При поддержке
Министерства информационных технологий и связи
Российской Федерации
АНО «Институт логики, когнитологии и развития личности»
ALT Linux

Четвёртая конференция разработчиков свободных программ на Протве

Обнинск, 23–25 июля 2007 года

Тезисы докладов

Москва,
Институт Логики,
2007

В книге собраны тезисы докладов, одобренных Программным комитетом Четвёртой конференции разработчиков свободных программ. Круг рассматриваемых тем весьма широк: от новейших системных и прикладных разработок до правовых проблем, вопросов организации работы в проектах и аналитики.

Программа конференции

22 июля

20.30–22.00: Регистрация в холле гостиницы

23 июля

10.00–12.30: Регистрация в холле гостиницы

12.00–12.30: Кофе

Дневное заседание
12.30–14.15

12.30–12.40: Открытие конференции

12.40–13.20: Доклад представителя Министерства информационных технологий и связи Российской Федерации

Алексей Смирнов

Свободное программное обеспечение для государства и общества ..... —

14.00–14.45: Обед

Вечернее заседание
14.45–19.15

16.05–17.05: Кофе-брейк

24 июля

Утреннее заседание
9.30–14.00

11.30–11.50: Кофе-брейк

14.00–14.40: Обед

16.40–17.10: Кофе-брейк

Вечернее заседание
17.10–19.00

17.10–19.00: Круглый стол по проблемам использования Свободного ПО в образовании. Ведущие: Н. Н. Непейвода (УдГУ, Ижевск), Г. В. Курячий (ALT Linux, Москва).

25 июля

Утреннее заседание
9.30–13.40

11.45–12.15: Кофе-брейк

13.55–14.45: Обед

Дневное заседание
14.45–15.45

14.45–15.30: Дискуссия

15.30–15.45: Алексей Новодворский, Москва, ALT Linux. Заключительное слово.

Вне программы

Константин Осипов Москва, MySQL
Проект: MySQL http://dev.mysql.com

Процесс разработки ПО в MySQL: проблемы роста

Данный доклад посвящён организационным аспектам разработки ПО на основе опыта MySQL. MySQL — распределённый открытый проект, за плечами которого стоит коммерческая компания с высоким уровнем организации и нацеленности на коммерческую выгоду. Начавшись как надстройка над существующей нереляционной базой данных, проект до 2001 года существовал во многом за счёт индивидуальных пользователей и их поддержки. В этот период разработка велась двумя-тремя инженерами во главе с Майклом «Монти» Видениусом, одним из основателей MySQL. Период 2001–2007 гг. — период бурного роста, когда и штат разработчиков, и масштабы компании удваивались ежегодно. Немногие проекты выживают в подобной ситуации, не будучи переписанными «с нуля», и ни для одного проекта подобного рода масштабирование не проходит без потерь. Наблюдениям и опыту «инсайдера» и посвящён этот доклад. Как программиста, меня в первую очередь интересовали технические аспекты процесса разработки, о них в основном и пойдёт речь:

- из каких частей или модулей состоит продукт;
- кто работает над разными модулями, группы;
- коммуникация — списки рассылки, IRC, этикет распределённого общения;
- средства управления версиями;
- процесс сборки и управление релизами;
- тестирование;
- личные аспекты — как выжить между молотом Калифорнии и наковальней Хельсинки;
- что такое по-настоящему «открытый проект».

Литература

[1] *Geoffrey A. Moore* Crossing the Chasm. http://en.wikipedia.org/wiki/Crossing_the_Chasm

[2] *Torvalds L.* Linus Torvalds talk on Git. http://video.google.com/videosearch?q=torvalds+on+git

[3] *Ben Collins-Sussman, Brian W. Fitzpatrick* How to protect your open source project from poisonous people. http://video.google.com/videosearch?q=subversion+poisonous

Александр Боковой Москва, Samba Team, IBM Linux Technology Center

Проект: Samba http://www.samba.org, http://ctdb.samba.org

Кластерная самба

Карнавальная самба — танец индивидуальный или групповой, поэтому партнёра не подразумевает.
Википедия

Несмотря на активное развитие программных комплексов на основе веб-служб, обмен файлами в локальной сети с использованием специализированных протоколов сетевых файловых систем по-прежнему остаётся доминирующим. При этом за последнее десятилетие значительно выросли объёмы информации и усложнились типичные сценарии её использования: от однопользовательских приложений произошёл переход к многопользовательским, от монопольного владения файлами в рамках одной организации — к необходимости совместной работы над одними и теми же файлами в распределённой среде, где пользователи могут быть географически удалены друг от друга и от файл-серверов.

Традиционно в таких ситуациях используются кластеры для распределения нагрузки и повышения отказоустойчивости системы. Однако в случае файловых систем использование файлового кластера ограничивается следующими особенностями:

- Во-первых, для совместной работы клиентов с одними и теми же файлами через разные узлы кластера необходима координация системной информации между этими узлами. К такой служебной информации относятся прежде всего блокировки файлов, метаинформация о файлах и режимах их использования.

- Во-вторых, сама файловая система, используемая узлами кластера для хранения файлов, должна поддерживать общую работу с этими файлами с разных узлов системы. Такая файловая система обычно называется кластерной или распределённой.

Забудем на время о первой особенности и обратимся ко второй. На сегодняшний день существует несколько кластерных файловых систем, которые можно было бы использовать для организации такого распределённого хранилища: свободные GFS, OCFS, PVFS2, Lustre, коммерческие GPFS (IBM), CXFS (SGI), Lustre. Однако для всех из них существуют свои ограничения, главное из которых (помимо отсутствия поддержки некоторых свойств первой особенности) — отсутствие клиента для распространённых клиентских систем (Microsoft Windows, Mac OS X). То есть, для того, чтобы, скажем, клиенты из-под Microsoft Windows работали с файлами, размещёнными на GFS, необходимо использовать какую-то другую сетевую файловую систему, которая скорее всего не будет кластерной. Забегая вперёд, скажем, что на самом деле кластерных сетевых файловых систем для таких клиентов нет вообще.

Становится понятно, что в такой ситуации некоторого «медиатора» не избежать. Таким «медиатором» уже давно является Samba, в рамках которой реализуются многие из функциональных возможностей сети Microsoft Windows: файловый обмен, печать, доменная структура. На практике можно было бы использовать Samba для доступа к файлам, хранящимся на кластерных файловых системах, и раньше, однако тут как раз важную роль играют составляющие первой особенности.

Допустим, что мы захотим разместить файлы на кластерной файловой системе и предоставить их клиентам посредством немодифицированной версии Samba. В этом случае нам необходимо будет удостовериться, что кластерная файловая система поддерживает блокировки файлов, к которым обращаются с разных узлов. Мы очень быстро убедимся, что из свободных реализаций только GFS умеет это делать относительно надёжно, но даже она не обеспечивает быструю работу с такими файлами. Из коммерческих систем это умеют GPFS и CXFS, однако их производительность тоже падает существенно (на порядки) по сравнению с неблокируемым вариантом. Это практически сводит на нет ценность такой работы.

Что же предлагается сделать? С одной стороны, необходимо модифицировать Samba для обеспечения совместной работы поверх кла-

стерной файловой системы, с другой — нужно добиться таких изменений, которые не приведут к снижению скорости работы системы в некластерном варианте (один сервер). Над решением этой задачи Samba Team работала более шести лет, в рамках разных проектов и при поддержке разных компаний, которые были заинтересованы в получении этой функциональности. Только в 2006 году удалось реализовать прототип, который демонстрировал принципиальную возможность такой модификации с потенциалом роста производительности, о нём рассказывалось на предыдущей конференции. Затем по результатам оценки достоинств и недостатков полученного решения был разработан принципиально иной подход к кластеризации, позволивший получить существенные результаты: кластерная Samba работает быстрее обычной на одном узле приблизительно на 20–30 %, а при переходе к нескольким узлам скорость возрастает на два порядка.

Одной из архитектурных особенностей Samba является использование вспомогательных баз данных для обмена информацией между отдельными её компонентами. Эти компоненты могут исполняться в рамках разных процессов (и в случае кластерной реализации — даже на разных узлах), а данные могут жить как короткое время, на время отдельного запроса, так и в течение всего клиентского сеанса, который может охватывать несколько TCP-соединений.

Процессы, использующие эти базы данных, синхронизируют свой доступ с помощью побайтовых блокировок определённых служебных файлов на диске (доступ к файлам осуществляется через отображение их в оперативную память там, где поддерживается функция `mmap()`, но это особенность реализации). Понятно, что разместив эти базы данных на кластерной файловой системе, мы добьёмся того, что несколько узлов будут пытаться удерживать блокировки на отдельные фрагменты файлов и будут блокировать друг друга, замедляя работу.

Новый подход основывается на том, что у нас нет необходимости размещать большинство баз в общедоступном хранилище, поскольку они содержат информацию, специфическую для конкретного клиентского TCP-соединения. Вместо этого необходимо добиться того, чтобы информация, которая относится к файлам, с которыми работает тот или иной узел, хранилась на этом узле, сокращая расходы на сетевое взаимодействие и позволяя локализовать их обработку для наиболее часто возникающей ситуации «этот клиент работает с этими файлами только через этот сервер» (штатный режим докластерной эпохи).

В результате те узлы, которые чаще работают с определёнными файлами, становятся ответственными за информацию о них, «перетягивая» на себя хранение и обработку соответствующих служебных записей в базах данных. При этом сами файлы находятся в кластерной файловой системе и их изменения видны другим узлам (и через них — другим клиентам). Если кто-то ещё обратится к этим файлам, информация о них будет запрошена у ответственного узла, который может перестать быть таковым, если запрашивающий станет активно работать с этими данными.

Таким образом, получается распределённая система, в которой ответственность за информацию о тех или иных ресурсах мигрирует с узла на узел следом за активностью пользователей, работающих с этими ресурсами через конкретный узел, а сама система следит за тем, чтобы процессы на разных узлах своевременно обменивались информацией между собой. Помимо обмена информацией система выполняет и ряд других функций: она следит за доступностью узлов и перестройкой кластера в случае падения узлов, за доступностью кластерной файловой системы и координации экспорта ресурсов через другие протоколы (NFS, HTTP, ...).

Анатолий Якушин, Раиль Алиев Москва, OpenOffice.org
Проект: OpenOffice.org http://ru.openoffice.org

Анализ международного и российского опыта перехода на стандарт ISO/IEC 26300:2006. Сравнительное исследование возможных решений по процессу миграции

1 мая 2007 года исполнился год с того момента, как Всемирная организация по стандартизации (ISO) приняла формат ODF (OASIS Open Document Format for Office Application) в качестве международного стандарта под именем ISO/IEC 26300:2006.

Прошедший год показал, что миграция на новый формат, даже признанный такой авторитетной организацией, как ISO, требует значительных усилий и творческого подхода, подчас соизмеримого с процессом создания нового формата данных.

Авторами проведён анализ международного и российского опыта практического перехода на стандарт ISO/IEC 26300:2006 правительственных организаций, муниципальных учреждений, а также предприятий различной формы собственности. При проведении анализа отдельное внимание уделялось исследованию сопутствующих процессу перехода правовых механизмов.

В качестве источников фактологического материала использовались открытые данных из сети Интернет, публикации международной организации ODF Alliance, статистические и аналитические материалы международного сообщества разработчиков OpenOffice.org, а также прямые контакты с координаторами национальных проектов по разработке и внедрению программных продуктов, использующих ISO/IEC 26300:2006 через международный проект Native Language Confederation. Кроме этого, проводилось заочное выборочное групповое анкетирование активных участников проекта OpenOffice.org.

В результате проведённого анализа были выявлены наиболее типичные сценарии процесса миграции на стандарт ISO/IEC 26300:2006.

Миграция «через стандарт» Наиболее масштабный и осознанный сценарий миграции, при котором основной задачей является внедрение стандарта ISO/IEC 26300:2006 в практическую деятельность, а собственно программные средства документооборота являются вторичными. Данному сценарию свойственна глубокая юридическая и техническая проработанность деталей, широкая зона охвата процессом внедрения (на уровне государства, региона или крупной компании), частые компромиссные варианты между использованием свободного и проприетарного программного обеспечения.

Миграция «через программное обеспечение» Распространённый вариант миграции, при котором основной задачей является внедрение свободного офисного приложения (наиболее часто OpenOffice.org), либо свободной платформы (как правило один из дистрибутивов Linux) с целью легализации используемого программного обеспечения и снижения издержек на закупку проприетарного ПО. При этом вопросы собственно использования ISO/IEC 26300:2006 отходят в данных проектах на второй план. Подобный сценарий миграции типичен для предприятий среднего и малого бизнеса, такие проекты отличаются зачастую невысокой

степенью проработанности деталей, испытывают острые проблемы массовой конвертации входящей и исходящей документации и проблемы этапности миграции.

«Хаотическая» миграция Типична для небольших предприятий и организаций с низкой производственной и организационной дисциплиной, где выбор программного обеспечения для организации документооборота отдан на откуп сотрудникам. При данном сценарии один или несколько работников по разным побудительным причинам начинают использовать программные продукты, поддерживающих ISO/IEC 26300:2006. В этом случае собственно формат ISO/IEC 26300:2006 не используется даже для внутреннего документооборота ввиду внутриофисной несовместимости.

В заключение можно привести десять элементов стратегии перехода на стандарт ODF, впервые сформулированных Hasannudin Saidin, одним из идеологов Малазийского проекта, и позже расширенных и дополненных в рамках ODF Alliance:

1. Продолжайте использовать старое офисное программное обеспечение для обработки проприетарных форматов файлов до полного завершения процесса миграции.

2. Используйте только такое программное обеспечение, которое поддерживает кроме ISO/IEC 26300:2006 ваши старые форматы файлов.

3. Начинайте переносить свои шаблоны в формат ODF от простых к сложным.

4. Чаще используйте Adobe Portable Document Format (PDF) при обмене информацией с внешними корреспондентами.

5. Старайтесь внедрить ODF сразу в рабочей группе или подразделении.

6. Выявите и обучите потенциальных местных экспертов и «гуру» для оказания быстрой помощи другим пользователям.

7. Установите точные правила и сроки по переходу на ODF.

8. Активно используйте ресурсы и информацию таких организаций, как ODF Alliance и OASIS ODF Adoption Committee.

9. Взаимодействуйте с другими организациями, заинтересованными в процессе миграции, в том числе и через национальные группы по поддержке OpenOffice.org.

10. Настоятельно рекомендуйте разработчикам заказного программного обеспечения для вашей компании поддерживать в своих проектах ODF.

Дмитрий Левин Москва, ALT Linux
Проект: Sisyphus http://git.altlinux.org

От SRPMS к GEAR

Аннотация

Основной смысл хранения исходного кода пакетов в git-репозитории [1] заключается в более эффективной и удобной совместной разработке, а также в минимизации используемого дискового пространства для хранения архива репозитория за длительный срок и минимизации трафика при обновлении исходного кода.

Идея gear [2] заключается в том, чтобы с помощью одного файла с простыми правилами (для обработки которых достаточно sed и git) можно было бы собирать пакеты из произвольно устроенного git-репозитория, по аналогии с hasher [3], который был задуман как средство собирать пакеты из произвольных srpm-пакетов.

Структура репозитория

Хотя gear и не накладывает ограничений на внутреннюю организацию git-репозитория (не считая требования наличия файла с правилами), есть несколько соображений о том, как более эффективно и удобно организовывать git-репозитории, предназначенные для хранения исходного кода пакетов.

Одна сущность — один репозиторий

Не стоит помещать в один репозиторий несколько разных пакетов, за исключением случаев, когда у этих пакетов есть общий пакет-предок.

Плюсы: Соблюдение этого правила облегчает совместную работу над пакетом, поскольку не перегруженный репозиторий легче клонировать и в целом инструментарий git больше подходит для работы с такими репозиториями.

Минусы: Несколько сложнее выполнять операции fetch и push в случае, когда репозиториев, которые надо обработать, много. Впрочем, fetch/push в цикле выручает.

Несжатый исходный код

Сжатый разными архиваторами исходный код лучше хранить в git-репозитории в несжатом виде.

Плюсы: Изменение файлов, которые помещены в репозиторий в сжатом виде, менее удобно отслеживать штатными средствами (`git diff`). Поскольку git хранит объекты в сжатом виде, двойное сжатие редко приводит к экономии дискового пространства. Наконец, алгоритм, применяемый для минимизации трафика при обновлении репозитория по протоколу git, более эффективен на несжатых данных.

Минусы: Поскольку некоторые виды сжатия одних и тех же данных могут приводить к разным результатам, может уменьшиться степень первозданности (нативности) исходного кода.

Распакованный исходный код

Исходный код, запакованный архиваторами (tar, cpio, zip и т. п.), лучше хранить в git-репозитории в распакованном виде.

Плюсы: Существенно удобнее вносить изменения в конечные файлы и отслеживать изменения в них, заметно меньше трафик при обновлении.

Минусы: Поскольку git из информации о владельце, правах доступа и дате модификации файлов хранит только исполняемость файлов, любой архив, созданный из репозитория, будет по этим параметрам отличаться от первозданного. Помимо потери нативности, изменение прав доступа и даты модификации может теоретически повлиять на результат сборки пакета. Впрочем, сборку таких пакетов, если они будут обнаружены, всё равно придётся исправить.

Форматированный changelog

В changelog релизного commit'а имеет смысл включать соответствующий текст из changelog'а пакета, как это делают утилиты `gear-commit` (обёртка к `git-commit`, специально предназначенная для этих целей) и `gear-srpmimport`. В результате можно будет получить представление об изменениях в очередном релизе пакета, не заглядывая в spec-файл самого пакета.

Правила экспорта

С одной стороны, для того, чтобы srpm-пакет мог быть импортирован в git-репозиторий наиболее удобным для пользователя способом, язык правил, согласно которым производится экспорт из коммита репозитория (в форму, из которой можно однозначно изготовить srpm-пакет или запустить сборку), должен быть достаточно выразительным.

С другой стороны, для того, чтобы можно было относительно безбоязненно собирать пакеты из чужих gear-репозиториев, этот язык правил должен быть достаточно простым.

Файл правил экспорта (по умолчанию `.gear-rules`) состоит из строк формата «`директива: параметры`», параметры разделяются пробельными символами.

Директивы позволяют экспортировать:

- любой файл из дерева, соответствующего коммиту;
- любой каталог из дерева, соответствующего коммиту в виде tar- или zip-архива;
- unified diff между любыми каталогами, соответствующими коммитам.

Файлы на выходе могут быть сжаты с помощью gzip или bzip2. В качестве коммита может быть указан как целевой коммит (значение параметра `-t` утилиты gear [2]), так и любой из его предков при соблюдении условий, гарантирующих однозначное вычисление полного имени коммита-предка по целевому коммиту.

Правила экспорта из gear-репозитория описаны детально в документации к gear [4].

Основные типы устройства gear-репозитория

Правила экспорта реализуют основные типы устройства gear-репозитория следующим образом:

Архив с модифицированным исходным кодом

С помощью простого правила «`tar: .`» всё дерево исходного кода экспортируется в один tar-архив. Если у проекта есть upstream, публикующий tar-архивы, то добавление релиза в имя tar-архива, например, с помощью правила «`tar: . name=@name@-@version@-@release@`», позволяет избежать коллизий.

Архив с немодифицированным исходным кодом и патчем, содержащем локальные изменения

С помощью двух правил,

```
tar: v@version@:.
diff: v@version@:. .
```

можно экспортировать всё дерево исходного кода в виде tar-архива с немодифицированным исходным кодом, помеченным тэгом `v@version@`, и патча, который нужно приложить к дереву с немодифицированным кодом для того, чтобы получить дерево с исходным кодом. Тэг `v@version@` должен быть зарегистрирован с помощью утилиты `gear-update-tag`.

Архив с немодифицированным исходным кодом и отдельными патчами

Если дерево с немодифицированным исходным кодом хранится в отдельном подкаталоге, а локальные изменения хранятся в gear-

репозитории в виде отдельных патч-файлов, то правила экспорта могут выглядеть следующим образом:

```
tar: package_name
copy: *.patch
```

Такое устройство репозитория получается при использовании утилиты `gear-srpmimport`, предназначенной для быстрой миграции от srpm-файла к gear-репозиторию.

Смешанные типы

Вышеперечисленные типы устройства gear-репозитория являются основными, но не исчерпывающими. Правила экспорта достаточно выразительны для того, чтобы реализовать всевозможные сочетания основных типов и создать полнофункциональный gear-репозиторий на любой вкус.

Литература

[1] *Junio C. Hamano*, «GIT — a stupid content tracker», `http://members.cox.net/junkio/200607-ols.pdf`

[2] *Dmitry V. Levin*, «gear — Get Every Archive from git package Repository», `http://git.altlinux.org/people/ldv/packages/?p=gear.git;a=blob_plain;f=gear.1.html;hb=html`

[3] *Дмитрий Левин*, Hasher: технология безопасной сборки пакетов // Первая международная конференция разработчиков свободных программ на Протве. Тезисы докладов. М., 2004. С. 28–30.

[4] *Sergey Vlasov*, «gear-rules — rule file for gear(1)», `http://git.altlinux.org/people/ldv/packages/?p=gear.git;a=blob_plain;f=gear-rules.5.html;hb=html`

Виталий Липатов Санкт-Петербург, Etersoft
Проект: WINE, EngCom http://winehq.org.ru
http://engcom.org.ru

Разработки Etersoft для миграции на Linux и вклад в сообщество

Аннотация

Рассмотрена история развития программного продукта WINE@Etersoft, созданного на базе свободного ПО. Перечислены сопутствующие инструменты и решения, позволяющие упростить процесс разработки, автоматизируя рутинный труд. Рассказано о свободных проектах, созданных при участии компании Etersoft. Свидетельствуется о возможности создания компании, успешно продвигающей популярные решения на базе свободного ПО.

Etersoft изначально была направлена на создание и поддержку полноценной и универсальной среды (операционной системы), которая по своим потребительским качествам не уступала бы системам семейства Microsoft Windows. Понятно, что эта задача включает в себя решение бесчисленного множества проблем, которые, впрочем, по силам решить, если взяться за дело дружно, что так или иначе мы видим на примере разработки свободного ПО в мире и в России.

Начав несколько лет назад с внедрения ALT Linux в сфере бухгалтерского учёта и торговли, мы столкнулись с очевидной проблемой: на компьютерах повседневно используется множество специализированных коммерческих программ, не имеющих свободных аналогов, и не могущих их иметь в силу того, что никто не будет бесплатно создавать правовую базу или обновлять еженедельно бухгалтерскую отчётность.

Таким образом, мы пришли к тому, что массовое внедрение Linux требует в начальный (и довольно длительный) период поддержки работы в нём разнообразных программ для Windows. Было принято решение участвовать в разработке проекта Wine, реализующего WinAPI на Unix-системах, особенно внимательно относясь к проблемам запуска популярных в России программ.

К концу 2005 года мы смогли представить первую коммерческую версию WINE@Etersoft. Была реализована возможность совместной

работы программ 1С:Предприятия 7.7 в сети и на терминальном сервере. Хочу отметить, что коммерческие версии часто вызывают возгласы «специалистов»: «ах, они закрыли исходники и нарушили GPL». Стоит сказать, что все наши исправления кода, естественно, открыты, исходники публикуются как в виде патчей, так и в составе исходных rpm-пакетов. Но поскольку WINE@Etersoft состоит из целого ряда программ и файлов, имеющих различные лицензии, для соблюдения прав третьих лиц все подобные файлы вынесены в отдельный пакет, называемый «коммерческой частью», в дополнение к «свободной части», имеющей лицензию LGPL.

Также мы публикуем свободные сборки Wine, выполняемые нами для всех дистрибутивов с нашими патчами.

Хочу отдельно отметить хорошие отношения, сложившиеся у нас практически со всеми отечественными разработчиками коммерческого ПО. Компания Аладдин обеспечила работу сетевых и локальных ключей HASP в WINE. Компания БЭСТ участвовала в исправлении проблем, связанных с использованием БЭСТ 4+ и БЭСТ 5. Проблемы с правовыми системами Гарант и Консультант+ мы исправляли, консультируясь с разработчиками. Компании 1С и ИнфоБухгалтер оказывали психологическую поддержку. Интерес отмечен со стороны практически всех производителей ПО, хотя многие ещё считают, что им «эти 5% рынка» неинтересны.

Многое пришлось взять на себя. На данный момент мы поддерживаем около трёх десятков дистрибутивов, и где-то столько же программ, работающих в Wine. Службе поддержки приходится отвечать на вопросы настройки ОС, различных сервисов, установки Windows-программ и их настройки. До сих пор производители ПО не готовы взять на себя поддержку пользователей своих программ в Wine, и пользователей обращаются к нам и дополнительно оплачивают поддержку для используемых программ.

Только недавно разработка WINE@Etersoft стала окупаться, рост популярности продукта связан с необходимостью лицензировать ПО на предприятии, повышением готовности Linux как настольной системы к коммерческим внедрениям и улучшением качества самого продукта.

На данный момент WINE@Etersoft — это единственное решение, позволяющее исполнять востребованные в России программы при использовании Unix-платформ, не требующее лицензии на Windows и имеющее адекватную поддержку.

Мы благодарны нашим клиентам — только благодаря их (финансовой) поддержке, их вере в то, что мы делаем нужное дело, мы смогли довести продукт до пригодного к эксплуатации уровня.

Безусловно, успехом мы обязаны и международной команде разработчиков Wine, и компании CodeWeavers, общаясь с сотрудниками которой мы делали свой вклад в Wine.

К сожалению, разработка Wine (а в настоящий момент это по сути сплошная отладка закрытого кода, копание в логах и дампах и написание юнит-тестов) очень сложна, и дело ведётся не так быстро, как хотелось бы. Даже когда необходимые исправления выполнены в коде, нужно потратить ещё в несколько раз больше усилий, чтобы сделать код идеальным, и добиться его включения в основную ветку Wine.

В настоящий момент в WINE@Etersoft сделаны следующие полезные улучшения:

- доступна возможность совместной работы с Windows-программами в гетерогенной сети;
- созданы различные тестовые программы, упрощающие жизнь службе поддержки и администраторам;
- разработан административный режим установки, когда единожды установленное Win-окружение доступно для всех терминальных или сетевых пользователей;
- исправлены проблемы с работой торгового оборудования (сканеров штрих-кода и фискальных регистраторов);
- поддержка сетевых ключей HASP, Smartkey Eutron, Sentinel и программы для их тестирования;
- исправлено множество проблем, мешающих нормальной эксплуатации: реализованы диалоги печати, исправлено управление окнами, работа с буфером обмена, оптимизирована загрузка процессора (более 200 зафиксированных ошибок);
- создана документация на русском языке.

Сейчас ведётся работа по обеспечению поддержки WINE@Etersoft на платформе Solaris благодаря инициативе обратившейся к нам компании Sun Microsystems. Имеются планы по поддержке MacOS.

Ещё нужно добавить, что в процессе работы над WINE@Etersoft был создан или доработан, внедрён ряд дополнительных программных средств:

Система продаж

Основанная на открытой CMS Joomla, система продаж, хоть и не является полноценным интернет-магазином, но позволяет клиентам в автоматическом режиме выписывать счёт, оплачивать и получать в электронном виде программные продукты. Имеется поддержка работы с партнёрами и разнообразная внутренняя статистика. Продаваемые продукты имеют уникальный регистрационный код, по которому можно проверить легальность продукта, зайдя на специальную страницу продукта.

Linux CIFS

Модуль ядра Linux, обеспечивающий монтирование сетевых файловых ресурсов. После проведения доработки (добавлены дополнительные флаги в системный вызов `open()`, позволяющие реализовать для Wine NT-семантику при открытии файлов) появилась возможность организовать в гетерогенной сети полноценный совместный доступ к файловым ресурсам, то есть совместная работа Windows-программ, запущенных как в Windows, так и в Linux, стала реальностью. В ходе испытаний в коде модуля CIFS был обнаружен ряд ошибок, которые были исправлены и начаты переговоры по внесению исправлений в основной код.

EterSafe

Кроме аппаратных средств защиты (например, HASP-ключей) производители ПО активно применяют методику привязки программ к уникальным характеристикам компьютера. Как правило, это реализовано через низкоуровневое обращение к контрольным суммам BIOS в специальном системном драйвере. Для Linux/FreeBSD был создан системный сервис EterSafe для обеспечения доступа из пользовательского процесса (через сокет) к необходимым уникальным кодам компьютера. Это позволяет реализовать защиту как Windows (в Wine), так

и Linux-программ. Драйвер имеет стандартный интерфейс и идеология его работы может быть перенесена и на другие ОС.

Транслятор SQL-запросов

Так сложилось, что ещё недавно было популярно использовать в качестве СУБД продукт компании Microsoft MS SQL Server. В частности, он необходим для работы SQL-версии программы 1С:Предприятие 7.7, которая у нас очень широко распространена. Чтобы обеспечить возможность полной миграции на свободное ПО, нами спроектирован и разработан транслятор SQL-запросов, решающий пока только частную задачу трансляции запросов из диалекта T-SQL в PortgreSQL и откликов сервера, что позволяет использовать свободную СУБД PostgreSQL на Linux-платформе для работы 1С:Предприятия 7.7.

PostgreSQL не в последнюю очередь был выбран потому, что он предлагается компанией 1С для работы их продукта 1С:Предприятие 8.1, а значит, будет тщательно протестирован и хорошо поддерживаться. К тому же, у нас используются новые типы данных, введённые в PostgreSQL для упрощения совместимости с MS SQL по заказу 1С: это типы `MCHAR` и `MVARCHAR`, позволяющие осуществлять регистронезависимое сравнение строк в них без дополнительных выражений.

Реализован транслятор в виде дополнительного модифицированного ODBC-драйвера PostreSQL, который представляется как ODBC-драйвер сервера MS SQL, и разбирает запросы, обращения к системным таблицам, адаптируя их для Postgres. Таким образом, не требуется вмешательства ни в сам сервер, ни в клиентскую программу. ODBC-драйвер выполнен в виде DLL-библиотеки, что позволяет использовать данное решение как в Wine, так и в Windows-системах.

Для построения кода разбора входных выражений применяются стандартные средства: лексический анализатор flex и генератор синтаксических анализаторов bison. На производительность транслятор влияет незначительно.

EterBuild

EterBuild — это достаточно условное название системы сборки, которая применяется в компании для сборки бинарных пакетов под множество целевых платформ из одного src.rpm. Состоит она из нескольких частей:

Пакета `rpm-build-altlinux-compat`, позволяющего собирать rpm-пакеты, оформленные в соответствии с правилами ALT Linux, на прочих системах, включая Debian.

Пакета `etersoft-build-utils`, включающего в себя небольшие утилиты, существенно упрощающие жизнь разработчика и мантейнера. В нём есть средства от простой сборки пакета командой `rpmbb файл.spec`, до автоматической сборки новой версии пакета и установки её в изолированную среду `hasher` для тестирования работоспособности. Всё это достаточно удобно, стоит только сказать, что автор использует этот пакет для поддержки более чем пятисот пакетов в репозитории Sisyphus. Также имеются команды для частичного преобразования сторонних спеков в принятый формат, а также возможность преобразования зависимостей пакета в названия, принятые в целевой системе, чтобы сборка проходила с контролем зависимостей.

Поверх этого реализована система eterbuild, которая либо по заказу от системы продаж, либо по расписанию, либо по команде разработчика выполняет сборку пакета для различных указанных систем, причём сборка Linux-пакетов выполняется на одном сервере, с использованием `chroot`, а для других систем (FreeBSD и Solaris) используются (пока) отдельные машины.

Сама по себе система сборки Eterbuild — удобное решение для разработчиков (коммерческих) программ, которым нужно собирать бинарные пакеты или тестировать собираемость на множестве платформ.

Свободный англо-русский словарь компьютерных терминов EngCom

В результате активной работы над переводами свободных программ был создан, пожалуй самый полный, хотя и несколько неформальный словарь компьютерной терминологии, где для каждого английского термина найден более или менее удачный перевод. Данный словарь рекомендуется переводчиками KDE и GNOME, он использовался при переводе таких программ как GnuCash, LyX, Dia, K3b и прочих, он регулярно пополняется и доступен в форматах `dict`, `mova`, `stardict`.

Таким образом, и инструменты, применяемые в компании, и создаваемые решения свидетельствуют о возможности успешного применения в бизнесе свободных программ.

Владимир Рубанов, Денис Силаков Москва, ИСП РАН
Проект: LSB Infrastructure http://ispras.linux-foundation.org

Центр верификации ОС Linux: вклад в развитие стандарта LSB и тестирование Linux платформы

Аннотация

В докладе представлены активности российского Центра верификации ОС Linux при Институте системного программирования РАН в области стандартизации и обеспечения надёжности Linux-платформы на примере проектов OLVER (по заказу Роснауки) и LSB Infrastructure (по заказу Linux Foundation). Описываются текущие результаты и статус. Отдельное внимание уделяется рассказу о назначении, текущем состоянии и планах развития стандарта Linux Standard Base (LSB) и соответствующей инфраструктуры.

О Центре верификации ОС Linux

Центр верификации ОС Linux (`http://linuxtesting.org/`) был создан при Институте системного программирования РАН осенью 2005 года при поддержке Федерального агентства по науке и инновациям (Роснаука). Миссия Центра — продвижение платформы Linux путём обеспечения её высокой надёжности и совместимости с помощью открытых стандартов и наукоёмких технологий верификации и тестирования. В деятельности Центра участвуют как специалисты ИСП РАН с многолетним опытом в сфере разработки и контроля качества программного обеспечения, так и молодёжь — студенты и аспиранты ведущих ВУЗов (МГУ, МФТИ и МГТУ им. Баумана).

Стандарт Linux Standard Base (LSB)

Важнейшим показателем популярности и полезности операционной системы является количество приложений для неё. Что мешает развитию Linux в этом плане? На момент написания статьи на сайте `http://lwn.net/Distributions/` зарегистрировано 549 (!) дистрибутивов Linux. И там не учитываются версии, сделанные для внутреннего применения различными компаниями и отдельными энтузиастами. Но что такое Linux с точки зрения производителя приложений? По сути,

это комбинация системных компонентов, таких как ядро и библиотеки, которые в совокупности предоставляют интерфейсы прикладного программирования для программистов приложений. Проблема заключается в том, что каждый дистрибутив Linux представляет собой уникальную комбинацию различных версий таких компонентов, что в итоге может означать разные интерфейсы, как по составу, так и по поведению. Поэтому написать приложение, которое будет работать на всех дистрибутивах, да ещё и без перекомпиляции (что важно для многих производителей ПО), может оказаться не таким простым делом. И это серьёзно сдерживает рост числа приложений под Linux. Можно делать разные версии приложений под разные дистрибутивы, но это дорого. Производители приложений хотят создавать приложения «под Linux», а не отдельно под Red Hat или SuSe.

Именно для решения проблемы обеспечения переносимости приложений между различными Linux было предложено разработать открытый стандарт Linux Standard Base (LSB), суть которого заключается в фиксации состава бинарных интерфейсов и требований к их поведению для некоторого подмножества базовой функциональности Linux, на наличие которой можно рассчитывать в большинстве дистрибутивов.

В настоящее время стандарт имеет версию 3.1 и включает порядка 30 000 интерфейсов из более чем 40 библиотек. Большинство основных производителей дистрибутивов сертифицированы на соответствие LSB.

Проекты Центра по тематике LSB

Центр верификации ОС Linux активно вовлечён в LSB-сообщество. Первым проектом Центра в этом направлении был Open Linux VERification (OLVER) — `http://linuxtesting.org/project/olver`. В проекте был проанализирован текст основной части стандарта LSB Core для около 1500 системных функций Linux, были формализованы требования на поведение этих функций и построены тесты для автоматической проверки дистрибутивов Linux на соответствие этим требованиям.

Результаты проекта OLVER заинтересовали комитет по стандартизации LSB — в тот момент консорциум Free Standards Group (FSG), который предложил ИСП РАН долгосрочное сотрудничество в области построения инфраструктуры использования и развития стандарта LSB, а также разработки технологий автоматизации тестирования и создания собственно новых тестов для Linux. Впоследствии, в результате

слияния FSG с OSDL был создан консорциум Linux Foundation и сотрудничество с ИСП РАН было расширено. Проект получил название LSB Infrastructure и на прошедшем в июне 2007 года Linux Foundation Collaboration Summit были представлены результаты его первой фазы:

- Переработана схема и проведена чистка главной базы данных LSB, содержащей всю информацию о стандарте и его окружении.
- Построена первая версия веб-портала LSB-разработчиков — LSB Navigator (`http://linux-foundation.org/navigator/`).
- Выполнен дизайн новой системы LSB-сертификации.
- Разработаны средства автоматизации запуска и визуализации результатов тестирования — LSB ATK/DTK Managers.
- Разработаны две технологии автоматизированного создания тестов для соответствующих уровней качества.
- Разработаны новые тесты для 5 библиотек.

Деятельность ИСП РАН в этом проекте — лишь малая часть усилий мирового сообщества, направленных на решение наболевших проблем переносимости приложений. На фоне наблюдаемого в последнее время роста популярности Linux, LSB получил огромный импульс на саммите Linux Foundation из-за возросшей актуальности, что было подчёркнуто как топ-менеджерами ведущих международных ИТ-компаний, так и простыми инженерами на различных технических заседаниях. Был рассмотрен и одобрен двухлетний план развития LSB, направленный на обеспечение массового его внедрения.

Михаил Гильман, Андрей Михеев, Пётр Михеев
Dania Beach FL USA, Москва,
Nova Southeastern University,
Консалтинговая группа Руна,
средняя школа № 57 г. Москвы

Проект: DIFFR http://developer.berlios.de/projects/diffr

Проект DIFFR — открытая система для моделирования двумерных задач дифракции

Аннотация

DIFFR — графическая среда, в которой двумерная задача дифракции на неровной поверхности может быть решена при помощи различных асимптотических и численных методов. Среда содержит набор уже существующих приближенных методов. Предполагается, что новые приближенные методы будут разрабатывать и загружать в систему исследователи-физики. Система позволит как сравнивать различные приближенные методы, так и решать конкретные задачи дифракции при помощи наиболее эффективного в данном диапазоне физических параметров алгоритма.

Введение

Двумерные задачи дифракции световых, электромагнитных и звуковых волн на шероховатой поверхности описываются хорошо известным уравнением Гельмгольца с различными дополнительными условиями на границах раздела сред и условиями на поведение поля в бесконечности.

Особенностью этих задач является то, что кроме нескольких вырожденных случаев у этих задач нет аналитических решений. В настоящее время разработано большое количество численных и асимптотических методов решения этих задач. Эти методы являются разнородными, зависят от различных дополнительных параметров и предположений, «работают» только в определённом диапазоне параметров физической задачи, могут вычислять не все характеристики решения.

В мире имеется большое количество исследователей и научных школ, развивающих свои методы решения двумерных задач дифракции. Однако в силу разнородности существующих алгоритмов и про-

грамм, для решения конкретной задачи оказывается крайне сложно найти подходящий метод, по тем же причинам очень сложно сравнить различные существующие алгоритмы.

В этих условиях было принято решение разработать единую среду для моделирования двумерных задач дифракции. В данной среде исследователям будет предложено для каждого своего алгоритма реализовать некоторый набор интерфейсов. Далее в среду можно будет загружать различные алгоритмы и моделировать с их помощью задачи дифракции.

При помощи разрабатываемой среды можно как сравнивать различные приближенные методы в определённых областях значения параметров задачи, так и решать конкретные задачи дифракции, выбирая наиболее эффективный для данных параметров дифракционной задачи алгоритм.

Среда является платформонезависимой, написана на языке Java.

Некоторые понятия предметной области

Задача (Task). Задача — это объединение задаваемых входных данных (поверхность и падающее поле), результата (если он посчитан) и текущего алгоритма (вместе с параметрами).

Вычисление. Вычисление состоит в нахождении отражённого поля при помощи приближенного алгоритма при заданных падающей волне и препятствии.

Поверхность (Surface) Поверхность всегда периодическая. Задаётся *формой поверхности* и *проводимостью*. Форма поверхности задаётся периодом, параметром `shift` и коэффициентами Фурье.

Падающее поле (Impinging field). Падающее поле — это плоская электромагнитная волна. Задаётся углом к вертикали (в градусах), длиной волны (в сантиметрах), амплитудой и поляризацией (E или H).

Отражённое поле (Reflected field). Это одна из составляющих результата. Состоит из нескольких плоских волн.

Поверхностные токи (Surface current). Это одна из составляющих результата. Комплексная функция от координаты на поверхности. Изображается в виде двух кривых — модуль и фаза.

Краткое описание интерфейса системы

Окно программы делится на две части. Справа находится изображение поверхности и падающего поля. Левая часть состоит из трёх вкладок: *Input data*, *Result* и *Algorithm*.

Вкладка *Input data* используется для изменения входных данных. Внутри неё ещё две вкладки — *Surface* (для изменения поверхности) и *Impinging field* (для изменения падающего поля).

Вкладка *Result* используется для просмотра результата вычислений. Состоит из *Reflected field* (Отражённое поле), *Passed field* (Прошедшее поле) и *Surface current* (Токи в поверхности). Если выбранный алгоритм не умеет вычислять какую-либо из частей результата, вкладка для неё отображена не будет.

Вкладка *Algorithm* используется для выбора алгоритма и изменения его параметров.

Состояние. В нижней части окна отображается текущее состояние. Оно может быть одним из:

- до запуска
- запущена
- завершена нормально
- завешена аварийно
- остановлена

Масштаб. В левом верхнем углу изображения *поверхности* и *падающего поля* есть надпись `Scale:` + число. Это масштаб изображения — сколько пикселов приходится на единицу измерения (сантиметр).

Возможные действия в системе

New task Создать новую задачу, с входными данными по умолчанию. Текущая задача при этом будет потеряна.

Load task Загрузить ранее сохранённую задачу. Текущая задача при этом будет потеряна.

Save task Сохранить текущую задачу.

Exit Выйти из системы. Текущая задача будет сохранена в файл `autosave.task` и автоматически загружена при следующем запуске программы.

Start Начать вычисление результата.

Stop Прервать вычисление результата.

Add algorithm Добавить новый алгоритм.

Remove algorithm Убрать ранее добавленный алгоритм.

Текущее состояние проекта

Проект опубликован на портале свободного ПО `http://developer.berlios.de`. В систему загружен первый приближенный метод решения задачи дифракции — асимптотический метод малых возмущений.

Руслан Хихин Москва, ОАО «Концерн Моринформсистема „Агат“»
Проект: ALT Linux, МСВС `http://www.morinsys.ru`

Опыт использования Linux в ОАО «Концерн Моринформсистема „Агат“»

Аннотация

В докладе рассмотрен вопросы, связанные с использованием Linux в ОАО «Концерн Моринформсистема „Агат“».

1. Специфика заказов ОАО «Концерн Моринформсистема „Агат“».
2. Доступ к исходному коду ОС — необходимое условие для нормальной разработки.
3. Сертификация средств защиты информации по требованиям безопасности информации.
4. Перспективы и преимущества использования Linux.

Рассмотрены проблемы и перспективы использования Linux на примере использования МСВС и дистрибутивов ALT Linux.

Преамбула

- «НПО Агат» имеет богатый опыт разработки собственной аппаратуры и ОС в 1950–1980-е годы. В «НПО Агат» в 1950–1980-е годы разрабатывались собственное математическое обеспечение, которое включало такие программы, как управление задачами,

управление памятью, резервирование вычислительных процессов, восстановление вычислительных процессов и задач при сбое и отказе оборудования.

- Неудавшийся опыт применения MS Windows и удачные решения на основе ОС Unix в 1980–1990-е годы.

Начиная с середины 1980-х годов на смену машинам с прошиваемой памятью пришли компьютеры с имеющейся операционной системой. Пришлось отказаться от разработки собственного системного программного обеспечения и перейти на имеющиеся операционные системы.

В результате вопросы резервирования и устойчивости комплексов к сбоям и отказам было переведены на уровень пользовательских задач, а многие имеющиеся наработки не могли быть применены в наших продуктах.

Требования, предъявляемые к ОС для применения на наших заказах

Исходя из опыта разработки можно выделить следующие требования, предъявляемые к операционным системам:

1. Возможность *модификации исходного кода* ОС и его программного обеспечения.

2. Возможность *предоставить исходный код* для сертификации. Важным моментом является необходимость сертификации гостехкомиссией наших программных продуктов. Одним из её требований является предоставление исходного кода наших программ и программ, работающих в наших комплексах. В случае с МСВС это означает просто ссылку на соответствующее решение, а в случае с ALT Linux "—- предоставление её кода и проведение различных мероприятий по доказательству необходимой функциональности, т. е. приравниванию её кода к нашим программам.

3. *Масштабируемость* решений. Всегда есть вероятность смены платформы применяемых процессоров, их мощности и т. п.

4. Необходимость обеспечения *взаимозаменяемости* рабочих мест и резервирования. В наших комплексах особенно важно обеспечение взаимозаменяемости вычислительных машин. Большим подспорьем было бы применение различных кластерных решений. Во многом их применение сдерживается закрытостью для нас исходного кода МСВС и его отставанием от современного уровня Linux.

Примеры использования ОС Linux в заказах «НПО Агат»

- Особенности наших вычислительных комплексов. Квалификация пользователей.

  – При выборе решений тех или иных проблем мы исходим из того, что квалификация пользователя в области навыков работы с компьютером минимальна. С одной стороны, это приводит к повышенным требованиям к интерфейсу пользователя, а с другой — к тому, что стереотипы нахождения тех или иных решений для поставленных задач не должны сильно отличаться от привычных пользователю. При решении задач важно, чтобы пользователь мог применить весь свой накопленный опыт. Вопрос состоит в том, чтобы пользователь не думал о применяемой ОС и о том, из каких частей состоит вычислительный комплекс, а думал о том, как наилучшим образом решить стоящую перед ним задачу.

  – Важным принципом является принцип ответственности и управляемости. Поскольку ответственным за принятые решения поставленной задачи является в первую очередь сам пользователь, важно, чтобы, с одной стороны, он мог в любой момент перехватить управление объектом, а с другой — имел полную информацию о текущей обстановке. И тут важен как вид представленных задачами комплекса решений, так и устойчивость вычислительного комплекса к различного рода отказам аппаратуры и сбоям в решении задач.

  – Техподдержка решений в течение 15–20 лет. Важной особенностью наших систем является долгий срок их службы. По отдельным экземплярам он доходит до 15–20 лет. Такой срок службы изделий требует предусмотреть возможность

модификации наших комплексов в процессе службы. Это опять поднимает вопрос об открытости исходного кода операционной системы.

- Применение ОС Linux в тренажёрах.
- Применение ОС Linux на объектах.

Вопросы сертификации

С самого начала возникновения «НПО Агат» наши системы сдавались военной приёмке и их программный код подлежал специальной проверке.

В современных условиях была введена Сертификация программного кода Гостехкомиссией РФ. Как известно, МСВС сертифицирована по третьему классу. Но применение только МСВС тормозит принятие современных решений.

МСВС слишком универсальна, а как всякая универсальная система, она всегда будет проигрывать операционной системе, разработанной под конкретную аппаратуру и конкретные условия. Сдача тренажёров показала, что наличие открытого кода у Linux позволяет проходить сертификацию единичных экземпляров вместе с кодом, созданным нами. Такой подход несколько удорожает стоимость получаемых продуктов, но позволяет создавать решения на современном научно-техническом уровне.

Проблемы и перспективы использования Linux в «НПО Агат»

- При рассмотрении возможных перспектив применение Linux позволяет в будущем модифицировать наши проекты и применять такие решения, как резервирование с учётом возможностей новой Samba, применение репликаций PostgreSQL на основе Slony, кластеризации ядра на основе Mosix и т. п.
- Система восстановления работоспособности. Важным моментом является на сегодня разработка системы восстановления на основе Linux, и в частности, на основе применения технологии LiveCD.

- Применение не-Intel архитектур. На сегодня актуальным становится применение архитектур, отличных от Intel, например, Sparc. При переносе наших программ на новые архитектуры важным подспорьем являются возможности Linux как кроссплатформенной среды: возможность переносить решения на другую архитектуру путём простого перетранслирования проектов.

Георгий Курячий, Александр Потапенко Москва, ALT Linux, ВМК МГУ

Проект: Babylon http://phobos.cs.msu.su/babylon/

Система хранения и публикации документации Babylon

Аннотация

В докладе рассматриваются особенности организации хранилища свободной документации, дающего возможности комфортного написания, редактирования и публикации текстов в небинарных форматах логической разметки, а также приводится прототип подобного хранилища. Кроме инфраструктурной части, описывается подсистема преобразования текстов из одного формата в другой с использованием графового представления логической структуры документа.

Введение

В современных сообществах разработчиков и пользователей свободного ПО важную роль играют обмен опытом и накопление информации о существующих решениях. Поэтому часто возникает необходимость создания хранилищ текстов (в первую очередь технических), обеспечивающих возможности максимально комфортного написания, редактирования и публикации свободной документации. В идеале предполагается, что затраты на обновление и переиздание документа из такого хранилища должны быть существенно ниже затрат на соответственно вхождение и издание. Соблюдение этих требований поможет привлечь к написанию документации большее количество авторов.

Существующие системы документооборота обычно относятся к одному из следующих классов:

- single-source, single-target;
- single-source, multi-target;
- multi-source, single-target;
- multi-source, multi-target.

Первые три класса ограничивают свободу пользователя, навязывая ему один или небольшое число форматов документов. Распространённой проблемой подобных систем являются также трудности с модификацией существующих и введением новых форматов — довольно часто этот вопрос решается в пользу удобства разработчиков, а не авторов текстов.

Реализация систем класса «multi-source, multi-target» осложняется тем, что в общем случае речь идёт о преобразовании текста из произвольного формата в произвольный. Однако если ограничиться лишь логической разметкой, представленной в небинарном виде, то для технических текстов задача преобразования становится гораздо проще.

Система Babylon

Основными составляющими системы являются преобразователь форматов и хранилище документов (текстов и бинарных приложений к ним) с подсистемой организации документооборота (набором скриптов для предварительной и постпубликационной обработки, а также редактором связей).

Преобразование документов в системе осуществляется в несколько этапов, на каждом из которых текст представляется в своём формате. В частности, вводится специальный промежуточный формат хранения, сокращающий программистские затраты по написанию конвертеров документации, а также форматы моделирования разметки, позволяющие избавиться от синтаксического анализа разметки при преобразовании текстов.

Каждая из подзадач, возникающих при таком подходе, легко решается силами программиста, администрирующего систему. Планируется также минимизировать его усилия по добавлению в систему нового типа документа, входного или выходного формата с помощью языков описания разметки.

В настоящее время в рамках проекта Babylon реализован цикл преобразования документов, написанных в формате M-K, в HTML. Также имеются наработки по автоматизации построения парсеров и генера-

торов для входных и выходных форматов разметки, которые будут использованы при создании пользовательских средств описания.

Авторы приглашают к сотрудничеству людей, пользующихся «самодельными» форматами логической разметки, и будут рады любой информации о том, как эти форматы устроены и какими средствами обрабатываются.

Алексей Гладков Москва, ALT Linux
Проект: Sisyphus http://sisyphus.ru

Сборочная система sisyphus (giter factory)

Сизиф — большой репозиторий пакетов. Согласно статистике на летний период за день появляются 50 новых или обновлённых пакетов. Пересборка каждого пакета требует значительных ресурсов, в том числе временных.

В ближайшее время меняется структура репозитария. Благодаря упрощённой схеме хранения пакетов ожидается значительный рост количества пересобираемых за один день пакетов.

Основные недостатки существующей системы следующие:

1. Эта система обновляется посредством публикации srpm. Этот формат содержит снапшот исходных текстов и *не позволяет интегрировать разработку с публикацией в репозитории*.

2. *Трудности групповой разработки*. Так как все пакеты приходят в виде srpms, то разработчикам зачастую приходится обмениваться патчами на снапшоты, не учитывая изменения после релиза.

3. *Задержки перед публикацией*. Эта проблема вытекает из первой. Так как разарботчик сам создаёт и отправляет srpm в incoming[1], то проходит некоторое время между фактическим релизом и публикацией в репозитории.

4. *Сложности публикации*. Публикации пакетов происходят с задержками, складывается ситуация, при которой зависимые пакеты от разных разработчиков могут собираться не в том порядке,

[1] Точка входа пакетов в репозиторий Sisyphus.

в каком были собраны srpm. Это приводит к нарушению сборочных зависимостей и неудовлетворённым зависимостям в процессе сборки.

5. *Сохранение обратной совместимости*. Многие мантейнеры не имеют возможности всегда пользоваться актуальными средствами в силу разных причин. Соответственно, переход на новые средства будет происходить в течение некоторого периода.

Для решения этих проблем была разработана другая сборочная система.

Прежде всего потребовалось отказаться от srpm как транспорта исходных текстов. В новой системе используется `gear` как основное средство получения снапшота исходных текстов. Это даёт возможность более прозрачно получать пакет из дерева разработки.

Сборочную систему условно можно разделить на две части:

Общедоступное хранилище репозиториев (`git.alt`) Общее хранилище обеспечивает возможность удобно вести совместную разработку. В `git.alt` можно хранить репозитории своих проектов. А также можно получать анонсы изменений в других репозиториях.

Система публикации пакетов Новая система публикации пакетов интегрирована с общим хранилищем. Таким образом, процесс разработки и публикации становится прозрачным. Для публикации пакета разработчику нужно лишь дать указание сборочной системе с именем git-тега.

Для обратной совместимости существует возможность публиковать пакеты в виде srpm'ов. В этом случае rpm-пакет конвертируется средствами `gear` в git-репозиторий и дальнейшая работа ведётся уже с ним.

Как только указание получено, репозиторий встаёт в очередь на сборку. Все запросы от разных пользователей сериализуются по времени.

Процесс пересборки параллелится по архитектурам, но не по положению в очереди. Такой подход позволяет просто сформировать порядок сборки зависимых пакетов.

После удачной пересборки пакет сразу же публикуется. Следующие пакеты из очереди пересобираются уже на новом публичном репозитории.

Андрей Михеев Москва, Консалтинговая группа Руна
Проект: RUNA WFE http://sourceforge.net/projects/runawfe

Проект RUNA WFE — свободная система управления бизнес-процессами предприятия

Аннотация

RUNA WFE — это open source решение по управлению бизнес-процессами, основанное на популярном workflow-ядре JBOSS-JBPM. Система ориентирована на конечного пользователя, является платформонезависимой (написана на Java), распространяется под лицензией LGPL.

Характеристики системы:

- графический редактор бизнес-процессов;
- удобный веб-интерфейс пользователя;
- гибкая система определения исполнителей на основе ролей;
- боты для выполнения автоматических заданий;
- возможность интеграции существующих разнородных приложений предприятия;
- простая интеграция с существующими реляционными базами данных;
- система безопасности, позволяющая интеграцию с LDAP/MS Active Directory;
- локализация на английский, французский, немецкий, голландский, испанский, итальянский, русский, украинский и китайский языки;
- поддержка операционных систем Windows, Linux, Solaris, FreeBSD.

Введение

Процессный подход к организации управления предприятием получает все большее распространение. В соответствии с этим подходом деятельность предприятия представляется в виде множества бизнес-процессов — наборов заданий, выполняемых как людьми, так и информационными системами предприятия.

На практике процессный подход на предприятиях реализуют компьютерные системы управления бизнес-процессами, также называемыми *workflow-системами*. Эти системы позволяют быстро разрабатывать и изменять бизнес-процессы предприятия, не меняя кода системы.

На предприятии с устойчивыми повторяющимися операциями внедрение, настройка и сопровождение workflow-системы оказывается эффективнее традиционного подхода к автоматизации, при котором для различных задач и подразделений разрабатываются отдельные приложения, бизнес-логика которых рассеивается по компонентам этих приложений. Разработать такие приложения оказывается дороже и дольше, чем внедрить workflow-систему, не говоря уже о поддержке и изменении бизнес-логики.

Workflow-систему можно также рассматривать как центральную часть современных систем масштаба предприятия, интегрирующую разнородные приложения в единую корпоративную информационную систему.

В течение последних лет активно развиваются свободные workflow-системы, они начинают составлять реальную конкуренцию проприетарным системам.

Демонстрация системы

Во время доклада предполагается продемонстрировать работу системы и редактора бизнес-процессов как на примерах разработки бизнес-процессов коммерческой организации, так и на примерах разработки и исполнения процессов административного управления госорганизации.

В качестве примеров бизнес-процессов коммерческой организации будут рассмотрены некоторые процессы консалтинговой группы Руна, в качестве примеров административного управления будут рассмотрены процессы, выделенные в рамках пилотного проекта по переводу Администрации г. Алексин Тульской области на стандарт ISO/IEC 26300:2006.

Краткое описание системы

Система раздаёт задания на выполнение как реальным людям, так и приложениям специального вида (ботам). В случае заданий для людей в задании содержится графическая форма, которая будет показана исполнителю, в случае исполнителей-ботов это не обязательно: боту передаётся имя задания и необходимые для его выполнения данные.

Для работы с заданиями в системе предусмотрен компонент «Task list». Он содержит очереди поступивших заданий, сортировки и филь-

тры. Графические формы заданий показываются при помощи компонента «Проигрыватель форм».

Ядро системы содержит набор определений бизнес-процессов и набор выполняющихся экземпляров бизнес-процессов.

Для просмотра состояний экземпляров бизнес-процессов в системе предусмотрен административный интерфейс.

Графический редактор бизнес-процессов является отдельным приложением, он позволяет создавать графы бизнес-процессов, задавать роли, переменные и графические формы для исполнителей заданий.

В системе можно конфигурировать ботов. (Приложения специального вида, которые, как и обычные пользователи, могут выполнять задания.)

Также система содержит подсистему управления правами доступа.

Краткая история проекта

Проект начался в сентябре 2003 года. В консалтинговой группе Руна система RUNA WFE эксплуатируется с июля 2005 г.: В настоящее время количество пользователей системы — около 600, одновременно работают с системой до 200 пользователей. Также система используется OpenSource-сообществом в различных странах — с портала sourceforge на дату 28 июня 2007 г. произведено 17 409 скачиваний системы.

Проект RUNA WFE стал дипломантом конкурса Java-технологий, проводившимся корпорацией Sun Microsystems при официальной поддержке Министерства информационных технологий и связи РФ, также проект получил Honorable Mentions статус на конкурсе JBoss Innovation Award в двух категориях: «Управление бизнес-процессами» и «Хранение информации».

Литература и ссылки

1. OnLine demo системы доступно по адресу: `http://runawfe.sourceforge.net/English/OnLineDemo/Online_demo.html`
2. Сайт проекта: `http://sourceforge.net/projects/runawfe`
3. Что такое системы управления бизнес-процессами: А. Михеев, М. Орлов. Цикл статей: «Перспективы workflow-систем». PCWEEK.

http://kis.pcweek.ru/Year2004/N23/CP1251/CorporationSystems/chapt1.htm
http://kis.pcweek.ru/Year2004/N28/CP1251/CorporationSystems/chapt1.htm
http://kis.pcweek.ru/Year2004/N43/CP1251/CorporationSystems/chapt2.htm
http://kis.pcweek.ru/Year2005/N36/CP1251/CorporationSystems/chapt1.htm
http://wf.runa.ru/Russian/Article/Current/supplement.html

Николай Шмырёв Москва
Проект: VoxForge http://voxforge.org

Синтез и распознавание русской речи с открытыми исходными данными

Аннотация

В докладе будет рассмотрена деятельность по поддержке русского языка в открытых программных пакетах CMU Sphinx и Festival. Основное внимание будет уделено проблеме создания открытых речевых баз и вариантам их использования.

Использование речи для коммуникации с персональными компьютерами и автоматическими системами очень перспективно и уже давно служит темой обширных исследований. Бытует мнение, что пока область недостаточно развита для коммерческого применения, на самом деле это не так. Перечислим основные задачи и степень их решения:

1. *Синтез речи.* В довольно широком смысле эта проблема решена, речь синтезаторов почти неотличима от речи обычного человека. Проблемы остаются только в области синтеза эмоциональной речи.

2. *Управление машиной с помощью команд.* В целом, это решённая задача. Проблемы появляются только при наличии помех или необходимости проявления интеллекта машиной.

3. *Диктовка связного текста.* Эта задача в общем случае не решена, но значительные успехи в этой области имеются.

Нужно учитывать, что идеала тут достичь сложно, даже человек не идеален в распознавании и синтезе речи. В некоторых областях машина распознаёт даже лучше, например, приведём такую таблицу из [1]:

Область речи	Объем словаря	Человек (% ошибок)	Компьютер (% ошибок)
Набор цифр	10	0,009	0,72
Разговор по телефону	2 000	3,8	36,7
Статьи из журнала	5 000	0,9	4,5
Бессмысленный текст	20 000	7,6	4,4

Таблица 12.1: Сравнение числа ошибок человека и компьютера на сходных задачах

В отличие от подхода с глубоким изучением особенностей речи, требующего работы высококвалифицированных специалистов, в настоящее время преимущественно накапливаются огромные базы данных речи, которые затем обрабатываются статистическими методами. Некоторое знание языка при данном подходе необходимо, но требования значительно более слабые. Возникают трудности совсем иного рода — огромный размер баз данных. Современная база для синтеза — порядка 30 часов речи одного диктора, для распознавания — 140–150 часов речи 1 000 человек.

Системы распознавания и синтеза с открытым кодом, в том числе и от Microsoft (проект HTK), доступны для использования и изучения. Практикуются автоматические методы обучения (создание модели марковских процессов, деревьев решений, взвешенных автоматов и нейронных сетей).

Например, создание системы распознавания состоит из следующих шагов. Сначала изучается язык и строится его описание. По большому набору текстов строится статистическая модель языка, создаётся фонетический словарь. Например, точный русский фонетический словарь не существует, но достаточно хорошее приближение можно построить с помощью небольшого набора правил и доступного словаря ударений [6]. На вход программы тренировки передаётся словарь, текст и озвучка этого текста большим количеством дикторов. В результате получа-

ется модель, которую можно использовать для распознавания слитной речи.

Таким образом, основная работа состоит в сборе больших баз данных, ценность которых огромна. Такие базы позволяют:

- создавать синтезаторы и системы распознавания;
- сопоставлять различные методы создания речевых систем;
- изучать особенности языка;
- свободные базы обладают ещё одним преимуществом — их можно модифицировать и пополнять.

Перечислим проекты, посвящённые речевым технологиям:

- система для *разработки синтезаторов* Festival [2];
- система *распознавания речи* CMU Sphinx [3];
- ресурс VoxForge [4] посвящён *сбору речевых баз*. Текущая база русского языка содержит 10 дикторов по 200 предложений, 2,5 часа речи. На тестовом наборе данных точность распознавания 80 %. Английская база — 15 часов речи;
- *синтез русской речи* — проект FestLang [5], посвящённый и синтезу русской речи. Предлагается база данных для синтеза речи и готовый синтезатор.

В настоящий момент работа идёт по следующим направлениям:

- разработка DSP-алгоритма с хорошими возможностями моделирования;
- интонационная база русского языка для синтезатора;
- сбор базы для распознавания слитной русской речи;
- диалоговая система по управлению рабочим столом GNOME, в рамках Google SoC развивается проект gnome-voice-control.

Литература

[1] *Huang, Xuedong*. Spoken Language Processing. Prentice Hall PTR, 2001.

[2] Festival TTS. `http://festvox.org`

[3] CMU Sphinx. http://cmusphinx.sourceforge.net

[4] VoxForge GPL Speech Corpora. http://voxforge.org

[5] Festival Voices Development. http://festlang.berlios.de

[6] *Зализняк А. А.* Грамматический словарь русского языка.

Рената Пожидаева Томск, Томский государственный университет
Проект: eSpeak http://espeak.sourceforge.net/

Русификация речевого синтезатора eSpeak

Несмотря на существование развитых OpenSource-синтезаторов, таких как Festival, Flite и FreeTTS, создание русскоязычного синтезатора речи долгое время оставалось проблемой. До недавнего момента было известно две разработки, способных функционировать в среде GNU/Linux — закрытый ru_tts и попытка русификации Festival, проводимая в МГУ. Последний проект затруднительно использовать на практике вследствие недостатков самого Festival.

Синтезатор eSpeak [1] построен на основе универсальных алгоритмов, которые в принципе можно использовать для обработки текстов любого языка. На практике работа программ синтеза речи состоит из двух этапов: преобразование текста в последовательность звуков некоторого языка и получение звукового сигнала на основе последовательности звуков. Фонетические составляющие речи в eSpeak делятся на три группы: гласные, глухие согласные и звонкие согласные. Гласные генерируются из опорных последовательностей, каждая из которых определяет вершину форманты. Меняя частоту этой форманты, можно изменять звучание гласной, что позволяет добиться звуков, специфичных для любого языка. На практике оказалось, что подбирать параметры генератора вручную, сравнивая получившийся результат со звучанием некоторой гласной буквы, очень сложно. Поэтому приходится использовать специальную программу, поставляемую разработчиком, анализирующую запись гласной буквы и подбирающую параметры автоматически. Звонкие согласные целиком синтезируются алгоритмами

eSpeak на основе подобранных параметров. Глухие согласные воспроизводятся на основе записанных и очищенных звуков человеческого голоса без какой-либо дополнительной обработки. В некоторых случаях звонкие согласные можно получить смешиванием методов генерации гласных и глухих согласных.

На основе этих алгоритмов в синтезаторе eSpeak реализована поддержка нескольких европейских языков и диалектов. Для реализации поддержки русского языка потребовалось формализовать правила разложения русского текста на фонетические составляющие и подобрать параметры алгоритмов для получения гласных и согласных звуков. Преподаватель филологического факультета Д. А. Катунин составил эти правила. Они оказались связанными с позицией буквы в слове, окружением другими буквами, а также с ударением. Учёт расстановки ударения решается путём использования заранее заготовленного словаря. Проведены работы по реализации всех этих алгоритмов для поддержки русского языка. В данное время синтезатор удовлетворительно озвучивает некоторые русские тексты.

В настоящее время закончена работа над подготовкой правил преобразования, но необходима работа над звучанием и длительностью, и синтезатор способен удовлетворительно озвучивать русские тексты.

Работа синтезатора eSpeak реализована в среде GNU/Linux, в настоящее время ведётся перенос его в ОС Windows. Подготовлена его RPM-версия для дистрибутивов ALT Linux со словарём ударений.

К числу достоинств синтезатора «eSpeak» нужно отнести его гибкую организацию. Алгоритмы реализованы в виде iso-библиотеки и выполнена утилита для вызова синтезатора из командной строки. При желании функция синтеза речи может быть использована в любом приложении путём подключения библиотеки. Получаемый звуковой сигнал может быть воспроизведён с помощью библиотеки PortAudio или сохранен в wav-файле.

Литература

[1] J. Duddington. ESpeak text to speech [Электронный ресурс] / Sourceforge.net; ред. — Электрон. Дан. — UK: J. Duddington, 1995. — Режим доступа: http://espeak.sourceforge.net/, свободный.

Павел Сёмин Обнинск, Обнинский Государственный Технический Университет Атомной Энергетики (ИАТЭ)

Проект: libocr

libocr: ядро системы распознавания текста для свободных ОС

Аннотация

В докладе рассказывается о библиотеке libocr — ядре системы распознавания текста, предназначенной для использования в свободных ОС. Акцент делается на вопросах, возникающих при практическом использовании библиотеки. В частности, большое внимание уделяется проблеме адаптации системы к новым языкам распознавания.

Когда в жарких спорах о неоспоримых преимуществах свободного программного обеспечения заходит речь о системах распознавания документов, бойцы краснознамённой добровольной народной дружины имени Ричарда М. Столлмена неожиданно теряют свой пыл, грустнеют и стараются побыстрее сменить тему разговора. Попробуем разобраться в том, почему сторонникам свободного программного обеспечения хвалиться в этой области пока что нечем.

Системой распознавания текста называют программное обеспечение, выполняющее функции по преобразованию изображения рукописного, рукопечатного (т. е. написанного от руки, но печатными буквами) или печатного текста в один из текстовых форматов, пригодных для редактирования. Первые системы распознавания появились в 30-х годах прошлого века и представляли собой хитроумные устройства, принцип работы которых был основан на законах геометрической оптики. С тех пор за подобными системами закрепилось название Optical Character Recognition (сокращенно — OCR), хотя очевидно, что и принципы работы современных систем распознавания совсем другие, и распознаются не просто отдельные символы, а целые документы со всевозможными таблицами, формулами, рисунками и т. п.

Если с системами распознавания документов для Windows всё в порядке (тут безоговорочным лидером является ABBYY FineReader), то ситуация со свободными ОС выглядит удручающей. Во-первых, свободных систем распознавания документов как таковых в природе не существует (вполне возможно, что они просто неизвестны автору) — есть

только системы распознавания текста. В чем разница? Все очень просто — если система распознавания документов работает с документами, которые помимо текста содержат рисунки, таблицы, формулы и другие нетекстовые элементы и корректно их обрабатывает, то система распознавания текста воспринимает только текст, а нетекстовые элементы, если они есть, в лучшем случае игнорирует, а в худшем — пытается интерпретировать их как текст. На самом деле, это не такой уж серьезный недостаток — множество «документов-из-реального-мира» можно ввести в компьютер и с помощью системы распознавания текста. Плохо другое (оно же во-вторых) — большинство проектов по созданию свободной системы распознавания давно и успешно заброшены. Наиболее известными проектами по созданию свободной OCR являются:

ClaraOCR	`http://www.geocities.com/claraocr`
GOCR	`http://jocr.sourceforge.net`
HOCR	`http://hocr.berlios.de`
Kognition	`http://kognition.sf.net`
Ocrad	`http://www.gnu.org/software/ocrad/ocrad.html`
ocre	`http://lem.eui.upm.es/ocre.html`
OCRchie	`http://http.cs.berkeley.edu/~fateman/kathey/ocrchie.html`
OOCR	`http://sourceforge.net/projects/oocr`
Tesseract	`http://code.google.com/p/tesseract-ocr`
Проект «Открытый код»	`http://ocr.apmath.spbu.ru`

Почему более крупные и сложные проекты выживают и успешно развиваются, а системам распознавания все время «не везёт»? Одна из причин этого — высокая «наукоёмкость» задачи, из-за чего получить приемлемые результаты не удаётся ни с первой, ни со второй, ни с третьей попытки — в итоге у разработчика опускаются руки и проект отправляется на свалку истории. Как ни странно, получить точность распознавания порядка 90–95 % на ограниченном классе документов относительно легко, но такой точности совершенно недостаточно для практического использования системы. Доведение этого параметра до приемлемого значения в 99 % требует по-настоящему титанических усилий.

Теперь, ознакомившись с текущим состоянием дел в области разработки систем распознавания для свободных ОС, перейдём непосредственно к теме доклада — библиотеке libocr. Если быть предельно крат-

ким, то библиотека libocr представляет собой ядро системы распознавания текста, которое специально проектировалось для использования в свободных ОС. Основные отличия libocr от других систем распознавания текста состоят в следующем:

- объектно-ориентированный API на языке C, что позволяет использовать библиотеку из различных языков программирования;
- добавление нового языка распознавания происходит без модификации библиотеки и не требует большого объёма специальных знаний;
- изначальная поддержка UNICODE;
- лицензия LGPL.

Список внешних зависимостей библиотеки libocr минимален, благодаря чему не должно возникать больших проблем с её переносом на различные ОС:

- библиотека GLib версии не ниже 2.10.0;
- библиотека libtiff.

Чтобы показать, насколько проста в использовании библиотека libocr, обратимся к небольшому примеру:

```
#include <stdio.h>
#include <libocr/ocr.h>
gint main(gint argc, gchar **argv)
{
 OcrImage *image;
 OcrEngine *engine;
 gchar *text;
 if(argc != 2) return -1;
 /* Загрузить изображение. */
 image = ocr_image_new_from_file(argv[1]);
 if(image == NULL) return -1;
 /* Загрузить сведения о языке распознавания. */
 engine = ocr_engine_new("russian");

 if(engine == NULL) {
```

```
 ocr_image_free(image);
 return -1;
 }
 /* Распознать изображение. */
 text = ocr_engine_recognize_image(engine,image);

 if(text) puts(text);
 /* Освободить ресурсы. */
 g_free(text);
 ocr_engine_free(engine);
 ocr_image_free(image);
 return 0;
}
```

Хорошо видно, что непосредственно за распознавание изображения отвечает единственная функция `ocr_engine_recognize_image()`. Ещё две функции используются для загрузки изображения и сведений о языке распознавания: `ocr_image_new_from_file()` и `ocr_engine_new()`. Остальной код отвечает за обработку ошибок и освобождение ресурсов, а значит, может быть существенно упрощён при использовании libocr из более высокоуровневого языка программирования (например, C++).

Каким образом происходит распознавание изображения? В составе ядра системы распознавания можно выделить три функциональных блока, работающих строго последовательно:

- блок предварительной обработки;
- блок сегментации;
- блок распознавания.

Блок предварительной обработки решает задачу определения угла наклона строк текста на изображении документа и поворота изображения таким образом, чтобы строки текста располагались строго горизонтально.

Блок сегментации предназначен для определения границ отдельных символов, их пространственного положения и степени их логической взаимосвязи. Результатом работы блока сегментации является граф, описывающий логическую структуру документа. Узлами графа являются отдельные структурные элементы исходного документа, а дуги задают пространственные отношения между ними. Свойства структур-

ных элементов описываются в виде атрибутов, связанных с соответствующими узлами графа.

Блок распознавания выполняет функцию по классификации изображений отдельных символов. На вход блока поступает растровое изображение символа, а на выходе формируется код символа в кодировке UNICODE. Для классификации изображений используется предварительно обученная искусственная нейронная сеть (персептрон с двумя активными слоями нейронов).

Что нужно сделать для добавления нового языка распознавания? Первым делом нужно подготовить обучающую выборку — набор изображений отдельных символов, на которых будет происходить обучение нейронной сети. Вопрос о качественном и количественном составе обучающей выборки до сих пор остаётся открытым, однако в качестве отправной точки можно принять следующие рекомендации:

- изображения, входящие в обучающую выборку, должны быть максимально близкими к тем, которые будут предъявляться нейронной сети во время работы, поэтому лучше, если в выборку будут входить реальные отсканированные изображения;

- не имеет смысла добавлять в выборку несколько очень похожих изображений одного и того же символа — это замедлит процедуру обучения, но практически никак не скажется на качестве распознавания.

Далее запускается специальная программа-тренер, которая проводит обучение нейронной сети методом обратного распространения ошибки. Процесс обучения состоит из эпох, в течение каждой из которых нейронной сети предъявляются для обучения все изображения из обучающей выборки. По окончанию каждой эпохи состояние нейронной сети (матрицы весовых коэффициентов) сохраняются на диске, и могут в дальнейшем быть использованы для распознавания. Последний этап — отобрать ту нейронную сеть, которая обеспечивает наилучшее качество распознавания.

На данным момент была проведена серия экспериментов по обучению системы распознавания для работы со строчными символами русского алфавита. В некоторых случаях точность распознавания на ограниченном классе предъявляемых документов составляла 99,8 %.

Алексей Куклин Москва
Проект: OpenVZ http://openvz.org

Развёртывание и поддержка мультисервисных серверов с применением виртуализации OpenVZ. Общие вопросы и вопросы применения на базе Debian GNU/Linux

Некоторое время назад SWSoft лицензировала ядро своего решения для виртуализации Virtuozzo под названием OpenVZ на условиях GPL2, что позволяет использовать его в некоммерческих и бюджетных проектах.

OpenVZ — это реализация технологии виртуализации на уровне операционной системы, которая базируется на ядре Linux. OpenVZ позволяет на одном физическом сервере запускать множество изолированных копий операционной системы, называемых Виртуальные Частные Серверы (Virtual Private Servers, VPS) или Виртуальные Среды (Virtual Environments, VE).

Одним из применений этой технологии является создание мультисервисных систем, объединяющих множество виртуальных серверов в рамках одного физического. Такое решение наиболее востребовано в случаях, когда нагрузка на системы ещё не оправдывает разделение на разные физические машины, но уже достаточно высока, чтобы требовалось управление ресурсами и достаточная степень безопасности.

К сожалению, на сегодняшний день отсутствует полноценная методическая информация по развёртыванию и администрированию такого рода систем. В результате системный интегратор, разворачивающий систему, сталкивается с большим количеством неочевидных и недокументированных проблем.

В докладе описывается опыт решения задачи развёртывания и поддержки серверов, обеспечивающих выполнение различных сервисов на одной физической машине с разделением доступа при помощи OpenVZ. Освещаются общие вопросы организации подобных систем, преимущества, недостатки, часто встречающиеся проблемы и архитектурные ошибки, демонстрируются способы решения типовых задач администрирования. Практические примеры показаны на базе дистрибутива Debian GNU/Linux.

Михаил Якушин Москва, ALT Linux
Проект: Xen, Red Hat Cluster http://xensource.com/xen/

Опыт объединения технологий виртуализации и кластера высокой доступности

Аннотация

В данном докладе рассказывается об опыте развёртывания и тонкостях применения технологий кластеризации и виртуализации в рамках одной системы. В докладе описывается место и особенности применения обоих технологии, а также требования к аппаратной части.

В последнее время было достаточно много разговоров о технологиях виртуализации, об их достоинствах и недостатках. Чаще всего говорится о разделении ресурсов, изоляции служб и абстрагировании их от аппаратной части, и о недостатках: накладных расходах на виртуализацию и общем усложнении системы.

Виртуализация удобна тем, что позволяет создать контейнер с выделенными ресурсами (процессором, памятью, дисковым пространством, сетевыми интерфейсами). Все эти ресурсы удобно выделить изолированной службе, которая будет решать определённую задачу, такую как СУБД или http-сервер со всем ему необходимым.

Благодаря тому что контейнер не привязан к аппаратной части, теоретически его возможно запускать на некотором множестве серверов, что будет абсолютно прозрачно для пользователей этой службы. Конечно, чтобы это работало на практике, нужно выполнение некоторых условий.

- Однородность сетевой структуры. Обычно это означает, что физические сервера должны быть в одной подсети, но это может быть обеспечено и другими средствами.

- Физический сервер должен «уметь» исполнять код контейнера (например, невозможно запустить контейнер, созданный для архитектуры x86_64 на сервере с архитектурой i586).

- Контейнер должен иметь возможность добраться до своих данных. Эта проблема может быть решена разными способами. Например файловая система контейнера может находиться на NFS или на общем хранилище SAN.

При выполнении этих условий контейнер может быть запущен на группе серверов, но при этом должны существовать средства управления запуском/остановкой/перемещением контейнеров. Для этих цели возможно применять кластер высокой доступности.

Кластер рассматривает контейнер как службу (в понимании кластера) и он их запускает, останавливает, проверяет состояние. В случае сбоя одного из серверов или его выключения по каким-то другим причинам, кластерное ПО автоматически запускает контейнеры на имеющихся узлах. Также кластерное ПО может следить за доступом к общему хранилищу данных.

Автором был развернут такой кластер, и сейчас ведётся разработка технологии быстрого развёртывания подобных кластеров для широкого спектра аппаратных конфигураций.

В качестве технологии виртуализации был использован xen. Xen — многоплатформенная, свободная технология паравиртуализации, поддерживающая и полную виртуализцаию (на некотором аппаратном обеспечении). Паравиртуализация — это тип виртаулизации, в котором каждая виртуальная машина имеет своё специально модифицированное ядро ОС, взаимодействующее с гипервизором, для доступа к абстрагированному оборудованию и других целей. Причём в этой модели нет хост-системы как таковой. Все экземпляры запущенных ОС исполняются внутри виртуальных машин, но только одна из них, Domain0, имеет по умолчанию полный доступ к аппаратной части, остальные имеют доступ только к определённым устройствам. Гипервизор — это фактически операционная система, которая распределяет ресурсы между виртуальными машинами, которые исполняются как программы в обычных ОС. Гипервизор загружается до виртуальных машин, запускает Domain0, который, в свою очередь, через запросы к гипервизору управляет другими виртуальными машинами. Обычно Domain0 является минимальной системой с Linux, в которой присутствуют только драйверы, средства управления жёсткими дисками и файловыми системами, средства удалённого администрирования и, в нашем случае, кластерное ПО.

В качестве кластера высокой надёжности использовался Red Hat Cluster. Это комплексный программный продукт, состоящий из компонентов:

- cman — Cluster MANager. Базовый компонент, отвечающий за членство узлов в кластере, т. е. отслеживающий вход/выход узлов в кластере и обеспечивающий взаимодействие между ними.
- Dlm — distributed lock manager. Библиотека + модуль к ядру Linux, занимающийся кластерными блокировками, используется почти всеми компонентами кластера.
- Fence — система отключения питания от узлов, физически отрезает узел от кластера в случае сбоя на нём.
- Clvmd — Cluster LVM Daemon. Демон, занимающийся управлением логическими томами на общем хранилище кластера. Ключевой компонент, так как диски виртуальных машин удобно хранить в LVM, этот демон решает многие проблемы, связанные с синхронизацией LVM между узлами.
- GFS — global file system. Кластерная файловая система, позволяет нескольким узлам иметь общие данные.
- Rgmanager Resource Group manager. Опять же ключевая программа, она заниматься непосредственно управлением службами.

На основе этих технологий была построена следующая система.

Аппаратное обеспечение:

- 8 узлов (2 двухядерных процессора 4 Гб памяти) — непосредственно узлы кластера.
- Общее хранилище данных, соединённое с узлами по FiberChannel
- 2 балансировщика нагрузки.
- Сервер резервного копирования.

На жёстких дисках узлов храниться только dom0 и xen, все остальные данные на общем хранилище. Весь объем хранилища (несколько Tb) разделён при помощи LVM, на котором выделены тома для виртуальных машин и их данных. Создано несколько виртуальных машин, которые решают прикладные задачи (веб-серверы, почтовые серверы,

СУБД), эти виртуальные машины запускаются кластерным ПО на разных узлах кластера. Балансировщики находятся в режиме failover, все запросы из внешнего мира они перенаправляют на соответствующие IP-адреса во внутренней сети кластера, принадлежащие виртуальным машинам. На сервере резервного копирования запущено 2 виртуальные машины, одна из которых собственно осуществляет резервное копирование, а на второй запущена система мониторинга.

В результате получается гибкая и надёжная система, обеспечивающая удобство обслуживания, отсутствие необходимости останавливать службы при изменении конфигурации и функционирование при отказе до 50 % аппаратного обеспечения.

Александр Московский Переславль-Залесский,
Институт Программных Систем РАН

Проект: T-Sim

Библиотека шаблонных классов C++ для поддержки параллельных вычислений

Аннотация

Современные высокопроизводительные вычисления немыслимы без использования параллельных вычислений. В последние годы массовое использование многоядерных процессоров ставит перед большинством программистов задачи, которые ранее решались лишь узким кругом специалистов в суперкомпьютерных центрах. В библиотеке T-Sim программисту предоставляется набор шаблонных классов и функций C++, который предоставляет возможность создавать сравнительно компактный код, переносимый между различными мультикомпьютерами (многоядерные процессоры, SMP-системы, вычислительные кластеры, метакластеры). Основа библиотеки — инкапсуляция ряда понятий (таких как «переменная», «функция» и некоторых других), необходимых для написания программ, близким по стилю к программам на функциональных языках. Предлагается также ряд шаблонных функций, реализующих наиболее часто применяемые методы распараллеливания вычислений.

T-Sim представляет собой библиотеку шаблонных классов C++, разработанную и реализованную на основе концепции T-системы[1].

Отличительными особенностями как T-Sim так и OpenTS — современной реализации T-системы — является использование бесконфликтной модели вычислений на основе чистых (не имеющих побочных эффектов) функций и «неготовых значений» как средства синхронизации доступа к результатам вычислений. При попытке доступа к данным, находящимся в «неготовой переменной» из потока потребителя, проверяется, вычислены ли уже эти данные потоком-производителем. Если да, то поток-потребитель продолжает работу, если нет, то работа потока-потребителя приостанавливается до тех пор, пока какой-либо поток не вычислит данные для неготовой переменной.

Библиотека T-Sim состоит из набора заголовочных файлов и статически связываемой библиотеки C++. При написании программы пользователь определяет «неготовые значения» при помощи соответствующих шаблонных классов. «T-функции» — чистые функции, которые среда выполнения программ может вычислять параллельно, декларируются программистом при помощи макросов. Библиотека T-Sim позволяет осуществлять распараллеливание вычислений, передавая вызовы «т-функций» для вычисления либо в отдельные потоки, либо на удалённые компьютеры (другие узлы вычислительного кластера). Механизм распределения нагрузки между узлами вычислительного кластера может быть определён пользователем. Предоставлен набор классов-стратегий на основе которых программист может сконструировать алгоритм выравнивания нагрузки из готовых «запасных частей».

T-Sim — высокоуровневое средство параллельного программирования. В отличие от общепринятых сейчас средств создания параллельных программ, таких как Message Passing Interface (MPI) и Posix Threads, T-Sim позволяет создавать достаточно компактные, простые в поддержке программы. По сравнению с аналогичными высокоуровневыми средствами T-Sim обладает, на сегодняшний день, следующими основными преимуществами:

- Использование среды исполнения программ C++, допускающих низкоуровневую оптимизацию.

- Переносимость программ — однажды написанная, программа на T-Sim может работать как на многоядерном процессоре, многопроцессорной системе, так и в системе с распределённой памятью — вычислительном кластере, федерации нескольких вычислительных установок (мета-кластере).

- Существующая реализация библиотеки поддерживает работу в гетерогенной среде за счёт использования сериализации данных на основе XML. При этом при обмене данными между однотипными узлами вычислительной среды могут использоваться более эффективные механизмы обмена данными.

В настоящее время проект находится в стадии подготовки к выпуску первой публично доступной версии. Создан ряд тестов, разрабатывается набор высокоуровневых функций-шаблонов, которые бы описывали методы распараллеливания вычислений в наиболее часто встречающихся ситуациях. В частности, разработаны и проходят тестирование шаблоны Map и Reduce, которые должны предоставить параллельно вычисляемые аналоги одноимённых функций высшего порядка, известных из функционального программирования.

Работа выполнена при поддержке программы фундаментальных исследований Президиума РАН «Разработка фундаментальных основ создания научной распределённой информационно-вычислительной среды на основе технологий GRID».

Литература

[1] Abramov S., Adamovich A. I., Inyukhin A., Moskovsky A., Roganov V., Shevchuk E., Shevchuk Yu., Vodomerov A. OpenTS: An Outline of Dynamic Parallelization Approach // Parallel Computing Technologies (PaCT)–2005, Krasnoyarsk, Russia. September, 2005. LNCS, vol. 3606. P. 303–312.

Алексей Турбин Рязань, ALT Linux Team
Проект: Sisyphus http://sisyphus.ru

Автоматический поиск зависимостей в rpm-пакетах

Аннотация

Разработана модульная система автоматического поиска зависимостей для сборочной среды на основе менеджера пакетов RPM. Использование автоматических зависимостей на основе виртуальных пакетов (Requires и Provides) необходимо для более точного учёта внутренних связей между пакетами на стадии их установки или обновления. Модульная система позволяет независимо разрабатывать и добавлять в сборочную среду новые типы зависимостей. В последней части доклада кратко рассмотрен алгоритм автоматического поиска и оптимизации сборочных зависимостей (BuildRequires) на основе трассировки сборки.

Введение

На последних стадиях сборки rpm-пакета происходит формирование зависимостей Requires (требуемые пакеты) и Provides (зависимости, предоставляемые пакетом). Помимо непосредственного указания (вручную) зависимостей в spec-файле пакета, предусмотрена процедура автоматического поиска зависимостей. Для этого `rpm-build` запускает скрипты `find-requires` и `find-provides`, результат работы которых добавляется к непосредственным зависимостям пакета.

Автоматические зависимости, как правило, формируются при помощи виртуальных пакетов, т. е. для них используется отдельное пространство имён, которое не пересекается с названиями самих пакетов. В частности, виртуальные пространства имён используются для поиска автоматических зависимостей между модулями интерпретируемых языков программирования. Так, виртуальные зависимости для модулей языка Perl имеют вид `perl(Getopt/Long.pm)`, а для языка Python (версии 2.4) — `python2.4(getopt)`. Использование виртуальных зависимостей позволяет более гибко изменять содержимое пакетов (и даже переименовывать пакеты), а также защищает от ошибок упаковки и несовместимых изменений.

К существенным недостатком системы автоматического поиска зависимостей можно было отнести её монолитность: добавление нового типа зависимостей требовало ad hoc модификации скриптов

find-requires и find-provides в пакете rpm-build. Вместе с тем, разработка программ для поиска новых типов зависимостей велась независимо.

Внедрение модульной системы поиска зависимостей происходило в два этапа. На первом этапе некоторые алгоритмы поиска специальных зависимостей были выделены в отдельные программы; но условия запуска каждой из этих программ по-прежнему оставались жёстко закодированными в скриптах rpm-build. На следующем этапе система стала уже полностью модульной: процедуры вызова программ были полностью унифицированы, а в скриптах find-requires и find-provides остался только вспомогательный код диспетчеризации.

Модульная система поиска зависимостей

В новой системе скрипты find-requires и find-provides работают единообразно. Рассмотрим работу find-requires.

Этот скрипт получает на входе список всех файлов пакета, а на выходе формирует список зависимостей (всех типов). Сначала для каждого входного файла определяется его «тип» при помощи утилиты file(1). Затем список файлов и их типов обрабатывается модульными программами поиска зависимостей.

Для каждого типа зависимостей существуют две программы. Первая программа — с суффиксом (расширением) .req.files — выделяет список файлов (на основе их «типа»), для которых будет производиться поиск зависимостей данного типа. Например, программа perl.req.files среди всех файлов пакета выделяет файлы с кодом на языке Perl. Вторая программа — с суффиксом .req (в данном случае perl.req) — последовательно производит поиск зависимостей данного типа для каждого файла из полученного списка.

Модульность системы состоит в том, что в процессе поиска зависимостей запускаются все доступные программы (по шаблону *.req.files и *.req) из каталога /usr/lib/rpm.

В скрипте find-provides аналогичным образом происходит диспетчеризация по суффиксу .prov (вместо .req). Таким образом, для добавления нового типа зависимостей нужно написать четыре программы: программу выбора файлов и программу обработки файлов, соответственно для Requires и Provides.

Модульные программы поиска зависимостей

Среди базовых модулей в пакете `rpm-build` выделим следующие (код был написан в основном Дмитрием Левиным и затем выделен в модули):

- `lib.req` и `lib.prov` — зависимости системных разделяемых библиотек; зависимости имеют вид `libc.so.6(GLIBC_2.0)`.
- `shell.req` — зависимости на исполняемые файлы в Bourne Shell скриптах; здесь не используются виртуальные пакеты, а все Requires-зависимости непосредственно отображаются в названия пакетов (так, использование программы `cat` порождает зависимость на пакет `coreutils`). Для реализации в пакет `bash` был добавлен специальный режим анализа кода (наподобие syntax check).
- `shebang.req` — анализирует первую строчку исполняемых скриптов (которая начинается с символов `#!`) и добавляет зависимость на соответствующий интерпретатор.
- `pam.req` и `pam.prov` — зависимости на модули подсистемы PAM в файлах `/etc/pam.d/*`. Зависимости имеют вид `PAM(pam_userpass.so)`.
- `pkgconfig.req` и `pkgconfig.prov` — отражает зависимости в файлах `/usr/lib/pkgconfig/*.pc`; это в основном зависимости между devel-пакетами, имеют вид `pkgconfig(glib-2.0)`, учитываются также версии.

Среди дополнительных модулей можно выделить следующие:

- `perl.req` и `perl.prov` — поиск зависимостей в скриптах и модулях Perl. Для поиска Requires-зависимостей реализован анализатор внутреннего дерева байткода `B::PerlReq`. Зависимости имеют вид `perl(Getopt/Long.pm)`, учитываются версии модулей. (Пакет `rpm-build-perl`, разработан автором доклада.)
- `python.req` и `python.prov` — поиск зависимостей в скриптах и модулях Python. В реализации использован стандартный модуль `parser`. Зависимости имеют вид `python2.4(getopt)`. (Пакет `rpm-build-python`, автор — Андрей Орлов.)
- `tcl.req` и `tcl.prov` – поиск зависимостей в скриптах и модулях Tcl. Зависимости имеют вид `tcl(Tk)`. (Пакет `rpm-build-tcl`, автор — Сергей Большаков.)

Разработан также поиск зависимостей в байткоде Mono (пакет `rpm-build-mono`, Ильдар Милюков), предложена реализация поиска зависимостей в Java-классах.

Автоматический поиск сборочных зависимостей

Отдельной задачей является автоматический поиск сборочных зависимостей BuildRequires. Программа `buildreq`, написанная Дмитрием Левиным, использует `strace(1)` для трассировки доступа к файлам в процессе сборки пакета. По списку файлов, используемых в процессе сборки, `buildreq` строит список пакетов, необходимых для сборки. Как правило, этот список пакетов является очень большим и поэтому малоинформативен для человека. Этот список поддаётся оптимизации. Например, если при сборке были использованы пакеты `glibc` и `glibc-devel`, то в списке достаточно оставить `glibc-devel`, поскольку этот пакет зависит от `glibc`. Какой-то другой пакет, в свою очередь, может зависеть от `glibc-devel`, что даёт возможность исключить последний.

Для оптимизации списка сборочных зависимостей был разработан новый алгоритм, который обладает следующими свойствами. 1) Оптимизация является сильной, т. е. алгоритм находит почти оптимальное подмножество пакетов, замыкание которого по зависимостям даст исходное множество. Неоптимальность возможна лишь при разрыве циклов. 2) Оптимизация является почти корректной; существуют очень редкие «патологические» случаи двусмысленных виртуальных зависимостей, при которых список может быть оптимизирован некорректно.

Алгоритм работает следующим образом:

1. По списку пакетов на входе выстраиваются пары пакетов с непосредственными зависимостями (`A->B` — пакет A требует пакет B). Помимо непосредственных зависимостей, используется также соединение по Requires и Provides, т. е. учитываются варианты `A->v->B`, где `v` — виртуальный пакет, предоставляемый пакетом B и требуемый пакетом A.

2. Полученное множество пар задаёт частичный порядок, который можно дополнить до линейного, т. е. линейно упорядочить список пакетов по взаимным зависимостям между ними. Для этого используется программа топологической сортировки `tsort(1)`, которая также помогает корректно разорвать циклы (простейшим

циклом является взаимная зависимость между пакетами A->B, B->A).

3. Последний проход алгоритма реализует идею т. н. «решета Эратосфена». Пакеты, расположенные в начале упорядоченного списка, содержат зависимости на другие пакеты, которые расположены в списке ближе к концу. Оптимизация состоит в «вычёркивании» из списка всех «зависимых» пакетов (аналогия с вычёркиванием составных чисел, которые «сводятся» к простым).

Полученный таким образом оптимизированный список BuildRequires добавляется в spec-файл пакета.

Заключение

Отвлекаясь от деталей, хочется подчеркнуть *технологичность* описанных решений. Программы не ошибаются, или, во всяком случае, не «опечатываются». В большинстве случаев это позволяет полностью освободить разработчика от указания зависимостей вручную.

Пётр Козорезов, Роман Макаров, Алексей Федосеев
Москва,
IBM East Europe/Asia

Проект: OpenPower http://openpower.itbu.ru

Проект OpenPower: разработка открытого ПО на платформе POWER

Проект OpenPower — это проект для поддержки разработчиков открытого ПО под платформу POWER путём предоставления удалённого доступа на данной платформе.

Платформа POWER отличается от доминирующей сейчас платформы x86, и, соответственно, существуют некоторые различия при написании программ. Программа, написанная под одну из платформ, не всегда будет работать на другой. Поэтому для популяризации платформы POWER есть смысл дать разработчикам возможность портировать (или писать с нуля) свои программы на данную платформу.

Проект состоит из двух частей. Первая — это веб сайт. Он является лицом проекта. Тут размещена информация о особенностях программирования под платформу POWER, о портировании приложений на эту платформу. Далее, на сайте ведётся учёт пользователей и проектов. Тут же расположен хостинг пользовательских проектов. Хостинг состоит из:

- общей информации по проекту;
- новостей;
- вики;
- обзора исходного кода.

Вторая часть проекта — это система для предоставления shell-доступа пользователям для сборки и тестирования приложений.

Для получения доступа к системе необходимо зарегистрироваться на сервере и отправить заявку на получения доступа к shell-серверу. После рассмотрения заявки администратор либо отклоняет её, либо добавляет пользователя в базу.

Все это реализуется на одном PowerPC-сервере (p520). На сервере делается два LPAR (Виртуальных раздела). Первый для веб-сервера и остальных приложений для обеспечения работы системы. Второй же — для работы пользователей.

Пётр Савельев Санкт-Петербург, ALT Linux

Проект: Connexion `http://radlinux.org:8088/connexion/wiki`

Connexion: +1

Аннотация

Проект Connexion изначально обязан своим появлением net-scripts. Не потому что net-scripts были чрезвычайно удобным инструментом, скорее, наоборот. Техзадание для программы настройки сети было простое. Необходимо было настраивать VLAN'ы, мосты, RPDB[1], а также уметь управлять интерфейсами с любым количеством адресов на них. Тем не менее, всего этого net-scripts на тот момент обеспечить в должном виде не могли. Нужна была замена.

[1] Routing Policy DataBase.

Предыстория

В качестве формата файлов конфигурации был выбран Cisco-like, обязательными требованиями стали возможность перечислять секции в произвольном порядке и возможность хранить конфигурацию в произвольном количестве файлов. Эти требования, как и многие другие, осуществить удалось легко, но, как бы то ни было, оставалась одна сложность. Конфигуратор целиком был написан на GNU Awk, языке простом и мощном, но, увы, на уровне большого проекта доступном лишь немногим. Это и определило судьбу: в изначальном виде дальше специализированных решений этот конфигуратор сети распространения не получил. Переучивать разработчиков на Awk было бы, несомненно, тяжёлым, но бесполезным подвигом.

Наиболее простым решением явилась изоляция уровней проекта в различных не зависящих друг от друга компонентах. Ядро было переписано на Python. Этот язык был выбран из-за простоты разработки, но в текущей реализации язык, на котором написано ядро, уже не важен. Оно практически не накладывает ограничений на внешние модули.

Архитектура и протоколы

Ядро Connexion выполняет функции диспетчеризации сообщений между компонентами системы, журналирования событий и авторизации доступа. Взаимодействие с пользователем осуществляется *интерфейсами*, работающими с с ядром через XML-RPC. Команды оператора, переданные интерфейсом, преобразуются в вызовы *модулей*, которые генерируют shell-код, идущий на исполнение к *сервисам*. Результат работы сервисов отправляется обратно в интерфейс. Для взаимодействия ядра, модулей и сервисов используется реализация стандарта `rfc2822` и MIME (`rfc2045-2047`), что позволяет обрабатывать поток данных shell-скриптами, например, в случае внешних модулей, без потери информации.

Данная схема позволяет, не меняя модулей, направлять весь результирующий shell-код на другой сервис. Например, вместо исполнения можно создать отладочный скрипт. Также от модулей не зависят интерфейсы, работающие с ядром по фиксированному API.

Особенности

Одним из ключевых понятий Connexion является *состояние*. Все модули собраны в дерево, и в каждом узле этого дерева доступно соответствующее подмножество команд. Это позволяет операторам избежать некоторых ошибок, а интерфейсам — более внятно описывать команды, выводя вместо «всех доступных 5 487 вариантов» лишь несколько и снабжая их короткой справкой.

Второе — *отложенные команды*. Модули могут реагировать на команды оператора как немедленно, так и по команде commit. Такой вариант удобен для проведения настройки «на лету», когда оператор может до применения конфигурации ещё раз окинуть её взглядом и, если потребуется, отклонить изменения.

Внешние модули могут быть легко реализованы по мере необходимости и подключены в ходе работы системы, а наличие независимых интерфейсов позволяет авторам модулей не вдаваться в детали представления информации. Ведь интерфейс может оказаться довольно необычным. На данный момент рабочим является cli[2], готовы наброски web- и mail-интерфейсов, в планах стоит snmp.

Евгений Чичкарёв, Тамара Назарко Мариуполь, Украина,
Приазовский государственный технический университет
Мариупольский институт Межрегиональной академии управления персоналом

Оптимизация и статистический анализ: опыт разработки расширений для OpenOffice

Аннотация

Данная работа посвящена процессу разработки расширений для пакета OpenOffice. Представлены структура расширений, взаимодействие их с UNO-объектами самого OO, анализ быстродействия, особенности разработки интерфейса на OOo Basic. Представлены примеры практического использования пакета расширений к OOo.

[2]Command Line Interface.

Данная работа посвящена разработке приложений для решения широкого круга задач оптимизации, нелинейных уравнений и статистического анализа.

В настоящее время широко применяется пакет Microsoft Office, включающий надстройки к процессору электронных таблиц Excel — «Пакет анализа» и «Поиск решения». С их использованием возможно решение целого ряда задач — расчёта параметров линейной регрессии, дисперсионного анализа, расчёта описательной статистики и др., решения задач линейного и нелинейного программирования. Использование этой надстройки возможно только в среде MS Excel, исходные коды скрыты от пользователя.

Для OpenOffice также разработаны подобные средства:

- Optimization Solver (основной разработчик — Kohei Yoshida);
- библиотека макросов OOStat (Sourceforge.net, макросы для OOo).

В рамках данной разработки по сравнению с Optimization Solver расширен набор алгоритмов оптимизации и добавлены алгоритмы, специально предназначенные для решения задач нелинейной регрессии (метод Гаусса—Ньютона и метод Левенберга—Марквардта). По сравнению с библиотекой OOStat переработан графический интерфейс пользователя и добавлены возможности автоматической генерации графических иллюстраций, а также переработан формат представляемой информации.

При планировании разработки данного проекта учитывалась возможность практического использования его результатов для статистического анализа всевозможных научно-технических и экономических данных. Технической предпосылкой для неё послужила вполне прагматическая задача — идентификация математической модели, предназначенной для оптимизации режимов вторичного охлаждения МНЛЗ. Необходимые производственные данные были представлены в виде электронных таблиц, поэтому для решения задачи идентификации был выбран путь построения расширений для изначально офисного пакета. В дальнейшем появилась серия задач экономического характера, также связанная со статистической обработкой значительного массива данных и последующей оптимизацией полученной математической модели.

Основная идея работы с точки зрения разработчика — предложить программное средство, простое в использовании, основанное на свободных средствах и переносимое между платформами Linux, Windows и FreeBSD. Поэтому для реализации проекта был выбран один язык программирования — OOo.Basic. По возможностям языка, объектно-ориентированному характеру, глубокой интеграции с офисным пакетом он близок к VBA (хотя и имеет довольно существенные отличия). Basic для OOo в значительной степени близок к Java. Все компоненты OOo — документы, рабочие книги, листы в них и т. п. рассматриваются в рамках концепции UNO — универсального сетевого объекта, и наиболее существенные отличия от VBA проявляются как раз в объектной модели при организации диалогов и доступе к элементам рабочей книги или листа.

Кроме того, реализован вариант проекта с организацией вычислительного ядра на Python, а графического интерфейса и задания целевой функции оптимизации в электронной таблице OOo.Calc. Такая организация привлекательна в том плане, что позволяет использовать в качестве вычислительного ядра мощный пакет SciPy, а в качестве средства формирования результирующих документов — OOo.Calc и OOo.Writer.

Примеры использования разработки и результаты тестирования представлены в докладе.

Андрей Черепанов Москва

Локализация свободного программного обеспечения

Аннотация

В данном докладе будут рассмотрены основные понятия локализации, реализация механизмов локализации в свободном программном обеспечении, технологии перевода и организационные аспекты локализации.

В настоящее время интерфейс и документация как коммерческих, так и свободных программ изначально пишется на английском языке. С одной стороны, использование единого языка упрощает коммуникации между разработчиками, внедренцами и пользователями программных продуктов. С другой стороны, большинство пользователей не так

хорошо владеют английским, как их родным языком, и они предпочитают программы с интерфейсом и документацией на их родном языке для постоянного использования, поэтому программное обеспечение требует адаптации как к национальному языку, так и к культурным традициям.

Процесс адаптации называется интернационализацией (i18n). Собственно адаптация называется локализацией (l10n). Это перевод ПО и адаптация его к языковым и культурным особенностям определённой аудитории.

Фактически разработчики свободного ПО на сегодня имеют готовую среду для создания ПО, готового к международному использованию. Проблему ввода и хранения помогло решить повсеместное использование Unicode, использование популярного средства интернационализации GNU gettext позволяет отделить процесс разработки от локализации и обеспечивает быструю адаптацию ПО на разные языки.

В мире ПО локализация распространяется на следующие компоненты:

1. Текст программ
2. Размещение виджетов
3. Графика и мультимедиа
4. Комбинации клавиш
5. Шрифты
6. Документация
7. Сопутствующие материалы (шаблоны документов, иллюстрации, примеры, мультимедиа и т. п.

На сегодня самым распространённым средством локализации свободного ПО является GNU gettext. Это программное обеспечение предоставляет разработчикам, переводчикам и даже простым пользователям интегрированные средства и документацию по локализации приложений. Библиотека gettext предоставляет механизмы подключения так называемых каталогов сообщений и получения в программах строк текста, переведённых в соответствии с локалью. Каталоги сообщений — обычные текстовые файлы определённого формата, содержащие оригинальные строки, перевод и служебную информацию. Хочется отметить две уникальных особенности gettext: пометка строк как черновых (перевод есть, но он не используется при работе программы) и поддержка множественных форм (plural forms), что позволяет получать правильное спряжение во фразе в зависимости от указанного числа.

Механизм gettext показал свою жизнеспособность не только для локализации кода, но и оказался подходящим для локализации документации.

Подобно gettext устроен механизм локализации приложений Qt.

Существуют иные механизмы локализации в мире свободного ПО (Mozilla, OpenOffice.org, большинства CMS). Они основаны на определении идентификаторов (а не оригинальных строк) и подстановке вместо идентификаторов переведённого текста. Они не получили широкого распространения в силу их негибкости и неудобства для переводчиков.

Ключевыми моментами в локализации является точность передачи информации, полнота и стандартизация используемых терминов. Существует ряд методологий, которые помогают осуществить качественный перевод: самой важной является «память переводчика».

Для локализации существует множество свободных программ, такие как KBabel, gtranslator и т. п.

При правильной организации локализация может эффективно производиться в рамках совместной работы, что привело к созданию специализированных веб-порталов. Однако эта технология имеет ряд недостатков: ограничения портала и проблемы организации рецензирования.

Для локализации совсем необязательно идеально знать язык оригинала. Качества, незаменимые для профессионального перевода: внимательность и хорошее знание родного языка.

База доступных для переводов текстов в свободном ПО огромна. Поэтому переводчики объединяются в команды, где существует субординация. Каждый принимает на себя столько ответственности, сколько может потянуть. Типичная команда перевода состоит из:

- руководитель команды;
- координаторы пакетов;
- ответственный за работу с новичками;
- рядовые члены.

Команды перевода разных проектов не могут существовать в информационной изоляции друг от друга. Поэтому идёт активный обмен информацией между командами перевода.

Если крупные проекты переведены практически полностью, достаточно качественно и эти переводы поддерживаются в актуальном состоянии, то ситуация с небольшими проектами неважная. Другой проблемой остаётся перевод документации.

В целом, процесс локализации свободных программ как в мире, так и в России развивается и мы будем очень рады увидеть вас в наших рядах. Неважно, хотите ли вы постоянно участвовать в переводе, пришлёте одноразовый перевод или просто сообщили об опечатке — мы рады принять любую помощь.

Сурен Чилингарян
Karlsruhe, Germany,
Forschungszentrum Karlsruhe

Проект: RusXMMS
http://RusXMMS.sf.net

Проект RusXMMS: Прозрачная работа с кодировками

Аннотация

Наверное, каждому русскоязычному пользователю знакомы проблемы с кодировкам: ID3-заголовки, ZIP-архивы, FTP-сервера и ICQ — вот неполный список задач, в которых часто можно увидеть абракадабру вместо понятного текста. Проект RusXMMS призван помочь программисту сделать в его приложениях работу с кодировками прозрачной для пользователей. В настоящий момент в рамках проекта разработаны две библиотеки: библиотека автоматического определения кодировки и библиотека перекодировки. Библиотеки оптимизированы для работы с многоязычными списками, помимо перекодировки предоставляют возможность транслитерации и, в случае наличия широкополосного доступа к интернет, перевода. В докладе будут коротко освещены основные возможности библиотек проекта, а также будет рассказано, как с их помощью можно быстро добавить поддержку перекодировки в существующие приложения.

Несмотря на широкое распространение юникода, разработчики приложений до сих пор вынужденны сталкиваться с большим количеством разнообразных кодировок, используемых пользователями на различных платформах. Проект RusXMMS призван сделать работу с кодировками в многоязычной среде прозрачной для пользователей и программистов. Используя функциональность, предоставляемую библиотекой, возможно с помощью нескольких строчек кода добавить в ваше приложение поддержку автоматического определения и конвертации кодировок, а также, при желании пользователя, транслитерации и перевода.

Итак, текущая версия библиотеки поддерживает:

- Перекодировку заголовков, причём кодировки можно как жёстко задавать, так и использовать автоопределение. Встроенная поддержка автоопределения имеется для русского и украинского языков. Восточноевропейские языки поддерживаются при помощи библиотеки `Enca`. Китайский, Японский и Корейский при помощи — `LibGUESS`.

- Многоязычные списки, а так же авто-определение языков при помощи библиотеки `Aspell`.

- Транслитерацию и перевод при помощи библиотеки `LibTranslate`, которая в свою очередь использует онлайн-переводчики. В частности, для перевода на русский и украинский языки используются `Translate.ru` и `Pereklad.online.ua`.

- Кэш перекодировок на основе базы данных `BerkeleyDB`, что позволяет существенно оптимизировать производительность в случае использования функции перевода.

- Конфигурационный файл, позволяющий пользователям фиксировать определённые кодировки, включать режим перевода, настраивать режим кэширования перекодировок и т. д.

- Для GTK-приложений доступен пользовательский интерфейс. Локализация интерфейса возможна через конфигурационный файл.

Программный интерфейс библиотеки

Поскольку библиотека предназначена для работы в многоязычной среде, то архитектура предусматривает работу не с конкретными кодировками, а с классами кодировок. Так, к примеру, в проигрывателе XMMS эти классы включают кодировки ID3-заголовков версии 1 и 2, кодировку списка проигрывания, кодировку названий файлов в списке проигрывания, кодировку файловой системы и кодировку отображения. Для каждого класса задаётся: имя, тип, способ получения кодировки по умолчанию и флаги. Тип класса и флаги позволяют варьировать функции библиотеки относительно класса в зависимости от его назначения. Так, к примеру, для класса, который используется для представления данных пользователю, можно использовать функцию перевода,

но её использование нежелательно для классов, которые сохраняют информацию для последующего использования. Для класса, задающего кодировку файлов, можно использовать поиск по файловой системе для автоопределения кодировки. Или, если мы хотим сохранять ID3 v.2 заголовки всегда в UTF-16, то можно объявить этот класс константным и запретить его изменения из конфигурационного файла.

В качестве кодировки по умолчанию обычно используется первая сконфигурированная кодировка текущего языка. Однако для некоторых классов можно указать, что по умолчанию должна использоваться указанная юникодная кодировка, кодировка, заданная указанной переменной локали, или кодировка заданного «родительского» класса.

Итак, для того чтобы добавить поддержку библиотек RusXMMS в ваше приложение, для начала требуется описать список классов, который предполагается использовать. Простейший пример:

```
static rcc_class classes[] = {
 {"id3", RCC_CLASS_STANDARD, NULL, NULL, "ID3 Encoding", 0 },
 {"fs", RCC_CLASS_FS, "LC_CTYPE", NULL, "FileSystem Encoding", 0 },
 {"out", RCC_CLASS_TRANSLATE_LOCALE, "LC_CTYPE", NULL, NULL, 0 },
 {"utf8", RCC_CLASS_STANDARD, "UTF-8", NULL, NULL,
RCC_CLASS_FLAG_CONST },
 { NULL }
};
```

Далее требуется инициализировать библиотеку, создать рабочий контекст, инициализировать кэш перекодировок и загрузить конфигурационный файл.

```
rccInit();
ctx = rccCreateContext(NULL, 0, 0, classes, 0);
rccInitDb4(ctx, NULL, 0);
rccLoad(ctx, "xmms");
```

Если предполагается использовать пользовательский интерфейс, его также необходимо инициализировать:

```
uictx = rccUiCreateContext(ctx)
```

В конце работы необходимо сохранить изменения в конфигурации и освободить использованные ресурсы:

```
rccUiFreeContext(uictx);
rccLoad(ctx, "xmms");
rccFree();
```

После того как библиотека инициализирована, можно выполнять перекодировку между классами при помощи команды `rccRecode`. Более сложный сценарий предусматривает использование двух функций: `rccFrom`, которая выполняет перекодировку во внутренний формат библиотеки, содержащий информацию о языке и кодировке, и `rccTo`, которая конвертирует из этого формата в указанный класс. Пример:

```
recoded = rccSizedRecode(ctx, RCC_CLASS_FROM, RCC_CLASS_TO, string,
stringlen)
```

Для того чтобы добавить настройки RusXMMS в пользовательский интерфейс, необходимо воспользоваться функцией `rccUiGetPage`, которая, в случае GTK-интерфейса, возвращает указатель на `GtkVBox`. В обработчике, вызываемом при применении новой конфигурации, необходимо вызвать функцию `rccUiUpdate`, которая изменит текущую конфигурацию в соответствии с текущими пользовательскими настройками. Примеры вызова функций:

```
vbox = (GtkWidget*)rccUiGetPage(uictx, NULL);
rccUiUpdate(uictx);
```

Кирилл Шутемов Минск, Velesys LLC
Проект: Sisyphus, QEMU http://sisyphus.ru,
http://fabrice.bellard.free.fr/qemu/

Порт Alt Linux Sisyphus на ARM

Аннотация

Рассматривается применимость QEMU для Embedded разработок на примере порта Alt Linux Sisyphus под ARM.

Разработка программного обеспечения для встраиваемых систем (embedded systems) имеет ряд особенностей, осложняющих их разработку, тестирование и сопровождение:

- *Ограниченные ресурсы целевой системы*. К встраиваемым системам часто предъявляются особые требования: низкое энергопотребление, малые габариты, отсутствие активных систем охлаждения. Подобные ограничения накладывают отпечаток на производительность системы.

- *Доступность целевой системы*. Часто разработка ПО для встраивемых систем ведётся параллельно с разработкой аппаратной начинки. В распоряжение разработчика ПО, в лучшем случае, попадает ограниченное количество инженерных образцов.

- *Другая архитектура*. По соображениям цены и энергоэффективности на встраиваемых системах чаще используют процессоры, отличные от x86. В настоящее время ARM является доминирующей микропроцессорной архитектурой для портативных цифровых устройств. Также используются архитектуры PowerPC и MIPS.

Можно выделить три подхода к сборке ПО для встраиваемых систем:

- *Нативная сборка*. В этом случае сборка ПО происходит на той же платформе, где и будет выполняться. Эта методика в подавляющем большинстве случаев используется для сборки ПО для «быстрых» платформ (x86, например), однако малопригодна для сборки для медлительных встраиваемых систем. К плюсам данного подхода можно отнести возможность вызова во время сборки произвольных программ целевой платформы. Например, тестов.

- *Кросс-компиляция*. Основана на использовании кросс-компилятора. Кросс-компилятор — компилятор, производящий исполняемый код для платформы, отличной от той, на которой исполняется сам кросс-компилятор. Данный подход позволяет собирать быстро и не требует наличия аппаратного обеспечения, но нет возможности выполнять тесты и другие программы целевой системы.

- *Сборка в эмулированной целевой среде*. Сравнительно новый, слабо распространённый способ сборки. На быстрой машине с

помощью средств виртуализации эмулируется среда целевой системы, в которой и происходит сборка. Этот подход позволяет запускать произвольные программы целевой платформы и не требует наличия аппаратного обеспечения. В этом случае критичным является полнота, корректность и скорость эмуляции.

Для портирования ALT Linux Sisyphus на ARM был использован третий вариант. В качестве эмулятора выбран QEMU. *QEMU* — быстрый эмулятор процессора, основанный на использовании динамической трансляции. Поддерживает 11 архитектур, в том числе и ARM. У QEMU есть два режима работы:

- *Полная эмуляция*. В этом режиме QEMU эмулирует систему полностью, включая процессор и периферию. Этот режим может использоваться для запуска другой ОС без перезагрузки или для отладки системного кода.

- *Эмуляция на уровне пользователя*. В этом режиме QEMU может запускать программы, написанные под одну архитектуру, на другой. На данный момент этот режим доступен только на Linux. В разработке поддержка Darwin.

Эмуляция на уровне пользователя работает существенно быстрее, поскольку через эмулятор исполняется лишь часть кода, находящаяся выше уровня системных вызовов. Этот режим был использован для портирования.

Для создания эмулированной среды QEMU регистрируется в качестве интерпретатора ELF-файлов для ARM через механизм binfmt. Далее статически собранный qemu-arm копируется по указанному при регистрации в binfmt пути внутрь корневой директории ARM. После этого переход в эмулированную среду осуществляется путём вызова команды chroot [1].

Собственно портирование осуществлялось в несколько этапов:

1. *Адаптация QEMU*. Для нормальной работы с полноразмерной корневой директорией была переписана работа с путями в QEMU. Также была реализована и исправлена эмуляция более десятка системных вызовов и приложено несколько сторонних патчей.

2. *Bootstrap*. С использованием корневой директории ARM дистрибутива Debian был пересобран набор пакетов, необходимых для работы hasher.

3. *Замкнутый по сборочным зависимостям репозиторий*. С помощью hasher пересобирается репозиторий до замкнутости по сборочным зависимостям.

В ходе третьего этапа портирования были пересобраны более 250 пакетов, в несколько десятков были внесены изменения. Чаще всего изменения касались отключения тестов, в частности, связанных с многопоточностью. К сожалению, стабильной работы многопоточных приложений при эмуляции на уровне пользователя добиться невозможно из-за нереентерабельности генератора кода QEMU. Исправление ситуации требует переписывания значительного объёма кода. Эту работу начал делать Paul Brook, однако она далека от завершения.

Литература

[1] *Кирилл Шутемов* Chroot into ARM root.
`http://www.freesource.info/wiki/AltLinux/Sisyphus/ChrootIntoArm`

Григорий Баталов Санкт-Петербург, ALT Linux

GPS в России и в Линуксе

Аннотация

Доклад освещает принцип действия GPS-приёмников, типичные примеры использования для навигации, доступные карты России и методы создания собственных карт в ОС Linux.

NAVSTAR GPS[1] — Система Глобального Позиционирования — разработана в 70-х годах Министерством обороны США. В 1983 году принято решение по окончании развёртывания (которое состоялось в

[1] Global Positioning System.

1995 году) предоставить сигнал и гражданским службам. В 2000 году также отключён механизм искусственного загрубления данных («Selective Availability»), что повысило точность определения координат бытовыми приёмниками со 100 до 20 метров. В 2005 году на орбиту выведены модернизированные спутники, вещающие дополнительный открытый сигнал, позволяющий гражданским двухчастотным приёмникам и вовсе сравняться с военными в точности. Систему по сей день обслуживают ВВС США.

Навигационные спутники движутся вокруг Земли по шести орбитам (по 4 спутника в каждой) на высоте 20 180 км. В любой точке планеты, включая приполярные районы, обеспечивается одновременная видимость не менее 6 из них. Приёмник, получив сигнал в микроволновом диапазоне, вычисляет расстояние до спутников в зоне видимости и при помощи несложных уравнений определяет собственные координаты. Для определения широты, долготы и высоты над уровнем моря достаточно данных с трёх спутников. Однако, чтобы минимизировать погрешность приёма, обычно требуется четвёртый. Пятый и последующие спутники ещё более повышают точность расчётов.

Бытовые GPS-приёмники пересчитывают текущие координаты через определённый интервал времени и записывают их в трек (последовательность путевых точек). В приёмниках с графическим дисплеем или в КПК трек обычно накладывается на карту местности, существенно упрощая ориентирование.

К сожалению, полностью открытых электронных карт России в настоящий момент не существует (за исключением *OpenStreetMap* [1], см. ниже). Вероятно, это связано с закрытостью федерального картографо-геодезического фонда для не лицензированных[2] Роскартографией пользователей, а также с распространением «Закона об авторском праве и смежных правах» и на картографическую продукцию.

Тем не менее, популяризация GPS привела к идее изготовления электронных карт путём записи треков и их последующей обработки. Во многих регионах России сложились коллективы энтузиастов, планомерно описывающие близлежащую территорию. Однако они не избегают самовольного использования чужих печатных материалов, что ставит под сомнение правовую чистоту результата их работы. Процесс разработки обычно при этом закрыт, а в сети Интернет выкладывается лишь окончательная версия.

[2]На выполнение топографо-геодезических и картографических работ.

В отличие от них проект *OpenStreetMap* [1] изначально основан на открытой модели. Все присылаемые пользователями треки сразу доступны на сайте проекта, любой желающий может внести изменения в карты и тут же увидеть результат. Для редактирования предложен ряд кросс-платформенных средств. Главный недостаток проекта — незначительное покрытие территории России. Пока что мало сограждан проявляют к нему интерес.

Для ОС Linux существует множество картографических и навигационных программ:

- *gpsd*[2] — демон, принимающий координаты с GPS-приёмника и транслирующий их по TCP.
- *qpeGPS*[3], *gpsdrive*[4] — растровые навигаторы (Qtopia и GTK+).
- *hugo*[5], *RoadNav*[6], *RoadMap*[7] — векторные навигаторы.
- *gpsbabel*[8] — конвертер треков из одних форматов в другие.
- *gpstrans*[9] — интерфейс к приёмникам Garmin для скачивания треков и путевых точек.
- *pygarmin*[10], *perl-GPS*[11] — интерфейсы к приёмникам Garmin на Python и Perl.
- *qgis*[12], *GRASS*[13] — масштабные геоинформационные системы.
- *Maemo mapper*[14] — навигатор для планшетов Nokia.
- *JOSM*[15] — векторный редактор карт для *OpenStreetMap*[1].

Растровые навигаторы позволяют привязать к координатам произвольное изображение, тогда как для векторных, скорее всего, придётся рисовать карты вручную.

Литература

[1] *OpenStreetMap* Открытые карты мира. `http://wiki.openstreetmap.org/`

[2] *gpsd* GPS-демон. `http://gpsd.berlios.de/`

[3] *qpeGPS* Растровый навигатор.
`http://qpegps.sourceforge.net/`

[4] *gpsdrive* Растровый навигатор. `http://www.gpsdrive.de/`

[5] *hugo* Векторный навигатор. `http://hugo.dementia.org/`

[6] *RoadNav* Векторный навигатор.
`http://roadnav.sourceforge.net/`

[7] *RoadMap* Векторный навигатор.
`http://roadmap.digitalomaha.net/`

[8] *gpsbabel* Конвертер треков. `http://www.gpsbabel.org/`

[9] *gpstrans* Интерфейс к приёмникам Garmin.
`http://gpstrans.sourceforge.net/`

[10] *pygarmin* Интерфейс к Garmin на Python.
`http://pygarmin.sourceforge.net/`

[11] *perl-GPS* Интерфейс к Garmin на Perl.
`http://search.cpan.org/dist/perl-GPS/`

[12] *Quantum GIS* Геоинформационная система.
`http://www.qgis.org/`

[13] *GRASS* Геоинформационная система. `http://grass.itc.it/`

[14] *Maemo mapper* Навигатор для планшетов Nokia.
`http://www.gnuite.com/nokia770/maemo-mapper/`

[15] *JOSM* Векторный редактор для *OpenStreetMap*[1].
`http://josm.eigenheimstrasse.de/`

Руслан Хихин Москва, ОАО «Концерн Моринформсистема „Агат“»
Проект: Sisyphus http://sisyphus.ru/

Сизиф как феномен живого дистрибутива, или зависимости в Linux

Аннотация

В докладе рассмотрены виды зависимостей, существующих между различными пакетами дистрибутивов Linux и их виды. Показано, что основная проблема заключается в том, что язык описания зависимостей, используемый в rpm, описывает только один вид зависимостей, хотя существуют три вида зависимостей. В свете сделанных предположений рассмотрены задачи, связанные с ними:

1. задача проверки целостности дистрибутива;
2. задача установки пакета;
3. задача сборки пакета.

Рассмотрены политики разрешения зависимостей и приоритеты в данных задачах. Также рассмотрены проблемы зависимостей и пути их правильного разрешения.

«Сизиф» — удачный пример постоянной среды разработки

Сизиф объединяет интересы мантейнера пакета, пользователя и разработчика дистрибутива.

О видах зависимостей между пакетами

Существуют следующие типы зависимостей по их сфере применения:

1. зависимости построения пакета;
2. зависимости установки пакетов в дистрибутив;
3. зависимости работы пакета.

В общем случае реальные зависимости имеют один из следующих видов (на примере зависимостей работы пакета):

1. «Программы, входящие в пакет, для работы требуют следующие файлы» (минимальная функциональная зависимость).
2. «Для полноценной работы с пакетом желательно, чтобы были установлены пакеты» (максимальная функциональная зависимость).
3. «Для работы с пакетом должен быть установлен один из следующих пакетов» (установочная зависимость).

Существуют также следующие разновидности зависимостей:

- временные зависимости (один пакет сменил другой пакет);
- конфликтные зависимости, пример: «пакет A требует установки или пакета B, или пакета C, но пакет B не может быть установлен одновременно с пакетом C».

Все виды зависимостей можно описать в виде двух вариантов требований: Для пакета A:

1. требуется пакеты `( ...Z)`, причём существуют множества пакетов `(m..n) ... (l...s)`, которые конфликтуют между собой;
2. желательны пакеты `( ...Z)`, причём существуют множества пакетов `(m..n) ... (l...s)`, которые конфликтуют между собой;

Разбиваем эти зависимости на несколько зависимостей типа :

1. требуется один из пакетов `( ...Z)`, причём `(C...Z)` не конфликтуют между собой;
2. требуется один из пакетов `(S ...U)`, причём `(S..U)` конфликтуют между собой:
3. требуется пакет все пакеты `(B...X)`, естественно, конфликтов внутри множества не может быть;

Математическое описание. Введём функции:

`A=>(B...Z)` Пакету A требуются все пакеты из множества `B...Z`.

`A->(B...Z)` Для пакета A желательны все пакеты из множества `B...Z`.

`A=>[B...Z]` Пакету A требуются один пакет из множества `B...Z`, причём `(B...Z)` не конфликтуют между собой;

`A->[B...Z]` Для пакета A желателен один пакет из множества `B...Z`, причём `(B...Z)` не конфликтуют между собой;

`A=>]B...Z[` Пакету A требуются один пакет из множества `B...Z`, причём `(B...Z)` конфликтуют между собой;

`A->]B...Z[` Для пакета A желателен один пакет из множества `B...Z`, причём `(B...Z)` конфликтуют между собой;

`A!B == !(A=>B)` Пакеты A и B конфликтуют;

`&0` нуль множество;

`A=>[B...Z,&0] === A->[B...Z]` Связь между желательно и требуется.

- Функция `A=>(B...Z)` эквивалентна `A=>B ,... A=>Z` — стандартное описание зависимостей в RPM.
- Функция `A->(B...Z)` или вообще не указывается в RPM, или заменяется на `A=>B,...A=>N`.
- Функция `A=>]B...Z[` эквивалентна введению Providers в пакетах `B...Z`, но для разных A могут быть разные `]B...Z[`.
- Функция `A=>[B...Z]` в RPM заменяется или на `A=>(B...Z)` или `A=>N`.

Для примера рассмотрим такую зависимость: `A=>[B,C]`. Раскладываем эту функцию на элементарные зависимости:

1 вариант: `A=>B`; как быть с C? Если C добавляет функциональность к A и нам нет необходимости урезать дистрибутив, то резонно считать, что `A->C`, т. е. Имеем `A=>[B,C] === A=>B && A->C == A =>C && A->B`.

2 вариант: `A=>B`, но при установленном B, ставить C уже бессмысленно `A=>[B,C] === A=>B && !(A=>C )== A=>C && ! (A=>B)` — неверно! В результате приходим к выводу, что на самом деле это зависимость вида `A=>]B,C[ === A=>B && !(A=>C )== A=>C &`

`& ! (A->B)`. Имеем проблему отображения в RPM зависимостей такого рода.

Проблемы

- отображение зависимостей различного вида в RPM;
- проблемы расчёта зависимостей в APT и удаление конфликтных зависимостей;
- предложение — заменить информацию в RPM с учётом существования различных типов зависимостей. И сведению к зависимостям типа: `A=>]B...Z[; A=>[B...Z]; A=>(B...Z)`.

Политики разрешения зависимостей и задачи проверки зависимостей

Можно выделить следующие задачи, в которых зависимости между пакетами играют существенную роль:

- Задача проверки целостности дистрибутива. Проверка того, что в зависимостях пакетов указаны только пакеты, которые реально есть в репозитарии.
- Задача установки пакета. Проверка того, что для установки данного пакета уже установлены все необходимые пакеты, и при необходимости, установка недостающих пакетов.
- Задача сборки пакета. Проверка того, что для сборки данного пакета уже установлены все необходимые пакеты, и при необходимости, установка недостающих пакетов.

При анализе этих задач можно выделить две основные политики:

- *максимальная политика* — проверять только максимальный список пакетов, т. е. если пакету требуется набор следующих пакетов: `(A,B,C) && [D,E]`, то проверять наличие пакетов `{A,B,C,D,E}`;
- *минимальная политика* — проверять только минимально необходимый список пакетов, т. е. если пакету требуется набор следующих пакетов: `(A,B,C) && [D,E]`, то проверять наличие пакетов `{A,B,C,D}`.

Максимальная политика применима к задаче проверки целостности дистрибутива, а минимальная — для задач установки и сборки пакетов.

При рассмотрении задач зависимостей можно сделать вывод, что они сводятся к той или иной разновидности известной в математической комбинаторике «Задаче коммивояжёра».

Предложим вольную аналогию: имеется страна Сизиф, в которой есть куча достопримечательностей и которая с радушием встречает туристов. В каждом городе есть сувениры, которые собирают туристы и отмечают ими посещения данного города. Сувениры объединены в наборы, и туристам важно собрать их полный набор. Часто бывает, что для получения сувенира города A, нужно получить сувениры городов `B,C,D...`

Задача проверки целостности дистрибутива.

Рассмотрим подробнее задачу проверки целостности дистрибутива. Продолжая аналогию. Задача проверки целостности дистрибутива состоит из двух частей:

- *Минимальная проверка* (проверка наличия). Туристический инспектор должен проехать по всем городам, и проверить, что ни в одном городе нет ситуации, когда для получения сувениров этого города надо посетить город, которого не существует на карте.

  Данная задача является наиболее простой — инспектор передвигается по городам в заранее заданном порядке (например, по алфавиту) и собирает список городов, которые требуются посетить туристам в каждом городе. Если требуется посетить город, которого нет на «карте» Сизифа, значит зависимость в этом городе выставлена неверно.

- *Максимальная проверка* (поиск конфликтов). Инспектор помимо проверки списка городов, должен проверить, что ни в одном из городов не возникает ситуация, что для получения сувенира надо посетить два города, которые конфликтуют между собой. Допустим, для получения сувенира «История Греции» требуется посетить Турецкий и Греческий Кипр, но посетив Турецкий Кипр, посетить Греческий Кипр уже невозможно.

Значит, надо выбрать один из следующих вариантов устранения проблемы:

- или отменять сувенир «История Греции» (удалить пакет из Сизифа);
- или помирить обе половины Кипра (убрать конфликт между пакетами);
- или убрать из туристской программы одну из частей Кипра (удалить конфликтующий пакет из дистрибутива) и откорректировать зависимости у пакета «История Греции»;
- или привести зависимость вида `(Турецкий Кипр, Греческий Кипр)` к виду `]Турецкий Кипр, Греческий Кипр[`.

Максимальная задача гораздо сложнее — надо проверить, что в списках зависимостей нет конфликтующих пакетов. Она усложняется тем, что конфликт зависимостей может быть скрытым, например: пакету A требуются пакеты B и C, пакету B требуется пакет D, а пакету C требуется пакет E, но пакеты D и E конфликтуют между собой. В любом случае, для задачи максимальной проверки целостности требуется выявить такого вида конфликты в зависимостях, с тем чтобы устранить такую ситуацию тем или иным способом.

Можно предположить, что перебор пакетов для данной задачи должен быть в «порядке путешествия туриста», но инспектор должен посещать и конфликтующие города (с тем чтобы проверить зависимости и в Турецком и в Греческом Кипре). Т. е. при проверке зависимости `A=>]B,C[` необходимо, проверив город A, посетить для проверки и город B, и город C.

Задача установки пакета

Задачу установки пакета можно интерпретировать как задачу сбора туристом сувениров при путешествии. Турист сам выбирает, какие сувениры ему необходимы, но для получения полного набора сувениров (работоспособной системы) надо получить также связанные сувениры из других городов.

Тут необходимо отметить небольшой нюанс в политиках разрешения зависимостей. Формулу зависимости `[A,B,C,D]` можно

- при максимальной политике трактовать как «посетите города из заданного списка, за исключением конфликтных городов»;

- при минимальной политике — «посетите один из городов из заданного списка».

При установке пакетов надо применять минимальную политику (в отличие от максимальной задачи проверки целостности, но там надо посещать и конфликтующие города).

Максимальную политику установке пакетов можно применять при выборе подмножества пакетов, которую надо положить в создаваемый на основе Сизифа дистрибутив (задачи типа `spt`).

Для задачи установки пакетов важен становится порядок, в котором заданы зависимые пакеты.

Если ввести понятие приоритетов пакетов и в списке зависимых пакетов типов `[A,B,C,D]` и `]A,B,C,D[` пакеты располагать в порядке приоритетов, решается ещё одна задача — минимизация установленных пакетов при заданном пользователем списке пакетов.

Задача сборки пакета

Задачу сборки пакетов можно интерпретировать как задачу изготовления сувениров.

Именно для зависимостей такого рода трудно разрешать циклические зависимости.

Разрешение таких проблем можно найти только найдя разрыв в цепочке, т. е. если для построения пакета C требуется пакет A или пакет D `[A,D]`, то циклическая зависимость разрешается построением пакета D.

На самом деле, циклические зависимости не так страшны, если в дистрибутиве уже есть все три пакета A, B, C предшествующих версий.

Всё становится гораздо проблематичней, если стоит задача перенесения Сизифа на новую архитектуру, и необходимо собрать все пакеты для новой архитектуры. Именно для этого случая важна иерархия зависимостей: нельзя собрать пакет A, пока не собраны пакеты B и C (или D).

Проблемы разрешения зависимостей

В силу ограниченного языка RPM при описании зависимостей пакетов возникают типичные ошибки, которые приводят к неправильному составу в установленном дистрибутиве и в составленном дистрибутиве.

Польза вариантных зависимостей для виртуальных пакетов.

Например: виртуальный пакет KDE требует определённое множество пакетов.

На самом деле для нормальной работы KDE необходимы не все пакеты из этого множества, а только отдельные из этого списка. Если, допустим, в данный момент нельзя установить `kde-icon-theme-nuvola`, то, вообще-то говоря, можно установить пакет `kde-icon-theme-tango` и работоспособность KDE никак не будет нарушена. Но при современном языке описания зависимостей включение в зависимость KDE помимо пакета `kde-icon-theme-nuvola` и пакета `kde-icon-theme-tango` приведёт к тому, что будут установлены оба пакета, что излишне.

Можно выделить основной вид проблемной зависимости:

Пакет имеет зависимость типа `A->[B,C]`, а заменяется на

- `A=>(B,C)` — избыточность устанавливаемых пакетов, пользователь всегда устанавливает пакеты и B и C, хотя реально ему нужен только один из них.
- `A=>B` — устанавливается только пакет B, но если зависимость A=>B входит в циклическую зависимость, то трудно найти разрешение её путём замены на `A=>C` или удалением зависимости `A=>B` (на самом деле, для пакета A совсем не нужен пакет B, он только желателен для создания дополнительной функциональности).
- A не зависит от B и C — при создании дистрибутива в него не войдут ни пакет B, ни пакет C, что обедняет дистрибутив.

Предложения

Три компонента установки:

- rpm как хранитель информации о пакете (зависимости);
- apt как «вычислитель» предполагаемых изменений;
- программа установки как интерактивная установочная среда.