АНО «Институт логики, когнитологии и развития личности»
ALT Linux

Десятая конференция разработчиков свободных программ

Калуга, 20–22 сентября 2013 года

Тезисы докладов

Москва,
Альт Линукс,
2013

УДК 004.91
ББК 32.97

Десятая конференция разработчиков свободных программ: Тезисы докладов / Калуга, 20–22 сентября 2013 года. М.: Альт Линукс, 2013. — 84 с. : ил.

В книге собраны тезисы докладов, одобренных Программным комитетом десятой конференции разработчиков свободных программ.

ISBN 978-5-905167-14-0

Программа конференции

20 сентября

14:30-15:30 Регистрация участников

Дневное заседание
15.30–18.40

15.30 А. Е. Новодворский. Вступительное слово

15.50–16.20 Д. В. Силаков

ROSA Updates Builder — автоматическое обновление пакетов из апстрима 6

16.20–16.50 А. С. Черепанов

Прошлое, настоящее и будущее школьного комплекта ALT Linux 9

16.50–17.20 Ф. С. Занько

О свободных форматах публикации результатов научных исследований 11

17.20–17.40 Кофе-пауза

17.40–18.10 А. Г. Михеев

Реализация алгоритма проверки ограниченности количества точек управления в свободной системе управления бизнес-процессами и административными регламентами RunaWFE 15

18.10–18.40 А. И. Бодренко, И. И. Бодренко

Система видеосвязи для невидимого интернета 21

21 сентября

Утреннее заседание
10.00–13.20

Дневное заседание
14.50–18.10

22 сентября

Утреннее заседание
10.00–13.50

10.00-11.00 Мастер-класс И. Власенко

11.00-12.00 Мастер-класс И. Воронин

12.00–12.20 Кофе-пауза

12.20-13.50 Обсуждение

Вне программы

Денис Силаков
Москва, РОСА

Проект: Updates Builder
http://wiki.rosalab.ru/ru/index.php/Updates_builder

ROSA Updates Builder — автоматическое обновление пакетов из апстрима

Аннотация

Разработчики многих современных дистрибутивов стремятся предоставить пользователям как можно более широкий выбор ПО и собирают для своих систем тысячи пакетов с различными программными компонентами. Но мало собрать пакет с той или иной программой; необходимо поддерживать его в актуальном состоянии — в частности, вовремя обновляться на новые версии из апстрима. Помноженное на частые релизы в апстриме, большое количество пакетов в дистрибутиве делает эту задачу достаточно ресурсоемкой. Однако процесс обновления пакета во многих случаях тривиален и сводится к замене архива с исходным кодом. Поэтому представляется разумным автоматизировать задачу мониторинга выхода новых версий в апстриме и их сборки в дистрибутив. Данный доклад посвящен инструменту Updates Builder, используемому для этих целей в РОСЕ.

Количество пакетов в современном дистрибутиве общего назначения исчисляется тысячами. Но далеко не все из них монстры наподобие LibreOffice, сборка и обновление которых может потребовать серьезных усилий. Большинство пакетовэто небольшие программы, библиотеки, модули интерпретируемых языков и прочие компоненты, которые собираются в дистрибутив с минимумом патчей, буквально парой инструкций. Например, ROSA Desktop Fresh R1 содержит около 1,500 пакетов с модулями texlive, более 2,000 модулей Perl, сотни дополнительных пакетов для R и так далее.

Минорные обновления многих подобных компонентов выходят достаточно часто. Как правило, обновление пакета при этом сводится к упаковке новой версии и базовой проверке того, что обновленный пакет устанавливается и им можно пользоваться. В случае одного пакета это недолго — скачать и запаковать новую версию можно за пять минут. Однако на тысячу пакетов уйдет несколько рабочих дней. При

этом простая перепаковка новых версий — занятие нудное и рутинное, и тратить на него силы людей не представляется рациональным. К тому же появление новых версий в апстриме надо своевременно отслеживать, и делать это вручную — тоже не очень разумно.

Для облегчения жизни мэйнтейнеров, использующих среду сборки ABF (в настоящее время — прежде всего разработчиков ROSA и OpenMandriva), мы разработали автоматический сервис отслеживания и сборки новых версий ПО. Сервис состоит из двух компонентов — инструмента мониторинга апстрим-версий и инструмента автоматической сборки этих версий на ABF.

В роли первого инструмента выступает Upstream Tracker, который в настоящее время также развивается РОСОЙ. Для целей сервиса используется его часть, отслеживающая актуальность версий пакетов в РОСЕ и OpenMandriva — Updates Tracker, который осуществляет мониторинг апстрима и всегда располагает сведениями о свежих версиях ПО.

Второй инструмент — это утилита Updates Builder. берущая на вход имя пакета, запрашивающая Upstream Tracker на предмет наличия новой версии в апстриме и в случае наличия таковой, пытающаяся ее собрать на ABF. Перед сборкой в Git-репозитории соответствующего проекта на ABF создается отдельная ветка, с которой и работает Updates Builder. В spec-файле в этой ветке обновляются версия пакета и версия архива с исходным кодом и сбрасывается релиз. Новый архив с исходным кодом помещается в файловое хранилище ABF (в отличие от многих систем сборки, ABF хранит бинарные файлы на отдельном файловом сервере, а в Git помещается только ссылка на нужный файл).

На основе данных из созданной ветки Git осуществляется попытка собрать обновленный пакет. Сборка производится в отдельный контейнер, без публикации пакета в какой-либо репозиторий.

Мэйнтейнеру пакета отсылается уже письмо о результатах сборки. В случае успеха он может сразу переходить к проверке функциональности обновленной программы, и если его все устраивает, то перенести новую версию из вспомогательной ветки Git в ветку, соответствующую целевому репозиторию. В дополнение к этому, в случае успешной сборки Updates Builder автоматически формирует Pull Request на перенос обновлений в основную ветку Git — так что мэйнтейнеры могут быстро просмотреть предлагаемые изменения и согласиться с ними нажатием одной кнопки.

Updates Builder уже более полугода успешно используется в POCE и OpenMandriva для отслеживания обновлений нескольких тысяч пакетов. Практика его использования показывает, что один человек вполне в состоянии обрабатывать до нескольких десятков пакетов за неделю, не сильно отвлекаясь от других занятий. Это позволяет повысить эффективность участия в поддержке дистрибутива людей, которые могут выделить на такое участие достаточно ограниченное количество времени.

Одним из опасений, высказываемых в отношении подобного сервиса, является соблазн полностью переложить процесс обновления пакетов на роботов, что может негативно сказаться на качестве системы — ведь ошибки будут выявляться уже после помещения пакетов в репозитории. В POCE с такой проблемой борются регламентными мерами — обновления пакетов в main-репозиторий уже выпущенных релизов, а также разрабатываемых релизов на стадии бета-тестирования, обязаны проходить через команду QA. И в интересах мэйнтейнера перед отправкой обновления на проверку удостовериться в его корректности.

Кроме того, полностью автоматическое обновление с использованием Updates Builder возможно только в случае относительно небольших изменений, которые вряд ли нарушат совместимость с предыдущей версией. В случае серьезных изменений (например, нового soname у библиотеки), мэйнтейнерам все равно придется вмешаться. И в таких случаях все зависит от их добросовестности — ограничатся ли они простым изменением soname в spec-файле или честно изучат проблемы, которые может такое изменение привнести. В конце концов, сервис избавляет их от изрядной доли рутинной работы — так почему бы не потратить освободившееся время на более тщательную проверку новых версий?

Черепанов Андрей Степанович
Москва, ALT Linux

Прошлое, настоящее и будущее школьного комплекта ALT Linux

Аннотация

Доклад посвящён истории разработки школьных комплектов ALT Linux и планам разработки комплекта Информика 7.0 Школьный.

История дистрибутивов ALT Linux, предназначенных для общеобразовательных школ, началась в 2001 году с выпуском ALT Linux 1.0. После этого были выпущены версии 1.1 и 2.0. Очередной этап разработки продуктов для школ начался в 2007 году с разработкой ПСПО (Пакета свободного программного обеспечения) по заказу Минобрнауки. Комплект, выпущенный в 2008 году, включал в себя три вида дистрибутивов, предназначенных для разного уровня компьютеров, терминального сервера на базе LTSP, дисков с документацией, в том числе и методическими материалами. В 2009 году уже по собственной инициативе выпускается Альт Линукс 5.0 Школьный, дополненный серверным дистрибутивом с MediaWiki, Moodle, электронным журналом РУЖЭЛЬ и сервером контентной фильтрации. Кроме того, в комплекте появились диск со свободным программным обеспечением под Windows и диск в видеоуроками, сделанными представителями сообщества. В 2012 году совместно с ФГАУ ГНИИ ИТТ «Информика» выпускается комплект школьных дистрибутивов на базе Шестой платформы. Теперь он уже состоит из одного серверного и трёх дистрибутивов для рабочих станций (один из которых предназначен для учителя).

В настоящий момент ALT Linux совместно с ФГАУ ГНИИ ИТТ «Информика» готовят новый комплект школьных дистрибутивов. Он, как и предыдущий комплект, состоит из четырёх дистрибутивов (одного серверного, десктопных дистрибутивов для учителя и для учеников с XFCE, а также десктопного дистрибутива для мощных компьютеров на базе KDE4. Перед комплектом ставились следующие задачи:

- Нулевая стоимость пользовательских лицензий.
- Возможность централизованного управления учебным классом.

- Централизованное управление аутентификацией через сервер каталогов.
- Прозрачное использование сетевых ресурсов в режиме Single Sign-On.
- Возможность сетевой загрузки бездисковых клиентов с сохранением данных на сервере.
- Дистрибутивы должны содержать достаточное количество свободного программного обеспечения для организации учебной деятельности по образовательным стандартам.
- Комплект должен быть максимально локализован на русский язык.

Планируемые изменения в седьмой версии:

- Обеспечена возможность установки в режиме UEFI (для 64-разрядных версий).
- При установке профиля сервера Информика 7.0 Школьный Сервер используется система инициализации SysVinit, при установке этого дистрибутива как рабочей станции и остальных дистрибутивов комплекта — Systemd. При выборе групп пакетов можно поменять систему инициализации.
- Обеспечена поддержка IPv6 (в т.ч. в Etcnet, модулях системы конфигурирования Alterator).
- Домен можно использовать для аутентификации компьютеров под управлением Windows.
- Обеспечена возможность установки на зашифрованные разделы (требуется незашифрованый /boot) с помощью LUKS.
- Улучшена поддержка исполнения 32-разрядных приложений в 64-разрядной среде.
- Обеспечена работа нативной версии 1С:Предприятие «из коробки».
- Улучшена отрисовка шрифтов (новый fontconfig и патчи от infinality.net).
- Централизованное резервное копирование компьютеров под управлением как Linux, так и Windows с помощью Bacula
- В серверный дистрибутив добавлен «Школьный портал».

Выпуск намечен на конец 2013 года.

В следующих версиях школьного комплекта планируется:

- Централизованное управление рабочими станциями со школьными дистрибутивами (управление программным обеспечением, параметрами настройки и их принудительным обновлением).
- Управление службами, разнесёнными на несколько серверов.
- Доработка и локализация образовательного программного обеспечения.
- Поиск и отбор лучшего свободного программного обеспечения для образовательных целей.
- Расширение документации и руководств для пользователей и администраторов.
- Сбор и включение в комплекты методических материалов в различных видах (методички, видеоуроки, презентации).

Филипп Занько
Казань, Исследовательский центр проблем энергетики КазНЦ РАН
Проект: Школа Python/Tk http://russianlutheran.org/python/python.html

О свободных форматах публикации результатов научных исследований

Аннотация

Новые компьютерные технологии на наших глазах сделали архаичными форматы распространения результатов научных исследований, которыми наука пользовалась последние столетия. Ключевые события: «умирание бумаги» как основного носителя информации и изменение роли научных издательств и авторского права с позитивной на негативную. В докладе делается попытка переосмысления взаимоотношений учёного с научным сообществом в новых условиях, даются практические рекомендации использования свободных форматов (в смысле свободного программного обеспечения и в смысле свободных лицензий на научные публикации) в научной работе и образовании. На примерах из личного опыта демонстрируется недооцениваемый пока потенциал и эффективность свободных форматов.

Симптомы кризиса

В начале 21 столетия мир столкнулся с уникальной во многовековой истории современных науки и образования ситуацией кризиса привычных форм представления информации. Одни симптомы этого кризиса развивались десятилетиями, другие появились буквально на наших глазах. Среди них:

- рост числа журнальных статей, журналов, монографий и т.п.;
- падение среднего уровня публикаций (субъективная оценка);
- «умирание» бумаги как основного носителя информации;
- электронное представление информации и повсеместное распространение Интернета превратили традиционные издательства и систему авторского права в факторы, сдерживающие развитие науки и образования.

Третий лишний

Монополизация коммерческими издательствами управления всеми научными публикациями привела к парадоксальной ситуации: даже опубликовав статью в престижном журнале, учёный не открывает свои результаты urbi et orbi, а, скорее, «закрывает» их, по крайней мере от коллег из слаборазвитых стран.

Выход из создавшегося тупика очевиден: научные издательства, ставшие огромным самостоятельным бизнесом, должны быть по возможности лишены своих посреднических функций между учёным и научным сообществом. В принципе, для подтверждения квалификации исследователя не нужно ничего кроме хороших научных результатов. Яркий пример тому — Григорий Перельман. Опубликованные на личном сайте под лицензией Creative Commons, научные материалы, если они действительно ценны и интересны, найдут и своего читателя, и своего квалифицированного критика. Конечно, за это не платят, но разве лучше тратить свою жизнь на статьи, которые не читает никто, кроме рецензента и редактора?

Выбор формата научной документации

В век компьютеров перед каждым исследователем встает проблема выбора формата для его документов [1]. Разнообразие существую-

щих форматов свидетельствует скорее о нерешённых проблемах, чем о богатстве выбора.

Формат	**Преимущества**	**Недостатки**
Визуальные форматы WYSIWYG (OpenOffice и др.)	Простота освоения, широкое распространение, лёгкость получения печатного документа среднего качества	Нечитаемый код разметки, сложность работы с большими файлами, различные сбои; генерируемый HTML-код не читаем
TEX/PDF	Простой, читаемый, мощный язык математической разметки, отличное качество печатного документа; легко конвертируется в читаемый HTML-код; удобен для работы с большими файлами	Трудоёмкий в освоении и использовании из-за концентрации на печатном представлении документа
DocBook	Универсальный формат, легко конвертируется во все остальные основные форматы документации	Сложный и громоздкий, код плохо читаем
Облегчённые языки разметки (Markdown и т.п.)	Очень легко читается исходный текст, легко конвертируется во все остальные основные форматы документации	Облегчённый язык разметки — это посредник, ещё один язык, который нужно осваивать пользователю

Мне кажутся наиболее важными следующие требования к форматам научной документации:

- формат должен быть открытым и свободным;

- основная форма существования научной документации — электронная, а не бумажная;
- код разметки должен быть легко читаем, допустимо смешение описания логической структуры документа с описанием его внешнего вида;
- следует избегать языков- и форматов-посредников.
- можно пожертвовать красотой оформления документа (но не читаемостью), чтобы сэкономить самый ценный ресурс — время.

Как ни странно, но всем этим требованиям вполне удовлетворяет написанный вручную обычный HTML-код (в основном, версии 3.2, листы стилей ухудшают читаемость кода разметки). Помимо других преимуществ такие документы обладают исключительной переносимостью, вплоть до электронных книг и сотовых телефонов.

Вариации на тему «литературного» программирования

Инструменты, о которых здесь пойдет речь, обязаны свои появлением желанию публиковать в Интернете не только научные статьи, но и тексты программ обработки данных. При этом хотелось, чтобы код легко понимался и не только его автором.

Предлагаются простые инструменты для работы с файлами, в которых код Python совмещён с разметкой HTML [2]:

- Процедура *html2py*() — это простой конвертер, способный преобразовывать HTML-страницы, содержащие код Python, в сценарии Python с расширением *.py* и запускать их на выполнение. Всё, что не находится между тегами `<CODE>` и `</CODE>`, превращается в комментарии Python. А между этими тегами как раз и находятся «кусочки» кода Python. В целом они представляют собой законченную программу на Python, её можно редактировать с использованием подсветки синтаксиса и сохранять в обычном редакторе, например в `IDLE`.
- Возможно и обратное преобразование: процедура *py2html*() убирает все добавленные комментарии, восстанавливая оформление исходного HTML-файла.

Подробное описание этих инструментов, их достоинств и недостатков, правила оформления HTML-файлов со встроенным кодом Python приведены в статье [2].

Совмещение программного кода и HTML-разметки «в одном флаконе» позволяет написать научную статью с традиционными описаниями, формулами, таблицами, рисунками, содержащую в себе готовую программу обработки данных, и опубликовать её в Интернете в виде одного текстового файла. Особенно привлекательна такая возможность, когда речь идет об учебных научных материалах.

Исследование выполнено при финансовой поддержке РФФИ в рамках научных проектов № 13-08-97063-р_поволжье_а, 13-08-97050-р_поволжье_а, 13-08-00359-а, 13-08-00504-а.

Литература

[1] *Реймонд, Э.С.*, Искусство программирования для Unix, М.: Издательский дом «Вильямс», 2005.– 544 с.

[2] *Занько, Ф.С.*, Комментарии Python в стиле HTML, `http://www.russianlutheran.org/python/literate/literate.html`

Миронов Андрей Михайлович, Михеев Андрей Геннадьевич, Пятецкий Валерий Ефимович
Москва, МГУ, Консалтинговая группа РУНА, НИТУ «МИСиС»

Проект: RunaWFE http://wf.runa.ru/rus

Реализация алгоритма проверки ограниченности количества точек управления в свободной системе управления бизнес-процессами и административными регламентами RunaWFE

Аннотация

В современных системах управления бизнес-процессами всвязи с ошибками проектирования бизнес-процессов могут возникать ситуации неограниченного возрастания количества точек управления в экземпляре бизнес-процесса, что приводит к неоправданно большой нагрузке . В докладе рассмотрена задача анализа схем бизнес-процессов, представлен алгоритм, проверяющий ограниченность количества точек управления, реализованный в системе RunaWFE, доказана теорема о корректности алгоритма

Проект RunaWFE

Проект RunaWFE развивает свободную, ориентированную на конечного пользователя систему управления бизнес-процессами и административными регламентами. Основная задача системы: раздавать задания исполнителям и контролировать их исполнение. Последовательность заданий определяется графом бизнес-процесса, который менеджер или бизнес-аналитик может быстро изменять при помощи редактора бизнес-процессов.

Проект RunaWFE предлагает использовать свободное ПО как средство кооперации генераторов идей и разработчиков промышленного ПО. В этом случае генераторы идей, передав свои идеи в проект, получают свободный инструмент, реализующий их идеи. Разработчики промышленного ПО, в свою очередь, получают идеи и теории, которые позволят разрабатываемому ПО получить качественные преимущества.

В настоящем докладе рассмотрен пример такого сотрудничества: решение сложной математической задачи и применение результатов в системе RunaWFE.

Проблема «взрывного» роста количества точек управления в экземпляре бизнес-процесса

В 4.0 версии RunaWFE в соответствии со стандартом BPMN 2.0 элемент «слияние» работает следующим образом: для каждого входящего перехода пришедшая в «слияние» точка управления ставится в очередь. Если для всех входящих в «слияние» переходов их очереди заполнены хотя бы одной точкой управления, то все точки управления, находящиеся на первой позиции очереди каждого перехода, уничтожаются и на выходе из «слияния» генерируется одна точка управления. Все остальные точки, находящиеся в очередях в «слияние», перемещаются на одну позицию вперед.

При отсутствии ограничений на комбинации элементов на схеме бизнес-процесса в случае такого определения поведения элемента «слияние» возможно появление экземпляров бизнес-процессов, в которых из-за ошибок проектирования количество точек управления бесконечно возрастает с течением времени. Такие процессы могут создать запредельную нагрузку на систему, что приведет к прекращению ее нормальной работы.

На рисунке 1 показан пример бизнес-процесса с бесконечно возрастающим количеством точек управления.

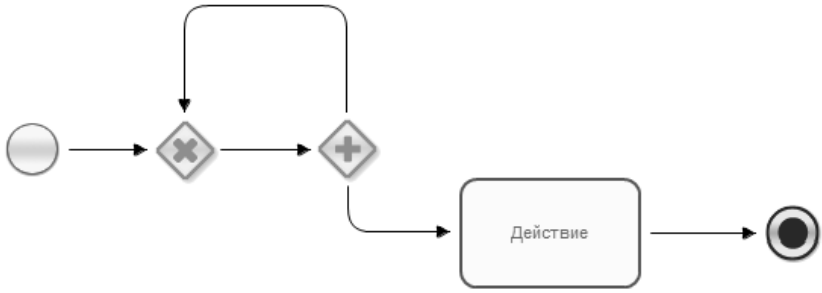

Рис. 1: Бизнес-процесс с бесконечно возрастающим количеством точек управления

Описание алгоритма

1. Нумерация узла «Начало» и всех переходов схемы

Посчитаем количество линий (стрелок) на схеме бизнес-процесса. Обозначим это количество — n. Поставим в соответствие узлу «Начало» номер 1. Всем переходам (стрелкам) схемы поставим в соответствие числа от 2 до n+1 в произвольном порядке.

2. Создание двух списков векторов и проверочного графа векторов

Создадим два списка векторов размерности n+1. Первая позиция каждого вектора соответствует узлу-началу, остальные позиции — переходу в соответствии с тем, как мы занумеровали переходы.

В первом списке (обозначим его V) будут содержаться вектора возможных изменений состояний. Все компоненты его могут принимать значения только из множества {-1, 0, 1}. Этот список будет один раз заполнен, и далее не будет изменяться.

Второй список называется «список векторов необработанных состояний». Все его компоненты могут содержать только целые неотрицательные значения. Значение в каждой компоненте обозначает коли-

чество точек управления на соответствующем переходе схемы. Этот список будет динамически изменяться во время работы алгоритма.

Проверочный граф представляет собой структуру из узлов, соответствующих векторам и направленных переходов (т.е. каждый переход направлен от одного узла-вектора к другому). Узлы являются векторами состояний (размерности n+1). Все компоненты каждого вектора могут содержать только целые неотрицательные значения. Значение в каждой компоненте обозначает количество точек управления на соответствующем переходе схемы бизнес-процесса. Проверочный граф тоже будет динамически изменяться во время работы алгоритма.

3. Заполнение списка V

А. Рассмотрим стартовый узел. Для каждого выходящего из стартового узла перехода добавим в список V вектор, в котором в позиции стартового узла (то есть, в первой позиции) находится «-1», в позиции выходящего перехода — «+1», а все остальные элементы — нули.

Б. Переберем в цикле все узлы «Параллельный шлюз» схемы бизнес-процесса. Для каждого элемента добавим в список V вектор, в котором для каждого входящего в узел перехода находится «-1», для каждого выходящего перехода — «+1», а все остальные элементы — нули.

В. Переберем в цикле все остальные узлы схемы бизнес-процесса. Для каждого узла, для каждой пары возможных комбинаций входящих и исходящих из этого узла переходов (входящий в узел переход, исходящий из узла переход) добавим в список V вектор, в котором в позиции входящего перехода находится «-1», в позиции выходящего перехода — «+1», а все остальные элементы — нули.

Далее вектора из списка **V** будем обозначать — **v(j)**.

4. Работа со списком векторов необработанных состояний и проверочным графом (основной этап работы алгоритма)

А. Поместим в проверочный граф вектор (1, 0, 0, ... , 0).

(В первой позиции, соответствующей узлу-Началу — одна точка управления, в остальных позициях — нули)

Б. Поместим в список «список векторов необработанных состояний» вектор (1, 0, 0, ... , 0).

В. Будем выполнять следующую итерационную процедуру до момента ее окончания:

Рассмотрим первый вектор из списка векторов необработанных состояний. Назовем этот вектор текущим вектором **u**.

Создадим для текущего вектора набор промежуточных векторов следующим образом:

Будем последовательно перебирать все вектора **v(j)** списка **V**:

К текущему вектору **u** покомпонентно прибавим вектор **v(j)**. Если ни в одной позиции полученного вектора не будет отрицательных чисел, то добавим этот вектор в набор промежуточных векторов.

Найдем в наборе промежуточных векторов все вектора, которые уже есть в проверочном графе. Для этих векторов создадим переходы из текущего вектора **u** в уже существующие вектора проверочного графа, совпадающие с промежуточными векторами, а эти промежуточные вектора удалим из списка промежуточных векторов.

Соединим текущий вектор **u** с оставшимися промежуточными векторами переходами и соответственно, удалим все оставшиеся промежуточные вектора из списка промежуточных векторов, поместим их в проверочный граф, а также добавим их в конец списка векторов необработанных состояний.

Удалим текущий вектор **u** из списка векторов необработанных состояний.

Рассмотрим список векторов необработанных состояний. Для каждого вектора из этого списка:

Построим множество векторов, из которых достижим данный узел (алгоритм построения этого множества описан ниже). Проверим, есть ли среди векторов, из которых достижим данный вектор, хотя бы один вектор, котором в каком-то компоненте число строго меньше числа из соответствующего компонента данного вектора, а в остальных — меньше, или равно.

Если хотя бы один такой вектор есть, то **алгоритм останавливается**, в редакторе процессов возникает сообщение — результат работы алгоритма: «**в бизнес-процессе может возникнуть ситуация с бесконечно возрастающим количеством точек управления**».

Далее итерационная процедура повторяется.

Если в какой-то момент времени список векторов необработанных состояний оказался пустым, то **алгоритм останавливается**, в редакторе процессов возникает сообщение — результат работы алгоритма: «**в бизнес-процессе не может возникнуть ситуация с бесконечно возрастающим количеством точек управления**».

5. Построение множества векторов, из которых достижим данный узел

Создадим два пустых список, которые назовем «список векторов» и «буфер»

Рассмотрим текущий вектор. Поместим в «буфер» все вектора, стрелки (переходы) из которых ведут в текущий вектор.

Далее проведем следующую итерационную процедуру:

Для всех векторов буфера ищем все возможные вектора, для которых существуют стрелки (переходы) из которых ведут хотя бы в один вектор из буфера. Удаляем из найденных векторов все вектора, содержащиеся в «списке векторов», или в буфере.

Переносим вектора из буфера в список векторов, а оставшиеся найденные вектора в буфер.

Если после этого буфер оказывается пустым, то прекращаем итерации

Повторяем итерацию.

Полученный «список векторов» и будет множеством векторов, из которых достижим данный узел.

В докладе будет представлено доказательство корректности работы данного алгоритма.

Результаты применения.

Для проверки бизнес-процессов на возможность бесконечного возрастания количества точек управления в экземпляре процесса, в редакторе процессов реализован алгоритм проверки.

В меню «Файл» добавляется команда «Проверить точки управления», которая активна, если в данный момент выбран бизнес-процесс в BPMN нотации, он сохранен и в нем нет ошибок (См. рисунок 2).

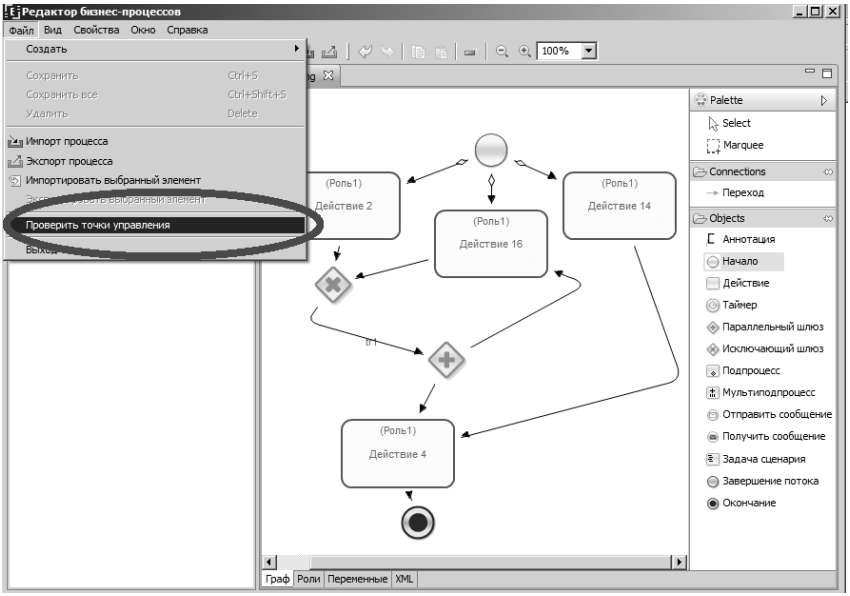

Рис. 2: Команда «Проверить точки управления»

Андрей Иванович Бодренко, Ирина Ивановна Бодренко
Волгоград, Волгоградский государственный университет

Система видеосвязи для невидимого интернета

Аннотация

Разработан интернет проект, представляющий собой систему видеосвязи и специальный браузер для сети I2P (невидимый интернет проект, открытое и свободное ПО). I2P является кросс-платформенным программным обеспечением, написанным на Java. В докладе представляются приложения для работы в сети I2P: программа организации видеоконференций, файлового обмена, упрощенной инсталляции сети I2P.

С помощью представленного программного обеспечения строится оверлейная (через I2P), самоорганизующаяся, распределенная, масштабируемая сеть. Каждому пользователю выделяется постоянный ад-

рес I2P в формате с расширением b32.i2p. Этот адрес можно рассматривать как статический IP адрес компьютера в обычном интернете. При этом пользователь имеет возможность сам изменить свой статический адрес в сети I2P. Показываются возможности применения этой безсерверной технологии, например, построение системы видео-наблюдения с большим количеством видео камер, системы мгновенных сообщений. Рассматривается пример портабельных приложений для запуска сети I2P, не требующих установки. Отдельно описывается взаимодействие представленных приложений с межсетевыми экранами, с применением NAT. Рассматриваются возможности работы приложений через fairwall.

В докладе отмечается надежность работы представляемого программного обеспечения. Представленная технология позволяет пользователям работать в сети I2P, используя все основные функции и возможности обычного интернета. При этом устойчивость соединений гарантируется безсерверностью сети I2P.

В докладе описываются некоторые недостатки сети I2P, к которым относится недостаточная, порой, скорость невидимого интернета. Рассматривается возможность развития представленной технологии для мобильных приложений и построения системы альтернативного интернета с использованием Wi-Fi.

Пожидаев Михаил Сергеевич
Томск, ALT Linux

Проект: Deepsolver `http://deepsolver.org`

Deepsolver: статус разработки и предложения

Аннотация

Deepsolver — это менеджер пакетов, разрабатываемый в ALT Linux. Доклад освещает состояние процесса разработки, встретившиеся трудности, а также содержит некоторые предложения для дальнейшего развития.

Настоящая статья содержит обзор процесса разработки менеджера пакетов Deepsolver, описание некоторых трудностей, которые были встречены в ходе работы, и ряд предложений для дальнейшего развития проекта. В отличие от намерений, высказанных разработчиками Deepsolver ранее, нынешняя реализация утилиты основана на алгоритме minisat. Все попытки применить 2-SAT алгоритмы приводили

к существенному ограничению гибкости и по этой причине были отклонены. Поддержка Deepsolver интегрирована в набор утилит girar и функционирует на сборочной площадке git.altlinux.org.

Одна из самых сложных проблем, с которой пришлось столкнуться в процессе разработки, заключается в том, что SAT-решение затрагивает необоснованно большое число пакетов. На практике это приводит к удалению из системы пакетов, присутствие которых не делает выполнение задачи пользователя невозможным. Рассмотрим для примера фрагмент SAT-уравнения: $(!r \vee a \vee i)$. Здесь r — некоторый зависимый пакет, a— некоторый пакет доступный для установки из репозитория, i — некоторый пакет, установленный в системе. Если алгоритм выбрал для установки a, i может быть как "истина", так и "ложь", и это никак не влияет на выполнимость запроса пользователя. В математической постановке обе ветви решений имеют одинаковый приоритет.

В настоящий момент Deepsolver содержит две реализованные оптимизации, но их недостаточно. Первая из них пытается по мере построения уравнения предугадать, какие переменные получат достоверно то или иное значение, после чего исключает из уравнения клозы, выполнимость которых оказывается очевидной. После исключения количество затронутых пакетов уменьшается, и это даёт частичный желаемый результат.

Другая идея учитывает пакеты, которые уже установлены в системе. Если установленный пакет появляется в некотором клозе без отрицания, но больше нигде не появляется в уравнении, то пакет должен остаться установленным, а весь клоз с ним считается истинным. Этот метод в терминологии Deepsolver назван "отложенными клозами", т. к. они добавляются в уравнение только после второго включения установленного пакета.

Предложенные методы частично решают описанные проблемы. К недостатку первого подхода, например, относится то, что он чувствителен к порядку вычислений. Они получили своё развитие в идее, которая описана ниже и находится сейчас в стадии реализации для постановки вычислительного эксперимента. Все клозы могут быть разбиты на группы, которые отражают действия, необходимые для возможности установки или удаления некоторого пакета. Между подобными группами может быть определено множество ссылок, которые отражают, какая группа приводит к необходимости обработки возможности установки или удаления пакета. Ссылки подсчитываются и после удаления некоторых из них запускается алгоритм "сборки му-

сора". Удаление ссылок возможно как в случае удаления установленных пакетов, которые нигде не включены с отрицанием, до запуска minisat, так и после него для тех переменных, значение которых не изменилось.

Ниже предлагаются несколько дополнительных идей для дальнейшего развития Deepsolver. Все они подразумевают введение третьего уровня управления пакетами без изменения текущей функциональности (первым уровнем считается librpm). Пользователи, привыкшие к поведению утилит в стиле APT, не заметят отличий в работе.

Первое предложение заключается в добавлении новых сущностей для обработки пакетным менеджером, таких как шрифты, jar-файлы и т. д. Внутренняя их поддержка должна быть реализована в гибкой форме, чтобы любое требуемое расширение их множества могло быть выполнено без модификации C/C++ кода. Любая информация, необходимая для сопоставления пакетов и сущностей в них, должна подготавливаться большей частью автоматически на этапе сборки пакета, не запрещая мэйнтейнеру вносить корректировки по его желанию.

Главная причина, по которой такие функции возложены именно на пакетный менеджер, в том, что только он в системе имеет полную информацию о приложениях и данных, которые не только уже были установлены пользователем, но и могут быть установлены из репозитория. В случае рассинхронизации расширенной информации вывод пакетного менеджера останется всегда корректным, но, возможно, будет неполным. Необходимо быть осторожным, выбирая, какие типы новых сущностей должны быть добавлены. Неясно, стоит ли таким образом обрабатывать, например, модули ядра. Вероятно, часть функций администрирования должны производиться по-старому.

Другое предложение подразумевает отслеживание пакетов, которые меняют поведение системы. Например, к ним относится пакет acpid-events-power. Важной особенностью этих пакетов является то, что при их установке или удалении не должны затрагиваться никакие другие пакеты. Оба предложения, очевидно, хорошо подходят для того, чтобы повысить удобство управления системой при помощи оконных утилит администрирования.

Пожидаев Михаил Сергеевич
Томск, ALT Linux

Проект: Deepsolver `http://deepsolver.org`

Deepsolver: development status and suggestions

Аннотация

Deepsolver is a package manager being under development in ALT Linux. The report covers various questions related to current work status as well as some suggestions to be considered by community members for further project evolution.

This report includes a survey of Deepsolver developing status, covers some noticeable difficulties faced during the work and offers several suggestions for further implementation. In contrast with the statements made by Deepsolver developers earlier at present Deepsolver got minisat-based solver. Any 2-SAT approaches considered for implementation previously cause a lack of flexibility and that makes them not suitable anymore. Deepsolver support is now integrated to girar utilities and successfully deployed on ALT Linux package building hosts.

Rather difficult problem faced amid development process is a unreasonably big number of the packages affected during user task processing. In most cases it can easily lead to removing from the system some set of packages which presence does not make the same general solution impossible. For example, consider the following situation written as a part of SAT-equation: $(!r \vee a \vee i)$. Here r is some dependent package, a is a package in a repository and i is a installed package. If algorithm once selected a i can be either "true" or "false" without an influence to a solution. Generally there is no priority between these branches.

To avoid any unnecessary consequences Deepsolver now tries to do a couple of optimizations described below but generally they are still insufficient and research should be continued. First attempt tries to predict which packages will be installed or removed anyway and exclude corresponding clauses from the equation as we already know that the result does not depend on them anymore. After an exclusion some previously affected packages are not involved in equation and this yields desired result.

The second optimization idea revolves around packages already installed in the system. Once they appear in equation there is a chance

they will not appear again and if corresponding clause does not imply intentional package removing we can consider such clause evaluated as “true” since installed package remains installed. This technique is called in Deepsolver as “postponed clauses”. After first appearance of installed package clause containing it is constructed but actually is added to equation only after second appearance.

These attempts partially solve the problem of unreasonably affected packages but that is not enough yet. The quality of described attempts is deeply depended on the order of processed packages. The further evolution of these ideas is described below and their implementation is in progress now. All clauses can be divided onto groups describing the actions needed for installation or removing of a particular package. We can define a set of references between these groups reflecting what group causes potential installation or removing of a package. The references should be counted and if some of them are removed we can launch a garbage collecting strategy to remove all groups left without any references to them. Removing of references is possible before and after minisat algorithm execution. The first case implies removing of clauses containing the installed packages which do not appear anywhere in a equation in negative state. After the minisat launch we can remove the references associated with the variables which state is not changed during SAT-solving. In both cases after the garbage collecting all variables used only in removed groups can be safely ignored even if SAT-solution yields change of their value.

We would like suggest several things to be considered by the community for further Deepsolver evolution. All of them imply creation of a new level of package management while current Deepsolver functions implement second level with using package formats libraries as first level (librpm, etc.). All proposed functions do not change anything in current architecture so all users used to work with any APT-like utilities will not notice any changes.

The first idea proposes adding new entities for control and processing by Deepsolver utilities. In particular such as fonts, jar-archives, etc. Internal implementation should be done in general form allowing easy and flexible enhancing with new types of entities vendor wants. Any additional information required for association between packages and entities should be collected on repository metadata construction either automatically or with package maintainers support.

The main reason why package manager is the most suitable place for a idea like that can be easily explained since the package manager is nearly the single place in a system with complete awareness what exact software and data are accessible through attached repositories. It can intersect enhanced information about proposed entities with list of really available packages and provide to user a available entities list of a given type precisely. For example, fonts. Even in case of any synchronization problems provided information will be always actual in sense of availability but probably incomplete. We would like to urge be cautious considering various types of files to be processed as a entity with package manager. For example, it is unclear whether kernel modules should be handled such manner or not. There are some cases that must be installed and removed with traditional administrative tools.

The second offer proposes designation of some packages like packages changing system behaviour. For example, packaged event file for acpid service changes buttons handling (in particular, power button). A user may want to view complete lists of such options with easy way to change their status (leading to package installation and removing). To be proper option as it is suggested above a package should be available for installation or removing without affecting other packages for consequent installation or removing. Only package manager is able to construct list of options satisfying all requirements so the best quality could be obtained only if this technique is a package manager essential part. Entities and options features of a package manager evidently are well-suited for integration to high-level GUI-instruments for system configuration making GNU/Linux desktop more user-friendly.

Пожидаев Михаил Сергеевич
Томск, ALT Linux
Проект: Luwrain `http://luwrain.org`

Luwrain: ОС для людей с проблемами зрения

Аннотация

Luwrain — это операционная система для людей с проблемами зрения. Она наследует наиболее успешные достижения проекта ALT Linux Homeros и развивает их. Доклад освещает архитектуру системы и основные принципы построения пользовательского окружения.

Luwrain — это операционная система (ОС), специально подготовленная для людей с проблемами зрения. Она основана на ядре Linux и предлагает пользовательское окружение, которое представляет все рабочие объекты в текстовом виде. Это существенно отличает Luwrain от ОС с графическим интерфейсом (GUI). GUI — это один из самых популярных вариантов пользовательской среды, но взаимодействие с ней осуществляется преимущественно при помощи мыши, практически полностью недоступной незрячему человеку. Это делает GUI крайне неудобной для работы при помощи речевого вывода.

Luwrain очень близка по своей архитектуре к Android (Java поверх ядра Linux), но она не основана на Android из-за большого количества уязвимостей, несмотря на некоторые преимущества. Вероятно, известные проблемы безопасности вызваны глубокой модификацией важных компонентов, таких как glibc, виртуальная машина Java и т. д. Luwrain использует все базовые пакеты такими, как их предлагают основные дистрибутивы GNU/Linux. Помимо прочего, этот подход позволяет рассматривать Luwrain как универсальную платформу без какой-либо ориентировки на конкретный тип компьютера: настольный компьютер, мобильный компьютер, планшет и т. д. Развитие информационных технологий идёт очень быстрыми темпами, и в таких условиях подобная универсальность позволяет обезопасить проект от потери актуальности из-за утраты интереса аудитории к некоторому типу устройства.

В архитектуре Luwrain есть три базовых компонента: ядро Linux, системные службы и пользовательский интерфейс на основе Java. Всё взаимодействие с необходимыми службами выполняется при помощи D-Bus. По этой причине предпочтение отдаётся тем из них, которые D-Bus поддерживают: udisks, network-manager и т. д. Для максимальной совместимости системы с другими дистрибутивами GNU/Linux все стандарты Free Desktop по мере возможности соблюдаются.

Реализация пользовательского окружения является одной из самых существенных задач в проекте. Она в значительной мере наследует традиции окружения emacspeak, широко известного в сообществе незрячих пользователей GNU/Linux. Главное преимущество, предлагаемое emacspeak, заключается в высокой скорости работы, которая обычно недостижима при использовании GUI в сочетании с программами экранного доступа. Luwrain старается сохранить весь положительный опыт emacspeak, устранив главные недостатки. Пользователь взаимодействует с несколькими прямоугольными текстовыми об-

ластями, отображаемыми на экране в виде тайлов. Текст выводится моноширинным шрифтом. Его размер и цвет, а также цвет фона устанавливаются общесистемно для всех областей по желанию пользователя. Каждая область связана с одним из запущенных приложений, и не все из них должны обязательно быть видимыми на экране. Пользователь может свободно и быстро переключаться между приложениями и областями, что качественно отличает Luwrain от emacspeak, который вовсе не имеет понятия приложения. Предусмотрен особый тип областей — так называемые всплывающие области, которые используются для организации диалога с пользователем. Их особенность заключается в том, что они могут быть показаны как один вызов метода в программном коде, который не возвращает управление до тех пор, пока область не будет закрыта. Две подобных всплывающих области имеют общесистемное значение — это главное меню и командная строка. Идея командной строки была унаследована от emacspeak, но в новом виде её поведение несколько отличается. С её помощью могут быть вызваны только команды, которые имеют общесистемное значение, и поведение которых не привязано к конкретной открытой области.

Стандартный набор приложений включает в себя файловый менеджер, утилиты для чтения почты, новостей, просмотра документов, текстовый редактор, календарь и т. д. Почти все из них подразумевают создание только оболочки в традициях Luwrain, поскольку полезный функционал большей частью уже доступен в виде библиотек. Эта возможность играла ключевую роль при выборе языка Java.

Есть некоторые вещи, которые не удовлетворяют концепции, изложенной нами. Функционально богатый веб-браузер не может быть представлен в терминах текстового интерфейса, поэтому он должен поставляться как внешнее приложение, работающее вне виртуальной машины Java. Существуют два возможных подхода к выбору веб-браузера:

- реализовать легковесный экранный чтец для AT-SPI и использовать в качестве веб-браузера Firefox, который имеет поддержку AT-SPI;
- использовать Chromium с включённым дополнением ChromeVox.

Первый вариант несколько сложнее, т. к. реализация чтеца для AT-SPI требует существенных усилий, но является более предпо-

чтительным, поскольку позволяет получить содержимое веб-страниц внутри виртуальной машины Java в доступном для операций копирования/вставки виде. Также это сделает доступными некоторые закрытые приложения, такие как, например, Skype.

Установку Luwrain незрячий пользователь может выполнять самостоятельно. Утилита для этого была подготовлена в рамках проекта ALT Linux Homeros и основана на технологии клонирования LiveCD.

Пожидаев Михаил Сергеевич
Томск, ALT Linux

Проект: Luwrain `http://luwrain.org`

Luwrain: text-based OS for blind persons

Аннотация

Luwrain is an operating system for blind and visually impaired persons. It adopts most successful ideas from ALT Linux Homeros project with an attempt to eliminate known disadvantages. The report provides general system design overview.

Luwrain is an operating system designed for blind and visually impaired persons. It uses Linux kernel and offers special type of user environment based on representation of objects in text form only in contrast to widely popular graphical user interface (GUI). GUI is a standard modern way to bring information to user but interaction is performed mostly through a mouse, usually not used by blind persons. That makes GUI rather inconvenient for this group of information technologies consumers.

Although Luwrain design in general is very close to Android ideas (Java on Linux kernel) it isn't based on Android. Even in case of some benefits, multiple Android vulnerabilities cause a lot of security concerns. Very likely known Android security issues are a consequence of deep modification of basic components: glibc, Java virtual machine, etc. Luwrain uses standard components as they are provided by GNU/Linux distributions including Linux kernel, Java SE, glibc and so on. This decision by the way allows consider Luwrain as universal platform without any bias towards any known computer type: desktop, laptop, tablet. As

device evolution goes quickly and mostly unpredictable this fact plays significant role.

There are three valuable components in Luwrain design: Linux kernel, system services and user environment based on Java as it was mentioned above. All necessary interaction with system services is performed through D-Bus activity. Luwrain intents to use udisks for removable media management, Network Manager for network interfaces operations and so on. Generally we would like to respect as many relevant freedesktop.org recommendations as it is possible making Luwrain compatible with usual Gnu/Linux desktop.

User interface approach is the most important part of the project. It is inspired by emacspeak package widely known in community of blind users of GNU/Linux. Its main advantage is a very high speed of work usually unapproachable with GUI-based systems in conjunction with screen reading software. We would like to save all positive parts of emacspeak experience eliminating its disadvantages. In Luwrain user interacts with several areas placed on a screen as tiles. Each of them always has rectangular form and filled with text written using monospaced font. Font size, font color and background color are defined globally and can be easily changed to be suitable for needs of a particular user. All areas are associated with launched applications and not all of them should be visible on the screen. User has convenient methods for quick applications and areas switching and that makes Luwrain different than emacspeak which has no applications at all. In addition Luwrain offers special type of areas called popup areas and designed for dialogs with a user. Their main difference is that they can be opened as single method call which does not return until popup area is closed. A couple of popup areas have system-wide meaning: main menu and command line prompt. The idea to offer easily accessible command line evidently was adopted from emacspeak but there it has slightly different behaviour. It provides the access only to commands with execution not related to a particular area. Commands invocation can be done in any time and yields the same result without dependence what applications are currently opened.

The standard set of applications includes file manager, mail and news reading applications, documents reading, text editor, address book, calendar and so on. Each of this tasks usually implies only creation of proper interface in terms of Luwrain as real back-ends for them mostly are available as various libraries for Java. This fact played nearly crucial role at language choosing decision.

But there are some exceptions from the strategy described above. Web browser cannot be implemented as text-based application so it should be available as a separate applications executed outside of Java environment. There are two possible alternatives for that:

- Implement AT-SPI-based light screen reader and let users launch Firefox as it has proper AT-SPI support.
- Include Chromium with ChromeVox add-on.

The former choice is more difficult since creation of libatspi client requires significant efforts but actually is more preferable because it allows collecting web-page contents in Java environment for copy-paste operations. In addition available libatspi client also gives a way for launching of some closed applications like Skype.

Luwrain installation program is based on the installation program previously developed as a part of ALT Linux Homeros project. It uses livecd-cloning technique and can be done by blind user without any additional sighted help.

Захаров Илья
Москва, Институт системного программирования РАН

Проект: Linux Driver Verification project
`http://linuxtesting.ru/project/ldv`, `http://www.google-melange.com/gsoc/project/google/gsoc2013/glaurung/5001`

Генерация модели окружения для группы модулей ядра для статической верификации

Аннотация

Система верификации LDV нацелена на проверку правил корректности использования интерфейсов ядра Linux драйверами при помощи инструментов статической верификации. Одновременно анализировать драйвер вместе с сердцевиной ядра затруднительно для существующих инструментов верификации из-за большого объема и сложности кода такой системы. Для верификации одного модуля ядра в системе LDV генерируется модель окружения, вместе с которой драйвер анализируется инструментом верификации. В случае взаимосвязанной группы модулей анализ каждого модуля отдельно от остальных приводит к использованию неполной и некорректной модели окружения и получению неверных результатов верификации. Доклад описывает

новый подход к верификации групп модулей ядра, который включает в себя выделение взаимосвязанных групп модулей, генерацию модели окружения для групп и их верификацию.

Система статической верификации драйверов LDV

Одним из направлений поиска ошибок в драйверах ядра Linux является применение инструментов статической верификации исходного кода. Инструменты, использующие в своей работе этот подход, позволяют проверять заданные правила корректности без выполнения программы. Анализ учитывает все возможные пути исполнения программы и не нуждается в специальном тестовом окружении.

Система верификации драйверов ядра Linux LDV [5] нацелена на проверку правил корректности использования интерфейсов ядра Linux драйверами при помощи инструментов статической верификации — *верификаторов*. В качестве верификаторов используются такие инструменты, как BLAST [3], CPAchecker [2] и UFO [1]. Примерами правил корректности, проверяемых системой LDV, являются правила вызова драйвером функций ядра `module_put/module_get`, `spin_lock/spin_unlock` и др. Всего система поддерживает проверку нескольких десятков различных правил корректности.

Верификация модулей ядра

Система верификации LDV позволяет верифицировать отдельные модули ядра, так как на сегодняшний день верификаторы не способны анализировать ядро Linux целиком из-за большого объема и высокой сложности исходного кода. Модули ядра сильно отличаются от обычных программ на языке Си. Каждый модуль тесно связан с остальной частью ядра, поэтому в системе LDV автоматически генерируется модель окружения, имитирующая взаимодействие сердцевины ядра и модуля [4]. Исходный код модуля анализируется совместно с такой моделью окружения верификатором.

Верификация группы взаимосвязанных модулей

В работе [6] *драйвером* называется код, предназначенный непосредственно для работы с конкретным устройством. А часть ядра, с которой взаимодействует данный драйвер, называется *вспомога-*

тельными библиотеками . Код драйвера может быть расположен в нескольких модулях и при этом опираться в своей работе на вспомогательные модули библиотек из одной или нескольких подсистем ядра. Поэтому при анализе каждого модуля ядра по отдельности сложно генерировать полную модель окружения — в ряде случаев лучше предоставить верификатору весь исходный код взаимосвязанных модулей. Для этого в данной работе предлагается анализировать модули драйвера вместе с используемыми ими модулями вспомогательных библиотек.

Для определения групп взаимосвязанных модулей предлагается объединять модули на основе экспортируемых и импортируемых символов. Для извлечения зависимостей применяется программа *depmod*, используемая также при определении порядка загрузки модулей в память. Получаемые от *depmod* данные позволяют построить ориентированный граф связанных модулей ядра. На основе полученного графа определяются взаимосвязанные модули драйверов и его вспомогательных библиотек.

Для полученной группы взаимосвязанных модулей генерируется общая модель окружения, учитывающая состав всех модулей группы. Как и в случае анализа одного модуля, полученная модель окружения вместе с выбранной группой модулей драйвера и его вспомогательных библиотек анализируется верификатором.

Литература

[1] *A. Albarghouthi, Y. Li, A. Gurnkel, M. Chechik*, Ufo: A Framework for Abstraction- and Interpolation-Based Software Verication, `http://www.cs.utoronto.ca/~aws/papers/cav12.pdf`

[2] *D. Beyer, M. E. Keremoglu*, CPAchecker: a tool for configurable software verification, `http://dl.acm.org/citation.cfm?id=2032305.2032321`

[3] *D. Beyer, T. Henzinger, R. Jhala, R. Majumdar*, The Software Model Checker Blast: Applications to Software Engineering, `http://www.sosy-lab.org/~dbeyer/Publications/2007-STTT.The_Software_Model_Checker_BLAST.pdf`

[4] *I. Zakharov, V. Mutilin, E. Novikov, A. Khoroshilov*, Generating Environment Model for Linux Device Drivers, `http://syrcose.ispras.ru/2013/files/SYRCoSE2013_Proceedings.pdf`

[5] *M. Mandrykin, V. Mutilin, E. Novikov, A. Khoroshilov, P. Shved*, Using linux device drivers for static verification tools benchmarking, `http://dl.acm.org/citation.cfm?id=2387324`

[6] *Yoann Padioleau, J. L. Lawall, G. Muller*, Understanding collateral evolution in Linux device drivers, `http://coccinelle.lip6.fr/papers/RR-5769.pdf`

Андрианов Павел
Москва, Институт системного программирования РАН
Проект: Linux Driver Verification `http://linuxtesting.org/project/ldv`

Оценка покрытия кода при статическом анализе

Аннотация

При динамическом анализе программ всегда возникает вопрос об оценке качества проведенного тестирования. Одним из широко применяемых подходов его оценки является измерение покрытия кода. Для статического анализа покрытие кода обычно считается стопроцентным. Однако, в действительности это оказывается не всегда так. В докладе рассматриваются причины возникновения такой проблемы и описывается подход к оценке покрытия кода, который был реализован в инструменте адаптивного статического анализа CPAchecker.

Тестирование является одной из важных частей разработки программного обеспечения. Но даже для достаточно простых программ тяжело перебрать и проверить все варианты её исполнения, поэтому возникает необходимость использовать некую количественную метрику, чтобы оценить, насколько хорошо проверена программа. Одной из таких метрик является *тестовое покрытие*, которое представляет из себя долю классов ситуаций, представители которых попали в тестовый набор.

В основном, тестовое покрытие применяется для динамического анализа, при котором проверяется работа программы на различных наборах исходных данных. В таком случае можно легко посчитать ту часть требований, которую мы проверили, или ту часть кода, которую мы реально выполнили. При тестировании программ удается воспроизвести не все варианты исполнения. Это становится важным минусом для программ, критических к ошибкам, поэтому наряду с

динамическим анализом применяется и статический. Для этого подхода оценка покрытия кода традиционно была высокой, так как исходный код анализируется весь целиком и без реального выполнения программы.

Основным минусом статического анализа всегда оставалось большое число ложных срабатываний, поэтому много усилий было сконцентрировано на повышении точности анализа. Учёт возможных значений переменных позволил не анализировать некоторые ветви условий. Использование нескольких циклов анализа, а также комбинация различных алгоритмов с последующим уточнением результатов также помогают исключить из анализа тот код, который при заданных ограничениях недостижим. При этом становится достаточно сложно отлаживать инструмент в случае, если он пропустил ошибку. Перед пошаговой отладкой можно посмотреть покрытие и выяснить, анализировал ли инструмент участок кода, содержащий ошибку. Еще одна причина, по которой какой-то код может стать недостижимым — это неполнота модели окружения, что является важным моментом для верификации драйверов. Это означает, что полностью смоделировать взаимодействие программы с операционной системой, пользователем и аппаратурой очень сложно, поэтому остается код, который не был проанализирован в рамках текущей модели. В этом случае нам очень полезно знать, какие именно функции были вызваны из модельного окружения. Тогда в случае необходимости можно уточнить модель или вручную добавить некоторые интересные функции. Таким образом, становится актуальным вопрос о том, какая же именно часть исходного кода была проанализирована.

В проекте LDV [1], целью которого является верификация драйверов Linux, используется инструмент статической верификации CPAchecker [2], который реализует идею адаптивного статического анализа (*Configurable Program Analysis*). Этот инструмент на основе графа потока управления строит граф достижимости и, обходя его в ходе анализа, собирает информацию о посещённых вершинах. Кроме того, у него есть информация о том, на в какую строчку исходного кода отображается каждая вершина. Таким образом, для решения задачи о поиске проанализированных строк нам необходимо в конце работы инструмента обойти все вершины и выяснить, каким строкам они соответствуют.

Формат вывода был выбран аналогичным утилите *gcov*, которая собирает информацию непосредственно во время запуска програм-

мы. Это было сделано для использования графической надстройки *Lcov*, которая по выводу *gcov* генерирует отчет в виде комплекта html-страниц со сведениями о покрытии в терминах исходного кода программы. После генерации файла в заданном формате запускается скрипт *Lcov*, который создаёт отчет. *Gcov* может считать, сколько раз выполнена та или иная строка, а также выводить информацию о покрытии ветвей. Ветви возможно выделять при достаточно сложных алгоритмах анализа, поэтому пока решено было остановиться только на отображении информация о покрытии строк и функций.

Инструмент оценки покрытия был использован на практике в проекте LDV. Это позволило обнаружить некоторые неточности ошибки в логике анализа и неточности модели окружения.

Рассмотрим пример отчета (рис. 1).

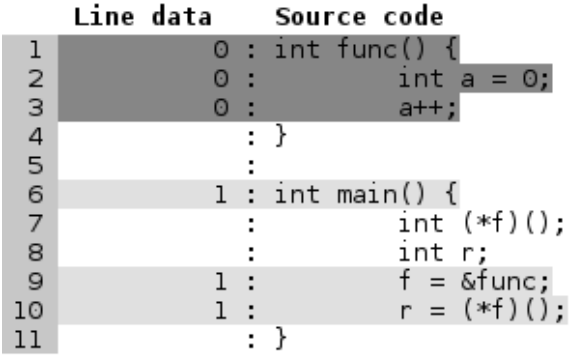

Рис. 1: Пример отчета

Темно-серым (в оригинале, красным) цветом подсвечиваются непосещённые строки, светло-серым (в оригинале синим) — посещенные. В данном примере в функции *main* вызывается функция *func* через функциональный указатель. Однако, без детального анализа верификатор не может понять, какая функция вызывается в строке 10, и функция *func* остается непокрытой.

В качестве дальнейших планов по развитию инструмента планируется попробовать реализовать поддержку покрытия ветвей.

Литература

[1] *M. Mandrykin, V. Mutilin, E. Novikov, A. Khoroshilov, P. Shved*, Using linux device drivers for static verification tools benchmarking, http://dl.acm.org/citation.cfm?id=2387324

[2] *D. Beyer, M. E. Keremoglu*, CPAchecker: a tool for configurable software verification, http://dl.acm.org/citation.cfm?id=2032305.2032321

Игорь Власенко
Киев, ALT Linux

Проект: Облачно-дружественная распределенная инфраструктура для сервисов автоматизации ALT Linux Team

Облачный кластер автоматизации сопровождения пакетов

Аннотация

создание программного обеспечения для облачного кластера автоматизированного импорта, сборки и тестирования программных пакетов из сторонних репозиториев СПО в репозиторий СПО Sisyphus и развертывание на базе полученного программного обеспечения облачного кластера автоматизированного импорта, сборки и тестирования программных пакетов для решения задач автоматизации полного цикла сопровождения в репозитории СПО для ряда классов пакетов программного обеспечения.

За просшедшее время наработки по автоматизации цикла сопровождения пакетов в репозиториях ALT Linux Team превратились в полноценный SDK для разработки приложений автоматизации на базе библиотек RPM::Source::Transform, RPM::Source::Editor, Source::Repository::Mass, RPM::Source::BundleImport.

На основе этого программного обеспечения разработано или находится в разработке более 20 приложений автоматизации, т.н. “роботов”, включая такие, как repocop, watch, cronbuild, croncopy, cronport, fcimport, mdkimport, mgaimport, pldimport, suseimport, jppimport, srpmbackport, sugarimport, octave-package-builder, mate-package-builder, perl-package-builder, nodejs-package-builder, CPAN-updater, girar-nmu-utils.

Однако при работе с утилитами автоматизации человеку приходится выполнять работы по целеуказанию (заданию пакетов, подлежащих сборке) и управлять процессом сборки и публикации пакетов. Для автоматизации и этих процессов было разработано программное обеспечение autorepo.

Соединение утилит автоматизации, утилит обслуживания списков пакетов и программнного обеспечения autorepo позволяет создавать сборочные узлы, которые под руководством или самостоятельно без вмешательства человека собирают и публикуют пакеты — автономные сборочные ноды.

Программное обеспечение «autorepo» упаковано в виде набора пакетов RPM autorepo-altnode-* и пакета autorepo-scripts. Пакеты autorepo-altnode-* являются служебными и предназначены для развертывания сборочных нод, нод тестирования и других служебных нод на кластере автоматизации. Сами же скрипты обслуживания субрепозитория находятся в пакете autorepo-scripts.

Скрипты пакета autorepo-scripts предназначены для обслуживания субрепозитория, т. е. некоторого репозитория, который является компонентой другого, большего репозитория, расширяет или перекрывает этот больший репозиторий. Например, репозиторий autoimports/Sisyphus расширяет репозиторий Sisyphus, а репозиторий autoports/p7 перекрывает (предоставляет обновления) репозиторий p7/branch. Компоненты позволяют логически разделить пакеты на разные классы (например, по происхождению или по уровням тестирования). Мотивацией создания autorepo-scripts послужили генераторы пакетов. Пакеты, созанные автоматически, нужно было вынести в отдельные репозитории, чтобы отделить их от пакетов, сопровождаемых вручную.

В autorepo-scripts есть 2 основных режима работы: автоматический режим работы и полуавтоматический режим работы. В автоматическом режиме работы нода может работать полностью самостоятельно: в цикле работы вызывается генератор списков имен на сборку, затем вызывается генератор пакетов, пакеты собираются и при успехе и отсутствии проблем (нет unmets, проходит install test) отправляются в репозиторий, также происходит уход за репозиторием, и при необходимости рассылка оповещений.

В полуавтоматическом режиме работы нода осуществляет уход за субрепозиторием, но не собирает пакеты. Уход за репозиторием включает в себя попытку пересобрать пакеты, в которых возникли unmets

(основной репозиторий, например, Sisyphus, не стоит на месте, поэтому в субрепозитории возникают unmets, даже если субрепозиторий не меняется), поиск и удаление устаревших пакетов (в зависимости от настроек для утилиты autorepo-purge), ведение архива репозитория, при необходимости рассылка оповещений. Чтобы собрать пакеты в субрепозиторий, работающий в полуавтоматическом режиме, нужна инициатива пользователя. Пользователь обновляет списки и запускает генератор пакетов, или иным образом выкладывает исходные src.rpm пакеты, которые нужно выложить в субрепозиторий, в подпапку ./OUT рабочего каталога, после чего запускает скрипт autorepo-mass-build. Кроме src.rpm пакетов, поддерживаются tar архивы, в формате, которым gear обменивается с hasher, а также транзакции из набора src.rpm или hasher tar архивов.

Таким образом, autorepo-scripts позволяют организовать свой собственный небольшой incoming и субрепозиторий.

Облачный кластер автоматизации предоставляет площадку для развертывания и администрирования роботов автоматизации. Его также можно рассматривать как еще одну альтернативу планируемым карманам: можно завести там отдельного псевдопользователя, развернуть под ним сборочную ноду, дать к ней доступ по ssh, и получить возможность сопровождать свой субрепозиторий с помощью autorepo-scripts.

Публичная страница статуса облачного кластера автоматизации доступна по адресу http://watch.altlinux.org/pub/monitor/index.htm.

На момент развертывания облачный кластер автоматизации обеспечивает работу более 20 роботов. Репозитории, поддерживаемые кластером, содержат более 3 тысяч дополнительных пакетов для платформы “p7” и более 10 тысяч дополнительных пакетов для платформы “Sisyphus”. Учитывая, что в репозитории “Sisyphus” более 4-х тысяч пакетов сопровождается с применением средств автоматизации, то уже сейчас число пакетов, сопровождающихся с применением средств автоматизации превысило число пакетов, сопровождающихся вручную.

Из других перспективных направлений автоматизации можно отметить автоматизацию портирования под другие архитектуры. Без автоматизации портирование под другие архитектуры достаточно дорогая операция, на примере недавних усилий по созданию armh репозитория.

При портировании операции с пакетами шаблонные, это отфильтровывание и замена зависимостей, а также отрывание/изменение опций configure и накладывание патчей в определенных последовательностях. При желании это все естественным образом автоматизируется.

В идеале можно было бы роботом генерировать десятками оптимизированные сборки, можно стабильных бранчей, под каждый дружественный к Linux планшет, е-книжку или другое устройство - там ресурсы скудные, и оптимизация будет цениться.

даже под ix86 было бы интересно сделать оптимизированные сборки Сизифа и p7 под atom, где тоже ресурсы относительно скудные, под i686, под athlon.

Облачный кластер автоматизации рассчитан на сопровождение сотен тысяч пакетов. Однако, чтобы раскрыть этот потенциал, необходима команда хотя бы из нескольких человек.

Шабалин Алексей
Москва, Лаборатория Касперского

Проект: Systemd http://www.freedesktop.org/wiki/Software/systemd

Systemd в ALTLinux

Аннотация

Systemd является альтернативной системой инициализации Linux, вобравшей в себя достоинства класического System V init и более современных launchd (Mac OS X), SMF (Solaris), Upstart (Ubuntu). В докладе пойдет речь об особенностях systemd в ALTLinux.

Историческая хроника

Первые упоминания в Интернет об прототипе systemd появились в конце апреля 2010г.:

`http://0pointer.de/blog/projects/systemd.html`,

`http://www.opennet.ru/opennews/art.shtml?num=26447`

7 июля 2010г. — представлена первая версия (systemd-v1).

Сентября 2010г., с systemd-v10, — начало добавления поддержки ALTLinux.

Февраль 2011г. — патчи с поддержкой ALTLinux приняты в апстрим (systemd-v18)

Январь 2012г. — в polkit-0.104 добавлена поддержка logind

Апрель 2012г. — импортирован проект udev в systemd.

Январь 2013г. — удаление из systemd поддержки любых дистрибутиво-специфичных конфигурационных файлов, systemd is now fully generic and distribution-agnostic. Systemd-v197

Сравнение систем нинциализации

Детальное сравнение систем инициализации опубликовано 28 апреля 2011 `http://0pointer.de/blog/projects/why.html` (перевод на русский `http://www.opennet.ru/opennews/art.shtml?num=30412`)

Рассматривайте это сравнение с большой долей скептицизма, т.к. большинство параметров для сравнения решались не внутри самого sysvinit, а внешними скриптами и настройками, что для конечного пользователя не существенно.

Стоит отметить следующие достоинства systemd:

- Поддержка SysV init-скриптов, как свох собственных сервисов
- Активация сервисов на основе сокетов: совместимость с inetd
- Интеграция с Linux Control Groups, автоматически создаёт cgroups для сервисов и пользовательских процессов для равномерного распределения времени CPU
- Поддержка контейнеров (как расширенная замена chroot())
- Загрузка, построенная на основе зависимостей
- Динамическая генерация сервисов
- Файлы запуска сервисов, совместимые с различными дистрибутивами
- Легкие для написания, расширения и обработки файлы управления сервисами
- Удобные средства диагностирования и анализа загрузки systemd-bootchart, systemd-analyze:

```
# systemd-analyze time
Startup finished in 5.810s(kernel)+16.161s(userspace)=21.971s
# systemd-analyze  blame
```

```
7.392s sysstat.service
7.332s ModemManager.service
6.102s altlinux-update_chrooted.service
4.707s lvm2-activation-net.service
3.885s systemd-udev-settle.service
2.183s NetworkManager.service
1.359s systemd-fsck-root.service
1.139s systemd-fsck@dev-disk-by\x2duuid-1dbbfa8d\x2d843c
               \x2d4ad4\x2daa61\x2dab499b942872.service
 904ms syslogd.service
 826ms systemd-tmpfiles-clean.service
 784ms systemd-random-seed.service
 534ms var-run.mount
 448ms colord.service
 ......
```

Особенности systemd в ALTLinux

- Systemd не единственная система инициализации в ALTLinux. В сервисах должна быть обеспечена поддержка SysV init скриптов.
- chkconfig, service, post_service, preun_service адаптированы для прозрачной работы с systemd и SysV. Мантейнеру не нужно специально использовать никаких новых макросов.
- /usr может быть на отдельном разделе диска.
- Изменены пути:

  `rootbindir=$(rootprefix)/bin` → `rootbindir=$(rootprefix)/sbin`
  `/usr/lib/systemd` → `/lib/systemd`
  `/usr/lib/tmpfiles.d` → `/lib/tmpfiles.d`
  `/usr/lib/binfmt.d` → `/lib/binfmt.d`
  `/usr/lib/modules-load.d` → `/lib/modules-load.d`
  `/usr/lib/sysctl.d` → `/lib/sysctl.d`

- поддержка «старых» ALTLinux-специфичных конфигурационных файлов:

  `/etc/sysconfig/i18n`
  `/etc/sysconfig/network (для определения hostname)`
  `/etc/sysconfig/keyboard и /etc/sysconfig/consolefont`

- Для поддержки старого именования сетевых интерфейсов возвращена функция переименования в udev. Старая схема неизменности имен сетевых интерфейсов также поддерживается.
- Для корректной локализации консоли сдвинут в конец инициализации systemd-vconsole-setup.service
- bash-completion адаптированы для bash3
- Часть программ вынесены в отдельный пакет systemd-utils (systemd-tmpfiles, systemd-binfmt, systemd-modules-load, systemd-sysctl) и предложены для использования в SysV
- Выключены сервисы, функции которых выполняются systemd (fbsetfont, keytable, killall, halt, single, netfs)

TODO

- Максимально убрать ALTLinux-специфичные изменения.
- Улучшение поддержки ALTLinux-специфичных настроек в /etc/sysconfig/ {consolefont,i18n,keyboard}. Перенос их обработки в генератор.
- Появление bash4 в ALTLinux сильно облегчит поддержку bash-completion.
- Отказ от prefdm. Замена на gdm.service, kdm.service, и т.п.
- systemd в initrd
- Генератор для cron ?
- Генератор для xined ?

Выдержка из беседы Линуса Торвальдса со студентами университета Аалто (23.12.2012):
Ок. Вопрос был о моем мнении на счет Systemd и о том, как некоторые люди думают, что она ломает философию unix и что она просто другая. Я не знаю скольких людей здесь волнует, что Systemd это такая замена традиционной модели Init. И она, в общем, пытается взять на себя множество других вещей в процессе запуска. Мне, на самом деле, нравится многое из того, что делает Systemd. Лично моя самая большая проблема с Systemd это то,

что многие вовлеченные люди похоже думают, что изменение — это хорошо само по себе. Я видел, как Леннарт Поттеринг(разработчик Systemd), например, говорил о том что что-то сделано плохо, потому что это что-то делалось 30 лет и все это — плохое по определению. Что для меня не имеет никакого смысла, потому что я думаю, если это работало 30 лет, оно определенно делает что-то правильно. Это моя точка зрения. В то время как некоторые люди из команды Systemd, похоже, имеют строго противоположные желания, говоря, что если оно работало таким образом 30 лет, то самое время это изменить. И такой склад ума заставляет меня очень нервничать, похоже, что иногда они делают изменения ради изменений и не сильно беспокоятся о том к чему люди привыкли и приспособились... Это, вероятно, причина почему Systemd генерирует столько негативных отзывов, потому что она выбивает людей из ощущения комфорта и чувствует себя неплохо по этому поводу. И в то же время я думаю, что многое, что она делает — интересно. Так что я немного нервничаю по поводу модели разработки и желания ломать вещи, что я считаю огромной ошибкой, но я также думаю, что она показывает множество перспектив.

Михаил Владимирович Быков
Москва, http://diglossa.ru

Проект: grunt-couch-mocha
https://github.com/mbykov/grunt-couch-mocha

Простой стек технологий для BDD-style разработки SPA на coffeescript

Аннотация

Как развернуть среду BDD-style разработки Single Page Application за 10 мин? Оказывается, можно. Вот этот нужный нам стек: node, grunt, coffeescript, mocha, phantom, couch

Пререквизиты

Предполагается, что вам известны и у вас установлены Node.js, npm, CouchDB.

```
- sudo apt-get install node npm
- sudo apt-get install couchdb
- rtfm
```

Клонируйте полный пример:

```
git clone https://github.com/mbykov/grunt-couch-mocha.
```

Все используемые npm-модули следует устанавливать в корне нашего проекта без ключа -g (если не сказано обратное явно), и удобно клонировать сам модуль по соседству, потому что обычно в нем есть примеры, которые существенно проясняют документацию.

SPA

Значение приложений, целиком работающих в браузере, растет очень быстро. Загрузки соответствующих модулей на npmjs.org измеряются тысячами в день, посмотрите, например, bower — 8 тыс установок в день. Меня особенно интересуют SPA — Single Page Application. Т.е приложения, которые при клике по линку не перегружают страницу целиком, но лишь подгружают необходимые данные. Однако настроить рабочую среду для разработки SPA и организовать код Javascript модульно и пригодно для тестирования считается сложной задачей.

AMD vs. CommonJS

Прежде всего, код должен быть разбит на маленькие модули, каждый из которых выполняет свою задачу (знакомая тема). Модулей в JS сейчас (до v.6) существую два типа — AMD (Asynchronous Module Definition) — и CommonJS. Второй не позволяет подгружать модули асинхронно. Но зато он является основой работы node.js. На основе AMD построены Yeoman и Bower, см http://yeoman.io — чрезвычайно популярная технология. Йомен решает именно ту задачу, которая была вынесена в заголовок настоящего доклада. Но это бегемот, устанавливая среду разработки, он устанавливает десятки и м.б. сотни npm-

модулей сплошным потоком. И в случае возникновения ошибки или сбоя найти концы будет невозможно, или по кр.м. этот процесс съест ровно то время, которое вы сэкономите, если воспользуетесь Йоменом. Я собираюсь за 10 минут сделать то же самое что делает Йомен, но пошагово, вручную и на основе стандартного CommonJS. Потому что AMD выглядит уродливо, а смысл в асинхронной загрузке модулей неясен — все равно, пока все необходимое не будет загружено, приложение не стартует.

Grunt

Grunt — это инструмент подобный Make, Rake, Cake etc. См `http://gruntjs.com`. Итак, создаем наш проект conf:

```
$ mkdir conf && cd conf
$ npm init
$ sudo npm install -g grunt-cli
$ grunt-init commonjs
```

Здесь мы создаем npm-описание проекта, т.е. файл package.json. Затем устанавливаем grunt-cli и выполняем grunt-init, который создает для нас центр всего проекта, файл Gruntfile.js, в котором пока что много лишнего, уберите все. Вот нужный минимум, можно просто его скопировать:

```
/*global module:false*/
module.exports = function(grunt) {
  // Project configuration.
  grunt.initConfig({
    // Task configuration.
  });
  // Default task.
  grunt.registerTask('default', ['xxxx']);
};
```

CouchDB

CouchDB — это документо-ориентированная БД, но с другой стороны, это скорее философия. Доступ к ней осуществляется по протоколу http://, данные внутри хранятся в json, основной используемый язык — javascript (но можно использовать что угодно также), и

можно использовать все тот же стандарт CommonJS. Таким образом, CouchDB сам себе веб-вервер. Приложения, построенные на основе кауча, называются couchapp.

Найдите на сайте npmjs.org модуль grunt-couch и установите его:

```
$ npm install grunt-couch --save-dev
```

Здесь `--save-dep` для того, чтобы зависимость сама прописалась в файле package.json, это удобно. Но мне привычннее создать директорию ddoc (от слов design-document), и скопировать файлы grunt-couch/test/fixtures/full в нее, получится что-то вроде:

```
_attachments/
|--templates
|--test
|   |--css
|   |--spec
|   |--js
|--css
|--js
|   |--vendor
|--index.html
filters/
_id
language
lists/
rewrites.json
shows/
updates/
validate_doc_update.js
views/
```

В файле Gruntfile.js добавляем две задачи, couch-compile и couch-push, и регистрируем модуль кауч:

```
  grunt.initConfig({
    // Task configuration.
      'couch-compile': {
          app: {
              files: {
                  'tmp/ddoc.json': 'ddoc'
              }
          }
```

```
    },
    'couch-push': {
        options: {
            user: 'admin',
            pass: 'kjre4317'
        },
        localhost: {
            files: {
                'http://localhost:5984/myapp': 'tmp/
                    ddoc.json'
            }
        }
    },
    .....
    grunt.loadNpmTasks('grunt-couch');
```

Мне обычно удобно прописать в файле /etc/hosts локальный хост

```
$ cat /etc/hosts
....
127.0.0.1       couch.loc
```

и прописать его в файле настроек Кауча:

```
$ sudo cat /usr/local/etc/couchdb/local.ini | grep
    couch.loc
```

```
[vhosts]
  ;example.com = /database/
  ...
  couch.loc:5984 = /myapp/_design/full/_rewrite/
```

Выполняем команды

```
$ grunt couch-compile
$ grunt couch-push
```

Можно создать задачу "couch объединяющие эти две:

grunt.registerTask('couch', ['couch-compile', 'couch-push']);

и теперь можно посмотреть на вновь созданную базу conference в Футоне по адресу `http://localhost:5984/_utils/`, и собственно работающее приложение — `http://couch.loc:5984/`

Mocha / PhantomJS

Самое главное: на nodejs.org находим, клонируем и устанавливаем модули

```
$ npm install grunt-mocha
$ npm install grunt-mocha-phantomjs --save-dev
```

Скопируйте файлы из директории `grunt-mocha/example/test/` в `ddoc/_attachments` по образцу.

Пропишите задачи mocha и mocha_phantomjs в файле Gruntfile.js — см. пример.

Вместо Mocha точно так же можно установить Jasmine, вместо PhantonJS — ZombieJS. Фантом использует реальный движок webkit, а Зомби — абстрактную реализацию HTML5 и CSS3, но он гораздо быстрее.

В файле, который мы вызываем в тесте, сейчас `ddoc/_attachments/test/test.html`, нужно указать

```
if (window.mochaPhantomJS) { mochaPhantomJS.run(); }
else { mocha.run(); }
```

это не сказано в документации к grunt-mocha, но если использовать grunt-mocha-phantomjs, работает именно именно этот вызов функции mocha.run(). Собственно, это и есть единственная хитрость всего процесса.

Теперь мы можем видеть, как выполняются наши тесты в браузере — `http://couch.loc:5984/test/test.html`, и в консоли:

```
Wombat
  ✓ should create a wombat with defaults
  ✓ should name itself if name passed in options
  #eat
    ✓ should throw if no food passed
    ✓ should return noms if food passed

Apple
  ✓ should go crunch


5 tests complete (243 ms)
```

Заключение

на этом можно было бы закончить, тесты выполняются. Но в реальной разработке также полезно установить модули

- grunt-coffee-jshint — проверяет корректность coffeescript
- grunt-contrib-jshint — проверяет корректность js
- grunt-mocha-cli — может быть полезно для прямого тестирования JS, минуя браузер
- grunt-browserify — конвертирует coffeescript в js и собирает единственный файл app.js
- grunt-contrib-concat — объединяет полученный app.js со статичными скриптами типа jQery
- grunt-contrib-watch — наблюдает за изменениями в указанных файлах, автоматически стартует выполнение тестов

Все они устанавливаются предельно просто, без каких-либо хитростей по соответствующим описаниям. Вместо Browserify.js можно использовать Component.js.

Voila.

Дмитрий Костюк, Алексей Шитиков
Брест, Брестский государственный технический университет

Оценка эффективности мультипрограммной работы оператора в современном графическом интерфейсе GNU/Linux

Аннотация

Выполнена сравнительная оценка эффективности взаимодействия оператора с несколькими приложениями в интерфейсе графических окружений KDE Plasma Desktop, Ubuntu Unity и Gnome Shell (включая модифицированный вариант последнего). Для оценки состояния оператора используются хронометраж, протоколирование, измерение сердечного ритма и энцефалограммы. Сравниваются темп работы, физическая нагрузка, концентрация внимания, а также время реакции. Выполнено ограниченное тестирование тех же оболочек на планшетных ПК. Обсуждается негативное влияние неотвлекающих полноэкранных интерфейсов на сосредоточенность оператора.

Ряд публикаций последнего времени ставит под вопрос пригодность для многозадачной работы графических окружений (DE), совершивших в 2011 г. отход от метафоры рабочего стола (DM) в сторону сенсорного управления, заимствованного у портативных устройств. Этот шаг влил конкретный смысл в термин «post-DM interface», долгое время бывший умозрительной абстракцией. Причиной перехода можно считать реакцию на планшеты, оказавшиеся весьма популярными, но оставлявшие чувство неудовлетворённости из-за слабого, в сравнении с ПК, функционала приложений и графических оболочек. Очевидно, что DE, удобное одновременно для планшета с сенсорным экраном и для управления мышью, получило бы огромное преимущество на рынке [1].

В этих «post-DE» наиболее заметны перемены, связанные с отказом от панели задач, с альтернативным управлением окнами и с запуском приложений. Изменения касаются также способов оповещения о событиях. В рамках концепции неотвлекающего интерфейса пользователя стимулируют работать с приложениями в полноэкранном режиме, скрывая неосновные элементы DE.

Реальная цена перемен не очевидна как на ПК, так и на планшетах. Поэтому, чтобы получить количественные критерии эффективности многозадачной работы, мы сформировали ряд тестов, нагружающих оператора взаимодействием с мультипрограмной средой и отслеживающих показатели его активности — скорость работы, величины физической и ментальной нагрузки, а также способность своевременно реагировать на события.

Разработанные тестовые программы взаимодействуют с графической оболочкой, требуя поочерёдной работы в нескольких окнах, с оценкой числа выполненных за фиксированное время типовых операций либо длительности их выполнения. Помимо темпа и точности действий, состояние оператора фиксируется монитором частоты сердечных сокращений (ЧСС) и электроэнцефалографом (ЭЭГ). В экспериментах нами использован спортивный пульсометр, фиксировавший среднюю и пиковую ЧСС за время теста, а также бытовой ЭЭГ NeuroSky Mindwave, встроенный контроллер которого вычисляет в условных единицах степень концентрации внимания оператора (т. н. технология eSense).

В качестве классических DE тестировались LXDE и KDE с панелью задач внизу экрана, а в качестве post-DE — Gnome Shell и Unity из дистрибутива Ubuntu 12.04. В ряде тестов рассматривалась также

модификация Gnome Shell с миниатюрами окон вдоль левой границы экрана. Мини-окна реализованы расширением WinThumbnails [2].

Тестирование проводилось преимущественно на ноутбуке с экраном 13 дюймов, а также на 10-дюймовом Android-планшете, получавшем удаленный доступ к DE по протоколу VNC. Близкое разрешение экрана в ноутбуке и планшете позволило убрать масштабирование в VNC-клиенте, что по возможности приближало работу к варианту запуска DE непосредственно на планшете.

Тестирование проводилось в три этапа. На первом этапе пользователь имел дело с двумя перекрывающими друг друга окнами, озаглавленными как «Source» и «Destination». В ходе итерации теста требуется без клавиатуры скопировать число из одного окна в другое с помощью контекстного меню, после чего нажать в исходном окне кнопку перехода к следующей итерации. На втором этапе для вставки чисел на каждой итерации указывалось одно из трех присутствующих окон «Destination», что позволило оценивать эффективность управления для многооконных приложений [1]. На третьем этапе окна «Source» и «Destination» поочередно показывают графические фигуры: сначала кнопку с изображением фигуры в окне «Source», затем 25 кнопок с фигурами в окне «Destination» (в ходе итерации требуется нажать кнопку с одинаковой фигурой в обоих окнах). Третий этап копирует методологию исследования запоминания и распознавания фигур, в свое время выполненного Р.М. Грановской и И.Я. Березной [3].

На первом этапе KDE и Gnome показали малое расхождение в темпе; зато включение мини-окон в Gnome позволило добиться максимального темпа. Монитор ЧСС предсказуемо зафиксировал минимальную физическую нагрузку на пользователя в KDE и максимальную в Gnome с мини-окнами. Концентрация внимания находилась в нейтральной зоне для всех оболочек и имела минимальный уровень в стандартной версии Gnome.

На втором этапе благодаря визуально-различимым миниатюрам Gnome получил преимущество, показав наибольший темп (а минимальный, аналогично [4], продемонстрировала оболочка Unity из-за неудобного переключения между окнами одного приложения). Физическая нагрузка также оказалась максимальной в Gnome. Наибольшую сосредоточенность пользователя демонстрировала оболочка Unity (однако к концу работы концентрация внимания несколько снижалась). Минимальной концентрация оказалась в KDE, а кроме того,

в KDE этот параметр характеризуется меньшим размахом колебаний (т. е. большей стабильностью).

Третий этап теста дополнительно к исследованию DE проводился в тайловом режиме, без переключения окон. В данном режиме тестируемые показали максимальный темп, и в среднем — более короткие паузы на поиск фигуры (время поиска). При сравнении графических оболочек максимальный темп продемонстрировали DE с панелью задач (преимущественно на данном этапе тестировалась оболочка LXDE), а минимальный темп был получен для Unity. При этом субъективно большинство тестируемых давали неверную оценку темпа, утверждая, что миниатюры окон ускоряют работу, позволяя начать выбор фигуры ещё в процессе переключения окон.

Среднее время поиска среди DE имело минимальные значения в LXDE и максимальные — в Unity. Однако интересен разброс пиковых значений времени поиска, т. е. ситуации, когда тестируемый впадает в ступор. Самый длительный ступор наблюдался преимущественно в Gnome, а второй по длительности — в тайловом режиме, что подтверждает роль переключения окон как полезной помехи, снижающей умственное напряжение [1]. Наименее же заметным ступор оказался при использовании панели задач. Средняя концентрация внимания предсказуемо была максимальна в тестах с тайлингом, а среди DE — с панелью задач. Далее по величине, как ни странно, идет концентрация внимания в Unity, а Gnome Shell замыкает ряд. Распределение пиковых значений концентрации внимания аналогично усредненным, за исключением того, что Unity и Gnome Shell меняются местами, т. е. уровень внимания в Gnome оказался менее ровным.

Учитывая наихудшую скорость прохождения теста и высокую концентрацию внимания в Unity, достаточно интересен вопрос, на что именно тратится умственная энергия: можно предположить, что на взаимодействие с механизмом переключения окон, а также на требующее большей сосредоточенности [4] горизонтальное перемещение мыши.

Относительно Gnome следует отметить, что за счет энергичного забрасывания указателя мыши в угол экрана, скорость прохождения теста в этой оболочке оказалась выше, чем в Unity. Более того, части тестируемых удалось «разогнаться» в процессе работы, приблизившись по скорости к LXDE (конечно, за счет большей ЧСС).

При тестировании пользователей на планшете применялся преимущественно второй этап теста. При этом не тестировались мини-

окна, и потому максимальный темп был продемонстрирован KDE, благодаря большим кнопкам панели задач и отсутствию переключения режимов; максимальному темпу предсказуемо сопутствовала и максимальная физическая нагрузка. В то же время максимальный уровень концентрации был достигнут в Gnome (даже в диапазоне повышенных значений), что с учетом худшего темпа прохождения теста показывает неэффективное расходование ментальной активности на планшете.

Меньшая скорость работы с планшетом по сравнению с ПК очевидна. Однако превышение темпа при управлении мышью было неодинаковым: минимальное отличие в 1.5 раза наблюдалось в KDE, а Gnome и Unity показали на планшете падение темпа приблизительно в 2.5 раза. Физическая нагрузка в KDE оказалась практически аналогична для сенсорного экрана и для управления мышью, а post-DE оправдали свое назначение, показав на планшете слегка меньше значения (в 1.1–1.2 раза) — не в последнюю очередь благодаря более крупным виджетам тем оформления. Умственная нагрузка на планшете и ПК, наоборот, была практически одинакова в оболочке Unity, а для настольных версий KDE и Gnome составила соответственно 0.8 и 0.7 от таковой в планшетном варианте.

В качестве вывода можно заметить, что в целом классические DE остаются высокоэффективными и предоставляют наиболее удобное переключение окон для сенсорного экрана, если пользователь может себе позволить место для массивной панели задач. Переключение в отдельный режим управления окнами вносит скорее негативный эффект: не обеспечивает высокой скорости работы и при этом приковывает внимание, что при продолжительном активном переключении окон приводит к росту усталости и может периодически вводить пользователя в состояние ступора. Это явно противоречит цели разработчиков создать неотвлекающую рабочую среду. Актуальным также является вопрос эффективности расходования ментальной активности: с учетом того, что в проведенных тестах показали себя наиболее проблемными Unity на ПК и Gnome на планшете, тезис о незначительных жертвах настольных DE в пользу удобного сенсорного управления ощутимо теряет в убедительности, а также делает целесообразным сравнительный анализ существующих модификаций данных графических оболочек.

Литература

[1] *Костюк Д.А. и др.* Исследование эффективности переключения окон в современных графических интерфейсах // Вестник БрГТУ., 2011. — № 5(71): Физика, математика, информатика. — С. 45–48.

[2] *Starun A.* WinThumbnails. `http://extensions.gnome.org/extension/335/winthumbnails`

[3] *Грановская Р.М., Березная И.Я.* Запоминание и узнавание фигур. / Л.: Изд-во Ленинградского ун-та., 1974. — 96 с.

[4] *Костюк Д., Дереченник С., Шитиков А.* Оценка эффективности управления окнами в современных графических оболочках // Седьмая конференция «Свободное программное обеспечение высшей школе»: Тезисы докладов / Переславль, 28–29 января 2012 года. М.: Альт Линукс, 2012. — С. 20–23.

Иван Хахаев, Дмитрий Державин
Санкт-Петербург, ОАО «НИИ ПС», СПбФ ОАО «Концерн «Вега»

Проблема доверенного компилятора в механизме сертификации ПО

Аннотация

В ходе аудита на предмет наличия недокументированных возможностей программных средств, написанных на компилируемых языках, возникает необходимость проверки соответствия исходным текстам программы двоичного кода, полученного в результате компиляции. Необходимым условием проведения такой проверки сейчас является наличие доверенного компилятора.

Обычный компилятор представляет собой чёрный ящик до тех пор, пока его код не прошёл аудит и пока он сам не был собран доверенным компилятором. В этот момент проявляется проблема курицы и яйца: чем собрать из проверенных исходных текстов доверенный компилятор? Решение оказывается настолько трудоёмким, что возникает вопрос — будет ли результат стоить затраченных ресурсов?

Проблема доверенного компилятора впервые была обозначена в 1974 году в закрытой публикации, а впервые публично сформулирована — десять лет спустя Кеном Томпсоном в его Тьюринговской лекции [1]. Тогда же была продемонстрирована техническая возможность

распространения вирусного кода через бинарные версии компиляторов без необходимости модификации их исходных кодов.

После долгих обсуждений проблему признали не имеющей решения и забыли о ней до 2005 года, когда Дэвид Уилер представил [2] методику двойной раздельной компиляции, позволившую достоверно распознать успешно проведённую атаку на компилятор. Сложность реализации методики заключается в необходимости наличия доверенного компилятора, способного собрать из исходных текстов компилятор, подлежащий проверке. Понятие «доверенный» в данном случае подразумевает гарантированное отсутствие недекларированных возможностей.

За 20 с лишним лет, прошедших с тех пор, как существование проблемы было доказано, до того момента, как было объявлено о её решении, «размножение» бинарных версий компиляторов происходило беспорядочно и бесконтрольно. В результате вплоть до настоящего времени найти доверенный компилятор, пригодный для сборки, например, gcc-4.7 практически невозможно. Таким образом, мы не можем с приемлемой степенью достоверности утверждать ни о наличии в программном обеспечении недекларированных возможностей, ни и об их отсутствии.

Этот важный вывод говорит о вероятном существовании фундаментальной уязвимости в современной системе сертификации программного обеспечения.

Фактически речь идёт о том, что в течение более чем 20 лет существует возможность тайно и беспрепятственно вносить вирусный код в программное обеспечение, применяемое в том числе в оборонной промышленности. И единственный известный на сегодняшний день способ противодействия атаке предполагает наличие доверенного компилятора, обладание которым в такой ситуации является ключевым моментом в области информационной безопасности.

Способ получения доверенного компилятора зависит от желаемой степени доверия. Учитывая масштабы проблемы, вариант с минимальной трудоёмкостью представляет собой получение эволюционно чистой линии, берущей начало от простейшего компилятора, написанного на языке, максимально приближенном к аппаратному уровню ЭВМ.

Учитывая возможность реализации атак, задействующих аппаратную часть компьютера, ещё большую степень доверия можно полу-

чить, используя к тому же аппаратную часть, разработанную по принципам Open Hardware.

Конечно, для достижения высоких степеней доверия потребуется координация усилий большого коллектива разработчиков. Результатом работы может стать стенд для сборки доверенного компилятора и проверки программного обеспечения методом двойной раздельной компиляции. Если такая работа будет проведена и опубликована, её результат сможет существенно повысить уровень доверия к свободному программному обеспечению, а так же стать ещё одним важным шагом к достижению независимости в области информационных технологий.

Литература

[1] *Thompson, Ken*, Reflections on Trusting Trust, `https://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf`

[2] *Wheeler, David A.*, Countering Trusting Trust through Diverse Double-Compiling (DDC), `http://www.dwheeler.com/trusting-trust/wheelerd-trust.pdf`

Михаил Шигорин
Киев, ALT Linux

mkimage-profiles как инструмент укрощения ARM

Сразу оговорюсь: речь пойдёт о платформах, предполагающих штатную возможность замены системного ПО, а не аппаратно-программных комплексах, поставляемых «как есть»; соответственно и об использовании в качестве основы бинарного пакетного репозитория, а не исходных кодов. Поэтому Buildroot этот проект не конкурент.

Краткая предыстория

Изначально mkimage-profiles, или m-p, задумывались как исследовательский проект с целью уменьшения степени внутреннего дублирования в mkimage-profiles-desktop и схожих профилях, применяемых

совместно с пакетной базой ALT Linux для создания решений на её базе.

По мере прояснения предположений и проработки реализации проект стал применяться для сборки публикуемых экспериментальных образов, а затем и официально выпускаемых ALT Linux «заготовок» для опытных пользователей (starterkits, сборки для ARM).

С тем, что такое форки и мержи, можно ознакомиться по моей отдельной тематической презентации «Скрестим вилки», представленной в Обнинске на конференции 2011 года; обзор mkimage-profiles доступен в ещё одном материале (2012).

Вводные

Сколь-нибудь единой ARM-платформы сегодня не существует: широчайшее применение процессоров данной архитектуры и отсутствие эталона вроде того самого IBM PC обусловило тесную адаптацию «железа» и ПО к конкретной задаче, решаемой конкретным поставщиком конечного устройства, а отсутствие возможности заменять ПО (полностью либо частями) привело к культуре прошивок, нередко одноразовых.

При этом даже появление претендующих на субплатформы вариантов оставляет весьма широкую вариативность по части периферии и организации загрузки.

Всё это приводит к тому, что создание образов для конкретных устройств приводит к взрывному росту количества случаев, которые имеют много общего, но и достаточно различий, чтобы без чёткого разделения общего и специфичного утонуть в дублях оказалось достаточно просто.

И вот здесь задумка m-p, которая как раз и включает такое разделение с возможностью наследования общего и описания специфики, оказалась достаточно эффективной.

Что получилось

На сегодня m-p позволяет собирать образы корневой файловой системы и чруты, предназначенные для развёртывания на соотвествующих ARM-устройствах.

В общем усилия сосредоточены на поддержке ARMv7 (репозиторий Sisyphus/armh), а в частности практический результат достиг-

нут на платформах Marvell ArmadaXP, Marvell Dove, NVIDIA Tegra3. При этом добавление поддержки платформ не создаёт особых проблем благодаря гибкости и модульности получившегося инструмента.

Публикуются сборки для Nexus 7 (архив ФС) и Cubox (образ ФС), причём в нескольких вариантах с различными DE для каждой платформы.

При этом возможность быстро взять существующую конфигурацию и легко дополнить/изменить её сообразно своим нуждам является одной из основных особенностей m-p, заложенной изначально. Так, сборка с MATE для Cubox потребовала написания всего трёх строчек конфигурации.

Ссылки

1. `http://altlinux.org/arm`
2. `http://altlinux.org/m-p`
3. `http://altlinux.org/cubox`
4. `http://altlinux.org/nexus7`

Глеб Фотенгауэр-Малиновский, Михаил Шигорин
Москва, ALT Linux

Крутим ARM в руках

ARM для конечного пользователя

В этом докладе рассматривается применение устройств на базе архитектуры ARM под управлением Linux, исключая встраиваемые контроллеры и портативную электронику под Android, с точки зрения обычного опытного пользователя.

Вопрос возник вместе с появлением более-менее доступных по цене и хакабельности вариантов, которые стало возможно применять как маломощные десктопные или серверные платформы.

Сервер

В качестве серверной платформы мы применяем в повседневной деятельности системы на базе Marvell ArmadaXP, которая содержит четыре раздельных узла на базе четырёхъядерных ARMv7-совместимых SoC на 1.6 ГГц с поддержкой памяти DDR3 ECC и SATA. Производительность одного ядра «на ощупь» соответствует примерно средним PIII, что вместе с большим объёмом памяти (2–8 Гб с учётом поддержки PAE) позволяет вполне комфортно реализовывать сборку пакетов и образов. Каждый отдельный узел способен загружаться с USB Flash и внутреннего SATA HDD/SSD; выбор осуществляется при помощи традиционного для ARM-систем загрузчика uboot, взаимодействие с которым возможно посредством интерфейса USB-serial. Работа с настроенной системой происходит через обычный Ethernet.

Замеченные недостатки: — существенно более низкая, чем теоретически возможно, пропускная способность при работе с RAM; — выключение или аппаратный сброс осуществляются сразу для всех узлов; — при активном использовании пассивного охлаждения не хватает, приходится применять продувку вентиляторами; — сложности с подбором совместимой памяти.

Достоинства: — достаточно мощная ARM-система; — относительно небольшой нагрев; — возможность работы с большим объёмом памяти, SATA-дисками, Ethernet.

В остальном система получилась достаточно пригодная к использованию и обслуживаемая.

Десктоп

В качестве законченного решения для миниатюрной настольной системы взят «кубик» SolidRun Cubox Pro на базе Marvell Dove; это одноядерная система (ARMv7-совместимый SoC на 800 МГц) с 2 Гб памяти и интерфейсами USB2, HDMI, S/PDIF, IR, eSATA. Она штатно загружается с microSD-карточки, что можно наблюдать с другого компьютера, подключив USB-порт кабелем к отладочному microUSB-порту Cubox, который заведён на USB-serial. Процессор «на ощупь» напоминает PII-300, т.е. под легковесными окружениями вроде XFCE и TDE машинка работает достаточно комфортно, а «современные» вроде KDE4 запускать запускает, но уже заметно медленней. Отчасти

это обусловлено небыстрым I/O, ведь microSD class 10 обеспечивает всего 10 Мб/сек; после загрузки и кэширования бинарников в памяти работать уже удобнее.

Ожидается вторая версия Cubox с улучшенными характеристиками по производительности SoC.

Замеченные недостатки: — весьма избирательная акселерация видео (youtube скорее не посмотреть); — CPU недостаточно мощный, чтобы тяжёлый Javascript вроде google search/mail обрабатывался без заметных задержек.

Достоинства: — весьма компактная законченная экономичная платформа; — очень удобная организация загрузки; — наличие возможности задействовать ускорение видео.

Портатив

Из мобильных компьютеров мы рассматривали Nexus 7 первого выпуска (на Tegra 3) — это четырёхъядерный SoC на 1,2 ГГц с 1 Гб памяти, загружающийся со встроенной флэш-памяти. Из возможных вариантов загрузки — в качестве единственной ОС или kexec-мультибут при помощи TWRP — был выбран последний как наиболее удобный для экспериментов с несколькими окружениями (Plasma Active и E17), размещёнными в соседних чрутах. Этот вариант также позволяет легче, быстрей и безопасней разворачивать новые чруты, не рискуя «брикнуть» устройство. К сожалению, задействовать видеоускоритель в полной мере не удалось (артефакты в плазме), хотя даже на программном рендеринге оба испробованных интерфейса достаточно быстры. В качестве одного из главнейших вопросов сразу же выступает экранная клавиатура, которая в наличии в обоих проектах, но доступна только для поддерживающих её приложений — нормально работать с xterm не получится. Также работает WiFi и звук, а вот с 3G-модемом проблемы: доступен код драйвера, но необходимые для работы юзерспейсные хелперы недоступны.

Можно было бы говорить о преимуществах, недостатках и дальнейшей разработке — но Google благополучно выпустил вторую версию на совершенно другом SoC, поддержки которого пока нет.

Выводы

На рынке потихоньку становятся доступными достаточно открытые реализации на базе ARM-процессоров, которые имеют практическую применимость. В общем случае она пока достаточно сильно ограничена либо отсутствующими драйверами под хотя бы часть аппаратного обеспечения конкретного устройства, либо недостаточной его мощностью. В частных случаях уже можно выбирать, обрабатывать напильником и применять.

Благодарности

Самая тяжёлая часть работы по поддержке ARMv7 в ALT Linux была проведена Сергеем Большаковым; также хочется сказать спасибо Ивану Овчеренко за примеры, советы и подсказки.

Игорь Воронин
МО, г. Шатура, Институт проблем лазерных информационных технологий РАН

Образовательный проект по робототехнике УМКИ на основе СПО

Аннотация

Подготовка инженерно-технических кадров на основе СПО. Управление множеством передвигающихся в пространстве роботов по заранее заданным алгоритмам. Локализация и идентификация положений роботов относительно реперных устройств и методом триангуляции. Использование сенсорных сетей для связи роботов в кластеры и группы с идентичными алгоритмами управления.

Введение

Типичная сенсорная сеть (см. рис. 1) состоит из множества дешевых, автономных, многофункциональных узлов (мотов), которые распределены в зоне мониторинга. Каждый узел состоит из набора блоков, таких как: сенсор, используемый для получения данных от окружающей среды, блок приема-передачи данных, микроконтроллер для обработки и управления сигналами и источник энергии. Узел

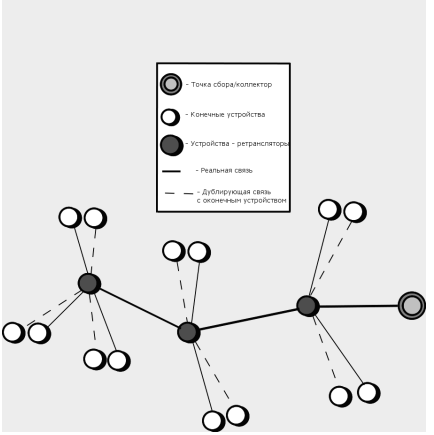

Рис. 1: Структура РСС

обычно питается от автономной батареи с ограниченным энергоресурсом, что приводит значительным ограничениям в энергопотреблении.

Работа каждого сенсорного узла может быть направлена на управление различными устройствами, в том числе и передвижным роботизированными платформами. Здесь время задержки может быть равно характерному времени изменения измеряемого параметра.

Контроль топологии сети таких устройств направлен на использование или уменьшение избыточных связей в сети в целях экономии ресурса. Управлять потреблением можно применяя различные энергосберегающие MAC протоколы и режимы работы устройств. Второй класс способов сохранения энергоресурса основан на количестве передаваемой информации, а так же на получение этой информации экономичными способами. Энергия потраченная на обработку информации не сравнимо меньше — требующейся для ее передачи, поэтому используется внутри-сетевая обработка данных, сжатие или предсказание данных. Так же используются мобильные стоки или ретрансляторы для экономии электроэнергии узлов сенсорных сетей.

В данной работе рассматривается разработка экономных алгоритмов маршрутизации систем мониторинга, проверке разработанных алгоритмов в применение на предприятии. В первой секции описыва-

ются известные алгоритмы маршрутизации. Вторая описывает разработанный алгоритмы маршрутизации, показан способ продления времени работы при помощи этого алгоритма, оценено реальное время работы сенсорной сети без обслуживания. Эта оценка экспериментально проверена на реальной сенсорной сети и позволяет судить о затратах связанных с обслуживанием распределенной сенсорной сети.

Существующие методы маршрутизации в РСС

Существующие методы маршрутизации можно разделить на несколько категорий [2] прямая, иерархическая и маршрутизация в зависимости от географического положения. Прямая маршрутизация подразумевает передачу сообщений от узла к узлу в сети где, каждый узел выполняет одинаковую функцию передачи и/или ретрансляции, в отличие от иерархической, где выделяется узел сбора и обработки информации. Минус прямой маршрутизации в том, что сети собирающие информацию с какой-то области будут посылать множество избыточной информации, особенно при значительной плотности сенсорной сети. Для того что бы избежать избыточности информации используют специальные алгоритмы, направленные на получение информации не от узлов, а от определенной области сети. Например в работе [3] описан алгоритм SPIN[1], где базовая станция посылает запрос к определенному региону сенсорной сети. Получив запрос, узлы области выполняют требование запроса, локально обмениваются данными и посылают обратно обобщенный ответ.

При иерархической же маршрутизации для сбора и обработки требуется использовать узлы с большим запасом энергии, что хотя и позволяет экономить на передачи уже обработанных данных значительно меньшего объема зачастую не приемлемо ввиду однородности используемых приборов или других трудностей. Что бы не использовать специализированные узлы существуют различные технологии. Например, в работе [4] описана технология LEACH[2], когда функцию сбора принимает поочередно несколько узлов сенсорной сети выбираемых по определенному алгоритму, тем самым распределяя нагрузку узла сбора.

[1] SPIN — Sensor Protocols for Information via Negotiation
[2] LEACH — Low-Energy Adaptive Clustering Hierarchy

Маршрутизация в зависимости от географического положения так же еще называется геометрическая маршрутизация, потому что для нахождения маршрута используется геометрическое направление на базовую станцию. При такой маршрутизации каждый узел передает сообщение своему соседу, у которого географическое положение ближе к стоку. Частным случаем маршрутизации в географических координатах является маршрутизация по виртуальным координатам [5], которые выстраиваются не только в зависимости от реального положения узла, а так же и учитывают естественные неровности поверхности, препятствия, уровень канала передачи и др.

Алгоритм маршрутизации с балансировкой трафика

Сеть представляется, как граф $G(N, M)$ с N узлами и M гранями, который представляет набор существующих узлов и возможные связи между ними соответственно. Каждый i-й узел изначально имеет запас энергии E_i. Каждая грань ij имеет цену e_{ij}, которая соответствует стоимости передачи одного пакета данных от узла i к j. Считается, что есть K направлений передачи, а информация генерируется со скоростью Q_c и передается по каналу связи c со скоростью q_c.

Время жизни каждого узла будет равняться в такой системе

$$T_i = \frac{E_i}{\sum\limits_{j \in M} q_c \sum\limits_{j,c} e_{ij}^c}$$

Тогда время жизни всей системы определим как:

$$T_{sys} = \min_{i \in N} T_i(q_c)$$

Задача максимизации времени жизни будет выглядеть:

$$Maximize \quad T_{sys}$$

При условиях:

$$q_{ij} > 0$$

$$\sum_{j \in M} q_c \sum_{j,c} e_{ij}^c \leq E_i$$

Последнее условие — это ограничение потребления энергии каждого узла.

Так же необходимо условие сохранения потока информации:

$$Q_c + \sum_{i \in S_j} q_{ij} = \sum_{k \in S_i} q_{ik},$$

где i — соответствует всем потокам информации от узлов к стокам k

Такая формулировка не применима напрямую, но служит для того что бы показать что задача максимизации может быть решена точно за полиномиальное время. В общем алгоритм маршрутизации должен соответствовать задаче, поставленной перед сенсорной сетью. Когда она направлена на сбор какой-либо информации, то основным показателем будет время работы и точность собранных данных. Если же целью сети является информирование или оперативная реакция на проявления окружающей среды, то в основным показателем будет, первую очередь, время задержки доставки сообщений по сети. Распределение узлов может быть как равномерным, так и хаотическим в зависимости от метода расстановки приборов. Если узлы были установлены вручную в определенные изначально места, то доставка сообщений может происходить по заранее заданным маршрутам. В случае произвольной установки необходимо либо использовать способы геометрической маршрутизации [5], либо устанавливать узлы с глобальной информацией о всех имеющихся маршрутах и связях, для расчета маршрутов и т.п.

Часто узлы могут быть блокированы из-за недостатка заряда батареи, выхода из строя по различным причинам, поэтому механизм маршрутизации должен быть защищен от такого рода изменений. Так же он должен учитывать особенности маломощных приемопередатчиков в БСС. Сенсорная сеть может содержать разное количество узлов, поэтому объем служебного трафика и данных не должен быть связан с числом узлов. Узлы сети работают от автономных источников питания, поэтому каждый алгоритм должен быть энергоэффективным. Для увеличения времени жизни предлагается к стандартному алгоритму маршрутизации по тому или иному способу добавлять так же алгоритм распределения трафика представленный ниже (см. рис 2).

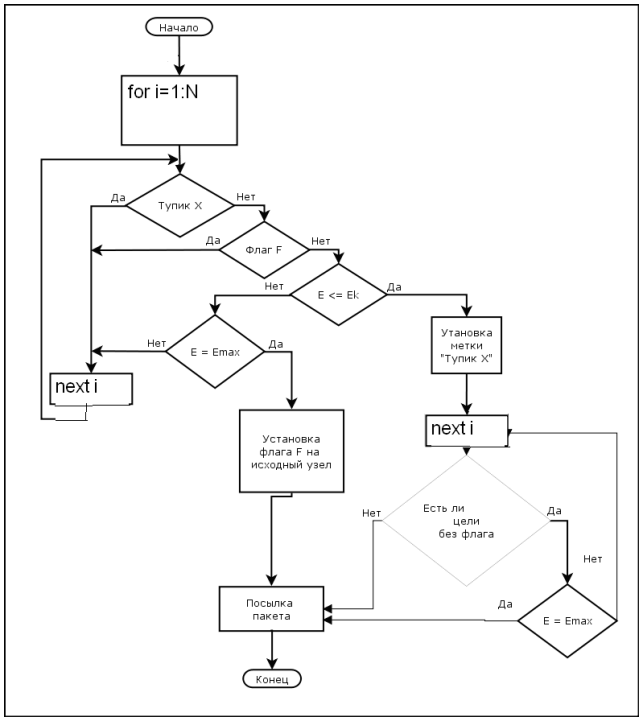

Рис. 2: Схема алгоритма распределения нагрузки. N — количество узлов соседних с текущим

Предложенный алгоритм маршрутизации основан на предположении, что каждый узел может запросить информацию о запасе энергии у своих соседних узлов. Требуется хранение информации о запасе энергии у каждого узла всей сети, которая, однако, распределена по всем нодам. Такой распределенный характер хранения информации позволяет масштабировать сеть до любых размеров. Так же в алгоритме есть механизм возврата и поиска нового пути доставки сообщения, если по первоначальному маршруту доставить пакет не удалось по той или иной причине. Основная цель алгоритма перераспределить нагрузку от узлов с меньшим запасом энергии к узлам с большим количеством энергии. Это позволяет увеличить время жизни всей сети

вследствие того, то на часто используемых маршрутах узлы будут выходить из строя позже.

Выводы

Задача экономии энергии в сенсорных сетях решается различными способами. В данной работе предложен простой алгоритм эффективной маршрутизации в сенсорной сети, основанный на учете остаточной энергии на узлах. Он легко реализуем в реальной сенсорной сети, в которой существует несколько возможных путей доставки пакетов данных. Алгоритм использует только локальные данные поэтому не требует дополнительных затрат энергии и памяти на хранение и обработку служебных данных, поэтому легко масштабируется для сети любого размера. Время жизни определяют как время до выхода их строя первого узла сети. Алгоритм направлен на перераспределение нагрузки от узлов передающих основной поток на всю сеть тем самым увеличивает время жизни всей сенсорной сети.

Литература

[1] *Giuseppe Anastasi, Marco Conti, Mario Di Francesco, Andrea Passarella*, «Energy conservation in wireless sensor networks: A survey», Ad Hoc Networks 7 (2009) pp. 537–568

[2] *Jamal N. Al-Karaki, Ahmed E. Kamal* «Routing techniques in wireless sensor networks: a survey» Wireless Communications, vol. 11 №6, (2004) pp. 6 — 28

[3] *J. Kulik, W. R. Heinzelman, and H. Balakrishnan*, «Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks», Wireless Networks, vol. 8, (2002), pp. 169–85

[4] *Heinzelman W., Chandrakasan A., Balakrishnan H.* «Energy-efficient communication protocol for wireless microsensor networks» // Proceedings of the 33rd Hawaii international conference on system sciences. Maui (USA), 2000.P. 8020–8029.

[5] *Баскаков С.С.* «Маршрутизация по виртуальным координатам в беспроводных сенсорных сетях», диссертация кандидата наук (2011) 218 стр.

Александр Гороховский *
Киев, Национальный технический университет Украины «КПИ»
Проект: Chemical cocktail `http://193.243.153.46/chemistry/`

Поиск уравнений реакций и стехиометрических коэффициентов для произвольно заданной смеси веществ, используя Chemistry::Harmonia

Аннотация

Рассмотрен концептуальный алгоритм и интерфейс новой подпрограммы Perl-модуля `Chemistry::Harmonia` — `stoichiometry`. С её помощью, для произвольно заданной смеси химических соединений можно найти все возможные „химически правильно“ сбалансированные уравнения реакций и фундаментальные наборы стехиометрических коэффициентов участников превращений.

Её применение разнообразно: CLI, GUI и Internet приложения, в хемоинформатике — создание химических баз данных, для образовательных и научно-аналитических задач.

Вдохновение для разработки автор черпал в сообществе GNU Linux, языка Perl и его модулей CPAN.

Среди большого разнообразия способов поиска стехиометрических коэффициентов (окислительно-восстановительных полуреакций, поатомный, по стадиям, структурно-схематический, основной и ускоренный электронный, электронно-ионный, механический) [1, 2, 3, 4, 5] для уравнивания химических реакций, как наиболее универсальный с точки зрения программной реализацией, наибольшее применение получил алгебраический способ.

Известными on-line ресурсами, предназначенными для уравнивания химических реакций являются:

- Химик: Уравнивание химических реакций (`http://www.xumuk.ru/uravnivanie/index.php`)
- Автоуравнитель (`http://www.ximicat.com/info.php`) или Balance Chemical Equation (`http://www.webqc.org/balance.php`)
- Calculator for Chemical Equation Balancer (`http://www.gregthatcher.com/Chemistry/BalanceChemicalEquations.aspx`)

* an.gorohovski@gmail.com

- Solve stoichiometric equation (`http://sciencesoft.at/equation/`)

Не менее многочисленно и семейство GUI программ обладающих теми же возможностями, но в разной степени. В первую очередь, для GNU Linux это — Kalzium (`http://edu.kde.org/kalzium/`).

В другом мире — для ОС Windows известными являются:

- Chemical Equation Expert (`http://www.educationsoft.net`),
- ChemEquations (`http://heilzhive.mylivepage.ru`),
- PL TABLE (`http://www.chemtable.com/ru/PLTable.htm`)
- Химический калькулятор (`http://heilzhive.mylivepage.ru/wiki/118/823`),
- Intelli Balancer (`http://www.intellibalancer.com`),
- Ural Chemical Calculator (`http://urchmclc.chat.ru/`),
- ChemBalance Wizard и др.

Следует отметить, что далеко не все из них распространяются для свободного и/или бесплатного использования. При этом, как это ни странно, программной реализации на языке Perl мне обнаружить не удалось.

Наиболее близкой к рассматриваемой здесь п/п `stoichiometry` Perl-модуля `Chemistry::Harmonia` является CLI программа, написанная на Java: Chemical Equation Balancer (`http://www.berkeleychurchill.com/software/chembal.php`), также имеющая ряд ограничений в своих возможностях.

Алгоритм уравнивания в `stoichiometry` — многоступенчатый и использует несколько различных способов, основным из которых является алгебраический. Концептуально он построен на следующих представлениях.

С точки зрения алгебры любое уравнение химической реакции можно рассматривать как *атомную* матрицу *однородной* системы линейных алгебраических уравнений (СЛАУ) [6, 7], неизвестными которой являются целочисленные стехиометрические коэффициенты исходных реагентов и продуктов, а постоянные при неизвестных — количества атомов химических элементов, из которых состоят участники превращений.

В общем случае, *атомная* матрица — прямоугольна, с размерностью $[a \times s]$, где a — количество строк (все атомы, из которых состоят

участники превращений), s — количество столбцов (все соединения — участники превращений). Например, для уравнения реакции:

$$x_1\,\mathrm{FeS_2} + x_2\,\mathrm{O_2} \longrightarrow x_3\,\mathrm{Fe_2O_3} + x_4\,\mathrm{SO_2} \tag{1}$$

при условии, что исходные реагенты слева расходуются и количество их атомов обозначено отрицательными значениями, атомной матрицей будет:

	$\mathrm{FeS_2}$	$\mathrm{O_2}$	$\mathrm{Fe_2O_3}$	$\mathrm{SO_2}$
Fe	-1	0	2	0
S	-2	0	0	1
O	0	-2	3	2

$$\rightarrow \quad A = \begin{pmatrix} -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 \\ 0 & -2 & 3 & 2 \end{pmatrix} \tag{2}$$

Здесь $a = 3$ соответствует числу алгебраических уравнений СЛАУ, $s = 4$ — числу искомых стехиометрических коэффициентов.

Порядок чередования строк и столбцов в матрице (2) не существенен. Кроме этого, разделение всех соединений на две категории: исходные реагенты и продукты, также не строго обязательно. Вполне достаточно точно указать один исходный реагент. В процессе решения противоположный знак найденных стехиометрических коэффициентов по отношению к обозначенному исходному реагенту подскажет какие из соединений должны относиться к противоположной категории, нежели указанной для них первоначально.

Такой подход позволяет оперировать всеми соединениями в целом как смесью и рассматривать весь спектр возможных уравнений реакций.

Не вызывает сомнения, что размерность атомной матрицы ограничена снизу минимальным числом атомов и соединений равным 2:

$$\{a, s\} \geqslant 2, \tag{3}$$

поскольку любое химическое превращение предполагает минимум два участника — исходный реагент и продукт и, как следствие, на каждый атом должно приходится два или более, включающих его в свой состав соединений.

В зависимости от числа атомов a и соединений s, можно выделить следующие три типа атомных матриц, т. е. уравнений реакций:

($a < s$) *недоопределённые* — число атомов меньше числа соединений. Это наиболее распространённый тип, представленный уравнением (1);

($a = s$) *определённые* — равенство числа атомов соединениям. По встречаемости занимает второе место, с большим отрывом от первого. Например:

$$(SCN)_2 + H_2O \longrightarrow HCNS + H_2SO_4 + HCN$$

здесь $a = s = 5$.

($a > s$) *переопределённые* — число атомов превосходит число соединений. Это наименее встречаемый тип уравнений. Например:

$$Na_3PO_4 + AgNO_3 \longrightarrow Ag_3PO_4 + NaNO_3$$

здесь $a = 5$, $s = 4$.

Первый и третий случай предполагает прямоугольные атомные матрицы. Второй, с квадратной матрицей, — охватывает две категории *тривиально* и *нетривиально* совместных уравнений.

Поскольку рассматриваемые СЛАУ являются однородными, то для *тривиально* совместных уравнений существует только *один набор* стехиометрических коэффициентов, в котором все они *равны нулю*.

Доказательством тривиальной совместимости является отличие от нуля определителя ($\Delta \neq 0$) квадратной атомной матрицы или равенство ранга её размерности ($r = s = a$). Такая ситуация является следствием неправильно составленного уравнения химической реакции — недопустимого или неполного сочетания участников превращений.

У *нетривиально* совместных — существует *фундаментальный набор* стехиометрических коэффициентов[1], среди которых все или только часть из них *не равны нулю*. Очевидно, что участники превращений с нулевыми коэффициентами являются лишними в уравнении реакции.

Поиск стехиометрических коэффициентов из прямоугольной атомной матрицы сопряжен с решением СЛАУ и представлением её в виде неоднородной системы, с помощью таких свойств, как ранг матрицы $r > 0$, ограничиваемый её размерностью $[a \times s]$ и количеством соединений s — определяющих число искомых стехиометрических коэффициентов.

Для перечисленных выше трёх типов атомной матрицы, между рангом r и числом соединений s возможны следующие соотношения:

[1] Фундаментальный набор содержит не кратные друг другу (линейно независимые) целочисленные коэффициенты.

($\mathbf{r = s}$) случай однородной определённой тривиально совместной СЛАУ. Все стехиометрические коэффициенты будут равны только нулю, т.е. уравнение реакции составлено *неправильно* и искомого решения нет.

($\mathbf{r = s - 1}$) нетривиально совместная матрица. Может быть получено только одно фундаментальное решение, т.е. один набор искомых ненулевых полностью стехиометрических коэффициентов. Это случай *правильно* составленного уравнения реакции.

($\mathbf{r = s - 2}$) также нетривиально совместная атомная матрица, однако, с двумя фундаментальными линейно независимыми решениями. Если это окислительно-восстановительная реакция (ОВР) с числом пар окислитель-восстановитель (e), то для отыскания из множества единственно возможного решения, потребуются дополнительные уравнения электронного баланса между окислителями и восстановителями. Иначе — заданное уравнение является некорректным и может быть множеством, как минимум 2-х алгебраически независимых реакций.

($\mathbf{r < s - 2}$) подобно предыдущему случаю, но заданное уравнение — множество как минимум $(s - r)$ алгебраически независимых реакций.

Таким образом, в основе критерия правильно составленного уравнения химической реакции[2] лежит определённый баланс между числом соединений s и рангом атомной матрицы r. С учётом (3) для правильно составленного уравнения реакции должен выполняться следующий набор условий:

$$\begin{cases} \{a, s\} \geqslant 2 \\ s - a - e < 2 \\ 0 < s - r < 2 \end{cases} \tag{4}$$

Из (4) следует, что ранг атомной матрицы r для определённых $(a = s)$ и переопределённых $(a > s)$ уравнений реакций, всегда должен быть меньше числа соединений s. Для недоопределённых уравнений $(a < s)$ — ранг может, но не всегда, соответствовать числу атомов, т. е. $r \leqslant a$.

Некорректно записанные уравнения ОВР с числом сочетаний пар окислитель-восстановитель $e \geqslant 2$ могут не подчиняться последнему

[2] С точки зрения возможности сбалансировать фундаментальным набором стехиометрических коэффициентов.

из условий (4). Здесь ограничение в правой части будет выше 2, на величину количества избыточных сочетаний пар.

Ранг атомной матрицы рассчитывается подпрограммой `Rank` из Perl-модуля `Math::Assistant`. Возвращаясь к примеру (1), ранг его атомной матрицы $r = 3$. Полученный результат удовлетворяет условиям (4), поэтому для уравнения (1) существует только один фундаментальный набор стехиометрических коэффициентов.

В общем случае, для *недоопределённых* уравнений реакций ($a < s$), которые правильно составлены (выполняется условие $s - r = 1$) одному из искомых коэффициентов можно первоначально присвоить любое минимальное целое число (например, 1), а соответствующее ему соединение (столбец) исключить из атомной матрицы, тем самым, сделав её квадратной.

В `stoichiometry` значения стехиометрических коэффициентов находят через определители методом Барейса [8], который реализован подпрограммой `Det` Perl-модуля `Math::Assistant`.

Несколько иначе выглядит ситуация с определёнными ($a = s$) и переопределёнными ($a > s$) уравнениями реакций, для которых также должны выполнятся условия (4). Здесь, чтобы воспользоваться описанной выше процедурой поиска стехиометрических коэффициентов, надо исключить из атомной матрицы $(a - r)$ строк — алгебраически линейно зависимых химических элементов.

Как было отмечено выше, уравнения реакций с условием $(s-r \geqslant 2)$ являются составными. Разделить такую смесь соединений на отдельные реакции с фундаментальными наборами стехиометрических коэффициентов можно перебором сочетаний соответствующих соединений. В этой категории особым случаем являются уравнения ОВР. Например, реакцию:

$$\mathrm{KMnO_4 + H_2O_2 + H_2SO_4 \longrightarrow K_2SO_4 + MnSO_4 + H_2O + O_2} \qquad (5)$$

с $s = 7$, $a = r = 5$ можно рассматривать в виде следующих уравнений:

$$\mathrm{2\,KMnO_4 + 2\,H_2O + 3\,H_2SO_4 \longrightarrow K_2SO_4 + 2\,MnSO_4 + 5\,H_2O_2} \qquad (6)$$

$$\mathrm{4\,KMnO_4 + 6\,H_2SO_4 \longrightarrow 2\,K_2SO_4 + 4\,MnSO_4 + 6\,H_2O + 5\,O_2} \qquad (7)$$

$$\mathrm{2\,KMnO_4 + 3\,H_2SO_4 \longrightarrow K_2SO_4 + 2\,MnSO_4 + 3\,H_2O_2 + O_2} \qquad (8)$$

$$\mathrm{2\,H_2O_2 \longrightarrow 2\,H_2O + O_2} \qquad (9)$$

среди которых только два будут алгебраически линейно независимыми.

Общеизвестно, что в реакции (5) обмен электронами проходит в основном между окислителем $KMnO_4$ и восстановителем H_2O_2 по схеме электронного баланса:

$$Mn^{+7} + 5\,\bar{e} \longrightarrow Mn^{+2}$$
$$O_2^{2-} - 2\bar{e} \longrightarrow O_2^0$$

С учётом этих представлений атомную матрицу можно дополнить 6-й строкой обмена электронами между $KMnO_4$ и H_2O_2:

	$KMnO_4$	H_2O_2	H_2SO_4	K_2SO_4	$MnSO_4$	H_2O	O_2
K	−1	0	0	2	0	0	0
Mn	−1	0	0	0	1	0	0
O	−4	−2	−4	4	4	1	2
H	0	−2	−2	0	0	2	0
S	0	0	−1	1	1	0	0
$\bar{e}$	5	−2	0	0	0	0	0

(10)

и после её решения получить ожидаемые стехиометрические коэффициенты:

$$2\,KMnO_4 + 5\,H_2O_2 + 3\,H_2SO_4 \longrightarrow K_2SO_4 + 2\,MnSO_4 + 8\,H_2O + 5\,O_2$$

В подпрограмме `stoichiometry` обмен электронами между окислителем и восстановителем и электронный баланс определяются с помощью уникального алгоритма во вспомогательных подпрограммах `oxidation_state` и `redox_test`. Эта принципиальная особенность алгоритма обеспечивает получение химически правильных решений в случаях ОВР, которые невозможно получить во всех других подобных программах.

Интерфейс `stoichiometry` в качестве основного аргумента ожидает строку, в которой разделителями реагентов и продуктов могут быть последовательности символов: *равно* (=) и/или *минус* (-), идущие совместно с одним или несколькими символами (или без него) *больше* (>). Например:

=, ==, =>, ==>, ==>>, -, --, ->, -->, ->>> и т. д.

Пробельные символы вокруг разделителя не имеют никакого значения. В случае, когда разделитель отсутствует (химическая смесь), формула последнего вещества рассматривается в качестве продукта[3].

Кроме разделителя реагентов и продуктов, в анализируемой строке `stoichiometry` ожидает найти химические формулы веществ, которые должны быть разделены одним из символов: *плюс* (`+`), *запятая* (`,`), *точка с запятой* (`;`) и/или *пробельными* символами. Некоторые вещества могут быть заданы со стехиометрическими коэффициентами.

Интерфейс stoichiometry

```
use Chemistry::Harmonia qw( stoichiometry );
use Data::Dumper;
my $ce = 'KMnO4 + H2O2 + H2SO4 --> K2SO4 + MnSO4 + H2O + O2';
print Dumper stoichiometry( $ce );
```

OUTPUT

```
'5 H2O2 + 3 H2SO4 + 2 KMnO4 == 8 H2O + 5 O2 + 1 K2SO4 + 2 MnSO4',
'2 H2O + 3 H2SO4 + 2 KMnO4 == 5 H2O2 + 1 K2SO4 + 2 MnSO4',
'6 H2SO4 + 4 KMnO4 == 6 H2O + 5 O2 + 2 K2SO4 + 4 MnSO4',
'2 H2O2 == 2 H2O + 1 O2',
'3 H2SO4 + 2 KMnO4 == 3 H2O2 + 1 O2 + 1 K2SO4 + 2 MnSO4'
```

Уравнение с огромными коэффициентами

```
print Dumper stoichiometry( '[Cr(CO(NH2)2)6]4[Cr(CN)6]3, KMnO4,
     H2SO4, K2Cr2O7, KNO3, CO2, K2SO4, MnSO4, H2O' );

1399 H2SO4 +10 [Cr(CO(NH2)2)6]4[Cr(CN)6]3 +1176 KMnO4 == 1879 H2O
 + 660 KNO3 + 35 K2Cr2O7 + 420 CO2 + 1176 MnSO4 + 223 K2SO4
```

Эксперимент с «безумной» смесью

```
print Dumper stoichiometry( 'H2 Ca(CN)2 NaAlF4 FeSO4 MgSiO3 KI
 H3PO4 PbCrO4 BrCl CF2Cl2 SO2 PbBr2 CrCl3 MgCO3 KAl(OH)4
 Fe(SCN)3 PI3 NaSiO3 CaF2 H2O' );

24 BrCl + 6 CF2Cl2 + 6 NaAlF4 + 119 H2 + 2 H3PO4 + 6 KI +
 6 MgSiO3 + 18 Ca(CN)2 + 12 PbCrO4 + 12 FeSO4 + 24 SO2 ==
 12 PbBr2 + 12 CrCl3 + 18 CaF2 + 110 H2O + 6 KAl(OH)4 +
 6 MgCO3 + 6 NaSiO3 + 2 PI3 + 12 Fe(SCN)3'
```

[3] Следует отметить, что присутствие такого разделителя не является обязательным — это может быть просто список веществ из которых будут составлены все возможные уравнения реакций.

Возможность `stoichiometry` уравнивать реакции была проверена автором более чем на 24700 уникальных уравнениях, собранных из отечественных и доступных зарубежных печатных изданий по химии.

Литература

[1] *Bottomley, J.* Note on a method for determining the coefficients in chemical equations / J. Bottomley // *J. Chem. News.* — 1878. — Vol. 37. — Pp. 110–111.

[2] Способы подбора коэффициентов в химических уравнениях / Л. Г. Берг, С. Д. Громаков, И. В. Зороацкая, И. Н. Аверко-Антонович. — Казань: Изд-во Казанского ун-та, 1959. — 148 с.

[3] *Harjadi, W.* A simpler method of chemical reaction balancing / W. Harjadi // *J. Chem. Educ.* — 1986. — Vol. 63. — Pp. 978–979.

[4] *Herndon, J. A.* On balancing chemical equations: Past and present / J. A. Herndon // *J. Chem. Educ.* — 1997. — Vol. 74. — Pp. 1359–1362.

[5] *DeKock, R. L.* Balancing redox equations / R. L. DeKock, B. M. Brandsen // *J. Chem. Educ.* — 2010. — Vol. 87, no. 5. — Pp. 476–477.

[6] *Степанов, Н. Ф.* Методы линейной алгебры в физической химии / Н. Ф. Степанов, М. Е. Ерлыкина, Г. Г. Филиппов. — М.: Изд-во МГУ, 1976. — 362 с.

[7] *Missen, R. W.* Chemical Reaction Stoichiometry (CRS): A Tutorial / R. W. Missen, W. R. Smith. — 1998. — 48 pp.

[8] *Bareiss, E. H.* Sylvestr's identity and multistep integer-preserving gaussian elimination / E. H. Bareiss // *Math. Comp.* — 1968. — Vol. 22. — Pp. 565–578.

Резцов Яков Евгеньевич
Ставрополь, администратор форума поддержки пользователей открытых офисных пакетов forumooo.ru

Проект: LanguageTool — свободное программное обеспечение для проверки грамматики и стиля. `http://languagetool.org/ru/`

Реализация модуля проверки русской грамматики на основе программы LanguageTool

Аннотация

В докладе рассматривается принцип реализации проверки грамматики, реализованный в программе LanguageTool. Рассмотрены особенности реализации проверки грамматики для русского языка. Рассматривается структура XML-правил, описывающих грамматику.

LanguageTool — это свободное программное обеспечение для проверки грамматики и стиля. С его помощью можно обнаружить ошибки, которые не могут быть обнаружены с помощью программного обеспечения для проверки орфографии по словарю. Программой проверяются взаимосвязи слов в предложении, а не только отдельные слова.

Проверку грамматики возможно реализовать либо на основе правил, либо на основе статистических методов (машино-обучаемая проверка). Анализ текста в Languagetool основан на правилах. Правила могут быть описаны на основе XML, либо в виде программных модулей Java.

LanguageTool был изначально создан Daniel Naber на Python в качестве его дипломной работы [1], а затем был портирован на Java. Сейчас ядро программы разрабатывается Daniel Naber и Marcin Miłkowski [2]. Программа для каждого языка имеет отдельный языковой модуль. У каждого модуля есть свой автор, полный список поддерживаемых языков доступен на страничке `http://languagetool.org/languages/`

Программа имеет интеграцию с Openoffice.org и LibreOffice, но может использоваться и отдельно. В независимой версии реализован графический интерфейс, консольное приложение и web-сервер.

Большинство грамматических правил описано на основе XML. Текст перед обработкой правилами предварительно разбивается на

* an.gorohovski@gmail.com

предложения. Это действие выполняет модуль на библиотеке *SRX segment library* [3]. В рамках Languagetool был создан специальный SRX файл (*segment.srx*) с описанием правил деления текста, в том числе и для русского языка. Этот файл может быть использован и в других системах обработки текста, например в системах автоматизации перевода.

Затем предложения разделяются на слова и знаки препинания. Далее идёт проверка частей предложения по словарю, и каждому слову сопоставляется часть речи (*POS tag*). Если слово не найдено, то ему сопоставляется тег *UNKNOWN*. Например, тег *ADV* означает наречие. Знакам препинания сопоставляются соответствующие теги.

При создании словаря для русского модуля программы автором доклада был использован русский семантический словарь рабочей группы AOT.ru (`http:\www.aot.ru`), который распространяется на основе лицензии LGPL. В этом словаре используются двубуквенные обозначения частей речи. Но такой формат плохо подходит для описания в правилах, поэтому был создан специальный набор обозначений.

Таблица преобразования присутствует в файле `https://github.com/languagetool-org/languagetool/blob/master/languagetool-language-modules/ru/src/main/resources/org/languagetool/resource/ru/russian_tags.txt`.

Автором была выполнена конвертация словаря в формат fsa, который использован в LanguageTool.

В файле *russian_ tags.txt* подробно описаны все варианты частей речи, которые используются в программе. Для описания форм слов используются только один тег, поэтому для обозначения некоторых частей речи теги разделены на логические поля знаком двоеточия. Например *NN:Masc:PL:Nom* означает существительное мужского рода множественного числа в именительном падеже. При создании правил используются регулярные выражения, что позволяет очень просто отобрать слова в необходимой форме, например выражение *NN:.*:.*:Nom* позволяет найти существительные в именительном падеже.

Анализ русского текста представляет сложную задачу. Вот основные особенности:

- богатая морфология
- возможен различный порядок слов

Рассмотрим пример XML-правила. Данное правило ищет в тексте слова *оба, обоих, обоим* или *обоими* и следующие за ними существи-

тельные женского рода. Предлагает исправить первое слово на *обе, обеих, обеим, обеими.*

```
<rulegroup default="on" id="Oba_obe" name="Числительные оба/обе">
  <rule>
    <pattern>
      <token regexp="yes">оба|обоих|обоим|обоими</token>
      <token postag="NN:Fem:.*" postag_regexp="yes">
      <exception negate_pos="yes"
                 postag="NN:Fem:.*"
                 postag_regexp="yes">
      </exception>
      </token>
    </pattern>
    <message>Числительное оба (обоих, обоим, обоими) употребляется
             только с существительными мужского рода:
    <suggestion>
      <match no="1" postag="Num:.*:(.*)"
             postag_regexp="yes"
             postag_replace="Num:Fem:$1">
      </match>
      <match no="2"></match>
    </suggestion>.
    </message>
    <url>http://ru.wikipedia.org/wiki/Собирательное_числительное
    </url>
    <short>Грамматическая ошибка.</short>
    <example type="correct">По обеим сторонам.</example>
    <example type="incorrect">По <marker>обоим сторонам</marker>.
    </example>
  </rule>
</rulegroup>
```

Рассмотрим структуру правил. Правила делятся на категории (элемент *category*), группы правил (элемент *rulegroup*), и правила (элемент *rule*). Далее идёт уже описание конкретного правила. Между элементами *pattern* включена искомая фраза. Элементы *token* включают в себя конкретное слово или знак препинания. Элемент $regexp = "yes"$ означает, что используется регулярное выражение, *postag* задаёт части речи, $postag\_regexp = "yes"$ — в описании частей речи используются регулярные выражения. Тег *message* содержит сообщение, которое выводится пользователю программы. Между тегами *suggestion* представлен вариант исправления. Элемент *match no* =

"1" означает подстановку первого слова из искомого шаблона. Элементы *short* описывают сообщение, выводящееся в контекстном меню OpenOffice.org. Элементы *example* содержат контрольный пример. Этих элементов может быть несколько. Элемент $regexp = "yes"$ означает, что используется регулярное выражение. Элемент *postag_ replace* задает словоформу, в которую нужно преобразовать искомое слово. Элемент *url* содержит адрес в сети Интернет, где размещена страничка с описанием грамматической ошибки.

В настоящее время в состав LanguageTool входит 227 правил для проверки русскоязычных текстов, с помощью которых можно обнаружить ошибки различных типов: грамматические, стилистические, пунктуационные, логические, употребление заглавных букв. Таким образом, программа может найти ошибки, которые не может обнаружить коммерческое программное обеспечение.

В докладе было показано, что можно только на основе свободного программного обеспечения создать полнофункциональную систему проверки текста.

Литература

[1] Naber Daniel, A Rule-Based Style and Grammar Checker, Diplomarbeit, Universität Bielefeld, Bielefeld, 2003

[2] Marcin Miłkowski, Developing an open-source, rule-based proofreading tool, Software – Practice and Experience, 2010

[3] Jarosław Lipski, Marcin Miłkowski, Using SRX standard for sentence segmentation in LanguageTool, in: Z. Vetulani (ed.), Human Language Technologies as a Challenge for Computer Science and Linguistics, Wydawnictwo Poznańskie, Fundacja Uniwersytetu im. A. Mickiewicza, Poznań, 2009