АНО «Институт логики, когнитологии и развития личности»
Базальт СПО

Тринадцатая конференция разработчиков свободных программ

Калуга, 01–02 октября 2016 года

Тезисы докладов

Москва,
Базальт СПО,
2016

УДК 004.91
ББК 32.97

Тринадцатая конференция разработчиков свободных программ: Тезисы докладов / Калуга, 01–02 октября 2016 года. М.: Базальт СПО, 2016. — 80 с. : ил.

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

ISBN 978-5-9908979-0-8

Программа конференции

30 сентября, пятница

15.00- Заезд

1 октября, суббота

9.30 – 9.45 Регистрация участников

Утреннее заседание
9.45–13.30

9.45–10.00 Алексей Новодворский (Базальт СПО). Приветственное слово

10.00–10.20 Савченко А.А., Окунев Д.Ю. (НИЯУ МИФИ)

10.20–10.40 Копытов А.А. (mysql.com)

10.40–11.00 Юрьев Л.В. (Positive Technologies)

11.00–11.20 Кантер Л.Б. (Cloud Linux Inc)

11.20–11.40 Кофе-пауза

11.40–12.00 Пынькин Д.А. (EPAM Systems)

Дневное заседание
14.30–17.00

2 октября, воскресенье

Утреннее заседание
10.00–12.40

Вне программы

Андрей Савченко, Дмитрий Окунев
Москва, НИЯУ МИФИ

Проект: clsync `https://github.com/xaionaro/clsync`

Живая синхронизация данных и иные применения clsync

Живая синхронизация файлов между различными узлами является типичной задачей в кластерных системах. Утилита `clsync` была разработана для решения этой задачи быстро, надёжно и безопасно. В первую очередь работа была ориентирована на системы высокой доступности, для которых требуется:

- высокая производительность (сравнима с производительностью локальной файловой системыпность)
- высокая доступность (отказ в обслуживании не более нескольких секунд)
- высокая надёжность
- универсальность

Существуют разные подходы к репликации и синхронизации данных, в частности:

- единые сетевые файловые хранилища
- блочная репликация
- файловая репликация

Сетевые файловые системы создают единую точку отказа, что в условиях не очень надёжной и перегруженной аппаратной инфраструктуры оказалось не приемлемым решением. Блочная репликация оказалась слишком медленной. Поэтому было использовано решение на основе `lsyncd`, осуществляющего синхронизацию контейнеров между узлами кластера и их резерное копирование. Однако, `lsyncd` оказался слишком требовательным к CPU (существенная часть кода написана на Lua), не пригодным к тонкой настройке и недостаточно надёжным. Поэтому было разработано собственное решение — `clsync`, где мы постарались учесть недостатки `lsyncd` и оптимизировать решение для наших задач.

По сути, `clsync` является демоном мониторинга изменений файловой системы по гибким `regexp`-подобным правилам и выполнения произвольных действий при наступлении любых отслеживаемых событий. Однако, приложение весьма гибко и может быть использовано и в режима работы обычно приложения для выполнения одноразовых операций, например, разовой синхронизации часто меняющихся данных с рекурсивной досинхронизацией изменений, возникших во время предыдущей синхронизации, что оказалось чрезвычайно полезным для задач web-хостинга.

Clsync написан на языке C стандарта C99, поддерживает различные механизмы уведомления о событиях файловой системы как на Linux, так и на BSD. Для Linux механизмом нотификации по-умолчанию является `inotify`, для FreeBSD — kqueue. В качестве бэкенда для синхронизации может использоваться любое приложение, скрипт или связываемая библиотека, но чаще всего применяется `rsync`.

Для обеспечения безопасности предусматриваются опциональные механизмы защиты (некоторые из которых могут ухудшить производительность):

- использование unshare для изоляции пространств имён;
- использование `capabilities` и сброс ненужных привелегий;
- запрет выхода за пределы наблюдаемой директории при помощи `pivot_root`, `chroot` и `unmount`;
- использование `seccomp` фильтрая;
- разделение процесса на привелигированную и непривелигированную части.

Для задач HPC, в частности, для синхронизации обновления программного обеспечения и конфигурации узлов кластера `clsync` оказался чрезвычайно полезен в связке с `pdcp`, а использование so-хендлеров позволило существенно уменьшить задержки операций по сравнению с запуском скриптов.

Алексей Копытов
Черноголовка
Проект: MySQL `http://mysql.com`

Сильные стороны MySQL для высоконагруженных проектов

Аннотация

В настоящий момент многим разработчикам и администраторам часто приходится иметь дело со многими разными СУБД. Знание сильных и слабых сторон каждого продукта становится всё более важным навыком. В мире веб-технологий наблюдается значительный интерес к сравнительному анализу двух наиболее популярных СУБД с открытым исходным кодом: MySQL и PostgreSQL. Сообщество PostgreSQL проявляет достойную уважения активность в освещении сильных сторон PostgreSQL и слабых сторон MySQL. В этом докладе мы поговорим об обратной стороне медали: о том, какие сильные стороны есть у MySQL, какие возможности позволяют этой СУБД обслуживать самые масштабные и высоконагруженные веб-проекты, а также о том, на какие особенности нужно обратить внимание в каждом конкретном проекте в случаях, когда необходимо сделать выбор между MySQL и PostgreSQL.

В последнее время в русскоязычном IT-сообществе активизировалась дискуссия о достоинствах и недостатках MySQL и PostgreSQL — двух самых популярных СУБД с открытым исходным кодом. При этом в силу разных причин информация, предоставляемая в статьях, обзорах и докладах весьма однобока: принято утверждать, что PostgreSQL превосходит MySQL по всем параметрам, а значит нет никакого смысла использовать эту СУБД, особенно когда речь идёт о новых проектах.

С этим можно было бы согласиться, если бы не два факта, которые не укладываются в такую картину:

- MySQL был и остаётся самой популярной СУБД с открытым исходным кодом. При этом, нет никаких тенденций к снижению этой популярности [1].
- MySQL по-прежнему остаётся самой популярной СУБД в интернет-гигантах. Примеров можно привести огромное количество,

вот лишь краткий список наиболее известных компаний и популярных интернет-сервисов: Facebook, Github, Wikipedia, Google, Youtube, LiveJournal, PayPal, Twitter, Booking.com, AirBnB, DropBox, Uber, LinkedIn, Alibaba.

Очевидно, что для этого есть определённые причины, но информацию на эту тему найти довольно трудно. За редкими исключениями (как, например, нашумевшая недавно публикация инженеров компании Uber [2] о причинах перехода с PostgreSQL на MySQL), интернет-гиганты редко деляться результатами внутреннего тестирования и реальных причин использования тех или иных технологий.

Этот доклад — попытка рассмотреть основные причины популярности MySQL, в том числе и в высоконагруженных проектах, с фокусом на функциональность, отсутствующую или реализованную в ограниченном виде в текущих стабильных версиях PostgreSQL.

- реализация репликации в MySQL превосходит PostgreSQL в функциональности и как правило более эффективна в использовании дисковых и сетевых ресурсов. Как показывает опыт таких компаний как Uber и Booking.com, эти преимущества могут стать критичными в масштабных и высоконагруженных проектах;
- InnoDB (основной «движок» хранения данных в MySQL) часто более производителен для многих типичных для веб-проектов операций с данным в силу различных функциональных и архитектурных особенностей: поддержка кластеризованных индексов, компрессии данных, небуферизованного и асинхронного ввода/вывода, изменяемого размера блока данных, а также относительная компактность представления данных — каждая из этих характеристик важна для любого нагруженного веб-проекта. В интернет-гигантах эти характеристики часто становятся критическими с точки зрения как производительности, так и стоимости необходимого оборудования.
- альтернативные «движки» хранения, оптимизированные под специфичные нагрузки и разрабатываемые сообществом MySQL, такие как TokuDB [3] или MyRocks [4], являются важным источником инноваций в MySQL и не имеют аналогов в PostgreSQL. Более того, реализация альтернативных способов хранения данных затруднена отсутствием архитектурной поддержки в PostgreSQL.

- важные в облачных и контейнерных средах функции, такие как прозрачное шифрование данных на диске или переносимые (transportable) табличные пространства обеспечивают популярность MySQL в хостинговых и SaaS компаниях, а также хорошо вписываются в популярную ныне концепцию микросервисов;
- кластерные решения в MySQL (Galera [5], Group Replication [6], NDB [7]) представляют из себя результат многих лет исследований, разработки, тестирования, являются ключевыми технологиями при построении распределённых и отказоустойчивых систем на базе MySQL и не имеют прямых аналогов в PostgreSQL;
- секционирование (partitioning) таблиц часто используется в нагруженных проектах с большими объёмами данных. Реализация этой функциональности в PostgreSQL достаточно специфична и сильно уступает MySQL;
- организация физического резервного копирования в PostgreSQL обладает целым рядом недостатков по сравнению с MySQL;
- поддержка key/value API в MySQL (встроенная memcached или сторонняя HandlerSocket) также часто используется в нагруженных интернет проектах, но не имеет аналогов в PostgreSQL;
- и многие другие, не столь популярные функциональные особенности, которые, тем не менее, могут быть критично важными в некоторых случаях.

Доклад не пытается дать ответ на вопрос «Какая СУБД лучше?», так как универсального ответа на этот вопрос просто не существует. Доклад также не пытается предоставить всестороннее сравнение двух СУБД, т.к. это слишком обширная тема. Целью доклада является восполнение информационного пробела в непростом вопросе выбора СУБД в каждом конкретном проекте.

Литература

[1] DB-Engines Ranking — Trend Popularity `http://db-engines.com/en/ranking_trend/`

[2] Why Uber Engineering Switched from Postgres to MySQL `https://eng.uber.com/mysql-migration/`

[3] Percona TokuDB `https://www.percona.com/software/mysql-database/percona-tokudb`

[4] MyRocks: A space- and write-optimized MySQL database `http://bit.ly/2cRgS8y`

[5] Galera Cluster for MySQL `http://galeracluster.com/`

[6] MySQL Group Replication `http://mysqlhighavailability.com/tag/mysql-group-replication/`

[7] MySQL NDB Cluster `https://www.mysql.com/products/cluster/`

Леонид Юрьев
Москва, Positive Technoglogies, Петер-Сервис

Проект: ReOpenLDAP `https://github.com/ReOpen/ReOpenLDAP`

ReOpenLDAP: сквозь тернии в «МегаФон», путь за год

Аннотация

Проект был инициирован в 2014 году для решения проблем, возникших при эксплуатации исходного OpenLDAP в инфраструктуре одного из крупнейших Российских операторов мобильной связи.

За два года проект выведен «на орбиту» высоконагруженной промышленной эксплуатации. Сейчас ReOpenLDAP успешно работает во всех филиалах ПАО МегаФон по всей России и одновременно доступен как OpenSource.

В очной сессии будет краткий рассказ о том, что удалось сделать за последний год и озвучены планы. В печатных тезисах дополнительно немного предыстории и напоминаний о ранее проделанной работе.

Кратко

Хочу кратко напомнить, как мы пришли к разработке собственно клона общеизвестного OpenLDAP. Затем рассказать, что было сделано в нашем проекте, с акцентом на последний год.

Если говорить о результате, то тесты и опыт эксплуатации позволяют сделать вывод, что наш ReOpenLDAP — один из немногих годных к высоконагруженной промышленной эксплуатации (возможно единственный).

Предыстория

Примерно в 2013 году Петер-Сервис приступил к разработке одной из подсистем (NGDR) для инфраструктуры «МегаФона». Один из вариантов реализации предполагал использование высокопроизводительного LDAP-сервера.

Следует отметить, что сценарий использования, а особенно расчетные нагрузки, совершенно не соответствовали традиционным режимам работы LDAP-серверов. Например, предполагаемые 10 тысяч обновлений в секунду на 1-2 порядка превышают то, что может обеспечить большинство LDAP-серверов.

Среди вариантов, конечно, также рассматривались предназначенные для таких сценариев «чемпионские» key-value хранилища, как например `tarantool.org`. Однако, парадигма LDAP и сам протокол очень любимы в Телекоме. Соответственно использование любого не LDAP-хранилища в скором времени потребовало бы реализации к нему доступа посредством LDAP. Поэтому всё-такие решили попробовать все доступные LDAP-серверы.

В процессе тестов быстро выяснилось, что необходимую производительность даёт только общеизвестный OpenLDAP. При этом крайне важным было то, что нагрузка по записи не приводила к деградации производительности по чтению, и в основном ограничивалась производительностью дисков. Это принципиальное отличие, как и производительность, хорошо объяснялись свойствами движка хранения LMDB.

OpenLDAP также поддерживает синхронизацию содержимого каталога, в том числе в «горячем» режиме, в том числе в топологии multi-master. Массовость использования и 25-летняя история открытого проекта позволяли надеяться, что проблем со стабильностью не будет. По совокупности мы опрометчиво решили закрыть глаза на однократное падение OpenLDAP в процессе наших предварительных тестов. Летом 2014 разрабатываемая подсистема была включена в тестовую эксплуатацию...

Стоит отметить, что изначально Петер-Сервис не собирался «чинить» OpenLDAP своими силами и тем более делать клон проекта. При первых проблемах компания обратилась за коммерческой поддержкой в Symas Corp. Однако Symas Corp не решилась взять обязательства с фиксированием каких-либо сроков и неустойкой за их несоблюдение.

Исходные проблемы

Для понимания контекста следует сказать пару слов о целевой среде: «NGDR представляет собой UDR (User Data Repository), согласно стандарту 3GPP 23.335, и является централизованным узлом для хранения данных обо всех видах услуг абонентов в ИТ-инфраструктуре оператора связи», см `http://bit.ly/2d0103K`

Всего шесть филиалов по России, в каждом:

- Объём данных до 100 Гб, от 10 до 70 миллионов записей.
- Порядка 10 тысяч обновлений в секунду, до 50 тысяч читающих запросов в секунду.
- Сбой при чтении — каскад сбоев в инфраструктуре, неполученная услуга, недовольство абонента и потеря прибыли оператором.
- Сбой при записи — потеря или искажение изменений в наборе услуг и тарифных опциях.

Проблемы были разные и многогранные, большинство из них мы долгое время не замечали. Так, например, сценарий воспроизведения сбоев репликации был найден только в августе 2015, а до этого мы списывали странные расхождения в данных на последствия других проблем.

Почти всегда было не ясно где искать причину. Временами приходилось переделывать все «подозрительные места», а это примерно 80% исходного кода ;)

Устраняли проблемы «послойно», по мере понимания и естественного смещения фокуса от «горим, пожар» до «хорошо-бы поправить».

1. Повреждение БД и падения внутри движка LMDB.
2. Последствия архитектурных изъянов LMDB.
   Например, выполнение «ldapsearch бла-бла-бла | less» приводило к переполнению БД, отказу по записи и последующей деградацией производительности до пересоздания БД.
3. Утечки памяти, отказы в обслуживании из-за OOM.
4. Разнообразные падения, особенно при разрыве соединений под высокой нагрузкой.
5. Распухание базы вплоть до удвоения её физического объема.

6. Проблемы в механизме репликации, от потери отдельных изменений до внезапного обнуления DIR.

7. Шатающиеся тесты, редкие сбои и падения.

Следует отметить, что для абонентов и оператора в целом, эти сбои не стали катастрофой благодаря «военным» регламентам внутри МегаФон, а также четкой работе инженеров службы эксплуатации.

Почему «форк»?

Вынуждено. В конце 2014 мы осознали, что не сможем эффективно устранять проблемы и одновременно возвращать изменения в OpenLDAP.

1. Первым «звоночком» была ситуация с ошибками в движке LMDB: Разделяемый несколькими процессами memory-mapped файл, lockfree алгоритмы и ни одного «volatile» в исходных текстах. При этом убедить авторов в наличие ошибки удалось только показательным тестом с assert-проверкой, а до этого «мы не понимали дизайна».

2. Затем меинтейнеры Symas отказались принимать патчи, мотивировав отказ использованием variadic macro в одном их них. Казус будет не понятен без ряда деталей:

   a) Суммарно наши патчи устраняли почти 5000 предупреждений, большей частью безобидных, но в этом море годами оставались незамеченными реальные проблемы.

   b) Только один из патчей использовал variadic macro, тогда как остальные устраняли явные ошибки и недочеты. При этом grep по исходному коду обнаруживал variadic macro и без наших изменений.

   c) Мы не нашли платформ/систем, на которых можно былобы собрать OpenLDAP и при этом не было компилятора с поддержкой variadic macro, ни пользователей таких систем, ни информации о последних успешных проверках/сборках на таких платформах.

В результате, мы пришли к выводу что нам «нужно ехать, а не шашечки», и в первых числах 2015 года основали ReOpenLDAP. При этом отдельный форк движка LMDB не предполагался. Но осенью

2015 его пришлось сделать по техническим причинам (при подготовке к докладу на Highload++2015), а название «MDBX» было выбрано, как первый подходящий вариант.

Доработки

Рассказывать или даже просто перечислять все исправления «падений» и утечек памяти смысла нет. Поэтому только ключевые/существенные доработки, которые есть только в ReOpenLDAP или в MDBX:

В 2014 починили то, что не давало спать:

- Поддержка LIFO при сборке мусора и повторном использовании страниц в MDBX. Этим мы решили проблему неэффективного использования кэша обратной записи, а также устранили лишнее вымывание кэша страниц ОС.
  Проще говоря, при наличии BBWC производительность позаписи может быть увеличена в разы. Подробно это было рассмотрено в отдельном докладе, см `http://www.highload.ru/2015/abstracts/1831.html`
- Механизм OOM-Handler в MDBX позволил смягчить (фактически нейтрализовать) проблему «зависших читателей» и устранить отказы в обслуживании из-за переполнения БД. Знатокам может показаться, что проблема решена в последних версиях OpenLDAP, так как реализован рестарт «долгих» транзакций чтения, в том числе при ожидании сети. Однако это не так, снижена лишь вероятность возникновения проблем.

В 2015 устранили падения и замеченные системные проблемы:

- В MDBX устранена специфическая проблема возможного разрушения БД при использовании read-write отображения в память. Может показаться странным, но оригинальный движок LMDB при некоторых обстоятельствах может безвозвратно потерять всю базу. Даже если приложение периодически явно выполняет синхронизацию данных с диском.
- Устранено сохранение лишних нормализованных значений LDAP-атрибутов, когда нормализованное и «обычное» значения равны. Эта незаметная и вроде-бы безобидная ошибка на деле приводит к удвоению размера каждой записи и к пропорциональной деградации производительности в лучшем случае.

- Поддержка «limit-concurrent-refresh» что позволяет избежать одинаковых одновременных обновлений на стороне получателя репликации и таким образом существенно ускорить синхронизацию, одновременно с уменьшением пикового потребления памяти.

За последний год

1. Во-первых, починили репликацию:
   a) Устранение логических ошибок в механизме репликации (синхронизации содержимого), которые приводят к потере отдельных изменений:
      i. Упущены несколько проверок при обработке уведомлений.
      ii. Преобразование UUID в DN ключи при репликации может быть некорректным, из-за чего синхронизатор может удалить пересозданную запись, либо перезаписать её последнее обновление старым значением.
   b) Устранение технических ошибок при формировании списка существующих записей (present list), которые приводят к полному или частичному удалению записей DIT:
      i. Может отправляться пустой present list, без попыток его локального формирования, с последующим удалением всех записей на получателе репликации.
      ii. Некорректная обработка ошибок при формировании present list приводит к его «обрезанию» и удалению всех упущенных записей на получателе репликации.
   c) Устранение недочета в RFC-4533 при работе в режиме multi-master с тремя или более участниками. Есть эффект «экранирования удаления».
      Достаточно сложно/запутанно, но суть в том, что легко могут накапливаться записи, которые были удалены на одном сервере, но остались на других и уже никогда не будут удалены механизмом репликации/синхронизации. При этом обнаружить расхождение можно только «построчно» сравнив участвующие в репликации базы.
      Наше текущее решение сводится к дополнительной сверке

данных, и этим не идеально. Зато так обеспечивается совместимость с OpenLDAP и старыми версиями ReOpenLDAP, что крайне актуально при промышленной эксплуатации. Подробности см `http://bit.ly/2d9z53B`

2. Во-вторых, прочие важные переделки:
   a) Проверки с AddressSanitizer, ThreadSanitizer, PVS-Studio, Coverity.
   b) Поддержка TCP-keepalive для входящих соединений к серверу, в том числе к поставщику репликации (syncprov) при обслуживании persistent search запросов.
   c) Реализация «Recursive Upgradeable Read-Write Lock» с устранением ряда проблем в Config Backend.
   d) Много работы по устранению редких сбоев встроенных тестов.
   e) Слияние liblber и libldap, переход на актуальные версии Automake и Autoconf с переделкой сборки. Важно, что при этом существенно выросла готовность (прозрачность) проекта к CMake и другим инструментам.
   f) Импортированные все релевантные патчи от Red Hat, ALT Linux, Debian/Ubuntu.
3. В-третьих, конечно переносились все патчи из оригинальных OpenLDAP и LMDB. Но порядка 20-30% были не актуальны для ReOpenLDAP и MDBX (у нас уже поправлено).

Всего порядка 800 коммитов, из которых на портированные из OpenLDAP приходится менее 1/8.

Ближайшие задачи и планы

1. **Вопросы с лицензией**.
   Цель в переходе на AGPL, как рабочий вариант на двойное лицензирование: исходная лицензия OpenLDAP, плюс AGPL для всех наших доработок и добавлений. Сейчас мы пытаемся договориться с OpenLDAP Foundation, одновременно поняв «как правильно». Советы и помощь приветствуются.
2. **Верификация в Coverity и PVS-Studio**.
   Проверка в Coverity выявила 444 «дефекта», см `https://scan.`

coverity.com/projects/reopen-reopenldap. Выглядит кошмарно, но на момент ответвления дефектов было почти в пять раз больше. Поэтому на самом деле тут колоссальный прогресс. Среди этих «дефектов» основная часть false positives, но их всё равно следует «погасить», а для этого требуется рафинирование многих функций в libreldap.

3. **Пакетирование**.
   К сожалению, эти задачи постоянно вытесняются более «пожарными». Поэтому тянем резину. Пока есть только сторонняя сборка rpm для Feroda/RHEL/Centos.

4. **Устранение редких сбоев тестов**.
   Уже проделана огромная работа, в ходе которой выявлено и поправлено много «плавающих» ошибок. Но некоторые тесты всё-таки иногда сбоят, особенно при параллельном тестировании (высокая, но нерегулярная нагрузка, задержки из-за свопинга). Частично проблема в самих тестах, частично в ещё не выявленных ошибках.

5. **Сравнительные тесты.**
   Опыт, полученный при починке репликации в multi-master режиме, позволяет сделать обоснованное предположение, что в этом режиме из всех LDAP-серверов стабильно работает только наш ReOpenLDAP. Возникает желание проверить это предположение, одновременно оценив итоговую производительность.

Кантер Леонид
Донецк, Cloud Linux Inc.
www.opennebula.org

История внедрения облачной платформы OpenNebula

Аннотация

Критерии выбора продукта OpenNebula в качестве корпоративной облачной платформы для разработки ПО. Опыт развёртывания и эксплуатации.

Предыстория

Наша компания занимается разработкой программного обеспечения для ОС Linux, в том числе и специализированной версии дистрибутива для провайдеров услуги shared hosting. Поэтому для нас очень важно обеспечить комфортные условия для работы наших программистов и тестеров — быструю, надежную и удобную инфраструктуру, которая позволяла бы разработчикам максимально быстро видеть результат их работы в действии, а сотрудникам QA тестировать продукты в условиях, максимально приближенных к тем, которые есть у наших клиентов.

Еще в начале 2014 года количество сотрудников нашей компании не превышало 40, и большая их часть была сконцентрирована в донецком офисе. Там же был небольшой датацентр, в котором размещались сборочный сервер, git и несколько серверов виртуальных машин на базе Parallels Cloud Server 6, которые использовались для разработки и тестирования. Но после известных событий практически все сотрудники были вынуждены перейти на удаленную работу и пришлось часть сервисов перенести на Hetzner. А потом за достаточно короткое время количество сотрудников увеличилось практически до 100 человек и возникла необходимость в построении для внутренних нужд небольшого частного облака.

Поскольку наиболее «раскрученной» платформой для корпоративных облаков является OpenStack, естественно, появилось желание внедрить именно его. Однако сразу возник вопрос — какой именно OpenStack брать? OpenStack — очень сложный проект, состоящий на текущий момент как минимум из 14 компонентов, и кроме свободно распространяемых исходных текстов, существует множество бинарных сборок со своими утилитами для развертывания. Практически, каждый вендор предлагает не просто OpenStack, а свой продукт со своими достоинствами и недостатками. Это и Red Hat OpenStack Platform с поддерживаемой сообществом версией RDO, и Ubuntu, и Mirantis со своим Fuel, не говоря о такой компании, как HP.

Выбор дистрибутива OpenStack

Поскольку мы в основном работаем с системами на базе Red Hat, начали сразу смотреть в сторону дистрибутива RDO. Но оказалось, что его установщик хорошо настраивает только all-in-one сер-

вер, а с кластером из нескольких серверов уже возникают проблемы. Mirantis Fuel Community Edition хорошо умеет разворачивать достаточно сложные конфигурации с использованием файлового хранилища Ceph, отдельными контроллерами и тд, но с точки зрения администратора установленный таким образом кластер представляет собой своего рода «черный ящик» — все изменения конфигурации делаются через веб-интерфейс, а то, что исправлялось вручную, может быть затёрто при, например, добавлении в кластер нового узла. Плюс требуется выделенный сервер в качестве управляющей машины и достаточно сложная конфигурация сети с тремя интерфейсами на каждый узел кластера. В результате было принято решение создать сценарий развертывания кластера OpenStack самостоятельно на базе Ansible, для чего была сформирована команда из четырёх человек. В качестве пакетной базы было принято взять за основу пакеты из RDO, некоторые пакеты пришлось пересобирать с изменениями.

Проблемы, возникшие при эксплуатации OpenStack, и поиск альтарнатиы

После обкатки решения на стенде и развертывания окончательного варианта OpenStack на кластере выбранной конфигурации первые разработчики и тестеры начали получать учетные записи. И если у тех, кто уже имел опыт работы с OpenStack, никаких проблем не возникло, то те сотрудники, которые привыкли работать с Virtuozzo, начали жаловаться на отсутствие полноценных снимков состояния VM (так называемых live snapshots) и невозможность создания VM с реальными адресами на интерфейсе — 1:1 NAT, используемый в OpenStack, не всегда удобен. Кроме этого, администрирование кластера OpenStack довольно трудоемко и затратно из-за его сложности. Поэтому руководство компании поручило рассмотреть альтернативные варианты, которые имеют поддерджку полных снимков состояния VM. Было рассмотрено три продукта — Apache CloudStack, Proxmox и OpenNebula. CloudStack был отклонен из-за сравнимой с OpenStack сложности в развертывании и использовании java/tomcat, что могло бы затруднить оперативное внесение изменений. Proxmox — из-за того, что он поставляется в виде дистрибутива ОС, а не в виде универсальных пакетов для ОС. В результате мы остановились на OpenNebula, которая представляет собой очень простую в установке

и настройке (всего три пакета) и написанную на языке Ruby открытую облачную платформу, предоставляющую всю необходимую нам функциональность.

Преимущества OpenNebula

Одним из важных преимуществ OpenNebula является модель разработки. Разработка координируется из единого центра и ориентируется на нужды конечных пользователей, вто время как OpenStack ориентируется на нужды участвующих в его разработке и конкурирующих между собой вендоров. Второе важное преимущество — другой взгляд на модель облака. Если OpenStack изначально разрабатывался по модели Public Cloud в качестве открытой замены AWS, и лишь потом начал проникать в корпоративный сегмент, то OpenNebula разрабатывалась по модели Enterprise Cloud, глядя на VMware vCloud в качестве образца.

Заключение

В результате успешного внедрения два кластера OpenNebula эксплуатируются без сбоев уже около 5 месяцев, что позволило нам значительно повысить эффективность разработчиков и тестеров, внедрив технологии непрерывной интеграции.

Денис Пынькин
Минск, EPAM Systems

Проект: LinuxContainers.org `https://linuxcontainers.org`

Система управления контейнерами LXD

Аннотация

Обзор современного состояния проекта Linux Containers project: LXC, LXD, LXCFS. Установка и настройка LXD для ALT Linux, с примером создания сервера с помощью mkimage. Использование LXC/LXD в качестве легковесной замены виртуальной машины при разработке ПО.

Введение

Проект Linux Containers (`https://linuxcontainers.org`) уже долгое время занимается развитием набора утилит `LXC`, позволяющего управлять локальными контейнерами. При этом, основное отличие `LXC` от других подобных систем, это создание и управление контейнерами уровня операционной системы [1].

В связи со взрывным ростом использования контейнеров в облачных средах, активно развиваются и системы управления контейнерами, такие как `Docker`, поэтому логичным шагом для проекта Linux Containers было создание своей системы управления `LXD`, которая позволяет унифицировать управление множеством контейнеров `LXC` на разных хостах.

В докладе рассказывается о настройке и использовании `LXD` [2], основываясь на опыте пакетирования проекта для ОС ALT Linux и портирования среды разработки коммерческого проекта в контейнер `LXC`.

Состав проекта

Проект состоит из 3 подпроектов:

- `LXC` — ядро системы, обеспечивающее низкоуровневое управление контейнерами;
- `LXD` — «высокоуровневая» часть системы управления, предоставляющая единообразное API управления, а также утилиты командной строки;
- `lxcfs` — файловая система на базе FUSE, «скрывающая» реальные подсистемы `procfs`, `sysfs` и `cgroupfs`, изолируя доступ к ним от программ, работающих внутри контейнеров.

Установка LXD

Установка `LXD`:

```
apt-get install lxd
```

В результате будут установлены 2 программы для управления `LXC`-контейнерами, написанные на языке Go — `lxd` и `lxc`, демон и клиент соответственно.

Вместе с LXD будут установлены необходимые подпроекты, а также их зависимости:

- lxcfs
- LXC

Требования:

- пакет shadow-submap — subuid и subgid для корректной работы непривилегированных контейнеров;
- criu — библиотека и утилиты для сохранения и восстановления контейнера, позволяющая, в том числе, организовать «живую» миграцию между хостами.

Для понижения привилегий пользователя root используется механизм ядра Linux user namespaces, для чего настраивается список подчиненных идентификаторов пользователя и групп. Эти идентификаторы используются чтобы установить соответствие пользователя внутри контейнера с реальным пользователем.

Так, чтобы пользователь root внутри контейнера имел реальный UID, отличный от '0', необходимо добавить соответствие подчиненных идентификаторов пользователя и групп:

Непривилегированный root

```
usermod -v 100000-165535 -w 100000-165535 root
```

Таким образом «root» внутри контейнера будет иметь UID и GID равными '0', но из-вне контейнера все процессы такого «администратора» будут принадлежать пользователю и группе из заданного диапазона.

Такое соответствие эффективно изолирует пользователя контейнера от возможностей, предоставляемых хостовой операционной системой пользователю root, обеспечивая повышенную защищенность хостовой ОС от контейнера.

Дисковое пространство

LXD поддерживает различные хранилища для контейнеров, начиная от LVM и заканчивая файловыми системами btrfs и ZFS, которые позволяют значительно ускорить создание новых контейнеров, а также сохранение и восстановление снэпшотов систем.

Хотя использование «классических» файловых систем допускается, но и эффективность работы со множеством однотипных контейнеров тоже падает. Для таких случаев предлагается воспользоваться скриптом `lxd-setup-lvm-storage` для создания файла-образа для организации дискового хранилища LVM.

Сеть

Поскольку `LXD` является, по сути, управляющей системой контейнерами `LXC`, то и сеть можно организовать по-разному, начиная от выделения контейнеру физического интерфейса и заканчивая рекомендуемым режимом с организацией виртуального моста (bridge).

Такой мост можно как создать и сконфигурировать самостоятельно, так и положиться на сервис-скрипт `lxd-bridge` — в обоих случаях достаточно добавить несколько строчек в системный файл конфигурации `LXD`: `/etc/sysconfig/lxd-bridge`.

Пользовательская конфигурация

Локальная конфигурация пользователя находится в:

```
~/.config/lxc
```

и включает в себя ключ и сертификат пользователя, а также настройки подключения к удаленным серверам `LXD`.

Системные образы

Одной из основных проблем, которые решает `LXD` является централизованное управление образами контейнеров.

Для старых версий `LXC` созданием контейнера по сути занимались отдельные скрипты-темплейты, которые не отличаются однообразием и пишутся отдельно для каждого дистрибутива.

При проектировании `LXD` было решено уйти от такой практики и использовать хранилище готовых образов (по сути архив rootfs), подготовить которые можно даже вручную.

Такой образ представляет собой готовую к разворачиванию систему, а также описание как самого образа системы, так и действий, которые необходимо произвести при создании или запуске контейнера — модифицировать файл `/etc/hosts`, например.

Типы образов:

- `unified` — все в одном, хорошо подходит для распространения готового продукта;
- `split` — раздельно metadata и rootfs, что очень удобно при разработке.

Конфигурация контейнера

Каждый контейнер использует свою собственную конфигурацию с описанием LXC-контейнера: имя, описание, какие устройства необходимо подключить и какие привилегии есть у контейнера. Кроме того можно использовать и низкоуровневое описание lxc-контейнеров, аналогичное для «классических» контейнеров LXC.

Для однотипных контейнеров можно использовать профили, которые можно указать при создании контейнера, либо добавить позднее, с помощью команды `lxc profile apply ...`. Такой профиль является аналогом «include» для описания контейнера, так что все элементы описанные в профиле включаются в описание самого контейнера, позволяя хранить общие настройки в одном месте.

Неполный пример профиля, выполняющий специфическую настройку для одного проекта, в котором указано, что:

- контейнер будет привилигированным
- с полным доступом к `procfs`, `cgroupfs` и частично ограниченным на запись к `sysfs`
- без автоматически добавляемой системы `devfs`
- разрешением на загрузку модулей ядра `loop` и `tun`
- одним сетевым интерфейсом, подключенным к локальному мосту
- пробрасыванием директории `/lib/modules` в контейнер в R/O режиме (для загрузки модулей от текущего ядра)
- вместо `init` при запуске будет использоваться свой собственный shell-скрипт `/sbin/lxcinit`

```
name: customername
config:
  linux.kernel_modules: loop,tun
  raw.lxc: |
```

```
    lxc.init_cmd = /bin/bash /sbin/lxcinit
    lxc.mount.auto = proc:rw sys:mixed cgroup:rw
    lxc.autodev = 0
    # tun
    lxc.cgroup.devices.allow = c 10:200 rwm
    # loop for images creation
    lxc.cgroup.devices.allow = b 7:* rwm
  security.nesting: "true"
  security.privileged: "true"
description: ""
devices:
  eth0:
  name: eth0
  nictype: bridged
  parent: lxcbr0
  type: nic
modules:
  path: /lib/modules
  readonly: "true"
  source: /lib/modules
  type: disk
```

Пример создания контейнера

Создание профиля из файла описания, приведенного выше:

```
lxc profile create customername
cat profile.yaml | lxd profile edit customername
```

Пример минимального файла-описания `metadata.yaml`:

```
architecture: "x86_64"
creation_date: 1470920887
properties:
  architecture: "x86_64"
  description: "Custom development image"
  os: "fedora"
  release: "1.2.3"
templates:
```

Создание образа для тестирования в формате `split`:

```
tar -czf custom.tar.gz metadata.yaml
tar -C <путь к корню> -czf rootfs.tar.gz .
```

Импортирование образа в `LXD` с присвоением ему уникального имени «`imagename`»:

```
lxc image import custom.tar.gz rootfs.tar.gz --alias imagename
```

Создание и запуск нового контейнера «`name`», созданного из образа «`imagename`» с конфигурацией, указанной в профиле «`customername`»:

```
lxc launch imagename name -p customername
```

LXD в ALT Linux

В дистрибутиве ALT Linux доступны пакеты со всеми необходимыми компонентами для установки `LXD`. Для тестирования работы `LXD` в чистом окружении, а также для образовательных целей [3], был разработан минимальный сервер, работающий, как Live-система.

Благодаря штатному средству создания дистрибутива `mkimage-profiles` [4], создание такого «кастомного» образа свелась к правильному конфигурированию списка пакетов, запускаемых сервисов, а также определению сети для контейнеров с использованием `lxd-bridge`.

Исходный код для сервера можно найти по адресу (бранч `lxd`): `http://git.altlinux.org/people/dans/packages/?p=mkimage-profiles.git;a=tree;h=refs/heads/lxd;hb=lxd`

Литература

[1] *Stéphane Graber* LXC 1.0: Blog post series `https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/`

[2] *Stéphane Graber* LXD 2.0: Blog post series `https://www.stgraber.org/2016/03/11/lxd-2-0-blog-post-series-012/`

[3] *Открытый проект* Linux development courses project `https://github.com/epam-llpd/linux_courses/`

[4] *Michael Shigorin* mkimage-profiles: долгая дорога в 0.9.x `https://lvee.org/en/abstracts/63`

Анатолий Гусев, Дмитрий Костюк
Брест, Брестский государственный технический университет

Решение проблемы интеграции курсора при запуске старинных операционных систем в QEMU

Аннотация

Представлена доработка проекта QEMU, обеспечивающая интеграцию указателя мыши при работе с устаревшими операционными системами. Разработанный патч обеспечивает эмуляцию одного из старейших планшетов Wacom с интерфейсом RS-232 и является единственным известным авторам средством интеграции курсора в гостевые ОС, выпущенные до эпохи USB (при характерном отсутствии драйверов от вендоров виртуальных машин). Рассмотрены особенности реализации и наличие драйверов (в т. ч. свободных) под различные ОС.

Отсутствие интеграции указателя мыши — неизменный спутник запуска старых ОС в эмуляторе. Проблема кроется в существовании устройств с двумя различными типами позиционирования курсора: с относительными координатами (компьютерные мыши) и с абсолютными (планшеты и сенсорные экраны). Устройства с относительными координатами возвращают вектор перемещения курсора, а не его новые координаты, и, в результате, различия в формулах ускорения курсоров хост-системы и гостевой ОС приводят к их рассинхронизации (рис. 1).

Виртуальные машины (ВМ) могут эмулировать устройства с относительным позиционированием (ОП), а иногда и некоторые устройства с абсолютным позиционированием (АП). Для QEMU ОП — это мыши RS-232 и PS/2, а АП — USB-планшет Wacom.

Режим АП свободен от рассогласования, позволяет курсору хост-системы работать с гостевой ОС. Такая интеграция доступна, если:

1. гостевая ОС имеет специальный драйвер, работающий напрямую с ВМ (пример — «виртуальная мышь» Vmware);
2. ВМ умеет эмулировать какое-то устройство АП, поддерживаемое гостевой ОС.

В остальных случаях настольные ВМ включают режим блокирования мыши, когда курсор хост-системы скрывается вплоть до нажатия специальной «освобождающей» комбинации клавиш, а пользователь может взаимодействовать только с гостевой системой. Этот

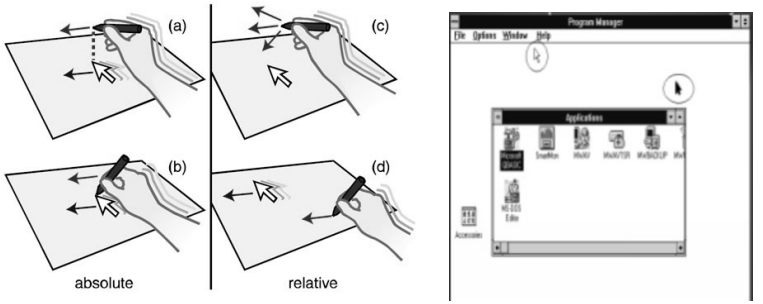


Рис. 1: Несовпадение курсоров, вызванное позиционированием в относительных координатах

режим менее удобен, а в ряде случаев вообще не может использоваться: например, проекты, предоставляющие веб-демонстрацию старых ОС и их программного обеспечения, вынуждены показывать пользователю два курсора, двигающиеся с разной скоростью (рис. 1).

В случае QEMU для режима интеграции мыши по второй схеме гостевая ОС должна иметь драйвер USB-планшета Wacom Pen Partner, который конечно же не может обеспечить интеграцию мыши гостевым ОС, созданным до наступления эпохи USB. Между тем, поддержка старых ОС важнее, чем может показаться. В последнее время появляются веб-проекты демонстрации исторических ОС на «экране», встроенном в HTML. Один из таких проектов — oldweb.today [1], и есть несколько других (отдельные демонстрации ОС от Microsoft, Apple, Amiga). Два курсора мыши, двигающиеся с разной скоростью, существенно затрудняют управление демонстрируемым объектом.

Наш собственный проект, страдающий от той же проблемы — «живой» таймлайн графических ОС настольных и мобильных компьютеров [2], где для запуска старых ОС используется QEMU в роли контейнера, напрямую или с вложенным эмулятором (который, в свою очередь, может работать под старой ОС, не поддерживающей USB).

Одно из решений — виртуальная мышь vmmouse, «придуманная» компанией VMware (ограниченно поддерживается и QEMU). Эта «мышь» плохо подходит для заявленных целей, т. к. требует написания драйвера под каждую гостевую ОС (что сопряжено с изучением больших объемов малоактуальной и очень разнородной документа-

ции). Более того, доступный код драйвера vmmouse использует инструкции процессора i386 и 32-битные адреса портов, что ставит под большое сомнение его реализуемость, например, для DOS.

Безусловно, вариант эмуляции устройства с АП более перспективен. Настоящая работа посвящана как раз такому проекту — реализации режима интеграции мыши для старых ОС путём эмуляции планшета с интерфейсом RS-232, аналогично уже имеющемуся в QEMU бэкэнду serial-mouse с таким же интерфейсом.

Для реализации был выбран планшет Wacom Pen Partner (модель CT0405R, выпускавшаяся в середине 90-х). Причины — интерфейс RS232, совпадение по названию с более новым USB-планшетом, уже поддерживаемым QEMU, возраст, достаточный, чтобы существовали драйвера для большого количества старинных графических ОС, достаточно хорошо описанный протокол Wacom IV [3].

Поддержка CT0405R в ОС до USB выглядит следующим образом. Для DOS, Windows 3.x, Windows 95 и MacOS 9 есть официальные драйвера Wacom, а в остальных системах действуют сторонние. Так, для Pen for OS/2 (надстройка, совместимая с OS/2 версий 2.x и 3.x) поддержка COM-планшетов с протоколом Wacom была реализована Masami Kitamura в виде драйвера, нехитро озаглавленного `W1_011`. Для MacOS X существует свободный драйвер TabletMagic (GPLv2), созданный Scott Lahteine. К сожалению, версия TabletMagic 2.0 потеряла возможность работать в до-USB версиях MacOC X, а версии 1.x более не доступны ни на авторском сайте, ни на sourceforge.net, ни в github; однако их всё ещё удаётся скачать прямым поиском по имени tabletmagicbeta на некоторых зеркалах (например, sourceforge.jp).

Также, учитывая недавние усилия по доработке QEMU m68k для запуска Amiga OS, представляет интерес сторонний драйвер под эту ОС, разработанный в 1999-2000 Denis Spach [4].

Наконец, в начале двухтысячных был создан проект linux-wacom [5]. Драйвера планшетов работают с XFree86 начиная с версии 3.3.1, и предположительно могут использоваться в дистрибутивах, относящихся ко второй половине 90-х. Заметим, что поддержка Wacom Pen Partner с интерфейсом RS-232 была прекращена в X Server 1.7 (в случае Ubuntu это 10.04), однако для решаемой задачи это не принципиально в силу поддержки USB-версии.

Таким образом, выбор планшета для эмуляции выглядел более чем обоснованным. После нескольких неудачных попыток написания кода «на ощупь», был приобретен на ebay реальный экземпляр CT0405R и

сконфигурированы образы ВМ с установленными гостевыми драйерами, действительно, а не в теории, способные с ним работать (FreeDOS, Windows 3.11 Windows 95, Ubuntu Karmic Koala). Исследовать работу планшета с драйверами под MacOS версий 9.2 и 10.2+ на текущий момент не удалось, т.к. в текущей версии qemu-ppc не обнаружилось признаков эмуляции RS-232.

Отслеживание работы с проброшенным в ВМ планшетом показало существенные расхождения в интерпретации протокола IV разными драйверами. Установка связи с планшетом начинается всеми драйверами с пробных переключений скоростных режимов соединения, затем следует запрос модели, версии прошивки и параметров передачи данных. Затем Windows-драйвер дополнительно отсылает команду с числовым кодом, не описанную в протоколе, которую мы идентифицировали как примитивную проверку подлинности (планшет должен ответить семью байтами, пять из которых фиксированные, а два вычисляются на основе переданного кода с помощью исключающего или). После этого драйвера дают команду начать отсылку координат и начинают их приём.

Также Windows-драйвер очень чувствителен к таймингу обмена данными (900 ± 200 кБод), но в остальном он достаточно небрежен: не сбрасывает планшет при тестировании скоростных режимов, не ожидает подтверждения в ответ на команды, не интересуется, каким терминатором заканчивается большинство команд, часто грешит разными окончаниями собственных (`\t`, `\r`, `\n`), и ограничивается минимумом команд в задачах управления курсором.

В противоположность этому драйвер linuxwacom стремится полностью реализовать себя в диалоге и педантично придерживается всей спецификации протокола, вплоть до сверки максимальных координат и т. д.

Наконец, драйвер для DOS — еще более простая версия Windows-драйвера (он не выполняет проверку на подлинность и не «следит» за скоростью.

В итоге, предложенное решение для QEMU снимает проблему несовпадения курсоров в до-USB ОС при наличии установленных драйверов и использовании дополнительно добавленного бэкэнда wctablet. Наработки, синхронизированные с текущей версией QEMU, на момент написания статьи доступны по адресу `https://github.com/avg206/fork-qemu`. Несмотря на сложности с добавле-

нием ретро-устройств в данном проекте[1], авторы ожидают вхождения кода в мэйнстримную версию QEMU (в идеале, 2.8), чему способствовало предварительное согласование и получение данной разработкой гранта по программе Google Summer of Code.

Литература

[1] Chang L. Use oldweb.today to see what the Internet looked like in the '90s. December 3, 2015. http://www.digitaltrends.com/web/oldweb-today-web-archives

[2] The ostimeline project. https://gitssh.com/fiowro/ostimeline/tree/master

[3] Serial Protocol IV. April 3, 2012. http://linuxwacom.sourceforge.net/wiki/index.php/Serial_Protocol_IV

[4] Spach D. PenPartner Wacom driver. February 15, 2001. http://dspach.free.fr/amiga/penpartner

[5] The Linux Wacom Project. http://linuxwacom.sourceforge.net

Анастасия Маркина
Брест, Брестский государственный технический университет

Проект: UXDump

Система параллельного тестирования эффективности человеко-машинного взаимодействия

Аннотация

В работе представлена авторская разработка — свободный проект UXDump, представленный для инструментальной оценки качества человеко-машинного взаимодействия с программными продуктами. Работа системы основана на мониторинге физического состояния

[1] У QEMU богатый опыт отвергнутых патчей для поддержки старинных устройств ввода: например, мышь Mouse Systems (http://lists.gnu.org/archive/html/qemu-devel/2014-01/msg01752.html) или NeXT Computer bus mouse (http://gunkies.org/wiki/Installing_NeXTSTEP_on_Qemu).

пользователя с помощью биометрических датчиков популярных гаджетов (фитнес-трекеров и др.). Рассматриваются архитектура, возможности и способы применения системы, разновидности гаджетов и типовые схемы их подключения.

Измерение параметров физического состояния пользователей (частота пульса, электропроводность кожи и др.) при работе с программным обеспечением — это подход, который позволяет быстро и эффективно определить «узкие места» интерфейса для доработки приложения, а также бывает полезен при выборе одного из нескольких конкурирующих программных продуктов. Исторически он разрабатывался менее активно из-за необходимости в редких биометрических устройствах; однако в последнее время требуемые датчики появились в достаточно большом числе популярных гаджетов, таких как спортивные пульсометры, фитнес-трекеры, «умные часы», развлекательные энцефалографы. Несмотря на то, что эти устройства ориентированы на индустрию спорта и развлечений, они способны передавать данные хост-системе и обладают достаточной точностью, для возможности их использования в usability-тестировании.

Ранее в ряде публикаций нами представлялись результаты тестирования, выполненные в рамках такого подхода [1, 2]. В частности, этот метод хорошо проявил себя при сравнении интерфейсов, перегруженных визуальными элементами, поскольку влияние неоптимальных композиционных решений в данном случае наиболее сильно сказывается на эффективности работы оператора [3, 4]. В результате проведения нескольких исследовательских проектов, построенных на данной методике, была разработана программная система UXDump, упрощающая приборную оценку эффективности человеко-машинного взаимодействия. Код данной разработки[1] распространяется под лицензий GPL v.2.

Основная часть системы написана на C++ и Qt (опробовалась для двух тестируемых платформ, GNU/Linux и MS Windows). Система ориентирована на одновременное параллельное тестирование ПО на нескольких компьютерах; поэтому для хранения результатов измерений была выбран MySQL, с которым разработанное приложение взаимодействуют по схеме клиент-сервер.

Проект UXDump построен по модульному принципу и включает в себя блоки, обеспечивающие добавление информации о проведен-

[1] `https://bitbucket.org/AsyaAliset/uxdump`

ных тестах, визуализацию данных экспериментов, администрирование базы данных, обеспечение сетевого взаимодействия (рис. 1). На компьютер, участвующий в исследовании, должна быть установлена либо полная копия клиентской части (т.е. вся система UXDump), либо, как минимум, модуль Launcher, позволяющий выбирать имеющиеся измерительные устройства и запускать выбранные тесты.

Архитектура системы показана на рис. 1 (передача управления) и рис. 2 (передача данных). Основной стартовый модуль — оболочка UXDump suite, которая передаёт управление модулям DataBase manager и Launcher. Launcher предназначен для запуска измерительных модулей (либо модулей-обёрток, через которые происходит передача данных от измерительных устройств) и выбранного пользователем тестового скрипта. Тестовый скрипт запускает программное обеспечение, участвующее во взаимодействии с пользователем (т.е. начинает эксперимент), и ожидает завершения его работы. По окончании теста управление возвращается к Launcher, который принудительно завершает измерительные модули, собирает созданные ими логи и передает управление библиотеке UXDump framework для добавления данных в базу. Эта же библиотека используется для получения данных из БД, например при визуализации результатов выбранных экспериментов модулем UXDump suite.

Измерительные модули принимают данные от надетых на пользователя устройств. В настоящее время система включает модуль FitBit для измерения частоты сердечных сокращений с помощью фитнес-трекеров Fitbit, модуль mindwave для оценки концентрации внимания с помощью энцефалографа NeuroSky Mindwave и модуль uxdumpbox, принимающий данные о пульсе и электропроводности кожи с аппаратного блока авторской разработки[2].

Измерительный модуль записывает в лог результаты измерений в максимально доступном объеме. Если это приводит к неоправданному превышению количества хранимой информации, то может использоваться модуль-обёртка, выполняющий фильтрацию снятых биометрических показаний и/или их первичную обработку. Пример последнего — модуль-обёртка phasic GSR, который анализирует «сырые» данные электропроводности кожи, фильтрует их и подсчитывает пики тонической кожно-гальванической реакции, соответствующие повышенному возбуждению оператора [4].

[2] `https://github.com/fiowro/uxdump`

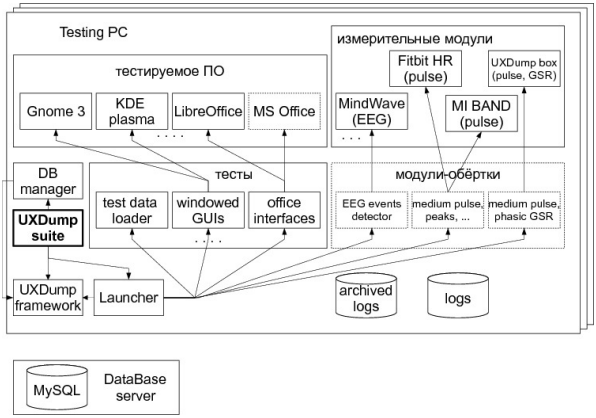


Рис. 1: Архитектура системы: передача управления

Как уже упоминалось, измерительные модули возвращают данные в виде текстовых логов (что заметно упрощает создание сторонних модулей). Такие же логи с данными могут генерировать и некоторые тестовые скрипты (например, показания о корректности выполнения заданий пользователем). После возвращения управления к модулю Launcher происходит анализ содержимого логов, подготовка информации и её запись в базу, а также передача исходных лог-файлов в архивное хранилище. Аналогичным образом информация может быть добавлена в базу модулем DataBase manager при импорте ранее сохранённых лог-файлов. Помимо результатов измерений база хранит данные о пользователях и условиях проведения эксперимента (заполняются перед началом очередного теста).

Обычно, при написании нового измерительного модуля, удаётся задействовать уже существующий свободный проект, предназначенный для извлечения данных из соответствующего устройства. Например, для mindwave — это библиотека mindwave-python, а для fitbit — проект galileo. Использование оригинального API от разработчика устройства, по нашему опыту, наименее вероятный вариант: из-за другого целевого назначения гаджета штатные средства обычно не позволяют передавать биометрические данные с нужной частотой и в необходимый момент времени. При этом некоторые производите-

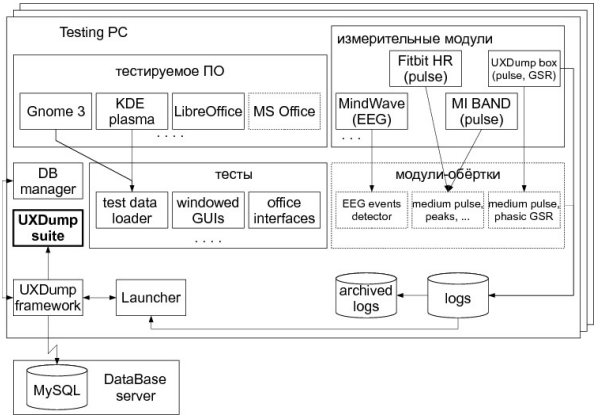


Рис. 2: Архитектура системы: передача данных

ли используют стойкую криптографическую защиту результатов измерений, и, в этом случае, необходима смешанная схема получения данных, когда стороннее ПО обеспечивает своевременное извлечение результатов и их передачу на сервер, а измерительный модуль, дождавшись завершения этой процедуры, запрашивает расшифрованные данные с сервера (рис. 3).

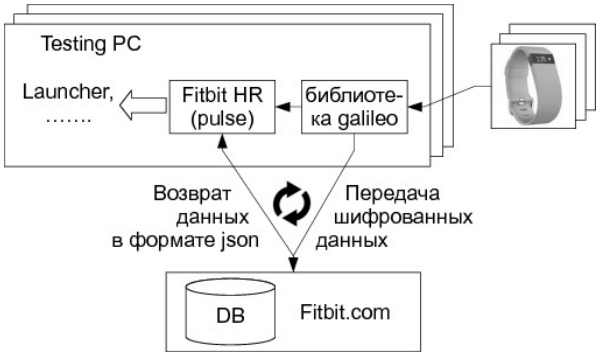


Рис. 3: Алгоритм взаимодействия с фитнесс-трекером

Автор выражает благодарность Стасу Фомину за экземпляры предоставленного оборудования.

Литература

[1] Журавский В.В., Костюк Д.А., Латий О.О., Маркина А.А. Программно-аппаратная система для сравнительных исследований эргономики обеспечения // Информационные технологии и системы 2015 (ИТС 2015): материалы международной научной конференции. Минск, БГУИР, 29 октября 2015 г. — С. 252–253

[2] Kostiuk D.A., Latiy O.O., Markina A.A. Software system for parallel usability testing // Шоста науково-практична конференція FOSS Lviv 2016: Збірник наукових праць. — Львів, 19-2 квітня 2016 р. — С.59–62

[3] Костюк Д.А., Латий О.О., Маркина А.А. Инструментальная оценка состояния пользователя в задаче сравнения интерфейсов офисных приложений // XII конференция разработчиков свободных программ. Тезисы докладов. — Калуга, 16-18 октября 2015 г. — М.: Альт Линукс, 2015. — С.8–12

[4] Костюк Д.А., Латий О.О., Маркина А.А. Об эффективности использования метафоры ленточного интерфейса // Одиннадцатая конференция «Свободное программное обеспечение в высшей школе»: Материалы конференции. — Переславль, 30-31 января 2016 г. — М.: Альт Линукс, 2016. — С.17–23

Сергей Александров, Константин Калмыков
Москва, ООО «НТЦ ИТ РОСА»

Работа с сертифицированными криптопровайдерами в отечественных дистрибутивах Linux

Аннотация

Перспективы работы с отечественными СКЗИ в российских дистрибутивах Linux. Предлагаемые способы решения прикладных задач для обеспечения ЭЦП и шифрования документов или файлов.

Dmitry V. Levin
Москва, Базальт СПО

Проект: strace `https://strace.io`

Can strace make you fail?

Аннотация

strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. This year strace has been extended to tamper with tracees using controllable syscall fault injection.

Введение

strace как инструмент мониторинга взаимодействия пользовательских процессов с ядром существует уже почти 25 лет и широко применяется для диагностики, отладки и изучения поведения ПО. Многочисленные параметры управления фильтрацией дают возможность пользователю strace легко и гибко настраивать отображение системных вызовов и сигналов. С каждым выпуском strace таких возможностей становится больше, а точность отображения – выше.

В этом году Nahim El Atmani, студент лаборатории LSE в École pour l'informatique et les techniques avancées (EPITA), в рамках одного из проектов GSoC 2016 [1] реализовал прототип новой функции [2] в strace, которая принципиально отличается от всех предыдущих и открывает новые направления применения strace.

fault injection как метод тестирования

Как тестировать обработку программой нетривиально воспроизводимых ситуаций вообще, и особенно ошибок, которые не происходят во время тестирования? Например, можно искусственным образом создавать у программы представление о том, что тестируемое условие произошло. Но как это сделать, не внося изменений в тестируемую программу? Можно попробовать изменять поведение системных вызовов, например, с помощью strace!

strace и системные вызовы

strace отслеживает системные вызовы, выполняемые подопечными процессами, с помощью системных вызовов ptrace и wait4. ptrace-запросами PTRACE_SYSCALL strace поручает ядру останавливать трассируемые процессы при выполнении ими системных вызовов. Эти процессы на входе в любой системный вызов переводятся ядром в состояние syscall-enter-stop, а на выходе из любого системного вызова – в состояние syscall-exit-stop. Об изменении состояния процессов ядро оповещает strace, ожидающий событий в wait4.

strace всякий раз, обнаруживая трассируемый процесс в состоянии syscall-enter-stop, выясняет номер и параметры системного вызова, после чего применяет фильтры, которые на основе этой информации формируют решение о том, отображать ли этот системный вызов. Если системный вызов подлежит отображению, выполняется соответствующий парсер, который согласно настройкам и семантике этого системного вызова отображает его надлежащим образом. Обработав состояние syscall-enter-stop, strace дает ядру запрос PTRACE_SYSCALL, после которого ядро продолжает выполнять приостановленный системный вызов до состояния syscall-exit-stop. Обнаружив трассируемый процесс в этом состоянии, strace при необходимости выясняет и отображает код возврата и, возможно, другие результаты работы системного вызова. Обработав состояние syscall-exit-stop, strace снова дает ядру запрос PTRACE_SYSCALL, после которого трассируемый процесс продолжает работу до следующего системного вызова.

fault injection системных вызовов

ptrace API позволяет не только считывать номер, параметры, и код возврата системного вызова, но и менять их. Например, путем замены номера системного вызова в состоянии syscall-enter-stop на -1 происходит замена системного вызова на заведомо несуществующий, а замена кода возврата системного вызова в состоянии syscall-exit-stop приводит к установке произвольного кода возврата.

Реализованный в strace интерфейс позволяет осуществлять fault injection произвольного множества системных вызовов, как всех подряд, так и выборочно, например, только N-й вызов системного вызова, каждый N-й вызов, и т.п. В сочетании с традиционным фильтром

по имени файла можно осуществлять fault injection системных вызовов, прямо или косвенно оперирующих определенными файлами.

Первые находки

Экспериментальные запуски strace в режиме syscall fault injection сразу выявили проблемы с обработкой ошибок в разных программах. Например, python3 не обрабатывает ошибку доступа к устройству /dev/urandom:

```
$ strace -P /dev/urandom -e fault=open:1:ENOENT python3
open("/dev/urandom", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory) (INJECTED)
Fatal Python error: Failed to open /dev/urandom
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x50} ---
+++ killed by SIGSEGV +++
Segmentation fault
```

Еще один пример показывает, что динамический загрузчик не всегда проверяет код возврата системного вызова mprotect:

```
$ strace -e mprotect -e fault=mprotect:1:EPERM pwd > /dev/null
mprotect(0x7fabcd00f000, 2097152, PROT_NONE) = -1 EPERM (Operation not permitted) (INJECTED)
mprotect(0x7fabcd20f000, 16384, PROT_READ) = 0
mprotect(0x606000, 4096, PROT_READ)     = 0
mprotect(0x7fabcd441000, 4096, PROT_READ) = 0
+++ exited with 0 +++
$ strace -e mprotect -e fault=mprotect:2:EPERM pwd > /dev/null
mprotect(0x7fabcd00f000, 2097152, PROT_NONE) = 0
mprotect(0x7fabcd20f000, 16384, PROT_READ) = -1 EPERM (Operation not permitted) (INJECTED)
pwd: error while loading shared libraries: /lib64/libc.so.6: cannot apply additional memory
protection after relocation: Operation not permitted
+++ exited with 127 +++
```

Что дальше?

Продолжением и расширением функции syscall fault injection могла бы стать функция syscall success injection. Тот же прием, который был использован для реализации fault injection, годится и для создания видимости успешного завершения системного вызова. Основная сложность в реализации success injection – сохранение семантики системных вызовов, результатами работы которых является не только код возврата, но и, например, запись определенных адресов оперативной памяти.

Область применения strace syscall fault injection – не только расширение тестового покрытия и поиск ошибок. Например, с помощью этой функции непривилегированный пользователь может легко организовать временное блокирование отдельных системных вызовов и обращений к отдельным файлам у определенных процессов, не прибегая к написанию seccomp filters.

Литература

[1] Google Summer of Code: strace.
https://summerofcode.withgoogle.com/organizations/5106770607341568/

[2] strace syscall fault injection.
https://brokenpi.pe/tools/strace-fault-injection

Эльвира Хабирова, Дмитрий Левин
Москва, ВМК МГУ им. М. В. Ломоносова

Проект: strace http://strace.io/

Действительно структурированный вывод в strace

Аннотация

strace — утилита для отладки программ. Она отображает сделанные отлаживаемым процессом (tracee) системные вызовы, пришедшие ему сигналы, изменения его состояния и пр. Вывод *strace* на данный момент нацелен на человекочитаемость и по этой причине тяжело поддается автоматической обработке. Кроме того, из-за отсутствия единой системы вывода в выводе могут присутствовать неточности, что еще сильнее усложняет задачу. Поэтому в рамках GSoC 2016 проблема автоматической обработки была решена разработкой такой единой системы. Как результат, стало возможным легко встраивать не только подсистемы любого формата вывода, но и дополнительные слои логики.

Обзор проблемы

В *strace* на входе и выходе из системного вызова вызывается ассоциированный с ним обработчик (декодер), которому передаются полученные из tracee аргументы. Ранее, *strace* кроме того, что обрабатывал эти аргументы, еще и немедленно печатал их. На входе могла печататься только часть аргументов (до первого out[1] аргумента). В

[1] out аргумент — аргумент, который указывает на участок памяти, в котором находятся данные, являющиеся выходными для вызываемой функции. Также существует понятие in-out аргумента: такой аргумент указывает на место в памяти с данными, являющимися входными и выходными

случае, если это имеет смысл и если указан достаточный уровень подробности вывода, *strace* копирует из tracee данные, находящиеся по указателям, переданным в аргументах, и печатает их. Этот процесс может быть весьма многоуровневым — в случае передачи массивов указателей на структуры, например. Дополнительно *strace* может собирать и печатать статистику по времени выполнения, печатать содержимое iovec и проч.

strace не содержал унифицированных средств по печати информации о системном вызове; каждый декодер самостоятельно обеспечивал тот формат вывода, который считал нужным (за исключением некоторых базовых вещей, таких как вывод строки, флагов, массивов, дескрипторов... Хотя и здесь есть исключения в случае отдельных декодеров, которым требуется нестандартный вывод).

Далее рассмотрим несколько примеров существующих декодеров системных вызовов.

```
accept4(3, {sa_family=AF_UNIX, sun_path="accept4.socket.connect"}, [110->25],
SOCK_NONBLOCK|SOCK_CLOEXEC) = 5
```

Рис. 1: Пример отображения информации о системном вызове *accept4*

Рассмотрим иллюстрацию 1. *accept4* принимает в качестве второго аргумента указатель на структуру *struct sockaddr*, в качестве третьего — размер структуры аргумента, и в качестве четвертого набор фагов. В данном случае *strace* скопировал из tracee структуру *struct sockaddr* и декодировал её согласно значению поля *sa_family* (т. е. интерпретировал последующие данные как поле *sun_path*); также *strace* прочитал значение по указателю, переданному в аргументе *addrlen*, причём и на входе, и на выходе, так как это in-out аргумент. Четвёртый аргумент декодирован как набор флагов.

```
setitimer(ITIMER_REAL, {it_interval={0, 222222}, it_value={0, 111111}}, NULL) = 0
```

Рис. 2: Пример отображения информации о системном вызове *setitimer*

Перейдем к иллюстрации 2. В качестве второго и третьего аргументов *setitimer* принимает указатель на *struct itimerval*, который, в свою очередь, являет собой пару структур *struct timeval*. Заметим, что декодер системного вызова выводит имена полей *struct itimerval*, но не *struct timeval*.

На иллюстрации 3 можно отметить сразу несколько особенностей вывода:

```
$ strace -etrace=pwritev -ewrite=1 -s2 ./preadv-pwritev
...
pwritev(1, [{iov_base="01"..., iov_len=3}, {iov_base="34"..., iov_len=5}, ...],
3, 0) = 15
 * 3 bytes in buffer 0
 | 00000  30 31 32                                          012             |
 * 5 bytes in buffer 1
 | 00000  33 34 35 36 37                                    34567           |
 * 7 bytes in buffer 2
 | 00000  38 39 61 62 63 64 65                              89abcde         |
```

Рис. 3: Пример отображения информации о системном вызове *pwritev* с печатью содержимого iovec

– Строки и массивы, если их размер превышает указанный в параметре `-s` (32 по умолчанию), сокращаются и терминируются многоточиями.

– Для I/O вызовов существует возможность, посредством задания опции `-ewrite=1`/`-eread=1`, вывода содержимого данных, передаваемых в этих вызовах, в формате шестнадцатеричного дампа.

```
execve("./execve", ["./execve"], [/* 41 vars */]) = 0
```

Рис. 4: Пример отображения информации о системном вызове *execve*

На иллюстрации 4 заметим нестандартный способ аббревиирования вывода массива переменных, который отличается от используемого в большинстве других системных вызовов многоточия.

```
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, {len=3,
filter=[BPF_STMT(BPF_LD|BPF_W|BPF_ABS, 0),
BPF_JUMP(BPF_JMP|BPF_K|BPF_JEQ, 0x3, 0, 0x1),
BPF_STMT(BPF_RET|BPF_K, SECCOMP_RET_ALLOW)]}) = 0
```

Рис. 5: Пример отображения информации о системном вызове *prctl* с *PR_SET_SECCOMP* в качестве аргумента *option*

На иллюстрации 5 показан пример системного вызова *prctl*. То, какие из аргументов *arg2*, *arg3*, *arg4*, *arg5* выведет *strace* и в каком формате, определяется значением первого аргумента *option*. В данном случае это команда установки BPF-фильтра *seccomp*. Другой особенностью данного примера является формат вывода команд BPF, которые являют собой структуры, но для которых в `<linux/filter.h>` определены макрокоманды для их определения в коде на языке Си; декодер системного вызова определяет по содержимому структур под-

ходяющую макрокоманду и печатает соответствующую макрокоманду.

```
mincore(0x2ba1f6293000, 131073, [11111111111111111111...]) = 0
```

Рис. 6: Пример отображения информации о системном вызове *mincore*

```
listen(3<TCP:[4490622]>, 1) = 0
```

Рис. 7: Пример отображения информации о системном вызове *listen*

```
rt_sigprocmask(SIG_SETMASK, NULL, [HUP INT QUIT ALRM TERM], 8) = 0
```

Рис. 8: Пример отображения информации о системном вызове *rt_setsigprocmask*

На иллюстрации 6 отметим формат вывода третьего аргумента, *vec*, который представляет собой массив байт, из каждого из которых значащий только один бит. На иллюстрации 7 интересен формат вывода, используемый для файловых дескрипторов (первый аргумент). В примере на рисунке 8, можно отметить специфический формат, используемый для вывода маски сигналов.

Идея структурированного вывода

Итак, количество техник вывода, применяемых в *strace*, с одной стороны, велико и разнообразно (вывод битовых масок, масок сигналов, различные варианты аббревиирования структур), с другой, всё же поддаётся классификации (есть некие общие правила вывода структур, указателей, массивов). В основе структурированного вывода лежит идея, что все особенности формата вывода можно оформить в виде reusable примитивов и использовать для вывода только их. Пример преобразования декодера показан на иллюстрации 9.

Второй важной особенностью является введение промежуточного представления системного вызова (см. рис. 10). Декодер заполняет это внутреннее представление, и оно может потом быть выведено независимо.

Переход к структурированному выводу позволяет избежать многочисленных однотипных вызовов печати (запятые, скобки, знаки равенства) и связанных с этим потенциальных багов.

Разделение процесса декодирования и вывода позволяет внедрить возможности, которые ранее были затруднительны в реализации. Например:

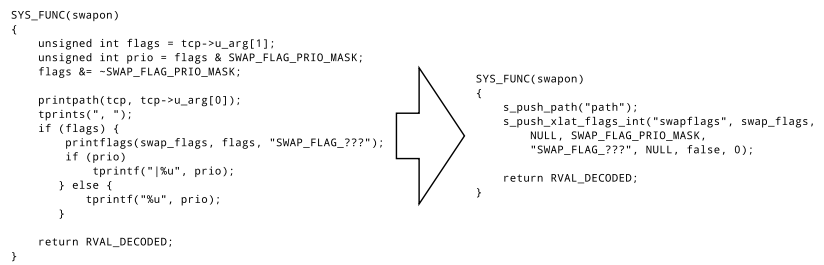


Рис. 9: Преобразование декодера системного вызова на примере *swapon*

– Наконец-то можно корректно реализовать *секретную* опцию `-z`, которая выводит только успешно завершившиеся системные вызовы. Ранее было неясно, как обходить случаи, когда системный вызов не мог быть полностью декодирован на входе, а также когда трассируемых процессов несколько.

– Изначально при реализации структурированного вывода было принято решение сохранять имена аргументов; ранее такой информации декодеры не предоставляли. Это позволило реализовать опцию `-N`, которая позволяет выводить имена аргументов для некоторых[2] или всех системных вызовов. В существовашей схеме вывода для добавления подобной функциональности потребовалось бы существенное усложнение логики всех декодеров и маловероятно, что в исходной схеме вывода такая возможность сама по себе была бы реализована.

– Также добавлена возможность привязки комментариев к аргументам, которая также может быть полезна для системных вызовов, вывод аргументов которых зависит от значний аргументов.

– Существенно упростилась поддержка out и in-out аргументов. Теперь декодер системного вызова сохраняет всю доступную информацию на входе и обновляет её на выходе, что гораздо более однообразно, чем использовавшаяся ранее сложная логика. Раньше аргументы системного вызова на входе декодировались

[2] В первую очередь это актуально для тех системных вызовов, в которых семантика одних аргументов зависит от значения других; см. *ipc*, *fcntl*, *prctl*, *keyctl*, *futex*, *quotactl*.

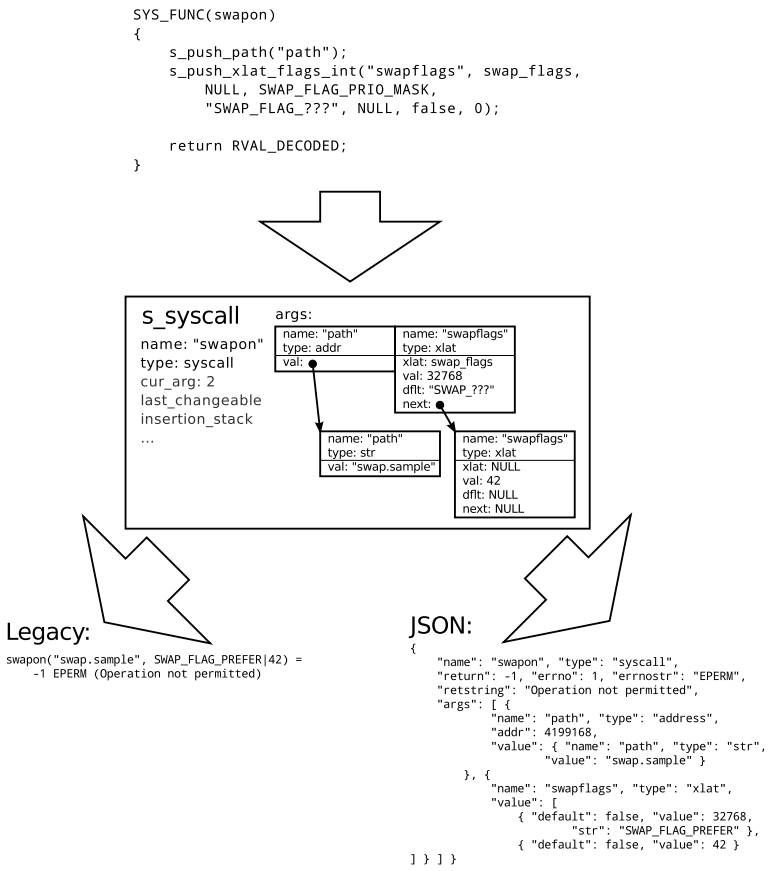


Рис. 10: Схема работы структурированного вывода

только до первого out аргумента; иногда приходилось сохранять значения последующих in аргументов для использования на выходе[3].

– И, главное, выделение процесса форматирования вывода в отдельную сущность позволило реализовать то, ради чего затевался переход на структурированный вывод — форматирование вывода в машиночитаемом виде. В качестве примера такого

[3] см. *prctl* в `process.c`, *accept*/*accept4* в `net.c`

формата был выбран JSON. Заложенная архитектура позволяет просто реализовывать дополнительные машиночитаемые и человекочитаемые синтаксисы. В качестве возможных вариантов есть pcap/pcapng [1], YAML.

Текущее состояние

Работы по переходу на структурированный вывод ещё не закончены. К настоящему моменту стабилизирован API структурного вывода [2], сконвертировано примерно 65% декодеров системных вызовов, реализованы выводы в форматах legacy и JSON, реализована базовая проверка, является ли вывод в формате JSON корректным JSON (проверка на отсутствие остатков непосредственных вызовов функций печати). Регулярно производится `git rebase` на master-ветку *strace*. Из несделанного — помимо оставшихся декодеров системных вызовов[4] — отсутствие тестов для формата JSON. Прогресс разработки можно отслеживать в соответствующем репозитории [3].

Литература

[1] PcapNg draft specification `https://github.com/pcapng/pcapng`

[2] Structured output readme. `https://github.com/lineprinter/strace/blob/structured/README-structured.md`

[3] Structured output branch repository. `https://github.com/lineprinter/strace/tree/structured`

[4] Среди которых можно отметить системный вызов *ioctl*.

Глеб Фотенгауэр-Малиновский
Москва, ООО «Базальт СПО»

Проект: ALT-RPM `https://www.altlinux.org/Rpm-4.13`

Альтернатива «горбатому» RPM — двугорбый RPM

Аннотация

Используемый в ALT пакетный менеджер RPM является форком RPM версии 4.0.4, выпущенной в феврале 2002 года. За эти годы оба проекта сильно изменились. В докладе будет рассказано о устройстве RPM и об успехах достигнутых за последние полгода в переносе нужного нам функционала на современную версию RPM – 4.13.

История RPM

В 1995 Эрик Троан и Марк Эвинг начали разработку RPM. Это была уже третья попытка (после RPP и PM) компании Red Hat сделать пакетный менеджер. С 1998 года к проекту подключился Джефф Джонсон и почти сразу стал основным разработчиком. В результате ухода Джеффа Джонсона из Red Hat в 2005 году, появилось два независимых проекта разработки RPM – RPM5.org, который продолжает разрабатывать Джефф Джонсон и RPM.org, который в основном разрабатывают сотрудники Red Hat.

Устройство RPM

RPM можно разделить на две части: установщик пакетов rpm и сборщик пакетов rpmbuild, которые сосуществуют в одном проекте больше по историческим причинам, чем по техническим. Установщик пакетов помимо собственно установки, удаления и обновления пакетов, обслуживания системной базы данных установленных пакетов, но и включает в себя библиотеку librpm, предоставляющую API для работы с базой данных и установки пакетов.

Ситуация с RPM в ALT

С 2002 года в RPM было добавлено множество фич, и им было собрано в Сизиф больше 84,5 тысячи бинарных пакетов из 17,5 тысяч

исходных пакетов. Главным поводом для обновления RPM является API librpm. Тот API что есть у нас уже много лет не устраивает современные проекты. Обновление RPM дело рискованное, потому что после обновления все бинарные пакеты должны продолжать устанавливаться, а все исходные продолжать собираться в такие же бинарные пакеты.

Было принято решение разделить обновление RPM на две стадии: обновление установщика (включая библиотеку и её API) и обновление сборщика rpmbuild.

Основные отличия rpm:

- Зависимости на символы в библиотеках;
- Учитывание времени сборки пакета при сравнении версий;
- Отсутствие epoch не считается эквивалентным epoch=0;
- Другое устройство файловых триггеров;

Результаты

На данный момент первую стадию можно считать завершённой – разделённые установщик rpm-4.13 и сборщик rpm-build-4.0.4 готовятся к отправке в Сизиф. Для этого обновления потребовалось исправить пакеты, использующие API librpm, в первую очередь – apt.

И. Захарьящев, Д. Медведев, А. Черепанов.
г. Москва, ООО «Базальт СПО»

Пути миграции корпоративных информационных систем на свободное программное обеспечение

Аннотация

Переход с проприетарных решений на свободные на предприятии вызван необходимостью экономить деньги, а также избавляться от vendor lock. В данной статье рассматривается миграция нежелательных по разным причинам проприетарных, популярных в прошлом, инфраструктурных систем на свободное программное обеспечение. Доклад не претендует на полноту охвата и рассматривает только критически важные архитектурные решения.

У многих организаций есть необходимость по разным причинам мигрировать с проприетарных решений на свободные в области управления рабочими станциями и корпоративной почтой. Хотя в интернете есть большое количество примеров такой миграции, они в основном ориентированы на малое (20-200 рабочих мест) предприятие. Для более крупных предприятий такие примеры полезны, но различия в масштабе диктуют другие решения. О таких решениях и пойдет речь в докладе.

В этом докладе рассматриваются решения, рассчитанные на работу с 10000-30000 пользователей и 1-2 рабочими площадками. Исходные данные — развёрнутая структура ActiveDirectory и Microsoft Exchange.

Проблемы, которые нужно решать:

- какой уровень совместимости с существующими клиентами необходим. Дело в том, что наиболее распространенные конфигурации включают Microsoft Outlook 2003, 2007 или 2010, которые умеют работать в двух разных по возможностям режимах — простого почтового клиента и клиента Exchange. Разные варианты перехода сохранят разные подмножества функциональности.
- какие изменения надо будет производить на клиентских компьютерах — это важно из-за больших трудозатрат.
- какие изменения в серверной инфраструктуре нужны с учетом двойной нагрузки в процессе миграции и тестовых конфигураций.
- как будет происходить тестирование конфигурации
- как будет происходить сам переход
- какие организационные меры и поддержка руководства нужна
- как будет происходить обучение пользователей.

Обычно при миграции руководствуются следующими критериями, соблюдение которых позволяет успешно её провести:

1. доступность (то есть время простоя должно быть сведено к минимуму);
2. масштабируемость (что критически важно для рассматриваемых масштабов);

3. сохранение ключевой функциональности (при этом наличие второстепенных возможностей могут быть проигнорированы);
4. сохранение данных пользователей.

При этом такие критерии как привычность и наличие графических интерфейсов администрирования не являются определяющими для крупной современной инфраструктуры. Уровень квалификации его администраторов достаточен для понимания и критического осмысления предлагаемых вариантов миграции.

Исходя из этого, было предложено несколько вариантов конечного решения с разной степенью изменений и секвестирования функциональности:

- Контроллер домена Active Directory на Windows + SOGo;
- Контроллер домена Active Directory на Linux + SOGo;
- Чистый Postfix+Dovecot с аутентификацией в Active Directory.

Стенды были созданы на нескольких виртуальных тестовых площадках и прорабатывались независимо друг от друга.

Евгений Синельников
Саратов, ООО «Базальт СПО»

Проект: Samba `http://samba.org`

Реализация и отладка механизмов репликации в Samba 4

Аннотация

В настоящее время реализация проекта Samba вплотную подошла к тому, чтобы стать полноценной заменой для серверных платформ Microsoft в крупных инсталляциях. Тем не менее, одним основных препятствий для такой замены является неполнота и некорректность реализации механизмов репликации. В связи с этим, при попытке включения включения Samba в качестве контроллера домена в сложных конфигурациях, возникает множество проблем. В данном докладе рассмотрены основные ограничения возможностей Samba в таких конфигурациях, а также некоторые методы отладки, позволяющие выявлять соответствующие ошибки.

В минимальном объёме проект Samba представляет собой целостный набор сервисов для SMB/CIFS клиентов. При этом проект нацелен на реализацию полноценной альтернативы для доменных служб Active Directory (AD) компании Microsoft. Архитектура проекта Samba включает в себя целый ряд подсистем, среди которых одной сложнейших и критичных в реализации является подсистема механизмов мультимастерной репликации.

Данный набор механизмов позволяет автоматизировать процесс поддержки отдельных узлов участвующих в организации инфраструктуры во взаимосогласованном состоянии. Стоит отметить, что полной независимости между контроллерами домена в службах доменов AD не предусмотрено, поскольку между контроллерами эксклюзивно распределены следующие FSMO-роли:

- Владелец схемы (уникален в рамках леса доменов);
- Владелец инфраструктуры (уникален в рамках домена);
- Владелец относительных идентификаторов (уникален в рамках домена);
- Владелец доменных имён (уникален в рамках леса доменов);
- Эмулятор основного контроллера домена (уникален в рамках домена).

Стоит отметить, что роли владельца доменных имён и зоны DNS в реализации Samba формально разделены, несмотря на то, что в оригинальной реализации AD такое разделение не предусмотрено.

```
   # samba-tool fsmo show
   SchemaMasterRole owner: CN=NTDS Settings,CN=W2K8R2-DC,
CN=Servers,CN=SDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   InfrastructureMasterRole owner: CN=NTDS Settings,CN=W2K-DC,
CN=Servers,CN=SDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   RidAllocationMasterRole owner: CN=NTDS Settings,CN=KDC-01,
CN=Servers,CN=KRDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   PdcEmulationMasterRole owner: CN=NTDS Settings,CN=KDC-01,
CN=Servers,CN=KRDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   DomainNamingMasterRole owner: CN=NTDS Settings,CN=KDC-01,
CN=Servers,CN=KRDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   DomainDnsZonesMasterRole owner: CN=NTDS Settings,CN=KDC-01,
CN=Servers,CN=KRDC,CN=Sites,CN=Configuration,DC=example,DC=ru
   ForestDnsZonesMasterRole owner: CN=NTDS Settings,CN=KDC-03,
CN=Servers,CN=KRDC,CN=Sites,CN=Configuration,DC=example,DC=ru
```

Данная особенность разделения ролей между контроллерами домена существенным образом влияет на особенности первичного включения и встраивания контроллеров домена на базе Samba (SambaAD) в инфраструктуру AD. Кроме того, важным моментом при встраивании SambaAD, является так называемый максимальный уровень схемы, который определяется для всего леса доменов контроллером-владельцем схемы.

На текущий момент Samba поддерживает уровень схемы не выше 2008 R2 [3]. Соответственно, для корректного подключения SambaAD к домену необходимо, чтобы владельцем схемы, а также и владельцем инфраструктуры, были контроллеры домена с уровнем леса версии не старше Windows Server 2008 R2. При этом стоит иметь в виду, что подобном режиме могут работать и более старшие версии серверов, например Windows Server 2012 R2.

Далее, когда все основные технические условия соблюдены и ограничения для SambaAD выполнены, полная работоспособность контроллеров домена на базе Samba не гарантирована. В достаточно крупных инсталляциях возникает множество дополнительных технических ограничений, ошибок в реализации, а также известных, в разной степени документированных, недоработок.

Ошибки, недоработки и ограничения в рамках реализации механизмов репликации включают в себя следующие основные моменты:

- Механизмы основной (выполняемой во время работы сервера) и первичной (выполняемой при подключении контроллера к домену) репликации в Samba реализованы по-разному. Механизмы первичной репликации имеют существенные проблемы в производительности — время репликации для данных в несколько гигабайт достигает более 30 часов в противовес десяткам минут в оригинальной реализации контроллеров AD на базе Microsoft Windows Server;
- Одним из неочевидных ограничений, которое не относится к основным механизмам репликации, но требует, тем не менее поддержания во взаимосогласованном состоянии, является содержимое скрытого системного каталога, доступного через общий каталог Sysvol. В данном каталоге, в частности, размещаются файлы с настройками групповых политик безопасности;
- Основной и главной проблемой в реализации механизмов репликации, в текущей реализации Samba, является корректная реа-

лизация протокола DRSUAPI [2]. Эта проблема, прежде всего, связана с устаревшими (по отношению к новым версия серверов) спецификациями, опубликованными компанией Microsoft, а также с их сложностью и неполнотой [1];

- Второй, не менее важной проблемой реализации Samba, является неполнота соответствия внутренней логики работы со схемой для объектов и атрибутов, которые хранятся в дереве каталогов и доступны по протоколу LDAP, оригинальной логике. В частности, логике обработки типов атрибутов из расширенной схемы, для которых используется дополнительный атрибут msDS-IntId.

В связи со сложностью воспроизведения конкретных сценариев в проекте Samba предусмотрен специальный набор утилит тестирования и выявления регрессий — Samba Torture. Тем не менее, конкретные сценарии в комплексном окружении управляемых сервисов выявляются только на реальных стендах и только уже потом воспроизводятся в системе тестирования.

В результате методика отладки проблем, связанных с репликацией, предполагает необходимость воспроизведения достаточно сложных конфигураций и последующего детального анализа протоколов удалённого вызова процедур (RPC). Основным инструментом, для точного выявления порядка выполнения DCERPC-вызовов в процессе работы, в проекте Samba является специально изменённый (для расшифровки payload-данных «на лету») Wireshark [4]. Кроме того, одним из возможных вариантов является методика сохранения записей о DCERPC-пакетах на уровне Samba-сервера.

Таким образом, относительно подхода к отладке SambaAD, хотелось бы подчеркнуть важность подготовки типовых, воспроизводимых конфигураций, отсутствие отладки в которых и приводит, на текущий момент, к значительному разрыву между реально необходимым уровнем поддержки механизмов репликации и возможностями разработчиков по анализу воспроизводимых проблем.

Литература

[1] Microsoft, Directory Replication Service (DRS) Remote Protocol Specification, 2016, https://msdn.microsoft.com/en-us/library/cc228086.aspx

[2] Samba Community, DRSUAPI on Samba wiki, 2015, https://wiki.samba.org/index.php/DRSUAPI

[3] Samba Community, Joining a Windows Server 2012 / 2012 R2 DC to a Samba AD, 2016, https://wiki.samba.org/index.php/Joining_a_Windows_Server_2012_/_2012_R2_DC_to_a_Samba_AD

[4] Stefan Metzmacher, DRSUAPI Replication, https://www.samba.org/~metze/presentations/2007/metze_sambaxp2007_drsuapi_repl.pdf

Игорь Власенко
Киев, ALT Linux Team

Проект: Сервисы автоматизации ALT Linux Team

Инфраструктура утилит обслуживания больших репозиториев На примере пакетов для perl и python.

Аннотация

Для ухода за очень большими (>10000 пакетов) репозиториями модулей для языка программирования требуется отдельная специализированная инфраструктура утилит. На примере пакетов для perl и python.

В докладе рассматривается сложившаяся инфраструктура обслуживания репозитория пакетов модулей для perl и разрабатываемая инфраструктура обслуживания репозитория пакетов для python2 и python3.

Михеев Андрей Геннадьевич
Москва, компания «Процессные технологии»
http://www.runawfe.org/

Изменение выполняющихся экземпляров бизнес-процессов в свободной системе RunaWFE. Новые возможности версии 4.3

Аннотация

Необходимость изменения экземпляров промышленных бизнес-процессов, находящихся на этапе выполнения, является проблемой, с которой сталкиваются предприятия, использующие системы управления бизнес-процессами и административными регламентами. В докладе показано, как решена эта проблема в версии 4.3 свободной системы RunaWFE.

Изменение бизнес-процессов во время выполнения

Срок выполнения некоторых бизнес-процессов может быть очень большим. За это время могут измениться должности сотрудников, внутренние процедуры компании, законодательство и другие условия. Эти изменения могут потребовать модифицировать часть схемы выполняющегося экземпляра бизнес-процесса, по которой еще не прошли точки управления. Также может потребоваться изменить выполняющийся экземпляр бизнес-процесса в случае логических или программных ошибок, которые не были выявлены на этапе тестирования бизнес-процесса.

Существуют два случая, требующих изменения бизнес-процессов во время выполнения. В первом случае изменяются конкретные экземпляры бизнес-процессов, оказавшиеся в некорректном состоянии. Во втором случае изменяются все выполняющиеся экземпляры определенной версии бизнес-процесса, так как некорректным является само определение бизнес-процесса.

Изменение экземпляра бизнес-процесса происходит следующим образом: Создается новый экземпляр бизнес-процесса, содержащий измененную схему. В новый экземпляр переносятся из старого экземпляра все структуры данных, точки управления также переносятся со схемы старого на схему нового экземпляра бизнес-процесса. Далее старый экземпляр бизнес-процесса заменяется новым экземпляром бизнес-процесса.

Проблема может возникнуть при переносе точек управления со старой схемы на новую. Если новая схема немного отличается от старой, то каждому узлу старой схемы можно легко поставить в соответствие узел новой схемы. Однако, что делать в случае, если новая схема бизнес-процесса кардинально отличается от старой схемы?

Предлагается не рассматривать старую и новую схему как независимые. Новая версия всегда должна являться модификацией старой версии. Тогда для каждого элемента новой схемы будет понятно,

остался ли он от старой схемы или это новый элемент. При переносе точек управления на новую схему их можно перенести в элементы с теми же идентификаторами, что и те, в которых точки управления находились на старой схеме. Если если элемента старой схемы, в котором находилась точка управления, нет на новой схеме, то такую точку управления не будем переносить. Корректность преобразования в этом случае должен обеспечить бизнес-аналитик исходя из знания общей логики бизнес-процесса.

Реализация в системе RunaWFE

Для изменения конкретного экземпляра на следующую версию определения бизнес-процесса в свойствах экземпляра бизнес-процесса, в строке "Версия" рядом с номером текущей версии определения бизнес-процесса была добавлена ссылка "Переключить на другую версию" (см. Рис. 1).

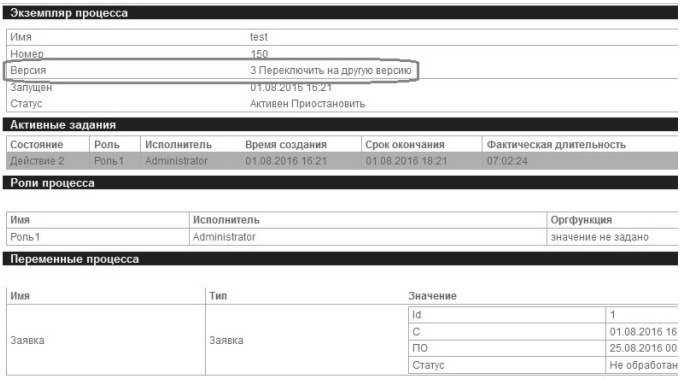


Рис. 1: Ссылка "Переключить на другую версию" в свойствах экземпляра бизнес-процесса

При клике на эту ссылку откроется форма, содержащая таблицу с информацией по доступным версиям определений данного бизнес-процесса, на которые можно перевести текущий экземпляр (См. Рис. 2).

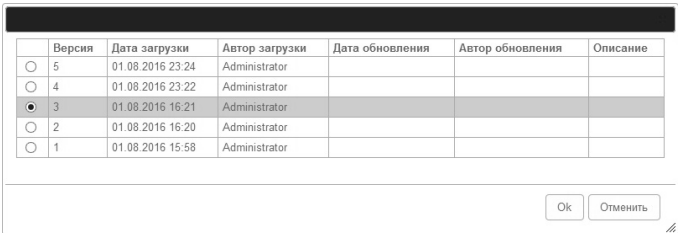


Рис. 2: Форма изменения версии для экземпляра бизнес-процесса.

После выбора нужной версии определения бизнес-процесса и клика на командную кнопку "Ok", для текущего экземпляра бизнес-процесса будет взята выбранная версия определения, точки управления будут перенесены на новую схему бизнес-процесса в соответствии с описанной процедурой. В свойствах экземпляра бизнес-процесса в строке "Версия" будет отображено новое значение текущей версии, в верхней части формы появится сообщение об изменении версии. (См. Рис. 3).

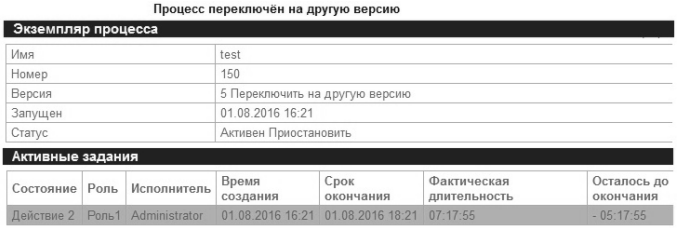
Для возможности изменения всех выполняющихся экземпляров определенной версии бизнес-процесса в свойствах определения бизнес-процесса в поле "Имя процесса" была добавлена ссылка "История" (См. Рис. 4)

При клике на ссылку "История" откроется страница, содержащая детальную информацию по всем загруженным в систему версиям данного определения бизнес-процесса (См. Рис. 5).

Рисунок 5. Страница, содержащая информацию по всем загруженным версиям бизнес-процесса

В колонке "Кол-во БП" отображаются разделенные косой чертой количество незавершенных экземпляров для данной версии определения бизнес-процесса и общее количество экземпляров для данной версии определения бизнес-процесса. При помощи клика на ссылку "Свойства" можно перейти на страницу свойств соответствующей версии определения бизнес-процесса.

На Рис. 6 представлена страница свойств бизнес-процесса. Секция "Изменить определение процесса" предназначена для загрузки нового определения бизнес-процесса.


Процесс переключён на другую версию

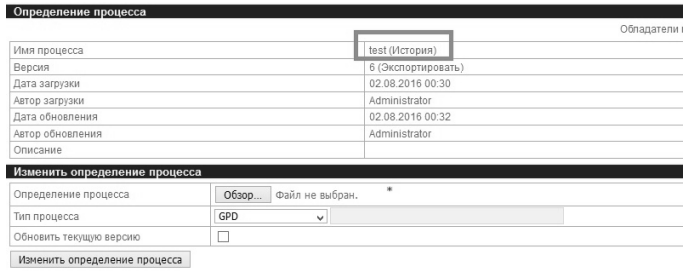
Экземпляр процесса

Имя	test
Номер	150
Версия	5 Переключить на другую версию
Запущен	01.08.2016 16:21
Статус	Активен Приостановить

Активные задания

Состояние	Роль	Исполнитель	Время создания	Срок окончания	Фактическая длительность	Осталось до окончания
Действие 2	Роль1	Administrator	01.08.2016 16:21	01.08.2016 18:21	07:17:55	- 05:17:55

Рис. 3: Сообщение об изменении версии в свойствах экземпляра бизнес-процесса.


Определение процесса

Обладатели

Имя процесса	test (История)
Версия	6 (Экспортировать)
Дата загрузки	02.08.2016 00:30
Автор загрузки	Administrator
Дата обновления	02.08.2016 00:32
Автор обновления	Administrator
Описание	

Изменить определение процесса

Определение процесса	Обзор... Файл не выбран. *
Тип процесса	GPD
Обновить текущую версию	☐

Изменить определение процесса

Рис. 4: Свойства определения бизнес-процесса с добавленной ссылкой "История" в поле "Имя процесса"

Кнопка "Обзор" позволяет выбрать файл измененного бизнес-процесса для загрузки в систему. Опция "Обновить текущую версию" позволяет обновить определение текущей выбранной версии, не изменяя ее номер. При этом произойдет обновление всех исполняющихся экземпляров текущей версии бизнес-процесса.

Другие изменения версии 4.3

- Добавлена поддержка Java8 + Wildfly10
- Добавлена подсистема отчетов
- Реализован механизм общих профилей

Определения процессов

▸ Вид Без фильтра

Всего:6

Имя	Версия	Описание	Тип процесса	Дата загрузки	Автор загрузки	Дата обновления	Автор обновления	Кол-во БП		
test	6		GPD	02.08.2016 00:30	Administrator	02.08.2016 00:32	Administrator	1 / 1		Свойства
test	5		GPD	01.08.2016 23:24	Administrator			0 / 0	Удалить	Свойства
test	4		GPD	01.08.2016 23:22	Administrator			0 / 0	Удалить	Свойства
test	3		GPD	01.08.2016 16:21	Administrator			0 / 0	Удалить	Свойства
test	2		GPD	01.08.2016 16:20	Administrator			1 / 1		Свойства
test	1		GPD	01.08.2016 15:58	Administrator			1 / 1		Свойства

Всего:6

Рис. 5: Страница, содержащая информацию по всем загруженным версиям бизнес-процесса

Определение процесса

Имя процесса	test (История)
Версия	2 Экспортировать)
Дата загрузки	01.08.2016 16:20
Автор загрузки	Administrator
Описание	

Изменить определение процесса

Определение процесса	Обзор... Файл не выбран. *
Тип процесса	GPD
Обновить текущую версию	☐

Изменить определение процесса

Рис. 6: Страница свойств версии 2 определения бизнес-процесса "test".

- Добавлена возможность работы из подпроцесса с переменными базового бизнес-процесса по ссылке
- Добавлена возможность приостановки бизнес-процесса
- Добавлена возможность удаления версии определения бизнес-процесса при отсутствии запущенных процессов этой версии
- Добавлена возможность указания выбранного перехода в конфигурации валидатора
- Добавлен обработчик для установки даты-времени в другом экземпляре бизнес-процесса
- Реализован механизм переноса срока выполнения задачи

- Добавлена возможность определять скрипт форм на уровне бизнес-процесса

Ссылки

1. Ссылка на сайт проекта RunaWFE: `http://runawfe.org/rus`

Антон Новиков
Москва, АО «МЦСТ»

Использование нативных данных конфигурации для сборки в кросс-режиме

Аннотация

Сборка пакетов программ на отличной от целевой программно-аппаратной платформе рождает ряд трудностей. Одна из них заключается в конфигурации пакетов под конкретные особенности оборудования и операционной системы целевой платформы. В докладе рассмотрены основные вопросы конфигурации пакетов при сборке в кросс-режиме и вариант решения с использованием нативных данных конфигурации.

При разработке дистрибутива операционной системы возникает вопрос поддержки нескольких различных программно-аппаратных платформ. Одно из наиболее эффективных решений данной проблемы — использовать систему кросс-сборки. Такая система должна обеспечивать удобный инструмент для сборки дистрибутива, как целиком, так и отдельных компонентов с учётом их зависимостей. Система сборки так же должна иметь ограниченный доступ к аппаратуре и окружению вычислительного комплекса, на котором она выполняется, иначе нельзя гарантировать воспроизводимость результатов сборки при замене сборочных серверов.

Сборка в кросс-режиме позволяет значительно ускорить разработку дистрибутива для продуктов, не требовательных к оборудованию (портативных устройств, автоматизированных рабочих мест и т. д.) за счёт выполнения ресурсоёмких задач на любых других производительных вычислительных комплексах.

Помимо очевидных плюсов, кросс-сборка имеет большое количество «подводных камней», например конфигурация пакетов программ

с использованием бинарных файлов. В данной работе рассмотрены вопросы связанные с инструментами autoconf. Посредством анализа выходных данных конфигуратора при нативном исполнении можно собрать и кэшировать необходимые значения для стандартных параметров autoconf. Полученные данные используются при сборке в кросс-режиме. Таким образом, периодически собирая данные для кэширования можно обеспечить максимальное соответствие кросс-сборки и нативной сборки пакета программ, а так же исключить критические ошибки при работе компонентов дистрибутива, связанные с некорректной конфигурацией пакета. Проблема различия нативной конфигурации и конфигурации в кросс-режиме существует для многих популярных сборочных систем и требует индивидуального подхода.

Решение рассмотренных проблем значительно повышает качество готового продукта и увеличивает скорость разработки дистрибутива операционной системы.

Мария Захарова, Николай Золотарёв
Москва, АО «МЦСТ»

Разнородное регрессионное тестирование

Аннотация

При комплексном тестировании программно-аппаратных комплексов разработчики сталкиваются с различными проблемами. К ним относится необходимость проверки вычислительных комплексов, отличающихся как с точки зрения программных составляющих, так и аппаратуры. Кроме того могут ставиться различные сроки, ограничивающие допустимое время на тестирование, что вызывает необходимость иметь несколько доступных сценариев проверки. В докладе описаны вышеперечисленные проблемы и способы их решения.

При осуществлении тестирования программного обеспечения должна быть обеспечена максимальная проверка как со стороны его корректной работы, так и производительности. Работа посвящена созданию системы тестирования вычислительных комплексов «Эльбрус».

Главной задачей тестирования является достижение определённого уровня качества программного обеспечения. На этом этапе необходимо определить основные критерии качества программного обеспечивания:

- Функциональность — выполнение всех возможностей данного программного обеспечения, заявленых разработчиком.
- Надежность — работа программного обеспечения без сбоев.
- Производительность — работа программного обеспечения с приемлемой скоростью.

Для обеспечения вышеперечисленных критериев качества программного обеспечения выделяются такие направления в области тестирования, как:

- Модульное тестирование — позволяет осуществить проверку на корректность работы отдельных модулей системы.
- Комплексное тестирование — проверяет систему из одельных модулей на предмет их коректной работы при взаимодействии друг с другом.
- Функциональное тестирование — осуществляет проверку на предмет реализации функциональных требований.
- Тестирование производительности — определяет, насколько быстро работает ПО на вычислительном комплексе.
- Тестирование безопасности — позволяет оценить ПО на предмет уязвимостей.

Исходя из сказанного, определяются цели тестирования:

- Проверка соблюдения всех требований к ПО — любое несоблюдение требования может привести к таким нежелательным последствиям, как ошибки, некорректная работа, замедление скорости работы. Поэтому такая проверка позволяет устранять такие проблемы на первых этапах тестирования.
- Проверка корректной работы всех модулей системы — по отдельности каждый модуль проверяется проще, что значительно ускоряет поиск ошибок.
- Обязательная перепроверка предпринятых исправлений разработчиком на последующей итерации тестирования — позволит сократить количество негативных статусов тестирования, которые возникают по одной и той же причине.

После оценки описанных целей предъявлены требования к процессу тестирования:

- Обеспечение автоматизированного тестирования по выбранным направлениям.
- Осуществление тестирования на вычислительных комплексах (ВК), отличных как по аппаратуре, так и по составу ПО.
- Обеспечение тестирование за допустимое время.
- Регулярное отслеживание регресии тестирования при обновлении ПО.
- Обеспечение доступа к статистике проведенного тестирования за определенное время.

Разнородное регрессионное тестирование реализует все предъявленные требования к системе тестирования. Его ключевыми особенностями являются:

- Автоматизация — позволяет уменьшить трудозатратность при осуществлении тестирования.
- Гибкость — за счет поддержки множества сценариев тестирование может быть осуществлено: как одной подсистемы, так и системы в целом; как на одном вычислительном комплексе, так и на нескольких сразу; за определенный временной промежуток или неограниченное по времени.
- Отслеживание найденных уязвимостей и почтовое уведомление с краткой информаций об этом отдел разработки.
- Хранение результатов тестирования.
- Веб-интерфейс для получения статистики за определенное время или информации о конкретном тесте.

Реализация разнородного комплексного тестирования позволит осуществлять детальную проверку программного обеспечения на различных вычислительных комплексах, а также обеспечит возможность отслеживания возможных регрессий для оперативного их устранения.

Михаил Шалаев
Москва, АО «МЦСТ»

Проблемы кросс-сборки: исполнение бинарных файлов

Аннотация

За время существования свободного программного обеспечения создано огромное количество программных пакетов, сборка каждого из которых может иметь свои особенности. Разработано несколько популярных и широко распространенных систем конфигурации, большинство из которых предоставляет вариативность выбора целевой операционной системы. Понятие кросс-компиляции и необходимость сборки для целевой аппаратной платформы появились достаточно давно, но до сих пор многие разработчики програмного обеспечения предполагают, что их продукты будут собираться только на нативных системах. К сборке таких продуктов нужно подходить индивидуально. В докладе раскрыты наиболее часто возникающие проблемы и возможные пути их решения.

При создании вычислительной машины новой архитектуры неизбежно возникает проблема её программного обеспечения. Работа посвящена проблемам создания пользовательского окружения в кросс-режиме для широко используемых и новых аппаратных платформ.

Исходные требования к системе создания пользовательского окружения следующие:

- наименьшие затраты на адаптацию программного кода к новой платформе (максимальная совместимость с существующей кодовой базой);
- ограниченный доступ к аппаратуре;
- возможность быстрого масштабирования процесса разработки;
- поддержка нескольких архитектур.

За время существования свободного программного обеспечения создано огромное количество программных пакетов, сборка каждого из которых может иметь свои особенности. Разработано несколько популярных и широко распространенных систем конфигурации, большинство из которых предоставляет вариативность выбора целевой операционной системы.

При постановке задачи сборки какого-либо из существующих linux дистрибутивов в кросс-режиме возникают дополнительные проблемы, связанные с особенностями сборочных сред этих дистрибутивов и дефакто отсутствием стандартов написания сборочных скриптов при добавлении пакетов. Наиболее часто встречаются следующие проблемы:

- отсутствие опций для определения целевой платформы сборки при конфигурации;
- явное определение переменных, использующихся при сборке, например компилятора, без возможности переопределения извне, несмотря на имеющиеся механизмы динамического определения значений для этих переменных;
- явное указание путей поиска исполняемых библиотек и заголовочных файлов.

Эти и подобные проблемы можно решить как правками «на лету», так и доработкой исходных текстов.

Еще одна проблема, возникающая при кросс-сборке некоторых пакетов заключается в необходимости запускать только что собранные исполняемые файлы. Это возникает в ряде случаев:

- запуск собранных исполняемых программ в процессе проведения внутренних тестов (make check);
- запуск программ для получения документации, например при использовании утилиты help2man;
- запуск промежуточных программ, используемых при генерации файлов, необходимых для дальнейшей сборки программного пакета.

Если решить первую проблему можно отключением внутреннего тестирования пакета, то для решения остальных можно предложить следующие подходы:

- сборка на вычислительном комплексе с подменой исполнения целевых бинарных фалов локальными;
- сборка на вычислительном комплексе с исполнением целевых бинарных файлов в виртуальной машине, например, qemu;
- сборка на вычислительном комплексе с исполнением целевых бинарных файлов на удаленной машине (использование remote shell);

- сборка на вычислительном комплексе с целевой архитектурой, компиляция проводится кросс-компилятором с использованием distcc.

При использовании данных подходов появляется возможность удовлетворить требованиям, поставленным к конкретному конечному продукту. Выбирать из них следует в зависимости от доступных программных решений и имеющегося количестава аппаратных комплексов.

Михаил Шигорин
Москва, Базальт СПО

Альт на «Эльбрусе»

Аннотация

Как только у нас появился шелл на системе с процессором «Эльбрус», мы захотели портировать туда наш RPM; после этого было само собой разумеющимся «завести» и hasher. Наличие самой рабочей станции оказалось ещё более полезным.

As soon as we've got a shell on Elbrus processor we wanted to port our RPM there; upon that, it was only natural to want hasher working too. The availability of a physical system didn't hurt at all.

Эльбрус — два семейства процессоров разработки российской компании МЦСТ: SPARC-совместимая ветка и оригинальная VLIW-архитектура. Речь пойдёт о второй. Особенностями платформы в настоящее время являются малодоступность (вследствие в т.ч. применения, например, в системах ПРО) и закрытость системного компилятора (вероятно, по тем же причинам). Используем рабочую станцию «Эльбрус-401», которая автором доклада найдена вполне симпатичной на ощупь. Работающая на ней хост-система — Linux (точнее, ОС «Эльбрус», во многом близкая к Debian 5.0/7.0 и местами новее).

Я работаю в компании «Базальт СПО», которая участвует в разработке репозитория ALT Linux Sisyphus. Как только у нас появился доступ на машину с процессором «Эльбрус-4С», возникло вполне естественное желание портировать туда нашу пакетную базу. Первым этапом стало портирование пакетного менеджера (RPM версии ALT Linux, он же ALT-RPM). Когда заработал rpm, следующим этапом стал запуск hasher – инструмента, с помощью которого собираются

пакеты Sisyphus (hasher спроектирован так, чтобы не допускать влияния собираемого пакета на хост-систему, а также взаимного влияния собирающихся пакетов).

Текущая работа опирается на труды многих других людей — начальное портирование RPM было выполнено glebfm@, процедуру бутстрапа альта ранее описал kas@ по мотивам ARM-порта, а код поддержки архитектуры мы получили от сотрудников МЦСТ.

На время написания тезисов доступна базовая сборочная среда ALT для сборки в автоматически создаваемом силами hasher чруте, за исключением некоторых архитектурнозависимых пакетов вроде binutils и компилятора, которые пока alien'изированы из предоставленных разработчиком системы deb-пакетов; в сумме 500 исходных пакетов.

Основные пройденные стадии сборки:

1. сборка/установка rpm вручную в хост-окружении;
2. упаковывание всего, что попадает в hasher chroot;
3. пересборка собранных пакетов уже в hasher.

Производится итеративная пересборка с откручиванием гаек вроде —disable static —without-ssl и корректировка полученной начальной пакетной базы для возможности включения её в основной разработческий репозиторий ALT Linux Sisyphus.

В целом, работа позволила оценить достоинства и недостатки:

- e2k как целевой платформы;
- ALT Linux как портабельного репозитория и набора инструментария;
- «бутстрапа напролом» и «раннепакетного».

Ссылки

1. `http://altlinux.org/bootstrap`
2. `http://altlinux.org/ports`
3. `http://altlinux.org/hasher`
4. `http://sdelanounas.ru/blogs/71419/`

Михаил Быков
Москва, http://diglossa.org
Проект: Морфей http://sa.diglossa.org

Морфей — архитектура и основные принципы

Аннотация

Морфей ставит целью облегчение чтения текста и понимания автора. Он по сути — простая автоматизация работы со словарем и грамматическими справочниками.

Морфей не ставит своей целью изучение источника и автоматизацию и облегчение такого изучения. Изучение источника (лингвистика, современная наука вообще) и понимание источника — противоположные активности. Понимать автора — значит дать слово автору, а не современному исследователю.

В идеале Морфей — программа, которая при клике на слово должна дать ответ на вопрос «что значит это слово» так, как ответил бы автор. В тексте Аристотеля, как ответил бы Аристотель, в тексте Панини — как ответил бы Панини. Это, разумеется, недостижимая цель. Она, однако, позволяет резко ограничить и очертить, и упростить задачу. Вдобавок в случае Санскрита есть источники, позволяющие точно синтезировать, создать, породить любое слово. Это работы древних грамматиков — Панини, Патанджали, etc — огромный корпус текстов. Морфей ставит себе ровно обратную задачу — по конкретной форме слова определить его (древние) морфологические характеристики и словарное значение. Кстати, отсутствие этой задачи — абсолютное отсутствие даже упоминания о такой задаче во всем огромном корпусе древней санскритской грамматики — само по себе кричащий о природе санскритского слова факт.

Используемая и описываемая здесь технология годится, однако, для любого языка.

Морфей использует только Javascript, на сервере это node.js, база данных CouchDB также использует JS. Веб — сам себе JS. Код можно посмотреть здесь: `https://github.com/mbykov`

Морфей основан на трех базовых модулях,

- relax.js — взаимодействие с CouchDB
- shiva-sutras.js — создание удобных подмножеств символов

- sandhi.js — разбиение и объединение строк согласно правилам sandhi

1. алгоритм

Процесс обработки сложного слова — samAsa схематически такой (модули подробно описаны в документации на гитхабе):

- запрос-словоформа очищается от влияния соседних слов и преобразуется в стандартную форму, например, анусвара M заменяется на словарную форму -m, etc (outer-sandhi.js);
- слово разбивается на цепочки-chains вероятных «чешуек«-flakes (vigraha.js);
- по словарю выделяются несклоняемые flake, — либо indeclinable-avyaya, либо готовые формы (a.k.a. termins)
- несклоняемые выделяются в отдельный массив, и из всех flakes остаются потенциально склоняемые
- для каждой склоняемой flake восстанавливается вероятная словарная форма
- отдельно для глаголов (tiNanta.js):
- и имен (subanta.js)
- образуется массив запросов к словарю, queries
- запрос несет в себе морфологическую информацию, (стем, окончание, тип окончания, часть речи, etc)
- образуется массив уникальных строк — стемов: stems
- массив stems проверяется на наличие в словарях
- из всех flakes выбираются обнаруженные в словаре, нужные flakes, теперь они называются, в соответствии с традицией, padas
- из всех queries выбираются соответсвующие найденным padas,
- из всех chains выделяются лишь те, которые состоят из найденных padas,
- им присваются веса, чтобы более правдоподобные шли в начале
- сейчас вес полагается равным произведению кубов длин padas, приведенному к кубу длины исходного сложного слова - samAsa
- результат отображается в консоли, на экране, etc

2. описание основных модулей:

в идеале, каждый модуль должен иметь два метода — синтезирующий словоформу по правилам Панини, и анализирующий - восстанавливающий словарную форму слова. Сейчас в Морфее этому условию удовлетворяет только sandhi.js

2.a — shiva-sutras.js

модуль конструирует произвольные необходимые наборы символов (звонкие согласные, долгие гласные, etc). Это вспомогательный модуль, позволяющий полностью разделить данные и код.

2.b — sandhi.js

имеет два метода, .del(samAsa, second) и .add(first, second) и набор взаимно дополнительных тестов

2.d — vigraha.js

разделение сложного слова — samAsa — на цепочки вероятных чешуек-flakes. Идея заимствована у SpamAsassin-a. Слово раскладывается на все возможные комбинации, входящие в массив частей от любого символа до любого следующего за ним. Невозможные чешуйки отбрасываются. Затем из них с помощию рекурсивной функции составляются вероятные цепочки-chains. Это очень ресурсозатратная процедура. Слово длиной 50 символов может давать 150 тысяч вероятных комбинаций. В зависимости от сложности и неоднозначности разбиений с учетом сандхи.

Я пока не знаю другой свободной программы, кроме Морфея, выполняющую эту работу, а было бы полезно, например, для немецкого или русского.

В качестве тестов vigraha.js я использую разложение всех сложных слов Бхагавад-Гиты, доступное в авторитетном источнике в сети. [1], [2]

И дополнительные, составленные вручную наборы тестов, всего около 800 тестов.

2.c — tiNanta.js — анализ морфологии глагола.

Модуль основан на результатах работы dr. Dhaval Patel & dr. Sivakumari Katuri из Хайдарабадского университета. Их программа 'SanskritVerb' [3] как раз является синтезирующей частью, порождающей около 245 тысяч словоформ. Которые являются тестами для модуля.

2.d — subanta.js — анализ морфологии имен.

Пока всего 1717 тестов, найденных вручную в сети.

2.c-d. и tiNanta.js, и subanta.js построены по одинаковому принципу: они создают набор проверяемых характерных окончаний на лету, прямо из набора тестов. Для конкретного теста, имеющего все формы склонения-спряжения, высчитывается общий stem, и набор окончаний. Некоторые наборы окончаний встречаются часто. Обычно они отображены в стандартных грамматиках. А в некоторых полученный stem равен вообще пустой строке. В это случае тест становится автоматически обработкой т.н. исключения. После чего тесты обязаны автоматически сходиться, и, очевидно, сходятся. (В tiNanta.js есть еще порождение соответствий корня-dhatu и всех его стемов. И проверка на присутствие полученного корня-dhatu в списке корней - DhatuPatha. Это отдельная тема).

2.e в процессе отображения в вебе используются вспомогательные модули

- akshara.js — простой редактор
- salita.js — конвертер nagari в транслитерации SLP1 и IAST для удобства работы в консоли
- sanote.js — преобразование традиционной морфологической нотации в европейскую и vise versa

3. — словари

Морфей пока что использует 4 словаря.

3.a — словари Monier-Williams и Apte [4], [5] в редакции группы волонтеров под руководством того же dr.Dhaval Patel [6].

3.b — словарь TS — termins. Терминами я здесь назвал конечные формы, сложные по строению, но очень часто встречающиеся в тексте. В основном это формы местоимений, числительные (пока отключено), и, возможно, любые исключения. Термины - потому что не требуют дальнейшего анализа.

3.c — словарь словоформ Бхагавад-Гиты [2]. Этот словарь содержит конечные словоформы, как словарь TS, но не имеет морфологической расшифровки. Это просто удобство, особенно для начинающих изучать язык. Эти словоформы часто встречаются в любых текстах.

Всего словари содержат около 250 тысяч словарных статей.

4. в версии 0.4 пока не сделано:

- анализ слов с суффиксами (kridanta и taddhita),
- сложных глагольных форм, таких как интенситивы, дезидеративы, etc
- звательного падежа
- типа сложных слов samAsa

Это будет выполнено в следующих версиях Морфея.

5. цель разработки

Здесь, на конференции в Калуге, хочется отметить, что моя цель в разработке этой свободной программы — создать не конкретный сервис, а дешевую технологию разработки подобного класса программ, для любого языка. Для этого необходимо иметь только лишь одну вещь — релевантный массив тестов. (И, очевидно, словари). Не обязательно иметь тесты, отображающие все вычурные тонкости теорий современного языкознания. Но необходимо отобразить в тестах все часто встречающиеся и характерные случаи речи и письма. Может быть, для ограниченных ситуаций, на ограниченном словаре, и т.д. Вдобавок полученные модули могут быть в дальнейшем как угодно улучшаться и развиваться, и заменяться на необходимые для более тонких задач. Мне кажется, пришло время разработки подобной программы для любого окружающего нас языка. По-моему, это как раз задача для сообщества разработчиков открытого софта, и не только программистов.

Нужно еще отметить, что составление массива тестов для произвольного языка может быть не очень дешевой задачей, но она в любом случае необходима. Составление свободного и доступного массива тестов для любого распространенного языка есть отдельная необходимая задача для составителей официальных корпусов национальных языков. По-моему, еще не осознанная. Что бы ни являлось результатом их работы, релевантный массив тестов является уже результатом и абсолютно необходимой основой для любой разработки и оставляет руки свободными для всех сторонних разработчиков. Однако пока мы этого нигде не видим.

Ссылки

1. http://sanskritdocuments.org/doc_giitaa/bhagvadnew.html?lang=sa
2. http://sanskritdocuments.org/doc_giitaa/bgwords.html?lang=sa
3. http://sanskritworld.in/sanskrittool/SanskritVerb/tiGanta.html
4. http://www.sanskrit-lexicon.uni-koeln.de
5. https://github.com/sanskrit-lexicon
6. https://github.com/drdhaval2785

Петров Евгений Николаевич, Кононова Александра Игоревна
Москва, НИУ МИЭТ

Проект: Pybico — Python Bibliography Converter
https://github.com/fiddenmar/pybico

Разработка свободного программного средства обработки и структуризации библиографических данных

Аннотация

Формирование библиографических отчетов для отдела научно-технической информации является задачей, требующей большого числа рутинных ручных операций по приведению библиографических данных различного вида и из различных источников в единую структуру требуемого формата, требования к оформлению которой могут со временем меняться. Данная работа посвящена разработке свободного программного средства, решающего описанную проблему посредством перевода библиографических данных различного вида в единое представление с возможностью дальнейшего экспорта в требуемый формат.

В настоящее время в вузах существует практика сбора сведений о публикациях преподавательского состава и студентов кафедр. Эти сведения подаются коллективно или в индивидуальном порядке ответственному лицу, которое на основе полученных данных формирует единый отчет в виде, требуемом отделом научно-технической информации.

Формирование отчета представляет собой рутинный труд по переносу предоставленных данных в сводную таблицу, отвечающую заявленным требованиям. Как и во всякой рутинной работе, выполняемой вручную, велико влияние человеческого фактора на результат, будь

то опечатки или пропущенные по неосторожности публикации. Помимо этого, требования к оформлению итогового отчета могут поменяться и всю работу придется переделывать заново. Дополнительной трудностью является то, что преподаватели подают сведения о публикациях в вольной форме, зачастую не соблюдая правила оформления: в виде библиографий, не всегда оформленных по ГОСТ 7.0.5-2008 [1], в виде таблиц и списков различных форматов.

Предлагаемым решением описанной проблемы является разработка программного средства обработки и структуризации библиографических данных. В основе работы данного программного средства лежит принцип преобразования входных данных в различных форматах в единое внутрипрограммное представление с занесением в базу данных, а также последующий перевод этого внутрипрограммного представления в требуемый отчетный вид, будь то таблица или библиографический список. Из-за использования переходного представления достигается независимость работы программы от формата как входных данных, так и выходных: однажды занесенные в базу данных сведения могут быть повторно использованы при изменении требований к отчету.

Возможность получения данных из готовых открытых источников позволяет существенно сократить трудозатраты как преподавательского состава, так и ответственного за формирование итогового отчета лица, по этой причине в разработанном программном средстве присутствует функционал по выборочному импорту данных из научной библиотеки elibrary на основе списка идентификаторов интересующих авторов. В то же время поддержка импорта из локальных файлов не менее важна, так как иногда требуется внести в отчет еще не вышедшие, но уже принятые к публикации материалы.

При разработке использовался язык программирования Python [2], что позволило осуществить быстрое прототипирование и показать первые результаты в кратчайшие сроки.

Разбор входных данных выполняется с помощью регулярных выражений [3] в зависимости от формата, передаваемого в качестве аргумента командной строки. Другие параметры включают отображение справки, пути к входным и выходным файлам, имя пользователя и путь к паролю от базы данных, формат выходных данных. В настоящее время поддерживается импорт текстовых списков библиографических сведений и автоматический сбор информации на основе

идентификаторов авторов на сайте elibrary. В качестве формата экспорта поддерживается сохранение базы данных в сводную таблицу.

В перспективе планируется расширить число поддерживаемых форматов импорта и экспорта, что позволит пользователю свободно переводить данные из одного представления в другое в зависимости от своих потребностей. Помимо этого, рассматривается возможность создания веб-сервиса в локальной сети вуза с целью создания единой библиографической базы данных и добавления гибкой фильтрации данных для настройки выдаваемых результатов под нужды разных кафедр.

Литература

[1] Стандартинформ, Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления., 2008

[2] *Python Software Foundation*, Our Documentation,`https://www.python.org/doc/`

[3] *Jan Goyvaerts*, Regex Tutorial, Examples and References `http://www.regular-expressions.info/`

Алексей Турбин
Рязань

Типизация ABI

Аннотация

Имеющаяся реализация бинарных зависимостей на основе set-версий позволяет контролировать наличие нужных символов в библиотеках. Показано, что реализация может быть расширена таким образом, чтобы учитывался и тип символа, а не только его имя; так чтобы обеспечить не только наличие нужных функций, но и совместимость их прототипов.

Введение

В предыдущей работе [1] показано, что понятие *обратной совместимости* часто является обманчивым; для достижения реальной *бинарной совместимости* между библиотеками и программами (ABI) нужно контролировать не только и не столько придуманные версии библиотек, сколько *разрешимость символов* (т. е. фактическое наличие нужных функций в библиотеках). Там же обсуждается, насколько компактным может быть вероятностное представление множеств, при котором каждый символ представлен n-битным хешем. Вскоре был найден и способ кодирования: *код Голомба–Райса* обеспечивает оптимальное сжатие дельт (расстояний) между соседними хешами [2]. Таким образом, у библиотек в репозитории появились зависимости вида `Provides: libfoo.so.1 = set:...`, а у программ — `Reqruies: libfoo.so.1 >= set:...` (где вместо троеточия идет закодированная последовательность хешей). При сравнении таких *set-версий* rpm выполняет проверку $R \subseteq P$.

После этого было реализовано автоматическое создание `debuginfo`-пакетов, а пакеты в репозитории стали компилироваться с флагом `-g` — с отладочной информацией в формате DWARF. Там содержится описание переменных и функций, в том числе аргументов функций и их типов. Используя эту информацию, символы можно наделить *типом*; тогда при сравнении множеств будет проверяться не только разрешимость символов по имени, но и совпадение их типов (для функций — прототипов). Это напоминает декорирование имен (name mangling) в C++, где к имени функции добавляется информация об аргументах. В обоих случаях типизация является *атомарной*: возможно только полное совпадение; никакой дальнейшей структуры в типизированных символах не усматривается. Если символ меняет свой тип, хотя бы и условно совместимым образом, то образуется другой, новый символ (с другим хешем).

Для языка Си полный учет типов порождает довольно жесткую конструкцию. Если тип изменяется условно совместимым образом, то желательно, чтобы символ остался прежним. Далее обсуждается, какой может быть *облегченная* типизация. Показано, что для ее реализации сборка с флагом `-g` требуется только для библиотек. Однако сборка приложений с флагом `-g` открывает интересные возможности для *дополнительной* типизации.

Типизация

Допустим, функция принимает аргумент типа `int`; а в новой версии тип аргумента меняется на `long`. При этом функция сохраняет совместимость: ведь на 32-битных архитектурах `int` и `long` — это один и тот же тип. На 64-битных архитектурах это два разных типа (32 и 64-битный), но при передаче параметра фактически используется 64-битный тип — либо за счет передачи через регистр, либо за счет выравнивания на стеке [3, с. 23]. Аналогично, при добавлении или удалении спецификатора `unsigned` шансы нормального вызова остаются высокими. Поэтому любой целочисленный аргумент наделяется типом `i`.

Рассмотрим теперь квалификатор `const` вместе с аргументом типа `char *` (Си-строка). При добавлении квалификатора совместимость сохраняется полностью, а при удалении совместимость ухудшается. Мы сейчас выполнили *структурное сравнение* на совместимость, которое противоречит принципу атомарности. Ничего не остается, как только полностью исключить квалификатор `const` из рассмотрения, признав его несущественным. Но можно привести и другие аргументы в поддержку этого решения. Использование `const` часто вызывает затруднения (кроме простейших случаев), см. напр. о типе `argv` в `getopt(3)`. В других языках вместо `const` используется явное задание in/out параметров. В третьих языках, таких как Rust, отслеживается владение (ownership) объектами. Нам не хотелось бы признавать истинной ту или иную языковую абстракцию и проверять бинарную совместимость в соответствии с ее догмами.

Рассмотрим типизацию структур, передающихся по значению. У структуры часто бывает два имени: тег структуры и короткое имя, заданное через `typedef`. Но с точки зрения бинарной совместимости вообще не следует привязываться к имени. Поэтому структуры наделяются типом `bN`, где `N` — размер структуры (`sizeof`).

Массивы типизируются как `aT`, где `T` — тип элемента массива (это касается только глобальных переменных; в параметрах массив преобразуется в указатель).

Указатели тоже хотелось бы типизировать как `pT`; однако, в отличие от массивов, тип `T` может быть непрозрачным (opaque). Кроме того, тип может быть полностью скрыт указателем `void *`. Получается, почти все указатели на данные должны иметь тип `p`; можно лишь учитывать уровень косвенности (тогда, например, аргумент

`char *argv[]` будет иметь тип `pp`). Кроме того, указатели на функции можно типизировать полностью. В качестве примера рассмотрим функцию `signal(3)`:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

Соответствующий ей символ наделяется типом так:

```
signal (i, p(i)) -> p(i)
```

Сопоставление

Как и в имеющейся реализации, для сопоставления требуемых символов с предоставляемыми используется загрузчик `ld.so` — он запускается в специальном отладочном режиме, похожим на `ldd(1)`. Каждый требуемый символ наделяется типом соответствующего предоставляемого символа; подразумевается, что на стадии сборки типы символов соответствуют (по этой причине не требуется сборки клиентского кода с отладочной информацией).

Однако такое сопоставление нельзя считать слабым местом всей конструкции. Сборка пакетов является моментом *наибольшего согласования* между клиентским и библиотечным кодом; при наличии несовместимости код не соберется, либо не отработает `make check`. Можно даже сказать, что одна из главных задач системы управления пакетами — это *перенос гарантий*, достигнутых во время сборки, на другие конфигурации. А именно, с помощью бинарных зависимостей мы пытаемся гарантировать, что при запуске программы символы будут разрешаться в некотором смысле *не хуже*, чем во время сборки. Логично, что в обоих случаях используется загрузчик `ld.so`.

Рассмотрим теперь, как можно хранить типы символов. Ведь `debuginfo`-пакеты помещаются в отдельную компоненту репозитория, которая недоступна при сборке. Тогда можно было бы поместить отдельный файл с описанием символов в `devel`-пакет. Однако хотелось бы иметь полностью автоматическое решение. Поэтому предлагается хранить дополнительную информацию о символах в самих библиотеках, в специальной секции `.rpm.proto`. Эта секция даже не обязательно должна быть структурированной, как другие секции ELF; ее содержимое можно сжать, как это делается с секцией `minisymtab`.

Для сжатия небольших текстовых файлов особенно хорошо подходит алгоритм `ppmd`.

В современной разновидности формата ELF, в которой используется секция `.gnu.hash`, экспортируемые символы в таблице `.dynsym` упорядочены по значению хеша, а неопределенные символы сгруппированы в начале таблицы [4, с. 9]. Это делает возможным особенно простое представление секции `.rpm.proto`: начиная с *первого экспортируемого символа*, записывать типы символов по одному на строку.

Дополнительная типизация

Дополнительная типизация не связана напрямую с символами. С ее помощью можно попытаться контролировать типы данных.

Пусть библиотека определяет какую-то структуру, а программа объявляет переменную этого типа. Тогда есть смысл контролировать размер этой структуры (так же, как при передаче структуры по значению). Структуру можно считать связанной с библиотекой, если 1) она используется в аргументах какой-либо функции библиотеки; 2) во время сборки библиотеки заголовочный файл с определением этой структуры устанавливается в `/usr/include`. Тогда библиотека предоставляет *псевдосимвол*, который описывает размер этой структуры.

Литература

[1] Алексей Турбин. Комплементарное хеширование подмножеств // Седьмая конференция разработчиков свободных программ. Тезисы докладов. М., 2010. С. 63–66.

[2] Felix Putze, Peter Sanders, Johannes Singler (2007)
Cache-, Hash- and Space-Efficient Bloom Filters

[3] Michael Matz et al. System V Application Binary Interface.
AMD64 Architecture Processor Supplement. Draft Version 0.99.7

[4] Ulrich Drepper. How To Write Shared Libraries
`www.akkadia.org/drepper/dsohowto.pdf`