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

Пятнадцатая конференция разработчиков свободных программ

Калуга, 28–30 сентября 2018 года

Тезисы докладов

Москва
Базальт СПО, МАКС Пресс
2018

УДК 004.91
ББК 32.97
П99

П99 Пятнадцатая конференция разработчиков свободных программ: Тезисы докладов. Калуга, 28–30 сентября 2018 года / отв. ред. Черный В. Л. — М.: Базальт СПО; МАКС Пресс, 2018. — 100 с. : ил.

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

ISBN 978-5-9908979-8-4
ISBN 978-5-317-05908-8 (МАКС Пресс)

Программа конференции

28 сентября, пятница

Дневное заседание
14.30–18.35

14.30-14.45 Новодворский А. Е., ООО «Базальт СПО». Приветственное слово

14.45–15.15 Федорчук А. В.

Будущее Linux'описательства: есть ли оно? 7

15.15–15.45 Миронова О. В., ООО «Базальт СПО»

Локализация СПО — больше, чем перевод 9

15.45–16.15 Савченко А. А., ООО «Базальт СПО»

Уязвимости в лицензиях СПО 12

16.15–16.45 Никифорова Т. С., Dentons

Правовые риски свободных программ: чем юристы пугают ваших заказчиков 15

16.45–17.05 Кофе-пауза

17.05–17.35 Левин Д. В., ООО «Базальт СПО»

Modern strace 18

17.35–18.05 Власенко И. Ю., ALT Linux Team

Стратегия развития систем автоматизации сопровождения пакетов 23

18.05–18.35 Липатов В. А., Державин Д. К., ООО «Базальт СПО»

Поддержка сторонних пакетов rpm в ОС Альт 24

29 сентября, суббота

Утреннее заседание
10.00–13.50

10.00–10.30 Латий О. О., Брестский государственный технический университет

Высокопроизводительный модуль отрисовки графиков на базе использования библиотеки Qt 27

10.30–11.00 Пынькин Д. А., Collabora Ltd.

Debos: human-friendly OS creator. Debos: утилита для создания ОС на основе Debian 33

11.00–11.30 Власенко И. Ю., ALT Linux Team

Система Логовед для автоматизации QA 36

11.30–12.00 Белявский Д. М., ООО «Криптоком»

Российская криптография в свободном ПО 38

12.00–12.20 Кофе-пауза

12.20–12.50 Костарев А. Ф., Братчиков И. А., ООО «Невод», ООО «Новая платформа»

Организация процесса DevOps на платформе контейнеризации docker 39

12.50–13.20 Кантер Л. Б., Cloud Linux Inc

Построение корпоративной системы управления идентификационной информацией на базе FreeIPA 44

13.20–13.50 Силаков Д. В, Virtuozzo

Шаблоны контейнеров в OpenVZ 7 47

13.50–15.10 Перерыв на обед

Дневное заседание
15.10–18.30

15.10–15.40 Шабалин А. В, ООО «Базальт СПО»

Сети в облаках 50

15.40–16.10 Ермаков Н. О., ООО «Базальт СПО»

Открытая архитектура RISC-V 51

16.10–16.40 Черепанов А. С., ООО «Базальт СПО»

Планы на дистрибутивы Альт 9.0 54

16.40–17.00 Кофе-пауза

17.00–17.30 Боковой А. Г., Red Hat

Технология удаленного шифрования дисковых подсистем в Red Hat Enterprise Linux 57

17.30–18.00 Синельников Е. А., ООО «Базальт СПО»

Реализация групповых политик в решениях на базе дистрибутивов ALT 63

18.00–18.30 Быстренин Г. А., Бубнов С. В., ООО «Базальт СПО»

Развертывание инфраструктуры средствами Ansible на ProxmoxVE сервере 67

30 сентября, воскресенье

Утреннее заседание
10.00–14.20

10.00–10.30 Симаков Р. А., ООО «Ред Софт»

Новые возможности СУБД Ред База Данных 3.0 70

10.30–11.00 Братчиков И. А., Костарев А. Ф. ООО «Новая платформа», ООО «НЕВОД»

Технологическая программная платформа Flexberry с открытым исходным кодом для профессиональной разработки программного обеспечения 73

11.00–11.30 Михеев А. Г., ООО «Процессные технологии»

Новые возможности свободной системы управления бизнес-процессами и административными регламентами RunaWFE 78

11.30–12.00 Шигорин М. А., ООО «Базальт СПО»

Альт на Эльбрусе: обе вершины 81

12.00–12.20 Кофе-пауза

12.20–12.50 Мельников И. А., Терёхин Д. С., ООО «Базальт СПО»

ALT на платформе MIPSel 83

12.50–13.20 Ставцев Р. Г., АО «Байкал Электроникс»
Процессор Байкал-Т1. Программное и аппаратное окружение 86
13.20–13.50 Колотников А. В., АО «Байкал Электроникс»
Оптимизация шифрования на Байкал-Т1 по ГОСТ28147-89 89
13.50–14.20 Медведев Денис Леонидович, ООО «Базальт СПО»
Метаинформация репозиториев: хранение и полезный состав 92

Вне программы

Турбин А. М.
zrec — формат метаданных репозитория 94

Алексей Федорчук
Москва

Проект: Система Cintu и Книга о ней `https://www.cinia.ru/`

Будущее Linux'описательства: есть ли оно?

Аннотация

В данных тезисах рассмотрены причины нынешнего упадка книгоиздательской деятельности и пути его преодоления. Один из путей к чему — сочинение нестандартных книг. Что демонстрируется на примере проекта *Книга о Cintu*, посвящённой одноимённой системе.

Ни для кого не секрет, что книгоиздание переживает сейчас не лучшие времена, и в первую очередь это касается так называемого «технического» книгоиздания. Об одной из его отраслей, касающихся UNIX, Linux и вообще Open Source, далее и пойдёт речь. Причём в этой тематике «кризис жанра» затронул не только не только «бумажное» книгоиздание, но и «электронное».

С первым аспектом, казалось бы, всё понятно: ввиду доступности «электронок» разного вида и формата желающих заполнять своё жилище бумажными томами нынче не много. Особенно с учётом того, что изрядная их часть устареет через конечный (и обозримый) промежуток времени. Однако и «электронных» книжек рассматриваемой тематики также не густо — по крайней мере, русскоязычных. И тех, и других за последние лет 10–12 не появилось от слова «вообще».

Отсутствие общих книг по Linux'у и Open Source особенно заметно на фоне бессчётного количества индивидуальных блогов и множества коллективных Wiki-страниц. Пользу которых, конечно, никто не отрицает — но, как показывают вопросы на форумах и в соцсетях, книг они не заменяют. Ибо знакомиться по ним с Linux'ом — всё равно что изучать квантовую механику по «Краткому справочнику по физике для инженеров и студентов ВТУЗов».

Впрочем, причины отсутствия книжек по Linux'у легко объяснимо. Ведь сочинение их, не смотря на кажущуюся простоту — занятие достаточно трудоёмкое. Даже в том случае, если весь запланированный материал уже был изложен автором в виде заметок в его блоге или вставок в коллективные Wiki. Кроме того,в последнем случае оно

достаточно скучно — ведь речь идёт о «причёсывании» и сведении воедино материала о вещах, которые уже были описаны. И работа такая требует веской мотивации — заниматься этим можно или из соображений финансовых, или из «чистого энтузиазма».

О финансовых соображениях говорить не будем — поскольку, как сказали бы Ильф и Петров, копирайта у нас уже нет, а копилефта ещё нет: ни одна из предложенных моделей замены стандартной издательской практики, сложившейся в «бумажные» времена, в эпоху «электронки» не работает.

Энтузиазм же может питаться из двух источников. Первый — это личный интерес автора к описываемому предмету, и второй — понимание востребованности написанного широкими народными массами (или хотя бы узкими их кругами). И это вещи тесно связанные: если автору было не интересно писать своё сочинение — наивно было бы ожидать, что кому-то будет интересно его читать.

Однако читательский интерес не гарантирован и в случае заинтересованности автора в предмете своего сочинительства. Ибо читатель, привыкший быстро находить решение своих сиюминутных проблем в упомянутых блогах или Wiki, может быть удручён кажущимся «многобуквием» книжки. Кроме того, он полагает, что всё описанное в ней можно найти и в более «компактном» виде. Что, во-первых, не всегда лишено оснований. А во-вторых, влияет и на интерес автора к своей работе: всякий Linux'описатель «со стажем» в один далеко не прекрасный момент понимает, что почти всё то, о чём стоило написать, он уже написал, и ему становится банально скучно. Что вызывает обратную связь — книжка, написанная со скукой, и читаться будет... аналогично.

Из этого замкнутого круга существует единственный выход — сочинять нестандартные книжки. Которые автору будет не скучно писать — и тогда есть шанс, что и читать их буде не скучно. Попытка такого рода представляется в виде проекта *Книга о Cintu*, которая посвящена одноимённой системе. Система эта представляет собой микст из базовых компонентов Ubuntu, оболочки Zsh (используемой как регистрационная и для пользователя, и для администратора), среды Cinnamon и сторонних утилит и приложений, в том числе и основанных на Qt/KDE: поскольку штатных приложений Cinnamon не имеет, тут есть широкий простор для фантазии. То есть Cintu — не столько дистрибутив, сколько алгоритм построения собственной системы,

укомплектованной и сконфигурированной в соответствие с запросами конкретного применителя.

Неотъемлемой частью системы и выступает *Книга о Cintu*. Это — не формальное руководство, а описание процесса построения такой системы, её комплектации приложениями и применения её для решения практических задач. Каковыми для целевой аудитории являются работы в области антропологии, геологии и сопряжённых научных дисциплин. Общее между ними то, что они требуют, с одной стороны, геопространственной привязки первичного фактического материала, с другой — его численной обработки, завершающейся написанием научных отчётов, иллюстрированных графиками, картами etc. Чем и определяется подбор соответствующих приложений.

Однако базис Cintu (так называемая *Малая редакция*, включающая только рабочую среду и минимум утилит и приложений для её функционирования и наращивания) может быть надстроена и приложениями для любых других задач — от музыкальной деятельности и вплоть до сочинения романов и даже, страшно сказать, стихов.

Миронова Ольга
г.Обнинск, ООО «Базальт СПО»

Локализация СПО — больше, чем перевод

Аннотация

Доклад посвящён обзору текущего состояния дел в сфере локализации СПО и перспективам развития данной области.

Локализация ПО — более сложная задача, чем просто перевод. Независимо от варианта её решения (выполнение перевода специализированными компаниями или сообществом), результат зависит от работы каждого участника процесса: разработчика, переводчика, редактора, тестировщика и т.д.

За 9 месяцев работы в этом направлении нашей компанией была проделана большая работа. Мы выполнили и дополнили перевод интерфейса, материалов справки или того и другого для таких программ и приложений, как модули «Параметров системы», Dolphin, KWallet, Spectacle, Sweeper, Minuet, Kdenlive, KTimer, FreeIPA, x2go, GIMP, KAddressBook, Dia, KMail, KAlarm, krunner, KNotes, Akregator, Sieve, KRuler, Konversation, KMouth, Kleopatra,

KMouseTool, blueberry, Kontact, KAppTemplate, KSystemLog, ksysguard, Recoll, и др. Объём выполненной работы исчисляется в сотнях тысяч слов.

Полученный опыт дал нам возможность сделать определённые выводы в отношении того, какие нюансы, возникающие в процессе локализации, следует учитывать, а также какие действия необходимо предпринять для повышения качества работы.

Залогом успешной локализации любого проекта являются следующие составляющие:

1. Глоссарий и терминологическая база.
2. «Чистая» память переводов.
3. Качество исходного текста. Единое руководство по стилю.
4. Ввзаимодействие с тех. специалистами, редакторами и корректорами. Обратная связь от разработчиков и пользователей.

Учитываться должны все эти моменты.

1. Отсутствие глоссария или его низкое качество ведёт к низкому качеству перевода: на выходе получаем файл перевода с разрозненной и зачастую неверной терминологией. Сейчас глоссарии составляются вручную на wiki для крупных и объёмных программ и проектов. Текущая версия глоссария общих терминов доступна по ссылке: `https://www.altlinux.org/Localization/General_Glossary` Глоссарий общих терминов.

2. «Почистить» имеющуюся память переводов крайне сложно, так как она основана на всех готовых переводах, качество которых не гарантировано. Поэтому сейчас мы «копим» свою, чистую память переводов и в дальнейшем, возможно, разделим её по продуктам.

3. Качество исходного текста особенно важно при переводе с английского на русский язык из-за разницы в структуре языков (правила словообразования, множественное число и т.д.). Отдельную проблему представляют объединённые строки, относящиеся к разным элементам интерфейса и требующие разного перевода. Исходный код не всегда помогает понять, к чему относится та или иная строка. Комментарии разработчиков встречаются, к сожалению, крайне редко. Излишняя оригинальность исходного текста также мешает созданию качественного перевода. Поэтому качество готового перевода напрямую зависит от того, как разработчики и технические писатели подготовят текст.

В этой связи, помимо прочего, нашей командой ведётся работа по созданию руководства по стилю, за основу которого взят про-

ект национального стандарта (ГОСТа) «Стиль изложения документации пользователя программного обеспечения», рекомендации по переводу от координаторов KDE, «Методическое и справочное руководство по переводу на русский язык, тематическому редактированию, литературной правке и оформлению инженерно-технической документации» от компании «Интент», а также руководства по стилю от Microsoft и IBM. Рабочая версия доступна по ссылке: `https://www.altlinux.org/Localization/Styleguide` Руководство по стилю.

4. Самая большая проблема в процессе локализации — низкая активность сообщества по данному вопросу. Заброшенные списки рассылки, давно не обновлявшиеся веб-страницы и т.д. Поэтому необходимо привлекать к сообществу свободного ПО больше людей, заинтересованных в локализации. Причём как переводчиков, так и технических специалистов, готовых дать профессиональную консультацию по тому или иному вопросу.

Заключение

Для получения качественного локализованного продукта необходимо объединяться, привлекать как можно больше компетентных специалистов, создавать удобные для них условия — предоставлять понятные инструкции по выполнению работы, всегда быть готовыми дать обратную связь или консультацию по любому вопросу, прилагать совместные усилия по внедрению грамотных инициатив по улучшению организации процесса локализации и повышению качества работы.

Андрей Савченко
Москва, Базальт СПО

Уязвимости в лицензиях СПО

Аннотация

Лицензии на программное обеспечение можно рассматривать как программы. Как и любые программы, лицензии не идеальны и содержат уязвимости, которые эксплуатируются злоумышленниками с целью ограничения прав и свобод пользователей и разработчиков. Ситуация осложняется тем, что из-за недоработок в лицензиях можно формально их выполнить, но нарушить суть и основную идею. В данной работе будет рассмотрено как это происходит и что можно попытаться с подобными явлениями сделать.

Введение

По мере роста сложности лицензий на программное обеспечение (ПО) они сами приобретают признаки программы, поскольку они описывают набор правил и алгоритмов по использованию ПО. Разумеется, это верно как для проприетарных, так и для свободных лицензий, но в данной работе внимание будет уделено лицензиям свободного ПО (СПО).

Как и любое ПО, лицензии не идеальны, они содержат как архитектурные ошибки, так и проблемы реализации, поскольку невозможно предусмотреть все краевые случаи и особые ситуации при разработке лицензии.

Текущее состояние

Философия СПО определяет четыре свободы пользователей: свободу на выполнение, копирование, изменение и распространение ПО [1].

Ярким примером нарушения этих базовых свобод при формальном соблюдении свободной лицензии является проблема тивоизации [2]. В GPLv2 содержится уязвимость, позволяющая производителям на аппаратном уровне блокировать свободу пользователя обновлять ПО на устройстве, оставляя при этом такую возможность для производителя; тем самым возможность пользователя использовать

модифицированное ПО и все связанные с этим свободы блокируются. Формально такое поведение не нарушает GPLv2 из-за несовершенства лицензии, не оговаривающей применение свобод ПО при использовании на целевом оборудовании.

Для решения данной проблемы и устранения иных недочётов была создана GPLv3. Однако, всему есть своя цена: если текст GPLv2 составляет 340 строк (на английском), то GPLv3 — 676 строк, т.е. почти в два раза больше. Проводя аналогию с ПО: есть больше кода для обработки краевых ситуаций, но он стал сложнее для понимания простыми людьми, особенно далёкими от юридических вопросов. Сложность и запутанность лицензии — высокая цена и она отпугивает часть пользователей.

Открытые вопросы

- *Обфускация изменений.* Например, патчи ядра RedHat, слитые в единое изменение, применённое к ядру [3]. Поскольку воедино сливаются сотни как связанных, так и независимых изменений на перекрывающиеся участки кода, разбор и последующая полезная модификация данных изменений чрезвычайно усложнены. Формально GPLv2 строго соблюдается, но фактически свобода модификации и использования кода сильно ограничены.

- *Форсирование несовместимых лицензий.* Проблема ZFS [4]: лицензирование модуля ядра под свободной лицензией CDDL, несовместимой с лицензией ядра GPLv2. Краевой случай, где корпорации пытаются ослабить ограничения копилефта и многие пользователи поддерживают их, поскольку им интересы технические решения и не важны лицензионные тонкости. Юридическая проблема здесь в том, что значит «производная/объединённая работа». Поскольку нет строгого юридического определения, нечистые на руку участники пытаются нарушить GPLv2, формально соответствуя ей.

- *Договорные ограничения поверх лицензии.* Патчи Grsecurity [5]: проект Grsecurity основан строго на исходных текстах ядра Linux (GPLv2) и не имеет смысла без него, но распространение кода и бинарных сборок ограничивается дополнительным договором подписки. Хотя это легально во многих юрисдикциях,

базовые свободы распространения и модификации СПО нарушаются.

Пути решения проблем

Похоже, хорошего решения нет:

- *Уточнение лицензий.* Более строгие и точные лицензии также будут намного больше и сложнее для понимания. Здесь могут помочь упрощённые пояснения и инструменты выбора лицензий для пользователей, как делают Creative Commons [6]. Но серьёзный подход всё равно потребует полного ознакомления с текстом лицензии.
- Развитие культуры отношения к лицензиям. На данный момент утопический сценарий, но было бы хорошо поднять культуру до уровня, когда затыкание лазеек было бы не нужным, потому ими никто не пользуется.

Литература

[1] Что такое свободная программа?
`https://www.gnu.org/philosophy/free-sw.ru.html`

[2] Что такое тивоизация? Как GPLv3 её предотвращает?
`https://www.gnu.org/licenses/gpl-faq.ru.html#Tivoization`

[3] Red Hat's "obfuscated" kernel source
`https://lwn.net/Articles/430098/`

[4] *Bradley M. Kuhn, Karen M. Sandle*
GPL Violations Related to Combining ZFS and Linux
`https://sfconservancy.org/blog/2016/feb/25/zfs-and-linux/`

[5] *Bruce Perens* Warning: Grsecurity: Potential contributory infringement and breach of contract risk for customers
`https://perens.com/2017/06/28/warning-grsecurity-potential-contributory-infringement-risk-for-customers/`

[6] Explore the Creative Commons licenses.
`https://creativecommons.org/choose/`

Татьяна Сергеевна Никифорова, магистр права (Оксфорд)
Санкт-Петербург, международная юридическая фирма Dentons
tatiana.nikiforova@dentons.com

Правовые риски свободных программ: чем юристы пугают ваших заказчиков

Аннотация

Выступление посвящено юридическим аспектам свободных программ. В начале доклада даётся краткая характеристика свободных программ в контексте российского законодательства, приводятся отдельные примеры свободных лицензий. Далее рассматриваются юридические риски, которые могут возникать у заказчиков (инвесторов) в связи с использованием свободных. На наглядных примерах разбираются вопросы, которые волнуют юристов заказчика и влияют на итоговый выбор заказчиком (инвестором) того или иного программного решения. В завершении даются рекомендации того, как повысить правовую защищённость своего продукта и сделать его более привлекательным для заказчика (инвестора) с позиции управления правовыми рисками.

Каждый ИТ-стартап рассчитывает рано или поздно привлечь солидного инвестора и вырасти в преуспевающий и стабильный ИТ-бизнес. Разумеется, главное на пути к этой цели — содержание самого ИТ-продукта, именно уникальный продукт позволит привлечь внимание к вашему проекту. Но простого внимания недостаточно. Чтобы привлечь инвестиции или получить заказ на разработку, нужно показать тот актив, в который инвестору предлагается вложить свои деньги. И в этот момент на передний план выходят юридические аспекты организации проекта. Ваш продукт, разработчики и методология разработки становятся объектом внимания юристов, задача которых выявить возможные правовые риски и доложить о них инвестору, чтобы уберечь его от необдуманных решений.

Картина, которую видит юрист, проводящий юридическую проверку (due diligence) стартапа, зачастую такова: договоры между разработчиками не заключались, создание программного продукта документально не оформлено. А тут ещё выясняется, что разработка на 80% состоит из оупенсорса... «Всё ясно, концов не найти!» — понимает юрист и выносит свой вердикт — «Высокий риск потери прав на объект интеллектуальной собственности».

С точки зрения методологии выявления правовых рисков, свободные программы не сильно отличаются от «несвободных». Однако, будучи под влиянием опасного заблуждения, что свободное ПО находится за пределами традиционного правового поля, разработчики свободных программ зачастую уделяют юридическим формальностям ещё меньше внимания, чем их коллеги из лагеря проприетарного ПО.

Небольшой экскурс в то, как смотрят на программные продукты юристы:

- Не бывает программного кода, который никому не принадлежит. Бывает программный код, автор которого неизвестен. Неизвестность — это риск.
- Программные продукты, в том числе свободные, охраняются авторским правом как литературные произведения, т.е. объектом правовой охраны является, прежде всего, программный код (текст).
- Свободной программу делает не то, что её можно бесплатно скачать в интернете, а тот факт, что программа распространяется на основании особого вида лицензионного договора — открытой лицензии.
- Поскольку открытая лицензия — это договор, то у него должны быть стороны — лицензиар (правообладатель, автор) и лицензиат (пользователь). Если вы не уверены в том, кто в действительности написал программу, тот факт, что она распространяется под открытой лицензией, не снимает риска незаконного использования объекта авторского права.
- Чтобы законно выпустить свой продукт под открытой лицензией, нужно подтвердить происхождение всех составляющих частей программного кода.

Как повысить юридическую, а значит, и инвестиционную, привлекательность своего проекта? Мой опыт работы в юридическом консалтинге показывает, что ваши шансы на положительное заключение юриста значительно возрастают, если вы можете внятно ответить на три следующих вопроса:

1. Кто авторы продукта?
2. Как подтверждается творческий вклад каждого автора?
3. Как юридически оформлены отношения между участниками проекта?

Из этих трёх вопросов самым проблемным оказывается, как правило, именно первый вопрос. Сложности возникают как в философском (кого вообще можно считать автором?), так и в сугубо организационном плане (разве упомнишь всех, кто когда-либо работал в проекте?). Тот факт, что продукт или какие-то его составные части распространяются по модели свободного ПО, не отменяют необходимости ответить на вопрос, кто является их автором.

После того, как определились с составом авторов, следует подумать о том, как можно подтвердить их авторство. В данном вопросе законодательное регулирование выглядит обманчиво простым: авторское право на программу возникает с момента её создания и не требует каких-либо обязательных формальностей (в отличие, например, от изобретений). Однако, не устанавливая обязательных требований к процедуре создания объектов авторского права, закон, тем не менее, не отменяет необходимости иметь такие доказательства в случае возникновения спора об авторстве. Традиционные доказательства, такие как служебные задания и отчёты работников, как правило, плохо приживаются в среде разработчиков. Однако при наличии в компании формализованной политики управления разработкой в качестве доказательства авторства могут использоваться отчёты из информационных систем, в которых ведётся трекинг создания продукта.

Далее нужно проверить, со всеми ли авторами есть договорные отношения. Это могут быть трудовые договоры, договоры оказания услуг или открытые лицензии, но они должны быть, т.к. только договор обеспечивает переход прав на программу от автора к иному лицу.

И если вы смогли подтвердить свои права на созданный программный продукт, представив убедительные ответы на три указанных выше вопроса, юрист будет уже почти готов дать инвестору своё положительное заключение. Останется только проверить, выполняются ли условия открытых лицензий и соответствуют ли проистекающие из этих лицензий ограничения бизнес задачам инвестора.

Дмитрий Левин
Москва, Базальт СПО

Проект: strace `https://strace.io`

Modern strace

Аннотация

strace — инструмент для отслеживания и влияния на взаимодействия пользовательских процессов и ядра Linux: системных вызовов, сигналов и изменений состояния процесса. Будучи традиционным инструментом, популярным среди разработчиков для диагностики, отладки и изучения поведения ПО, проект продолжает активно развиваться, и за минувшие несколько лет в strace реализованы новые полезные возможности.

Введение

strace как инструмент мониторинга взаимодействия пользовательских процессов с ядром существует уже почти 27 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.

Начиная с версии 4.13, выпущенной в июле 2016 года, расписание выпусков новых версий strace синхронизировано с расписанием выпусков новых версий ядра linux. Таким образом новые интерфейсы, добавляемые в релизы ядра linux, сопровождаются соответствующими парсерами, добавляемыми в релизы strace.

strace относится к традиционным инструментам, и пользователи со стажем зачастую не подозревают о новых возможностях, появившихся в strace за последние несколько лет, таких как

- отслеживание файловых путей: параметр `-y`;
- отслеживание сетевых протоколов: параметр `-yy`;
- отслеживание стека вызовов: параметр `-k`;
- сбор статистики по времени, проведённому отслеживаемыми процессами в системных вызовах: параметр `-w`;

- фильтрация системных вызовов по файловым путям: параметр `-P`;
- фильтрация системных вызовов по именам: с помощью регулярных выражений и новых классов системных вызовов;
- модификация системных вызовов: инъекции ошибок, кодов возврата, сигналов, и задержек.

Отслеживание файловых путей и фильтрация по файловым путям

Начиная с версии 4.7, выпущенной в мае 2012 года, реализована возможность отслеживания файловых путей, соответствующих дескрипторам, с помощью параметра `-y`. Например,

```
$ strace -y -e %fstat cat /dev/null > /dev/full
fstat(3</etc/ld.so.cache>, {st_mode=S_IFREG|0644, st_size=14318, ...}) = 0
fstat(3</lib64/libc-2.27.so>, {st_mode=S_IFREG|0755, st_size=1800216, ...}) = 0
fstat(1</dev/full>, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 7), ...}) = 0
fstat(3</dev/null>, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
+++ exited with 0 +++
```

В той же версии была реализована возможность фильтрации системных вызовов по файловым путям, соответствующим аргументам системных вызовов и файловым дескрипторам, с помощью параметра `-P`. Например,

```
$ strace -P /etc/ld.so.cache ls /var/empty
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=14318, ...}) = 0
mmap(NULL, 14318, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f21cc43c000
close(3) = 0
+++ exited with 0 +++
```

Отслеживание сетевых протоколов и устройств

Начиная с версии 4.10, выпущенной в марте 2015 года, реализована возможность отслеживания характеристик сетевых протоколов, соответствующих дескрипторам сокетов, с помощью параметра `-yy`. Начиная с версии 4.22, выпущенной в апреле этого года, с помощью этого же параметра можно отслеживать и характеристики устройств. Например,

```
$ netcat -l 127.0.0.1 12345 & strace -yy -e %network,%desc netcat 127.0.0.1 12345 < /dev/null
...
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3<TCP:[4100654321]>
connect(3<TCP:[4100654321]>, {sa_family=AF_INET, sin_port=htons(12345),
```

```
  sin_addr=inet_addr("127.0.0.1")}, 16) = 0
poll([{fd=3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>, events=POLLIN},
  {fd=0</dev/null<char 1:3>>, events=POLLIN}], 2, -1) = 1 ([{fd=0, revents=POLLIN}])
read(0</dev/null<char 1:3>>, "", 2048) = 0
shutdown(3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>, SHUT_WR) = 0
poll([{fd=3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>, events=POLLIN},
  {fd=-1}], 2, -1) = 1 ([{fd=3, revents=POLLIN|POLLHUP}])
read(3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>, "", 2048) = 0
shutdown(3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>, SHUT_RD)
  = -1 ENOTCONN (Transport endpoint is not connected)
close(3<TCP:[127.0.0.1:34567->127.0.0.1:12345]>) = 0
+++ exited with 0 +++
```

Отслеживание стека вызовов

Начиная с версии 4.9, выпущенной в августе 2014 года, реализована экспериментальная возможность отслеживания стека вызовов функций с помощью параметра `-k`. Начиная с версии 4.23, выпущенной в июне этого года, усовершенствованная реализация этой возможности с использованием библиотеки libdw уже не носит экспериментальный статус. Например,

```
$ strace -k -e rmdir rmdir /
rmdir("/") = -1 EBUSY (Device or resource busy)
 > /lib64/libc-2.27.so(rmdir+0x7) [0xe9fa7]
 > /bin/rmdir(main+0x138) [0x401748]
 > /lib64/libc-2.27.so(__libc_start_main+0xe6) [0x21bd6]
 > /bin/rmdir(_start+0x29) [0x401939]
rmdir: failed to remove '/': Device or resource busy
+++ exited with 1 +++
```

Сбор статистики

Начиная с версии 4.9, выпущенной в августе 2014 года, реализована возможность сбора статистики по времени, проведённому отслеживаемыми процессами в системных вызовах, с помощью параметра `-w`. Например,

```
$ strace -cw -e execve,nanosleep sleep 1
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.99    1.000041     1000040         1           nanosleep
  0.01    0.000124         123         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    1.000164                     2           total
```

Фильтрация системных вызовов по именам

Начиная с версии 4.17, выпущенной в мае 2017 года, синтаксис описания множества системных вызовов существенно расширился.

В описании имён системных вызовов теперь поддерживаются регулярные выражения:

`strace -e trace=/`*`regexp`*`.`

В описании множества системных вызовов поддерживаются описания, которым не соответствует ни одного системного вызова:

`strace -e trace=?`*`set`*`.`

Имена классов системных вызовов теперь начинаются с префикса `%`:

`strace -e trace=%`*`class`*`.`

Прежний способ именования классов без префикса `%` поддерживается, но уже считается устаревшим и не рекомендуется для применения.

Добавлены новые классы системных вызовов:

- `%stat` – `stat`, `stat64`, `oldstat` и их вариации;
- `%lstat` – `lstat`, `lstat64`, `oldlstat` и их вариации;
- `%fstat` – `fstat`, `fstat64`, `fstatat64`, `newfstatat`, `oldfstat` и их вариации;
- `%%stat` – `stat`, `lstat`, `fstat`, `fstatat`, `statx` и их вариации;
- `%statfs` – эквивалент `/^(.*_)?statv?fs`;
- `%fstatfs` – эквивалент `/fstatv?fs`;
- `%%statfs` – эквивалент `/statv?fs|fsstat|ustat`.

Модификация системных вызовов

Начиная с версии 4.15, выпущенной в декабре 2016 года, реализован механизм инъекций ошибок в системные вызовы. Синтаксис syscall fault injection выглядит следующим образом:

`strace -e fault=`*`set`*`[:error=`*`errno`*`][:when=`*`expr`*`]`

Начиная с версии 4.16, выпущенной в феврале 2017 года, реализован более общий механизм вмешательства в системные вызовы, который, помимо инъекций ошибок, позволяет осуществлять инъекции кодов возврата и сигналов, а начиная с версии 4.22, выпущенной в апреле 2018 года, ещё и инъекции задержек. Интерфейс этого нового механизма выглядит следующим образом:

`strace -e inject=`*`set`*`[parameters]`

где `parameters` могут принимать следующие значения:

- `:error=`*`errno`* либо `:retval=`*`value`* – инъекция ошибки либо кода возврата;
- `:syscall=`*`syscall`* – подменяемый системный вызов для инъекции ошибок либо кода возврата;
- `:signal=`*`sig`* – инъекция сигнала;
- `:delay_enter=`*`usecs`* – инъекция задержки перед системным вызовом;
- `:delay_exit=`*`usecs`* – инъекция задержки после системного вызова;
- `:when=`*`expr`* – правило применения инъекции.

Пример инъекции ошибки:

```
$ strace -e trace=/open -e inject=/open:when=3:error=EACCES cat /dev/null
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/dev/null", O_RDONLY) = -1 EACCES (Permission denied) (INJECTED)
cat: /dev/null: Permission denied
+++ exited with 1 +++
```

Пример инъекции задержки:

```
$ dd if=/dev/zero of=/dev/null bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.00211354 s,
5.0 GB/s

$ strace -e inject=write:delay_exit=100000 -e write -o /dev/null \
  dd if=/dev/zero of=/dev/null bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 1.10658 s,
9.5 MB/s
```

Литература

[1] *Дмитрий Левин*, strace: новые возможности // Четырнадцатая конференция разработчиков свободных программ. Калуга, 22-24 сентября 2017 года. Тезисы докладов. Москва, Базальт СПО, 2017, стр. 64–67.

[2] *Дмитрий Левин*, Can strace make you fail? // Тринадцатая конференция разработчиков свободных программ. Калуга, 1-2 октября 2016 года. Тезисы докладов. Москва, Базальт СПО, 2016, стр. 36–41.

Игорь Власенко
Киев, ALT Linux Team

Проект: autoimports

Стратегия развития систем автоматизации сопровождения пакетов

Аннотация

В докладе рассказывается о продолжающейся работе по созданию инфраструктуры полной цепочки автоматизированного сопровождения пакетов в дистрибутиве. Обсуждаются уже готовые решения, задачи в процессе разработки и потребности развития.

На первый взгляд развитие в Base ALT систем автоматизации сопровождения пакетов не несёт в себе никаких сюрпризов. Но на самом деле это не так. Последние несколько лет происходит рост невидимой глазу инфраструктуры, как картошки в огороде летом. И уже в ближайшие год - два придёт, как надеюсь, время «сбора урожая».

Как сложилась такая ситуация? Развитие систем автоматизации само по себе является подчинённой задачей для задачи сопровождения пакетов. Поэтому их разработка постоянно рискует попасть в самозатягивающуюся петлю: разработка улучшает производительность системы (удельные затраты времени на 1 пакет в новой системе меньше) — с её помощью можно собирать и сопровождать больше пакетов — замечательно! собираем всё большее число пакетов, пока большее число пакетов не потребует большего (в абсолютных величинах) времени на своё сопровождение и не останется времени на разработку.

Когда-то такая ситуация казалась мне неизбежной, а выходом казалось привлечение к средствам автоматизации других майнтайнеров для совместной работы над пакетами. К сожалению, экстенсивное развитие «не пошло» — Барьером послужили высокий порог вхождения и слабая распараллеливаемость задач в рамках одного проекта. Тем не менее некоторые проекты, такие, как perl, удалось «продавить», когда рывком число пакетов увеличивается на порядки, выходя на естественное «плато», связанное с исчерпаемостью источника пакетов, а время, затрачиваемое при этом на их сопровождение, наоборот, даже уменьшается.

Первая поспешная попытка распространить успех на модули python и изучение «историй успеха» показало, что эффект «продавливания» возникает при достаточно полной автоматизации всей цепочки производственного процесса. Отсюда и дальнейшая стратегия развития: временный отказ от количественной экспансии и вложение всего свободного времени в инфраструктуру полной цепочки автоматизации. Одними из последних звеньев в разработке являются проекты «Логовед» (представлен на этой конференции) и DistroMap (был представлен на конференции год назад). Сегодняшнюю ситуацию можно хорошо описать сравнением со стратегическими играми. Как в «Railroads Tycoon», сейчас все свободные средства вкладываются в прокладку длинных ж/д маршрутов, запуск которых должен разом окупить вложения.

Текущими целями являются модули python и проект Autoimports. Проект Autoimports занимается автоматизированным импортом пакетов из других дистрибутивов в ALT Linux и генерацией пакетов для упаковки библиотек скриптовых языков. В репозитории autoimports/Sisyphus сейчас 31.200 пакетов, расширяющих репозиторий ALT Linux Sisyphus.

Виталий Липатов, Дмитрий Державин
Санкт-Петербург, ООО «Базальт СПО»
https://www.basealt.ru/

Поддержка сторонних пакетов rpm в ОС Альт

Аннотация

Традиционно дистрибутивы операционных систем семейства Альт рассматриваются разработчиками в первую очередь как реализация программной платформы для совместных программных и программно-аппаратных решений. При этом сами разработчики ОС рекомендуют технологическим партнёрам оформлять свои программные продукты в виде пакетов rpm, собранных в соответствии с правилами, принятыми для пакетов ОС Альт, и с помощью нативных инструментов ОС Альт

К сожалению, не все разработчики с готовностью внедряют соответствующие технологии и соблюдают правила. Иногда в совместных проектах встречаются пакеты, разрушительно влияющие на другие компо-

ненты и систему в целом. Авторы доклада предлагают краткий обзор возможных подходов к решению этой проблемы.

Основной источник проблемы — умолчальное разрешение на использование произвольных сценариев в пакетах. База данных rpm позволяет отслеживать конфликты между пакетами на уровне файлов. Но, если с точки зрения разработчика стороннего по отношению к системе пакета какой-то файл, предоставленный другим пакетом, просто необходимо заменить — это легко можно сделать на уровне сценариев, входящих в состав пакета и автоматически выполняющихся при его установке. Конфликта при установке пакета не возникнет. При проверке соответствия дерева каталогов записям в базе данных rpm конфликт будет замечен, конечно же, но тогда будет уже поздно — пакет установлен и «всё работает». При этом для такого разработчика не вполне очевидно, что «работает» только его часть, а остальные компоненты совместного решения вполне могут пострадать.

К обсуждению предлагается три возможных подхода к решению проблемы: запретить выполнение сценариев при установке и удалении пакета, разрешить выполнение сценариев только для доверенных пакетов, отслеживать файловые операции в сценариях и считать их такими же модификациями дерева каталогов, как и установку файлов из пакета.

Полный запрет выполнения сценариев в пакетах с точки зрения обеспечения целостности системы выглядит многообещающе. Но для реализации этой идеи потребуется как-то формализовать и автоматизировать рутинные операции, для которых установочные сценарии сейчас используются ответственными разработчиками. В этом смысле понятно, что можно сделать, например, с добавлением пользователей и групп — добавить их в виде объектов в базу данных rpm. При этом появится дополнительное преимущество в виде возможности устанавливать соответствующие зависимости. С запуском и остановом служб, предоставляемых пакетом, направления поиска решения тоже понятны — можно попробовать так же интегрировать в базу данных rpm unit-ы системы инициализации systemd. Возможно, трудоёмкость этой задачи окупится преимуществами отказа от выполнения произвольных сценариев и новыми типами зависимостей. Но что делать, например, со сценариями миграции — вопрос более сложный. Действительно, существует ряд плохо формализуемых за-

дач, решение которых в случае отказа от выполнения сценариев потребует отдельной исследовательской работы.

Ещё один важный аспект запрета сценариев — реакция сообщества коммерческих компаний-разработчиков на такое изменение. Здесь важно иметь в виду не столько психологические моменты, связанные с новыми запретами, сколько неожиданные трудозатраты, необходимость которых может быть не очевидной. Поэтому, возможно, имеет смысл одновременно предлагать разработчикам новые для них технологии, позволяющие оптимизировать сборку и упаковку ПО для различных отечественных ОС. Здесь перспективными выглядят технологии, разработанные и успешно применяемые в компании Этерсофт — в частности, Коринф и epm, уже неоднократно обсуждавшиеся на этой конференции. Таким образом, перейдя на единый сборочный сценарий, разработчики смогут сэкономить ресурсы и одновременно сделать свои пакеты более пригодными для использования в составе совместных решений.

Второй из предлагаемых подходов к решению проблемы сценариев — разрешение на их выполнение только при установке доверенных пакетов. Например, подписанных участниками Alt Linux Team. Этот вариант хорош тем, что может стимулировать коммерческих разработчиков вствпать в Team и сопровождать там пакеты типа -preinstall, которые сейчас широко используются в качестве прослойки между дистрибутивными и недистрибутивными пакетами, предоставляя недостающие зависимости. Как свойственно полумерам, это «решение» по сути не решает проблему выполнения произвольных сценариев, а просто переносит её в другую плоскость. Очевидно, введение такого решения в качестве единственного может вызвать очередное «соревнование снаряда и брони» в виде расширения списка разрешений на уровне конкретных внедрений.

Третий подход, предлагаемый авторами к рассмотрению — интегрировать в rpm технологию, позволяющую полностью отслеживать изменения дерева каталогов, происходящие в процессе установки пакета, и трактовать их одинаково вне зависимости от того, чем они были вызваны — установкой файлов или работой сценариев. В качестве возможных направлений реализации можно рассмотреть файловые системы на базе оверлеев. Очевидно, что дополнительная проверка вызовет рост накладных расходов, но проблема выполнения произвольного кода с максимальными привилегиями настолько существенна, что, предположительно, эти расходы могут окупиться.

Авторы не предлагают при рассмотрении проблемы ограничиваться каким-то одним из предложенных подходов к решению. Возможно, имеет смысл применять их частично и комплексно. Также, выступление является приглашением к обсуждению проблемы как на конференции, так и за её рамками.

Олег Латий
Брест, респ. Беларусь, Брестский государственный технический университет
latijoo@tut.by

Высокопроизводительный модуль отрисовки графиков на базе использования библиотеки Qt

Аннотация

В докладе представлен авторский высокопроизводительный компонент библиотеки Qt для отрисовки 2D-графиков в реальном времени. Приводятся архитектура и особенности реализации, а также результаты сравнения с другими средствами построения графиков для платформы Qt. Код проекта доступен по адресу https://github.com/lattoo/plotter под лицензией GPL v.2.

Обзор существующих средств построения графиков в Qt

При необходимости реализовать функционал построения графиков Qt-разработчик сталкивается с выбором из нескольких вариантов, причём это одна из немногочисленных задач, где на сегодняшний день всё ещё сохраняется конкурентноспособность проприетарных Qt-компонентов. Так, к актуальным построителям графиков для Qt можно отнести следующие:

- QCustomPlot — виджет на C++ для построения графиков и визуализации данных, который фокусируется в первую очередь на создании визуально-привлекательных 2D-графиков, и при этом декларирует высокую производительность для визуализации в реальном времени [1].
- QChart — набор для отрисовки графиков от Qt (до релиза 5.6.0 библиотеки Qt платный) [2].
- Qwt — библиотека, содержащая дополнительные виджеты, компоненты и утилиты GUI, включая фреймворк для 2D-графиков [3].

- ChartDirector — профессиональный коммерческий компонент для построения графиков, известный широким охватом типов графиков [4].

Особенности программной реализации отрисовщика

Причиной разработки собственного инструмента построения графиков для Qt, который описывается в данной работе, появилась недостаточная производительность одних бесплатных фреймворков, которая делает их малопригодными для отрисовки динамично изменяющихся графиков в реальном времени, и сильно ограниченные визуальные возможности других.

Для написании виджета отрисовки графиков выбор между языком C++ и языком разметки QML был сделан в пользу C++ в силу требований максимальной производительности кода.

Программная реализация компонента Plotter представляет собой многоуровневую архитектуру, что позволяет настроить построитель под требования пользователя.

В общем случае структура классов принимает вид, изображённый на рисунке 1. Как видно из данной диаграммы, основные интерфейсные базовые классы включают IGraphicItem, IFilledItem, ITextItem, ITitleItem.

Подклассы, унаследованные непосредственно от IGraphicItem соответствуют различным видам линий: MainGridX и MainGridY для отрисовки главной сетки, SubGridX и SubGridY для отрисовки вторичной сетки, ZeroLineX и ZeroLineY для отрисовки нулевых линий, MainTicksX и MainTicksY для отрисовки штрихов, соответствующих главной сетке, SubTicksX и SubTicksY для отрисовки штрихов, соответствующих вторичной сетке и IFilledItem для отрисовки графических элементов с заливкой.

Список подклассов интерфейса IFilledItem включает: Background для заливки внешней области виджета, PlotArea для заливки области построения, а также класс IChart, предоставляющий возможность заливки области под графиком.

Подклассы, унаследованные от ITextItem отвечают за отрисовку текстовых полей и включают AxisLabelX и AxisLabelY для отображения подписей осей, AxisTitleX и AxisTitleY для отображения названия осей, LegendText для отображения легенды графиков, Title для отображения названия графиков.

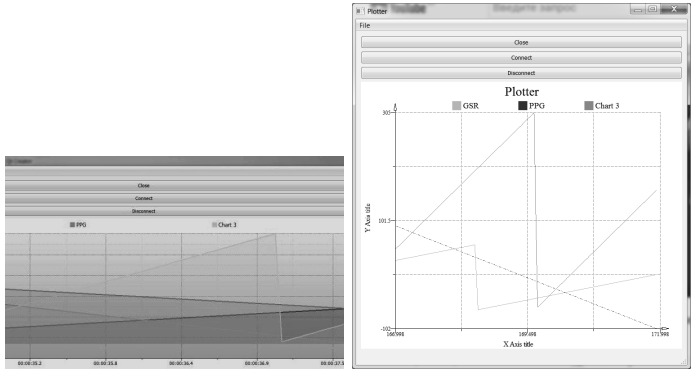


Рис. 1: Диаграмма основных классов и внешний вид отрисовщика

Также стоит упомянуть, что благодаря многоуровневой архитектуре расчёты для отрисовки графических элементов отделены от непосредственной отрисовки, а в качестве дополнительных бонусов реализован экспорт графиков в формат масштабируемой векторной графики SVG, и опциональное хранение пакета настроек графика в конфигурационном файле.

Сравенение производительности

Очевидно, что возможности построения и оформления графиков у всех рассмотренных компонент различаются. Таблицы 1 и 3 показывают результаты сравнения, полученные с помощью равноценных по визуальному оформлению и структуре кода тестовых программ, использующих компоненты в минимальной конфигурации: одновременная отрисовка трёх кривых со сплошными линиями толщиной 1 пиксель, с легендой и заголовком, без каких-либо заливок областей построения. По условиям тестирования был выставлен минимальный интервал обновления данных и отрисовки графиков, равный 50мс. Тестирование выполнялось на системе с мобильной версией процессора Intel Celeron B820 частотой 1,7ГГц с операционными системами Ubuntu Linux 17.10.1 x64 и Windows 7 x32.

Как можно видеть из таблицы 1 в сборках отладки и выпуска относительно загрузки ЦП, лидирующую позицию занимает виджет QCustomPlot.

Таблица 1: Сравнительная характеристика инструментов построения графиков (операционная система Windows 7 x32)

Отрисовщик	Сборка	Загрузка процессора, %	Объём занимаемой оперативной памяти, МБ
QCustomPlot	Отладка	0	25
	Выпуск	0	18
QCharts	Отладка	17	22
	Выпуск	12	12,7
Plotter	Отладка	12	22
	Выпуск	0	15,5

Таблица 2: Сравнительная характеристика инструментов построения графиков при использовании средств дополнительного визуального оформления (операционная система Windows 7 x32)

Отрисовщик	Сборка	Загрузка процессора, %	Объём занимаемой оперативной памяти, МБ
QCustomPlot	Отладка	16	25
	Выпуск	14	18
Plotter	Отладка	8	22
	Выпуск	0	15

По аналогии дополнительно было проведено сравнение производительности отрисовщиков графиков при использовании дополнительных средств визуального оформления: заданные толщину и стиль линии, градиентная заливка области построения, градиентная заливка области заднего плана. Результаты сравнения представлены в таблицах 3 и 4. Можно заметить, что лидирующую позицию занимает представляемый модуль Plotter.

Пример использования

В листинге ниже приведён минимальный код для подключения, инициализации, добавления данных и отрисовки графика.

```
1  Plotter* plotter = new Plotter;
```

Таблица 3: Сравнительная характеристика инструментов построения графиков (операционная система Ubuntu 17.10.1 x64)

Отрисовщик	Сборка	Загрузка процессора, %	Объём занимаемой оперативной памяти, МБ
QCustomPlot	Отладка	10	16
	Выпуск	8	15,7
QCharts	Отладка	18	11
	Выпуск	19	11
Plotter	Отладка	10	15,8
	Выпуск	10	15,7

Таблица 4: Сравнительная характеристика инструментов построения графиков при использовании средств дополнительного визуального оформления (операционная система Ubuntu 17.10.1 x64)

Отрисовщик	Сборка	Загрузка процессора, %	Объём занимаемой оперативной памяти, МБ
QCustomPlot	Отладка	37	15,8
	Выпуск	36	15,8
Plotter	Отладка	9	18,3
	Выпуск	8	15,5

```
2  const QString chart_name = "Chart 1";
3  plotter->add_chart(chart_name);
4  const QPen pen = QPen(QColor(220,170,170), 1, Qt::SolidLine);
5  plotter->chart(chart_name)->set_pen(pen);
6  plotter->chart(chart_name)->add_data(x, y);
7  plotter->scroll_graph();
8  plotter->replot();
```

Строка 3 отвечает за добавление графика на область построения.
Строка 5 отвечает за инициализацию пера отрисовки.
Строка 6 отвечает за добавление данных.
Строка 7 необходима для прокрутки области построения графика.
Строка 8 инициирует общее обновление для области построения.

Общий вид графика представлен на рисунке 1 справа.

Для работы с настройками отрисовщика был разработан дополнительный модуль, реализованный по шаблону проектирования MVP (model-view-presenter) [5]. Данный подход позволяет улучшить разделение ответственности в презентационной логике, а именно: модель содержит всю бизнес-логику, представление реализует отображение данных, представитель реализует двустороннее взаимодействие между моделью и представлением. Также использование данного подхода реализации связи интерфейса с данными позволяет увеличить уровень портируемости модуля.

Модуль представляет собой набор написанных и скомбинированных Qt-виджетов, позволяющий довольно быстро и удобно отредактировать вышеупомянутые настройки отрисовщика.

Литература

[1] Qt Plotting Widget QCustomPlot — Introduction `www.qcustomplot.com/`

[2] QChart Class `https://doc.qt.io/qt-5.10/qchart.htm`

[3] Qwt User's Guide: Qwt — Qt Widgets for Technical Applications `qwt.sourceforge.net/`

[4] ChartDirector Chart Component and Control Library `www.advsofteng.com/`

[5] Model-View-Presenter — Википедия — `https://ru.wikipedia.org/wiki/Model-View-Presenter`

Денис Пынькин
Минск, Collabora Ltd.

Проект: Debos `https://github.com/go-debos/debos`

Debos: утилита для создания ОС на основе Debian

Аннотация

Утилита Debos предназначена для создания кастомизированных версий операционных систем с использованием пакетной базы ОС Debian. Debos разработан в качестве инструмента для максимального упрощения стандартных задач, возникающих при модификациях ОС, оставляя при этом достаточно возможностей для реализации любого нетривиального этапа процесса сборки.

Утилита `Debos` (`https://github.com/go-debos/debos`) создана Sjoerd Simons в качестве альтернативы существующим системам подготовки образов дисков на базе дистрибутива Debian, с прицелом на встраиваемые системы. Основная задача, которую решает `Debos` — максимальное упрощение для конечных пользователей описания создания образов систем, готовых к исполльзованию на целевых устройствах.

Для реализации основной задачи используются несколько принципов, заложенных в архитектуру проекта:

- вся информация о сборке должна находится в конфигурационном файле или «рецепте» (recipe) проекта;
- строго последовательное выполнение действий при создании образа;
- для каждого отдельного действия (action) создаётся свой модуль;
- действия должны быть самодостаточными и, в идеале, не связанными друг с другом;
- каждое действие должно быть простым — для сложных задач декларируется использование пользовательких скриптов и/или внешних программ.

Стандартная проблема, которая возникает при сборке образа — это необходимость использования повышенных привилегий для некоторых шагов, таких как установка пакетов. Разными утилитами

и дистрибутивами эта задача решается по-разному. Для Debos используется библиотека fakemachine https://github.com/go-debos/fakemachine, написанная Sjoerd Simons. Эта библиотека использует виртуальную машину Qemu, позволяя работать с повышенными привилегиями в текущей системе. Кроме того, такой подход позволяет без дополнительных затрат организовать сборку образа под любую архитектуру, поддерживаемую в Qemu.

В задачу утилиты не входит создание повторяемого сборочного окружения. Подразумевается, что для каждого проекта такое окружение уникально и должно создаваться другими средствами, например, Docker.

Синтаксис для recipe

Конфигурационный файл для создания образа представляет собой файл, описывающий пошаговое выполнение действий, в формате YAML. Для расширения возможностей, при описании конфигурации используется текстовый шаблонизатор (https://golang.org/pkg/text/template).

Рецепт, условно, можно разделить на 2 части:

- Заголовок

```
# Declare variable 'Var'
{{ $Var := "Value" }}
# Header
architecture: arm64
```

- Список действий для пошагового выполнения

```
actions:
  - action: ActionName1
    property1: true
  - action: ActionName2
    property2: {{$Var}}
```

Для комментирования используется символ «#».

На данный момент поддерживаются следующие базовые действия, которые можно использовать для создания образа системы:

- apt — установка пакетов в целевое окружение с помощью apt
- debootstrap — создание базового окружения с помощью утилиты debootstrap

- download — скачивание файла и распаковка, если это необходимо
- filesystem-deploy — «перенос» подготовленного окружения на образ диска
- image-partition — создание файла-образа диска, разметка и создание разделов
- ostree-commit — фиксирует ревизию («коммитит») в репозиторий в формате ostree
- ostree-deploy — использует версию из репозитория в формате ostree для подготовки целевого окружения
- overlay — позволяет копировать файлы и директории со сборочной системы в целевое окружение
- pack — создаёт архив с целевым окружением, как правило для использования в других сценариях или с помощью `chroot` либо аналогов
- raw — запись файла напрямую в образ диска или раздела по указанному смещению — чаще всего используется для загрузчиков
- run — запуск пользовательского скрипта или команды. Можно запускать как в сборочном, так и в целевом окружении
- unpack — распаковать архив в целевом окружении

Постоянно обновляемая документация по существующим «действиям» (actions) доступна по адресу `https://godoc.org/github.com/go-debos/debos/actions`, а пример для создания загружаемого образа для Raspberry Pi 3 — `https://github.com/go-debos/debos-recipes`.

Реальное применение

На данный момент утилита `Debos` уже используется в проекте «Apertis» (`https://apertis.org`) для подготовки загружаемых образов дисков и `rootfs` (корневых файловых систем). При этом поддерживаются как «классические», так и `ostree`-based системы для архитектур x86_64, armhf, arm64, в том числе и для контейнеров LXC. По мере расширения использования и вовлечения в проект новых людей,

находятся и закрываются ошибки, а также появляются новые идеи для новых действий (actions).

Игорь Власенко
Киев, ALT Linux Team

Проект: Логовед

Система Логовед для автоматизации QA

Аннотация

Система «Логовед» предназначена для разбора логов сборки и FTBS логов beehive и сбора статистики, автоматизации починки пакетов по информации из лога неудавшейся сборки, и автоматизации ряда других работ QA. На базе заложенных в него принципов разработано новое современное и универсальное решение, интегрированное с другими средствами автоматизации и предназначенное для решения широкого круга задач.

Система «Логовед» предназначена для автоматизации починки пакетов, разбора логов сборки и автоматизации работ по QA. Прототипом системы «Логовед» послужили скрипты разборки и обработки логов и починки пакетов для системы автоматизации сборки библиотек perl «cpanbuilder», которые частично использовались и с инструментами поддержки библиотек perl в Сизифе «watchscripts/perl». На их плечах уже почти 5 лет поддержка более 26 тысяч пакетов perl-* в репозитории autoimports.

Сердцем системы «Логовед» является «Logoved DB», база знаний сообщений об ошибках и предупреждений и библиотека для работы с ней. Сообщения об ошибках и предупреждения в простейшем случае хранятся как регулярное выражение, но может быть и набор регулярных выражений, и подпрограмма, и внешний скрипт-фильтр. В базе хранятся не только различные сообщения об ошибках но и во многих случаях действия, которые нужно предпринять для исправления ошибки (к примеру, для редактирования srpm пакета – ссылка на скрипт для библиотеки RPM::Source::Editor). Поэтому «Logoved DB» и называется базой знаний.

«Logoved DB» может быть использован и напрямую, например, в системе тестирования приложений путём их установки и запуска в виртуальной машине, для распознавания их сообщений в консоли.

Однако наиболее важно применение этой системы для обработки логов сборки пакетов (и пересборки в beehive), как удачных (для поиска предупреждений и мелких ошибок) так и неудачных логов, для помощи в починке пакетов. Отсюда и название системы, Логовед.

Для обработки логов сборки/пересборки пакетов для корректного применения к ним данных из «Logoved DB» мало найти соответствующее сообщение, необходимо знать его контекст. В процессе сборки в зависимости от этапа сборки одно и то же сообщение может быть признаком build error, warning, или же вообще должно быть проигнорировано как, к примеру, сообщение «Package gtk+-2.0 was not found in the pkg-config search path» на этапе выяснения BuildRequires пакета в hasher для подготовки chroot.

Для уточнения контекста сообщения «Logoved DB» нужен посредник, парсер логов hasher. Для «Logoved DB» таким парсером выступает подсистема «Logoved-Stream» (Доступна в виде отдельной независимой библиотеки). «Logoved-Stream» позволяет разбить лог сборки на отдельные секции, а также определять, успешно ли завершилась конкретная секция.

Текущими задачами для Логовед являются:

1. Разбор FTBS логов пересборки репозитория (Sisyphus, branch, autoimports) и генерация отчёта со статистикой.
2. Починка не(пере)собирающихся пакетов в распознанных случаях (где есть обработчик).
3. Сбор предупреждений из логов собравшихся пакетов для вывода в гeросор, в некоторых случаях — починка таких пакетов.

Дмитрий Белявский
Москва, ООО «Криптоком»

Проект: OpenSSL, OpenVPN, XMLSec и все-все-все
https://github.com/gost-engine/engine

Российская криптография в свободном ПО

Аннотация

Поддержка российской криптографии в свободном ПО становится всё более актуальной в связи с текущими политическими тенденциями. Доклад рассказывает об основных вариантах доработок свободного ПО, использующего OpenSSL, для работы с российской криптографией.

Компания «Криптоком»[1] последовательно занимается внедрением поддержки российской криптографии в Open Source продукты в течение последних 15 лет. За это время накоплен опыт доработок свободного ПО для поддержки криптографии по ГОСТ.

Первое, что мы сделали — приняли решение, что поддержка российской криптографии будет встроена в OpenSSL. Это один из самых популярных инструментов для работы с разного рода алгоритмами и протоколами. Поддержка была добавлена в линейку версий 1.0, которая остаётся актуальной до конца 2019 года. Она включала в себя как модификацию архитектуры, так и свободную реализацию алгоритмов по ГОСТ. Впоследствии engine был выделен в отдельный проект и сейчас он развивается отдельно, хотя и параллельно. Многие более мелкие патчи, добавляющие российские алгоритмы, расширения и т.п. в закрытые списки, также включены в upstream.

Для большинства приложений, использующих OpenSSL, достаточно обеспечить загрузку engine. До ветки 1.1.* это делалось вызовом функции OPENSSL_config(). Сейчас актуальный способ требует вызова функции OpenSSL_add_all_algorithms() и сборки -DOPENSSL_LOAD_CONF.

Авторы некоторых приложений, особенно сравнительно старых, содержат прибитый гвоздями в лучшем случае список алгоритмов, а в худшем — только RSA. Если такие приложения удаётся перевести на EVP-интерфейс — универсальный для OpenSSL последних версий —

[1] http://www.cryptocom.ru/

то годятся предыдущие советы. Если нет, то может оказаться проще этот интерфейс хотя бы частично дописать.

В случае с библиотекой XMLSec пришлось полностью отклонировать структуры callback-функций и наполнить их ГОСТ-овым содержанием с опорой на уже упомянутый EVP интерфейс.

Для приложений, использующих нестандартные протоколы, такие как OpenVPN, недостаточно механически заменить криптопримитивы. Так, для OpenVPN при работе с поддержкой ГОСТ используется содержательно другая технология имитозащиты (имитовставка по ГОСТ существенно отличается от HMAC) и PRF (Pseudo Random Function) с использованием ГОСТ. В этом случае требуется содержательный анализ кода.

Костарев А. Ф., Братчиков И. А.
Пермь, ООО «НЕВОД», ООО «Новая Платформа»

Проект: Платформа Flexberry, Сокол-Аналитика
`https://hub.docker.com/u/flexberry/`

Организация процесса DevOps на платформе контейнеризации docker

Аннотация

В докладе рассматриваются вопросы использования технологии контейнеризации docker, кластеризации docker swarm, docker-образов на основе дистрибутива ALT Linux для построения полноценного решения с поддержкой практик DevOps

Одна из важных задач стоящая перед разработчиком конечных решений для Заказчиков — выбор подходящего дистрибутива Linux. Какой бы дистрибутив не был выбран, как правило, вскоре для тестовых или иных целей встаёт задача разворачивания в нём программного обеспечения, отсутствующего в выбранном дистрибутиве. Портирование стороннего ПО, как правило, трудоёмкий процесс даже при наличии исходных кодов.

Большие проблемы также вызывает разработка и тестирование различных версий разрабатываемого и используемого программного обеспечения, перенос новых версий Заказчику и перевод функционирующих систем на новые внедряемые версии.

Одним из решений данных проблем является использование технологии контейнеризации *docker* и запуска сервисов в кластерном окружении *docker-swarm*. Это обеспечивает:

- поддержку при разработке, внедрении и эксплуатации различных версий программного обеспечения;
- использование микросервисной архитектуры при создании и интеграции системы;
- оперативную интеграцию сервисного и клиентского ПО различных Linux-дистрибутивов;
- запуск и тестирование конечного решения в различных конфигурациях: моносервер, кластер в рамках одной сети, кластер в рамках нескольких сетей включая облачные сервисы;
- простоту конфигурирования и развёртывания кластерного решения при использовании технологии *docker swarm*.

В докладе описывается опыт использования технологии docker для поддержки полного цикла разработки программного обеспечения.

Ввиду подверждённой надёжности и удобства использования в реальных проектах для построения docker-образов был выбран docker-образ дистрибутива *Альт на платформе p8*. Данный дистрибутив входит в *Единый реестр российских программ для электронных вычислительных машин и баз данных*.

Технология docker поддерживает мощный механизм наследования образов. Он позволяет значительно сократить ресурсы, необходимые для создания *docker-решений*: дисковую и оперативную память, объём передаваемых слоёв образов по сети.

На рисунке 1 приведено дерево основных используемых docker-образов.

Корневым для большинства дочерних образов является образ *flexberry/alt.p8*[1], содержащий минимальный набор команд и библиотек необходимых для запуска и отладки серверных приложений.

Сложные информационные системы строятся на различных стеках технологий. Технология docker за счёт микросервисной архитектуры позволяет легко интегрировать ПО, созданное в различных языках программирования и стеках технологий, например, в одной системе можно соединить *Java*, *.NET* и *PHP*-решения. В частности, в наш

[1] `https://hub.docker.com/r/flexberry/alt.p8/`

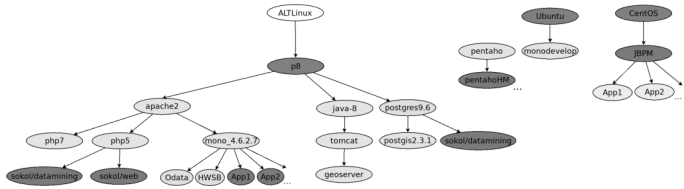


Рис. 1: Основное дерево docker-образов используемых в проектах

стек технологий входит программное обеспечение (ПО) *платформы Flexberry*, написанное на языке *C#*.

Данное ПО функционирует в дочернем docker-образе (*flexberry/alt.p8-apache2-mono*) в состав которого включён Web-сервер Apache2 и пакет mono4. Этот образ становится базовым для прикладного решения, работающего на *mono4*.

WEB-приложения пользователей создаются на основе программного обеспечения с открытым исходным кодом *Flexberry Ember* (`https://github.com/Flexberry/ember-flexberry`) и оформляются в виде отдельных docker-образов в состав которых наряду с кодом Ember-приложения включается *OData*-сервис, обеспечивающий *REST*-интерфейс для общения приложения с базой данных *PostgreSQL*.

База данных также функционирует в виде docker-контейнера образа *flexberry/alt.p8-postgresql*[2].

В рамках разрабатываемых проектов используются официальные или дочерние от них образы различных сервисов (Pentaho, Alfresco, jBPM, MongoDB, и др.) репозитория *hub.docker.com*. По мере возможности эти сервисы путём сборки из исходных кодов или переноса бинарных кодов и файлов конфигурации с использованием механизма сборки «*multistage build*» переносятся в дочерние образы собираемые на основе *flexberry/alt.p8*.

Интеграция сервисов производится на основе кластерной системы *docker-swarm*, обеспечивающей:

- распределённый запуск серверных приложений в кластере;
- минимизацию работ по настройке файлов конфигурации системы за счёт использования внутреннего *DNS*;

[2] `https://hub.docker.com/r/flexberry/alt.p8-postgresql/`

- дополнительный уровень защиты за счёт организации внутренней сети и закрытию прямого доступа к внутренним портам (база данных, *OData*-запросы, и т.д.) проекта;
- репликацию части сервисов с балансировкой нагрузки;
- отказоустойчивость за счёт перезапуска приложений на узлах кластера при выходе из строя одного из узлов.

Разрабатываемые docker-образы по конкретному проекту являются дочерними к вышеперечисленным образам и часто содержат закрытую информацию что исключает их размещение на общедоступных ресурсах (в частности *hub.docker.com*). В связи с этим для работы с ними используются внутренние регистраторы *docker-образов*. Все разрабатываемые образы проекта с префиксом *dh.flexberry.ru/* хранятся на внутреннем репозитории docker-образов *dh.flexberry.ru* и передаются Заказчику в рамках локальной сети которого также создаётся внутренний репозиторий docker-образов домена *dh.flexberry.ru*.

К сожалению, в настоящий момент невозможно использование docker-технологии в сертифицированных решениях. Необходимо «приземлять» docker-контейнеры (запускать сервисы контейнеров непосредственно в HOST-системе). Использование единого корневого образа *flexberry/alt.p8* упрощает данный процесс , так как механизм установки, компиляции или запуска сторонних бинарных кодов отлаживается на этапе разработки проекта в рамках docker-образа.

Все исходные коды проекта включая коды для сборки образов (*Dockerfile*, файлы конфигурации, вспомогательные файлы, и т.д.) ведутся с использованием git-репозитория (см. рис. 2). Это позволяет поддерживать технологию *DevOps* с версионированием исходных кодов и docker-образов и возможностью поддержки различных версий программного обеспечения и *docker-образов* с гибким механизмом переключения между различными версиями как в процессе разработки, так и в процессе функционирования проектов у Заказчиков.

Кроме этого *docker-swarm* технология позволяет использовать облачные платформы *Windows Azure*, *Amazon* и другие для включения в состав сервисов проекта приложений, запущенных в этих облачных платформах.

Использование данного подхода позволило обеспечить поддержку основных циклов *DevOps*: планирование, программирование, компиляции, тестирования, формирования версий и релизов, разворачива-

ние решения у Заказчика, функционирования и мониторинга разрабатываемых систем.

В итоге это позволяет значительно снизить накладные расходы и ресурсы, необходимые для поддержки полного цикла разработки решений.

Все открытые родительские docker-образы платформы Flexberry располагаются в репозитории `https://hub.docker.com/u/flexberry/`.

Документацию по созданию docker-образов и разворачиванию на их основе решений можно найти на сайте `https://docs.docker.com/`.

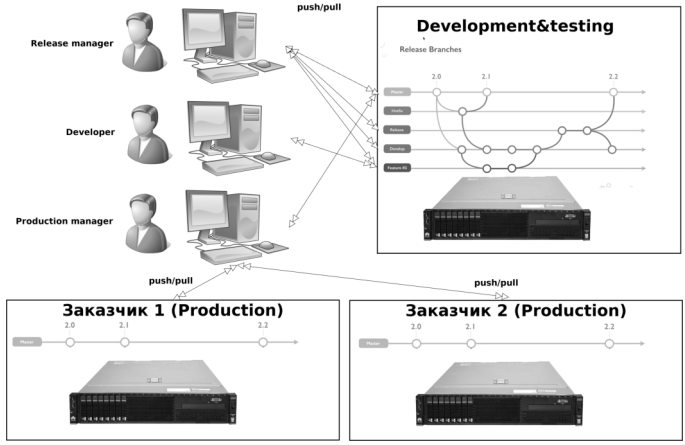


Рис. 2: Поддержка технологии DevOps за счёт использования git и docker-репозиториев у Разработчика и Заказчика

Литература

[1] *Usman Ismail, Bilal Sheikh*, Continuous Integration and Deployment with Docker and Rancher. January 2016

[2] The New Stack: The Docker and Container Ecosystem, eBook SeriesThe New Stack. AUTOMATION & ORCHESTRATION WITH DOCKER & CONTAINERS

[3] Технологическая программная платформа Flexberry для профессиональной разработки программного обеспечения `http://flexberry.ru/`.

Леонид Кантер
Донецк, ДНР, Cloud Linux Inc
https://www.cloudlinux.com

Построение корпоративной системы управления идентификационной информацией на базе FreeIPA

Аннотация

Доклад посвящён опыту построения и внедрения корпоративной системы управления идентификационной информацией на базе свободных продуктов FreeIPA и Ipsilon с поддержкой единого входа в систему (SSO) по протоколу SAML для сторонних поставщиков услуг.

Постановка задачи

В процессе развития любой компании неизбежно возникает момент, когда администрирование пользователей становится невозможным без единой корпоративной системы управления идентификационной информацией сотрудников, которая с одной стороны хранила бы организационную информацию о пользователях, такую как имя, контакты, отдел, должность; с другой стороны, позволяла бы управлять политиками доступа к ресурсам. К этому моменту мы уже пользовались пакетом G Suite, в котором были заведены все пользователи, кроме этого, необходимо было предоставлять доступ к OpenNebula, сборочной системе, веб-серверам, серверам БД и другим корпоративным ресурсам, поскольку пользователи на всех этих ресурсах добавлялись индивидуально. В случае увольнения сотрудника необходимость проверки наличия учётных записей пользователя на всех этих ресурсах вызывала большую нагрузку на IT-отдел.

Выбор FreeIPA

Поскольку у нас уже была готовая база сотрудников в G Suite, звучали предложения использовать именно его в качестве первичного источника идентификационной информации. И действительно, вход на некоторые ресурсы уже был организован с использованием Google OAuth2. Но организовать таким образом доступ на системы Linux по

ssh в консольном окружении не представлялось возможным. Поэтому необходима была такая система, которая позволяла бы управлять политикой консольного доступа по ssh и предоставлять идентификационную информацию для G Suite одновременно. Основными аргументами в пользу FreeIPA стали:

- Появление в CentOS 7.2 поставщика идентификации (IdP) в виде пакета Ipsilon 1.0.0, работающего с FreeIPA;
- Появление на стороне пакета приложений от Google поддержки единого входа в систему (SSO) по протоколу SAML с использованием сторонних IdP.

Возможности FreeIPA

FreeIPA (от **Free** **I**dentity, **P**olicy, **A**udit) — открытый проект для создания централизованной системы масштаба предприятия по управлению идентификацией пользователей, задания политик доступа и аудита сетей. Поддерживается компанией Red Hat. Используемые цели и механизмы схожи с Microsoft Active Directory. Включает в себя следующие компоненты: 389 Directory Server в качестве сервера LDAP; MIT Kerberos 5.0 для аутентификации и единого входа; DogTag для управления сертификатами; NTP; BIND и DHCP для управления DNS; WEB-интерфейс управления. Начиная с версии 3.0.0, FreeIPA также использует Samba для интеграции с Active Directory от Microsoft путём доверительных отношений.

Ipsilon

Ipsilon представляет собой поставщик идентификационной информации (Identity Provider, IdP), поддерживающий протоколы SAML2, OpenID Connect, OAuth2, Persona, OpenID 2.0, а также набор инструментов, позволяющий настраивать поставщики услуг на базе веб-сервера Apache. Представляет собой приложение, работающее на mod_wsgi. Аутентификация пользователей выполняется на отдельно стоящем сервере FreeIPA.

Внедрение

После испытаний на тестовом домене G Suite были установлены основной сервер FreeIPA и две реплики для обеспечения отказоустой-

чивости. Подготовлены инструкции для пользователей по использованию Kerberos для консольного и графического входа. На IPA были заведены рабочие группы. Из основного домена G Suite был экспортирован список пользователей. Был написан скрипт, который для полученного списка пользователей выполнил следующие действия: добавление пользователя в IPA; генерация начального пароля; отправка начального пароля и инструкции по смене пароля пользователю. После получения подтверждения от всех пользователей о возможности входа на веб-страницу Ipsilon домен G Suite был перенастроен на использование SSO. После этого началась работа по интеграции существующих и новых сервисов.
В настоящее время с FreeIPA интегрированы следующие внешние и внутренние сервисы:

- G Suite — SAML-авторизация через Ipsilon
- Корпоративные веб-сайты на Apache — ipsilon-client (mod_auth_mellon)
- Консольный доступ на Linux-системы — ipa-client (Kerberos/sssd)
- OpenNebula — LDAP
- Jupyterhub — локальная pam-авторизация через sssd
- OpenVPN — локальная pam-авторизация через sssd
- Сборочная система — библиотека python-saml на уровне приложения

В обозримое время планируется интеграция таких внутренних сервисов, как Jenkins и Sentry.

Ссылки

1. `https://freeipa.org`
2. `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/linux_domain_identity_authentication_and_policy_guide/index`
3. `https://ipsilon-project.org/`
4. `https://fedoraproject.org/wiki/Features/SSSD`

Денис Силаков
Москва, Virtuozzo

Проект: OpenVZ `https://openvz.org/`

Шаблоны контейнеров в OpenVZ 7

Аннотация

Пользователи OpenVZ активно пользуются шаблонами контейнеров для создания новых виртуальных окружений. В "legacy" OpenVZ (с ядром 2.6.x) шаблон представляет собой архив с готовым образом файловой системы, который можно напрямую использовать для новых контейнеров. В репозиториях OpenVZ содержится несколько десятков как официальных шаблонов, так и шаблонов от сообщества. Официальные шаблоны для OpenVZ создавались не копированием образа ФС заранее подготовленного контейнера, а специальными инструментальными средствами, входящими в коммерческие версии Virtuozzo. Возможности этих инструментов существенно шире, чем одноразовая подготовка образа ФС для контейнера. Начиная с OpenVZ 7, код этих утилит открыт и администраторы могут использоваться ими не просто для создания контейнеров, но и для их сопровождения на протяжении всего жизненного цикла.

Виды

В OpenVZ 7 есть шаблоны двух типов — ОС и приложений. Первые используются для создания контейнеров с заданным дистрибутивом, вторые — для установки, удаления и обновления конкретного приложения или стека программ в контейнере, где уже установлена ОС. Каждый шаблон приложения рассчитан на использование в контейнере, созданном из шаблона определённой ОС — можно создать контейнер с CentOS 6 на основе шаблона ОС CentOS 6, а затем добавить туда MySQL с помощью шаблона приложения MySQL для CentOS 6.

Шаблоны ОС разделяются на базовый (устанавливающий ОС в минимальном варианте) и кастомизированные, отличающиеся от базового настройками или набором предустановленных программ.

Согласно правилам именования, базовый шаблон содержит имя, версию и архитектуру ОС (например, «centos-6-x86_64»), кастомизированные содержат некоторый дополнительный суффикс («centos-6-x86_64-server»), а имя шаблона приложения содержит название при-

ложения и название базового шаблона целевой ОС («mysql-centos-6-x86_64»).

Структура

Шаблон в OpenVZ 7 -это набор конфигурационных файлов, на основе которых готовится образ образ ФС. Эти файлы вместе с закэшированными образами ФС хранятся в директории /vz/template. Структура поддиректорий соответствует названию ОС шаблона — например, «centos/6/x86_64». Конфигурация базового шаблона ОС хранится в файле config/os/default, кастомизированных — в файлах config/os/<суффикс>. Файлы шаблонов приложений для данной ОС — в config/app/<имя_приложения>/default. Имена файлов самоговорящие — packages, package_manager, repositories, summary и так далее.

Шаблоны могут содержать скрипты, выполняемые на определённой стадии их развёртывания — перед либо сразу после завершения одного из шагов cache, install, upgrade и remove. Скрипт, выполняемый перед тем или иным шагом, имеет префикс «pre-» (например, `pre-install`), а после завершения шага — префикс «post-».

Утилиты

Управление шаблонами осуществляется с помощью утилиты `vzpkg`.

Для установки шаблона ОС необходимо дать ей команду `install template` в OpenVZ 7 эта команда сводится к установке RPM-пакета с файлами соответствующего шаблона. После установки шаблона ОС, желательно сразу создать кэш входящих в него пакетов и образ диска с установленными пакетами командой `vzpkg create cache`.

Имя шаблона ОС можно передать стандартным утилитам при создании контейнера:

```
# prlctl create c1 --vmtype=ct --ostemplate=centos-6-x86_64
```

Установить шаблон приложения в работающий контейнер можно с помощью `vzpkg`:

```
# vzpkg install c1 mysql
```

Утилита vzpkg может также обновлять внутри работающих контейнеров пакеты, относящиеся к тому или иному шаблону, обновлять

кэшированные образы и производить много других полезных действий — см. `man vzpkg`.

Механизм работы

При создании кэша для шаблона ОС, разворачивается заранее подготовленное минимальное chroot-окружение с «родным» для целевой системы пакетным менеджером, модифицированным для целей vztt. С помощью этого менеджера осуществляется установка необходимых пакетов (перечисленных в os-packages) из репозиториев, указанных в `os_repositories`.

Chroot-окружения и модифицированные пакетные менеджеры входят в пакеты vzpkgenv*. Обновляются пакеты vzpkgenv достаточно редко — только если в новых версиях пакетных менеджеров и связанной с ними инфраструктуре (например, в структуре метаданных репозиториев) происходят радикальные изменения. Если же имеющийся пакетный менеджер способен устанавливать пакеты от новой версии дистрибутива, то нет никаких причин его не использовать.

Создание шаблонов

Проще всего создать шаблон на основе уже существующего. Достаточно создать каталог для шаблона в соответствии с его именем, скопировать файлы из исходного шаблона и внести необходимые изменения. Если все модификации сводятся к изменению набора пакетов или настроек по умолчанию, то много времени это не займёт.

Можно создать шаблон с нуля с помощью утилиты `vzmktmpl` из пакета `vztt-build`, которому на вход надо передать файл с инструкциями. Примеры инструкций есть в `/usr/share/vztt/samples`.

Алексей Шабалин
Москва, ООО «Базальт СПО»

Сети в облаках

Аннотация

Публичные, приватные, гибридные облака получили широкое распространение в последние годы. Сеть в облаке является одним из видов ресурсов, наравне с вычислительными ресурсами, системами хранения. Знания, которыми раньше обладали сетевые администраторы, теперь необходимы и для администраторов и разработчиков облачных приложений.

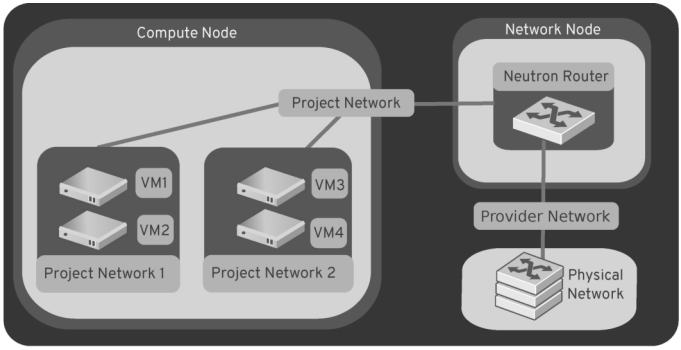
Начиная пользоваться облачными сервисами, администратор вынужден создать не просто виртуальную машину, где разместит нужное приложение или набор приложений, но и целиком виртуальную сетевую инфраструктуру, к которой будут подключены создаваемые виртуальные машины.

Для создания сетей в облаке предоставляются такие же компоненты, как и в реальных сетях, только виртуальные:

- Switch (виртуальный L2 коммутатор)
- Router (виртуальный L3 маршрутизатор)
- Firewalls (FWaaS, правила фильтрации для маршрутизатора)
- Load balancers (LBaaS, виртуальный балансировщик нагрузки)
- VPN (VPNaaS виртуальный сервис VPN)

Различные публичные облака накладывают разные ограничения, которые необходимо учитывать при планировании. Например:

- в создаваемых виртуальных сетях адреса выдаются только по DHCP. Самостоятельно изменить адрес на виртуальной машине средствами ОС невозможно.
- IPv4 адреса выдаются только из приватного(RFC1918) диапазона адресов.
- выданный ресурс «публичный IP адресс» назначается не на сетевой интерфейс виртуальной машины, а привязывается через DNAT к внутреннему адресу.



В простых сценариях не возникает проблем с использованием сетей в облаках. Но в более сложных сценариях, например, географически распределённых приложениях, при соединении сетей с помощью VPN, необходимо учитывать все нюансы при планировании виртуальных сетей.

Никита Ермаков
Москва, Базальт СПО

Открытая архитектура RISC-V

Аннотация

RISC-V является архитектурой с открытым набором команд (ISA) основанной на принципах компьютера с сокращённым набором команд (RISC). На данный момент существует несколько реализаций RISC-V, например системы на кристалле (SoC) от компании SiFive. Среди разработчиков операционных систем (ОС) на базе ядра Linux, таких как Red Hat, ALT, Fedora, Debian, начат процесс портирования ОС на RISC-V. В данном докладе приведены текущие успехи, а также дальнейшие планы, по портированию ALT на RISC-V.

Архитектура набора команд (ISA) это спецификация по инструкциям или машинному коду которые передаются в процессор, а также другие детали о том, как работает данное семейство процессоров. Современные ISA это огромные и сложные документы, но что более важно, большинство ISA имеют патенты и агрессивные авторские права.

Например если некто решил независимо реализовать ARM процессор, ему потребуется понести расходы связанные с лицензиями [1].

Открытые ISA, среди которых которых RISC-V, напротив имеют пермиссивные лицензии. Например, спецификации RISC-V имеют лицензию Creative Commons license (CC BY 4.0), все инструкции RISC-V свободны от патентов. Помимо RISC-V существуют другие открытые ISA, например OpenSPARC [2], OpenRISC [3]. Проект OpenSPARC был начат корпорацией Sun Microsystems в 2005 году, были открыты спецификации процессора UltraSPARC T1 и T2. На данный момент дизайн OpenSPARC является устаревшим. Другой проект OpenRISC берёт своё начало в 2000 году. На сегодняшний день существует единственная версия архитектуры OpenRISC 1000 (or1k), которая описывает семейство 32-х и 64-х битных процессоров имеющих опциональные расширения такие, как поддержка векторных операций и чисел с плавающей запятой. В 2011 году ядро Linux 3.1 получило поддержку OpenRISC и в 2014 году появился порт Debian. Однако в 2016 году порт Debian закрылся вследствие проблем с авторскими правами [4]. Активность по изменению кода ядра связанного с OpenRISC замедлилась.

RISC-V является little-endian архитектурой. Одна из главных особенностей архитектуры RISC-V это её расширяемость, что позволяет создавать разнообразные реализации. Минимальная спецификация архитектуры RISC-V включает в себя только инструкции для записи, сохранения, переходов и целочисленной арифметики. Код инструкций не зависит от размера регистра, минимальная спецификация обозначается RV32I, RV64I или RV128I для размеров операнда 4, 8 или 16 байт соответственно. Существуют различные расширения, некоторые из них [5]:

- **M** — умножение и деление целых чисел;
- **A** — атомарные операции;
- **F** — число с плавающей запятой одинарной точности;
- **D** — число с плавающей запятой двойной точности;
- **Q** — число с плавающей запятой четверной точности;
- **L** — десятичная плавающая точка;
- **C** — сжатые 16-битные инструкции.

Большинство Linux совместимых чипов имеют спецификации RV32IMAFDC или RV64IMAFDC. Среди разработчиков операцион-

ных систем на базе ядра Linux, таких как Red Hat, ALT, Gentoo, Fedora, Debian идёт активный процесс портирования ОС на RISC-V. Крупные компании также заинтересованы в использовании RISC-V. Например, NVIDIA планирует использовать RISC-V для замены процессора Falcon на графических картах GeForce [6], компания Western Digital планирует использовать RISC-V в своих будущих проектах [7]. Таким образом архитектура RISC-V заполучила большое внимание рынка ИТ и является перспективной открытой технологией которая, возможно, позволит человечеству продвинуться вперёд на пути к открытости аппаратной начинки наших устройств.

Литература

[1] A long look at how ARM licenses chips, https://www.semiaccurate.com/2013/08/07/a-long-look-at-how-arm-licenses-chips/

[2] About OpenSPARC, http://www.oracle.com/technetwork/systems/opensparc/opensparc-overview-1562924.html

[3] OpenRISC Project Overview, https://www.openrisc.io

[4] Debian port to the OpenRISC/or1k, https://wiki.debian.org/OpenRISC

[5] The RISC-V Instruction Set Manual Volume I: User-Level ISA, https://riscv.org/specifications/

[6] Слайды презентации RISC-V in NVIDIA, 6^{th} RISC-V Workshop, https://riscv.org/wp-content/uploads/2017/05/Tue1345pm-NVIDIA-Sijstermans.pdf

[7] Western Digital to accelerate the future of next-generation computing architectures for big data and fast data environments, https://riscv.org/2018/05/western-digital-blog-top-five-from-risc-v

Андрей Черепанов
Москва, ООО «Базальт СПО»
Проект: Sisyphus altlinux.org

Планы на дистрибутивы Альт 9.0

Аннотация

В докладе рассматриваются ключевые особенности, которые планируются в дистрибутивах Альт 9.0.

В первом квартале 2019 года планируется выпуск Девятой платформы, на базе которой будут выпущены дистрибутивы Альт 9.0: для серверов, рабочих станций, образовательных учреждений и домашних пользователей.

Серверный дистрибутив

1. Особый упор будет сделан на направления виртуализации и управления. Помимо PVE manager, который поставлялся в Альт Сервер 8, планируется включение Kubernetes для автоматизации развёртывания и управления контейнерами. Для этого были сделаны официальные шаблоны контейнеров Альта.
2. Особое внимание будет уделено управления конфигурациями в рамках сети. Для упрощения управления через технологию Ansible будет предоставлен Semaphore, для автоматизации с помощью Puppet — платформа Puppet Server с бэкендом PuppetDB, управляемая с помощью Puppetboard.
3. Как в мире рабочих станций перешли с программ на приложения, мы предлагаем не настраивать отдельные службы вручную, а предоставить унифицированную платформу *deploy*, которая поможет командами в консоли развернуть контроллер домена Active Directory или FreeIPA, сервер каталогов 389-ds, различные службы, а также просмотреть их состояние и корректно удалить. Платформа проектируется модульной для быстрого добавления поддерживаемых служб.
4. Долгое время проблемой администрирования контроллеров домена Active Directory на базе Samba было отсутствие свободного приложения, аналогичного Remote Server Administration Tools —

RSAT на платформе Windows. Мы включим графическое приложение по администрированию пользователей и групп Active Directory, разрабатываемое на Qt.

5. В составе дистрибутива будут средства по клонированию подготовленных дектопов и массового развёртывания их на компьютеры по сети.

Десктопные дистрибутивы

При проектировании дистрибутивов следующего поколения практически единогласно была высказана идея улучшения дизайна, который для Альта долгие годы был предметом нареканий пользователей. Поэтому мы решили более внимательно подойти к этому вопросу.

1. Расположение элементов на рабочем столе будет унифицированным и сделано привычным для пользователей, ранее использовавших Windows.
2. Планируется разработать собственную тему, которая позволит единообразно выглядеть приложениям GTK+ и Qt/KDE.
3. Тема значков будет выбрана более современная с элементами Material Design и плоским дизайном.

Помимо дизайна, особое внимание будет уделено управляемости операционных систем в корпоративных сетях:

1. В дистрибутивах будут идти необходимые компоненты по управлению как с помощью Puppet, так и с помощью Ansible.
2. Для применения различных параметров у доменных пользователей, объединённых в группы, будут предложены Fleet Commander на платформе FreeIPA и базовый набор типовых групповых политик (GPO) на платформе Active Directory, для чего будет доработан SSSD.
3. Предоставление удалённого доступа по протоколам VNC, Spice, X.2GO.

Для упрощения поиска, установки и удаления приложений в дистрибутивы будет включён gnome-software, который стал фактически стандартом центров управления программным обеспечением в Линуксе. Помимо поддержки общепринятого в Альте apt-rpm, можно будет устанавливать приложения в форматах Appimage и Flatpak. При

этом будет переработана и локализована база приложений в формате Appdata.

Поддержка отечественных криптографических алгоритмов

Важным шагом будет предоставление удобных средств электронной подписи и кодирования с помощью отечественных алгоритмов по ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012:

1. Поддержка алгоритмов по ГОСТ в OpenSSL, OpenVPN и OpenSSH;
2. Возможность создавать ключи и подписывать сертификаты в Alterator (для использования доступа через VPN, к примеру);
3. Поддержка взаимоотношений как со встроенным удостоверяющим центром (alterator-ca), так и с внешними (в том числе для квалифицированной электронной подписи);
4. Удобные собственные графические приложения для работы с ЭП и шифрованием с поддержкой как КриптоПро, так и OpenSSL.
5. Вычисление и проверка контрольной суммы по ГОСТ Р 34.11-2012 непосредственно из диспетчера файлов.

Разные аппаратные архитектуры

Выпуск Альт 9.0 интересен тем, что впервые планируются дистрибутивы на разных аппаратных архитектурах. Помимо 32-битной и 64-битной архитектуры Intel, планируется выпустить дистрибутивы для Эльбруса (4C и 8C), десктопные дистрибутивы на Байкал-T1 и ARMv7, а, синхронно с появлением отечественных процессоров с архитектурой Aarch64, — и на архитектуре Aarch64. Также возможна публикация отдельных сборок на архитектурах Little-endian MIPS64 (Loongson) и RISC-V.

Александр Боковой
Эспоо, Финляндия, Red Hat

Проект: Clevis/Tang `https://github.com/latchset`

Технология удалённого шифрования дисковых подсистем в Red Hat Enterprise Linux

Аннотация

Если данные зашифрованы, кто и когда должен иметь к ним доступ при облачном хранении? Как предохранить украденные данные от расшифровки? Как уменьшить риски воровства при транспортировке физического носителя? Свободные проекты Clevis и Tang обеспечивают привязку данных к внешним ресурсам. Вместе они создают простую и масштабируемую платформу для шифрования хранимых данных и обеспечения контроля за их доступом.

Типичная ситуация когда пропадает электричество — машины в датацентре необходимо загрузить заново. Диски этих машин зашифрованы, так что сотрудникам приходится бегать между консолями и вводить пароли. Хотелось бы определённой автоматизации.

Если вчера мы в основном работали над тем, чтобы стандартизировать криптографические примитивы и их использование, то сегодня фокус на автоматизации процессов. Стандарты важны, но их использование тоже должно быть удобным. А завтра нам потребуются правила применения автоматизации, потому что вся эта автоматизация сегодня бинарная: «можно ли разблокировать автоматически этот том? да/нет». Но когда мы мы всё автоматизируем, станет понятно, что мы хотим расшифровывать данные в разных условиях и согласно разным правилам. В конце концов, всё сведётся к ответу «да/нет?», но путь, который к этому ответу приводит, может быть извилистым. Так что политики применения или правила будут скорее «слайдером», чем простым переключателем.

Однако первый вопрос, на который стоит ответить — «Как идёт автоматизация?». Всё начинается со страшной тайны — данных, которые мы хотим зашифровать. Эти данные шифруются с помощью симметричной криптографии. Ключ шифрования наших данных сам подвергается шифрованию с помощью ещё одного симметричного ключа шифрования ключа шифрования. Это стандартная модель, используемая много где, в частности, в подсистеме LUKS в ядре Линукс. Стандартная модель подразумевает, что любой, кто должен расшифровать

ключ шифрования, должен иметь доступ к паролю ключа шифрования ключа шифрования.

Чтобы избежать такого распространения паролей, мы используем значительно более стойкий ключ, который сохраним в системе депонирования ключей. В случае необходимости мы запросим этот ключ из системы депонирования ключей. Проблема с этой архитектурой в том, что она подразумевает целый ряд не всегда озвучиваемых требований:

- канал обмена данными с системой депонирования ключей нужно шифровать
- клиент должен удостовериться, что запрос обработан доверяемым сервером системы депонирования ключей
- сервер системы депонирования ключей должен удостовериться, что запрос пришёл от доверяемого клиента
- обе стороны должны поддерживать одну и ту же цепь доверительных отношений, чтобы эта аутентификация и проверка сработала (общий удостоверяющий центр или KDC)
- система депонирования ключей, общий удостоверяющий центр или KDC должны своевременно бэкапиться.

Фактически, нам нужно бэкапить состояние и ключи всех вовлечённых центров и систем. Если мы потеряем ключи, хранимые в системе депонирования, мы потеряем данные на наших машинах. Если потеряем ключи удостоверяющего центра или KDC, не сможем организовать защищённый канал обмена с серверами системы депонирования.

Это стандартная модель депонирования ключей, здесь нет ничего экзотического, её используют многие решения на рынке. К сожалению, уязвимость HEARTBLEED показала, что даже для хорошо налаженной TLS системы возможны сбои и чтение зашифрованных данных в сети.

HEARTBLEED научила нас, что

- опасно предполагать, что перенос ключей можно защитить средствами TLS
- сложность архитектуры увеличивает поверхность для атаки
- системы депонирования ключей сложны для внедрения, даже если использовать средства автоматизации

- x.509 сложно использовать правильно. Ситуации, когда приватный ключ отсылается вместо публичного совсем не единичны. Код, реализующий стандарты семейства x.509, тоже подвержен ошибкам, не говоря уже о классических проблемах с кодированием и декодированием данных в нотации ASN.1.

Однако, то, что мы делаем для получения изначальной страшной тайны, выглядит фактически как обмен ключами. Стандартный протокол Диффи-Хеллмана на эллиптических кривых позволяет участникам создать общий секрет для порождения ключа для шифрования. Каждый участник генерирует пару приватный-публичный ключи (C и c для клиента, S и s для сервера, соответственно). Оба участника обмениваются своими публичными ключами и используют их для вычисления общего секрета K. Если оба участника не используют дополнительную информацию друг о друге, обмен можно считать анонимным. Полученный общий секрет K либо прямо используется для шифрования, либо используется для порождения ключа для шифрования.

Клиент	Сервер
$C \in_R [1, p-1]$	$S \in_R [1, p-1]$
$c = gC$	$s = gS$
$c \longrightarrow$	$\longleftarrow s$
$K = gSC = sC$	$K = gCS = cS$

Однако этот подход подразумевает, что клиент хранит свой приватный ключ для взаимодействия с сервером. Мы же хотим добиться восстановления ключа шифрования только при наличи сервера. Разобьём процесс на стадию генерации (provision) и стадию восстановления (recovery).

Первый шаг — сервер генерирует приватный ключ. Используя приватный ключ, он генерирует публичный ключ и публикует его для тех, кто его запросит. Клиент может взять этот публичный ключ каким-угодно способом — либо связавшись с сервером или получив этот ключ в оффлайне. Когда клиент захочет зашифровать данные, то он сгенерирует свой приватный ключ и вычислит свой публичный ключ. Используя публичный ключ сервера и свой приватный ключ, он может вычислить общий секрет K. Этот ключ (или его производную) клиент затем использует для шифрования данных.

Когда данные зашифрованы, клиент уничтожает общий секрет K и приватный ключ клиента. Это означает, что у клиента остаётся

только его публичный ключ и публичный ключ сервера. Клиент больше не может вычислить общий секрет K самостоятельно.

Для восстановления клиент может послать собственный публичный ключ серверу. Сервер вычислит общий секрет K и отправит его назад. Этот режим значительно легче, чем традиционная система депонирования ключей. У сервера нет никакого состояния клиента. Всё, что ему достаточно знать — свой приватный ключ. К сожалению, этот режим полностью небезопасен.

Во-первых, общий секрет K теперь посылается по сети в открытом виде, так что он доступен пассивному наблюдателю. Даже если этот наблюдатель не сумел перехватить обмен с сервером, ему достаточно получить доступ к публичному ключу клиента, чтобы самому повторить этот обмен. К тому же, сам сервер, получив доступ к публичному ключу клиента, вычисляет общий секрет K и может скомпромитировать клиента. То есть, публичный ключ клиента в этом обмене должен быть секретным.

Мы на самом деле можем этого добиться небольшим изменением ECDH, которое было изобретено Натаниелем МакКаллумом и Бобом Релья из Red Hat. На стадии генерации ничего не меняется. Все изменения — при восстановлении ключа. Как и прежде, клиент вычисляет общий секрет K, шифрует данные, уничтожает K и свой приватный ключ, так что теперь невозможно расшифровать данные в одиночку.

Генерация:

Клиент	Сервер
$C \in_R [1, p-1]$	$S \in_R [1, p-1]$
$c = gC$	$s = gS$
$K = gSC = sC$	$\longleftarrow s$
Выбросить: C, K	
Сохранить: c, s	

Когда нужно расшифровать данные, клиент генерирует эфемерную пару ключей. Она создаётся каждый раз, когда клиенту необходимо получить ключ для расшифровки данных. Клиент складывает свой основной публичный ключ с эфемерным публичным ключем. Результат x отсылается на сервер. Сервер выполняет свою часть протокола Диффи-Хеллмана, посылает общий секрет y назад. И клиент может вычислить настоящий общий секрет K, вычитая из полученного у произведение публичного ключа сервера на приватный эфемерный ключ. Это очень простая математика, но если мы никогда не

распространяем эфемерную пару ключей, то и наш публичный ключ клиента остаётся секретным.

Восстановление:

Клиент	Сервер
$E \in_R [1, p-1]$	
$e = gE$	
$x = c + e$	
$x \longrightarrow$	
	$y = xS$
	$\longleftarrow y$
$K = y - sE$	
потому что $K = gCS + gES - gSE$	

Сервер получает что-то, что выглядит как случайный публичный ключ. Алгоритму всё равно, над чем производить вычисление, но этот обмен также не содержит никакой информации, которая могла бы идентифицировать клиента (кроме IP-адреса соединения). Для сервера это простая операция, а пассивный наблюдатель не может ничего поделать с перехваченным результатом, потому что у него нет доступа к эфемерной паре ключей.

Этот обмен удивительно быстр. В случае соединения TLS первым делом выполняется обмен Диффи-Хеллмана. В нашем случае нам не требуется выполнять ничего больше. Шифрование данных в сети не требуется, потому что это просто обмен публичными ключами. На сервер не требуется хранить какое-либо состояние для каждого клиента. Никаких дополнительных бэкапов.

Если ключ сервера поместить в крипто-оборудование, то сам сервер не будет иметь доступа к какому-либо крипто-материалу, который можно было бы украсть. Атакующий сможет только добиться подписывания нескольких ключей, но не сможет унести приватный ключ сервера с собой.

Проект, который реализует серверную часть, называется Tang. Сервер очень прост: это минимальный HTTP сервер, общающийся с клиентом при помощи JOSE (JSON Object Signing and Encryption). Он достаточно быстр — однопоточный сервер на обычном ноутбуке справляется с более чем 2000 запросов в секунду. Код минимальный, имеет минимальные зависимости и хорошо переносим.

На стороне клиента мы представили всю логику в рамках проекта Clevis. У него тоже минимальные зависимости, он интегрирован с си-

стемой ранней загрузки в Fedora/RHEL/CentOS и со средой GNOME. Clevis доступен в Fedora 24+, RHEL 7.4+, а также в Debian Testing.

Зашифрованные данные представляют из себя стандартный токен JSON Web Encryption.

Зашифрованный блок данных с привязкой к серверу Tang можно хранить где угодно. Например, его можно хранить в заголовке метаданных раздела формата LUKS. Это применимо как к основному тому, который разблокируется при загрузке, если есть доступ к серверу Tang, так и к динамически подключаемым томам в графической среде GNOME.

Разблокирование в момент загрузки подразумевает сетевой доступ из initrd. В результате, полученный том можно разблокировать только при наличии сетевой связности с соответствующим сервером Tang. Если зашифрованный диск перевозят между центрами обработки данных, то за сохранность данных можно не беспокоиться.

Но что делать, если необходимы гарантии доступности данных только в рамках конкретного сервера, если диск вставлен в его шасси?

На помощь приходит другая известная математическая модель. Мы хотим не просто автоматизировать доступ, но сделать это на основании определённых правил. В 1979 году израильский криптоаналитик Ади Шамир предложил пороговую схему разделения секрета между несколькими участниками так, что

- для восстановления секрета достаточно определённое количество участников
- меньшее количество участников не могут получить никакой информации о секрете.

Схема разделения секрета Шамира позволяет строить деревья решений, для которых можно предопределить какие ветки должны присутствовать обязательно одновременно на каждом уровне. В Clevis реализована схема разделения секрета Шамира для связывания различных пинов между собой. Таким образом, можно потребовать ввод пароля, запрос от сервера Tang или что-то ещё и построить политику на основании нужных нам требований. Все операции (пины) Clevis в заданой политике вызывает параллельно друг другу, так что они выполняются независимо.

Одно из новшевств в бета-версии RHEL 7.6 — поддержка стандарта TPM 2.0 в качестве пина Clevis. На платформах, где присутствует TPM 2.0, можно хранить ключ для разблокирования раздела LUKS

в аппаратном модуле TPM 2.0. Если в описании политики Clevis использовать схему разделения секрета Шамира и потребовать дополнительно привязку к серверу Tang, то при использовании порога 2 мы фактически требуем одновременно нахождение раздела LUKS в определённом шасси и доступ к определённому серверу Tang.

Евгений Синельников
Саратов, ООО «Базальт СПО»

Проект: Group Policy Object Manager https://github.com/altlinuxteam/gpom

Реализация групповых политик в решениях на базе дистрибутивов ALT

Аннотация

Групповые политики — это набор правил и настроек для серверов и рабочих станций, реализуемых в крупных корпоративных решениях. Реализация групповых политик требует тесной интеграции множества независимых модулей операционной системы. Данный доклад посвящён реализации поддержки групповых политик Active Directory в решениях на базе дистрибутивов ALT и Sisyphus.

Существует два основных вида так называемых **групповых политик**. Это *политики для компьютеров* и *политики для пользователей*. Причём пользователи обычно добавляются в группы, что вносит неоднозначность — о каких группах и политиках идёт речь? Если о группах пользователей, то причём тут компьютеры? Если об отдельных объектах пользователь или компьютер, то почему политики групповые?

Так вот, кроме групп безопасности (в которые обычно включаются пользователи) в протоколе LDAP предусмотрена иерархия объектов, хранящихся в дереве каталогов (и в соответствующей иерархической базе данных). При этом объекты пользователи и компьютеры в этом дереве объектов могут быть созданы в разных контейнерах и могут переноситься из одного контейнера в другой. А группировка объектов, на которые распространяются групповые политики, прежде всего, распространяется именно на расположение объектов в этих контейнерах. В Active Directory такие контейнеры принято называть организационными подразделениями (organizational units).

Задача реализации поддержки групповых политик в Linux-решениях отличается, прежде всего, тем, что связана скорее с клиентской стороной и проблемой, «как применить те или иные политики?», а не только с тем, «где их хранить и как их получить на клиенте?» Тем не менее, для Active Directory, второй вопрос становится тоже очень важным, поскольку исходное хранилище настроек изначально ориентировано только на клиентские рабочие станции на базе Microsoft Windows. И решению именно этого вопроса, а также момента о том, как интерпретировать неродные Windows политики на Linux клиентах и посвящён этот доклад.

Стоит отметить, что подходы по управлению конфигурациями на уровне компьютеров (Configuration Management) уже давно существуют в виде решений на базе Chef, Puppet, Ansible и т. п. инструментов. А вот увязка такого управления с управлением пользователями в рамках целостной инфраструктуры и единый подход к хранению и управлению этими конфигурациями — задача более высокого уровня. Эта задача включает в себя, со стороны клиента, не только механизм управления настройками отдельных приложений, но и интеграцию применения дополнительных настроек в процессе аутентификации и авторизации, а также управления элементами графического интерфейса и привязку этих настроек отдельным сессиям пользователей, причём не только локальным, но и сетевым.

В связи с этим, для применения пользовательских настроек в проекте gpom мы разделили пользовательские политики на следующие категории:

- политики применяемые при входе пользователя, на этапе создания сессии (PAM session и NSS initgroups);
- пользовательские политики, требующие административных привилегий (настройка CUPS и т.п.);
- политики, требующие контекст графической сессии, выполняемые с пользовательскими привилегиями (фон рабочего стола, дополнительные ярлыки на рабочем столе и т. п.).

На текущий момент в рамках проекта по применению групповых политик [1] мы детально разобрали последовательность чтения групповых политик (рис. 1):

- хранение в LDAP (на уровне привязки (GPLink) объектов групповых политик (GPO) к LDAP-объектам пользователи и компьютеры);

- сценарий их применения (на уровне порядка действий по учёту специальных ограничений — ntSecurityDescriptor).

Рис. 1: Последовательность чтения групповых политик

Кроме того, мы детально разобрали последовательность действий по применению нескольких групповых политик — установка программного обеспечения, смена настроек рабочего стола, установка принтера. Цепочка действий по применению групповых политик представлена на рис. 2:

- фильтрация и отображение существующих политик на уже реализованные;
- чтение из Sysvol (на уровне списка расширений групповых политик — GPE и настроек конкретных клиентских расширений — CSE) [2].

Следующий шаг по включению данного механизма предполагает дальнейший разбор и классификацию существующих групповых политик, включение дополнительных сервисов которые позволят применять групповые политики без перезагрузки и выхода пользователя из системы, а также распределение различных видов политик в

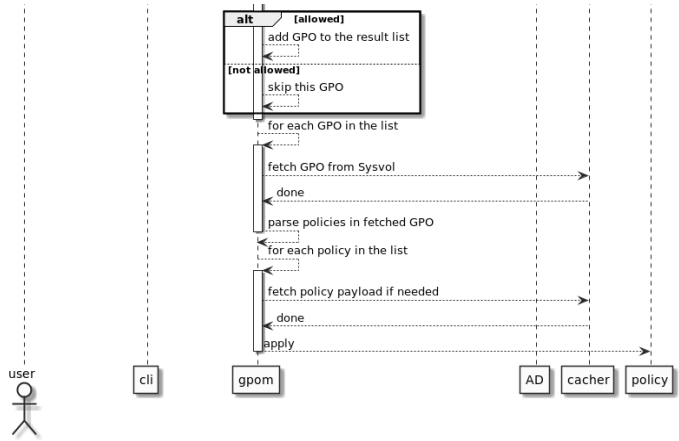


Рис. 2: Последовательность применения групповых политик

рамках соответствующих системных сервисов — например *Restricted Groups* [3] на уровне SSSD или дополнительного NSS-модуля.

Литература

[1] David Mulder, Samba Group Policy for AD DC, 2017, `https://sambaxp.org/archive_data/SambaXP2017-SLIDES/Day3/Track2/SambaGroupPolicyforADDC-DavidMulder.odp`

[2] David Mulder, Group Policy Client Side Extensions, 2018, `https://sambaxp.org/fileadmin/user_upload/sambaXP2018-Slides/Mulder-SambaXP-2018_2--klein.pdf`

[3] Marc Muehlfeld, Managing local groups on domain members via GPO restricted groups, 2017, `https://wiki.samba.org/index.php/Managing_local_groups_on_domain_members_via_GPO_restricted_groups`

Георгий Быстренин, Сергей Бубнов
Саратов, ООО «Базальт СПО»

Проект: Infra `https://github.com/altlinuxteam/infra`

Развёртывание инфраструктуры средствами Ansible на PVE сервере

Аннотация

В данном докладе представлены наши наработки по автоматическому развёртыванию боевых и тестовых виртуальных машин средствами Ansible на Proxmox VE Сервере (на базе платформы Альт Сервер 8). Особенностью данного способа автоматического управления виртуальными окружениями является возможность быстрого восстановления инфраструктуры к рабочему состоянию в случае сбоев, а также возможность быстро пересоздать виртуальные машины с ранее заданным состоянием для целей тестирования.

В решениях на базе дистрибутивов ALT в качестве промышленной системы виртуализации используется Proxmox Virtual Environment (PVE-сервер). Программный интерфейс для управления виртуальными окружениями в PVE предусмотрен [1], но средства управления, кроме как web-интерфейс, нужно устанавливать отдельно. В связи с этим возникает проблема автоматизации развёртывания.

Одной из целей, при решении данной задачи, является автоматическое тестирование изменений, а также воспроизводимость ошибок и выявление регрессий, особенно в случае невозможности получения доступа в закрытый контур заказчика. Другой, не менее важной целью, является автоматическое развёртывание боевой инфраструктуры, аналогичной той, которая развёртывается при тестировании.

В рамках подготовки средств автоматического управления для PVE-сервера, в проекте Infra, задача была разделена:

- автоматизация подготовки рабочих образов;
- автоматизация процесса интеграционного тестирования и развёртывания рабочей инфраструктуры.

Для автоматической генерации образов был выбран Packer [2], как инструмент позволяющий получить из публично-доступных образов дистрибутивов установленные и подготовленные образы для запуска в системе виртуализации.

Для автоматической установки приложений и запуска необходимого набора виртуальных узлов были рассмотрены следующие варианты:

- Vagrant, как инструмент разработчика через плагин-провайдер к PVE-серверу;
- Terraform [3], как инструмент для развёртывания облачных решений, для которого существуют реализации провайдера proxmox, написанные на Go;
- Ansible — решение, которое может быть использовано как для запуска сценариев (playbooks) через ssh-протокол, так и для управления виртуальными узлами на Proxmox VE через модуль proxmoxer [4].

В итоге, наиболее стабильным, надёжным, универсальным и подготовленным для работы с PVE-сервером показал себя последний вариант.

Логически конфигурация проекта состоит из нескольких файлов-описаний на языке yaml:

- для приложений в виде последовательности Ansible сценариев;
- для окружений в виде набора настроек виртуальных узлов (публичные ssh-ключи разработчиков, URL-репозиториев источников, настройки доступа в виртуальное окружение — на PVE-сервер);
- для описания стека сетевых настроек, количества и типа виртуальных узлов.

Приложения

Приложениями являются конечные сервисы с которыми тем или иным образом общается конечный пользователь стека виртуальных узлов (объектов стека). Для примера: samba domain controller, ntp-server, dhcp-server, tftp-server, bind, NFS-share, redmine, и т.д.

Каждое приложение жёстко привязано к объекту(ам) стека и разворачивается на них. Если же идеология приложения подразумевает функции Высокой доступности и/или Fault Tolerance, то оно может быть развёрнуто на нескольких объектах стека.

Также приложение может иметь собственный набор настроек в рамках конкретного стека. Эта возможность существует скорее для

удобства и помогает поддерживать общую конфигурацию приложений лаконичной и не загромождать её лишними деталями.

Окружение

Определения в конфигурации окружения описывают общие сущности характерные для всех объектов. Такими могут являться: список пользователей с их **ssh-ключами** и **pgp-ключами**, источники **apt-репозиториев**, учётные данные для доступа к системам виртуализации и/или облакам, и т.д.

Стек

Стеком является непротиворечивое с точки зрения конфигурации множество объектов исполняющих приложения развёртываемых в рамках конкретного окружения. Ими могут быть: виртуальные машины, железные сервера, экземпляры в облачных сервисах, и т.д. Основным и единственным строгим требованием к объекту стека является наличие доступа по **ssh** и беспарольный **sudo**, либо доступ по **ssh** под пользователем **root**.

Конфигурация стека может включать в себя: имя домена, список DNS-серверов, список виртуальных машин с параметрами, конфигурация сети и другие общие для всех приложений параметры.

Инфраструктура

Набор настроек в проекте **Infra** разработан так, чтобы детали инфраструктуры были отделены от основных Ansible-сценариев. Рабочее окружение для развёртывания стеков состоит из трёх частей:

1. набор дополнительных Ansible-сценариев и ролей;
2. конфигурация окружений и стеков;
3. база с паролями от шифрованных параметров окружений и стеков.

Конкретные примеры файлов описаний можно найти в документации проекта.

Литература

[1] PVE Community, Proxmox VE API, 2018, https://pve.proxmox.com/wiki/Proxmox_VE_API

[2] HashiCorp, Intro Packer, 2018, https://www.packer.io/intro/index.html

[3] HashiCorp, Intro Terraform , 2018, https://www.terraform.io/intro/index.html

[4] Red Hat, Inc., Ansible Documentation: proxmox — management of instances in Proxmox VE cluster, 2018, https://docs.ansible.com/ansible/2.5/modules/proxmox_module.html

Роман Симаков
Муром, ООО «РЕД СОФТ»
Проект: СУБД Ред База Данных www.reddatabase.ru

Развёртывание инфраструктуры средствами Ansible на PVE сервере

Аннотация

Доклад направлен на обзор изменений в СУБД Ред База Данных 3.0 и описание самых важных функций. С момента выпуска версии 2.6 прошло много времени и в СУБД удалось реализовать множество улучшений и дополнений: многопоточная архитектура суперсервера с общим кешем, поддержку пакетов, оконные и статические функции, совместимость с SQL-2008, двунаправленные курсоры, 64 битные счётчики транзакций, новые методы аутентификации, в том числе доменной и всё это по-прежнему имеет сертификат ФСТЭК.

Архитектура сервера

Новая версия СУБД Ред База Данных 3.0, основана на ядре Firebird 3.0. Ключевой особенностью является переработанная архитектура, полностью поддерживающая многопоточность. Общий кеш многопоточного сервера способен более эффективно использовать вычислительные мощности, повышая в разы масштабируемость и производительность прикладных систем. Блокировки страниц в такой

архитектуре заменяются на более легковесные латчи. Тесты TPC-C показывают кратный прирост производительности.

Кроме того, в новой версии были унифицированы исполняемые файлы. Конфигурирование архитектуры осуществляется опциями файлов конфигурации. В том числе это унифицирует и упрощает работу мейнтейнеров пакетов в ОС.

Переход к модульной архитектуре позволяет использовать для каждой версии файла БД свой провайдер, реализованный в виде плагина и забыть о проблеме поддержки обратной совместимости файлов разных версий.

Изменение в языке SQL

В области функциональности Ред База Данных 3.0 реализует триггеры на операции DDL, позволяя обрабатывать в БД события подключения и отключения пользователей, начала и завершения транзакций, создания и удаления объектов.

Поддержка пакетов (Packages), обеспечивает группировку процедур и функций и позволяет целостно управлять правами доступа к ним. Кроме этого, появилась возможность создавать SQL-функции и подпрограммы.

В соответствие со стандартов SQL-2008 реализованы оконные и статистические функции, полезные для выполнения аналитических запросов, а также полная поддержка оператора MERGE.

Кроме этого, реализованы двунаправленные курсоры, позволяющие двигаться по набору вперёд, назад и на N позиций относительно текущего положения, тип данных BOOLEAN, уникальные столбцы, упрощающие создание первичных ключей, database linger, удерживающий некоторое время ресурсы сервера после отключения последнего подключения.

Увеличены ограничения СУБД

Начиная с версии СУБД Ред База Данных 3.0, максимальный размер файла БД составляет 64Тб.

Счётчики транзакций стали 64-х битными, что позволяет практически забыть об их переполнении, даже при большом количестве транзакций в единицу времени. Дополнительные расходы на хранение не появляются пока счётчики меньше значения 2^{32}.

Методы аутентификации

В дополнение к методам аутентификации версии 2.6: по паролю, по сертификату, с использованием LDAP, реализован плагин проверки подлинности GSS, который обеспечивает автоматическую аутентификацию пользователей, уже получивших тикет при входе в систему.

Безопасность

Классы безопасности и уровни защищённости СУБД Ред База Данных 3.0 наследует в рамках сертификата ФСТЭК России №2729, действующего до 8 октября 2018 г., подтверждающего соответствие требованиям 5 класса защищённости по Руководящему документу «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищённости от несанкционированного доступа к информации» и требованиям по 4 уровню контроля недекларированных возможностей по Руководящему документу «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей». СУБД Ред База Данных 3.0 может применяться в государственных информационных системах до первого класса защищённости включительно и информационных системах персональных данных до первого уровня защищённости персональных данных включительно.

Заключение

СУБД Ред База Данных продолжает развиваться и решать задачи промышленного масштаба. Она обладает всеми необходимыми инструментами, входит в реестр отечественного программного обеспечения (№ 1) и развивается силами полностью компанией российских разработчиков.

Костарев А. Ф., Братчиков И. А.
Пермь, ООО «Невод», ООО «Новая Платформа»

Проект: Технологическая программная платформа Flexberry
http://flexberry.ru/, https://github.com/Flexberry

Технологическая программная платформа Flexberry с открытым исходным кодом для профессиональной разработки программного обеспечения

Аннотация

В докладе рассматривается разрабатываемая на принципах свободного программного обеспечения платформа Flexberry — платформа для проектирования, разработки и поддержки сложных информационных систем. Платформа включает в себя UML-редактор и визуальный конструктор форм приложений, фреймворк для создания мобильных и Web-приложений, подсистему для хранения, создания, редактирования, анализа и отображения географической информации, сервисную шину и другие компоненты, позволяющие ИТ-подразделениям предприятий и компаниям-разработчикам оптимизировать процесс проектирования, разработки и поддержки информационных систем.

Программная платформа Flexberry вобрала в себя богатый опыт разработки информационных систем на протяжении последних 20-ти лет развития.

Платформа базируется на свободном ПО и открытых технологиях. Компоненты платформы также разрабатываются в открытых репозиториях под свободной лицензией. Это позволяет создавать открытые решения с использованием платформы.

В настоящее время система широко используется в разрабатываемых системах класса:

- Комплексные учётные системы.
- Автоматизация бизнес-процессов и документооборота.
- Веб-приложения.
- Мобильные приложения.
- Интеграция информационных систем.
- Аналитические системы.
- Геоинформационные системы.

Использование платформы Flexberry даёт разработчикам следующие преимущества:

- Быстрое создание прототипов программного решения;
- Повышение эффективности взаимодействия с заказчиками за счёт применения визуальных моделей систем и процессов;
- Снижение трудоёмкости процесса разработки и сопровождения информационных систем;
- Возможность применения готовых программных компонент и готовой архитектуры;
- Быстрое разворачивание в гетерогенных сетях включая облачные платформы;

В основе технологии создания систем на платформе Flexberry лежит архитектура, управляемая моделью (Model Driven Architecture) [1]. В рамках этой архитектуры сначала с помощью редактора в формате UML или визуального конструктора форм создаётся модель предметной области проектируемого приложения. По модели создаётся прототип работающего приложения методом кодогенерации. Данный прототип дорабатывается программистами. Затем эта модель уточняется в ходе развития и поддержки проекта, эти изменения прозрачным образом автоматически переносятся в реализованное приложение.

Координация и разработка платформы распределённой группы разработчиков производится в git-репозитории `https://github.com/Flexberry`. Платформа открыта для подключения новых разработчиков к процессу её разработки.

Процесс разработки с использованием Flexberry включает в себя следующие этапы:

- Анализ и проектирование с построением UML-моделей.
- Генерация на основе созданных моделей базы данных и кода.
- Подключение готовых функциональных подсистем.
- Ручная доработка кода.
- Демонстрация Заказчику прототипа приложения.
- Детализация требований, постановок.
- Модификация UML-моделей, перегенерация кода.
- Выпуск версии продукта, переход к поддержке и сопровождению.

Архитектура системы включает в себя (см. рисунок 1):

- инструментарий по проектированию и программированию системы (этап проектирования — Design-Time);
- набор компонентов для создания информационной системы (этап исполнения программы — Run-Time).

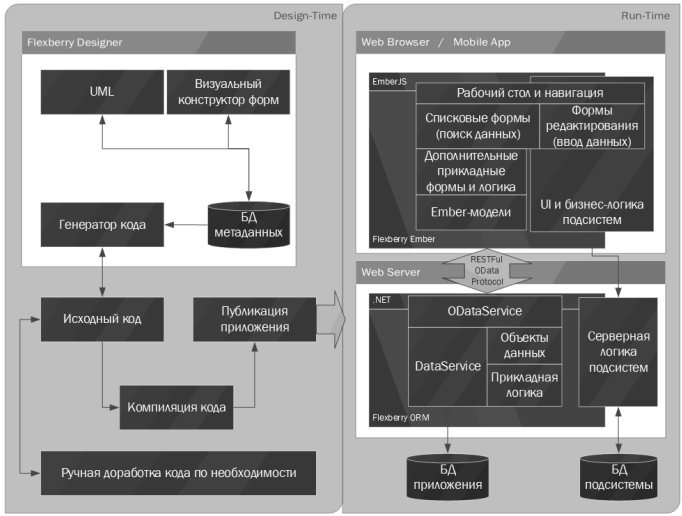


Рис. 1: Основные архитектурные компоненты платформы

Генераторы кода (см. рисунок 2) позволяют на основе модели приложения получать исходный код приложения. Сгенерированное приложение является работоспособным приложением и может быть запущено без необходимых доработок [2].

Платформа включает в себя следующие программные компоненты:

- Flexberry Designer — CASE-инструмент[1], реализующий стандартную нотацию UML[2] и визуальный (WYSIWYG) конструктор форм приложения;

[1] `https://ru.wikipedia.org/wiki/CASE`
[2] `http://uml.org/`

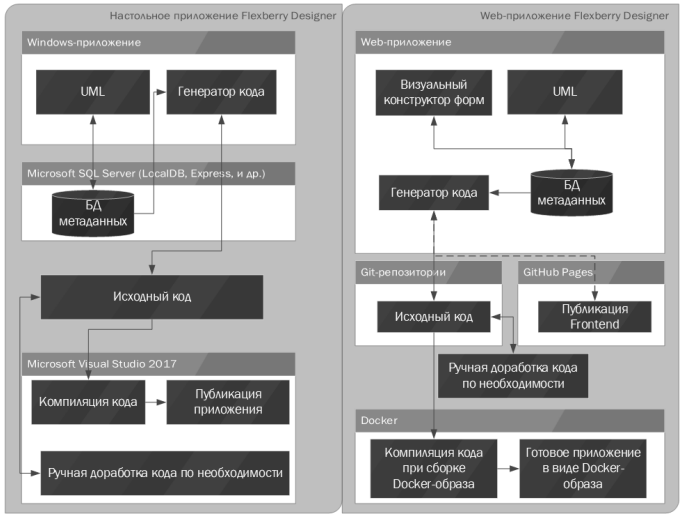


Рис. 2: Архитектура инструментария проектирования Flexberry Designer

- Flexberry Ember — фреймворк для создания Web- и гибридных мобильных приложений;
- Flexberry GIS — функциональная подсистема хранения, создания, редактирования, анализа и отображения географической информации;
- Flexberry Service Bus — сервисная шина — средство интеграции систем посредством обмена сообщениями (docker образ);
- Flexberry Analytics — функциональная подсистема отчётности и бизнес-аналитики (docker образ);
- Flexberry BPM — технология для быстрой автоматизации рабочих процессов в компании (набор docker-образов);
- и другие компоненты.

Разработанный продукт разворачивается в виде стека docker-сервисов, включающий в себя следующие сервисы:

- Ember-приложение с Odata backend;

- Сервис базы данных Postgres.

EmberJS обеспечивает возможность быстро и качественно создавать универсальный мобильный и WEB-интерфейс.

REST-интерфейс OData позволяет построить микросервисную архитектуру вокруг единой точки предоставления данных.

База данных Postgres является «стандартом de-facto» при создании систем на основе открытого программного обеспечения.

Docker-технология позволяет интегрировать данные сервисы в конечное кластерное решение [3, 4].

При необходимости в состав стека, функционирующего в составе кластера могут быть включены дополнительные docker-сервисы платформы Flexberry: Flexberry Service Bus, Flexberry Analytics, Flexberry BPM и другие.

Литература

[1] Архитектура, управляемая моделью: [Электронный ресурс] // Википедия. URL: `https://ru.wikipedia.org/wiki/Архитектура,_управляемая_моделью` (дата обращения: 01.09.2018).

[2] *Шмидт И. А.*, Архитектура платформы для разработки бизнес-приложений, Современные проблемы науки и образования. 2014. № 6. С. 348.

[3] Docker as Platform for Assignments Evaluation, 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, DAAAM 2014

[4] Docker: [Электронный ресурс] URL: `https://www.docker.com/` (дата обращения: 01.09.2018).

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

Новые возможности свободной системы управления бизнес-процессами и административными регламентами RunaWFE

Аннотация

В докладе рассматриваются возможности, появившиеся в свободной системе RunaWFE за последний год. В частности, рассматриваются дополнительные возможности фильтрации экземпляров бизнес-процессов, использование источников данных, использование глобальных ролей, новые возможности работы бота внешнего хранилища с SQL-таблицами.

Дополнительные возможности фильтрации экземпляров бизнес-процессов

В системе появилась возможность выполнять фильтрацию по названию задания с дополнительными режимами проверки прохождения точки управления через данный узел. Для ее использования надо отметить галочкой элемент «Название задания», ввести в поле фильтра название узла-действия выбрать один из следующих режимов проверки:

- Задача активна
- Задача выполнена
- Задача не выполнялась

После сохранения фильтра, в списке запущенных процессов появятся все существующие экземпляры бизнес-процессов, соответствующие заданному критерию.

Использование глобальных ролей

Глобальная роль — роль уровня предприятия, доступная во многих бизнес-процессах. Глобальная роль привязывается к контейнеру,

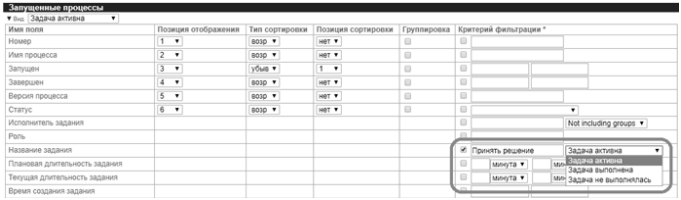


Рис. 1: Режимы проверки прохождения точки управления через узел-действие

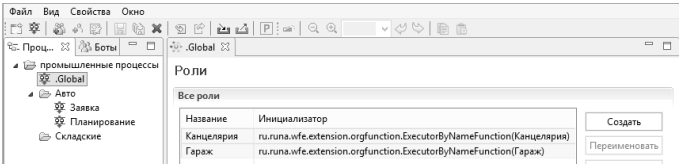


Рис. 2: Глобальные роли

название которого начинается с точки. После этого глобальная роль становится доступной во всех бизнес-процессах, относящихся к контейнеру. Ее так же, как и обычные роли, можно выбирать в узлах-действиях, обработчиках, формах, инициализаторах и других элементах среды разработки.

Использование источников данных

Источником данных может быть База данных или Excel таблица. Реализована поддержка работы с базами данных MSSQL, Oracle и PostgreSQL. Источники данных используются в обработчиках: «Внешнее хранилище данных» и »Выполнить запросы SQL».

Для работы с источниками было добавлено окно «Источники данных»:

Возможны три типа источников данных:

- Excel
- JDBC

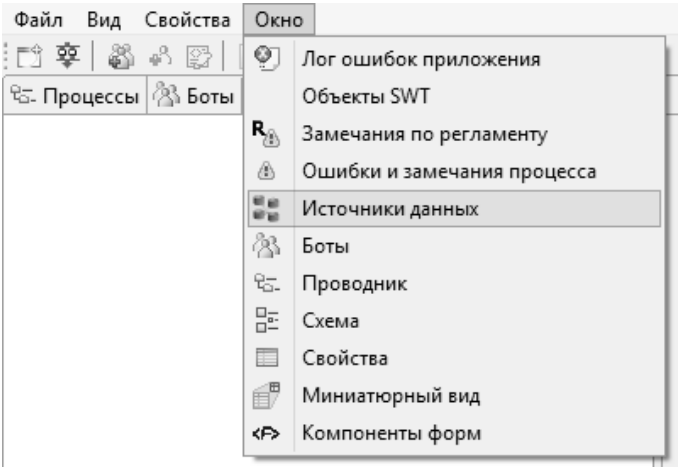


Рис. 3: Выбор окна источников данных

- JNDI

Пароль подключения к БД задается на сервере в разделе «Источники данных». Среда разработки не хранит пароли подключения к БД.

На сервер источники данных могут быть экспортированы из среды разработки или из файла. Работа с источниками на сервере ведется в разделе «Источники данных».

Новые возможности работы бота внешнего хранилища с SQL-таблицами

Раньше бот внешнего хранилища мог в качестве хранилища данных использовать только файлы Excel-документов. Теперь для бота работы с внешним хранилищем можно также использовать SQL-источники данных, которые выбираются в соответствующих формах настройки ботов.

Ссылки

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

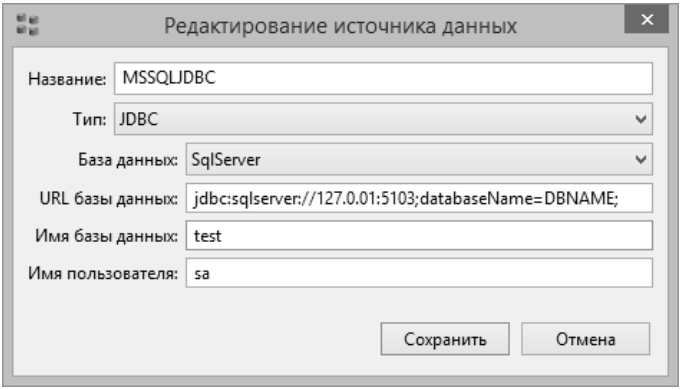


Рис. 4: Настройка JDBC — источника данных

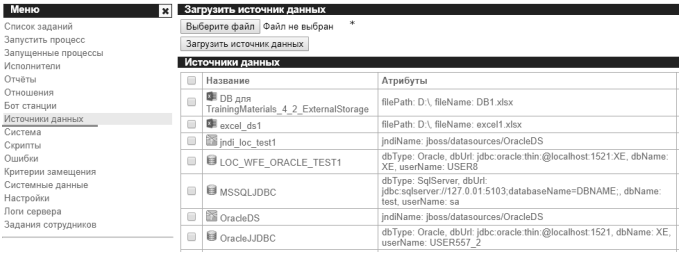


Рис. 5: Работа с источниками данных на сервере

Михаил Шигорин
Москва, Базальт СПО

Альт на Эльбрусе: обе вершины

Аннотация

Рабочая станция «Эльбрус 801-PC»[1] поддерживает три дисплея в штатной комплектации, но может быть расширена до двухместной конфигурации (где каждое место поддерживает опять же до трёх

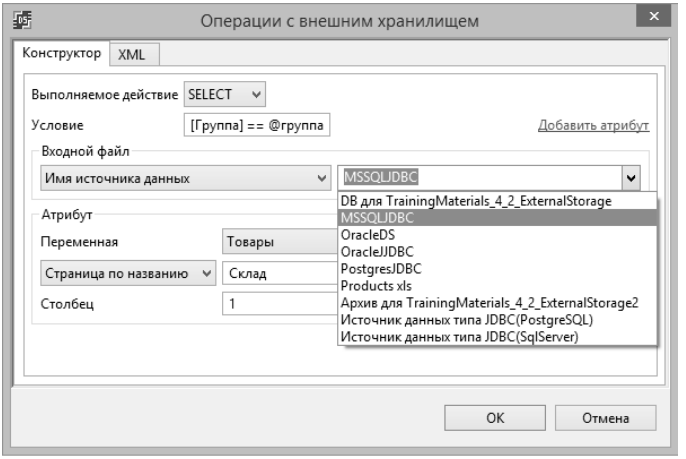


Рис. 6: Выбор источника данных для бота внешнего хранилища.

дисплеев); мы так и сделали в инсталляторе Альт Рабочая станция для Эльбрус.

Рабочая станция «Эльбрус 801-PC» идёт в комплекте с трёхголовой видеокартой Radeon, и если добавить ещё одну такую же — возможно разместить на одной машине два полноценных рабочих места: ресурсов для них более чем достаточно (8 ядер, 32 Гб ОЗУ), а вот экономический эффект получается кратный.

Мы решили опробовать «многоголовый» вариант этой весной в рамках подготовки к конференции OS Day 2018[2]; сперва воспользовались свободными выходами уже имеющейся видеокарты, но такая конфигурация с промежуточной прослойкой Xephyr получилась небыстрой из-за отсутствия 3D-ускорения (у нас уже MATE 1.20, собранный с GTK+3, но ещё не было VirtualGL), плюс достаточно громоздкой и неудобной в настройке и запуске (хотя по факту и она тоже возможна) — а вот с раздельными видеокартами всё стало простым и лаконичным, на данном этапе даже не потребовалось ничего патчить.

Описаны разные варианты[3,4,5] — с одинаковыми или различными GPU, с задействованием logind или без него; с учётом особенностей платформы и своих предпочтений остановились на двух Radeon R5

230 (один штатный и один дополнительный во втором слоте PCIe x8) и wdm[6], который умеет поднимать несколько X-серверов штатным образом.

Реализация упакована в составе пакета xorg-conf-e801-dualseat[7] и вместе с доработкой инсталятора дистрибутива Альт Рабочая станция для Эльбрус[8] попала в августовский выпуск репозитория и образов[9] (доступны клиентам МЦСТ) — так что обладатели рабочих станций «Эльбрус 801-PC» могут уже сейчас обратиться в МЦСТ для получения дистрибутива и его развёртывания.

Ссылки

1. http://www.mcst.ru/elbrus_801-pc
2. http://osday.ru/
3. http://web.archive.org/web/20101030044616/http://wiki.c3sl.ufpr.br/multiseat/index.php/Main_Page
4. https://wiki.gentoo.org/wiki/Multiseat
5. https://wiki.archlinux.org/index.php/xorg_multiseat
6. http://altlinux.org/X11/DualSeat
7. http://git.altlinux.org/people/mike/packages/?p=xorg-conf-e801-dualseat.git
8. http://basealt.ru/products/alt-workstation/
9. http://altlinux.org/ports/e2k

Иван Мельников, Дмитрий Терёхин
г. Саратов, ООО «Базальт СПО»

Проект: ALT MIPSel https://www.altlinux.org/Ports/mipsel

ALT на платформе MIPSel

Аннотация

В докладе рассмотрены следующие темы: динамика сборки пакетов в Sisyphus_mipsel, сборка и прошивка образов ALT на основе Sisyphus_mipsel для Таволги и BFK 3.1, проблема графики на BFK 3.1.

Таблица 1: Таблица по неделям.

Date	Totally missing	EV is different (we need the latest version)	EV is the same, release is smaller (we need the latest build)	EV is the same, release is larger (we need to push the fixes)	same NEVR in both repositories	present in new repo only
2018-06-29	12595	817	1844	26	3269	91
2018-07-06	12513	818	1871	27	3304	91
2018-07-13	12513	843	1875	26	3278	88
2018-07-20	12489	827	1926	37	3286	88
2018-07-27	12466	829	1912	38	3333	77
2018-08-03	12643	786	1499	40	3611	51
2018-08-10	12644	722	1250	43	3936	53
2018-08-17	12532	295	760	49	4965	33
2018-08-24	12426	228	707	55	5164	29
2018-08-31	12370	216	650	59	5289	27
2018-09-07	12358	214	659	57	5298	28
2018-09-14	12335	218	629	55	5353	29

Есть репозиторий Sisyphus_mipsel.

Есть сборочница, gyle-secondary.office.basealt.ru, ПО girar.

Есть робот, который перекладывает задачи сборки из основного Сизифа в эту сборочницу. Есть люди, которые помогают роботу там, где он не справляется, аппрувят его таски там, где он справляется и дособирают то, чего не хватает.

Есть статистика по сборке пакетов: `http://ftp.altlinux.org/pub/people/iv/reports/mipsel/stats.txt`

На основе репозитория Sisyphus_mipsel собраны образы ОС Alt для компьютера «Таволга Терминал» 2BT1, в дальнейшем именуемом Таволга, с функциональностью тонкого клиента и для оценочной платы BFK 3.1.

В образы для Таволги включено ПО клиентов удалённого рабочего стола для протоколов RDP, SPICE, RFB (используется VNC), NX (используется X2Go): connector, remmina, freerdp, tigervnc, x2goclient. [1] А также офисное ПО (LibreOffice). Имеются образы с окружением рабочего стола MATE и Xfce.

Образ для BFK 3.1 поддерживает только текстовый интерфейс пользователя. Имеется ПО для работы в сети.

Ссылка на каталог с образами:

`http://ftp.altlinux.org/pub/people/jqt4/mipsel-images/`

Прошивка образа Alt на Таволгу выполняется из recovery системы на основе Linux, установленной производителем. При её загрузке за-

пускается интерфейсная программа, похожая на BIOS x86. Образ существует в виде файла recovery.tar определённого формата. Он включает в себя тарбол с rootfs, скрипт прошивки recovery.rc, текстовый файл с приглашением к прошивке, отображаемый в интерфейсе.

Прошивка может быть выполнена с USB флешки. Также может быть сделано восстановление ПО Таволги с файла recovery.tar, записанного на SSD при прошивке.

Если прошивка осуществляется с флешки, скрипт recovery.rc форматирует SSD, разбивает его на разделы, разворачивает тарбол с rootfs и создаёт squashfs с копией recovery.tar. При восстановлении с SSD форматирование, разбивка на разделы и создание squashfs пропускается. [2]

Файл для прошивки Alt на BFK 3.1 представляет собой образ диска. Запись образа на диск производится после загрузки в Embedded Linux командами zcat и dd [3].

Для BFK 3.1 имеется видеокарта на основе графического процессора SM750. Запуск графики с использованием этой видеокарты возможен, однако после запуска текст в текстовых консолях становится не читаемым и восстанавливается только аппаратной перезагрузкой. Также на экране не отображается правая часть картинки — видна широкая чёрная полоса.

Литература

[1] ALT Linux Team, Использование «Таволга Терминал» 2BT1 с клиентами удалённого рабочего стола, 2018, `https://www.altlinux.org/Ports/mipsel/TavolgaRemoteDesktopHowto`

[2] ALT Linux Team, Прошивка образа Alt на «Таволга Терминал» 2BT1, 2018, `https://www.altlinux.org/Ports/mipsel/TavolgaHowto`

[3] ALT Linux Team, Прошивка образа Alt на BFK3, 2018, `https://www.altlinux.org/Ports/mipsel/BFK3Howto`

Роман Ставцев
Москва, Зеленоград, АО «БАЙКАЛ ЭЛЕКТРОНИКС»

Проект: Комплект средств разработки ПО (SDK) для BE-T1000
https://www.baikalelectronics.ru/products/T1/

Процессор Байкал-T1. Программное и аппаратное окружение

Аннотация

Краткий обзор СнК Байкал-T1, оценочной платы BFK3.1 и комплекта средств разработки ПО SDK (BE–T1000) на основе СПО.

BE–T1000

Микропроцессор BE–T1000, другое название Байкал–T1 относится к типу Система-на-кристалле. Микропроцессор (МП) содержит два ядра MIPS32r5 P5600 Warrior. Тактовая частота ядра в МП составляет 1,2 ГГц. Перечислим некоторые основные параметры ядра:

- Адресное пространство расширено до 4 Тбайт (40–бит адрес);
- 16-ступенчатый конвейер с выборкой 4 команд за 1 цикл;
- Арифметический сопроцессор с блоком SIMD для операции с 32 128-бит регистрами;
  - Размерности векторов в регистрах: 8×16 бит, или 16×8 бит, или 4 ×32 бит, или 2 ×64 бит;
  - Операции с 8–, 16– и 32–бит целыми числами;
  - Операции с 16–, 32– и 64–бит с числами с плавающей запятой, стандарт IEEE–754;
  - Тактируется частотой ядра;
- Выполнение за один цикл 4 команд с целыми числами и 2 операций SIMD;
- 64-бит кэш команд;
- 64-бит кэш данных;
- Прогнозирование ветвлений;
- Поддерживается технология аппаратной виртуализации;

  - Обеспечивает уровни привилегии для гостевой и корневой ОС;
  - Поддерживает до 15 гостевых ОС;
  - Поддержка буфера ассоциативной трансляции TLB и контекста сопроцессора COP0 для гостевой и корневой ОС. Полная изоляция ОС друг от друга;
- Программируемый блок управления памятью;
  - Буфер TLB 1 уровня, 16 записей команд ввода 32 записей данных;
  - Буфер TLB 2 уровня, одновременный доступ, фиксированные и переменные размеры страниц;
  - Буфер VTLB, 512×2 записей.

МП производится по 28–нм технологии на тайваньской фабрике TSMC. Рабочий диапазон температуры МП подтверждён испытаниями в пределах 0–70°C, по расчётным данным диапазон рабочих температур может достигать –45. . . 70°C. МП выпускается в 578–выводном BGA корпусе размером 25×25 мм. Энергопотребление не превышает 5 Вт.

В каждом процессорном ядре МП содержит 64–Кбайт кэш данных и 64–Кбайт кэш команд. На кластер из двух ядер приходится 8-канальный ассоциативный кэш L2 ёмкостью 1 Мбайт. Микропроцессор снабжён универсальным контроллером памяти SDRAM. Основные параметры контроллера:

- Соответствует спецификации JEDEC DDR3 SDRAM Specifications JESD79–3E;
- DDR3–1600, рабочая частота памяти составляет 800 МГц;
- Поддерживает объём памяти SDRAM до 8 Гбайт;
- Внешняя шина шириной 32 бита с 8-битным кодом исправления ошибок;
- Поддерживаются микросхемы SDRAM с шириной шины данных 8 бит и 16 бит;
- Поддерживаются два ранга памяти.

Интерфейсы МП разделяются на низкоскоростные и высокоскоростные. Перечислим состав:

- Низкоскоростные интерфейсы:

  - 32–бит порт GPIO;
  - 3–бит порт GPIO;
  - 2 порта UART;
  - 3 порта SPI;
  - 2 порта I2C;
- Высокоскоростные интерфейсы:
  - 10Gb Ethernet;
  - 2 GMAC RGMII;
  - PCIe Gen3. x4;
  - 2 SATA 6G;
  - 1 USB 2.0 host.

BFK3.1

Компанией для собственных нужд были разработаны и произведены блоки функционального контроля (БФК). Которые трансформировались отладочные платы/оценочные платы для заказчиков. Ранее выпускавшийся тестовый комплект ТК–Т1(БФК–1.6) в формате microATX (244×244 мм), сменила оценочная плата BFK3.1 в формате FlexATX (229×191 мм). На плате установлен МП и реализован доступ ко всем низкоскоростным и высокоскоростным интерфейсам.

Комплект средств разработки ПО (SDK)

Комплект средств разработки программного обеспечения (далее SDK) полностью создан на базе СПО. SDK содержит кросс-компилятор языков C и C++, редактор связей, отладчик, утилиты и системные библиотеки, достаточные для разработки системного и прикладного ПО, исполняемого на микропроцессором. В состав SDK входит минимальная дистрибуция ОС Linux для целевой платформы с микропроцессором Байкал-Т1 (на основе системной утилиты busybox). SDK поддерживает следующие целевые платформы:

- QEMU MIPS32el
- Тестовый комплект ТК–Т1 (БФК–1.6)
- Тестовый комплект ТК–Т1+ (БФК–1.6+)
- Оценочная плата BFK3.1

Колотников Алексей Владимирович
Москва, Байкал Электроникс

Проект: Байкал T1 https://www.baikalelectronics.ru/

Оптимизация шифрования на Байкал-T1 по ГОСТ28147-89

Аннотация

Особенности применения процессора Байкал-T1 для шифрования. Результаты на задаче IPsec, которые достижимы для этого решения.

Обзор

Таблица 1: 3 варианта шифрования

Алгоритм	на CPU и регистрах общего назначения	с применением векторного сопроцессора MSA	модулярный сопроцессор
Скорость	2 * 150 Mbit/s	2 * 350 Mbit/s	2500 Mbit/s
Минимальное время подготовки для начала шифрования	0 — без задержки	0 — без задержки	5мкс
Время шифрования блока	~450 нс	~180 нс	~20 нс

Важно: построение шифрующего маршрутизатора потребует использования всех алгоритмов для разных размеров шифруемого пакета.

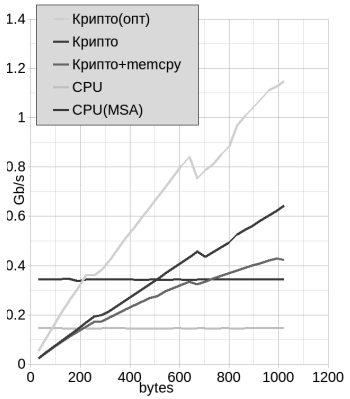
Шифрование на CPU и регистрах общего назначения

Шифруется 1 блок размером 64бит.

Время шифрования 1 блока по 64 бит — 450 ns

Важно: Наиболее гибкий подход, позволяет реализовать любой вариант зацепления(CTR, CBC и тд.), но достаточно медленный.

Скорость модели **~150 Mbit/s на ядро**. (Байкал T — 2 ядра.)



Шифрование с применением векторного сопроцессора MSA

Шифруется 4 блока по 64 бит.

Время шифрования 4 блоков по 64 бит — 720 ns

Скорость модели **˜350Mbit/s на ядро.** (Байкал T — 2 ядра.)

Важно: Работает для данных без зацепления (ECB, CTR)

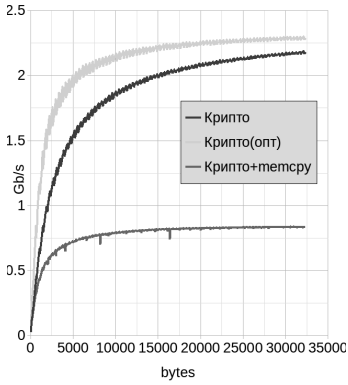
Прим. Есть некоторые сложности с использованием MSA в ядре линукс т. к. текущая реализация предполагает, что использование MSA возможно только для некоторых процессов пользовательского уровня

Решения:

1. полностью отключать поддержку MSA+FPU и использовать MSA только для шифрования
2. включить поддержку для всех процессов и всегда сохранять контекст MSA

Шифрование на модулярном сопроцессоре

Иллюстрация ниже показывает сильную зависимость скорости шифрования от размера пакета. т. к. в сетевом трафике много пакетов небольшого размера (64байт) ,то использование только модулярного сопроцессора не даёт хороших результатов.



Тестирование туннеля

Iperf режим: TCP/IP

IPsec ESP (MTU=1500)

Модельный пример для проверки концепта смешанного алгоритма шифрования с зависимостью от размера пакета.

Частота CPU 1.2GHz

ECB

Пакет менее 128байт шифруется на CPU+MSA

Пакеты больше 128байт шифруются модулярным сопроцессором:

поток **231** Мбит/с

дуплекс **159+152** Мбит/с

Разнесение прерываний по ядрам даёт значительный прирост:

поток **282** Мбит/с

дуплекс **235+184** Мбит /с

Смесь 7*66+4*518+1*1450 дуплекс **109+109** Мбит/с

CTR

Пакет менее 128 байт шифруется на CPU+MSA

Пакеты больше 128 байт шифруются модулярным сопроцессором (счётчик готовится на CPU модулярный сопроцессор шифрует в режиме ECB):
поток **200** Мбит/с
дуплекс **115+101** Мбит/с
Разнесение прерываний по ядрам с отключённым модулярным сопроцессором:
поток **177** Мбит/с
дуплекс **166+126** Мбит/с
Смесь 7*66+4*518+1*1450 дуплекс **87+87** Мбит/с

CBC + Imito

Пакет менее 128 байт шифруется на CPU
Пакеты больше 128 байт шифруются модулярным сопроцессором:
поток **224** Мбит/с
дуплекс **181+108** Мбит/с
Разнесение прерываний по ядрам даёт значительный прирост:
поток **281** Мбит/с
дуплекс **216+188** Мбит/с
Смесь 7*66+4*518+1*1450 дуплекс **92+92** Мбит/с

Медведев Денис Леонидович
Москва, ООО "Базальт СПО"

Метаинформация репозиториев: хранение и полезный состав

Что такое метаинформация репозитория

Метаинформация репозитория — это информация, которая в общем случае в нем не содержится, но связана с его содержимым. Метаинформацию можно классифицировать по способу её задания, как данные:

- Предлагаемые разработчиком
- Предлагаемые мантейнером пакета
- Предлагаемые составителем репозитория

- Предлагаемые пользователями репозитория — составителями дистрибутивов
- Предлагаемые конечными пользователями

Где можно посмотреть часть метаинформации в настоящее время

- sisyphus.ru
- packages.altlinux.org
- appstream
- cve-manager

Они достаточно жесткие с точки зрения добавления метаинформации, считающейся важной для других вышеупомянутых категорий пользователей, кроме составителя репозитория.

Возможные виды метаинформации

- Описания применения пакета
- Участие пакета в составе образов
- Сведения о безопасности пакета, закрытии CVE
- Рейтинг качества, безопасности пакета
- Дополнительная информация о способах сборки пакета, замечания.
- Причины удаления или изменения функциональности (например, выключения сборки документации)
- Ссылки на зависимые логически (не в смысле зависимостей по репозиторию) пакеты. пакета из репозитория
- Особые замечания
- Личные записи мантейнера/пользователей пакета/ссылки на форумы

Требования к метаинформации

- Интегрированность
  Информация должна быть актуальной, отражать текущее содержание репозитория.

- Изменяемость Пользователь системы метаинформации должен иметь возможность настроить ее содержимое по своему вкусу, а также иметь простую возможность оставлять комментарии.
- Конфиденциальность и доступность Информация, должна быть доступна постоянно, внутренняя информация пользователей должна защищаться согласно правам доступа. Возможно, что внутреннюю информацию надо хранить на отдельных системах.

Возможный подход к реализации

- Предлагается подключение WIKI к сборочнице `+git+packages.altlinux.org` или импорт в `altlinux.org`
- Также возможно создание ноды diaspora.

Алексей Турбин

`zrec` — формат метаданных репозитория

Аннотация

Новый формат сжатия записей обеспечивает высокий коэффициент сжатия и в то же время сохраняет возможность произвольного доступа к записям. Для этого поток записей разбивается на куски (chunking). Поддерживается синхронизация / частичное скачивание через HTTP range requests.

Введение

Перед обновлением из репозитория скачивается большой файл с *записями пакетов* (records); в `apt-rpm` это файлы `pkglist.xz` и `srclist.xz`, содержащие урезанные rpm-заголовки (в `repomd`-репозиториях записями можно считать сегменты XML-файла). Файл `pkglist.xz` хорошо сжат, но распаковка его занимает несколько секунд, и далее он хранится в `/var/lib/apt` в разжатом виде, занимая довольно много места (200–300 Мб; здесь не столько жалко места на диске, сколь возникают «тормоза» при доступе к файлу). Пережать же в более «лёгкий» формат `pkglist` нельзя, поскольку `apt` требует произвольный доступ к записям (команда `apt-cache show` делает `lseek(2)` и считывает запись).

В 2017 г. автор предпринял попытку разработать «лёгкий» формат `zpkglist` для сжатия записей после скачивания [1]. За основу была взята библиотека сжатия `LZ4`; для неё была адаптирована техника сжатия со словарём, реализованная в библиотеке `zstd`. Перед сжатием записи группировались по 4 штуки, что значительно улучшало коэффициент сжатия, однако произвольный доступ уже требовал распаковки группы из четырёх записей.

В мае 2018 г. Джонатан Дайетер (Jonathan Dieter) анонсировал похожий проект `zchunk` [2] (формат файла и библиотека сжатия). Поддержка `zchunk` добавлена в `dnf`, планируется к использованию в Fedora 29 либо Fedora 30. `zchunk` основывается на алгоритме `zstd`, который способен, в отличие от `LZ4`, обеспечить значительно более высокий, «релизный» уровень сжатия (хотя и несколько не дотягивает до `xz`). Поэтому интерес автора стал смещаться в сторону использования нового формата для сжатия на сервере (сжатый файл не требует распаковки на клиенте).

Файл в формате `zchunk` можно понимать просто как сжатый поток байтов. При сжатии поток нарезается на куски (chunking) и каждый кусок сжимается хотя и отдельно, но *со словарём*, хранящимся в том же файле (что значительно улучшает коэффициент сжатия). Естественно, при сжатии записей нарезка идёт по границам записей, но в одном куске может содержаться и несколько записей. В начале же файла хранится *индекс* кусков: их размеры и хеш-суммы. Наличие индекса делает возможным *синхронизацию*: клиент сначала скачивает индекс и потом — через HTTP range requests — недостающие куски, перестраивая старый файл в новый.

Формат `zchunk` не устраивает автора лишь в деталях. Далее рассмотрены некоторые специальные особенности сжатия и синхронизации, которые приводят к созданию альтернативного формата — `zrec` (сокр. от compressed records).

О пользе сортировки

Главным просчётом в формате `zchunk` является его общность: он сжимает какие-то куски, которые распаковываются во что угодно. Формат `zrec` вместо этого постулирует, что сжимаемые записи *упорядочены* каким-либо образом (напр., естественным образом — по имени пакета, но возможна и группировка по `src.rpm`). Этот постулат имеет далеко идущие последствия.

Во-первых, в отсортированном списке у записей возникает *локальное сходство*, которое «не улавливается» словарём. Например, пакеты `perl-*` имеют между собой сходство в части зависимостей и т. п. Значит, группировка соседних записей при нарезке должна привести к значительному улучшению сжатия. А библиотека тогда может реализовать процедуру сжатия с автоматической группировкой.

Во-вторых, сортировка меняет требования к сопоставлению кусков. Это даже две разные математические задачи: 1) чтобы уникально идентифицировать каждый кусок среди всех остальных, требуется 96-битный хеш (при числе кусков порядка 10^5 и вероятности коллизии порядка 10^{-18}); 2) чтобы определить, изменился ли кусок, требуется 60 битов хеш-материала (т. к. $2^{-60} \approx 10^{-18}$). Поэтому можно считать, что когда куски «расставлены по местам», то для идентификации куска в «локальной окрестности» достаточно 64-битного хеша.

Группировка записей

Группировка записей может заметно, хотя и не радикально, улучшить сжатие. Возьмём файл `srclist.xz` (1.86 Мб) — в разжатом виде 10 Мб. «Солидное» сжатие `zstd -19` даёт 1.97 Мб. Если сжимать каждую запись по отдельности, получается 5.8 Мб — мало избыточности! Радикальное улучшение даёт сжатие записей со словарём — 2.44 Мб (словарь 512 Кб, в сжатом виде 177 Кб). Если теперь ещё группировать записи парами, получается 2.26 Мб, по три — 2.21 Мб, по четыре — 2.18 Мб и далее очень медленно. В общем, группировка по 2-3 записи улучшает сжатие на 8-10%, а также уменьшает в 2-3 раза размер индекса.

Записи однако нельзя группировать в ровных количествах, это приведёт к *сдвигу границ* и сделает невозможным синхронизацию. Так, если из потока записей, сгруппированных парами: `AB CD EF...` будет удалена запись `B`, то границы пар сдвинутся: `AC DE F...` и все куски перестанут совпадать. Классический подход к решению этой проблемы состоит в том, что нужно нарезать куски переменной длины псевдослучайным образом, основываясь на данных внутри кусков (content-defined chunking). А именно, данные сканируются скользящим окном и нарезаются, когда контрольная сумма в окне принимает определённое значение. Упрощённая реализация такого подхода известна как rsyncable gzip. Об оптимальной нарезке см. статью 2005 г. [3] и свежую работу [4].

Наша идея *ультракороткой оптимальной нарезки* состоит в следующем: записи можно сравнивать по их хеш-коду. Тогда если напр. $\underline{A < B < C} > \underline{D < E}$, то упорядоченная подпоследовательность и образует группу, а нарушение порядка даёт разрез. Нарушение порядка можно также вынужденно допустить в первых двух элементах группы. Это приводит к следующей **Стратегии**: 1) Поместить в очередь A, B и C. Если $B > C$, отрезать AB (и тогда C становится новым A). 2) Иначе добавить в очередь D. Если $C > D$, отрезать ABC. 3) Иначе отрезать $ABCD$.

Отметим, что хотя на шагах 1) и 2) сравнение одно и то же, вероятность его разная: $\Pr(B < C) = 1/2$, а $\Pr(C < D) < 1/2$. И поскольку условие одно и то же, то для преодоления рассинхронизации даётся более одной возможности «зацепиться за нужное место», и в то же время рассинхронизация быстро пресекается.

Отметим также важную *практическую модификацию* процедуры нарезки: для использования в качестве $A, B \ldots$ желательно хешировать не всю запись, а лишь имя пакета (или имя `src.rpm` пакета без версии при группировке по `src.rpm`). Тогда изменение версии у пакета не приводит к изменению нарезки: изменение оказывается полностью локализованным внутри куска; к рассинхронизации может привести только удаление и добавление новых пакетов.

Мы взяли файл `srclist.xz` от 1 июня и 1 июля 2018 г. Без группировки записей для регенерации нового файла недостаёт 923 куска общим объёмом 271 Кб, а с описанной группировкой — 809 кусков объёмом 568 Кб. При этом группировка также уменьшает размер индекса примерно с 80 до 30 Кб, однако в данном случае увеличившийся объём кусков в несколько раз перекрывает уменьшение индекса. Видно, что при нерегулярных обновлениях группировка оказывается невыгодной.

Хеширование и расстановка

При любом изменения репозитория индекс будет скачиваться полностью, что может составлять значительную часть скачанного. Поэтому представляет интерес дальнейшее уменьшение объёма хеш-материала внутри индекса.

Выделим подзадачу: пусть каждый кусок наделяется 32-битным хешем; нужно установить соответствие между кусками в старом и новом файле. Ясно, что соответствие определяется совпадением хеша,

но при этом совпадения, нарушающие «общий порядок», считаются коллизиями и отсеиваются.

В общем виде эта задача известна как Longest common subsequence problem, и одним из её решений является алгоритм Майерса [5], используемый в `diff(1)`. Однако алгоритм Майерса выполняется, вообще говоря, квадратичное время, как и все остальные решения этой задаче в наиболее общем виде или в худшем случае. Несколько точнее, сложность алгоритма Майерса — $O(nd)$, где n — общее число элементов в двух последовательностях, а d — число отличающихся элементов. Увы, на больших файлах `diff(1)` хорошо работает, только если изменения в них оказываются небольшими. В нашем же случае при нерегулярных обновлениях число отличающихся кусков может быть большим. Однако существует другой алгоритм — алгоритм Ханта-Шиманского со сложностью $O((r+n)\log n)$, где r — общее число всевозможных совпадений в двух последовательностях. Этот алгоритм деградирует, когда число неоднозначных совпадений велико (например, из-за малой мощности алфавита, или когда в текстовых файлах много одинаковых строк). Но в нашем случае при общем количестве кусков порядка 2^{16} неоднозначными совпадениями (коллизиями 32-битного хеша) можно пренебречь, и задача расстановки решается за $O(n \log n)$. Позже были открыты другие алгоритмы с похожей асимптотикой, которые используются в биоинформатике [6, с. 291] (в биоинформатике задача расстановки называется global alignment). См. тж. обзоры [7, 8].

Естественно, 32-битного хеша маловато для надёжного сопоставления. Поэтому формат вводит хеширование второго уровня: пусть $h(A)$ и $h(B)$ — хеши идущих подряд кусков, тогда в дополнение к ним вычисляется $H(A+B)$ — 32-битный «проверочный» хеш. Важно, что функции h и H отличаются, т. е. H предоставляет *дополнительную* информацию о каждом куске. Тогда каждый кусок оказывается наделённым 64-битным хешем (поскольку с точки зрения каждого куска другой кусок можно считать параметром). А составная конструкция содержит 96 битов хеш-материала, что уже достаточно для уникальной идентификации. Проверочный хеш используется на второй стадии синхронизации: если $h(A)$ и $h(B)$ совпадают, а $H(A+B)$ не совпадает, значит, куски A и B надо скачать повторно.

Теперь остаётся только применить тот же алгоритм группировки кусков, который применяется для группировки записей: для наделения кусков составным хешем их нужно нарезать по 2-4 штуки. Сов-

падение составных кусков разбивает задачу расстановки на мелкие подзадачи.

О надёжности конструкции

Таким образом, мы получили, что для достаточно надёжного сопоставления в индексе требуется всего около 5.5 байтов хеш-материала на кусок, или же около 2 байтов на запись. Конечно, такую конструкцию нельзя считать криптографически безопасной. Криптографическую проверку мы просто делегируем на уровень выше: когда файл перестроен, клиент проверяет криптографическую сумму перестроенного файла. В худшем случае, когда сумма не совпадает, остаётся только скачать весь файл заново.

Более вероятным представляется несовпадение из-за того, что клиент скачал разные версии `pkglist` и `release` файлов. Решением является включение в `release` не только суммы всего `pkglist.zrec` файла, но и отдельно суммы его индекса, чтобы такое несовпадение можно было бы сразу же обнаружить.

Литература

[1] Alexey Tourbin. `zpkglist` — Compressed file format
`https://github.com/svpv/zpkglist`

[2] Jonathan Dieter. What is `zchunk`?
`https://www.jdieter.net/posts/2018/05/31/what-is-zchunk/`

[3] Kave Eshghi, Hsiu Khuern Tang (2005). A Framework for Analyzing and Improving Content-Based Chunking Algorithms

[4] Wen Xia et al. (2016). FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication

[5] Eugene W. Myers (1986). An O(ND) Difference Algorithm and Its Variations

[6] Dan Gusfield (1997). Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology

[7] L. Bergroth et al. (2000). A Survey of Longest Common Subsequence Algorithms

[8] Mike Paterson, Vlado Dancik (1994). Longest Common Subsequences

Научное издание

ПЯТНАДЦАТАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ СВОБОДНЫХ ПРОГРАММ

Тезисы докладов

Ответственный редактор В.Л. Черный

Подготовка оригинал-макета: «ООО Базальт СПО»,
Оформление обложки: А. С. Осмоловская
Вёрстка: В. Л. Черный

Подписано в печать 24.09.2018 г.
Формат 60x90/16. Усл.печ.л. 7,25. Тираж 150 экз. Изд. № 194.
Издательство ООО «МАКС Пресс»
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.

Отпечатано с готового оригинал-макета
в ООО «Грин Принтер»
105064, Москва, Нижний Сусальный переулок, д. 5, стр. 10
Заказ №