI, платформо-зависимый код.
- Core — ядро/движок, платформо-независимый код.
- Platform — реализация поддержки файловой системы, сети и устройств, платформо-зависимый код.

Стрелки на схеме означают:

- CoreAPIUsers → CoreAPI — Команды от UI к Core, например «Перейти на такой-то URL в таком-то табе».
- UIEventIssuers → UIEventHandler — События в UI, которые должны быть обработаны Core, например клик мыши.
- CoreEventIssuers → CoreEventHandler — События в Core, которые должны быть обработаны в UI, например перерисовка области экрана.
- Some interface → Some implementation — вызовы кода, реализующие «платформенный интерфейс», то есть отвечающего за поддержку файловой системы, сети и устройств на данной платформе.

Таким образом, для того, чтобы добавить поддержку какой-либо платформы, надо:

– Решить, какие возможности браузера и устройства хочется поддерживать на этой платформе.
– Разработать реализацию UI и платформенных интерфейсов, с учётом выбранных возможностей.

То есть, если какая-то возможность браузера (например, поддержка GPS) не будет активирована на данной платформе — не надо и реализовывать платформенный интерфейс GPS-устройства. Подробнее включение/отключение возможностей программы будет описано ниже.

Как решать проблемы

Проблема	**Решение**
Разные скорости CPU и объёмы памяти	Включение/отключение возможностей на стадии компиляции
Разные размеры экранов и их количество	Задание размера экрана/окна при инициализации/изменении
Разные устройства управления	Поддержка в UI layer, возможно частично в Platform layer
Разные дополнительные устройства (камера, GPS)	Поддержка в Platform layer, обязательный `#ifdef SOME_DEVICE_SUPPORT`
Разные компиляторы, в том числе и капризные (ADS)	Ответственное использование C++ (namespaces, templates, exceptions, RTTI, global vars, static)
Особые методы запуска приложений (BREW, P2K)	Реализация UI как библиотеки
Разные API у разных операционных систем	Поддержка в Platform layer

Выше были упомянуты некоторые проблемы, присущие кроссплатформенной разработке. Приведенная таблица кратко описывает, как их решать.

Включение/отключение возможностей на стадии компиляции

Некоторые платформы, например feature phones имеют не очень-то много памяти, как для хранения кода приложения, так и для его работы. Отключение некоторого функционала приложения на стадии компиляции помогает уменьшить размер кода этого приложения, расход памяти, а также ускоряет его работу.

Также, чем менее мощны устройства определённой платформы — тем, как правило, меньший спектр встроенных устройств (камера, GPS, etc) они поддерживают. Код для неподдерживаемых встроенных устройств на данной платформе также следует исключать из компиляции.

Для того, чтобы реализовать отключаемую поддержку определённого функционала или устройств на стадии компиляции для определённой платформы, используется простой способ:

1. Обрамить код поддержки отключаемого функционала или устройства в #ifdef SOME_FEATURE_SUPPORT.
2. Описывать (#define) или сбрасывать (#undef) соответствующий макрос в глобальном заголовочном файле для соответствующей платформы.

Ниже приведён фрагмент кода на C++, который иллюстрирует описанный способ:

```
#define FEATURE_1_SUPPORT
#undef  FEATURE_2_SUPPORT
    ...
    // Код feature 1 будет скомпилирован,
    // потому что решено поддерживать
    // feature 1 на этой платформе.
#ifdef FEATURE_1_SUPPORT
    m_component_loader->LoadFeature1Data();
#endif
    ...
    // Код feature 2 не будет скомпилирован,
    // потому что решено не поддерживать
```

```
    // feature 2 на этой платформе.
    // Это уменьшит размер исполняемого кода,
    // потребление памяти во время исполнения
    // и увеличит его скорость.
#ifdef FEATURE_2_SUPPORT
    m_component_loader->LoadFeature2Data();
#endif
    ...
```

Как ещё уменьшить размер кода

Существуют также и другие способы уменьшить размер кода:

- Использовать код повторно, удалять более неиспользуемый код
- Уменьшить размер временных буферов и/или их время жизни (совет для runtime)
- Больше использовать библиотеки платформы, не дублировать их функционал в коде приложения
- Урезать используемые third-party библиотеки, всё-таки включённые в приложение; таким же образом (#undef FEATURE), как урезается основной код
- Избегать зависимостей от слабых мест компилятора, (пример: раздувание template-кода)
- Кросс-компилировать лучшим компилятором
- Компилировать в более компактный набор инструкций процессора (пример: Thumb для ARM)
- Сжимать код, в ROM или RAM
- Использовать плагины/оверлеи
- Исполнять части кода в другом месте (пример: Opera Mini)

Пример платформенного интерфейса

Ниже приведён пример платформенного интерфейса для сокета TCP. Реализовав этот платформенный интерфейс, разработчик обеспечит работу TCP-соединений на данной платформе. Для Unix-подобной платформы реализация может использовать BSD sockets, для Windows — WinSock, для Telium OS — LinkLayer и т. д.

```
// Будет реализовано в платформенном коде.
class TCPSocket
```

```
    {
        static TCPSocket* New();
        Status SetListener(TCPSocketListener* listener) = 0;

        Status Connect(const char* host, unsigned short
port) = 0;
        Status Disconnect() = 0;
        Status Send(const char* data, size_t length, size_t&
accepted_length) = 0;
        Status Recv(char* data, size_t length, size_t&
received_length) = 0;
    };

    // Будет реализовано в коде ядра.
    class TCPSocketListener
    {
        Status OnConnected() = 0;
        Status OnDisconnected() = 0;
        Status OnDataSent(size_t length) = 0;
        Status OnDataAvailable(size_t length) = 0;
        Status OnError(Error code) = 0;
    };
```

Опыт выбора и применения современных систем мониторинга

Наим Шафиев, Баку, Азербайджан*

Modern heterogeneous infrastructure, which contains different types of network devices and servers, needs modern monitoring systems. Also the main problem of finding proper solution is that it should fulfil requirements of flexibility, openness, good support from community. The article presents an overview and experience of modern free monitoring systems, which fulfil spoken above requirements in case of a middle-size ISP.

Основные требования

Проблема роста и взаимозаменяемости сотрудников для решения задач мониторинга и DevOps проявляются в любой компании при росте от нескольких человек до так называемого среднего размера.

Рассмотрим, какие основные требования предъявляются к системам мониторинга в современных условиях:

- опрос агентами ресурсов (CPU, Disk I/O, Ram, Network) серверов;
- опрос сетевых устройств различных устройств (очень желательно,чтобы были готовые MIB);
- опрос VMware-серверов;
- гибкая система оповещений с многоуровневой системой зависимости;
- opensource.

Также очень важны понятная документация и легкий старт — как минимум, для взаимозаменяемости сотрудников.

Практика использования

Ниже представлены особенности, родившиеся из опыта поиска и использования различных решений (таких, как Nagios, Zabbix, LibreNMS, NetXMS) на размере более 500 активных узлов в ISP

*shafiev@gmail.com, http://lvee.org/ru/abstracts/159

среднего размера. Рассмотрев все эти решения мы пришли к следующим выводам:

- Nagios — к сожалению, началось разделение на бесплатную Core-версию и XI-версию, все это привело к появлению более технологичного форка Icinga. Также из минусов стоит отметить не доведенную до промышленного использования многоуровневую систему взаимосвязи объектов и сервисов.
- Zabbix — лучший opensource-продукт, возможен высокий уровень детализации данных, развитое community. Из минусов — сложность в настройке, сложно дорабатывать.
- LibreNMS — наиболее перспективный свободный продукт. Очень легкий старт (сравнимый с PRTG), использование стандартных компонентов. Включает огромное количество готовых MIB. Из минусов — невозможность без осложнений задать детализированные графики (менее 5 минут), нету развитой системы зависимостей (в данный момент автор решает эту задачу).
- NetXMS — один из наиболее технологичных продуктов индустрии. Содержит высокопроизводительное ядро, возможен высокий уровень детализации и другие стандартные для проприетарных систем функции. Из минусов отмечается сложный старт, ручная настройка, малое community.

Автор в данный момент дорабатывает LibreNMS для замены связки Nagios/Zabbix+MRTG.

Coreboot. Практическое знакомство со свободной альтернативой BIOS

Николай Стомчик, Минск, Беларусь*

In modern Open Source world we need an open alternative to the proprietary product called BIOS. This alternative exists and is called coreboot. Coreboot is not fully equal to BIOS, it's only does initialization of RAM, execution of binary vendor blobs and starting some payload. The report main task is to cover how to start using it.

Введение

Для построения полностью свободного стека ПО необходимо иметь свободную альтернативу проприетарным прошивкам (firmware): классическому BIOS, либо UEFI в более новых системах. Такая альтернатива есть, и она называется coreboot.

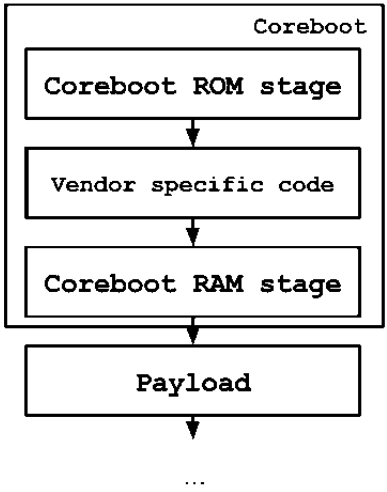
Практические причины для использования CoreBoot [1] могут быть:

- Нужда в свободной альтернативе UEFI;
- Создание единого командно-интерфейсного слоя в уровне «прошивки» для кластерных решений на различных системах и архитектурах;
- Ускорить загрузку системы до ОС (до $\sim$ 3 с)
- Разместить всю ОС (GNU/Linux, FreeBSD или другую) во flash-памяти, позволив работать в режиме восстановления либо в полноценном режиме без использования жесткого диска;
- Общеобразовательные :)
- Предотвращение угрозы полумифического BadBIOS [2, 3];

Процесс загрузки Coreboot

В процессе загрузки, устройство с coreboot проходит через следующие стадии:

*mn3m00@gmail.com, http://lvee.org/en/abstracts/160



- ROM stage — инициализация памяти и начальная инициализация чипсета; помимо общей схемы действий эта часть по понятным причинам очень специфична для конкретного оборудования, и здесь в обязательном порядке присутствует выполнение некоторых частей бинарной прошивки, которая определена производителем как обязательная для инициализации RAM, CPU или других системных устройств;
- RAM stage — нумерация устройств, назначение ресурсов, создание таблицы ACPI, SMM handler;
- Payload — выполнение некой полезной нагрузки, которая в свою очередь может быть неким приложением либо загрузить операционную систему.

Coreboot не является полной заменой BIOS, т.к. не содержит никакой активной части по выполнению «полезных действий»: эта часть вынесена в отдельный модуль, называемый «payload» (полезная нагрузка). Достоинство такого разделения — возможность менять payload. Например, автором в качестве payload использован SeaBIOS, но есть альтернативы, такие как «GRUB2», «OpenBIOS». Можно «выполнить» в качестве payload ядро Linux или FreeBSD, а можно создать собственный: например, очень нужный в реальной практике payload, который пишет «hello world» и завершается:

```
include <libpayload.h>
int main(void)
{
    printf("Hello, world from CoreBoot! :)\n");
    return 0;
}
```

Сборка этого шедевра функциональности может быть выполнена следующим способом:

```
$ lpgcc -o hello.elf hello.c
```

После чего, данный Payload необходимо добавить в Ваш бинарный файл Coreboot.rom и можно перейти на этап загрузки его в ПЗУ.

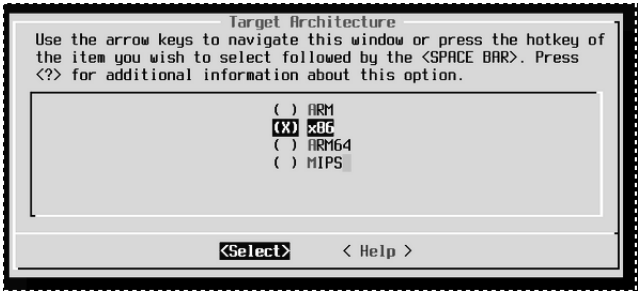
Подготовка coreboot к прошивке

По понятным причинам поддержка всего многообразия чипсетов и материнских плат проектом coreboot оставляет желать лучшего. Однако, если ваше оборудование попало в число поддерживаемых, процедура перепрограммирования может оказаться не самая простая, особенно в случае если производитель ноутбука либо компьютера поспешил обезопасить себя от желающих самостоятельно заменить базовую микро-программу.

В частности, подготовительные шаги для установки coreboot на основе личного опыта автора по перепрошивке ThinkPad x201[4] выглядели следующим образом:

- тщательное изучение документации по разборке ноутбука, чтобы отыскать чипы памяти BIOS и CMOS;
- чтение datasheet'ов на ROM и выяснение специфики его программирования;
- покупка (либо сборка) программатора и разъёмов (SOIC-8P) для ISP программирования;
- извлечение videorom.bin из оригинального проприетарного BIOS, которую можно получить из ОЗУ в процессе работы системы;
- создание резервной копии firmware с помощью программного или аппаратного программатора (эксперименты с подобными материями быстро излечивают от неоправданного оптимизма);

– клонирование coreboot и его зависимостей из git, выполнение ‘make menuconfig’ и ‘make’, с последующими n часами сборки с удовлетворением всех зависимостей;
– программирование в ПЗУ результирующего бинарного файла;
– многочасовое тестирование компьютера с новой прошивкой, с использованием нагрузочных тестов и проверкой всего оборудования.

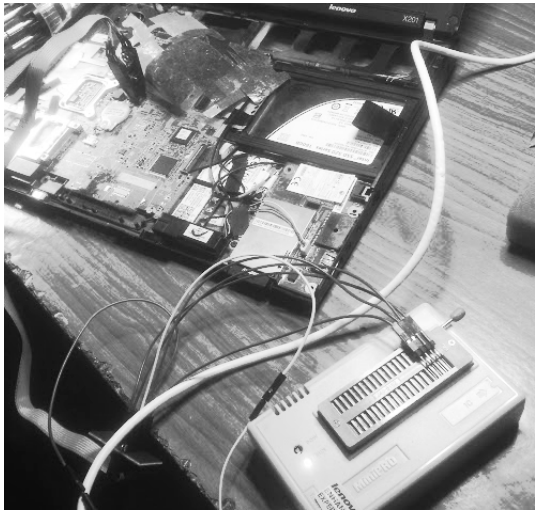


Саму запись микропрограммы в постоянную память можно выполнить из Linux её штатными средствами — «flashrom», либо, если этот этап завершился неудачей — аппаратным программатором. В случае с Thinkpad x201 для программирования MX25L6445E (8MB Serial Flash) подходят программаторы:

– BusPirate v4;
– minipro TL866C;
– Raspberry PI;
– любой другой, который умеет программировать через SPI.

Из-за того, что программная перепрошивка оказалась заблокирована вендором (а способ разблокировки, применяемый для обновления фирменной прошивки, на данный момент неизвестен), для LENOVO Thinkpad x201 пришлось воспользоваться аппаратным программатором minipro TL866. При работе из GNU/Linux команда перепрошивки должна выглядеть примерно следующим образом:

```
./minipro -i -p "MX25L6445E @SOP8" -w {Patch_for_your_
coreboot}build/coreboot.rom
```



Советы по внутрисхемному программированию

- Важно следить за уровнем питания микросхемы на плате. Если питания не достаточно, есть смысл воспользоваться внешним блоком питания. Обычный стандартный блок ATX для таких задач вполне подходит (необходимо +3.3V);
- Для получения корректных результатов считывания и записи «Земля» (GND) программатора, корпуса ноутбука и блока питания должны быть соединены;
- Перед записью будет не лишним попробовать прочитать содержимое микросхемы N раз и проверить контрольные суммы — SHA или MD5. Если они не совпадают, проблема наверняка кроется в двух предыдущих пунктах.

Работоспособность после прошивки

В результате экспериментов с ThinkPad x201 были выявлены следующие мелкие баги (которые, возможно, на момент чтения этой статьи уже будут исправлены):

- ограничение «на перегрев CPU» теперь отсутствует, что требует добавить ещё один скрипт проверки на перегрев, либо понизить немного частоту CPU до стабильной, что бы этот перегрев и не возникал;

– каждые 10 минут возникает «засыпание» экрана, вне зависимости от установок xset (Исправляется `xset -dpms off`);
– при подключенном втором дисплее перестает регулироваться яркость подсветки основного дисплея используя горячие клавиши;
– понадобилось переписать скрипт регулировавший частоту процессора из-за новых значений шага для частоты CPU;
– проигрывание звука на колонках отключается программно, то есть при включенных наушниках можно включить воспроизведение на встроенных динамиках ноутбука;

Однако список работающего функционала оказался внушительным:

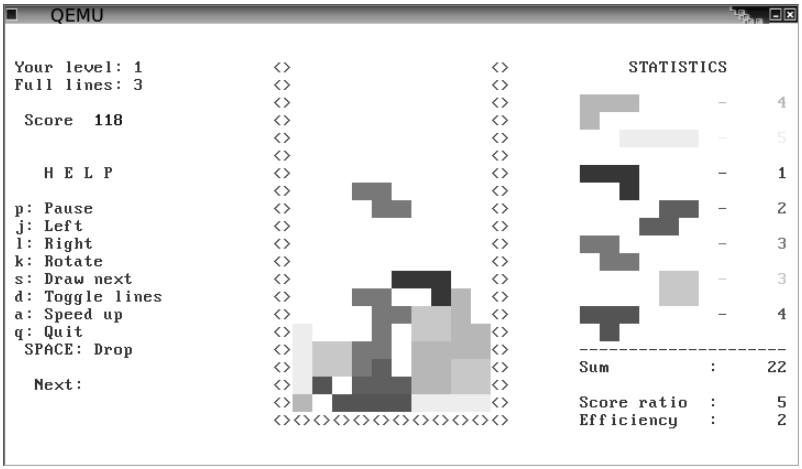
– 3G GPRS modem;
– 4G+4G RAM;
– функциональные клавиши ThinkPad (фонарик, громкость);
– расширения VT-x процессора;
– аппаратная поддержка AES в CPU;
– WIFI;
– Ethernet;
– Все порты ноутбука (USB, DP, HDMI, DSUB), а также докстанция;
– Устройство считывания отпечатков пальцев;
– Управление питанием устройства, контроль за состоянием батареи.

Время загрузки микропрограммы, инициализации всего аппаратного обеспечения и передача управления начальному загрузчику ОС (GRUB2) сократилось с ~ 50 с до ~ 4 с, и при том большую часть из этого времени SeaBIOS ожидает нажатия F12 для выбора устройства загрузки. Хотя ускорение загрузки не являлось целью данной работы, нельзя не отметить, что логотип coreboot соответствует его значению:

Конечно, данное обстоятельно не может не радовать.

Отладка

Подготовленный payload можно протестировать в эмуляторе QEMU-KVM перед запуском, чтобы удостовериться, что всё работает как требуется. В самом же устройстве можно использовать традиционный для этих задач индикатор POST-кодов, отображающий значения, отправленные в порт 80h, последовательный порт — либо PC-speaker как средства диагностики состояния и процесса загрузки.



Литература

[1] http://www.coreboot.org

[2] Bruce Schneier. badBIOS // Schneier on Security. https://www.schneier.com/blog/archives/2013/11/badbios.html

[3] Jonathan Brossard. Hardware backdooring is practica. // DEFCON 20 presentation. http://www.slideshare.net/endrazine/defcon-hardware-backdooring-is-practical

[4] github.com/mn3m0nic/ffts/blob/master/thinkpad.md

Qucs — свободный симулятор электронных схем: новые возможности релиза 0.0.19

Вадим Кузнецов, Калуга, РФ*

Qucs (Quite Universal Circuit Simulator) is open source circuit simulation CAD tool. The main features of Qucs are considered. New features of upcoming 0.0.19 release and spice4qucs subsystem are reviewed. Spice4qucs allows to simulate circuits from Qucs using external simulator kernels such as Ngspice and Xyce.

В настоящее время существует не так уж и много open-source САПР. Тем не менее, среди САПР для электроники (EDA) есть весьма достойные продукты. Доклад посвящён симулятору электронных схем с открытым исходным кодом Qucs `http://qucs.sourceforge.net`. Qucs написан на C++ с использованием фреймворка Qt4. Qucs является кроссплатформенным и выпущен для ОС Linux, Windows и MacOS. Текущей версией проекта является 0.0.18. В настоящее время ведётся подготовка к релизу версии 0.0.19.

Разработку данной САПР начали в 2004 году немцы Michael Margraf и Stefan Jahn (в настоящее время не активны). Сейчас Qucs разрабатывается интернациональной командой, в которую входит автор статьи. Руководителями проекта являются Frans Schreuder и Guilherme Torri.

Qucs позволяет проводить следующие виды моделирования:

1. Моделирование на постоянном токе (DC analysis)
2. Моделирование в частотной области (AC analysis)
3. Моделирование во временной области (Transient analysis)
4. Параметрический анализ (Parameter sweep)
5. Моделирование S-параметров в частотной области (S-parameter)
6. Синтез пассивных фильтров, согласованных схем, расчёт коаксиальных и микрополосковых линий.

*`ra3xdh@gmail.com`, `http://lvee.org/en/abstracts/162`

Результаты моделирования можно визуализировать в виде графиков в декартовых (2D и 3D) и полярных координатах, а также в виде таблиц и диаграмм Смита.

В настоящее время существуют следующие open-source средства моделирования электронных схем:

1. Ngspice `http://ngspice.org` — консольный симулятор электронных схем. Совместим с индустриальным стандартом моделей электронных компонентов SPICE.
2. Xyce `http://xyce.sandia.gov` — новейший spice-совместимый консольный симулятор, поддерживает параллельные вычисления через openMPI. Вышел в 2014 году. Совместим с индустриальным стандартом моделей электронных компонентов SPICE.

Недостатком вышеперечисленных симуляторов является отсутствие графического интерфейса, что сильно затрудняет ввод схемы. Для преодоления данного недостатка был разработан набор патчей spice4qucs, разработанный автором совместно с Mike Brinson (London Metropolitan University). Данный набор патчей позволяет использовать Qucs в качестве фронтенда для Ngspice или Xyce. Включение данного набора патчей в основную ветку ожидается в версии 0.0.19. В настоящее время поддерживаются все основные виды моделирования и компоненты. Текущий статус разработки можно отследить в репозитории проекта: `http://github.com/Qucs/qucs/issues/77`

Подсистема spice4qucs позволяет:

1. Моделировать схему Qucs при помощи внешнего симулятора Ngspice или Xyce
2. Использовать систему параметрического моделирования, совместимую со SPICE
3. Использовать постпроцессор Ngnutmeg
4. Использовать SPICE-модели из документации электронных компонентов без ограничений
5. Использовать специфические виды моделирования, совместимые с Ngspice и Xyce: (Fourier analysis, Distortion analysis, Noise analysis)
6. Проводить моделирование при помощи скрипта Ngnutmeg, задаваемого пользователем.

Функция поддержки внешних симуляторов, реализуемая подсистемой Spice4qucs, не имеет аналогов в проприетарном ПО.

Будущими задачами является разработка в следующих перспективных направлениях:

1. Расширение библиотеки компонентов
2. Разработка редактора библиотек
3. Разработка системы связи с KiCAD
4. Поддержка электромагнитного симулятора openEMS
5. Разработка системы экспорта моделей в формат Verilog-A

Таким образом, можно сделать вывод о том, что Qucs представляет собой быстроразвивающуюся САПР, по отдельным параметрам не уступающую проприетарным аналогам. можно рекомендовать Qucs для моделирования электронных схем в академических целях, на малых предприятиях и индивидуальным разработчикам электроники, а в некоторых случаях Qucs можно использовать и на крупных предприятиях для замены проприетарного ПО, закупаемого за рубежом.

Создание робота для Roborace

Дмитрий Склипус, Брест, Беларусь*

Author shares his development experience of Arduino-based robots targeted at Roborace competitions. Specifics of the competition is explained as far as some details of the hardware platform and the contol algorythm.

Специфика Roborace

Roborace — это состязания, в которых соревнуются роботы-автомобили на специальной кольцевой трассе. Можно провести некоторую аналогию между Roborace и гонками Формулы 1, за исключением двух моментов.

- Во-первых, вместо полномасштабных гоночных болидов участвуют уменьшенные модели авто и оригинальные конструкции с габаритными и весовыми ограничениями (максимальные ШхД=25x50 см и вес до 3 кг).
- Во-вторых, вместо пилотов автомобилем управляет бортовой компьютер, который анализирует показания различных датчиков и ориентирует автомобиль на трассе, выбирает скорость движения, предотвращает столкновения с препятствиями и соперниками. Собственно «поведение» авто на трассе определяется управляющей программой бортового компьютера.

Roborace проводится в виде чемпионата, состоящего из этапов, которые организуются в различных городах Беларуси и за рубежом. Участие в чемпионате принимают как конструкции начального уровня (например, на базе конструктора типа LEGO), так и сложные робототехнические устройства. Регламенты соревнований формируются таким образом, чтобы охватить как можно более широкий спектр характеристик и возможностей робототехнических конструкций.

Рассмотрим трассу, изображенную на рисунке 13.1, по которой предстоит перемещаться роботу. Обязательными элементами являются черные линии и стенки. Исходя из этого можно строить стратегию движения робота по трассе: например, оборудовать робота

*sklipus@gmail.com, http://lvee.org/en/abstracts/163

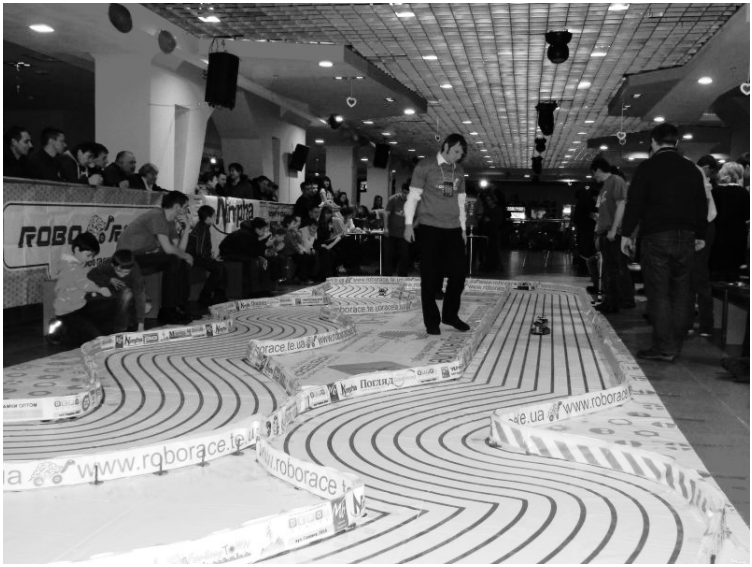


Рис. 13.1. Трасса для гонок роботов

датчиками черной линии и использовать линии трассы для навигации, или установить дальномеры для обнаружения препятствий и двигаться вдоль стенок.

В рамках данной статьи представлен один из роботов, разработанный мной для Roborace по второй стратегия (движение на основе показаний дальномеров).

Этапы создания робота

Создание робота для Roborace начинается с выбора шасси. Сейчас магазины предлагают большой выбор гусеничных и колесных платформ. Я рекомендую остановиться на классической схеме, когда задние колеса приводятся в движение электродвигателем, а передние управляются сервоприводом [1]. На рисунке 13.2 изображен робот для Roborace, построенный по подобной схеме.

Мне повезло за небольшие деньги приобрести модель в местном клубе радио-моделистов (читатели могут попытаться сделать то же самое в своем регионе: обычно у них много устаревших моделей). Так как в приобретенной модели не было тягового электродвигате-

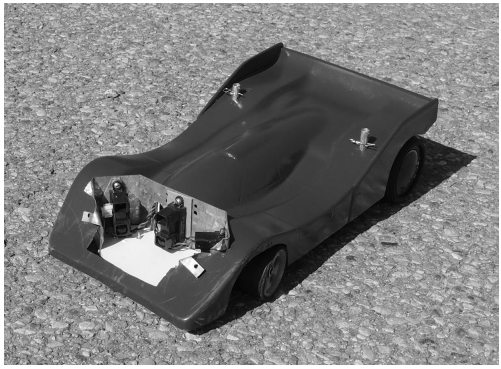


Рис. 13.2. Робот

ля, на нее был установлен установил купленный 12-вольтный мотор. Можно было также использовать обычную игрушку: они обычно довольно живучи, и требуется только модифицировать рулевое управление.

Так как в моем случае сервопривод уже был установлен, с ним проблем не возникло.

Следующий этап — выбор платы управления. Тут есть множество вариантов. Я выбрал Arduino как самый простой вариант. То же можно порекомендовать и читателю, особенно при недостатке опыта. Исходя из моего достаточно большого опыта, для таких роботов достаточно обычных 8-битных микроконтроллеров. Поэтому если не планируется использовать для отслеживания движений робота камеру, не стоит усложнять его более мощным процессором.

Сервопривод можно напрямую подключить к Arduino — например, через sensor shield, изображенный на рисунке 13.3. К нему также удобно подключать датчики.

Мотор подключить к Arduino напрямую не получится. Нужно использовать специальные Motor Driver. Сейчас из достаточно много в продаже, и есть инструкции по подключению. Я использовал Motor Driver, разработанный в нашей лаборатории (рис. 13.4).

В соревнованиях роботов приходится много внимания уделять батареям. Я использую литий-полимерные аккумуляторы. Они очень хорошо себя зарекомендовали. Один из хаков, которые я применяю в своем роботе, касается преобразователя напряжения. Штатный преобразователь в Arduino не очень хорош, поэтому для

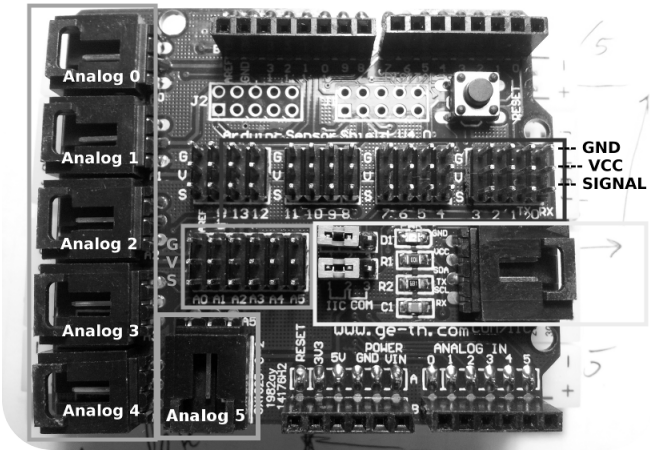


Рис. 13.3. Sensor shield v4

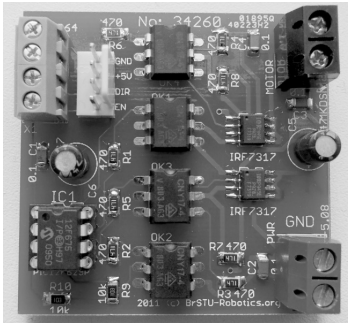


Рис. 13.4. Motor driver

экономии энергии аккумулятора неплохо использовать Step-Down регулятор [2]. Конечно, можно использовать и обычный линейный преобразователь.

Самая главная часть робота это датчики — то, что обеспечивает его информацией об окружающем мире, о препятствиях и о других роботах. В средней ценовой категории мы можем выбирать из *ультразвуковых* и *инфракрасных* датчиков. В своем роботе я использую инфракрасные датчики GP2Y0A02YK0F. Мне не нравятся

ультразвуковые датчики из-за того, что может происходить зашумление одного датчика другим. Например, у меня возникали такие ситуации: правый датчик посылал сигнал, а левый его принимал. Я все ещё работаю над правильным размещением ультразвуковых датчиков и над управлением ими. Надежду их запустить постоянно подпитывает их цена.

На представленной здесь модели робота установлено три инфракрасных датчика. Датчики можно увидеть на рисунке 2. Они установлены в глубь корпуса по двум причинам:

1. для уменьшения мертвой зоны датчика, которая у данной модели составляет 20 см;
2. корпус робота защищает датчики от механических повреждений во время столкновений с другими роботами.

Боковые датчики установлены под углом 45 градусов. Хорошо, если в конструкции робота предусмотрена регулировка угла их установки. Общую схему робота можно посмотреть на рисунке 13.5.

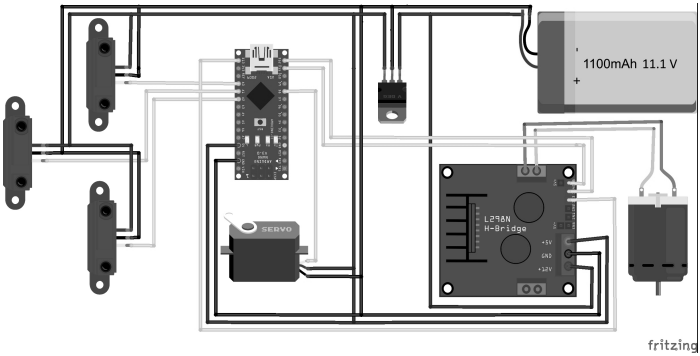


Рис. 13.5. Общая схема робота

Программирование робота

Так как на роботе используется Arduino, то программирование выполняется с использованием Arduino IDE. Программа робота представляет собой замкнутый цикл, который состоит из следующих блоков:

1. Фильтрация показаний датчиков;

2. Вычисление угла и скорости движения робота;
3. Передача управляющих сигналов на механизмы.

В данной структуре отсутствует блок получения информации от датчиков. Так как датчики возвращают аналоговый сигнал, в Arduino IDE есть функция analogWrite(). Данная функция замечательно работает, если не важна скорость измерения. Но так как робот разрабатывался для соревнований, было принято решение вынести обработку датчиков в прерывание.

Все платы Arduino, построенные на микроконтроллере ATmega, имеют возможность проводить измерения АЦП в автоматическом режиме. Нужно один раз настроить этот режим, а потом пользоваться полученными значениями. В результате контроллер постоянно проверяет датчики, не тратя на это процессорное время. Фильтрация показаний датчиков осуществляется медианным фильтром с окном в три элемента.

Для движения по трассе был выработан следующий алгоритм. Робот сравнивает расстояния до правой и левой стенки, и в соответствии с этим поворачивает колеса в нужное направление. Если впереди робота нет препятствий, скорость увеличивается, но также уменьшается максимально возможный угол поворота колес. Это нужно для того, чтобы на прямых участках робот ехал более прямо. При обнаружении препятствия угол поворота колес увеличивается, и робот притормаживает.

Есть конечно и нерешенные проблемы. Например, робот не знает кривизну поворота, поэтому тормозит перед каждым поворотом.

Посмотреть код проекта можно на GitHub [3].

Литература

[1] http://en.wikipedia.org/wiki/Servo_%28radio_control%29

[2] http://sites.google.com/site/sklipusrobotsystems/hardware/power-of-robot

[3] https://github.com/sklipus/roborace/tree/master/robots/FreshSRDEDITION

Использование пакета Blender при работе над анимационными проектами

Виктория Бабахина, Рязань, РФ*

The article gives an overview of the free software usage in film-making industry. It covers all the basic steps of creating an animated film with specifics related to Blender.

При работе над анимационным проектом перед художником-аниматором часто возникает ряд весьма нетривиальных задач. Если речь идет не о крупной студии, то аниматор часто становится и художником по фонам, и моделлером, и композером, и даже специалистом по видео-монтажу. В таких условиях очень важно иметь под рукой удобное программное обеспечение, позволяющее выполнять как можно большее количество задач из разных областей создания готового мультипликата, обладающее при этом достаточной интерактивностью, удобством в обращении и, что весьма важно, адекватной ценой. Здесь мы рассмотрим, как с этой ролью справляется пакет Blender.

Задачи

Вот лишь небольшой перечень задач, с которыми может столкнуться в работе художник-аниматор: создание фонов, моделирование и анимация трехмерных объектов, анимация двухмерных объектов и кривых, создание эффектов, работа с motion-графикой, композитинг.

Выбор рендера

Перед началом работы часто встает вопрос о том, каким рендером пользоваться. Blender предоставляет для работы следующие:

*vitorry@gmail.com, http://lvee.org/ru/abstracts/158

Internal. Рендер, более старый, чем Cycles, не фотореалистичный. Internal — это рендер с допущениями. К его несомненным плюсам можно отнести то, что он позволяет рендерить дым, волосы, ночные пейзажи и т. д., при рендере на CPU он значительно быстрее, отлично справляется со стилизованной картинкой, motion-графикой, стилизацией под 2D и со всем, где не нужен реалистичный рендер. Из минусов — больше работы с настройками материалов, рендер с допущениями, и в нем нет глобального освещения и цветных рефлексов.

Cycles. Плюсы: фотореалистичный рендер, быстрая и более удобная настройка материалов, просчет на GPU, что более удобно для графики, рендер, удобный для работы с интерьерами. Минусы — много проблем с шумом при рендере, трудности с рендером частиц и OSL.

Freestyle render. Предназначен для стилизации под 2D-изображение. Freestyle генерирует двумерные линии из набора объектов, линии могут быть стилизованы различными способами (разные цвета и толщины или добавление случайной неровности) для создания художественного (рисунка от руки) или технического (чертёжного) стилей. Freestyle может использоваться как дополнение к другим рендерам. Freestyle для Blender имеет два дополнительных режима для стилизации линий: параметрический редактор и режим Python-скриптов.

3D — моделинг, риггинг, текстуры, анимация

Основная функция пакета Blender — это работа с 3D-графикой. При работе над мультипликационным проектом, работа с трехмерной графикой бывает нужна даже в тех случаях, когда речь идет исключительно о 2D-анимации. К примеру, при создании фонов для упрощения и ускорения рабочего процесса бывает удобно сделать трехмерную модель сцены и рендерить нужные ракурсы при необходимости и в нужном количестве. Для сглаживания разницы между трехмерным фоном и нарисованными плоскими персонажами удобно использовать Freestyle-рендер, о котором шла речь ранее.

Часто более трудоемким является создание трехмерной модели персонажа. Обычные этапы создания трехмерного персонажа вы-

глядят так: Подготовительный этап — сбор референсов, работа над образом персонажа в эскизах, создание листов персонажа.

- Моделирование и текстуры — непосредственная работа над моделью.
- Риг и лицевой риг — оснащение персонажа скелетом и мимикой.
- Липсинк — создание речевой анимации и фонем.

В блендере есть возможность автоматизировать процесс, соответствующий последнему пункту. Для этого необходимо провести подготовительный этап — создать шейпы движения губ, соответствующие определенным буквам, и грамотно их переименовать. Затем требуется помощь стороннего приложения для кодирования звука — такой например, как Papagayo. В стороннее приложение загружается звуковой файл с текстом, который считывается и кодируется в формат, пригодный для чтения Blender. Результат загружается в Blender, с помощью специального плагина привязывается к модели и автоматически создает ключевые кадры с анимацией.

2D-анимация и motion-графика

Помимо работы над 3D-анимацией, Blender располагает обширным инструментарием для работы с 2D-графикой, который постепенно расширяется и развивается — порой в весьма неожиданных направлениях.

Ранее уже было сказано о рендере Freestyle, позволяющем создать имитацию рисованной анимации. Помимо стилизации во Freestyle, в Blender возможна работа над настоящими 2D-марионетками. С помощью плагина «Import image as plane» в блендер загружается персонаж — это можно сделать либо по частям, либо и одним изображением, чтобы выполнить его нарезку уже в Blender. Затем марионетка оснащается скелетом и анимируется.

Еще один прием, который способен сильно выручить при работе не только с двумерной анимацией, но и с motion-графикой — это анимация процедурных текстур. С ее помощью сложно получить реалистичные эффекты, какие получились бы при работе с симуляторами огня, дыма или воды; однако когда речь идет о более стилизованной графике, процедурные текстуры становятся незаменимы. К несомненным плюсам работы с ними относится высокая скорость просчета, отсутствие рывка движения при окончании цик-

ла анимации, возможность сделать анимацию любого размера без потерь качества.

В последнее время появился еще один, весьма неожиданный способ работы с 2D-анимацией в Blender — grease pencil. Изначально это была довольно удобная, но далекая от анимации функция рисования в окне программы. С ее помощью можно было выделить для себя важные моменты при моделировании, наметить траекторию будущей анимации и другие вещи, полезные в работе. И так было до выхода мультипликационного фильма «For You», созданного в технике покадровой анимации исключительно с помощью grease pencil.

Композитинг

Изображение, получаемое непосредственно после рендера — далеко не финальный результат. Огромное количество работы над изображением ведется на этапе постобработки — композитинга.

Финальная картинка при рендере — это финальная работа сочетания огромного количества слоев. То есть, даже если не ведется специальная работа над композитингом, он все равно ведется глубоко на уровне кода. В Blender эти слои называются *пассы*. Пассы позволяют влиять на свет, тени, переотражения, блики, цвета, материалы, маски, глубину и многое другое.

Работа с пассами — это не все, что можно осуществить при композитинге. С помощью различных входных нодов возможно, к примеру, заменить фон на изображение, сократив таким образом время рендеринга в разы. Также возможно разделить объекты рендера из одного изображения в разные слои и рендерить по отдельности все или несколько объектов сцены.

Помимо прочего, при композитинге есть возможность работы с масками (когда необходимо убрать, вставить или выделить какую-то определенную часть изображения), кеингом (замена однотонного яркого фона на что-то иное), трекингом (определение местоположения объектов с помощью камеры и последующая работа с полученными точками).

Таким образом, от раннего этапа создания фильма до постобработки Blender будет полезен художнику-аниматору и, обладая огромным количеством полезных функций, поможет справиться в том числе с довольно сложными и нетривиальными задачами.

Любительская малобюджетная аэрофотосъёмка с использованием свободного программного обеспечения

Дмитрий Самсонов, Москва, РФ*

This work concerns some tasks needed for KAP (Kite Aerial Photography) with free/libre software (mostly from Debian GNU/Linux). This method was successfully used for taking aerial photos of several open-airs in 2013 and 2014. Nowadays it is very important to have cheap and simple solution.

Иногда возникает необходимость сделать аэрофотосъёмку. Например, на фестивале, проводящемся на открытом воздухе. Свежепостроенные фестивальные объекты могут не попасть на общедоступные спутниковые снимки, поэтому снимать их приходится самостоятельно. В данной работе рассмотрена методика, применявшаяся при аэрофотосъёмке фестивалей «Пустые Холмы — Город Золотой» осенью 2013 года и «Фестиваля 17» летом 2014 года.

Задача состоит в том, чтобы, подняв фотоаппарат на достаточную высоту, сделать серию вертикальных снимков, после чего «склеить» их между собой, привязать к местности и привести в удобную для дальнейшей работы форму.

Немаловажна в наше время низкая стоимость как создания, так и эксплуатации решения подобной задачи.

Аппаратная часть

Готовые решения в виде специализированных самолётов стоят довольно дорого, а в самостоятельном производстве трудозатратны. Радиоуправляемые вертолёты («дроны») стоят дешевле, однако дрон, способный устойчиво поднимать в воздух относительно тяжёлый фотоаппарат, стоит существенно дороже дешёвых моделей (а лёгкие фото-видеокамеры дороги сами по себе). К тому же

*samson.samson.samson@gmail.com, http://lvee.org/ru/abstracts/161

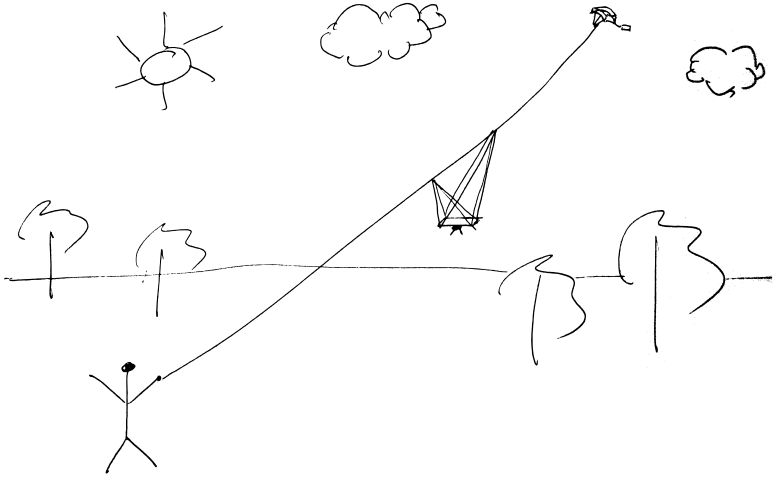


Рис. 15.1. Схематичное изображение процесса аэрофотосъёмки

у дронов высокие эксплуатационные издержки. Относительно дешёвы конструкции типа *воздушный змей* и *воздушный шар*. Был выбран первый тип.

Для создания *пикавета* — платформы, обеспечивающей вертикальность съёмки — была использована часть деревянной коробки, найденная среди бытовых отходов. Крючки для прикрепления пикавета к лееру были найдены в заброшенном цеху НИИДАРа (научно-исследовательский институт дальней радиосвязи).

Обеспечить серийную съёмку недорогими аппаратами стало возможным благодаря проекту CHDK — имеющийся функционал по выполнению скриптов позволил запрограммировать необходимое поведение даже у дешёвой «мыльницы» *Canon PowerShot A480*. Отдельным преимуществом является возможность работы от AA-батареек, что гораздо удобнее в полевых условиях, чем аккумуляторы «проприетарных» форматов.

Для привязки к местности снимаются GPS-координаты нескольких заметных объектов, однако для удешевления можно воспользоваться данными OpenStreetMap и публично доступными спутнико-

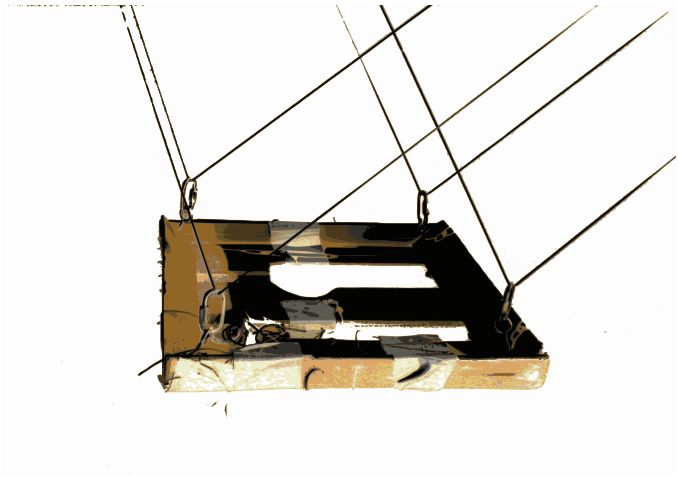


Рис. 15.2. Пикавет

выми снимками. В данном случае использовался *Garmin GPSMAP 62*.

Программные средства

Стоит отметить, что почти всё необходимое программное обеспечение (за исключением CHDK) можно обнаружить в дистрибутиве Debian GNU/Linux, что удобно в полевых условиях.

CHDK

Canon Hacker's Development Kit [1], так называемая «неофициальная прошивка», а на самом деле — резидентная программа, работающая без модификации прошивки фотоаппарата. Поддерживаются более сотни моделей аппаратов. Позволяет запускать самописные скрипты, программируя логику поведения аппарата в том числе в зависимости от внешних обстоятельств. Для данной задачи оказалось достаточным скрипта для съёмки timelapse, доступного «из коробки».

Hugin

Hugin [2] позволяет склеивать фотографии между собой для разных задач. Хотя пакет позиционируется как инструмент для создания круговых панорам, он имеет, наряду с прочими, возможность и «плоской склейки», нужной для данной задачи.

Склейка производится по контрольным точкам. Программа умеет автоматически их находить по похожим участкам изображения, однако на практике склейка проводится в полуавтоматическом режиме — требуется ручное вмешательство, как для не очень удачных снимков, так и из-за параллакса.

Hugin позволяет вычислить точность совмещения снимков, однако за пределами операции склейки в этом мало практического смысла.

QGis

QGis [3] — геоинформационная система. В нашем случае используется всего лишь для привязки к местности по GPS-точкам (с трансформацией по необходимости) склеенного изображения и получения на выходе изображения в формате GeoTIFF — содержащего метаданные о географической привязке.

gdal2tiles

Для генерации «тайлов», которые могут быть использованы для нужд удобного отображения полученного результата, может быть использован gdal2tiles [4].

OpenLayers

OpenLayers [5] — JavaScript-библиотека для удобной визуализации через браузер. Впрочем, также можно посоветовать использовать для этих целей Leaflet [6].

Итоги

Стоимость

Воздушный змей, способный поднимать несколько килограмм полезной массы («грузовой» или «лифт»), стоит чуть больше $100,

но при желании может быть сшит самостоятельно. Важной составляющей является *леер*, километровый моток которого стоит $50 (выдерживая нагрузку до 40 кг), однако на практике (после одного случая) оказалось более надёжным взять рыболовный шнур «на сома» меньшей длины по цене $30 (выдерживает нагрузку более 100 кг). Платформа для крепления фотоаппарата и мотовило для сматывания леера делаются самостоятельно, хотя при желании на рынке можно найти и готовые решения. Фотоаппарат подойдёт практически любой CHDK-совместимый, в данном случае использовался *Canon PowerShot A480* (купленный «с рук» за $20), хотя можно было бы достичь лучших результатов с помощью *Canon PowerShot SX20* (купленного «с рук» за $100).

Недостатки и преимущества

Выбранное решение обладает как рядом существенных недостатков, так и преимуществами.

Для подобного рода задач очевидны проблемы по трудозатратам на ручной отбор удачных снимков и склейку, борьбу с искажениями и параллаксом, зависимость от погодных условий («нелётная погода» при сильном ветре), проблемы безопасности находящихся под камерой людей. Для воздушного змея также характерна неполная управляемость и требования по силе ветра для старта: например, для использованного змея скорость ветра для успешного запуска должна превышать 3 м/с. Также необходимо соблюдать паритет между противоречивыми требованиями длины леера, его прочностью и весом, грузоподъёмностью воздушного змея и требованиями по минимальной скорости ветра для старта.

Попытки запуска с велосипеда при отсутствии естественного ветра не увенчались успехом — достичь высоты, необходимой для стабильного полёта, не удалось. Возможно, подобная задача требует конструктивного усовершенствования.

Ограничением также является необходимость открытого пространства для запуска и сложность в перемещении среди препятствий.

Также стоит отметить практическую сложность оценки погрешности точности определения местоположения каждой конкретной точки снимка при данном подходе, однако это приемлемо для любительской съёмки.

С другой стороны, использование воздушного змея позволяет, при удачном стечении обстоятельств, поднять его на высоту, где он может находиться неограниченно долго, пока позволяют погодные условия — в любое время суток, даже при отсутствии ветра у поверхности земли. Что позволяет его использовать для постоянного «лифта» не только для аэрофотосъёмки, но и для обеспечения мобильной связи в тех случаях, когда у поверхности земли она затруднена (на площадке размещается смартфон, а при необходимости поднять его повыше — также и мобильный роутер).

Применение

Полученную аэрофотосъёмку можно использовать не только в эстетических целях, но и для нужд любительского картографирования, а также в качестве базы для других проектов: например, можно отображать на карте фотографии, сделанные посетителями конкретного мероприятия — использование реалистичной подложки упрощает восприятие информации.

Перспективы дальнейшего усовершенствования метода

Для нивелирования некоторых недостатков погодных условий (отсутствие ветра для запуска) можно рассмотреть вопрос использования воздушного шара (BAP — Balloon Aerial Photography: чёрный воздушный шар, поднимающийся за счёт нагрева солнечными лучами воздуха внутри него) или применение более лёгкого воздушного змея, который может быть использован в качестве «лифта» для основного змея после поднятия на достаточную высоту для стабильного полёта.

Литература

[1] http://chdk.clan.su/
[2] http://hugin.sourceforge.net/
[3] http://qgis.org/ru/site/
[4] http://www.gdal.org/gdal2tiles.html
[5] http://openlayers.org/
[6] http://leafletjs.com/

Кратко о SO_BUSY_POLL

Евгений Рыбак, Минск, Беларусь*

Performance of Linux network stack was always kept in mind during any implementation. Increasing demand for cloud services, multimedia content, high performance computing forces to search new ways of optimization. SO_BUSY_POLL is one of the way to improve processing of incoming network messages without changing source code of the product.

Введение

Оптимизация скорости обработки входящих пакетов всегда была одной из главных задач сетевого стека. Рост облачных сервисов, объемов мультимедийного контента, высокопроизводительных систем (HPC) заставляет выискивать новые резервы для оптимизаций. SO_BUSY_POLL является еще одним способом улучшить производительность системы, практически не меняя исходный текст продукта.

Уровни стека TCP/IP

Как известно, стек TCP/IP состоит из уровней, каждый из которых решает четко поставленную задачу.

Физический уровень отвечает за передачу и прием электрических данных по сетевому кабелю, в соответствии с методами кодирования.

Канальный уровень отвечает за преобразование из электрического сигнала в цифровой кадр, аппаратную адресацию (MAC-адрес).

Сетевой уровень отвечает за связывание отдельных подсетей, в целую сеть. Идентификация узла в сети осуществляется с помощью IP-адреса.

Транспортный уровень отвечает за идентификацию процесса, в рамках узла, которому необходимо передать данные. Идентификация процесса осуществляется с помощью номера порта.

*engler@tut.by, http://lvee.org/en/abstracts/156

Модель OSI

Блоки	Транспортный безопасное и надёжное соединие точка-точка
Пакеты	Сетевой Определение пути и IP (логическая адресация)
Кадры	Канальный MAC и LLC (Физическая адресация)
Биты	Физический кабель, сигналы, бинарная передача данных

Методы обработки входящих пакетов

Классически, есть два основных подхода для того, что бы узнать, что на сетевой адаптер пришли новые пакеты:

– метод прерываний (IRQ);
– метод опроса (polling).

Каждый из методов имеет свои достоинства и недостатки: метод прерываний хорошо себя показал для небольшого объема трафика, когда прерывания генерируются не часто, не мешая процессору выполнять свои задачи. В случае большого и очень большого объема трафика метод является не очень эффективным; именно в этом случае метод опроса устройства является наиболее подходящим. Busy Poll (новая опция SO_BUSY_POLL) является усовершенствованием классического метода опроса, который уменьшает задержку (Latency) и увеличивает пропускную способность (Throughput).

Обзор опции SO_BUSY_POLL

Начиная с версии ядра 3.11, ядро осуществляет поддержку опции сокета SO_BUSY_POLL. В эко-системе SO_BUSY_POLL участвуют следующие компоненты:

– ядро операционной системы;
– драйвер сетевой карты;
– настройки системы.

Поддержка ядра операционной системы

Ядро должно быть собрано с опцией CONFIG_NET_RX_BUSY_POLL.

Драйвер сетевой карты

Драйвер сетевой карты должен реализовать обработчик функции, который отвечает за busy polling.

Настройки системы

В сетевых настройках появились две новые опции: busy_read, busy_poll. Значением является количество микросекунд, в течении которого ядро будет опрашивать входящую очередь сетевой карты (Ethernet RX queue). Прочитать и записать значения можно следующим образом:

```
sysctl net.core.busy_read
sysctl net.core.busy_poll
sysctl net.core.busy_read=50
sysctl net.core.busy_poll=50
```

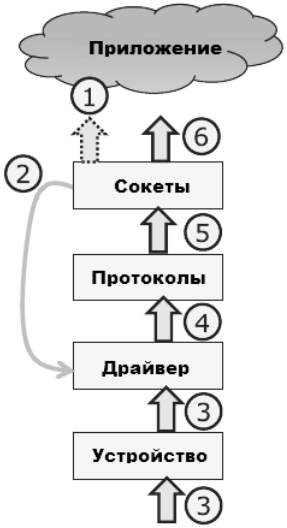
Также рассмотрим влияние опции SO_BUSY_POLL на различные методы чтения TCP/UDP-сокетов.

Отличие busy poll от NAPI poll

На рисунке выше показана работа приложения с точки зрения NAPI poll. Схема включает следующие этапы:

1. Приложение создает сокет и пытается из него прочитать данные.
2. Данные отсутствуют, ожидаем момент, когда они появятся в буфере сокета.
3. Данные приходят на интерфейс.
4. Генерируется прерывание, данные передаются на уровень протоколов.
5. Данные передаются в буфер сокета.
6. Приложение может прочитать данные из сокета.

В отличие от предыдущей, схема работы с включенной опцией SO_BUSY_POLL выглядит так:



1. Приложение создает сокет и пытается из него прочитать данные.
2. Если данных в буфере сокета нету, переходим к функции драйвера busy poll.
3. Данные приходят на интерфейс.
4. Генерируется прерывание, данные передаются на уровень протоколов.
5. Данные передаются в буфер сокета.
6. Приложение может прочитать данные из сокета.

Одна из отличительных возможностей busy poll заключается в том, что если данные в буфере сокета отсутствуют, то нам не надо ждать прерывания от драйвера сетевой карты, как в случае с NAPI poll. Busy poll инициирует «принудительную» передачу пакетов из входящей очереди драйвера в очередь сокета, тем самым исключая «лишнее» переключение контекста и прерывание.

Еще одним моментом, на который стоит обратить внимание, является то, что в случае обработчика NAPI poll система передает количество пакетов, которое она готова обработать в данный момент. Если количество пакетов в очереди драйвера больше, чем количество пакетов, которое система готова обработать в текущий момент, то обработчик NAPI poll будет еще раз поставлен в очередь вызовов; через некоторый промежуток времени обработчик будет вызван в очередной раз.

В случае с busy poll, обработчик busy polling будет вызываться в течении времени, которое будет указано в опции SO_BUSY_POLL или настройках busy_{read,poll}

Источники

Литература

[1] A way towards Lower Latency and Jitter. http://www.linuxplumbersconf.org/2012/wp-content/uploads/2012/09/2012-lpc-Low-Latency-Sockets-slides-brandeburg.pdf

[2] Open Source Kernel Enhancements for Low Latency Socket. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/open-source-kernel-enhancements-paper.pdf

[3] Net: low latency Ethernet device polling. https://lwn.net/Articles/540281/

[4] Understanding Linux Network Internals. http://www.amazon.com/Understanding-Network-Internals-Christian-Benvenuti/dp/0596002556

[5] Linux Kernel Development. http://www.amazon.com/Linux-Kernel-Development-Robert-Love/dp/0672329468/ref=sr_1_1?s=books&ie=UTF8&qid=1434287714&sr=1-1&keywords=Linux.Kernel.Development

[6] Linux Kernel Sources 3.18.14. https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.18.14.tar.gz

Data Science for Network Security

Dmitry Orekhov, Minsk, Belarus*

Today network traffic is absolutely out of human control, this is something that human mind cannot manage. On the other hand, network security becomes more and more important, since more and more of human activities are moving to the Network. The solution could be a software, which is able to learn from past Data incoming, and then to make assumptions about new Data and predictions about the future. Though algorithms for this domain are well-known, there is a problem to implement them, because they are often very resource-consuming. Fortunately, cloud technologies now afford building cheap and productive clusters, and Open Source solutions like Spark provide a powerful tool to build advanced analytics software on top of them.

Data collecting

The first important task for Network Analytics building is a data collection facility. The main idea is to place sensors, which collect statistics data and send it to be collect and analyze.

Fig.1: Statistics collection: the big picture

Sensors

Sensors are classified by Vantage and Domain. **Vantage** is a placement of sensors within a network. Sensors with different vantages would see different parts of the same events.

*Dmitry_Orekhov@epam.com, http://lvee.org/en/abstracts/164

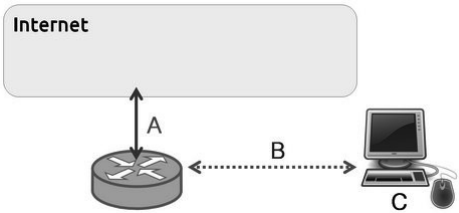


Fig.2: Vantages

Domain defines which part of thraffic statistics and metrics to collect. In the other words, it's the information the sensor provides, whether that's at the host, a service on the host, or the network. Sensors with the same vantage but different domains provide complementary data about the same event. For some events, you might only get information from one domain.

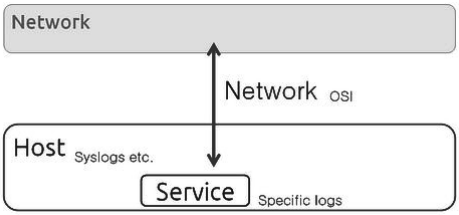


Fig.3: Domains

Use Case: DNS Tunneling

Tunneling – a mechaninsm to encapsulate low-lewel protocols into high-level protocol.

In case of DNS tunneling, DNS Resource Records so as Questions may contain Canonical Names; these Names can be easily encoded/decoded with Base32 or Base64 methods and can be used to transport unauthorized traffic or botnet protocol commands.

Methods to discover

Payload analysis

– Size of request and response

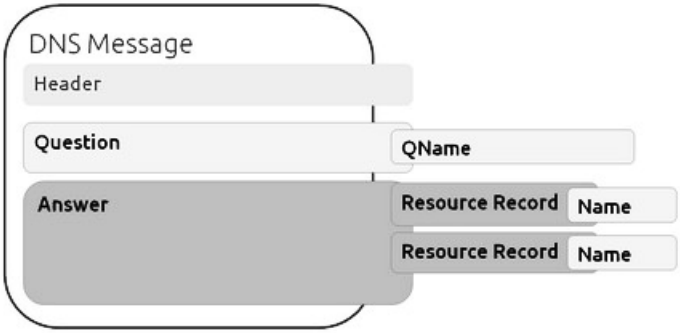


Fig.4: DNS message, a Big Picture

0 A	8 I	16 Q	24 Y
1 B	9 J	17 R	25 Z
2 C	10 K	18 S	26 2
3 D	11 L	19 T	27 3
4 E	12 M	20 U	28 4
5 F	13 N	21 V	29 5
6 G	14 O	22 W	30 6
7 H	15 P	23 X	31 7

Fig.5: Base32 encoding

- Entropy of hostnames
- Statistical Analysis
- Uncommon Record Types

Traffic analysis

- Volume of DNS traffic per IP address
- Volume of DNS traffic per domain
- Number of hostnames per domain
- Geographic location of DNS server
- Domain history
- Orphan DNS requests

The problem is that the most effective methods to discover DNS traffic demands deep packet inspection. Also, in case of models based

on long-term history, we have to store much statistical data and recompile the intrusion detection model in a batch job, to be up-to-date.

Algorithms

All algorithms, which could be used for DNS discovering, realate to Machine Learning area; the domain of it is building system, which can be learnt from Data sets and then make predictions about future trends. Today this is very important part of Data Science.

Online algorithms. It start with an initial state and analyze each piece of data serially one at a time.

- Generally require a chain of Map Reduce jobs
- Good fit for Apache Spark, Storm
- Primarily batch, good for Lambda architectures

Example 1: Outlier detection

- Median Absolute Deviation: Telemetry is anomalous if the deviation of its latest datapoint with respect to the median is X times larger than the median of deviations
- Standard Deviation from Average: Telemetry is anomalous if the absolute value of the average of the latest three datapoint minus the moving average is greater than three standard deviations of the average.
- Standard Deviation from Moving Average: Telemetry is anomalous if the absolute value of the average of the latest three datapoints minus the moving average is greater than three standard deviations of the moving average.
- Mean Subtraction Cumulation: Telemetry is anomalous if the value of the next datapoint in the series is farther than three standard deviations out in cumulative terms after subtracting the mean from each data point
- Least Squares: Telemetry is anomalous if the average of the last three datapoints on a projected least squares model is greater than three sigma
- Histogram Bins: Telemetry is anomalous if the average of the last three datapoints falls into a histogram bin with less than x

Example 2: Stream Classification

- Hoeffding Tree (VFDT) incremental, anytime decision tree induction algorithm that is capable of learning from massive data

streams, assuming that the distribution generating examples does not change over time
- Half-Space Trees ensemble model that randomly spits data into half spaces. They are created online and detect anomalies by their deviations in placement within the forest relative to other data from the same window

The possible topology Big Picture could be like this:

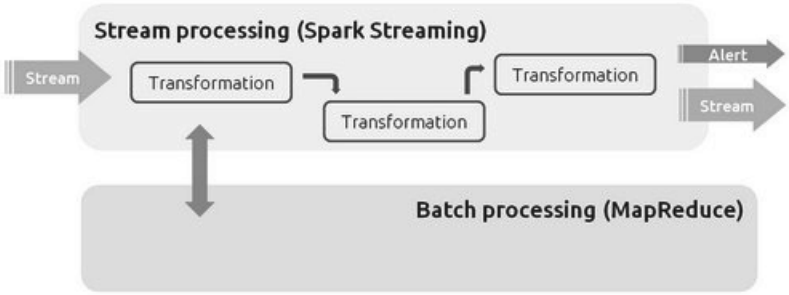


Fig.6: Online algorithms, Topology

Offline algorithms. It analyzes entire data set at once.
- Generally a good fit for Apache Hadoop/Map Reduce
- Model compiled via batch, scored via stream processor

Example: Hypothesis Tests
- Chi2 Test (Goodness of Fit): A feature is anomalous if the data for the latest micro batch (for the last 10 minutes) comes from a different distribution than the historical distribution for that feature
- Grubbs Test: telemetry is anomalous if Z score is greater than the Grubb's score.
- Kolmogorov-Smirnov Test: check if data distribution for last 10 minutes is different from last hour
- Simple Outliers test: telemetry is anomalous if the number of outliers for the last 10 minutes is statistically different then the historical number of outliers for that time frame

Also, some other types of algorithms can be used
- Decision Trees/Random Forests

– Association Rules (Apriori)
– Auto Regressive (AR) Moving Average (MA)

In fact, at using Offlimne algorithms, all analysis is performed as batch tasks, the Streaming part just applies rules compiled by the Batch part. A possible topology Big Picture could be like this

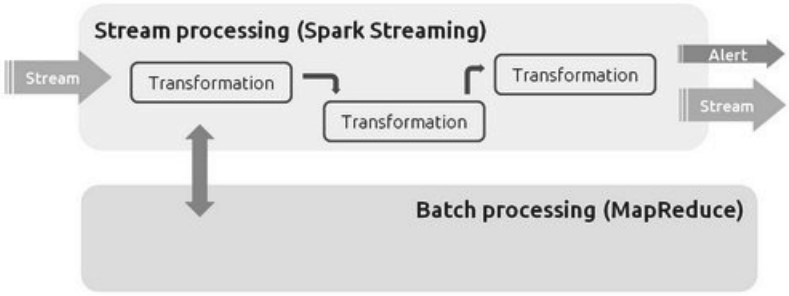


Fig. 7: Offline algorithms, Topology

Technology stack: everything Free

It's very important, that all software to build Analytics based on Machine Learning so as for Data storage is Free/Libre one. All solutions which we are using are publiched under Apache 2.0, MIT and so on. For Streaming part it was used Spark Streaming and Spark for Batch jobs as well. Message Queues which we are using are Kafka and RabbitMQ.

История одного маленького дистрибутива Linux

Денис Пынькин, Минск, Беларусь*

Article presents the creation and support history of the Linux-based infrastructure in ECM department of BSUIR. Evolution and current state of technical decisions used tp create Linux-based infrastructure are described.

Введение

Инфраструктура в образовательных заведениях играет первоочередную роль в обучении студентов самым разным технологиям. Если нет системы, на которой можно потренироваться, понять, что работает как описано, а где возникают «нюансы работы», то никакие видео-обзоры и тонны литературы не заменят реального опыта. Именно поэтому в учебных лабораториях необходимо иметь возможность поработать с различными операционными системами, включая ОС Linux.

Историческая справка

Первые эксперименты по внедрению ОС Linux в обучение на кафедре ЭВМ БГУИР начались примерно в 2002–2003 годах с классического подхода к установке системы параллельно с основной ОС. К счастью, этот подход не прижился — нехватка места на диске, легкость «поломать все», неопытность внедряющих сделали свое дело.

Следующим шагом стала попытка минимизировать простои систем в случае поломок. Хотя задача и была решена «в лоб» с помощью банального восстановления соответствующих разделов, это позволило добиться более-менее стабильной работы обеих используемых ОС, поскольку появилась возможность восстанавливать работоспособную систему в течении одного перерыва.

*denis_pynkin@epam.com, http://lvee.org/en/abstracts/167

Интересным оказался эффект такого подхода. Во-первых, появилась возможность «доверить» студентам права администратора на студенческих машинах, а во-вторых, после введения соответствующего пункта в меню загрузчика, задача восстановления работоспособности машины перекочевала на плечи студентов.

Примерно с 2003–2004 годов начались попытки «поиграться» с сетевой загрузкой — начиная с загрузки самописной системы восстановления и заканчивая полноценной загрузкой систем с развернутым корнем на сервере [1]. Тем не менее, возникли те же проблемы, что и с локально установленной системой и, в дополнение к этому, добавились проблемы с общим разделяемым ресурсом, добавилась необходимость отслеживания состояния этих развернутых систем и невозможность студентам починить систему самостоятельно. А ломались и «ломались» такие системы довольно часто — как случайно, так и намеренно.

Интересным опытом оказалось использование рабочих машин в качестве тонких клиентов вплоть до исключительно ssh-клиента на серверную систему. Это было настоящее раздолье для любопытствующих личностей — начиная от fork-бомб и получения привелегий рута, парализующих работу целой лаборатории, и заканчивая файлами «посмотри-меня-xxx-в-sd-качестве.avi» с установленными executable битами.

Примерно в 2005-2006 годах стало понятно, что от идеальной учебной системы требуется, чтобы она:

- загружалась по сети;
- работала локально на студенческих системах;
- не использовала НЖМД;
- работала в режиме R/O;
- содержала максимально полный набор ПО для разработки;
- имела графическое окружение, не вызывающее желания тут же убежать.

К тому моменту такие системы уже существовали в виде Live-CD и initrd-only систем. Несмотря на все достоинства использования initrd в качестве базовой системы, у нее есть один недостаток, перечеркивающий все плюсы такого подхода — вся система находится в памяти, отъедая этот дефицитный ресурс у полезных программ. Кроме того, появляется жесткое ограничение на размер образа, что уменьшает количество полезного ПО и требует зачастую нетривиальных оптимизаций. Отличие же Live-CD от такой

же системы, но загружающейся по сети, минимально и, фактически, заключается в начальной инициализации системы[2].

Примерно в это же время удалось добиться полноценной работы ОС семейства Windows, что дало возможность отказаться от дисковых систем при создании компьютерных лабораторий на кафедре ЭВМ БГУИР начиная с 2007 года[3].

Дистрибутив

На данный момент архитектура построения stateless-систем, загружающихся по сети [2], уже считается классикой и используется во всех распространенных дистрибутивах.

Разница по большей части заключается в наборе ПО, которое содержится на stateless-системе, и его изначальной конфигурации. Видимо, максимально близко к нужному составу ПО стоит Knoppix, разрабатываемый Клаусом Кноппером, однако все равно необходимо проводить адаптацию к целевой инфраструктуре.

Кроме того, некоторое ПО сугубо специфично для некоторых курсов, читаемых на кафедре ЭВМ, например MPI и Cuda.

Отдельным пунктом хотелось бы упомянуть, что зачастую недостаточно просто установить нужный пакет из репозитория — ему требуется дополнительная настройка для корректной работы: например, OpenMPI требует корректной настройки ssh, а также корректировки настроек для работы с несколькими сетевыми картами.

В дополнение к вышесказанному необходимо отметить, что хотелось бы иметь более-менее современные версии ПО, что приводит к еще одной проблеме — весь состав ПО, все правки постоянно изменяются во времени.

Все вышесказанное приводит к тому, что фактически приходится разрабатывать свой собственный дистрибутив, предназначенный для stateless-работы в учебных заведениях. Естественно, что создание такого дистрибутива с чистого листа крайне ресурсоемкая задача, а так как универсального решения, как и универсального «эталонного» дистрибутива, пока не существует, то необходимо делать выбор в пользу одного из существующих решений.

Чтобы не вступать в дискуссии о том, какой дистрибутив брать в качестве базового для решения этой задачи, просто отмечу, что до начала 10-х годов XXI века далеко не все дистрибутивы задавались вопросом разработки целостной платформы для создания

своих собственных «кастомных» решений. Одним из счастливых исключений является ALT Linux, который используется в качестве базовой системы для построения нужного дистрибутива.

Сейчас в дистрибутивах семейства ALT Linux используется система сборки дистрибутива «mkimage-profiles» [4]; по сути она — набор Makefile'ов, формат которых знаком каждому программисту.

Человеческий фактор

В связи с тем, что у Министерства Образования нет четкой политики по использованию открытого ПО в целом и ОС Linux в частности, в ВУЗах используют, если вообще используют, тот дистрибутив, который нравится местному системному администратору [5]. Поэтому при смене Linux-администратора на кафедре сборка текущего дистрибутива прекратила свое развитие, а создание новой сборки, на новой базе — достаточно долгий, тонкий и кропотливый процесс, отнимающий много времени и сил.

Немаловажным фактором являлась фактически непубличная разработка предыдущей версии дистрибутива. Причин масса, однако главными из них являются стыд и лень: стыдно за те «хаки», которые использовались при кастомизации дистрибутива, а с учетом достаточно большого количества мелких изменений всегда возникает соблазн отложить «генеральную уборку» еще «на чуть-чуть» — до тех пор, пока не станет слишком поздно.

Масла в огонь подлило открытие совместной лаборатории БГУИР и Epam, в которой проводятся занятия по изучению ОС Linux [6]. Внезапно выяснилось, что, помимо ожидаемой разницы между различными дистрибутивами ОС Linux в структуре и работе различных утилит, в рамках даже одного дистрибутива изменений уже достаточно, чтобы сорвать часть занятия.

Таким образом, под влиянием внешних факторов воля была собрана в кулак и началось приведение помойки почти трехлетней давности в порядок, с учетом накопленного опыта.

Новое время — новые вызовы

Первый этап по генеральной чистке и приведению правил сборки в порядок был успешно преодолен с помощью разработчика «mkimage-profiles» Михаила Шигорина.

Но, к сожалению, одной только чистки и облагораживания недостаточно: какое-то ПО развилось до неузнаваемого, какое-то исчезло, появились новые интересные приложения и популярные языки программирования — все это необходимо осознать и интегрировать. Добавились современные системы виртуализации и управления контейнерами. При этом все же желательно уложиться в лимит 4GB, чтобы иметь возможность создавать DVD-образ.

И, наконец, ведется работа по адаптации получившейся системы в сетях ОС Windows, чтобы была возможность загружать ОС Linux в учебных заведениях (и не только) без изменения их инфраструктуры.

Литература

[1] Р.Х. Садыхов, Д.А. Пынькин. Технология построения кластерных систем для образовательных целей. Доклады Международной научной конференции SSA'2004. Минск, 26–28 октября 2004.

[2] Пынькин Д.А, Бездисковые рабочие станции на базе технологий ALT Linux, `http://lvee.org/media/presentations/lvee2008_03-1.pdf`

[3] Д.А. Пынькин, И.И. Глецевич, А.В. Отвагин. Использование бездисковых рабочих станций в образовательном процессе БГУИР // Материалы 4 международной конференции «Информационные системы и технологии» IST'2008. Минск, 2008. С. 310–315.

[4] М.А.Шигорин. Макраме из дистрибутивов: mkimage-profiles. Девятая конференция разработчиков свободных программ: Тезисы докладов / Обнинск, 23–24 июля 2012 года. С.48–49, `http://www.altlinux.ru/media/protva-2012.pdf`

[5] S.S. Derechennik, D.A. Kostiuk, D.A. Pynkin. Free/libre software usage in the belarusian system of higher educational institutions // Друга міжнародна науково-практична конференція FOSS Lviv-2012: Збірник наукових праць/ Львів, 26–28 квітня 2012 р.

[6] Д.А. Пынькин, В.В. Шахов. Обучение Linux в корпоративном секторе. Зимняя международная конференция LVEE'2013. Тезисы докладов. `http://lvee.org/en/abstracts/57`

Модуль инструментальной оценки состояния пользователя

Латий О.О., Костюк Д.А., Брест, Беларусь*

An open hardware project to measure physical state changes of the user while his/her interaction with software is presented. Galvanic skin response, heart rate and blood pressure are used as measured parameters. A schematics is proposed to get these parameters from electric and optical sensors. Arduino platform is engaged in getting data from developed sensors and passing them via USB cable to the receiving software, to store log in CSV format.

Введение

Измерение физического состояния пользователя при работе с программным обеспечением позволяет определить «узкие места» интерфейса гораздо эффективнее, чем такие более типичные методы, как опросы пользователей или составление тестовых заданий и экспертный анализ их выполнения. Более того, для результатов, выдаваемых измерительным устройством, легко выполнить качественное сравнение в разных исходных условиях (графических оболочках, офисных пакетах и др.), не полагаясь на квалификацию usability-эксперта. Как следствие, инструментальная оценка позволяет быстро сформировать набор предложений по улучшению ПО.

Ниже нами представлен разработанный на принципах open hardware аппаратный проект, позволяющий эффективно выполнять такую оценку. Разработка доступна по адресу `https://github.com/fiowro/uxdump`.

Измеряемые параметры

Представляемый здесь модуль одновременно оценивает три параметра: электрическую проводимость кожи (ЭПК), сердечный ритм и относительное изменение кровяного давления.

*`denis_pynkin@epam.com`, `http://lvee.org/en/abstracts/168`

ЭПК варьируется в зависимости от влажности кожи, которая обеспечивается потовыми железами, контролируемыми симпатической нервной системой [1, 2]. По этой причине электропроводность часто используется как показатель психологического или физиологического возбуждения. Однако на результаты измерений ЭПК заметно влияют как внешние факторы (температура, влажность), так и внутренние (воздействие принятых медикаментов). По этой причине измерения ЭПК обычно используются совместно с регистрацией других показателей: сердечного ритма, ритма дыхания, кровяного давления и др. Очевидно, что легче всего регистрировать среди перечисленных параметров сердечный ритм.

При физической нагрузке, изменении эмоционального состояния, а также под воздействием иных факторов частота сердечных сокращений (ЧСС) увеличивается, так как организм человека реагирует на требование органам и тканям повышенного кровоснабжения увеличением сердечных сокращений. Кровяное давление, в свою очередь, является одним из главных показателей здоровья человека, и также известно как индикатор стрессового состояния.

Определение ЭПК, как электрической характеристики — технически простая задача. Есть также несколько несложных способов автоматического определения ЧСС. Наиболее простой в реализации способ основан на принципе фотоплетизмографии (ФПГ), когда информация об изменении объема крови в тканях считывается оптическим методом. Фотоплетизмограф недостаточно точен для получения абсолютной величины объема, но позволяет четко отслеживать его относительные изменения, и потому хорошо подходит для определения интервалов времени.

Похожим способом, по методу определения времени распространения пульсовой волны (ВРП), может быть оценено относительное изменение давления (авторы благодарят Юрия Адамова за указание на данный метод). ВРП обычно определяется как время, затрачиваемое кровью для преодоления расстояния от сердца, с момента ее выброса, до какой-либо точки, обычно пальца. Зная время задержки между пиками на графиках пульса или скорость нарастания пульса, можно оценить изменение кровяного давления.

Описанные принципы измерения доступны для реализации в относительно несложных устройствах, что и послужило побудительным мотивом для создания представленной разработки.

Аппаратная платформа и особенности реализации

В качестве основы для измерительных модулей нами выбрана платформа Arduino [2]. Программирование и обмен данными с ПК выполняется через USB-обертку последовательного интерфейса.

Схема разработанного нами измерительного блока, расширяющего платформу Arduino для совместного измерения ЭПК и ЧСС представлена на рисунке 19.1 (с поправкой на то, что для оценки изменений давления реальное устройство включает не один, а два блока измерения ЧСС). Элементы схемы включают обеспечение электрического смещения ИК-диода, соответствующее электрическое смещение фотодиода, ВЧ-фильтрацию для удаления низкочастотных артефактов движения и дребезга, а также НЧ-фильтр с цепью усиления.Аналоговый сигнал поступает с измерительного блока на АЦП Arduino, передающий цифровые отсчеты на ПК.

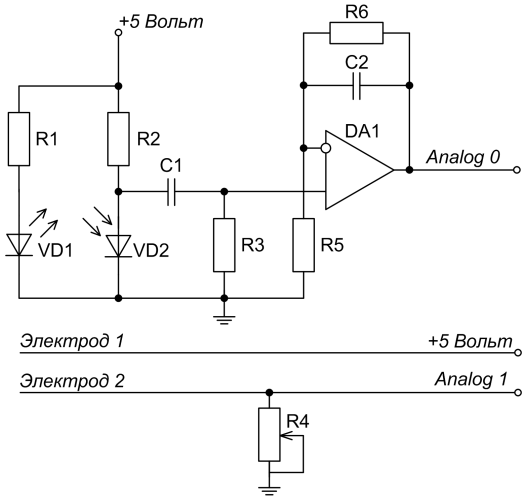


Рис. 19.1. Измерительная подсистема

На рисунке 19.2 можно видеть модель для изготовления корпуса устройства методом 3D-печати.

Для подключения щупов (одного для измерения ЭПК, и двух для ЧСС) применяется обычный аудио-разъем 3.5 мм TRS. Для крепления датчиков на текущий момент используются текстильные застёжки (места креплений можно видеть на рисунке 19.2).

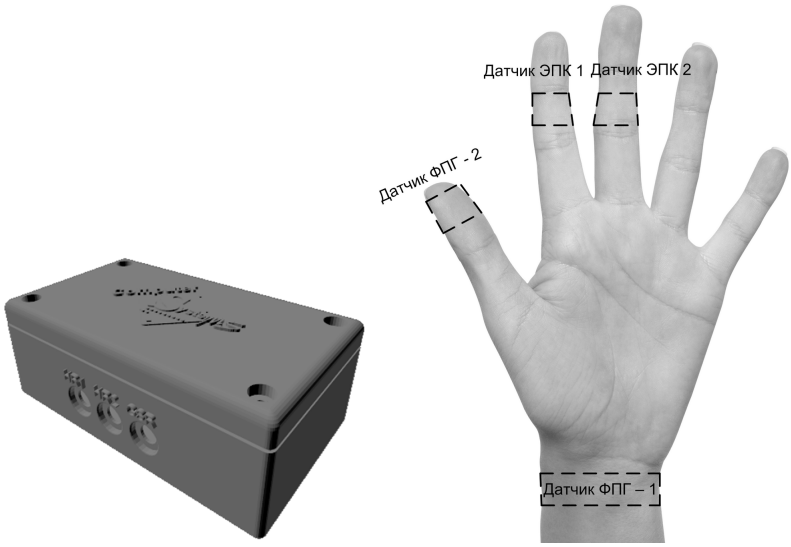


Рис. 19.2. 3D-модель корпуса и схема крепления датчиков

Данные передаются в ПК по шине USB, которая одновременно осуществляет питание устройства. Таблица, формируемая принимающим данные ПО, сохраняется в формате CSV для последующего анализа (рисунки 19.3 и 19.4 демонстрируют иллюстративный экран отрисовки снимаемых кривых и фрагмент формируемой таблицы).

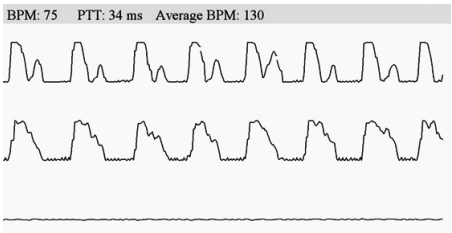


Рис. 19.3. Первичная визуализация средствами processing

	A	B	C	D	E	F	G	H	I	J
1	№	Time	Date	Beats per minu	PTT	Value GSR	Average val	Average va	Average va	Average value
2	391	18:38:54:227529	18.01.2015	73	543	51.0	103	133	133	147
3	392	18:38:54:228291	18.01.2015	78	32	50.0	103	133	133	147
4	393	18:38:55:229104	18.01.2015	74	52	52.0	103	133	133	147
5	394	18:38:56:229949	18.01.2015	71	33	52.0	103	133	132	147
6	395	18:38:57:230719	18.01.2015	79	13	52.0	103	133	132	147
7	396	18:38:57:231433	18.01.2015	83	50	50.0	103	133	132	75
8	397	18:38:58:232234	18.01.2015	76	34	49.0	97	133	132	75

Рис. 19.4. Фрагмент лог-файла

Литература

[1] Kostiuk D.A., Derechennik S.S., Shitikov A.V., Latiy O.O. Approach to evaluate effectiveness of human-computer interaction with contemporary GUI // Третя міжнародна науково-практична конференція FOSS Lviv 2013: Збірник науковых праць, Львів, 18–21 квітня 2013 р. – Львів, 2013. – С. 85–87.

[2] Лацій А.А., Касцюк Д.А. Апаратны модуль ацэнкі стану карыстальніка ПК на базе Arduino // П'ята науково-практична конференція FOSS Lviv 2015: Збірник наукових праць. Львів, 23–26 квітня 2015 р. – С. 64–66.

Некоммерческая 3D-печать

Алексей Бабахин, Рязань, РФ*

3D printing or additive manufacturing is a process of making three-dimensional solid objects from a digital file. The creation of a 3D printed object is achieved using additive processes, when object is created by laying down successive layers of material one by one until the entire model is recreated. Each of these layers can be seen as a thinly sliced horizontal cross-section of the eventual object.

Введение

Наиболее простой доступной для домашнего применения является технология FDM (метод послойного наплавления). Большим толчком для развития домашней 3D-печати послужили различные стадии открытого проекта RepRap, из которых (или на их базе) сейчас выросло большинство конструкций современных принтеров, представленных на рынке. Большинство принтеров можно рассматривать как конструктор, включающий три ортогональные подсистемы:

- станину принтера — его механическую часть, которая может различаться очень сильно от модели к модели;
- контроллер и его прошивку (controller firmware) — управляющий блок принтера, который приводит механизм в движение;
- слайсер (slicer) — программу, конвертирующую 3D-модель в послдовательность движений принтера (в G-код)

Независимо от того, собран принтер самостоятельно из подручных материалов или это — купленная «высокотехнологичная» заводская модель, которая должна (по заявлению производителя) печатать «из коробки» – в той или иной степени принтер все равно остаётся конструктором.

*tamerlan311@mail.ru, http://lvee.org/ru/abstracts/166

Станина

От типа и жесткости исполнения конструкции сильно зависит качество распечатанных изделий, скорость и ускорение, с которыми сможет печатать принтер.

Контроллер

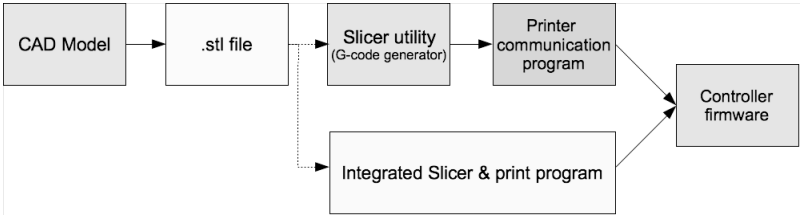
Как правило, это модуль, управляемый восьмибитным микроконтроллером, задачей которого является непрерывное управление всеми элементами принтера: 4–5 шаговыми двигателями, 2–3 нагревательными элементами, опционально вентиляторами и графическим экраном. Программу печати контроллер может получать как с flash-карты, так и по кабелю с компьютера (с помощью более-менее специализированного коммуникационнго ПО).

Слайсер

Программа, которая «разрезает» трехмерную модель формата STL или OBJ по слоям и просчитывает все необходимые движения принтера, выдавая на выходе G-код. В зависимости от сложности модели, процесс может требовать значительных вычислительных ресурсов, а также иметь огромное количество настроек. По этой причине слайсинг требует большого количества экспериментов для получения хорошего результата.

Конвейер 3D-печати

Стадии, которые проходит 3D-модель от ее создания в САПР до печати, представлены на следующей схеме:



Среди популярных слайсеров можно отметить Skeinforge [1] (GPL v.3, изобилие настроек, интерфейс на TCL/TK), его форк

skFrontend [2], а также Slic3r [3] (AGPL v.3, поддержка многопоточности, интерфейс на wxWidgets, считается более эргономичным и быстрым, но менее совершенным, чем Skeinforge). В качестве коммуникационного ПО может использоваться Printrun [4] (GPL v.3). Кроме того, есть несколько интегрированных решений all-in-one, часто создававшихся для конкретного 3D-принтера, но в последствии ставших достаточно универсальными: cura [5] (AGPL v.3, частично использует код Skeinforge для слайсинга), RepetierHost [6] (лицензия Apache, позволяет выполнять слайсинг с помощью Skeinforge либо Slic3r), Repsnapper [7] (GPL, в отличие от предыдущих проектов, преимущественно написанных на Python, данный переписан с нуля на C++ и, уступая в настоящий момент в функциональности, взамен обеспечивает наиболее быстрый слайсинг).

Расходные материалы

Для печати по технологии FDM используются различные пластики в виде круглой нити, намотанной на катушку. Стандартные диаметры — 3 мм и 1.75 мм. Наиболее популярными являются пластики PLA и ABS. PLA считается самым простым в использовании, т. к. не требует обязательного горячего стола, печатается при наиболее низкой температуре и не выделяет вредных веществ в процессе печати. Пластик ABS технологичнее (менее хрупкий, не разлагается со временем), но даёт большую усадку при остывании (детали могут деформироваться в процессе печати), а поэтому требует обязательного подогрева платформы, на которой происходит печать. Помимо PLA и ABS также иногда используются менее популярные варианты: Нейлон, Поликарбонат, Полистирол, Полиэтилентерефталат (PET), различные полимерные смеси имитирующие дерево, бронзу и т. д.

Литература

[1] http://fabmetheus.crsndoo.com
[2] https://github.com/ahmetcemturan/SFACT
[3] http://slic3r.org/
[4] https://github.com/kliment/Printrun
[5] https://github.com/daid/Cura
[6] https://github.com/repetier/Repetier-Host
[7] https://github.com/timschmidt/repsnapper

Голос спонсора: EPAM Systems

Голос спонсора: SaM Solutions

Голос спонсора: Wargaming.net

Голос спонсора: ITS Partner

Голос спонсора: Postgres Professional

Интервью с участниками

По традиции в сборник материалов входят интервью, в которых активные участники сообщества open source делятся своим мнением о свободном ПО, открытых технологиях, роли и месте GNU/Linux, рассказывают, как видят проблематику свободных проектов. В этот раз мы решили расспросить трёх участников конференции, какое-то время назад перебравшихся из Беларуси на территорию Европейского Союза.

1 Александр Боковой — principal software engineer, Red Hat, Эспоо, Финляндия

LVEE: Традиционно первый вопрос — как ты познакомился с открытым ПО?
Александр Боковой: В 1995 году. Я учился на третьем курсе БГПУ им. Максима Танка, и одна из курсовых работ была посвящена фрактальной геометрии. Необходимо было написать приложение, которое бы отрисовывало и в интерактивном режиме позволяло бы исследовать множества Жюлиа для соответствующих точек из множества Мандельброта.
L: Звучит очень наукоёмко...
A: Программу я писал на Паскале, и в какой-то момент стало не хватать стандартной памяти в 16-битном режиме.
L: Под MS DOS.
A: Да. Вариантов использования 32-битного режима было немного, поскольку требовалось еще и приличный интерфейс пользователя обеспечить. Значит, нужна была не только графическая библиотека, но и виджеты, обработка клавиатуры и так далее. И я нашел такую библиотеку — SWORD, написанную французом Эриком Николя на C++ и поставлявшуюся вместе с DJGPP.
L: A DJGPP — это...
A: DJGPP — это первый порт программ проекта GNU на платформу Intel x86, сделанный еще в 1989 году DJ Delorie. Ричард Столлман выступал на встрече Northern England Unix Users Group в компании Data General, где работал тогда DJ Delorie, и на вопрос о переносе GCC под MS DOS, ответил, что это невозможно, поскольку

gcc слишком большая программа, а MS-DOS работает в 16-битном режиме. DJ понял, что это вызов, и принял его. Так что в 1990 мы уже имели компиляторы GNU, Emacs, binutils и много разных библиотек, все под MS DOS в 32-битном режиме — DJ пришлось написать свой DOS Extender для того, чтобы компилировать gcc под MS DOS.

L: Как скоро ты осознал, что это вот — свободное ПО, сообщество, коллективная разработка?

A: Практически сразу. DJGPP поставлялся со всеми исходными текстами, в документации было написано, где можно задавать вопросы. Я подписался на рассылки и первое время просто читал — и переписку, где люди отвечали не только на вопросы об использовании тех или иных компонент системы, но и обсуждали бытовые темы. Выглядело все это очень по-домашнему, а если кто-то предлагал патчи, то это предполагало прежде всего устранение необходимости патчить то же самое место в следующей версии — rsync еще не был написан (он появился только в 1996), а DJGPP распространялся по FTP. На наших узких линиях (64Кбит/с на весь университет) тогда приходилось прежде всего думать, а потом делать.

L: Итак, ты использовал DJGP. А каким образом пользователь СПО стал его разработчиком?

A: Со SWORD и DJGPP я и начал. В 1996 вышла вторая версия DJGPP, независящая от коммерческих компонент для своей пересборки. Главное, что случилось с DJGPP в 1994–96 годах — это взрывной рост популярности, привлекший огромное число терпеливых и общительных людей в списки рассылки. Можно было задавать вопросы и получать ответы на них, вне зависимости от того, насколько плох был твой английский язык. В 1995 году сделали зеркало в рассылку в виде группы USENET `comp.os.msdos.djgpp`, она стала доступна на локальном NNTP-сервере университета.

L: Не помнишь, как вообще оформилась мысль: а сделаюка я публичный патч? Или это получилось как-то незаметно: пообсуждал, поисправлял — и вдруг люди уже пользуются?

A: Непубличные патчи поддерживать было неудобно, поскольку сам комплект DJGPP распространялся в виде архивов. Так что старался отправлять исправления сразу. К тому же, библиотеки были мне нужны для работы, но не являлись главным ее содержимым.

Лицензия SWORD — GNU General Public License, которую я уже читал и видел в применении к остальным компонентам GNU.

В 1998 я вместе с Эриком работал над третьей версией SWORD. Эта работа привела к тому, что через несколько лет я ушел из аспирантуры, так и не закончив свою работу над диссертацией, потому что вместо работы над методикой преподавания фрактальной геометрии сосредоточился над SWORD — нужда в нормальной интерфейсной библиотеке, работающей под MS DOS и GNU/Linux на тот момент еще не отпала, поскольку Qt до 2000 года выходила под неудачной с точки зрения свободного ПО и написания GPL-программ лицензией и не поддерживала MS DOS.

Правда, после ухода из аспирантуры я сосредоточился на сетевых файловых системах, а Эрик переписал SWORD с нуля с учетом прогресса в Qt и проект был перезапущен в 2005: `http://www.erik-n.net/software/sword/`.

На GNU/Linux я перешел где-то в 1996–1997, практически сразу, как появился собственный компьютер.

L: На какой дистрибутив?

A: Начал со Slackware. А в декабре 1999 перевел на белорусский язык программу установки Mandrake Linux. Она вошла в Mandrake Linux Russian Edition, а потом и в основной Mandrake Linux.

Другой проект, который «втянул» меня в себя в приблизительно то же время, это Midgard, система ведения веб-сайтов. Изначально придуманная финнами Генри Бергиусом и Юккой Зиттингом для сайта своего реконструкторского общества в 1998, система переросла викингов и стала довольно успешно использоваться как конструктор различных сайтов, в том числе и для интранетов. Я выступал с докладом о Midgard на первом FOSDEM в 2001 году, а в середине 2000-х даже интегрировал Midgard и Samba для того, чтобы обеспечить прозрачную авторизацию в интранет-приложениях на Midgard в среде Active Directory.

L: И тут мы наконец подобрались к твоему участию в проекте Samba.

A: Получается забавная ситуация: практически все проекты, над которыми я работал и работаю, в той или иной мере связаны между собой. В 2001–2004 годах мы с Игорем Вергейчиком работали над системой хранения, где требовалась поддержка различных сетевых файловых систем, и я столкнулся с необходимостью внести какие-то изменения в Samba. Мы написали ряд патчей, отправили их в рассылку, часть из них приняли, часть — нет. Потом Игорь дора-

ботал Samba до поддержки Unicode. Потом я написал поддержку множественных модулей виртуальной файловой системы. И в 2003 меня пригласили в Samba Team. Принцип был простой: мой код практически не требовал дополнительных доработок, поэтому мне дали прямой доступ к изменению исходного текста.

Когда в декабре 2003 мы получили заказ на разработку поддержки Active Directory в нашей системе хранения, Эндрю Триджелл, создатель Samba, просто сказал нам: «Зачем пытаться добавить патчи в версию 2.0, лучше помогите мне закончить 3.0, где я уже много добился». То есть, взгляды апстрима и даунстрима совпали, получилось сделать многое. Конечно, не в тот срок, который обещал Эндрю, но в 2005 у нас был вполне работающий продукт.

L: Какие различия бросаются в глаза, если сравнивать опенсорс-комьюнити в СНГ с англоязычным?

A: Если тебе нужны какие-то изменения к существующему коду, ты их пишешь, оформляешь патчи, отправляешь в рассылку и обсуждаешь с другими разработчиками. Патчи могут принять сразу, могут не принять совсем, но чаще всего приходится объяснять и находить компромисс. Работа над отдельными изменениями может затянуться на годы. В СНГ есть разработчики свободного ПО (и их много), но очень мало сообществ разработчиков свободного ПО как таковых. Те, кто заинтересован, участвуют в международных проектах различного масштаба. Новые проекты с преимущественно русскоязычным общением – редкость, они мало кому из разработчиков нужны. Они, безусловно, нужны пользователям, но сколько времени разработчики могут посвятить локальным пользователям?

С другой стороны, уровень знания английского языка может препятствовать активному участию в существующих проектах, даже если кто-то готов написать код, часто сталкиваешься с тем, что довести работу до конца они не могут — нужна документация на английском, участие в дискуссиях, причем в темпе активности конкретного проекта, а не разработчика. В этом смысле разница в Европе особенно бросается в глаза, здесь проблем с английским языком среди разработчиков свободного ПО нет, даже в традиционно неанглоязычных странах. Английский — lingua franca свободного ПО.

Другой аспект взаимодействия в проектах свободного ПО, это значительно меньший накал страстей в рассылках по сравнению с тем, что я вижу в русскоязычной среде.

L: О да! В этом сезоне организаторская рассылка LVEE переживала как раз такую драму :)

A: Наблюдается заметное ослабление эмоций при общении с пользователями при продвижении с востока на запад в Европе — если, скажем, польские пользователи еще пишут с активным выражением своей позиции в отношении разработчиков на IRC-каналах, то там, где преобладают английские или американские пользователи, атмосфера менее накалена. В программном обеспечении есть и будут ошибки, никто не идеален, поэтому поиск источника ошибки — рабочая ситуация, не требующая перехода на личности. Почему-то русскоязычное пространство переполнено полярным выражением собственных эмоций.

L: Кстати, возвращаясь к теме работы над продуктами. Как бы ты охарактеризовал свой личный опыт использования СПО в корпоративном секторе?

A: Мне повезло, я последние лет пятнадцать использую свободное ПО в рабочем окружении. В последние пять-семь лет с этим стало совсем хорошо из-за активного продвижения мобильных платформ и веб-приложений, которые вынесли из многих компаний специализированные плагины и прочие платформо-зависимые клиентские компоненты.

Работа над Samba и FreeIPA предполагает, что приходится иметь дело с проприетарной инфраструктурой и клиентским ПО, но для обеспечения собственной жизни в корпоративной среде мне они практически не нужны. Гораздо сложнее с проприетарным ПО на серверной стороне — даже если интерфейс к нему позволяет использовать свободное ПО на клиентской стороне, доступность данных в большинстве таких систем завязана на производителя. Это данных наших компаний, но извлечь их в структурированном виде и перенести куда-то еще мы часто просто не можем.

L: Последний вопрос, о Redhat. Как это выглядит изнутри?

A: По-домашнему. В прямом смысле — большую часть времени я работаю из дома. У нас небольшой офис в Эспоо, рабочее место у меня есть, но появляюсь я в офисе нечасто, поскольку моя команда разбросана по миру. Из инструментов общения — электронная почта, IRC, интернет-телефония и видео-конференции. Раз или два в год получается встретиться лично, это время используется для интенсивных дискуссий, особенно в феврале, когда в Брно (Чехия) проходит традиционная конференция `devconf.cz` — на нее съезжаются ребята из многих команд и есть шанс обсудить предстоящие

задачи на год вперед с теми, с кем не получается пересекаться «в эфире» из-за часовых поясов.

Самым удивительным для меня четыре года назад было то, как мало информации скрыто от посторонних глаз. Если Red Hat участвует в разработке какого-то проекта, то вся информация доступна на сайте апстрима. Внутри только детали планов интеграции конкретных апстримных версий в продукты компании, а весь дизайн новых функций и их разработка ведутся публично.

А моя история, кстати, замкнулась: DJ Delorie работает в Red Hat и обеспечивает нас работающими компиляторами вот уже более шестнадцати лет.

2 Андрей Шадура — software engineer, Collabora, Братислава, Словакия

Андрей Шадура: Моё знакомство со свободным ПО произошло, когда я в школьные времена ещё пользовался DOS и Windows и программировал для них на турбопаскале. Некоторые библиотеки, которыми я пользовался, поставлялись в бинарном виде и без исходных кодов, некоторые были с исходниками и README о том, что коммерческое использование запрещено, а к некоторым прилагался объёмный файл COPYING с текстом лицензии. Примерно в то же время я узнал об альтернативных операционных системах из тогдашних компьютерных журналов, и идея того, что ОС можно «похачить» изнутри меня очень заинтересовала. Позднее я скачал несколько однодискетных дистрибутивов (кроме как по dial-up, мне Интернет был слабодоступен) и поиграл с ними, но «настоящий» Linux я не попробовал до учёбы в университете.

L: И какие были впечатления от этого самого первого опыта, от Unix-подобных систем?

A: Как раз первый из этих однодисковых дистрибутивов и был сертифицированный Unix — демо-версия QNX. Было конечно интересно увидеть что-то совсем другое, и там был такой GUI! Но возможности у этой версии были сильно ограниченными. Затем был очередной однодискетный Linux-дистрибутив, Trinux. Я о нём где-то прочитал, не то в «Компьютерной газете», не то в «Хакере»...

L: Эксперимент прошёл с тем же примерно успехом?

A: Да. А затем меня увлекло местное движение «даунгрейдеров», и я провел несколько лет в окружении FreeDOS, GEM, ViewMAX,

других древних систем и их опенсорсных реинкарнаций. Но, кстати, в DOS меня буквально бесили все тамошние недокументированные функции.

L: Ты имеешь в виду тамошний зоопарк системных вызовов? Все эти int 21h?

A: Да. Ещё в школьные годы я часами в библиотеке просиживал, выуживая прерывания и номера функций из старой литературы и новых журналов. В сравнении с этим, а также недокументированными функциями Windows, жизнь в Linux — просто раздолье.

L: Итак, следующий этап — уже в университете?

A: Это был уже 2005 год, там я получил от Дениса Пынькина, который уже был ALT Linux developer, копию ALT Linux 2.2 Master. Это, кстати, было в преддверии выхода версии 2.4, и он меня уговаривал подождать, но мне хотелось здесь и сейчас, и я на следующий же день проинсталлировал 2.2. Получилось без звука, были проблемы с X-сервером (конечно, без всякого опыта), и это конечно было круто — иметь действующую Linux-систему, но основной ОС она в тот раз для меня не стала.

L: А когда наконец стала?

A: Годом позже. Работал в лаборатории института ядерных проблем, я получил аккаунт на сервере под управлением Debian «sarge», попросил — и мне сделали бутстрап этой инсталляции на мой жёсткий диск. Потом эта машина у меня дома занималась маршрутизацией, пока в 2009 году не заменил ее маленьким роутером на MIPS. Ну а через несколько месяцев общения с этой машиной Debian стал и моей десктоп-системой. И тогда же захотелось как-то контрибутить в проект. Сначала начал делать патчи и баг-репорты к тому, чем пользовался. А в 2009 запакетировал первое приложение.

L: Что это было?

A: Случайное, практически на спор. Кто-то пожаловался, что это очень тяжело — делать пакеты для Debian, я ответил, что нет ничего проще, и услышал в ответ: «Ну давай, запакетируй мой проект, посмотрим, сколько это у тебя займёт времени». И за пару часов подготовил пакет для gdigi.

А потом, уже на LVEE, Дмитрий Бородаенко (в то время — единственный Debian Developer из Беларуси) побудил меня на больший вклад. Первый пакет, который я по-настоящему мэйнтэнил, был tclxml.

Позже, в 2010, занялся исправлением некоторых багов ifupdown, инструмента конфигурирования сети в Debian, ну и так далее.

L: Ты принципиальный Debian'щик?
A: Конечно, не только Debian. Я вообще вношу вклад время от времени в разные свободные проекты, да и свои собственные есть. Кроме того, время от времени вношу правки в википедию, а еще, достаточно регулярно — в OpenStreetMap.
L: Теперь — к переезду. Скажи, отъезд из Беларуси как-нибудь повлиял на твои взаимоотношения с миром свободного ПО?
A: В некотором смысле повлиял: проще, ближе и быстрее стало ездить на всевозможные конференции и прочие спринты и hackweeks. Из событий, на которых я побывал недавно: LinuxDays.cz в Праге (три раза), FOSDEM (два раза), Cambridge Debian Miniconf... Ну и недавняя hack week в Копенгагене.
L: Неполиткорректные соотечественники должны в этот момент воскликнуть «дорвался» :)
A: Ага, так и есть. Но вообще, с кругом общения сложнее. Чтобы было понятнее, я за чуть более, чем три года в Словакии переезжал два раза. Первое время здесь я жил в деревне, с кругом было вообще никак.

Но я активно этот круг искал в других местах. Например, познакомился с местным сообществом OpenStreetMap (Freemap.sk) и через две недели после переезда поехал с ними на mapping party. Общаться было сложновато, потому как по-английски я хоть и говорю, но мне хотелось научиться говорить по-словацки, а знания были очень слабы. А по-белорусски меня понимали слабо :)
L: Все эти переезды были связаны с работой?
A: Да, одна закончилась, новая находилась в другом месте. Через год, кстати, я приехал на университетскую конференцию OSSConf в Жилине, о которой узнал случайно, и познакомился там с кучей сторонников free software.

Да, еще забыл. Через пару месяцев после mapping party в результате активных поисков я узнал, что в Братиславе есть хакерспейс Progressbar, и направился туда на одно из мероприятий. мероприятий там вообще много проводилось, собирались какие-то питонисты, опенстритмапперы и прочие, но поездка туда занимала бы 5 часов в одну сторону, поэтому часто посещать их не получалось, пока я не переехал в конечном итоге в Братиславу.
L: Вопрос по поводу членов опенсорс-комьюнити: какие-то отличия после переезда? Бросалась в глаза какая-то разница?

A: С одной стороны, здесь я заметил, что линуксами, в основном Ubuntu, пользуются иногда люди, далекие от IT вообще. И это меня удивило.

L: Ну да, у нас это обычно члены семей линуксоидов.

A: Из примеров вспоминается одна знакомая, которая мне рассказывала о том, какая замечательная Ubuntu и какой ужасный Debian. При этом она ни разу в жизни, как мне кажется, не видела командную строку ни одного, ни другого. Её работа вообще связана с образовательными программами...

L: Ты говоришь, что это с одной стороны. А с другой?

A: С другой стороны, пассивность «активистов». В Жилине, где я жил, есть некоторое количество людей, пользующихся Linux и знающих про свободное ПО (часть из них работает в местном университете). И за целый год проводится одно, максимум два события на тему: тот самый OSSConf, и иногда OSS Weekend. Это при том, что в городе вроде как третий по величине технический университет страны...

Ну да ладно, Жилина, маленький городок. Берем Братиславу. Здесь есть STU, Словацкий технический университет, здесь есть Progressbar. Progressbar организует небольшие митапы иногда, но нерегулярно. Я предлагал создать регулярные встречи, подобие минских линуксовок. Никто энтузиазма не проявил, но один человек рассказал, что он когда-то пробовал делать какие-то встречи, но всё угасло.

Подобный вопрос я поднял на OSS Weekend, который в прошлом году (и в этом тоже) проводился в Братиславе. «Надо бы...» был ответ :)

Но, с третьей стороны, как ни странно, местное Ruby-сообщество достаточно активное. Встречи проводят несколько раз в месяц, называются Рубислава.

3 Евгений Калюта — experienced developer, Ericsson, Хельсинки, Финляндия

L: Традиционный первый вопрос — твое первое знакомство с открытым ПО. Может быть первые впечатления, если они были?

Евгений Калюта: Я расскажу долгую историю :)

L: Отлично :)

E: Я из провинции. Доступность как информации, так и техники тогда была не на высоте. Учитель информатики у нас был молодой, активный, сразу после института. Это был 7-ой класс, когда нам поставили «Корветы».

L: Действительно, издалека :)

E: Программы обучения толком не было, нас в класс пускали, но директор строго говорила «седьмому классу только игры». Однако учитель некоторым пытливым показал книжки по Basic и давал основы алгоритмизации (в моём классе нас таких пытливых было двое). Он же (учитель) как-то рассказал, что для настоящего программирования бывает ассемблер (что это я тогда представлял с трудом), и C.

L: Этого хотелось?

E: Этого очень хотелось. Но книг в доступности не было (начало девяностых).

Однажды в книжном я таки увидел какую-то брошюрку, то ли про C, то ли про что-то ещё, но главное, что в предисловии было замечено, что вот такой вот он язык C, и на нём написали Unix, на котором работает Интернет.

Очень захотелось как C, так и Unix. При мысли о них в душе возникал некий трепет.

Заработать на первый PC мне удалось кажется на третьем курсе. Где-то в это время, кажется в «Компьютерной газете», пробежала статья с заголовком «Попробуйте Linux». Это был Unix, этого хотелось. Плюс мысль о том, что можно посмотреть в исходный код настоящего ядра настоящей операционной системы, вызывала ощущения на грани....

L: Напишем, что мысль вызывала катарсис.

E: Хорошо :) Но этого негде было взять (из моего круга общения, ясное дело, который на тот момент охватывал не очень много людей, приобщённых к IT). Первый диск, привезённый одной компьютерной фирмочкой, нёс на себе две безнадёжно испорченные версии дистрибутива «Caldera», ни одна из них не могла поставиться по объективным причинам.

L: Из-за неумелой перепаковки?

Ну как, если правильно подмонтировать распакованный tar.gz одиного из них как umsdos, то может шансы и были бы, но я тогда я не имел об этом ни малейшего представления. Я потрогал консоль инсталлятора, смог даже перенести файл на ДОС-раздел, испытал... катарсис, понятное дело, ну и как бы на этом всё.

L: А твой первый работающий Linux?

E: Первым работающим оказался русский клон Redhat 4.2 — он назывался «Красная шапочка 5.0», он умел ставиться, он умел грузиться, на нём собиралось ядро и, если мне не изменяет память, KDE 1.0 (к тому моменту у меня уже были контакты, у кого это можно было взять).

Подытоживая, пришёл к открытому ПО я случайно (я о нём ничего не знал) из желания приобщиться к великому, к Unix, и (за неимением) других вариантов не искал.

L: Несколько слов про твой путь из пользователей свободного ПО в разработчики?

E: Ну, вообще контрибуций у меня не очень много. В детстве я был «хорошим советским мальчиком» и очень боялся публичного порицания. Поэтому долгое время в «серьёзные» проекты было лезть страшновато... Что очень зря. С большего, по отношению к открытым проектам, это прошло только пару лет назад. А с Debian в период большого желания просто случился неприятный казус, который затормозил мой путь в Debian Developer.

L: У нас в этом году снова тематическое интервью. Поэтому еще группа вопросов, инспирированная отъездом интервьюируемого из Беларуси. Какие различия бросаются в глаза, если сравнивать опенсорс-комьюнити в СНГ с англоязычным? Европейских и белорусских (и вообще русскоязычных, наверное) разработчиков?

E: Про англоязычные комьюнити особенно говорить бессмысленно, ибо белорусы — такие же полноправные участники этого коммунити. Сами и всё видят, и несут вклад в общую атмосферу.

L: Ну, речь скорее о локальных комьюнити.

E: Локального, про Финляндию могу сказать чуть личного.

L: Очень хорошо.

E: Я бы отметил, что различные открытые проекты тут занимают видимую часть общественной жизни — тут вообще модны всевозможные общественные обсуждения и инициативы). Участие студентов в opensource очень естественно: помню, поразился количеством Debian Developers.

L: Если подумать, не только Debian — в конце концов, происхождение Linux как такового...

E: Финляндия дала миру open source кроме ядра Linux и некоторые менее известные вещи, такие как протокол IRC, клиент irssi,

оконный менеджер ion (славящийся своим проблемным автором), протокол ssh, почтовый сервер dovecot — это навскидку.

Местные заинтересованные вполне чувствуют себя частью мирового движения, активно участвуют в проектах, конференциях, инициативах по всему миру, устраивают их у себя. Первый мой debconf был в Финляндии, пару раз был на дебиановских bug squashing party, они тоже как правило не совсем локальные. Можно, наверное, сказать, что открытости в сообществе порядочно поболе.

А еще для пущего приобщения студентам в плюс к традиционным конференциям устраивают различного рода встречи и круглые столы с известными в мире open source людьми, и они как правило не ограничиваются студентами. Я был на лекциях Столлмана и Торвальдса (на той самой, про Nvidia).

И потом, обсуждать вопросы с местными ребятами мне лично очень приятно — это, как правило, спокойно, по делу, без излишнего давления.

В общем, все это очень повлияло и на личную систему «свой — чужой». Она слабо коррелирует с границами и языками.

Кроме того, на момент переезда (2006 год), проникновения IT в общественную жизнь было порядочно больше, чем в Беларуси, поэтому и восприятие околоайтишных движений более серьёзно.

В остальном точно так же: лобби коммерческих компаний имеет больший вес. Слышал истории об оспаривании некоторых государственных тендеров, на что банально не хватило денег.

L: Еще интересный вопрос — твой личный опыт использования СПО в корпоративном секторе. Понятно, всегда есть работодатель. . .

E: Используется активно, там где не противоречит коммерческим интересам. Но, как мы знаем, построить коммерческую систему на базе свободного ПО и поддержки комьюнити у Нокии не получилось. Вклад она при этом внесла очень порядочный, надо отметить.

В Эриксоне, безусловно, оно тоже используется в разных местах, и даже что-то выползает наружу.

L: В смысле наработок, которые отдаются сообществу?

E: В целом, отдавать из корпорации назад обычно сопряжено с трудностями. Как правило, это связано с законодательством США и нежеланием рисковать, ну или и простая жадность иногда. Но

есть и позитивные случаи: на память сразу приходят TIPC[1] и Eclipse.

А если про компании вообще, уже не из личного опыта — понятно, раньше Nokia и окружающие её компании задавали тон, но и сейчас тут присутствует некоторое количество компаний, серьёзно вкладывающих в разработку открытых проектов: это небольшой (по их меркам, в несколько сотен) офис OTC Intel, это наверное наиболее «честный» контрибьютор, но есть Huawei, Samsung, (члены Linaro), Nvidia опять же. В определённом смысле присутствуют Red Hat и TI.

[1] `http://en.wikipedia.org/wiki/TIPC`

20 YEARS
OF CRAFTING SOFTWARE

ENGINEERING IS IN OUR DNA.

Since 1993, we've helped the world's leading companies imagine, design, engineer, and deliver software that changes the world. Today, we are more than developers. We think globally, but act locally to help our customers build their brands.

EPAM is committed to being more than good for our employees and our customers. We believe in being good for the communities in which we operate and in driving positive change for the world at large. Our people are world class engineers and our results speak for themselves.

For more information visit epam.com

In operation since 1993	50+ active clients
Headquartered in Munich, Germany	600+ employees
Development centers in Eastern Europe	Regional offices in the Netherlands, USA

OUR EMBEDDED DEVELOPMENT

- A broad range of project types involving embedded software devel
- Manual and automated testing on all software development phase system and acceptance tests)
- Solutions for a variety of platforms and architectures (x86, ARM, DSP, AVR8/32, RISC, CISC, SHARC, PLCPIC, etc.)
- Considerable know-how accumulated over the years, own components, tried and tested code base
- Network protocols experience
- Management and control of industry-standard servers
- Mature development methodologies (source code/documentation control, change management, defect tracking and continuous integration)
- Cross-platform development tools
- Porting of kernels, bootloaders and BSP (Board Support Packages) adaptation and extension
- Databases, including small footprint databases
- Porting of Android to new architectures and platforms
- Creation of device drivers
- Cross-architecture migration, e.g. from RTOS's to embedded Linux environment
- Device compatibility verification for Android Compatibility Defi

PLATFORMS & CPU ARCHITECTURES:

- **ARM**
  - FreeScale (i.MX*)
  - Texas Instruments (OMAP 4/5, AM335X)
  - NVidia (Tegra)
  - Samsung (S3Cxxxx)
  - Marvell
  - QUALCOMM (Snapdragon)
  - M0, M3 and others
- **X86**
  - Intel
  - VIA
  - AMD
- **PowerPC**
  - FreeScale (ex-Motorola)
  - AMCC (ex-IBM)
  - IBM
- **AVR**
- **PIC**

EMBEDDED SOFTWARE DEVELOPMENT AND TESTING FOR:

Network equipment (NAS, routers, Wi-Fi APs, mobile communications equipment)

Onboard equipment and infotainment systems

Medical equipment

POS and warehouse equipment

Experimental and scientific equipment

Development boards and kits

Extensive control of servers regardless their system status

Consumer electronics (e-books, set-top boxes, GPS navigation systems, media boxes, smartphones and tablets)

KERNEL • BOOTLOADERS, EMBEDDED DEVICE DRIVERS AND FILE SYSTEMS • APPLICATIONS • SYSTEM LIBRARIES • MIDDLEWARE • GUI

WARGAMING.NET
LET'S BATTLE

ОБЪЕДИНИ СВОЮ МЕЧТУ С НАШЕЙ!

МЫ – КОМПАНИЯ, ПРОРУБИВШАЯ ОКНО К УСПЕХУ НА РЫНКЕ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ИГР.

МЫ – ЭТО БОЛЕЕ 3500 ВЫСОКОКВАЛИФИЦИРОВАННЫХ СПЕЦИАЛИСТОВ И 16 ПРЕДСТАВИТЕЛЬСТВ ОТ СИДНЕЯ ДО СИЭТЛА.

МЫ – ЭТО СПЛОЧЕННАЯ КОМАНДА ЕДИНОМЫШЛЕННИКОВ, МЕЧТА КОТОРОЙ – ОБЪЕДИНИТЬ ФАНАТОВ ГЕЙМДЕВА ПО ВСЕМУ МИРУ!

ЕСЛИ ТЫ СТРАСТНО ЛЮБИШЬ ИГРЫ, ТВОРЧЕСКИ АКТИВЕН И МЕЧТАЕШЬ РАБОТАТЬ В ЭТОЙ СФЕРЕ – ПРИСОЕДИНЯЙСЯ К НАМ!

ВМЕСТЕ С WARGAMING ТЫ СМОЖЕШЬ:

- СТАТЬ НЕОТЪЕМЛЕМОЙ ЧАСТЬЮ НАШЕГО УСПЕХА
- ОБРЕСТИ УНИКАЛЬНЫЙ ОПЫТ РАБОТЫ С ЛУЧШИМИ ПРОЕКТАМИ ГЕЙМДЕВА
- БЫТЬ УВЕРЕННЫМ В ЗАВТРАШНЕМ ДНЕ И ПОСТОЯННО РАЗВИ ВАТЬСЯ

МЫ НАЦЕЛЕНЫ НА ДОСТИЖЕНИЕ НОВЫХ ВЫСОТ И ПРЕДЛАГАЕМ ВАМ ПОКОРИТЬ ИХ ВМЕСТЕ С НАМИ!

УЗНАЙ БОЛЬШЕ: WWW.WARGAMING.COM

Software Outsourcing Company
www.itspartner.net

ИТС Партнёр – **кто мы такие**?

Мы - белорусская компания, на рынке с 2009 года, резидент Парка Высоких Технологий

Разрабатываем:

- встроенное ПО для систем хранения данных и роутеров;
- веб-сервисы & приложения;
- мобильные приложения;

40+ сотрудников и растем, 70% разработчиков с опытом более 5 лет.

Более 1 миллиона устройств с разработанным нами встроенным ПО продаётся ежегодно.

Самый крупный проект – более 25 000 человеко-часов.

Стабильная команда профессионалов.

Чем интересны?

Наши проекты на C++:

Ready NAS OS – встроенное ПО для устройств (класс SMB) хранения данных

Ready Data – встроенное ПО для устройств (класс Enterprise) хранения данных: платформа для единого хранения данных (NAS/SAN) на базе технологий корпоративного класса для бизнеса любого уровня.

Arlo – встроенное ПО для камер видеонаблюдения. Определение и отслеживание движения, обучение и распознавание образов: люди, домашние животные и машины. Основано на библиотеке OpenCV. Подробонсти пока засекречены, проект еще не вышел на рынок.

Плюшки и бонусы:

- Возможность карьерного роста
- Работа в распределённых командах
- Регулярные командировки в Европу и США
- Оплата профессиональной литературы, конференций, и других вариантов повышения квалификации
- Бесплатный английский в офисе и за его пределами
- Компенсация стоимости спортивных занятий
- Чай\кофе\сливки\печенье\фрукты
- Интересные корпоративные праздники
- Подарки детям сотрудников на Новый год
- И конечно же традиционный Pizza Day☺: каждую 2ю пятницу месяца заказываем пиццу или суши или еще какую-нибудь вкусняшку за счет компании + готовим интересную тему для обсуждения и здорово проводим вместе время.

А кто *вы*?

Пишите: info@itspartner.net

Звоните: +375 25 6922319

Будем рады знакомству с вами ☺

Postgres Professional

Postgres Professional – российский разработчик и вендор PostgreSQL. Компания основана в 2015 году российскими разработчиками PostgreSQL О.Бартуновым, А.Коротковым и Ф.Сигаевым, прикладным разработчиком И. Панченко и основателем Nvision Group А.Сушкевичем.

Компания участвует в разработке PostgreSQL по нескольким ключевым направлениям (кластер, pluggable storage engines, слабоструктурированные данные, полнотекстовый поиск, оптимизация и др.) и оказывает различные коммерческие услуги на базе PostgreSQL, включая 24-часовую техническую поддержку, помощь в миграции прикладных систем, консалтинг и разработку, занимается продвижением PostgreSQL в России, проводит образовательные и научно-практические мероприятия, в частности, конференции PgConf.Russia.

Компания привержена принципам Open Source и прилагает усилия к публикации своих разработок.

http://postgrespro.ru/

PROFESSIONAL Postgres

PostgreSQL – свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая открытая СУБД в мире, являющаяся реальной альтернативой коммерческим базам данных.

Основные возможности PostgreSQL

- **НАДЕЖНОСТЬ И УСТОЙЧИВОСТЬ PostgreSQL**
  Надежность PostgreSQL является известным и доказанным фактом на примере многих проектов, в которых PostgreSQL работает без единого сбоя и при больших нагрузках на протяжении нескольких лет.
- **КОНКУРЕНТНАЯ РАБОТА ПРИ БОЛЬШОЙ НАГРУЗКЕ**
  PostgreSQL использует многоверсионность (MVCC) для обеспечения надежной и быстрой работы в конкурентных условиях под большой нагрузкой.
- **МАСШТАБИРУЕМОСТЬ**
  PostgreSQL использует современную архитектуру многоядерных процессоров - его производительность растет линейно вплоть до 64-х ядер. Кластерные решения на основе PostgreSQL-XL обеспечивают горизонтальную масштабируемость.
- **КРОССПЛАТФОРМЕННОСТЬ**
  PostgreSQL поддерживает все виды Unix, включая Linux, FreeBSD, Solaris, HPUX, Mac OS X, а также MS Windows.
- **РАСШИРЯЕМОСТЬ**
  Расширяемость PostgreSQL позволяет добавлять новую функциональность, в том числе и новые типы данных, без остановки сервера и своими силами.
- **ДОСТУПНОСТЬ**
  PostgreSQL распространяется под лицензией BSD, которая не накладывает никаких ограничений на коммерческое использование и не требует лицензионных выплат. Вы можете даже продавать PostgreSQL под своим именем.
- **НЕЗАВИСИМОСТЬ**
  PostgreSQL не принадлежит ни одной компании, он развивается международным сообществом, в том числе и российскими разработчиками. Независимость PostgreSQL означает независимость вашего бизнеса от вендора и сохранность инвестиций.
- **ПРЕВОСХОДНАЯ ПОДДЕРЖКА**
  Сообщество PostgreSQL предоставляет квалифицированную и быструю помощь. Также коммерческие компании предлагают свои услуги по всему миру.

Технические детали PostgreSQL

- Высокий уровень соответствия ANSI SQL 92, ANSI SQL 99 и ANSI SQL 2003, 2011.
- Интерфейсы для Tcl, Perl, C, C++, PHP, Json, ODBC, JDBC, Embedded SQL in C, Python, Ruby, Java, ...
- Интеграция защиты данных с операционной системой (SE-Linux).
- View (materialized), sequences, inheritance, outer joins, subselects, referential integrity, window functions, CTE (WITH queries).
- Продвинутый планировщик запросов позволяет оптимизировать сложные запросы.
- Поддержка пользовательских функций, процедур, триггеров.
- Процедурные языки pl/PgSQL, pl/Perl, pl/Python, pl/V8, pl/Java и другие.
- Расширяемый набор типов данных с поддержкой индексов (GiST, GIN, SP-GiST).
- Встроенная гибкая система полнотекстового поиска с поддержкой русского и всех европейских языков.
- Поддержка NoSQL: слабо-структурированные данные (xml, json, jsonb).
- Горячее резервирование и репликация (синхронная, асинхронная, каскадная), PITR.
- Полная поддержка ACID и эффективной сериализации транзакций.
- Функциональные и частичные индексы.
- Интернационализация, поддержка Unicode и locale.
- Загружаемые расширения, например, поддержка геоинформационных данных POSTGIS, нечеткий поиск с помощью триграм, эффективная работа с массивами.
- Поддержка SSL и Kerberos аутентификации.
- Foreign Data Wrappers (writable), поддержка всех основных баз данных.

Имея более чем 20-летнюю историю развития, одно из самых больших и квалифицированных сообществ в мире и отличную репутацию надежного и высококачественного продукта, PostgreSQL поможет вам выполнить ваш проект без проблем!

http://www.postgresql.org

Научное издание

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов одиннадцатой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2015

(Гродно, 25–28 июня 2015 г.)

Фото на обложке Арсения Случевского

Ответственный редактор Д.А. Костюк
Компьютерная верстка А.О. Шадура

Подписано в печать 18.06.2015.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная.
Ризография. Усл. печ. л. 6,9. Уч.-изд. л. 6,5.
Тираж 200. Заказ 1847.

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.