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

Четырнадцатая конференция разработчиков свободных программ

Калуга, 22–24 сентября 2017 года

Тезисы докладов

Москва
Базальт СПО

УДК 004.91
ББК 32.97
Ч-54

Ч-54 Четырнадцатая конференция разработчиков свободных программ: Тезисы докладов. Калуга, 22–24 сентября 2017 года / отв. ред. Черный В. Л. — М.: Базальт СПО, 2017. — 130 с. : ил.

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

ISBN 978-5-9908979-3-9

Программа конференции

22 сентября, пятница

Вечернее заседание
14.30–18.50

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

14.45–15.15 Половинский В. В., Министерство экономического развития Калужской области

Вопросы внедрения свободного программного обеспечения отечественных разработчиков в органах исполнительной власти Калужской области 8

15.15–15.45 Якушин А. А.

Мюнхенский проект. Победа, которую объявили поражением 11

15.45–16.15 Мидюков А. П., АО «ЕВРАЗ ЗСМК»

Свободное ПО для промышленности 15

16.15–16.45 Израйлев И. А., ЛАНИТ-Урал

Комплексная методология проведения проектов миграции с инфраструктуры Microsoft — практика организации проектных работ и использования технологических компонентов 18

16.45–17.05 Кофе-пауза

17.05–17.35 Липатов В. А., ООО «Этерсофт», Никулин Д. В., ГУАП

.NET Core 2.0 как универсальная платформа разработки. Миграция с .NET Framework: среда разработки и альтернативные средства создания графического интерфейса 24

23 сентября, суббота

Утреннее заседание
10.00–14.10

Дневное заседание
15.30–18.20

18.20–18.40 Сыромятников Е., Red Hat Czech s. r. o.

Некоторые аспекты разработки и пакетирования out-of-tree модулей Linux 86

24 сентября, воскресенье

Утреннее заседание
10.00–14.30

10.00–10.30 Боковой А. Г., Red Hat

Перенос FreeIPA на Python 3 или как мы танцевали Samba. Поддержка смарт-карт во FreeIPA 4.5 90

10.30–11.00 Державин Д. К., Базальт СПО

Поддержка информационных систем с квалифицированной электронной подписью в ОС Альт 94

11.00–11.30 Волнейкин П. А., Базальт СПО

Аутентификация по аппаратным токенам в дистрибутивах Альт 97

11.30–12.00 Белявский Д. М., Технический Центр Интернет

ГОСТ в OpenSSL: 12 лет международного взаимодействия . 105

12.00–12.40 Кофе-пауза

12.40–13.10 Макаренко Б. В., ФССП России

Криптографические операции в окружении рабочего стола ОС «Гослинукс» 106

13.10–13.30 Матвеев Д. А., Электронные Офисные Системы

Система криптографического обеспечения «КАРМА» — Универсальная система для криптографической защиты информации и применения электронной подписи (ЭП) для любых приложений 109

13.30–14.00 Селедкин А. Е., Цифровые технологии

Разработка средства электронной подписи Trusted eSign на базе СПО 110

Вне программы

Половинский Владимир Викторович
г. Калуга, Министерство экономического развития Калужской области

Вопросы внедрения свободного программного обеспечения российских разработчиков в органах исполнительной власти Калужской области

Аннотация

В докладе рассматриваются вопросы, сдерживающие внедрение свободного ПО в органах государственной власти Калужской области, приводится положительный пример перевода организации на использование отечественного ПО, включенного в Единый реестр российских программ для электронных вычислительных машин и баз данных

1. Что мешает внедрению свободного ПО в государственных органах

Главная проблема сегодня состоит в том, что за предыдущие годы органы власти всех уровней слишком глубоко успели интегрировать в свои информационные системы проприетарное ПО.

Например, крайне сложной для разработчиков ПО представляется задача миграции инфраструктуры госорганов на Linux-платформу, которая чаще всего строится на почтовых серверах Microsoft Exchange и службах каталогов Microsoft для операционных систем семейства Windows Server — Active Directory (AD). Их миграция на аналоги, создаваемые по модели Open Sourse, крайне затруднительна и трудоемка.

Эту проблему можно классифицировать, как внутреннюю, которую органы власти могут решить самостоятельно, либо с привлечением специалистов разработчиков СПО в качестве методологов, консультантов или исполнителей.

Но есть проблемы «внешние», которые самостоятельно ни региональные, ни муниципальные органы власти решить не в состоянии. Это проблемы, связанные с использованием свободного ПО при взаимодействии с существующими федеральными государственными информационными системами (т.н. ГИС-ами), построенными федеральными ведомствами и организациями, использующими проприетарное

ПО. Зачастую такие ГИС в обязательном порядке требуют наличие установленного на рабочих местах пользователей проприетарного ПО. В частности, речь идет о системе закупок по 44-му Федеральному закону. Среда эмуляции операционной системы Windows — WINE не в полной мере решает данную проблему.

В данном случае разработчики свободного ПО подчеркивают, что большая часть проприетарного ПО (например, среда ActiveX, определяющая структуру программной системы) ориентирована на работу с браузером Internet Explorer, не поддерживаемым ни в каких современных операционных системах, кроме Windows.

Еще хочется остановиться на одном классе проблем использования СПО в госорганах, это — проблемы совместимости офисных программ. Возникают они из-за разнородности используемых в органах власти всех уровней форматов документов. В Российской Федерации принят ГОСТ Р ИСО/МЭК 26300-2010 (Информационная технология. Формат Open Document для офисных приложений (OpenDocument) v1.0). Open Document основан на XML и позиционируется как альтернатива закрытым форматам, таким как DOC, XLS, PPT и Microsoft Office Open XML, однако он редко используется. В качестве положительного примера использования данного формата хочется привести городской округ «Город Калуга», где в первой половине 2016 года в соответствии с распоряжением Городского главы все муниципальные организации и учреждения были переведены на использование офисного пакета LibreOffice и работают с ним до настоящего времени.

В основном же, для обмена информацией используются закрытые форматы, из-за чего затруднен, а за частую невозможен, документооборот между различными органами в силу программных различий в поддержке шрифтов, которые по-разному отображаются в разных средах. Негативный эффект еще больше усугубляется при сложной верстке документов — с использованием таблиц, диаграмм и макросов, работающих только в среде Microsoft Office.

2. Что сделано

Проанализировав все данные факторы нами было принято решение о начале внедрения свободного ПО в организации, минимально нуждающейся в интеграции с внешними информационными системами.

Для обеспечения функционирования данной организации требовалась организация новых рабочих мест. Мы остановили свой выбор на операционной системе ALT Linux по нескольким причинам.

Во-первых, на тот момент (середина 2016 года) линейка продуктов операционных систем ALT Linux была самая широкая из операционных систем, включенных в Единый реестр российских программ для электронных вычислительных машин и баз данных, опубликованный на сайте Минкомсвязи России, т.н. реестр отечественного ПО.

Во-вторых, седьмая версия операционной системы ALT Linux была полностью бесплатная.

В-третьих, относительная простота в установке и настройке системы, принципиально отличающейся от ОС Windows. В условиях, когда все системные администраторы обучены и имеют опыт работы только с линейкой операционных систем Windows, данный фактор для нас был крайне важен.

Работы по организации автоматизированных рабочих мест на базе ОС ALT Linux были начаты в сентябре 2016 года и успешно завершены в октябре 2016 года вводом на объекте заказчика рабочих мест в промышленную эксплуатацию.

Успех перевода целой организации на альтернативное программное обеспечение российского производителя был обеспечен благодаря:

- во-первых, тому, что данной работе предшествовала очень кропотливая работа по выбору и изучению свободного ПО, которая проводилась с начала 2016 года и продолжается по сегодняшний день, в том числе с обучением специалистов на специализированных курсах, организуемых разработчиками свободного ПО;
- во-вторых, тому, что была правильно выбрана организация, минимально нуждающаяся в интеграции с внешними информационными ресурсами. Требовалось только обеспечить работу в сети Интернет, с корпоративной почтой, доступ к корпоративным справочно-правовым системам и установить пакет офисных программ.

3. Что в будущем

Задачи:

- продолжение обучения системных администраторов администрированию СПО, а в дальнейшем и пользователей программ;

- продолжение изучения линейки пакетов офисного программного обеспечения, уже созданного и создаваемого отечественными разработчиками, с целью выбора наиболее оптимального для внедрения в органах власти области;
- изучение новых программных продуктов, вносимых в Единый реестр российских программ для электронных вычислительных машин и баз данных, который ведет Минкомсвязь России, с целью применения их в работе сотрудниками органов власти региона;
- перевод структурных подразделений органов власти Калужской области, минимально нуждающихся в интеграции с внешними информационными ресурсами, на свободное ПО российских разработчиков.
- ожидание от федерального центра разработки методологических материалов по переводу региональных органов власти и органов местного самоуправления на свободное ПО.

Якушин Анатолий
Москва

Мюнхенский проект. Победа, которую объявили поражением

Аннотация

В докладе рассматривается анализ проекта по информатизации муниципалитета Мюнхена с момента его старта до сегодняшнего дня. Исследуются основные особенности реализации, истинные и мнимые недостатки проекта, типичные ошибки и трудности при массовом внедрении продуктов со свободными лицензиями.

Под термином Мюнхенский проект в современной аналитической литературе принято понимать масштабный проект по информатизации подразделений г. Мюнхен (Бавария, Федеративная Республика Германия). Другим распространенным наименованием является LiMux, по названию дистрибутива GNU Linux, разработанного в рамках данного проекта.

Проект по информатизации Мюнхена был широко анонсирован в 2003 году, однако его практическая реализация началась несколько

позже. В 2013 году муниципалитет и разработчики проекта объявили о его успешной реализации.

Поэтому сообщество свободных разработчиков с большим удивлением отнеслось к массовым публикациям, которые стали появляться в прессе с февраля 2017 года, в которых говорилось о крахе проекта, полном провале, отказе от предыдущих разработок в манере эпохи войн операционных систем.

Следует отметить, что со времени первых анонсов о Мюнхенском проекте, его реализация находилась под пристальным вниманием всех заинтересованных разработчиков, ввиду значимости проекта и его масштабности. Несоответствие публикаций 2017 года информации последних пятнадцать лет заставило автора вновь проанализировать историю реализации проекта и его нынешнее состояние.

Как показал проведенный анализ, слухи о крахе проекта сильно преувеличены. В полном соответствии с заявлениями разработчиков, свободное ПО установлено на 15 тысячах рабочих мест и исправно функционирует, обеспечивая инфраструктуру огромного города.

В дальнейшем планировалось создать новую web-ориентированную версию, однако в планы информатизации вмешалась большая политика. В 2014 году Кристиана Удэ, который занимал пост мэра Мюнхена с 1993 года сменил Дитер Рейтер, который не скрывает того, что является давним приверженцем Microsoft. Одним из первых мероприятий нового мэра была широкомасштабная поддержка по переезду европейской штаб-квартиры Microsoft в Мюнхен.

Несмотря на столь мощный ресурс сторонников проприетарного ПО городской совет в марте 2017 года принял весьма осторожное решение, допускающее использование продукции Microsoft на рабочих местах муниципалитета, и не более того. Нового клиента планируется создать к 2020-2022 годам, и каким он будет, покажет время.

Проведенный анализ позволил выявить типичные проблемы, возникающие при реализации масштабныз проектов, даже вне зависимости от лицензий ПО, применяемого при реализации. Из уважения к титаническому труду разработчиков, нет желания называть эти проблемы ошибками. В очень многих вопросах команда Мюнхенского проекта была первопроходцем, а наша задача увидеть эти проблемы и найти пути их решения.

На какие проблемы стоит обратить внимание, тем более, что они повторяются и при реализации других крупных проектов со свободными лицензиями.

Крайне высокая степень политизированности проекта. Разработчики с самого начала очень много говорили о своеобразии своего проекта, использовании свободного ПО, отказе от проприетарных форматов данных. Им рукоплескали сторонники по всему миру, однако собственно пользователей проекта подобная шумиха скорее пугала, о чем есть немало публикаций. Особый статус в некоторое моменты приводил к полному неприятию со стороны пользователей и жалобам на недостатки системы, хотя имевшие место проблемы никак с выбранным ПО связаны не были.

Отсутствие поддержки со стороны конечных пользователей. Следствием высокой политизированности стало отсутствие адекватной поддержки со стороны конечных пользователей. Разработчики не сумели наладить с ними полноценный диалог, создать краудсорсинговые группы из наиболее продвинутых пользователей, склонить их на свою сторону при разработке и внедрении.

Заниженная стоимость проекта. Изначально широковещательно объявлялось о рекордно низкой стоимость проекта, за счет бесплатности свободного ПО. Данный прием широко использовался на рубеже 2000 годов для пропаганды внедрения СПО, однако уже тогда было понятно, что стоимость проекта на СПО ниже проекта на проприетарном программном обеспечении только на стоимость лицензий. И это в самом идеальном варианте. Реально стоимость свободного проекта может быть весьма высока, хотя бы ввиду того, что требует найма высокооплачиваемых специалистов. Стремление к экономии заставило разработчиков принять машинный парк as is, что в последующем существенно замедлило работу по внедрению ввиду огромного парка устаревшей техники.

Отсутствие договора на реинжениринг. Как показывает описание IT-инфраструктуры Мюнхена перед стартом проекта, эта городская служба представляла собой мешанину из разных систем, каждый отдел муниципалитета имел свой IT-отдел, поддержку оказывало три различных, конкурирующих между собой фирмы и т.п. Разработчики изначально предполагали проведение реинжениринга бизнес-процессов города, провели колоссальную работу по фиксации имеющегося городского хозяйства и разработке плана по его модернизации, но в конечном итоге так и не смогли добиться от городского совета реализации предварительных договоренностей. В итоге систему внедряли на имеющихся структурах, что существенно сказалось на итоговом качестве.

Создание собственного дистрибутива. Созданный в рамках проекта дистрибутив LiMux так много описывался в прессе, что создается впечатление, что его создание было конечной целью проекта. В реальности, разработанный дистрибутив является только платформой. Собственно проект базируется на разработанном программном продукте WollMux — java приложении, работающим через UNO с OpenOffice/LibreOffice. Следует отметить, что WollMux является кроссплатформенным и прекрасно работает на любой современной операционной системе. Таким образом создание собственного дистрибутива в какой-то момент стало самоцелью, отнимающей огромное количество времени. Следует признать, что в начале 2000-х подобный подход был общепринятым, оригинальные дистрибутивы создавались по любому поводу. Гораздо позже стало очевидным, что полноценная поддержка дистрибутива на всех этапах жизненного цикла, это весьма нетривиальная техническая задача, решить которую успешно может специальный коллектив, ориентированный только на создание платформ. И совместить полноценное дистрибутивостроение с разработкой и поддержкой прикладного ПО практически невозможно.

Сроки проекта. Все перечисленные и множество других проблем привели к тому, что реализация проекта затянулась на сроки, вдвое превышая обещанные. Это несомненно сказалось на реализации ожиданий и участников и заказчиков и разработчиков, во многом породив негативное отношение к проекту в IT-индустрии, причем по большей части негатива, реально неоправданного.

В итоге следует отметить, что авторы Мюнхенского проекта доказали, что сколь угодно крупная инфраструктура может быть реализована только с использованием свободного программного обеспечения и результат как минимум не будет хуже, чем при использовании проприетарного ПО.

Антон Мидюков
Прокопьевск, АО «ЕВРАЗ ЗСМК»

Свободное ПО для промышленности

Аннотация

Промышленность нуждается в средствах автоматизированного проектирования (CAD), как механических устройств (MCAD), так и электронных устройств (EDA), средствах технологической подготовки производства изделий (CAM), и наконец числовом программном управлением (ЧПУ) для станков, чтобы эти изделия изготовить. Также необходимо ПО для автоматизации систем управления технологических процессов, состоящее из систем верхнего уровня — диспетчерское управление и сбор данных (SCADA) и нижнего уровня — программируемые логические контроллеры (PLC). Использование свободного ПО в промышленности позволит уйти от привязки к поставщику и в целом повысить уровень безопасности предприятий. В этом докладе будет рассказано о том, какое свободное ПО есть для решения выше обозначенных задач. В конце доклада участникам конференции будет предложен демонстрационный Live с ПО, упомянутым в докладе.

Часть 1 — От чертежа к готовому изделию

Этапы создания нового изделия:

1. Проектирование 2-мерной или 3-мерной модели.
2. Моделирование изготовления, его оптимизация и отладка.
3. Изготовление на станке ЧПУ или 3D принтере.

Каждому этапу соответствует свой класс программного обеспечения:

1. CAD (англ. computer-aided design/drafting) — средства автоматизированного проектирования (САПР). CAD программы разделяются на:
   a) MCAD (англ. Mechanical Computer-Aided Design) — механическое проектирование, которые в свою очередь предназначаются для 2D или 3D проектирования.
   b) EDA (англ. electronic design automation) — проектирование электронных устройств.

2. CAM (англ. Computer-aided manufacturing) — средства моделирования и отладки изготовления изделия.
3. CNC (англ. computer numerical control) — числовое программное управление (ЧПУ) станком или 3D принтером.

Для всех этапов создания изделия существует свободное программное обеспечение.

LibreCAD — САПР для 2-мерного черчения и проектирования, создана на основе Qcad.

Inkscape — векторный графический редактор, который также может использоваться для создания 2D моделей. Есть плагин gcodetools для генерации g-code.

FreeCAD — параметрический САПР, использующий подход верстаков. Инструменты сгруппированы в верстаки в соответствии с задачами, к которым они относятся.

Meshlab — система редактирования 3D моделей, полученных в результате 3D-сканирования. Используется также для упрощения 3D моделей.

Проектирование электронных устройств:

Qucs — программа моделирования электрических цепей

KiCAD — САПР электронных устройств для всех этапов проектирования электронного устройства от электронной схемы до создания gerb-файла и 3D модели готовой печатной платы.

За этапом проектирования следует этап моделирования, оптимизации и отладки процесса изготовления изделия.

FlatCAM — система моделирования изготовления печатной платы, позволяющая генерировать и отлаживать g-code для гравировального станка.

PyCAM — система моделирования изготовления 3D деталей, поддерживается 3-оси.

CAMotics (бывший OpenSCAM) — симулятор g-code. Может использоваться как для симуляции изготовления 3D деталей, так и гравировки. Планируется добавить симуляцию 4 и 5 оси, что в будущем позволит использовать его для моделирования изготовления изделий на современных много-осевых станках.

Для 3D-принтеров вторым этапом является разделение 3D-модели на слои при помощи программ-слайсеров. На выбор есть Slic3r и Cura.

Заключительным этапом является отправка и исполнение g-code на станках ЧПУ и 3D-принтерах.

LinuxCNC (EMC2) — программное обеспечение для создания ЧПУ на базе компьютера. Поддерживается до 9 осей, есть возможность управления как шаговыми двигателями так сервоприводами, может быть использована для ЧПУ любого уровня сложности. Требует пропатченное ядро Linux и модуль RTAI (Real Time Application Interface).

Для 3D-принтеров заключительным этапом является отправка g-code в 3D-принтер при помощи программы printrun.

Часть 2 — Автоматизация производства снизу вверх

Автоматизированные системы управления технологическим процессами (АСУ ТП) состоят из верхнего и нижнего уровня. Верхний уровень — SCADA (от англ. Supervisory Control And Data Acquisition) — диспетчерское управление и сбор данных. Нижний уровень — ПЛК (программируемые логические контроллеры) и полевые устройства.

Для программирования ПЛК, отладки программы на компьютере, а также создания программных ПЛК существует открытый проект Beremiz. Beremiz — это интегрированная среда разработки для ПЛК с открытым исходным кодом, которая полностью соответствует стандарту МЭК-61131-3, позволяющая также создавать HMI (англ. Human-machine interface — человеко-машинный интерфейс).

Beremiz используется в качестве среды разработки и исполнения на ПЛК серии СМ1820М ПАО «ИНЭУМ им.И.С. Брука» на базе отечественных микропроцессоров «Эльбрус» и SPARC, а также линеек, основанных на микропроцессорах ARM и x86. Поддерживается ряд протоколов связи с системами верхнего уровня: Modbus TCP/RTU, IEC 104 и собственный протокол SM1820 TCP Protocol. Гибкая структура среды разработки и исполнения позволяет оперативно дорабатывать необходимые коммуникационные протоколы. [1]

Также beremiz используется в ПЛК следующих компаний:

- Smarteh (Slovenia/Tolmin)
- ООО НПК «Нуклерон» (Россия/Пермь)
- АО «Нефтеавтоматика» (Россия/Уфа)
- НТЦ «Арго» (Россия/Иваново)
- ООО «НГП Информ» (Россия/Уфа) [2]

Верхний уровень АСУ ТП представлен проектом OpenSCADA, который отличается модульной архитектурой, и практически неограни-

ченными возможностями по масштабированию. На демонстрационном Live представлены два демо-проекта:

- Динамическая модель технологического процесса парового котла.
- Динамическая модель технологического процесса АГЛКС.

Таким образом, на сегодняшний день существует свободное ПО для большинства потребностей промышленности за исключением CAM-систем для многоосевых станков.

Литература

[1] Система программирования Beremiz.
http://www.sm1820.ru/produktsiya/programmnoe-obespechenie/sistemy-programmirovaniya/item/3-beremi

[2] Beremiz Products. http://www.beremiz.org/apps

Израйлев Иван Алексеевич
Челябинск, ЛАНИТ-Урал
Проект: КИПАРИС URL проекта

Комплексная методология проведения проектов миграции с инфраструктуры Microsoft — практика организации проектных работ и использования технологических компонентов

Аннотация

Доклад посвящен методологии реализации проектов и миграции информационных систем с платформы Microsoft на свободную платформу.

Нам повезло быть свидетелями одного из самых масштабных изменений информационных технологий в России за всю историю — процесса перехода на свободную архитектуру. Большой вклад в интенсификацию этого процесса вносят инициативы государства вокруг преодоления критической зависимости от импортных проприентарных информационных систем. Сторонники свободных программ с разными чувствами наблюдают за этой историей, но ее влияние нельзя не отметить. При этом, тот внушительный объем нормативно-правовых

актов по этой тематике на всех уровнях государственного управления, который вышел за последние 4 года — к сожалению, не содержит практически никаких ответов на вопросы как конкретно осуществлять проекты перехода.

А ведь эти проекты чрезвычайно сложны — мы переходим на совершенно чуждые текущим технологические платформы, а объем изменений столь велик, что требуется серьезный подход к организации проектных работ.

Хотел представить Вашему вниманию комплексную методологию проведения таких проектов из 12-ти функциональных блоков, для проработки:

а) процессов управления на всех этапах жизненного цикла программных и аппаратных средств, б) специальных технологических компонентов без которых такой объем работ просто не осилить и в) задач управления ресурсами, прежде всего, человеческими.

Вначале мы делаем этап планирования — «Дорожную карту». Как может выглядеть структура документа:

1. Описание текущей ситуации в организации
2. Цели и задачи проекта миграции
3. Организация проектных работ
4. Анализ альтернатив в разрезе замещаемого ПО
5. Высокоуровневый план проектов миграции
6. Ожидаемые результаты миграции, критерии успеха и механизмы измерения

Здесь наиболее критичный блок — это наши цели. Они могут быть минимальные — например, обеспечить приемлемую работоспособность только вновь устанавливаемого свободного ПО без смены платформы. В других организациях цели могут быть более амбициозные — например полный переход на свободные технологии к 2020 году. Цели как обычно определяют все — и бюджеты которые будут нужны, и компетенции людей, и масштабы изменений.

Также на этом этапе проводится анализ альтернатив текущим технологиям на платформ Microsoft.

Следующий важный шаг после обоснования — это создание Руководящего документа «Стратегия миграции» с детализацией на среднесрочный период, где мы детально планируем весь портфель проектов. Структура документа может выглядеть так:

1. Обоснование, основные цели и задачи (разделы из «Дорожной карты»)
2. Детальный план проектов миграции (для каждого проекта):
   a) Организация проектных работ в разрезе этапов проектов:
      i. Содержание работ
      ii. Результаты
      iii. Ресурсы
      iv. Трудозатраты
      v. Длительность
      vi. Стоимость
   b) Расчет полной стоимости работ, спецификаций ПО и СВТ
   c) Детализация задач проекта в нотации Ганта
3. Организация процессов управления жизненным циклом ПО и СВТ в рамках стратегии миграции

Итак, 12 потоков работ по порядку (далее будет удобно возвращаться к иллюстрации процессной модели см. Рис. 1 для отслеживания логики):

Первый этап — это проработка инфраструктуры, которая нужна проекту — это, например, мощности для резервного копирования данных пользователей, тут обычно большие масштабы, ресурсы для документов, для технических средств и так далее — это каналы, мощности, объемы. Далее это изменения в текущих сервисах, например, нужно понять в какой домен войдут АРМы пользователей, если мы их отмигрируем на Linux, а сервера оставим на 2-й этап, как обеспечить совместимость с файловыми серверами в части прав пользователей и так далее.

Группа обеспечения должна не подставить остальных коллег и обеспечить проект техническим фундаментом.

Далее нужно разработать так называемый Стандартный образ операционной системы. Зачем? Дело в том, что если у Вас больше условно 150-ти компьютеров, то их физически невозможно оббежать — миграция может быть выполнена только в автоматическом режиме. И нужен не просто образ — а технология его сопровождения. Задайте себе вопрос сколько раз даже в простом проекте Вы будете собирать образа? Добавить драйвер — это новый образ, изменить настройку — новый образ, устранение ошибок на этапе тестирования —

Рис. 1: Двенадцать потоков работ

это сотня новых образов по 5–6 в день. Нужно построить машину по быстрой генерации образов и учету его состава, иначе вся история может сильно затянуться.

Далее, несмотря на стандартизацию образа, у нас много ПО стоит не у каждого пользователя. На практике — это более 70% всего ПО, поэтому это ПО нельзя включать в образ. А значит нам нужно создать механизмы управления таким ПО — репозитарии, упаковку пакетов для автоматического развертывания и так далее, и это крайне трудоемкий этап работ.

Вообще средства доставки ПО до АРМ — это сердце всего проекта, а потом — сердце процесса управления изменениями АРМ.

В результате в организации должен появится Репозитарий типового программного обеспечения, оформленный в виде стандарта и технологии.

Уверен, все осознают, что приличная часть программ по Windows на Linux не заработает. Более того, смею заявить, что даже при свер-

хусилиях в переходный период 3-5 лет ПО под платформу Майкрософт будет в каждой организации.

А значит нужно либо заставить такое ПО работать в окружении Linux с помощью эмуляции, виртуализации разных типов, терминальных систем, инфраструктуры VDI — способов не мало и каждый эффективен в разных случая по разному. Либо, выбрать альтернативное ПО работающее под Linux и перейти на него в рамках проекта. В любом случае, стратегически вопрос решается только переходом, потому что содержать гибридную инфраструктуру дело хлопотное.

Группа миграции отвечает за миграцию данных пользователей. Это очень специфический объем работ — во-первых, данных очень много, нужно уметь их анализировать, во-вторых — любые операции с данными (типа того что их удалить или переместить, или иначе структурировать) — все это требует согласования с пользователем или его руководителем. Нужно делать специальный интерфейс.

Здесь создается еще один серьезный операционный механизм — технологии для миграции пользовательских данных и настроек.

Информационная безопасность очевидный поток работ в любом проекте. Здесь особенность — Вы уже видите, в рамках такого проекта создается свежайшая организационная и техническая документация. Грех не воспользоваться и подготовить ОРД для текущих и перспективных автоматизированных систем в защищенном исполнении. У Вас отличный шанс внедрить единую технологию создания АСЗИ.

Когда все «разработчики» отработали свои куски — создан «стандартный образ», репозитарий типового программного обеспечения, механизмы совместимости, миграции данных и настроек пользователей, остальные механизмы — руководство передается этой группе, которая должна собрать из кусочков целостный операционный механизм миграции. Здесь работают архитекторы всего проекта.

Офис — В этом месте скажем честно — миграция с Microsoft Office на любую другую систему это чудовищная головная боль. Проблема в том, что офис проник везде — нам придется в 40 программах придумать как смигрировать отчетность, в 20 вывод печать, в 10 — интеграционный обмен.

Поэтому, выделяется отдельная группа для сквозных работ по всех функциональным блокам в части офисных систем.

Тестовая лаборатория — зачем она? Мы знаем, что даже у Windows есть проблемы с драйверами, а у нас тут достижения в

этой области скажем скромно — чуть хуже. И на нашей чуть хуже в части драйверов платформе — куча приложений, которые печатают, сканируют, соединяется друг с другом на куче разнородной техники. И вспомним что Группа совместимости запускает все это на Linux весьма нетривиальными способами.

Поэтому отработка испытаний на реальном железе — критический элемент проекта миграции.

Как Вы понимаете, предстоит большой объем работы с людьми. Не скрою, такие проекты проходят скажем так с небольшой перчинкой, даже если все идет по плану. А как Вы хотели? Это глобальное, существенное изменение в огромном куске жизни людей. Среднестатистический работник умственного труда проводит со своим компьютером более половины времени жизни. Гораздо больше, чем с супругами и детьми. И тут мы резко меняем внешний вид и поведение ключевого спутника жизни — представьте себе реакцию. В общем, если повезет грамотно организовать работу с пользователями — поверьте, Вы решите самый главный, и самый ключевой риск проекта миграции.

В итоге — пользователи не должны отторгать систему.

Методика управления проектом обычная — планы, отчеты, жесткий контроль за сроками, работа с бюджетом проекта, с командой и тд и тп. Просто проекты очень сложные.

Апофеоз всего проекта — это результат потока работ «Перевод в эксплуатацию». Смотрите, когда мы все отмигритуем, что у нас еще останется? А останется у нас во первых, тонна накопленных знаний, высоклассный технический инструментарий и команда которая умеет в любых масштабах устанавливать любое ПО (включая перспективное) на любую технику. С невиданным ранее качеством и скоростью.

Эта группа берет проектные стандарты и процессы, и внедряет на постоянную жизнь. Некоторые вещи даже просятся в оргструктуру.

Мы считаем, что это и есть главный результат, не просто отмигрировать, а построить качественно новую ИТ платформу — выйти из проекта с существенным повышением зрелости управления ИТ в целом на много лет вперед.

В заключении хочется отметить — да, проекты миграции дело не простое, но никто и не обещал. А предлагаемая методология комплексно и детально описывает структуру реализации таких проектов.

Литература

[1] *Брукс Ф.*, Мифический человеко-месяц, или как создаются программные системы., http://www.lib.ru/CTOTOR/BRUKS/mithsoftware.txt

[2] Национальный стандарт РФ — ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»

Липатов Виталий Александрович, Никулин Дмитрий Владимирович
Санкт-Петербург, ООО «Этерсофт», ГУАП

Проект: .NET Core, .NET Framework, Nuget, Avalonia, Electron Edge, Visual Studio Code https://www.altlinux.org/.NET_Core

.NET Core 2.0 как универсальная платформа разработки. Миграция с .NET Framework: среда разработки и альтернативные средства создания графического интерфейса

Аннотация

Платформа .NET Core является кроссплатформенным аналогом .NET Framework на уровне библиотек базовых классов (BCL) .NET Framework. Она реализует спецификацию .NET Standard 2.0. В докладе будут рассмотрены средства разработки, варианты сред разработки и способы создания графического интерфейса, ориентированные на создания кроссплатформенных пользовательских приложений.

Платформа .NET Core

По сути `.NET Core` является повторной разработкой `.NET Framework`, но имеющей открытую лицензию (MIT) и работающей на всех платформах (Windows, Linux, MacOS). Обеспечивается совместимость с `.NET Framework` на уровне базовых классов (BCL), согласно стандарту `.NET Standard` 2.0, к которому, по замыслу архитекторов Microsoft, должна стремиться любая существующая реализация `.NET`.

На сегодняшний день .NET Core претендует на то, чтобы быть полноценной платформой промышленного уровня.

В составе SDK поставляется компилятор C# с открытым исходным кодом из состава `.NET Compiler Platform - Roslyn`.

Как и полагается современной платформе, имеется менеджер пакетов (Nuget), и репозиторий пакетов. К сожалению, реализация такова, что вносит зависимость от интернета во время сборки приложения.

Платформа очень быстро развивается (сотни коммитов различных разработчиков каждый день), появляются и исчезают проблемы, меняется схема сборки, учитываются нюансы различных дистрибутивов Linux. К сожалению, во многом платформа выглядит чуждой на Linux-системах.

Ключевым в `.NET Core` является то, что проект обязательно должен быть собран под `.NET Core` (в отличие от Mono, совместимого с `.NET Framework` и запускающего программы без пересборки), а также то, что пока отсутствует поддержка и спецификации для графического интерфейса.

Тем не менее графический интерфейс может быть реализован следующими способами:

- Создание приложения в модели `ASP.NET Core` и отображения клиенту интерфейса через произвольный браузер;
- Использование Electron для отображения интерфейса. Это придаст приложению свойства десктопности (отдельность в панели задач, нахождение в трее при необходимости, взаимодействие с операционной системой);
- Использование Avalonia (свободный межплатформенный GUI для `.NET`);
- Использование проекта electron-edge, выполняющего `.NET` и `Node.js` в одном процессе в Electron. На данный момент проект поддерживает только старую версию `.NET Core` 1.0 и не рассматривается.

Среды разработки и средства разработки GUI

Electron

Electron — это фреймворк для создания нативных приложений с использованием веб-технологий. Electron позволяет использовать браузерный движок в качестве основы для построения нативных при-

ложений с GUI, добавляя к нему набор своих API, недоступных в обычном браузере.

Electron построен на основе `Node.js` и Chromium. Приложение на Electron имеет основной процесс (Main), который является точкой входа в приложение, и по меньшей мере один процесс браузера (Renderer), в котором производится отрисовка GUI.

`Main` — это `Node.js`-приложение, которое управляет окнами (Renderer), а также имеет доступ к API, связанным с системным GUI: строка меню, контекстное меню, работа с треем, диалоги, модальные окна и т. д. В `Main` также доступны встроенные модули `Node.js` (fs, http, net...) и сторонние, устанавливаемые через пакетный менеджер npm.

В Renderer загружается HTML-страница, таблицы стилей и JS-скрипты. От обычного браузера отличается наличием доступа к модулям из `Node.js` и возможность обмена сообщениями с Main при помощи встроенного в Electron механизма IPC.

Примером реального приложения, написанного на Electron, является редактор Visual Studio Code.

Visual Studio Code

VS Code — это редактор для кода с открытым исходным кодом от компании Microsoft, не имеющий ничего общего с небезызвестной IDE Visual Studio.

VS Code позиционируется как редактор, но по сути это нечто среднее между редактором и IDE. Основные возможности:

- подсветка синтаксиса;
- автодополнение с учётом языковых конструкций;
- поддержка отладки;
- интеграция с системами контроля версий — поддержка `Git` встроена.

VS Code не завязан на какую-либо конкретную платформу или язык, а благодаря механизму плагинов можно добавлять поддержку различных языков и дополнительных инструментов. Поддержка JavaScript и TypeScript есть «из коробки». Редактор хорошо интегрируется со всеми популярными языками: C/C++, Java, Python, C#, Ruby, PHP, Go.

Avalonia

Avalonia — свободный кроссплатформенный GUI-фреймворк для .NET, способный работать под .NET Core, который доступен под Linux. Avalonia во многом похож на WPF, использует XAML и подразумевает использование MVVM-архитектуры приложений, но при этом не привязан к Windows-платформе и Microsoft.

Avalonia работает не только на десктопе (Linux, Windows, MacOS), но и на мобильных устройствах (Android, iOS). Фреймворк имеет свой набор GUI-элементов, которые одинаково выглядят на всех системах.

Фреймворк пока что находится в alpha-версии, но он активно развивается и выглядит довольно перспективным решением для создания кроссплатформенных графических приложений. Его важным преимуществом является сходство с WPF, вплоть до использования тех же XAML-файл с описанием интерфейса, что упрощает портирование существующих WPF-приложений.

Антон Бондарев, Александр Калмук, Денис Дерюгин
Санкт-Петербург, Embox

Проект: Embox https://github.com/embox/embox

Embox — Essential toolbox for embedded development

Аннотация

Embox — открытый проект по созданию конфигурируемой операционной системы для встроенных систем. Одной из причин, побудивших студентов и преподавателей Мат-Меха СПбГУ создать собственную операционную систему, было желание использовать возможности Linux на системах с ограниченными ресурсами.

Свободное программное обеспечение (СПО) находит все большее распространение в самых различных сферах. Одним из самых распространенных проектов, можно сказать, символов СПО, безусловно является Linux Kernel (ядро Linux). На основе данного проекта базируются множество экосистем — от виртуализации и облачных технологий до встроенных систем. Последние находят все большее распространение поскольку применяются, например, для построения интернета вещей (Internet of Things (IoT)). Linux успешно применяется и в области embedded systems. Примерами могу служить хорошо

известные проекты OpenEmdedded, OpenWTR, YoctoProject, uClinux и т.д. Но поскольку для вычислительных систем из этой области характерны ресурсные ограничения (resource restrictions), то Linux не в полной мере удовлетворяет требованиям предъявляемым к аппаратуре для подобных систем. Например, наличие небольшого количества памяти, отсутствие MMU, желание исполнять исходный код напрямую из встроенной flash-памяти микроконтроллера, все это делает применение Linux в подобных устройствах сильно затруднительным.

Частично аппаратные ограничения пытался решить uClinux, основной идеей проекта было дать возможность запускать Linux приложения на платформах без аппаратной поддержки MMU (без трансляции адресов). Изменения частично внесены в основное ядро, позволив использовать режим NOMMU в Linux. Но данный проект не позволял решить проблему запуска ПО на платформе с несколькими сотнями килобайт или парой мегабайт памяти. На решение подобных задач направлены другие проекты с открытым исходным кодом: eCos, FreeRTOS, Contiki и другие. Данные проекты обычно принято называть realtime operating system (RTOS), вероятно потому, что они подразумевают куда больший контроль над всем ПО в системе, хотя это и не совсем корректно.

Подобные проекты RTOS хоть и решают задачу размещения ПО в ограниченном объеме памяти и более легкого доступа к аппаратуре, но зачастую порождают проблему совместимости ПО. Дело в том, что API в этих системах не стандартизовано, и код прикладных программ приходится адаптировать, а зачастую и разрабатывать с нуля, тем самым ограничивая применение других проектов с открытым кодом.

Подобную проблему частично пытался решить eCos, но поскольку это была не основная идея проекта, полностью решить проблему не удалось. И на текущий момент есть два проекта, нацеленных на создание «мини-Linux», иными словами, некой платформы позволяющей достаточно просто использовать богатство прикладных программ из мира Linux с posix интерфейсом, но при этом выполняться на платформах с ограниченными ресурсами. Это проекты NuttX и Embox.

В проекте Embox достаточно эффективно удалось решить проблему построения «мини-Linux». С одной стороны система сборки позволяет очень гибко настраивать все параметры системы, включая параметры ядра (например, какой планировщик использовать вытесняющий или нет), с другой стороны достаточно развитая собственная стандартная библиотека позволяет и схожая с Linux модель драйве-

ров, позволяют фактически без изменений использовать сторонние приложения с открытым кодом, а также разрабатывать свои, переносимые приложения и библиотеки. Для упрощения использования сторонних проектов, был разработан набор скриптов встроенных в общую систему сборки, способный скачивать исходный код открытого проекта, накладывать необходимые патчи, сконфигурировать и собрать его стандартным предусмотренным в стороннем проекте методом. При этом результат сборки проекта включается в состав конечного образа Embox и может быть вызван в том числе из командной строки.

Изначально Embox развивался как студенческий проект на Мат-Мехе СПбГУ. Он применялся для практического ознакомления с операционными системами, системным программированием, сетевым программированием и встроенными системами. В дальнейшем проект получил развитие как открытый проект, и в нем стали принимать участие и другие разработчики. Кроме того, на базе открытого проекта ведутся и коммерческие разработки.

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

Свободная система управления бизнес-процессами и административными регламентами RunaWFE. Новые возможности последних версий

Аннотация

В докладе рассматриваются возможности последних версий свободной системы RunaWFE. Показана работа элементов генерации и обработки событий, объяснен «непрерывающий» режим работы обработки событий и таймеров, представлены дополнительные настройки фильтров списков задач и экземпляров бизнес-процессов, а также другие появившиеся возможности.

Дополнены настройки фильтров экземпляров процессов

В фильтр списка экземпляров была добавлена возможность фильтрации по следующим параметрам

- Исполнитель задания (с опцией — «включая/не включая вхождение в группы»)
- Название роли
- Имя задания
- Длительность задачи
- Текущая длительность задачи
- Время получения задания
- Время взятия задания на исполнение
- Срок задачи

Фильтрация работает следующим образом:

Рассматривается список незавершенных экземпляров бизнес-процессов, в них рассматриваются все узлы-действия, в которых в данный момент находятся точки управления.

Далее из списка отбираются экземпляры, какая-либо точка управления которых находится в узле-действии, для которого выполняется:

- Если элемент «исполнитель задания» отмечен галочкой, то — исполнителем является указанный в фильтре пользователь (или группа, в случае соответствующей опции).
- Если элемент «название роли» отмечен галочкой, то название роли совпадает с указанным в фильтре
- Если элемент «имя задания» отмечен галочкой, то название задания совпадает с указанным в фильтре
- Если элемент «длительность задачи» отмечен галочкой, то длительность задачи попадает в границы, указанные в фильтре
- Если элемент «текущая длительность задачи» отмечен галочкой, то текущая длительность задачи попадает в границы, указанные в фильтре
- Если элемент «время получения задания» отмечен галочкой, то время получения задания попадает в границы, указанные в фильтре
- Если элемент «время взятия задания на исполнение» отмечен галочкой, то время взятия задания на исполнение попадает в границы, указанные в фильтре
- Если элемент «срок задачи» отмечен галочкой, то срок задачи попадает в границы, указанные в фильтре

Экземпляры, для которых выполняется условие, отображаются в таблице. Также по данным элементам возможна сортировка и группировка экземпляров

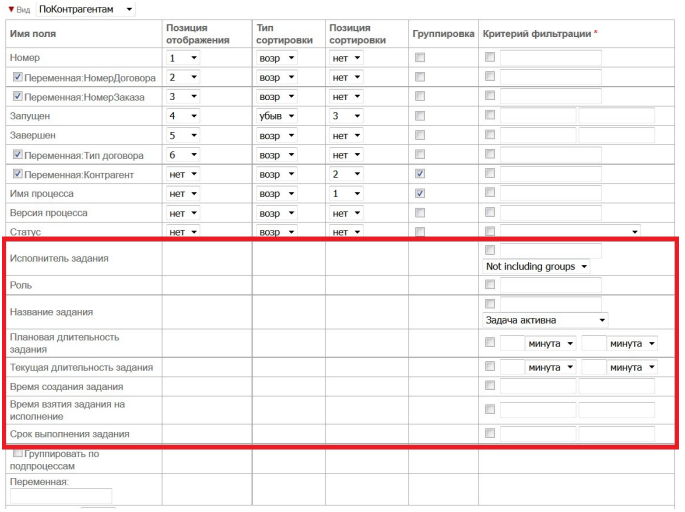


Рис. 1: Новые поля в фильтре списка экземпляров бизнес-процессов

Для отфильтрованного списка экземпляров добавлена возможность экспорта в формат MS Excel

Дополнены настройки фильтров задач

В набор полей настройки списка задач были добавлены поля:

- Создана (Время создания задания)
- Получена (Время назначения задания на группу или взятия на исполнение)
- Длительность (Текущая длительность задания)

По полю «Владелец» и «Создана» была добавлена возможность фильтрации, сортировки и группировки.

Также в список заданий была добавлена возможность экспорта в формат MS Excel

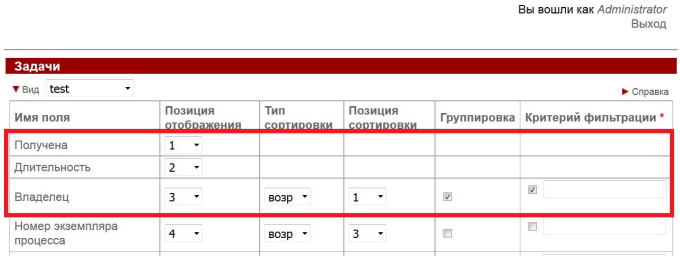


Рис. 2: Новые поля в фильтре списка задач

Добавлены элементы генерации и обработки событий

В палитре среды разработки элементы «отправить сообщение» и «получить сообщение» были заменены на более общие — «Генерация события» и «Обработка события». У этих элементов есть свойство, в котором можно выбирать тип события. Возможны следующие типы:

- Сообщение
- Сигнал
- Ошибка
- Отмена

Для элемента «Обработка события» реализована возможность присоединить его к узлу-действию в правом-нижнем углу (По аналогии с таймером). К такому элементу можно присоединить исходящий переход, по которому точка управления будет переходить из узла-действия в случае возникновения присоединенного к элементу события.

Иконка для сигнала — треугольник в кружочке. Иконка для ошибки — буква N в кружочке, для отмены — крестик в кружочке.

Реализован «непрерывающий» режим работы обработки событий и таймеров

У присоединенной к узлу-действию обработки события или таймера добавлено свойство «Прерывающий». По умолчанию остается

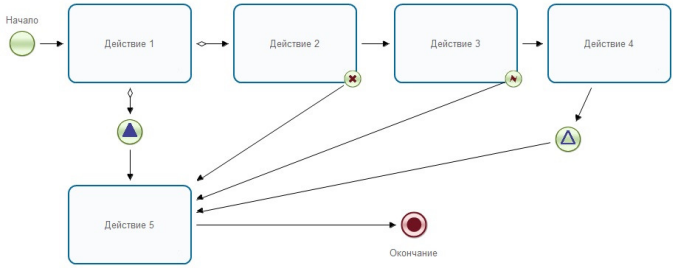


Рис. 3: Пример использования новых элементов

поведение, реализованное в предыдущих версиях («Прерывающий» = «Да»).

Если свойство — «Нет», то при срабатывании обработки события (Сигнала, Сообщения, Ошибки, Отмены) или таймера на присоединенном переходе (если он есть) генерируется новая точка управления, а старая точка управления остается в узле-действии. В случае значения «Да» — старая точка управления удаляется.

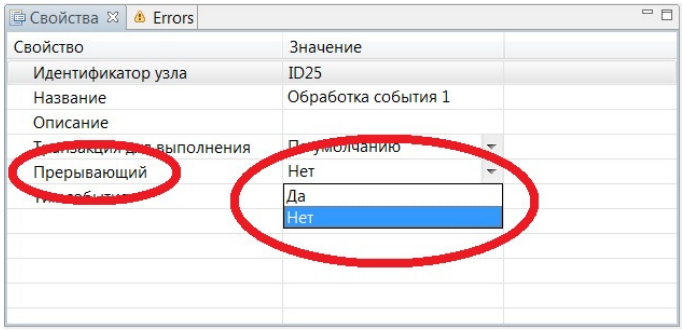


Рис. 4: Настройка параметра «Прерывающий» в свойствах Обработки события

Другие изменения

- Реализован просмотр задач других пользователей
- Добавлены режимы завершения потока в композициях
- Расширена возможность использования переменных в режиме чтения из другого экземпляра бизнес-процесса
- Добавлена возможность управления границами транзакций при выполнении экземпляров процессов
- Добавлена поддержка переменных типа Bigdecimal

Ссылки

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

Сомова Ирина Игоревна
Уфа, ООО «РН-«УфаНИПИнефть»
http://www.runawfe.org/

Развитие свободной системы RunaWFE в процессе внедрения и эксплуатации в ООО «РН-«УфаНИПИнефть»

Аннотация

В процессе внедрения в промышленную эксплуатацию свободной системы с открытым кодом RunaWFE в проектном институте численностью более 750 человек потребовалось адаптировать систему и разработать дополнительную функциональность. В докладе рассказывается, о процессе выполнения работ и достигнутых результатах.

Задачи внедрения системы RunaWFE в ООО «РН-«УфаНИПИнефть»

Одной из задач, выполняемой сотрудниками проектного института, является обработка данных геологических исследований скважинам, которые в электронном виде в большом объёме поступают от подрядчиков. Для приема и загрузки получаемой информации требуются большие трудозатраты.

Задачи проекта:

- повысить производительность труда сотрудников, выполняющих работу по приёму данных геологических исследований скважин;
- провести анализ, реинжиниринг и оптимизацию процессов по приему данных геологических исследований скважин;
- внедрить механизм контроля передачи заданий на загрузку данных по скважинам в корпоративные базы данных, и отслеживания соблюдения регламентных сроков по загрузке данных.

Для решения поставленных задач было принято решение о внедрении на предприятии системы управления бизнес-процессами и административными регламентами.

Был проведен анализ представленных на рынке BPMS систем. При выборе системы внимание уделялось следующим критериям:

- стоимость;
- наличие примеров стабильной работы системы на предприятиях численностью более 500 человек;
- скорость разработки и внедрения процессов;
- русскоязычная поддержка;
- простой доступный для понимания интерфейс;
- возможность интеграции с MS Active Directory.

В результате проведенного сравнения систем выбор был остановлен на российском проекте RUNA WFE, который разрабатывается и поддерживается ООО «Процессные технологии».

Ход и результаты внедрения

Внедрение информационной системы RUNA WFE было начато на предприятии в 2015 г. Во втором полугодии 2016 г. система официально введена в промышленную эксплуатацию.

В процессе внедрения возникли следующие сложности:

- выявлены особенности настройки аутентификации Керберос, вызванные политиками безопасности;
- обнаружена недостаточность информации о пользователях, импортируемой RunaWFE из AD (LDAP).

Все проблемы, возникшие в процессе внедрения, были решены разработчиками в оперативном порядке путем доработки системы и расширения функциональных возможностей.

Адаптация и разработка новой функциональности RunaWFE в процессе внедрения

В процессе внедрения информационной системы RunaWFE возникла необходимость в адаптации и разработке новой функциональности системы:

- были расширены возможности по получению информации ИС RunaWFE из AD (LDAP);
- доработан механизм инициализации ролей;
- добавлена возможность полностью скрывать на схеме бизнес-процесса некоторые технические элементы, не информативные для бизнеса;
- доработан механизм замещения исполнителей заданий;
- доработан механизм работы с профилями (настройками фильтров задач и экземпляров бизнес-процессов);
- реализована возможность ежедневных e-mail напоминаний о просроченных заданиях.
- реализован бот распределения полномочий с использованием значения переменных бизнес-процесса.

Также были реализованы другие изменения.

На данный момент на предприятии получены следующие результаты от внедрения ИС RunaWFE:

- автоматизировано 8 процессов геологической службы и отдела ППиСП;
- все работники предприятия имеют доступ к системе;
- за 2 года запущено более 5 000 экземпляров процессов.

Решены задачи:

- повышение производительности труда работников за счет исключения переписки по почте и уточнения заданий;
- появление инструмента для электронного согласования внутренних документов, имеющих нестандартные маршруты согласования;

- появление инструмента контроля соблюдения регламентных сроков выполнения заданий.

Ссылки

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

Александр Дубицкий, Дмитрий Костюк, Анастасия Маркина, Станислав Фомин
Брест, Брестский государственный технический университет

Проект: UXDump https://bitbucket.org/AsyaAliset/uxdump

Применение айтрекеров для юзабилити-исследований ПО в GNU/Linux

Аннотация

Представлен анализ технологии окулографии с учетом физиологических и психологических особенностей человека, а также возможные пути использования современных окулографических устройств (айтрекеров) для отслеживания направления взгляда пользователей Linux-систем. Обсуждаются особенности и степень работоспособности программных и аппаратных реализаций. Приводится схема эксперимента, использованная авторами для исследований взаимодействия пользователя с графическими приложениями в GNU/Linux. Выведены ограничения использования, особенности анализа результатов на основе интерпретации визуальной информации.

Особенности окулографических исследований

Окулография или айтрекинг — это анализ движения взгляда пользователя, зон визуальной фокализации, на которых концентрируется взгляд, а в более узком смысле — определение точки пересечения оптической оси глазного яблока и наблюдаемого на экране объекта. В отличие от других методик UI-тестирования, в этом случае испытуемые не комментируют свои действия. Поскольку окулографические исследования позволяют избежать искажений, вызванных вербализацией, они достаточно актуальны при исследовании эргономики графических приложений.

В устройстве современных айтрекеров обычно используются цифровые камеры, которые снимают с высокой частотой кадров глаза

пользователя и регистрируют их движения. Айтрекер монтируется на голову испытуемого, либо располагается в поле зрения стационарно и автоматически отслеживает движения головы.

Можно выделить следующие направления применения айтрекера в задачах оценки эффективности человеко-машинного взаимодействия:

- определение причин UX-проблем, связанных с заметностью элементов, точками фокуса внимания, ментальной нагрузкой и отвлечением внимания;
- определение особенностей поведения пользователей: стратегий визуального поиска, паттернов чтения и сканирования;
- сравнение нескольких дизайнерских решений (в сочетании с другими видами тестирования: анкетированием, биометрической оценкой и др.).

Возможности применения в GNU/Linux

Разработчики, желающие выполнить тестирование ПО для GNU/-Linux с помощью айтрекера, имеют несколько вариантов действий. Наиболее бюджетный вариант — использование программного решения. Несколько свободных проектов (различной степени завершённости и кросс-платформенности) реализуют распознавание лица и глаз пользователя с помощью веб-камеры: eviacam, TrackEye, WebGazer.js. Хотя их можно использовать для отслеживания направления взгляда [1], на результат существенно влияют такие ограничения веб-камеры, как частота кадров, эффективная точность позиционирования взгляда (вдвое меньше разрешения камеры), ограничения в распознавании поворота головы. На противоположном конце шкалы находятся профессиональные айтрекеры, рассчитанные на исследовательские лаборатории с большим бюджетом. Некоторые из них являются Linux-совместимыми (предоставляют драйвера либо позволяют получать данные по сети). Однако существует компромиссное решение — использование их аналогов, ориентированных на сферу развлечений, которые позволяют добиться точности, достаточной для решения ряда исследовательских задач [2, 3]. Айтрекеры этой последней категории, производимые компанией Tobii, использовали совместно с GNU/Linux и авторы настоящей работы.

Айтрекеры Tobii делятся на дорогостоящую профессиональную серию Tobii Pro и устройства потребительского сегмента, для которых производитель изначально разработал два вида SDK: низкоуровневый кроссплатформенный Gaze SDK и более простой в применении EyeX SDK. Авторами опробованы два айтрекера, Tobii REX (совместимый с Gaze) и Tobii EyeX. Оба айтрекера — компактные устройства, размещаемые непосредственно под дисплеем ноутбука либо ПК и подключаемые к интерфейсу USB 3.

К сожалению, Gaze SDK объявлен устаревшим, и компания Tobii энергично занимается избавлением пользователей от возможности какого-либо доступа к его кросс-платформенным вариантам в результате таинственного разочарования в собственном ПО. Взамен Gaze SDK Tobii предлагает новый вариант, Core SDK 2017 г. выпуска, в котором заявлена совместимость с GNU/Linux и MacOS. Однако на текущий момент распространяются только его Windows-версии.

В результате, на текущий момент единственной работоспособной схемой отслеживания взгляда Linux-пользователей является следующая. В состав рабочего места входят два компьютера: ПК1, работающий под управлением MS Windows нужной версии, и ПК2 с Linux и ПО для тестирования. Айтрекер закрепляется на корпусе ПК2 так, чтобы распознать лицо оператора, но электрическое подключение айтрекера выполняется к разъему USB3 ПК1. На ПК1 выполняется калибровка и приём данных, а пользователь находится перед ПК2 и выполняет тестовые задания. Возможны две разновидности данной схемы:

- применяются два ноутбука — например, поставленные «встык», как показано на рис. 1 слева. Схема предусматривает два посадочных места: экспериментатора, и подопытного, выполняющего тестовые задания.

- ПК1 подключен к сети в режиме сервера, и может не иметь дисплея (headless mode). Изображение с ПК1 через клиент удалённого доступа поступает в отдельное графическое окно, отображаемое на дисплее ПК2.

Заметим, что очевидное решение с помещением MS Windows в виртуальную машину является проблематичным из-за требований Tobii к версии и пропускной способности шины USB.

Рис. 1: Схема подключения айтрекера

Специфика интерпретации результатов

Результатом работы айтрекера является массив координат, соответствующих положению взгляда в различные моменты времени. При его интерпретации необходимо учитывать разницу между периферическим и центральным зрением. Центральное зрение предоставляет информацию о мелких деталях изображения, но охватывает достаточно малую часть поля зрения. В результате информация поступает с помощью зрительной выборки, а периферическое зрение указывает центральному на следующую точку фокусировки взгляда [4]. Фактически, взгляд перемещается быстрыми скачками (саккадами) с короткими паузами между ними (фиксациями). Так, при работе с текстом за одну саккаду считывается 7–9 букв (рис. 2), а продолжительность фиксации составляет около 250 миллисекунд, но восприятие распространяется на вдвое большее число букв из-за участия периферического зрения [5].

При исследовании движений глаз анализируется только центральное зрение, а периферическое не учитывается. Поэтому при интерпретации результатов важно избегать «ошибки выжившего»: например, если взгляд часто фиксируется на каких-то элементах управления, это может быть свидетельством не их востребованности и удачного расположения, а временного сбоя когнитивных процессов, вызванного связанной с ними информационной перегрузкой.

В общем случае при проведении UX-исследований с использованием айтрекера учитывают такие параметры:

- число фиксаций;
- длительность каждой фиксации;

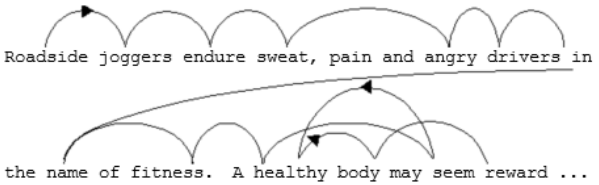


Рис. 2: Саккады при работе с текстом

- время до первой фиксации;
- длительность первой фиксации;
- общее число фиксаций;
- сколько было фиксаций до посещения зоны интереса;
- общее время фиксаций.

Наглядная визуализация результатов отслеживания взгляда предусматривает два варианта:

- построение тепловой карты, подсвечивающей части рабочего окна программы в зависимости от интенсивности фиксации взгляда (разновидность — «туманная карта», скрывающая части изображения, не привлекавшие внимание пользователя);
- построение карты перемещения взгляда во времени.

В обоих случаях это предполагает наложение соответствующей карты на изображение, с которым работал пользователь, либо на результат видеопротоколирования. При обработке в GNU/Linux для построения тепловых карт по координатам удобно использовать математические пакеты общего назначения (например octave), а в случае карты перемещения взгляда — ПО отрисовки графов (graphviz).

Литература

[1] *Semmelmann K., Weigelt S.* Online webcam-based eye tracking in cognitive science: A first look // Behav. Res. 2017. http://rdcu.be/tjzF.

[2] *Titz J., Scholz A., Sedlmeier P.* Comparing eye trackers by correlating their eye-metric data // Behav. Res. 2017. https://www.ncbi.nlm.nih.gov/pubmed/28879442.

[3] *Gibaldi A. et al.* Evaluation of the Tobii EyeX Eye tracking controller and Matlab toolkit for research // Behav. Res. V. 49, 2017. P.923–946.

[4] *Ludwig C.J.H., Davies J.R., Eckstein M.P.* Foveal Analysis and Peripheral Selection during Active Visual Sampling. // Proc. of the National Academy of Sciences of the USA. V. 111(2), 2014. P. E291–E299.

[5] *Goodman K.S.* On Reading. / Portsmouth, NH: Heinemann, 1996. 152 p.

Шамонин Вадим Павлович, Костюк Дмитрий Александрович
Брест, Брестский государственный технический университет, Брестский государственный медицинский колледж

Проект: emgdump https://github.com/shadimX/emgdump

Электромиографическое распознавание биопотенциалов человека

Аннотация

В докладе представлен открытый аппаратно-программный проект для распознавания электромиографических сигналов человека. В основе аппаратной части задействована платформа Arduino, которая принимает данные от накожных датчиков и передаёт в принимающее программное обеспечение для обнаружения и классификации мышечной активности. Рассмотрены результаты апробации системы при распознавании движений пальцев рук. Обсуждаются особенности применения разработки для управления исполнительными устройствами, а также в качестве источника информации для диагностического оборудования.

Метод поверхностной (неинвазивной) электромиографии заключается в исследовании биоэлектрической активности мышц, регистрируемой с помощью помещаемых на кожу поверхностных электродов. ЭМГ-сигнал, снимаемый с таких электродов, содержит осцилляции различной амплитуды, частоты и периодичности (от 100-150 мкВ до — 100-3000 мкВ в зависимости от степени сокращения мышцы, состояния и возраста подопытного и др. факторов).

Электромиографические исследования мышечной активности человека проводились с начала XX века; однако применение компьютерного анализа электромиограмм, их автоматической классификации и распознавания в сочетании с относительной простотой регистрации делает данные методы актуальными за пределами чисто медицинской сферы применения. Выделение в ЭМГ-сигнале паттернов, характерных для напряжения конкретных мышц, может использоваться для создания средств управления виртуальной реальностью, мониторинга физической активности человека, при создании различных вспомогательных устройств и др.

Несмотря на широкие возможности ЭМГ, существующие в настоящее время коммерческие устройства рассчитаны на узкое применение, а их производители не заинтересованы в свободном распространении и более универсальном применении своих наработок. Это послужило причиной создания данного аппаратно-программного проекта, доступного, благодаря принципам открытого аппаратного и программного обеспечения, для развития заинтересованными разработчиками и различных альтернативных применений. Наработки проекта доступны по адресу https://github.com/shadimX/emgdump.

Основой для разработки устройства послужила платформа Arduino. При помощи неё данные обрабатываются при подключении к ПК. Для более точного выявления биопотенциалов используется специализированный OpenHardware-модуль цифрового усиления сигналов muscle sencor v3, позволяющий фиксировать электрическую активность мышц при помощи стандартных накожных Ag-электродов (рис. 1), широко используемых в медицинских целях при записи ЭКГ.

Разработка программной подсистемы, отвечающей за распознавание сигналов, была проведена на задаче применения движений пальцев рук в качестве источника управляющих сигналов. Применение ЭМГ в данном случае делает возможным как создать более удобный аналог перчаток виртуальной реальности (датчики крепятся на предплечье, оставляя кисть свободной), а также перспективно в области реабилитации пациентов после инсультов и нейротравм.

Практическим методом была установлена взаимосвязь амплитуды получаемого сигнала на конкретном участке мышцы от степени сгибания пальцев руки. Были опробованы две методики регистрации сигналов. В рамках 1-й методики положения №4 и №5, также как и №6 и №7 регистрировались одним электродом (рис. 1). Сигналы записывались относительно канала Reference, положение которого выби-

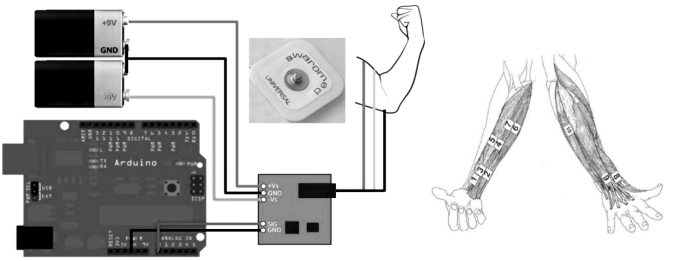


Рис. 1: Схема подключения усилителя биопотенциалов, вид используемых накожных электродов Ag/AgCl и точки их крепления

ралось на участке выше локтя, на котором отсутствуют сокращения мышц при движении пальцами. В рамках второй методики те же положения регистрировались отдельными электродами, при положении канала Reference таком же, как в 1-й методике, а кроме того, снимались 5 дифференциальных каналов между положениями 1 и 2; 3 и 4; 5 и 6; 7 и 8; 9 и 10. В результате методика позволила улучшить отношение сингал/шум, что обеспечило существенно лучшее качество распознавания движений. Пример распознавания сигналов приведен на рис. 2.

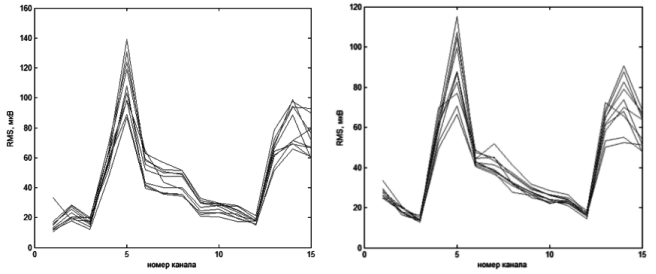


Рис. 2: Распознавание движений мизинца (слева) и указательного пальца (справа)

Отметим, что использование некоторых каналов не улучшает работу классификатора. Для 2-й методики достаточно использовать 5 дифференциальных и 2 отдельных канала. В результате проверки выбранных методик измерения были получены 65% и 95% вероятности правильного определения сгибания пальца 1-й и 2-й методик соответственно. Время работы наиболее удовлетворительного алгоритма классификации составляет 0,5 мс, что позволяет применять его в режиме реального времени.

В данный момент на основе `emgdump` ведётся разработка устройства контроля осанки. После установки электродов в районе мышц, отвечающих за поддержание осанки, фиксируются значения ЭМГ-сигналов в крайних (согнутом и выпрямленном) положениях. Далее задаётся интервал значений, при котором осанка является удовлетворительной. Таким образом, при расслаблении мышц значение электромиограммы выходит за пределы допустимых, после чего срабатывает звуковой сигнал или вибромотор.

Рис. 3. Визуализация мышечной активности и расположение электродов на группах мышц, формирующие осанку

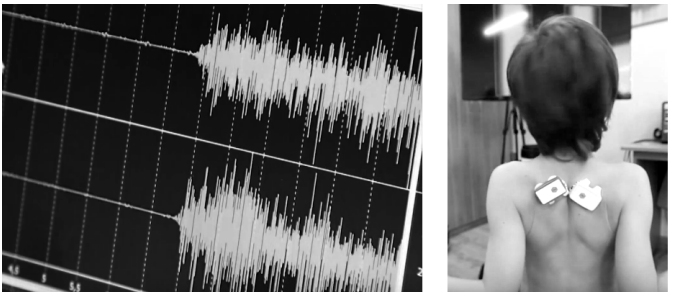


Рис. 3: Визуализация мышечной активности и расположение электродов на группах мышц, формирующие осанку.

Константин Пархоц
Брест, Брестский государственный технический университет

Проект: trackball-learn https://github.com/parchoc/trackball-learn

Использование библиотеки scikit-learn в задаче классификации движений трекбола

Аннотация

Описывается применение свободной библиотеки машинного обучения scikit-learn для решения задачи распознания нестандартных движений трекбола. Описаны особенности библиотеки, её возможности, приведено сравнение с другими свободными библиотеками машинного обучения. Для анализа особенностей перемещений трекбола использован метод опорных векторов. Рассмотрено создание классификатора, а также результаты его обучения для программной реализации в трекболе дополнительной функции прокрутки вращением шара в горизонтальной плоскости. Представлен экспериментальный драйвер, позволяющий генерировать данным способом события прокрутки при использовании любого механического либо оптического трекбола.

К машинному обучению относят подходы, где используется не прямое решение задачи, а обучение программы на задачах-аналогах — на реальных задачах в ходе эксплуатации или учебных (для предварительной «дрессировки» системы на правильные ответы). Свободных проектов реализации алгоритмов машинного обучения достаточно много, они отличаются по производительности, языку, универсальности или ориентации на конкретную сферу применения [1]. Scikit-learn [2] — одна из таких библиотек. Она реализована на языке Python, лицензируется под упрощенной BSD и включает средства классификации, регрессионного и кластерного анализа. Особенности, на которые делают упор разработчики — простота использования, высокая производительность, подробная документация и понятный API. Производительность scikit-learn обеспечивается компилированием кода, рядом высокоэффективных библиотек на Python (NumPy), а также реализация подпрограмм линейной алгебры на C/C++. Особенностью API является активное использование интерфейсов для доступа к объектам, что упрощает код и позволяет легко заменять фрагменты кода и используемые алгоритмы [3].

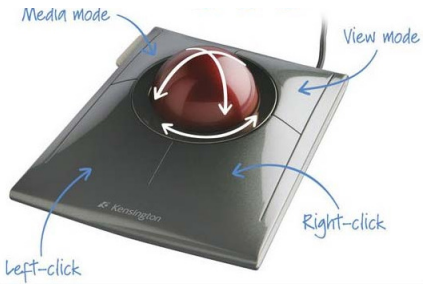


Рис. 1: Kensington Slimblade Trackball

Реальная прикладная задача распознавания, в которой библиотека scikit-learn использовалась авторами — распознавание вращательных движений шара трекбола. Трекбол является достаточно распространенным альтернативной устройством перемещения курсора. Конструктивно он представляет собой опрокинутую механическую мышь (пользователь вращает шар вместо того, чтобы перемещать корпус устройства). Из-за неподвижного корпуса трекбол требует меньше места, а неподвижное запястье уменьшает нагрузку при длительном использовании, и это предупреждает/ослабляет туннельный синдром.

Существенной проблемой в конструкции многих трекболов является интеграция колеса прокрутки, для которого найти удобное место расположения труднее, чем в случае мыши. Варианты расположения бывают очень разные (вплоть до использования большого подвижного обруча вокруг шара), и в отличие от мыши, ни один из них не оказался достаточно удобным, чтобы стать стандартом. Также прокрутку можно реализовать программно, вращением шара с зажатой клавишей (например, опциями `EmulateWheel` и QEmulateWheelButton в `xorg.conf`), однако это увеличивает физическую нагрузку пользователя. Самое минималистичное решение применено в модели Kensington Slimblade Trackball, где дополнительный оптический датчик отслеживает вращение шара в горизонтальной плоскости (рис. 1); вращение по часовой стрелке означает прокрутку в прямом направлении, а против — в обратном.

Во всех остальных трекболах горизонтальное вращение предсказуемо вызывает незначительные «хаотические» перемещения курсора.

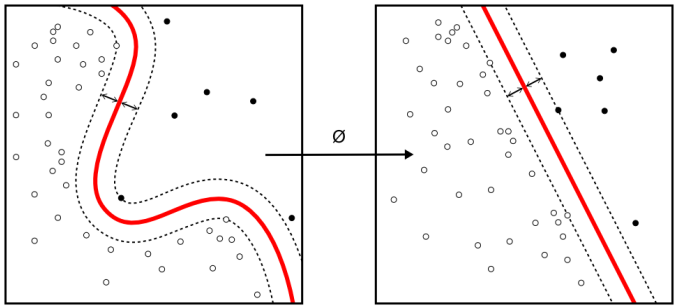


Рис. 2: Нелинейный классификатор на основе SVM

Однако распознавание такого вращения в потоке векторов перемещения позволило бы добавить интуитивную прокрутку во множество более дешевых устройств.

При исследовании возможности такого распознавания средствами scikit-learn использовался метод опорных векторов (support vector machines или SVM). Каждый объект данных (упорядоченный набор из n чисел) представляется в SVM как вектор (точка) в n-мерном пространстве. Каждая из этих точек принадлежит одному из двух классов, и множество точек делится такой гиперплоскостью размерности $n - 1$, что обеспечивает максимальный зазор между классами (для более надежной классификации). Для построения нелинейного классификатора скалярное произведение при построении гиперплоскости заменяется на нелинейную функцию (многочлен, функцию Гаусса, радиальную базисную функцию, сигмоидальную функцию на основе гиперболического тангенса и др.) — т. н. «Kernel trick». Формально эту функцию считают скалярном в пространстве с еще большей размерностью (рис. 2).

На странице https://github.com/parchoc/trackball-learn находится программный модуль, разработанный для экспериментов с распознаванием движений трекбола. Вращение шара в scikit-learn распознает класс SVC. Классификатор обучается с помощью функции fit, с указанием массива данных и целевых значений для них. Опционально обучены классификатор можно сохранить для дальнейшего использования с помощью функции joblib.dump. Далее новые

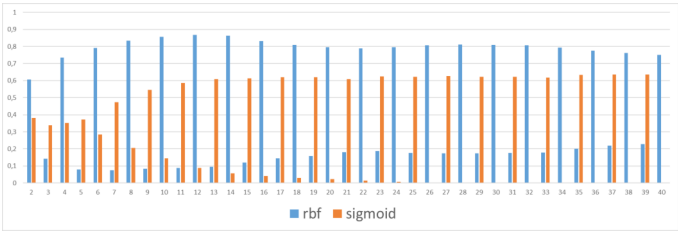


Рис. 3: Точность распознания всех типов движения при разном размере скользящего окна

значения распознаются функцией predict с массивом данных в качестве аргумента (результат — значение вероятности принадлежности к заданному классу).

В качестве функции активации лучше всего проявили себя сигмовидная функция, которая показала наибольшую точность распознавания, и радиальная базисная функция (RBF). Исследование работы классификатора при различных размерах скользящего окна от 2 до 40 показало, что RBF более эффективно при размере окна 12, а сигмоидная функция — при значении 37. Для RBF результаты оказались заметно лучше при четном размере окна, тогда как для сигмоидной функции — при нечётном.

На основе данных результатов была создана пробная программная реализация, генерирующая события прокрутки на основе распознавания типа движения в реальном времени. Программа написана на Python с графическим интерфейсов на основе PyQt. На текущий момент данный модуль работает под Windows. Он использует классификатор с RBF в качестве ядра. Координаты курсора снимаются через равные промежутки времени средствами фреймворка Qt, а затем вычисленная скорость перемещения курсора передаётся в классификатор. В зависимости от отклика классификатора генерируется событие прокрутки (обращением к системной библиотеке user32), либо перехваченное перемещение курсора возвращается в систему для дальнейшей обработки.

Литература

[1] *Yegulalp S.* 11 open source tools to make the most of machine learning // InfoWorld, Dec 4, 2014. http://www.infoworld.com/article/2853707/robotics/11-open-source-tools-machi

[2] Scikit-learn: machine learning in Python. http://scikit-learn.org/stable/

[3] *Pedregosa F. et al.* Scikit-learn: machine learning in Python // Journal of Machine Learning Research, No. 12, 2011. — P. 2825–2830.

Денис Силаков
Москва, Virtuozzo

Проект: ReadyKernel, kpatch https://readykernel.com/, https://src.openvz.org/projects/UP/repos/kpatch

ReadyKernel — инструментарий и сервис обновления ядра без перезагрузки на основе kpatch

Аннотация

Обновление ядра в Linux сопряжено с перезагрузкой системы, а следовательно — с временной недоступностью ее сервисов. Инструментарий kexec снижает время перехода на новое ядро, однако и при его использовании время недоступности системы может достигать десятков секунд. Для ряда пользователей даже десять секунд — достаточно заметное время простоя, которого хочется избежать. При этом ядро обновлять все-таки необходимо, но часто не для получения каких-то новых возможностей, а для исправления ошибок и уязвимостей. Подобные исправления обычно носят локальный характер и могут быть применены к работающему ядру «на лету» с помощью соответствующих инструментов. Пионером в этой области был ksplice, ныне принадлежащий Oracle, однако в последние годы развиваются полностью открытые альтернативы — kpatch, kGraft и livepatch. Данный доклад посвящен основанному на kpatch инструментарию ReadyKernel и одноименному сервису, используемым в Virtuozzo для установки критических обновлений клиентам без перезагрузки их систем.

Технология

Изменение ядра на лету включает в себя подмену тел функций и (в некоторых реализациях) глобальных структур данных. Новая реализация загружается в память параллельно старой, а все обращения к старому коду перенаправляются на новый — например, вставкой безусловного перехода в начало старой функции (KernelCare), либо с использованием механизма трассировки Ftrace. Последний позволяет перехватывать вызовы заданных функций и выполнять в этот момент ряд действий — например, изменять значения регистров с целью передать управление в другую область памяти.

Реализации

Закрытые: Ksplice, KernelCare. Открытые: Kpatch (RHEL), kGraft (SUSE), Livepatch (объединение идей Kpatch и kGraft, добавляется в апстрим начиная с 4.0, используется в Ubuntu).

ReadyKernel — сервис, основанный на kPatch с небольшими изменениями. Штатное средство доставки критических обновлений в ядрах Virtuozzo 7. Опробован также на CentOS, RHEL и Ubuntu. В перспективе рассматривается переход на Livepatch.

Модели целостности

Как выбрать момент для подмены, чтобы не нарушить целостность работающих процессов? Есть несколько моделей целостности:

- *«Никакая»* (Livepatch в ядре 4.0) — не делаем никаких проверок. Быстро, работает для одиночных функций.
- «Clean cut» (Kpatch) — получаем набор активных процессов (в kpatch — через stop_machine), изучаем их стеки вызовов и проверяем — нет ли там целевых функций, которые надо патчить. Недостатком является вызов stop_machine, но эта модель позволяет пропатчить несколько взаимосвязанных функций.
- *«Per-task»* (kGraft, Livepatch в ядре 4.12) — реализация принципа RCU (Read-Copy-Update). Для каждого потока поддерживается флаг, индицирующий — старую или новую реализацию функции он должен использовать. Переключение на новую происходит в момент выхода в userspace. Позволяет учитывать рекурсивные вызовы, но не наборы функций.

- «Hybrid» (Livepatch в будущем) — объединение предыдущих моделей, использующая проверку стека, но для каждого процесса в отдельности.

Подготовка патчей

Патчи оформляются как модули ядра. Подготовка патчей автоматизирована, однако результат требует обязательной проверки человеком.

Использование на практике

Для любого механизма обновления ядра на лету самое главное — своевременная подготовка патчей. Ввиду трудоемкости подготовки, в настоящее время патчи предоставляются только в рамках коммерческих сервисов, появление или исчезновение которых определяется наличием спроса. В частности, сервисы обновления ядра «на лету», основанные на открытых инструментах, предоставляются в рамках подписок Virtuozzo, SUSE и Ubuntu. RedHat предоставляет kPatch в статусе Technical Preview (при этом в RHEL 7.4 вошел kpatch 0.4.0, включивший, помимо прочего, правки от Virtuozzo).

ReadyKernel для Virtuozzo мы считаем коммерчески оправданным — в течение года с момента выпуска Virtuozzo 7 он позволяет нам избегать незапланированных обновлений ядра. SUSE и Ubuntu также начали продвижение своих сервисов относительно недавно, и на данный момент считают их перспективными. В конце концов, востребованность проприетарных Ksplice и KernelCare позволяет надеяться, что в целом подобные сервисы будут пользоваться спросом достаточным для того, чтобы коммерческие компании вкладывались в развитие открытой альтернативы в апстриме.

Медведев Денис
Москва, Базальт СПО

Системы принятия решений: пара полезных свободных инструментов

Аннотация

Предлагается обратить внимание на два свободных пакета для использования в целях обработки знаний и принятия решений — python-module-pyke и spamassasin

Системы принятия решений — Управление знаниями.

Принимаемые в организации решения основываются на текущих данных, собственном опыте и экспертных исторических знаниях.

Люди, принимающие решения в организации, часто не являются экспертами в области принятия таких решений. Им нужна дополнительная информация, основанная на знаниях, накопленных экспертами организации и накопленных просто в организации.

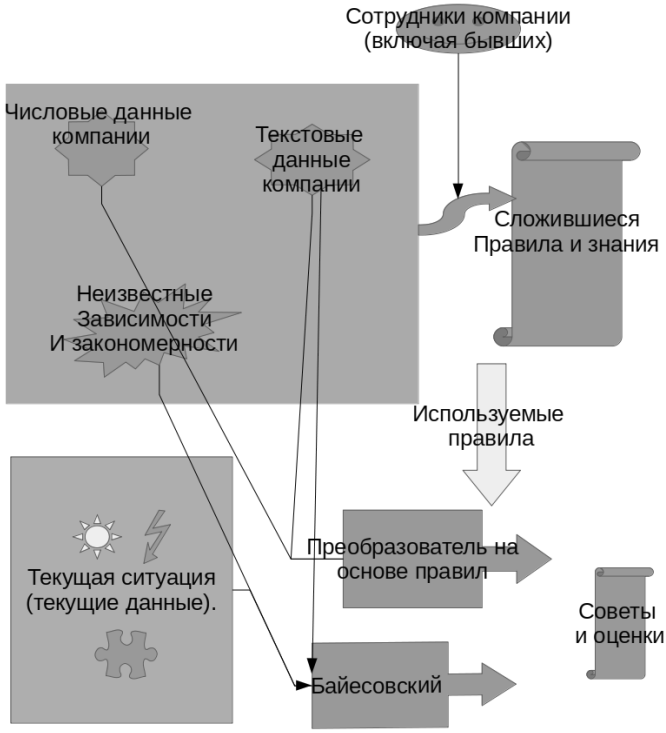
Задача принятия решений.

По имеющимся в организации данным и правилам сгенерировать предложения и оценки, базируясь на текущей информации и состоянии дел в организации

Проблемы

1. знания имеются у людей, которые не умеют программировать и не знают математики
2. знания могут быть неявно выражены, и выражаться весьма хитрыми закономерностями

Для решения этих проблем могут быть применены следующие технологии и инструменты.



Технологии

- Байесовские системы обучения/сортировки
- Системы на основе правил
- Нейронные сети
- Машинное обучение/ прогнозы.

Предлагаемые инструменты

Принципы выбора — поменьше математики, но проверенные и стабильные решения.

- SpamAssasin

Используется как «чёрный ящик», потребляющий обучающую выборку с экспертно-выбранными оценками и затем выдающий оценку для текстового потока входящих сообщений.

- python-module-pyke

Работает на основе правил прямого и обратного вывода, а также может задавать вопросы оператору в процессе вывода. Правила задаются в отдельных файлах.

Области применения

- Оценка безопасности систем
- Прогноз использования расходников
- Оценка пакетной базы
- Прогнозы и оптимизация бизнес-решений.

Игорь Власенко
Киев, ALT Linux Team

Проект: Инфраструктура автоматизации сопровождения пакетов

Инфраструктура автоматизации сопровождения пакетов

Аннотация

Для дистрибутивов ALT Linux (P8, Sisyphus) развернута специализированная инфраструктура для автоматизации ряда задач сопровождения пакетов, начиная от оповещения (watch) и кончая контролем качества (repocop). Обсуждается ее текущее состояние и перспективы.

Работа системных администраторов и инженеров, выпускающих дистрибутивы Linux, исторически связана между собой. До появления дистрибутивов их работа сливалась в одно целое, которое выполнялось каждым администратором системы по отдельности. Появление дистрибутивов позволило снять с системного администратора целый класс задач предварительной подготовки программного обеспечения, оставив за ним только окончательную настройку программного обеспечения и его сопровождение инструментами дистрибутива. В работе системного администратора критерием идеальной работы является

ее отсутствие — идеальная система настроена так, что за ее работой остается только наблюдать.

Этот же идеал приложим и к организации выпуска дистрибутивов. Конечно, он достижим не полностью. Ломка наработанных схем, которая для сисадмина происходит единовременно, во время миграций на новое ПО, при разработке дистрибутивов происходит все время, из-за практически постоянного процесса разработки исходного программного обеспечения.

Тем не менее, разбив процесс сопрвождения ПО в дистрибутивове на ряд мелких шагов, автоматизировать наиболее механические из этих шагов, а в отдельных случаях и все шаги, можно.

Основу разработанной для дистрибутивов ALT Linux инфраструктуры автоматизации представляют собой т.н. «роботы», сложные скрипты, которые запускаются по крону от псеводпользователя и выполняют некоторый шаг автоматизации, либо запускаются вручную с целью контроля, но готовы и к автономному режиму работы. Статус автономно работающих роботов доступен на странице
http://watch.altlinux.org/pub/monitor/index.htm.
С их помощью решается ряд задач сопровождения пакетов, начиная от оповещения (watch), собственно сборки пакетов, и кончая контролем качества (repocop).

В докладе обсуждается текущее состояние имеющейся инфраструктуры автоматизации и ее перспективы.

Андрей Черепанов
Москва, ООО «Базальт СПО»

Проект: Sisyphus altlinux.org

Процедуры и программные средства, упрощающие сборку и сопровождение пакетов

Аннотация

В докладе рассматриваются процедуры и программные средства, снижающие трудозатраты по сборке и поддержки актуальности пакетов в рамках репозитория Sisyphus

При сопровождении большой пакетной базы, которым является репозиторий Sisyphus, для каждого мейнтейнера становятся важными следующие факторы:

- упрощение сборки новых пакетов;
- единообразие типовых файлов .spec (спеков);
- снижение рисков ошибок при написании спеков;
- снижение трудозатрат для создания и сопровождения своих пакетов.

Для решения этих задач были созданы различные инструменты, облегчающие сопровождение пакетов.

Сборка новых пакетов

Для создания спеков по типовым шаблонам был написан простой скрипт *genspec* на Python с шаблонами для обычного пакета со сборкой на autotools (configure;make;make install), noarch (для скриптов), модулей Python, Ruby и Java (со сборкой посредством Maven). В отличие от специализированных скриптов (типа gem2spec) было выбрано создание универсального решения.

Изначально все параметры нужно было указывать в параметрах командной строки:

```
genspec \
-t python \
-n aeroolib \
-v 1.0.0 \
-s "Low level python library for Aeroo Reports" \
-l 'GPLv3' \
-u 'https://github.com/aeroo/aeroolib' \
-c '- Initial build in Sisyphus' \
-d "Low level python library for Aeroo Reports."
```

Позднее скрипт стал работать в том числе и в интерактивном режиме, запрашивая у пользователя поля подстановки. Наличие стандартизованного API у github.com, на котором расположены огромное количество свободных проектов (особенно это касается модулей Ruby), привело к созданию скрипта *github2spec*, который может брать сам код и метаданные для заполнения спека, преобразовывать в gear-репозиторий и добавлять заполненный спек:

```
github2spec -u https://github.com/cryptosphere/rbnacl
```

После этого достаточно закоммитить изменения и отправить на сборку.

Обновление пакетов

Отслеживание новой версии пакета может осуществляться как традиционным способом — через файл .watch (в стиле Debian), так и современным способом — через отслеживание новых тегов в апстримных репозиториях Git (https://www.altlinux.org/Gear/remotes). Обновление пакета с помощью файла .watch осуществляется командой *rpm-uscan*, которая не только скачает архив с новой версией, но и правильно обновит gear-репозиторий и спек.

Использование утилит gear-remotes* из пакета *perl-Gear-Remotes* состоит из следующих этапов:

1. сохранение ссылки на апстримный репозиторий Git: *gear-remotes-save*;
2. возможное исправление фильтрования и преобразования тегов;
3. вызов *gear-remotes-watch* для поиска нового тега;
4. вызов *gear-remotes-uscan* для собственно сборки новой версии в текущем gear-репозитории.

Пересборка пакетов

Ещё одной достаточно трудоёмкой задачей становится массовая пересборка пакетов при обновлении системообразующих тулкитов и библиотек. Например, для обновления версии Ruby потребовалось пересобрать более 80 пакетов. Для массовой пересборки пакетов потребовалось всего лишь для каждого пакета склонировать и обновить gear-репозиторий с https://git.altlinux.org и воспользоваться утилитой *srpmnmu* из пакета *perl-RPM-Source-Editor*:

```
srpmnmu -i "$(gear-rules-print-specfile)" \
  -ch "- Rebuild with Ruby 2.4.1" \
  --nextrel nmuadd
```

после чего закоммитить, установить тег и добавить в общее задание на пересборку.

Глеб Фотенгауэр-Малиновский
Москва, ООО «Базальт СПО»
http://git.altlinux.org/people/glebfm/public/bootstrap.git

Автоматизация портирования ОС «Альт» на новые аппаратные платформы

Аннотация

Альт несколько раз был портирован на различные аппаратные платформы. Среди них x86_64, PowerPC, разные версии архитектуры ARM. Существовал даже экспериментальный порт на x32. Все эти порты делались без учёта того, что опыт этого портирования может пригодиться в будущем. На самом деле такой опыт можно не просто записывать, но и оформлять в виде сценариев для автоматического портирования. Такие сценарии можно исполнять не только тогда, когда нужно получить ОС для новой аппаратной платформы, но и по мере изменения основного репозитория для поддержания инструментов автоматического портирования в работоспособном состоянии.

Стадии портирования

Портирование ОС «Альт» на новую платформу можно разделить на следующие стадии:

1. минимальная сборочная среда (toolchain, shell, make);
2. сборщик пакетов (rpmbuild);
3. сборочная среда в виде пакетов (для сборки в изолированном сборочном окружении);
4. массовая сборка пакетов;
5. пересборка всех собранных пакетов;
6. интеграция новой архитектуры в общую систему сборки пакетов (возможно, в «догоняющем» режиме);

Способы портирования

Есть два основных способа ручного портирования ОС на базе GNU/*/Linux на новую аппаратную платформу:

- на основе существующего порта другого дистрибутива;

- с помощью кросс-компилятора под целевую архитектуру.

Первый способ кажется самым простым, потому что позволяет пропустить стадию 1 и сразу перейти к стадии 2, а сборку toolchain отложить до стадии 3. Второй способ кажется более сложным, но позволяет собирать порт без зависимости от других дистрибутивов, или если порт на эту архитектуру вообще отсутствует.

Сложность портирования

Бóльшая часть проблем в процедуре портирования, как ни странно, касается не особенностей конкретной архитектуры, а устройства программных компонентов, которые часто требуют других компонентов, которые ещё не собраны. Такие зависимости часто образуют циклы, которые приходится «разрывать» для каждого продукта по-своему.

Автоматизация портирования

Всю работу, проделываемую при портировании, которая касается сложности портирования, а не особенностей конкретной архитектуры, можно автоматизировать. В таком случае можно будет поддерживать систему автоматического портирования в работоспособном состоянии, а при необходимости сделать порт на новую платформу решать уже только проблемы конкретной платформы.

Результаты

На данный момент готов набор скриптов, который позволяет полностью автоматически собрать среду для кросс-компиляции, стадию 1 и стадию 2.

Михаил Шигорин
Москва, Базальт СПО

«Эльбрус» на Альте

Аннотация

В докладе рассматривается интеграция поддержки архитектуры «Эльбрус-2000» в репозиторий проекта Сизиф, её основные этапы, задачи и текущее состояние, а также особенности архитектуры и её сравнительные возможности.

Состояние: 2017

В докладе 2016 года речь шла о текущем результате в виде полутысячи альтовских пакетов, собранных нативно на рабочей станции «Эльбрус-401» в чруте под управлением ОС Эльбрус; за год произошли существенные сдвиги, позволившие задуматься над добавлением поддержки архитектуры e2k в нашу сборочницу:

- «Эльбрус-401» с весны 2017 года загружен под альтом;
- объём репозитория перевалил за 2500 пакетов;
- собраны многие ключевые библиотеки;
- введён в эксплуатацию сервер «Эльбрус-4.4».

Среди заметных результатов — важные для возможности сборки целых сегментов Sisyphus библиотеки Boost и Qt 5.

В чисто прикладном плане за это время была обеспечена возможность применения «Эльбрус-401» под управлением ОС Альт в качестве базовой рабочей станции (несколько графических сред, включая Xfce, LXQt, MATE; браузер Firefox ESR; разнообразные утилиты), а также развита серверная часть вплоть до поддержки функционирования в качестве контроллера домена Active Directory.

Задачи

Обозримая часть маршрута выглядит так:

- базовая сборочная среда;
- наработка репозитория;

- развёртывание транзакционной сборочницы;
- перевод в режим «догоняющей» сборки;
- публикация репозитория.

На сегодня мы дошли до пробного развёртывания girar-builder (системы, обеспечивающей возможность транзакционного изменения репозитория по мере новых сборок пакетов), но упёрлись в необходимость выставления существенного количества «ручек» –without/–disable, отключающих недоступную сейчас функциональность (например, сборку с ещё не собранными библиотеками).

Для полноценного добавления поддержки архитектуры, тем не менее, остаётся нерешённым ряд задач технического плана. Среди них наведение порядка в имеющемся репозитории: уже произведено устранение остаточных alien-пакетов, на сейчас таковыми остались компилятор и его библиотеки, но пока остаются многие временные пакеты, формально предоставляющие недостающие требуемые где-либо зависимости или являющиеся «мостиками» при различии имён. Также продолжается сборка недостающих ключевых компонентов, что позволит продолжить включение отключенных на предыдущей стадии портирования «ручек»; всё-таки предстоит перенос остаточных «отключаторов» в конфигурацию сборщиков, так как все сто процентов доступного программного обеспечения пересобрать заведомо не получится в силу объективных различий платформ и их экосистем (например, для e2k нет смысла собирать intel-gpu-tools).

Особенности архитектуры

Сизиф портирован на несколько аппаратных архитектур и каждая из них чем-то да выделяется; у архитектуры «Эльбрус-2000» в целом на сегодня наиболее выделяющимся отличием является отсутствие компилятора gcc – в качестве системного компилятора применяется lcc, имеющий как неоспоримые достоинства в плане возможностей оптимизации (аналогичные выигрышу icc на x86/ia64, только более ярко выраженные), так и проблемы любого не-gcc в плане совместимости с GNU-расширениями стандарта языка Си и конкретными опциями вызова.

В силу выбранного типа архитектуры (VLIW с явным параллелизмом) характерна сравнительно невысокая скорость работы компилятора, усугубляющаяся тем, что оптимизация оптимизирующе-

го компилятора для работы на таких платформах особенно сложна; при этом практического быстродействия достаточно для пересборки за сутки более чем 1600 пакетов на шестнадцати процессорных ядрах.

По части аппаратного обеспечения (используются системы на базе микропроцессора «Эльбрус-4С») можно выделить такие особенности:

- невысокая скорость работы относительно текущих x86;
- хорошая перегрузочная способность (меньше «проседает»);
- хорошая масштабируемость многопроцессорной системы;
- поддержка больших объёмов памяти.

В частности, время сборки больших проектов (ядро Linux, браузер Firefox) на четырёхпроцессорной машине уменьшалось практически линейно по сравнению с однопроцессорной.

Значительный объём ОЗУ — 24 Гб штатно на рабочей станции, 96 Гб на сервере – сильно облегчает параллельную сборку и сборку крупных пакетов, не вызывая подкачки страниц памяти с диска (на ARMv7 и MIPS с этим было куда сложней, что лишь усугубляло недостаток производительности процессора).

При этом надо понимать, что потенциальная пропускная способность интерфейсов DDR3 и SATA2 на 4С задействована не в полной мере.

Стоит отметить, что итоговая производительность системы на VLIW-архитектуре сильно зависит от того, насколько удалось реализовать параллелизм выполнения кода при его компиляции; поэтому на одном и том же ПК «Эльбрус-401» система может напоминать как Pentium III, так и Core2 по производительности в зависимости от версии компилятора и переданных ему опций. Разумеется, порой решающее значение имеют всё-таки ручные правки кода, которые мы всё так же получаем от коллег из МЦСТ.

Ссылки

1. http://altlinux.org/ports/e2k
2. http://0x1.tv/20170128J
3. http://lvee.org/ru/abstracts/251
4. http://sdelanounas.ru/blogs/96816
5. http://mcst.ru

Дмитрий Левин
Москва, Базальт СПО

Проект: strace https://strace.io

strace: новые возможности

Аннотация

strace — инструмент для отслеживания и влияния на взаимодействия пользовательских процессов и ядра Linux: системных вызовов, сигналов и изменений состояния процесса. За минувший год в strace реализовано много нового и интересного.

Введение

strace как инструмент мониторинга взаимодействия пользовательских процессов с ядром существует уже почти 26 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 летом прошлого 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.

Помимо многочисленных улучшений отображения системных вызовов, за минувший год в strace было реализовано много нового и интересного.

System Call Specification

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 % ещё поддерживается, но уже считается устаревшим.
- 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.

System call tampering и fault injection

Механизм syscall fault injection, прототип которого подробно рассматривался на этой конференции год назад, был доработан и включён в strace, начиная с версии 4.15, выпущенной в декабре прошлого года. По сравнению с прототипом, синтаксис syscall fault injection изменился и выглядит следующим образом:

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

Начиная с версии 4.16, выпущенной в феврале прошлого года, реализован более общий механизм вмешательства в системные вызовы, который, помимо syscall fault injection, позволяет осуществлять syscall return value injection и signal injection. Интерфейс этого нового механизма выглядит следующим образом:

-e inject=*set*[:error=*errno*|:retval=*value*][:signal=*sig*][:when=*expr*]

Netlink socket parsers

В strace версии 4.19, выпущенной в начале сентября, реализовано детальное декодирование трафика, проходящего через netlink sockets,

что позволяет использовать strace для отладки приложений, работающих с netlink-протоколами. Так, например, может выглядеть вывод парсера трафика семейства NETLINK_ROUTE:

$ hsh-run –mount=/proc – strace -e trace=sendto,recvmsg ip r l t all

sendto(3, {{len=40, type=RTM_GETROUTE, flags=NLM_F_REQUEST|NLM_F_DUMP, seq=1357924680, pid=0}, {rtm_family=AF_UNSPEC, rtm_dst_len=0, rtm_src_len=0, rtm_tos=0, rtm_table=RT_TABLE_UNSPEC, rtm_protocol=RTPROT_UNSPEC, rtm_scope=RT_SCOPE_UNIVERSE, rtm_type=RTN_UNSPEC, rtm_flags=0}, {nla_len=0, nla_type=RTA_UNSPEC}}, 40, 0, NULL, 0) = 40

recvmsg(3, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000}, msg_namelen=12, msg_iov=[{iov_base=[{{len=60, type=RTM_NEWROUTE, flags=NLM_F_MULTI, seq=1357924680, pid=12345}, {rtm_family=AF_INET, rtm_dst_len=32, rtm_src_len=0, rtm_tos=0, rtm_table=RT_TABLE_LOCAL, rtm_protocol=RTPROT_KERNEL, rtm_scope=RT_SCOPE_LINK, rtm_type=RTN_BROADCAST, rtm_flags=0}, [{{nla_len=8, nla_type=RTA_TABLE}, RT_TABLE_LOCAL}, {{nla_len=8, nla_type=RTA_DST}, 127.0.0.0}, {{nla_len=8, nla_type=RTA_PREFSRC}, 127.0.0.1}, {{nla_len=8, nla_type=RTA_OIF}, if_nametoindex("lo")}]}, {{len=60, type=RTM_NEWROUTE, flags=NLM_F_MULTI, seq=1357924680, pid=12345}, {rtm_family=AF_INET, rtm_dst_len=8, rtm_src_len=0, rtm_tos=0, rtm_table=RT_TABLE_LOCAL, rtm_protocol=RTPROT_KERNEL, rtm_scope=RT_SCOPE_HOST, rtm_type=RTN_LOCAL, rtm_flags=0}, [{{nla_len=8, nla_type=RTA_TABLE}, RT_TABLE_LOCAL}, {{nla_len=8, nla_type=RTA_DST}, 127.0.0.0}, {{nla_len=8, nla_type=RTA_PREFSRC}, 127.0.0.1}, {{nla_len=8, nla_type=RTA_OIF}, if_nametoindex("lo")}]}, {{len=60, type=RTM_NEWROUTE, flags=NLM_F_MULTI, seq=1357924680, pid=12345}, {rtm_family=AF_INET, rtm_dst_len=32, rtm_src_len=0, rtm_tos=0, rtm_table=RT_TABLE_LOCAL, rtm_protocol=RTPROT_KERNEL, rtm_scope=RT_SCOPE_HOST, rtm_type=RTN_LOCAL, rtm_flags=0}, [{{nla_len=8, nla_type=RTA_TABLE}, RT_TABLE_LOCAL}, {{nla_len=8, nla_type=RTA_DST}, 127.0.0.1}, {{nla_len=8, nla_type=RTA_PREFSRC}, 127.0.0.1}, {{nla_len=8, nla_type=RTA_OIF}, if_nametoindex("lo")}]}, {{len=60, type=RTM_NEWROUTE, flags=NLM_F_MULTI, seq=1357924680, pid=12345}, {rtm_family=AF_INET, rtm_dst_len=32, rtm_src_len=0, rtm_tos=0, rtm_table=RT_TABLE_LOCAL, rtm_protocol=RTPROT_KERNEL, rtm_scope=RT_SCOPE_LINK, rtm_type=RTN_BROADCAST, rtm_flags=0}, [{{nla_len=8, nla_type=RTA_TABLE}, RT_TABLE_LOCAL}, {{nla_len=8, nla_type=RTA_DST}, 127.255.255.255}, {{nla_len=8, nla_type=RTA_PREFSRC}, 127.0.0.1}, {{nla_len=8, nla_type=RTA_OIF}, if_nametoindex("lo")}]}], iov_len=32768}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 240

broadcast 127.0.0.0 dev lo table local proto kernel scope link src 127.0.0.1

local 127.0.0.0/8 dev lo table local proto kernel scope host src 127.0.0.1

local 127.0.0.1 dev lo table local proto kernel scope host src 127.0.0.1

broadcast 127.255.255.255 dev lo table local proto kernel scope link src 127.0.0.1

recvmsg(3, {msg_name={sa_family=AF_NETLINK, nl_pid=0, nl_groups=00000000},

```
msg_namelen=12, msg_iov=[{iov_base={{len=20, type=NLMSG_DONE, flags=NLM_F_MULTI, seq=
1505349241, pid=12345}, 0}, iov_len=32768}], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 20
```

Advanced syscall filtering syntax

В рамках GSoC 2017 Николай Марчук реализовал прототип ещё более гибкого синтаксиса фильтрации системных вызовов:

[***action***(|***filter_expression***[;***arg1***[;***arg2***...]]|)]

где

action это **trace**, **abbrev**, **verbose**, **raw**, **read**, **write**, **fault**, **inject**, или **stacktrace**;

arnN это аргументы ***action***;

filter_expression состоит из комбинации фильтров, каждый их которых, в свою очередь, состоит из имени фильтра и аргументов фильтра.

Поддерживаются следующие фильтры:

syscall *set* : множество системных вызовов, соответствующих описанию ***set***.

fd *fd1...* : множество системных вызовов, оперирующих дескрипторами, соответствующими описанию ***fd1***...;

path *path* : множество системных вызовов, оперирующих файлами, соответствующими описанию ***path***.

Комбинация фильтров осуществляется с помощью операторов **not**, **and**, **or** и круглых скобок.

Более подробно новый синтаксис описан в strace(1).

Advanced syscall tampering and filtering with Lua/LuaJIT

В рамках GSoC 2017 Виктор Крапивенский реализовал прототип Lua-скриптинга, который позволяет не только производить фильтрацию и подмену системных вызовов с большей гибкостью, но и производить success injection с сохранением семантики системного вызова, которая, в частности, может заключаться в записи определённых данных в адресное пространство процесса. Подробнее об этом расскажет автор в своём докладе.

Advanced syscall information tool

В рамках GSoC 2017 Эдгар Казиахмедов реализовал прототип новой утилиты asinfo, предназначенной для получения разнообразной информации о системных вызовых. Подробнее об этом расскажет автор в своём докладе.

Крапивенский Виктор
Москва, Московский Физико-Технический Институт

Проект: strace https://strace.io

Lua-скриптинг в strace

Аннотация

strace — инструмент для отслеживания взаимодействия пользовательских процессов и ядра Linux: системных вызовов, сигналов и изменений состояния процесса. Ранее, в рамках одного из проектов GSoC 2016, в strace была добавлена функция fault injection, которая позволяет подменять результаты системных вызовов.

В рамках проекта GSoC 2017 в strace появилась поддержка Lua-скриптинга, которая позволяет не только производить фильтрацию и подмену системных вызовов с большей гибкостью, но и производить success injection с сохранением семантики системного вызова, которая, в частности, может заключаться в записи определённых данных в адресное пространство процесса.

Обзор

В рамках одного из проектов GSoC 2016 в strace была добавлена поддержка подмена кодов возврата системных вызовов.

Была реализована возможность подменять коды возврата произвольного подмножества системных вызовов; при этом, логика ограничивалась лишь счётчиком — можно было подменять результаты всех системных вызовов, только N-ного, либо N-ного и далее каждого K-того.

При этом, не было возможности производить success injection с сохранением семантики системного вызова, которая, в частности, может заключаться в записи определённых данных в адресное пространство процесса.

В этом году, strace получил поддержку Lua(JIT)-скриптинга, что открывает новые возможности в этом направлении.

Lua, LuaJIT и FFI

Lua — легковесный встраиваемый скриптовый язык программирования.

LuaJIT — Just-In-Time компилятор для Lua, называемый «одной из самых производительных реализаций динамических языков программирования».

LuaJIT также предоставляет набор библиотек и расширений, отсутствующих в стандартной поставке Lua.

Наиболее важна для strace библиотека ffi, которая позволяет взаимодействовать с кодом на Си и включает в себя парсер деклараций типов и констант, поддерживающий, за небольшими незначительными исключениями, C99 в полной мере.

```
ffi.cdef[[
typedef struct foo { int a, b; } foo_t;  // Declare a
    struct and typedef.
int dofoo(foo_t *f, int n);  /* Declare an external C
    function. */
]]
```

Библиотека `ffi` также даёт возможность создавать объекты, соответствующие известным типам данных Си, и производить с ними различные манипуляции.

Существуют также отдельные реализации библиотеки ffi для обычной реализации Lua. Они также поддерживаются strace.

Реализация

strace «скармливает» библиотеке ffi декларацию типа `struct tcb` — типа, поля которого совпадают с первыми полями структуры, в которой strace хранит информацию о текущем состоянии системного вызова для каждого процесса:

```
/* Trace control block */
struct tcb {
 int pid;                                /* Tracee's PID */
 unsigned long u_error;                  /* Error code */
```

```
kernel_ulong_t scno;                    /* System call
    number */
kernel_ulong_t u_arg[/* MAX_ARGS */];/* System call
    arguments */
kernel_ulong_t u_rval;                  /* Return value */
unsigned int currpers;                  /* Current
    personality */
```

А также декларации ещё нескольких необходимых структур и типов: Также, strace, через модуль strace.C, предоставляет функции и константы, необходимые для обеспечения функциональности скриптинга.

После этого загружается библиотека, написанная на Lua, предоставляющая удобные обёртки над этими функциями и константами.

Основное взаимодействие с strace осуществляется через функции strace.next_sc, strace.C.monitor и strace.C.monitor_all.

strace хранит множество тех системных вызовов, которые нужно возвратить из strace.next_sc при входе в системный вызов, и множество тех, которые нужно возвратить по выходу из него.

Изначально оба этих множества пусты; библиотека или пользователь могут выбрать несколько системных вызовов по номеру для мониторинга с помощью strace.C.monitor, или же потребовать информировать скрипт обо всех системных вызовах с помощью strace.C.monitor_all.

Библиотека предоставляет более удобный интерфейс информирования о системных вызовах, а именно возможность повесить функцию-обработчик на вход либо выход из системного вызова по его номеру, имени, либо классу системных вызовов.

Примеры

Подсчёт количества порождённых процессов:

```
n = 0
assert(strace.hook({'clone', 'fork', 'vfork'}, 'exiting',
    function(tcp)
  if tcp.u_rval ~= -1 then
    n = n + 1
  end
end))
```

```
strace.at_exit(function() print('Processes spawned:', n)
    end)
```

Использование препроцессора для извлечения констант из заголовочных файлов:

```
ffi = require 'ffi'
f = assert(io.popen([[cpp - <<EOF | grep -v '^#'
#define _GNU_SOURCE
#include <fcntl.h>
enum { f_setpipe_sz = F_SETPIPE_SZ };
EOF]], 'r'))
ffi.cdef(f:read('*a'))
f:close()
assert(strace.hook({'fcntl', 'fcntl64'}, 'entering',
    function(tcp)
  if tcp.u_arg[1] == ffi.C.f_setpipe_sz then
    assert(strace.inject_error('EPERM'))
  end
end))
```

Использование препроцессора для извлечения определений структур из заголовочных файлов:

```
$ uname
Linux
$ strace -l pretend-win.lua -e trace=none uname
Windows
+++ exited with 0 +++
$ cat pretend-win.lua
ffi = require 'ffi'
f = assert(io.popen([[cpp - <<EOF | grep -v '^#'

#include <sys/utsname.h>
EOF]], 'r'))
ffi.cdef(f:read('*a'))
f:close()
assert(strace.hook('uname', 'exiting', function(tcp)
  if tcp.u_rval == -1 then
    return
  end
  local u = assert(strace.read_obj(tcp.u_arg[0], 'struct
      utsname'))
```

```
  local s = 'Windows'
  assert(ffi.sizeof(u.sysname) >= #s + 1)
  ffi.copy(u.sysname, s)
  assert(strace.write_obj(tcp.u_arg[0], u))
end))
```

Текущее состояние

Правки пока не приняты в strace; текущее состояние проекта можно отслеживать в репозитории
https://github.com/shdown/strace/tree/luajit-current.

Эдгар Казиахмедов, Дмитрий Левин
Москва, МФТИ(ГУ)

Проект: strace https://strace.io/

Расширенный инструмент для вывода информации о системных вызовах

Аннотация

Утилита strace предназначена для диагностики и отладки программ, написанных под ОС Linux. За годы работы, инструмент приобрел уникальную в своем роде базу системных вызовов для различных архитектур, таких как microblaze, riscv, avr32, всем известная x86 и т.д. Утилита `asinfo` в свою очередь оперирует с данной базой системных вызовов и предоставляет любую информацию о них в наиболее удобном формате.

Введение и обзор проблемы

Год за годом `strace` становится все более мощным инструментом для отладки приложений, написанных под Linux. Помимо этого, инструмент собирается под огромное количество архитектур. Прежде всего, при добавлении очередной платформы необходимо предоставить список системных вызовов, что представляет собой довольно трудоемкий процесс, так как каждому системному вызову необходимо задать такие свойства как группа и количество входных аргументов.

В силу этого, на данный момент база системных вызовов, предоставляемая strace не имеет аналогов. Для наглядности рассмотрим объявление для системного вызова `open` для архитектуры x86_64:

```
[   2] = {  3, TD|TF, SEN(open), "open" },
```

Как видно, объявление этого системного вызова требует человеческого вмешательства, так как генерация объявлений не может быть унифицирована путем обработки скриптом исходных кодов ядра Linux. Несмотря на обширность данной базы, она все еще не может нести дескриптивного характера для конечного пользователя. Попросту говоря, необходим дополнительный слой абстракции в виде программы-оператора. Поэтому была поставлена задача - разработать программу `asinfo`, которая могла бы работать с массивами системных вызовов для каждой поддерживаемой `strace` архитектуры и выводить о них доступную информацию.

Существующие решения

Представим, что мы решили настроить журналирование всех обращений к 80-ому порту, проходящих через системный вызов `listen`. Воспользуемся подсистемой audit:

```
auditctl -a exit,always -S listen
```

Теперь оказалось, что у нас 64/32-битное пользовательское окружение. В таком случае хотелось бы, чтобы `listen` улавливался в обоих случаях. По сути, для этого необходимо лишь убедиться, что номер системного вызова одинаков и там и там. На сегодняшний день вариантов всего два:

- Воспользоваться программой `ausyscall`[1], входящей в пакет `auditd`;
- Найти номер в исходных кодах ядра.

Стоит отметить, что `ausyscall` предоставляет довольно скудный набор архитектур и не поддерживает возможные двоичные интерфейсы приложения. Так или иначе, даже не принимая во внимания данных фактов, существующие решения не позволяют получить необходимую информацию о системных вызовах за один раз с возможностью гибкой настройки входных параметров. Таким образом, этот факт лишь еще раз доказывает необходимость реализации утилиты `asinfo`.

Архитектура программы

На основе проведенного анализа, к программе `asinfo` были выдвинуты следующие требования:

- Предоставлять краткую информацию об архитектуре;
- Учитывать существующие двоичные интерфейсы приложений;
- Печатать название системного вызова по его номеру и наоборот;
- Предоставлять возможность поиска точных совпадений и по вхождению (название системного вызова);
- Предоставлять возможность использования regex;
- Выводить системные вызовы по группам: network, desc, file и т.д.;
- Предоставлять возможность работы с несколькими архитектурами за раз, для наблюдения расхождений в параметрах;
- Определять по возможности двоичный интерфейс окружения;
- Использовать форматированный/сырой формат вывода.

На основе данных требований программа приобрела архитектуру, показанную на рис. 1.

Как видно, программа представляет собой конвейер. На первом этапе command_dispacher занимается тем, что обрабатывает входные параметры и проверяет их корректность, тем самым гарантируя отсутствие ошибок последующим блокам. Далее, arch_dispatcher разворачивает сжатый формат описания архитектур `architecture` и, основываясь на входных данных `reqs`, заполняет `arch_list`. На данном этапе, `arch_list` включает в себя все возможные двоичные интерфейсы, поэтому `abi_dispatcher` занимается тем, что отфильтровывает `arch_list`, основываясь либо на окружении, либо на входных данных. В последнюю очередь включается `syscall_dispatcher`, цель которого уже для маркированных бинарных интерфейсов отметить указанный набор системных вызовов. Стоит отметить, что `syscall_dispatcher` в своем составе использует заимствованную из `strace` фильтрацию системных вызовов. Таким образом, например, мы можем вывести все системные вызовы, работающие с файловым дескриптором для архитектуры `x86_64` с 64-битным и x32 интерфейсами и посмотреть расхождения в номерах одной командой:

```
asinfo --set-arch x86_64,x86_64 --set-abi 64bit,x32 --get
    -sname %desc
```

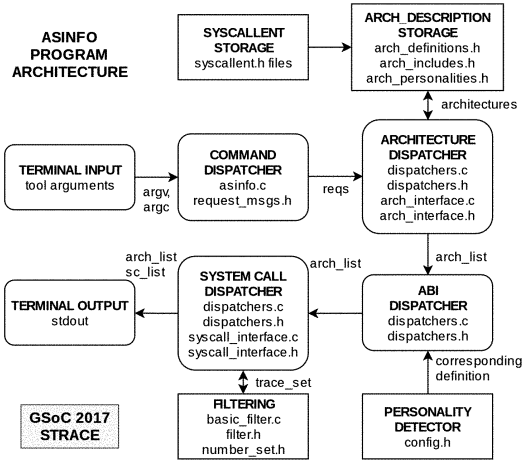


Рис. 1: Архитектура программы asinfo

Текущее состояние

На данный момент проведены все подготовки в кодовой базе `strace` для слияния кода `asinfo`[2] и выполнены поставленные задачи. Из не сделанного — отсутствие готовых к слиянию тестов.

Ссылки

1. ausyscall(8) — Linux man page, [HTML]
https://linux.die.net/man/8/ausyscall
2. GSoC 2017: Work product submission, [HTML]
https://edosedgar.github.io/gsoc2017/2017/08/27/work-product-submission.html

Александр Боковой
Эспоо, Финляндия, Red Hat

Проект: Samba https://www.samba.org

Samba AD и MIT Kerberos

Аннотация

Samba 4.7 — первая версия Samba AD, которая поддерживает работу с MIT Kerberos вместо Heimdal. Доклад расскажет о том, почему потребовалось более пяти лет, чтобы реализовать поддержку MIT Kerberos и что из этого вышло.

Впервые поддержкой режима контроллера домена Microsoft Active Directory в рамках проекта Samba занялись в 2002 году. Архитектура контроллера домена Active Directory подразумевает одновременную доступность к информации об учетных записях и их состоянии средствами разных протоколов. Изменения, вносимые посредством вызовов DCE RPC, моментально отражаются и в LDAP, и в Kerberos. Это фактически означало, что контроллер домена на основе Samba должен интегрировать LDAP и Kerberos так, чтобы контекст обработки данных был единым между тремя разными системами.

На тот момент этого можно было достичь только тесно интегрировав между собой все компоненты. К тому же, многие из расширений протоколов LDAP и Kerberos, внесенные Microsoft в Active Directory, не были реализованы в свободных проектах (OpenLDAP, Heimdal или MIT Kerberos). Тесная интеграция означала не просто исполнение кода с доступом к единой базе данных, требовалось изменить и код серверов LDAP и Kerberos единым образом. Фактически, на какое-то время нужно было трем проектам (Samba, OpenLDAP и какой-то из свободным реализаций Kerberos) стать единым целым с точки зрения координации изменений и выпуска версий. Этот подход оказался крайне сложным, поскольку все проекты преследовали свои цели и не были готовы к такой плотной координации.

В результате, Эндрю Триджелл, автор Samba, в тот момент работавший в IBM Research, принял решение написать свою реализацию сервера LDAP (и базы данных для него), а также интегрировать в Samba реализацию Kerberos из проекта Heimdal. Последнее решение было обосновано тем, что Heimdal позволял интегрировать сервер

ключей Kerberos (KDC) как библиотеку уже в существующую программу. MIT Kerberos такой возможностью не обладал и не обладает до сих пор.

Версия Samba, поддерживающая работу контроллера домена Active Directory, была интегрирована в основную ветку проекта Samba и выпущена в эксплуатацию как Samba 4.0.0 в 2012 году. Она активно использовалась в различных организациях и поставлялась в дистрибутивах на основе Debian и Ubuntu, однако отсутствовала в Fedora / CentOS / Red Hat Enterprise Linux и openSUSE / SUSE Linux Enterprise Server. Эти дистрибутивы использовали MIT Kerberos вместо Heimdal. Оба проекта совместимы на уровне сетевого протокола, но внутренние структуры в предоставляемых библиотеках организованы по-разному и не могут быть использованы одновременно.

Когда в 2012 году встал вопрос о поддержке MIT Kerberos в Samba AD DC, стало ясно, что требуется дополнительно решить целый ряд задач совместимости. Как уже говорилось, функционал KDC в MIT Kerberos нельзя было интегрировать в существующее приложение. Heimdal Kerberos предоставлял ряд внутренних функций, которые отсутствовали в MIT Kerberos и за десятилетие развития Samba AD DC стали неотъемлемой частью кода Samba. К тому же, необходимо было добиться полного покрытия тестами среды исполнения контроллера домена Active Directory при использовании группы процессов.

Реализация этих задач потребовала создания новых системных компонент. Поскольку контейнеризация процессов на момент слияния ветки Samba AD DC с основной веткой разработки Samba в 2005 году была еще неразвитой, Samba Team пошла по пути создания среды, которая симулировала пересылку данных по сети (`socket_wrapper`, Йелмер Верноой, 2005 год), чтение информации о пользователях и доменных именах через системные API (`nss_wrapper`, Стефан Метцмахер, 2007 год), изоляцию приложений (`uid_wrapper`, Эндрю Триджелл, 2008 год), прямые вызовы DNS (`resolv_wrapper`, Андреас Шнайдер, 2014 год), а также аутентификацию приложений (`pam_wrapper`). Все эти компоненты стали частью проекта CWrap (https://www.cwrap.org) и успешно используются при тестировании многих других проектов.

Работа по поддержке MIT Kerberos была сложной. Во-первых, необходимость понимания двух независимых реализаций не самого простого протокола существенно сужала круг возможных исполнителей. Во вторых, эта работа требовала времени и финансирования. Ан-

дреас Шнайдер и Гюнтер Дечнер из Red Hat фактически оказались единственными, кто сумел сосредоточиться на переносе на годы. К маю 2014 стало возможным вручную запустить сервер KDC из MIT Kerberos как часть Samba.

Вся эта работа сопровождалась обнаружением различных недостатков как в компонентах CWrap, так и ошибках или отсутствию необходимых вызовов в MIT Kerberos. Параллельно разработка Samba AD DC не стояла на месте и всё больше функций Heimdal Kerberos, несовместимых с MIT Kerberos, обнаруживалось в различных компонентах Samba.

К лету 2015 года набор исправлений для поддержки MIT Kerberos в Samba AD DC состоял из более чем 140 патчей. Все их предстояло рассмотреть и включить в основной код Samba. К этому моменту еще не работали 69 тестовых конфигураций. Забегая вперед, к лету 2017 в режиме сборки с MIT Kerberos в тестировании Samba AD DC исполнялось 14658 тестов из 2030 тестируемых сценариев, это полный набор тестов в Samba 4.7.

В августе 2015 половина исправлений уже была интегрирована в основной код Samba. Samba научилась запускать внутри себя KDC и сервис управления учетными записями (kadmind). В этот момент пришло осознание, что изменение свойств учетных записей в протоколе kadmin не поддерживает динамическое разграничение доступа (access control lists). Работа над соответствующим API в MIT Kerberos ведется до сих пор — катализатором этой работы выступает проект FreeIPA. В Samba Team было принятно решение реализовать собственную версию протокола kadmin, чтобы интегрировать со внутренней моделью безопасности, совместимой с ACL в Active Directory.

Active Directory позволяет реплицировать данные между контроллерами в рамках домена. Для этого, в частности, необходимо извлекать данные ключей Kerberos, правильно их шифровать в соответствии с протоколом MS-BKRP и пересылать между системами. Чтобы сделать эту часть функционала независимой от реализации Kerberos, код обработки ключей был переписан на использование GnuTLS. Выяснилось, что GnuTLS содержит ошибки и не содержит некоторые необходимые для MS-BKRP алгоритмы, в частности, RC4. Их поддержка была добавлена в GnuTLS в конце 2015.

Работа над поддержкой MIT Kerberos была заморожена в конце 2015 до середины весны 2016 из-за серии уязвимостей Badlock. Когда же работа возобновилась летом 2016, выяснилось, что Samba невер-

но рассчитывает хэши для паролей при установке доверительных отношений между доменами. Всё это потребовало полное переписывание механизма передачи клиентских паролей и ключей внутри libsmb. Вдобавок, поведение некоторых общих для MIT Kerberos и Heimdal функций инициализации данных из хранилищ ключей различалось, а некоторые функции GSSAPI, позволяющие изолировать эту разницу, отсутствовали в Heimdal.

В декабре 2016 года MIT Kerberos выпустил версию 1.15, которая должа была содержать практически все исправления, необходимые для Samba. Увы, эта версия также убрала функцию, позволявшую модулям базы данных в KDC создавать свои структуры данных и уничтожать их при уничтожении данных о принципалах внутри KDC. Эта ошибка была исправлена в версии 1.15.1, где необходимый деструктор данных был снова добавлен.

Все оставшиеся патчи и исправления в различных компонентах Samba были добавлены в основную ветку разработки 30 апреля 2017 года. Samba AD DC с MIT Kerberos поддерживает работу как контроллер домена, доверительные отношения уровня леса предприятия и внешние доверительные отношения с доменами NT. Еще предстоит реализовать целый пласт функционала:

- поддержка PKINIT и смарткарт
- поддержка имперсонификации на уровне Kerberos (S4U2Self, S4U2Proxy), в том числе и между доменами
- поддержка контроллера домена в режиме только для чтения (для этого и нужна библиотека KDC)

Пять лет были потрачены на итеративное обнаружение ошибок, недоработок и отсутствие интеграции между различными свободными проектами. Многие из этих недоработок были критичными для полного покрытия кода функциональными тестами. Можно ли было бы добиться этого взаимодействия между проектами в 2002-2012? Наверное да, но были и сдерживающие факторы, в первую очередь отсутствие эффективных средств ведения распределенной разработки. Проект Git появился в 2005 году как реакция на разногласия между Эндрю Триджеллом (Samba, Rsync) и Ларри МакВоем (BitKeeper). Фактически, ошибка Торвальдса в оценке ситуации с анализом сетевого протокола BitKeeper, выполненным Эндрю Триджеллом, породила git как систему распределенной разработки и навсегда изменила подход к взаимодействию между многими открытыми проектами. В

2002-2012 тот процесс, который занял у Андреаса Шнайдера и Гюнтера Дечнера пять лет, мог легко растянуться на восемь и больше лет. К тому же, в тот период не нашлось желающих эту работу профинансировать.

Евгений Синельников
Саратов, ООО «Базальт СПО»

Проект: Samba stand with Vagrant
http://github.com/mastersin/samba-stand

Отладка и валидация сложных инфраструктурных решений с помощью Vagrant

Аннотация

Воспроизводимость проблемы является основным условием возможности её исправления. В случае сложных инфраструктурных решений из нескольких узлов, многократное соблюдение этого условия становится всё более трудоёмким. В этом докладе представлен один из эффективных сценариев отладки развёртывания и валидации конфигурации Active Directory на базе Samba с помощью механизма управления конфигурациями виртуальных машин Vagrant.

Отладка сложных клиент-серверных решений, которые для воспроизведения рабочих конфигураций требуют несколько вычислительных узлов, существенно упрощается при использовании современных средств виртуализации. Важной характеристикой таких средств, для задач отладки, является возможность автоматического создания, запуска, остановки и контроля за состоянием виртуальных машин с помощью команд внешнего управления средой виртуализации. В более широком виде такие возможности предоставляются специализированными средствами управления конфигурациями. Одним из таких средств является открытый проект Vagrant [1]. В данной работе рассматривается вариант отладки рабочих станций и контроллеров домена для клиент-серверных решений на базе проекта Samba.

Для создания и конфигурирования виртуальной среды Vagrant поддерживает базовый набор встроенных средств виртуализации (VirtualBox, Hyper-V и Docker), который может быть расширен с помощью сторонних расширений [2]:

```
$ gem list --remote ^vagrant-
          | head -e vmware-fusion -e 'aws ' -e 'xenserver '
vagrant-aws (0.7.2)
vagrant-vmware-fusion (0.0.1)
vagrant-xenserver (0.0.14)
$ gem list --remote ^vagrant- | wc -l
507
```

Средства виртуализации в терминологии среды управления конфигурациями Vagrant называются «провайдерами» (Providers), а механизмы управления виртуальными машинами — «развёртывателями» (Provisioners). В качестве «развёртывателей» Vagrant поддерживает Ansible, Chef, Puppet и Salt, а также копирование файлов (File) и запуск скриптов через SSH и WinRM (Shell).

Для настройки виртуальных машин, прежде всего настроек сети, Vagrant поддерживает определение различных Linux дистрибутивов. Начиная с версии 2.0.0 в него включена поддержка определения дистрибутивов ALT и настройка etcnet, включая возможность задания имени хоста, маски подсети в CIDR-нотации и управления через NetworkManager. Минимальный вариант рабочей конфигурации (Vagrantfile) с двумя видами сетевых интерфейсов выглядит следующим образом:

```
Vagrant.configure("2") do |config|
  config.vm.box = "mastersin/basealt-p8-server-systemd"
  config.vm.network "private_network", ip: "192.168.2.2",
                        netmask: "24", gateway: "192.168.2.1"
  config.vm.network "public_network", nm_controlled: "yes"
  config.vm.hostname = "example.com"
end
```

Стенд для отладки Samba

Основная задача отладки Samba состояла в том, чтобы в заданном наборе дистрибутивных решений ALT (как серверных, так и клиентских) иметь возможность сказать:

- какие механизмы, необходимые для развёртывания инфраструктуры Active Directory работают, а какие требуют доработки;

- какие службы, «из коробки», сконфигурированы корректно, а какие требуют дополнительного конфигурирования;
- какое пакетное окружение влияет на работоспособность базовых сценариев развёртывания и для каких дистрибутивных решений оно специфично;
- какой порядок действий подходит для заданного пакетного окружения, а какой требует уточнения.

Для настройки использовались следующий параметры конфигурации:

- статическая настройка сетевых интерфейсов в `/etc/net/ifaces/$IFACE/`;
- настройка DNS в `/etc/net/ifaces/$IFACE/resolv.conf`;
- настройка Kerberos в `/etc/krb5.conf`;
- настройка Samba в `/etc/samba/smb.conf`;
- комплексная настройка аутентификации PAM (`/etc/pam.d/system-auth`) и авторизации NSS (`/etc/nsswitch.conf`) через SSS (`/etc/sssd/sssd.conf`) с помощью встроенной утилиты `system-auth`.

Для отладки дистрибутивных решений было подготовлено три виртуальных образа или, в терминологии Vagrant, «бокса» (Box) (https://app.vagrantup.com/mastersin/):

- `mastersin/basealt-p8-server` — сервер на базе SysVinit.
- `mastersin/basealt-p8-server-systemd` — сервер на базе Systemd.
- `mastersin/basealt-p8-workstation` — клиент на базе Systemd и NetworkManager.

Минимальный сценарий отладки (Рис. 1) строился на использовании одного и того же «серверного бокса» как для сервера (контроллера домена), так и для клиента (рабочей станции в домене Samba). Сначала для образов на базе SysVinit, а затем для образов на базе Systemd. Последним в цепочку отладки был добавлен «клиентский бокс» на основе минимального дистрибутива Альт Рабочая станция.

На текущий момент с помощью рассмотренного подхода выявлена и оперативно исправлена ошибка некорректных путей к модулям ldb (ALT#33427). Стабильно воспроизводится проблема обновления DNS после подключения к домену:

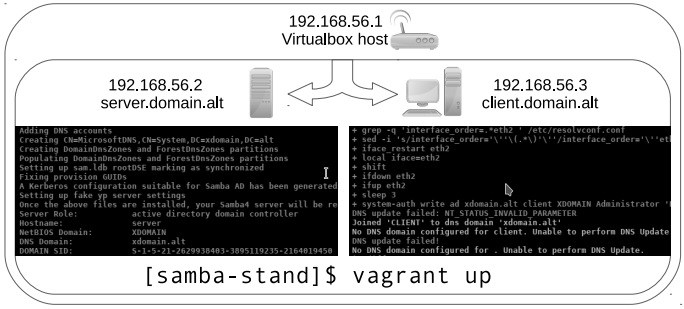


Рис. 1: Схема развёртывания минимальной конфигурации стенда

```
==> client.domain.alt: DNS update failed:
                                    NT_STATUS_INVALID_PARAMETER
==> client.domain.alt:
                  Joined 'CLIENT' to dns domain 'domain.alt'
==> client.domain.alt: No DNS domain configured for client.
                                  Unable to perform DNS Update.
==> client.domain.alt: DNS update failed!
==> client.domain.alt: No DNS domain configured for .
                                  Unable to perform DNS Update.
```

Также, процесс отладки во время написания тезисов моего доклада позволил выявить проблему падения `net ads join` при наличии двух доменов в списке поиска локальных доменов (search) файла `/etc/resolv.conf` (SAMBA#13022).

В связи с поддержкой дистрибутивов Альт и даже шире — любых дистрибутивов на базе Sisyphus в новых версиях Vagrant, предлагается более широкое использование рассмотренного сценария отладки для решения следующих задач:

- демонстрации воспроизводимых рабочих конфигураций желающим ознакомиться пользователям;
- выявления регрессий в процессе жизненного цикла дистрибутивных решений;
- последовательного внедрения во время отладки решений таких современных средств оркестрации, как Ansible, Chef и Puppet;

- непосредственного использования отладочного стенда для решения проблем разработки и отладки.

Литература

[1] *Mitchell Hashimoto, Vagrant*: Up and Running, 2013

[2] *Vagrant Project wiki, Available Vagrant Plugins*, 2017, https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins

Михаил Быков
г. Москва, diglossa.org
http://diglossa.org/chinese

Морфей для Китайского языка

Аннотация

Принципы, применявшиеся для разработки приложения Морфей для древних языков, доказали свою эффективность и при работе над языком современным, а именно Китайским.

Скринкаст: https://youtu.be/JkLvujnKg_g

Установить:
https://github.com/mbykov/morpheus-eastern/releases/latest

Морфей не ставит задачу выполнения перевода анализируемого текста. Но задачу автоматизации процесса чтения и понимания каждого читаемого слова.

Морфей:

- позволяет читать Китайский, и упрощенный, и классический варианты
- расширения для иных идеографических письменностей (Тибетский, Японский, etc) в работе
- приложение основано на CouchDB, а следовательно, является распределенным, многопользовательским, и легко масштабируемым
- в качестве локальной БД используется PouchDB, то есть для каждой платформы подключается свой, оптимизированный и рекомендованный вариант БД

- кроссплатформенный — Windows, MacOS, Linux, мобильные будут позже
- интерфейс основан на Electron.js, то есть это по сути, браузер Chromium
- работает оффлайн, синхронизируется с сервером, когда доступна сеть
- синхронизация гарантирует постоянную актуальность словарей
- работает где угодно на десктопе, а не только в браузере
- многопользовательский (пользователь может добавлять/редактировать записи)
- можно на лету подключать/заменять словари, англ, нем, русский, etc, в том числе специализированные словари
- синхронизируются только установленные словари, возможна тонкая настройка репликации (filtered replication)
- одновременно обрабатывает большой объем (несколько абзацев) текста
- выполняет рекурсивную сегментацию длинного слова (сегментирует сегменты)
- преобразует упрощеннное-традиционное написание текста по желанию читателя

Из недостатков метода нужно отменить его ресурсоемкость. Несколько подключенные словарей, большой объем разом анализируемого текста — слабую машину вешают на десятки секунд.

Из позитивного нужно отметить, что работа с локальной базой данных радикально снижает нагрузку на сервер и обеспечивает очень быстрое выполнение запроса.

Благодаря CouchDB синхронизация с сервером появляется «из коробки», и не требует вообще никаких усилий по настройке.

Рекурсивная сегментация позволяет выполнять углубленный анализ слова. А также, что немаловажно, решает интерфейсную и эргономическую проблему, хорошо заметную в аналогичных приложениях. Они либо не показывают внутреннюю структуру сегмента текста, либо приводят весть набор возможных внутренних сегментов сразу, что затрудняет восприятие, и следовательно, понимание текста.

Петер Хауер (linguasoft, Vienna) создал для Морфея словарь ECBT («Early Chinese Buddhist translations»), основанный на фундаментальной работе проф. Seishi Karashima (Soka University, Tokyo). См. http://www.dila.edu.tw

Я надеюсь на появление также иных вспомогательных словарей, например, словаря компьютерной лексики, etc

То, что Морфей не ставит задачу перевода текста, делает его нишевым продуктом. В это смысле он вне мейстрима. Но хорошо очерченная узкая задача (автоматизация понимания ино-культурного текста, а не чтение готового перевода) позволяет сделать приложение достаточно эффективным очень малыми средствами.

Евгений Сыромятников
Брно, Red Hat Czeh s. r. o.

Некоторые аспекты разработки и пакетирования out-of-tree модулей Linux

Аннотация

Доклад являет собой отражение имеющегося у автора опыта разработки, адаптации (backporting) и пакетирования различных модулей ядра Linux для различных дистрибутивов Linux. В рамках доклада планируется рассмотреть некоторые особенности разработки и адаптации out-of-tree модулей ядра Linux, а также некоторый инструментарий, облегчающий процесс пакетирования данных модулей.

Out-of-tree modules

Linux — монолитное ядро операционной системы, поддерживающее загрузку модулей — кода, который может быть загружен и исполнен (и, зачастую, выгружен) в адресном пространстве ядра в процессе его работы.

Out-of-tree модули ядра Linux — модули Linux, собранные вне основного процесса сборки ядра. Это могут быть модули ядра, разрабатываемые независимо (например, под несовместимой лицензией или имеющие существенные причины, препятствующие их включению в состав кодовой базы Linux), или же различные варианты адаптации

модуля, включённого в одну версию кодовой базы, для использования с ядром, собранным на основе другой версии кодовой базы ядра. Наиболее частым случаем второго варианта является backporting — адаптация модуля ядра из более новой версии кодовой базы к использованию в более старой версии. Другим, менее частым, случаем, является адаптация заброшенных и удалённых staging-драйверов к современным версиям Linux.

Разработка

В отличие от некоторых других проектов, Linux, начиная с версии 2.6, не пытается поддерживать стабильность внутренних API. К счастью, подавляющее большинство изменений API синтаксически несовместимо (изменение количества аргументов функций, переименование макросов, и т. п.). Это позволяет достаточно легко, без необходимости инспектировать все изменения в релевантном коде для всех целевых версий, обнаруживать эти изменения и вносить изменения, требуемые для обеспечения работоспособности модуля ядра в целевой версии.

В случае необходимости поддержки широкого диапазона версий Linux (что, например, актуально для модулей ядра, разрабатываемых независимо), ситуация становится несколько интереснее.

В заголовках присутствует макроопрделение `LINUX_VERSION_CODE`, содержащее версию ядра в некоем числовом формате, и макроопределение `KERNEL_VERSION`, позволяющее конструировать числовое представление для необходимой версии ядра. Соответственно, можно определять разные варианты реализации в зависимости от целевой версии ядра.

В случае же ядер, в которых присутствует код, портированный из более свежих версий mainline-ядра (например, ядра в таких дистрибутивах, как CentOS, SLES, или Ubuntu), приходится использовать иные методы. Можно, например, полагаться на различные системы конфигурации сборки или же прибегать к разнообразным приёмам, которые позволяют использовать одну и ту же кодовую базу для разных версий без необходимости конфигурации сборки.

Пакетирование

Правила для обеспечения установки out-of-tree modules в систему могут варьироваться в различных дистрибутивах.

Большинство дистрибутивов, особенно community-based, предполагают возможность использования пользователями самостоятельно собранных ядер и не пытаются предоставить бинарные пакеты всех доступных out-of-tree модулей ядра для всех доступных в репозитории вариантов ядер. Вместо этого (или в дополнение к ограниченному набору бинарных пакетов out-of-tree модулей) используется инфраструктура, обеспечивающая автоматизацию сборки out-of-tree модуля под используемые на данной вычислительной системе ядра. Среди вариантов инструментов, обеспечивающих подобную функциональность, можно упомянуть `dkms` и `akmods`.

dkms

`dkms` [1] — инструмент для управления out-of-tree модулями Linux, разработанный компанией Dell и представленный в 2004 году.

`dkms` отслеживает состояние всех зарегистрированных в рамках него модулей (зарегистрирован, собран, установлен, и. т. д.) и позволяет управлять ими (собирать, устанавливать, удалять).

Для описания необходимых операций по сборке out-of-tree модуля используется конфигурационный файл, являющий собой (как и весь остальной код `dkms`) скрипт на `bash`, и включающий в себя информацию о модулях, командах сборке и иную информацию (например, варианты ядра, для которых данный модуль предполагается возможным к использованию).

Помимо этого, `dkms` обладает следующими возможностями:

- Создание source или binary tarball. Source tarball может использоваться для сборки модуля на целевой системе, binary tarball может быть использован на целевой системе без необходимости сборки модуля.
- Поддержка создания source/binary пакетов в формате RPM и DEB.
- Поддержка создания Red Hat Driver Disk и SuSE Kernel Module Package.

Дистрибутивная интеграция (в тех дистрибутивах, которые её предоставляют) включает в себя триггеры, выполняющие сборку и

установку имеющихся в системе out-of-tree модулей при установке новых ядер с помощью пакетного менеджера, и удаление собранных модулей, которые больше не требуются.

akmods

`akmods` [2] — Fedora-specific инструмент автоматизации пересборки пакетов модулей, существующий с 2007 года. Представляет из себя несколько скриптов на `bash`, автоматизирующих пересборку для конкретных ядер из source RPM и последующую установку out-of-tree модулей.

С точки зрения пользователя функциональность `akmods` схожа с `dkms`, но `akmods` ориентирован на использование RPM и предполагает использование специальным образом написанных spec-файлов, в то время как `dkms` не привязан к конкретной системе пакетирования (хотя и включает в себя поддержку сборки DEB или RPM пакетов); создание инфраструктуры `akmods` обосновывается именно сложностью (для конечного пользователя) и перегруженностью `dkms` [3].

Литература

[1] Matt Domsch, Gary Lerhaupt. *Dynamic Kernel Module Support: From Theory to Practice* Proceedings of the Linux Symposium, Volume One. July 21st—24th, 2004. Ottawa, Ontario, Canada. P. 187–202. https://www.kernel.org/doc/ols/2004/ols2004v1-pages-187-202.pdf

[2] https://rpmfusion.org/Packaging/KernelModules/Akmods

[3] https://rpmfusion.org/Packaging/KernelModules/Kmods2#Why_not_simply_use_dkms

Александр Боковой
Эспоо, Финляндия, Red Hat

Проект: FreeIPA https://www.freeipa.org

Перенос FreeIPA на Python 3 или как мы танцевали Samba

Аннотация

Поддержка Python 2.7 прекращается апстримом приблизительно в 2020 году. Для систем с долгосрочной коммерческой поддержкой это решение проекта Python означает существенное увеличение рисков и вложений ресурсов. Уже несколько лет в Fedora Project идет процесс по освобождению базовой системы от зависимостей на Python 2.7. О том, во что это вылилось для проекта FreeIPA – этот доклад.

Первая стабильная версия Python 3 (3.0) вышла в 2008 году. Разработчики интерпретатора CPython активно пытаются убедить пользователей переписать свои приложения с Python 2.x (2.7) на Python 3 с тем, чтобы прекратить поддерживать две несовместимые реализации. Fedora Project занят практическим вопросом перевода всего дистрибутива на Python 3 уже несколько лет. На сайте http://fedora.portingdb.xyz/ можно посмотреть текущее состояние этой работы. На середину сентября 2017 года отслеживалось 3494 пакета. Из них 455 были выпущены с поддержкой только Python 3, 1591 мог использоваться как с Python 3, так и Python 2.7, 100 пакетов были неправильно собраны, но не были проблематичными с точки зрения кода. Наконец, 857 пакетами никто не занимался, 405 пакетов зависили от работы над другими, а 86 пакетов портировать не собирались, поэтому зависящие от них проекты вынуждены переписывать свой код полностью.

Проект FreeIPA направлен на создание инфраструктуры уровня предприятия. Он объединяет в себя средства развертывания серверов LDAP, Kerberos, Samba, DNS и других, а также развитую среду по управлению ресурсами. Эта среда, как ее серверная, так и клиентская части, написана на Python и отслеживается в рамках проекта портирования как группа "Identity Management":

http://fedora.portingdb.xyz/grp/identity/. Эта четвертая по величине группа пакетов: 224 пакета, из которых перенос 180 практически завершен.

Во многих случаях поддержка Python 3 не означает отказ от поддержки Python 2.7. Это так в случае FreeIPA, поскольку требуется поддерживать уже выпущенные версии FreeIPA в рамках Red Hat Enterprise Linux, где Python 2.7 выступает в роли основного интерпретатора, а версии FreeIPA все еще обновляются. Так, RHEL 7.4 вышел с версией FreeIPA 4.5 и скорее всего в следующей версии RHEL 7 произойдет обновление FreeIPA. Поддержка двух параллельных веток кода на Python 2 и Python 3 экономически неэффективна, поэтому основной целью ставилась задача приведения имеющегося кода в совместимое состояние с обеими версиями Python.

По счастью, эта задача интересует многих в сообществе и были написаны специальные библиотеки, которые позволяют сгладить разницу в синтаксисе и семантике между Python 2.7 и Python 3.5/3.6 для прикладных разработчиков. В рамках проекта по переносу приложений на Python 3 в Red Hat было написано руководство по «консервативной миграции», http://portingguide.readthedocs.io/en/latest/, которое и рекомендуется к прочтению.

Одним из сдерживающих факторов при переносе приложений являюся зависимости, написанные на C и интегрируемые в приложения на Python с помощью внутреннего API интерпретатора CPython. Структуры данных и вызываемые функции между интерпретаторами Python 2 и Python 3 различаются, да и рекомендуемые способы интеграции библиотек на C тоже изменились. Для большинства приложений это не так и важно, потому что переход на новый метод интеграции не критичен, но FreeIPA использует библиотеки из проекта Samba, где почти весь код модулей для Python автоматически генерируется из описаний IDL и тесно интегрирован с управлением памяти в Samba.

В Samba Team было решено обеспечить поддержку как Python 2.7, так и Python 3, поэтому сотрудниками Red Hat были написаны специальные макросы, которые позволяют компилировать один и тот же код модуля на C для обоих семейств интерпретаторов Python, py3c: http://py3c.readthedocs.io/en/latest/.

Результат использования этого набора макросов существенно упрощает поддержку модулей для двух языков. В качестве приме-

ра можно привести исправление, добавляющее поддержку Python 3 для модуля samba.messaging:

```
diff --git a/source4/lib/messaging/pymessaging.c b/source4/lib
    /messaging/pymessaging.c
index d83f5e6a637..514f6d16970 100644
--- a/source4/lib/messaging/pymessaging.c
+++ b/source4/lib/messaging/pymessaging.c
@@ -20,6 +20,7 @@
 */

 #include <Python.h>
+#include "python/py3compat.h"
 #include "includes.h"
 #include "python/modules.h"
 #include "libcli/util/pyerrors.h"
@@ -36,7 +37,6 @@
 #include <pytalloc.h>
 #include "messaging_internal.h"

-void initmessaging(void);

 extern PyTypeObject imessaging_Type;

@@ -505,19 +505,29 @@ PyTypeObject imessaging_Type = {
                    "Create a new object that can be used to
                        communicate with the peers in the
                        specified messaging path.\n"
 };

-void initmessaging(void)
+static struct PyModuleDef moduledef = {
+    PyModuleDef_HEAD_INIT,
+    .m_name = "messaging",
+    .m_doc = "Internal RPC",
+    .m_size = -1,
+    .m_methods = NULL,
+};
+
+MODULE_INIT_FUNC(messaging)
 {
    PyObject *mod;

    if (PyType_Ready(&imessaging_Type) < 0)
            return;
+           return NULL;

-   mod = Py_InitModule3("messaging", NULL, "Internal RPC");
+   mod = PyModule_Create(&moduledef);
```

```
    if (mod == NULL)
-           return;
+           return NULL;

    Py_INCREF((PyObject *)&imessaging_Type);
    PyModule_AddObject(mod, "Messaging", (PyObject *)&
        imessaging_Type);
    PyModule_AddObject(mod, "IRPC_CALL_TIMEOUT", PyInt_FromLong
        (IRPC_CALL_TIMEOUT));
    PyModule_AddObject(mod, "IRPC_CALL_TIMEOUT_INF",
        PyInt_FromLong(IRPC_CALL_TIMEOUT_INF));
+
+   return mod;
 }
```

Samba 4.7.0, вышедшая 21 сентября 2017 года, стала первой версией Samba, поддерживающей сборку модулей Python для версии Python 3.6. Это дало возможность полностью мигрировать на Python 3 и FreeIPA. Однако прежде чем, это произошло, нужно было перенести на Python 3 и ряд других проектов, написанных на Python.

Одним из наиболее сложных стал модуль поддержки работы с LDAP. Автор python- ldap принял решение полностью переписать модуль с нуля при переносе на Python 3, сделал его несовместимым с версией для Python 2, а также более соответствующим RFC, описывающим LDAP. Как результат, новая версия не только стала неудобной для миграции (требуется разный код для разных версий Python), но и просто не поддерживает работу с разнообразными реальными серверами LDAP, поскольку реальность достаточно серьезно расходится с академичностью оригинальных RFC. В результате, ряд пользователей оригинальной версии python- ldap сделали форк, который перенесли на Python 3, сохранив совместимость с предыдущей версией Python. Этот новый проект получил название pyldap.

Поскольку представление данных при передаче по сети у протокола LDAP базируется на правилах кодирования данных ASN.1, еще одним препятствием стала библиотека pyasn1. При переносе на Python 3 поведение некоторых ее компонент было изменено и привело к неработоспособности pyldap и других библиотек, используемых FreeIPA. Автор pyasn1, Илья Этингоф, активно помогал с миграцией и переписывал pyasn1 так, чтобы вернуть совместимость или улучшить код в других проектах.

Для того, чтобы успешно отслеживать изменения, ведущие к неработоспособности в той или иной версии Python, в процесс CI во

FreeIPA были добавлены целевые тесты всего кода на соответствие Python 2.7 и Python 3.6, а также работоспособность как одной, так и второй версии. Также был добавлен тест устанавливаемости клиента средствами pypi — этот клиент активно применяется в OpenStack для автоматической регистрации машин во FreeIPA. Всё это увеличивает время тестирования каждого отдельного исправления (типичный тест среды с двумя репликами и одним клиентом занимает около получаса), но позволит сэкономить время в поддержке в будущем. Учитывая, что сам процесс миграции на Python 3 занял более трех лет, это достаточно важный фактор.

FreeIPA 4.6 стал первой версией с поддержкой Python 3. Эта ветка FreeIPA станет основной в дистрибутиве Fedora 27.

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

Поддержка информационных систем с квалифицированной электронной подписью в ОС Альт

Аннотация

Традиционно свободные операционные системы отличаются высоким уровнем поддержки современных криптографических алгоритмов и средств их применения. К сожалению, это не в полной мере относится к российским криптографическим алгоритмам, продвижение которых в качестве международных стандартов по ряду причин замедляется по сравнению с зарубежными разработками.

В результате складывается парадоксальная ситуация — поддержка российской криптографии в отечественных операционных системах иногда отстаёт от поддержки зарубежных аналогов. Но тем интереснее наблюдать за прогрессом: с выходом каждой новой версии ОС, включённой в Реестр отечественных программ, картина существенно меняется в лучшую сторону.

Основная задача, стоящая сейчас перед разработчиками ОС Альт — предоставить «обычному пользователю» — не важно, физическому или юридическому лицу, действующему на общих основаниях, возможность, работая в ОС из Реестра отечественного ПО, полноцен-

но использовать информационные системы, в которых задействована квалифицированная электронная подпись.

Например, для электронных торговых площадок и порталов государственных услуг под полноценным использованием подразумевается в первую очередь возможность аутентификации на соответствующем сайте по сертификату, размещённому на отдельном физическом носителе, и возможность электронной подписи документов, сформированных в интерфейсе сайта.

На эту тему уже проведён ряд исследовательских работ, результатом которых стало заключение о том, что, в принципе, всё работает. Но здесь важно понимать, что большинство исследований совместимости с порталами государственных услуг РФ современных криптосредств, работающих под ОС Linux, проводилось в лабораторных условиях. Например, при наличии специальных договорённостей между исследователем и удостоверяющим центром, выдающим сертификат. К сожалению, по результатам таких работ о реальных возможностях лиц, действующих на общих основаниях, судить сложно.

Что же выяснилось в результате исследования, на что можно рассчитывать сейчас и какие возможности появятся в ближайшей перспективе?

В первую очередь интересен не сам факт поддержки криптографических алгоритмов семейства ГОСТ, а возможность применения их для выполнения юридически значимых действий. Таких, например, как подпись официальных документов.

Так как аккредитованные государством удостоверяющие центры сейчас выдают квалифицированные сертификаты электронной подписи в основном на физических носителях, важно обеспечивать поддержку таких устройств на уровне операционной системы. Для всех современных носителей ключей с аппаратной реализацией поддержки алгоритмов ГОСТ, пригодных для квалифицированной электронной подписи, такая поддержка в ОС Альт реализована.

Следующий важный момент — поддержка взаимодействия с вебпорталами федерального и регионального уровня, оказывающими государственные услуги физическим и юридическим лицам. Здесь ситуация сложнее, так как сказывается многолетняя ориентация отечественных разработчиков сертифицированных криптосредств в первую очередь на ОС Windows. Например, из пяти федеральных электронных торговых площадок сейчас только три поддерживают

работу с операционными системами из Реестра отечественных программ.

Другой пример — из-за недостаточно чётко проработанных регламентов обеспечения совместимости один и тот же аппаратный носитель ключей не получится использовать одновременно на сайте Госуслуг и на электронных торговых площадках. Придётся применить два аппаратных носителя с двумя парами ключей и двумя сертификатами соответственно.

Но при всём этом ситуация с поддержкой веб-порталов постепенно меняется в лучшую сторону. Так например, за время подготовки этого обзора операторы одной из пяти федеральных торговых площадок успели обеспечить поддержку ОС из Реестра.

Ещё одно важное применение электронной подписи — подпись файлов, офисных документов и сообщений. Основная проблема здесь — графический интерфейс пользователя, который пока реализован не для всех типичных сценариев работы. Именно на этом направлении в том числе сейчас сосредоточены усилия разработчиков ОС Альт.

И последняя по списку, но от этого не менее важная область применения электронной подписи — аутентификация пользователей. Здесь ОС Альт на данный момент среди операционных систем из Реестра лидирует с большим отрывом. Кроме собственно возможности аутентификации пользователя по сертификатам электронной подписи на аппаратных носителях ключей, реализована поддержка алгоритмов ГОСТ, блокировки и разблокировки экрана по событиям подключения и отключения носителя, автоматизированы механизмы настройки типовых конфигураций. А в качестве демонстрации возможности встраивания в существующую инфраструктуру открытых ключей реализована поддержка аутентификации пользователей по квалифицированным сертификатам.

Уже при установке ОС можно включить режим, при котором для аутентификации в системе достаточно иметь любой действующий квалифицированный сертификат, выданный аккредитованным удостоверяющим центром. Таким образом удалось задействовать самую крупную в стране публичную инфраструктуру поддержки электронной подписи.

Подводя итоги можно сказать, что поддержка информационных систем с квалифицированной электронной подписью в ОС Альт существует уже далеко не в начальной стадии. Реализована основная функциональность, дорабатываются пользовательские инструменты

и главное — ведётся непрерывное взаимодействие с технологическими партнёрами по обеспечению взаимной совместимости.

Павел Волнейкин
Санкт-Петербург, Базальт СПО
Проект: Sisyphus http://sisyphus.ru/

Аутентификация по аппаратным токенам в дистрибутивах Альт

Аннотация

Кодовая база линейки дистрибутивов Альт с недавних пор содержит улучшенную поддержку аутентификации по аппаратным токенам. В ходе работ были устранены ошибки, расширена совместимость с популярными моделями российских токенов, значительно упрощена и отработана процедура комплексной настройки программ, имеющих отношение к этой задаче.

Состав и настройка программного обеспечения

Набор программ, имеющих отношение к аутентификации по аппаратным токенам, довольно большой. В него мы включаем:

- собственно модуль аутентификации — `pam_pkcs11.so`;
- криптографическую библиотеку `libcrypto.so` (OpenSSL);
- дополнения к криптографической библиотеке, реализующие разные способы шифрования (engines);
- модули доступа к токену (различные реализации PKCS#11);
- модули и библиотеки, реализующие различные способы установки соответствия между записанными на токене данными и учётными записями пользователей;
- служба, обеспечивающая раздельный, низкоуровневый доступ к токену (обычно по по USB);
- служба, следящая за такими событиями, как подключение и удаление токена;
- программы, выполняющиеся как реакция на указанные события;

- программа, управляющая пользовательскими сеансами;
- программа-гритер, предоставляющая диалог входа в систему.

Нетрудно догадаться, что взаимосвязанная настройка всего этого комплекса программ — задача совсем не простая. Поэтому в первую очередь мы подумали о том, как упростить её решение. При этом ориентировались сразу на два сценария настройки: настройка, выполняемая администратором системы и предварительная настройка, выполняемая на этапе подготовки образа дистрибутива и его установки. Последнее связано с тем, что Альт является не только системой для конечного пользователя, но и *платформой* для изготовления дистрибутивов.

Упростить процедуру настройки значит, прежде всего, уменьшить количество параметров, с которыми имеет дело пользователь, объединить или сгруппировать их. Для этого нам пришлось выделить достаточно крупные, относительно независимые области действия во всей исходной массе параметров. При этом, на чисто логическую классификацию, которая, как известно, может быть очень и очень свободной, свой отпечаток накладывала конкретная архитектура тех программ, которые нужно было настроить.

В итоге, у нас получились следующие группы параметров:

- взаимодействие с определённой моделью токенов;
- отображение сертификатов на имена пользователей;
- доверие и проверка сертификатов;
- прочие параметры PKCS#11, включая политику взаимодействия с токеном (ожидание подключения, запрос PIN-кода и т. п.);
- политика входа в систему (без использования токена / с обязательным использованием токена и т. д.);
- отслеживание состояния токена и реакция на события;
- настройка автоматической регистрации учётных записей.

Почти для всех этих групп определённый набор конечных параметров задаётся теперь отдельным файлом, а общая конфигурация определяется выбором того или иного файла для каждой из них (через `control`). Это позволяет легко добавлять новые варианты конфигурации и поставлять их в пакетах. Исключение пока составляет настройка набора доверенных сертификатов и шифров.

Обстоятельством, препятствующим избирательной настройке набора доверенных сертификатов, прежде всего, является их количество. Здесь я имею в виду сертификаты удостоверяющих центров ГОСТ, поскольку именно на использование шифрования по ГОСТ нацелен сегодня рынок токенов в России. Большое количество сертификатов и отсутствие общепринятого разделения этого множества на части не позволяет сейчас реализовать удобную процедуру управления ими в системе.

Как можно видеть, даже после группировки количество параметров остаётся довольно значительным, и следовательно, в таком виде процедура настройки аутентификации по аппаратным токенам представляет ещё известную трудность для пользователя. Чтобы ещё больше упростить её, был написан модуль Альтератора `alterator-auth-token`. Главные из перечисленных выше параметров представлены в нём в наглядном виде, а другая часть параметров скрыта и поставлена во взаимно-однозначную связь с видимыми. Модуль представляет интерес не только для конечных пользователей и администраторов систем, но и для разработчиков дистрибутивов, поскольку он полностью готов для встраивания в инсталлятор и поддерживает передачу параметров для *преднастройки*.

Предварительная конфигурация во время установки ОС

Что такое преднастройка в Альтераторе и для чего она может понадобится?

Преднастройка основана на механизме передачи параметров через URI модуля, который указывается в `.desktop`-файле. Это открывает возможность иметь несколько `.desktop`-файлов для разных вариантов использования одного и того же модуля. Преднастройка нужна тогда, когда мы хотим привести модуль в определённое состояние при переходе к нему. Даше могут быть два сценария: изменённое состояние отображается в интерфейсе пользователя, а действительное состояние системы не изменяется, либо же изменения вносятся в системную конфигурацию сразу и отображаются затем обычным образом. Какому из сценариев следует отдать предпочтение зависит от назначения модуля.

С необходимостью реализовать преднастройку мы столкнулись тогда, когда подключили `alterator-auth-token` к инсталлятору. Тогда мы обратили внимание, что молуль инсталлятора отличается от обыч-

ного молуля тем, что его диалоговое окно играет роль *предложения* по настройке системы, а не обычную роль индикаторной панели, рассказывающей о текущем состоянии системы. Это наверняка справедливо для любого модуля инсталлятора, но для нашего усугублялось тем, что в нём для режима инсталлятора нельзя было ограничиться одним единственным набором значений по умолчанию. Было ясно, например, что, в зависимости от целевого назначения дистрибутива и аудитории, будет уместно предложить пользователю включить схему аутентификации для определённой модели или семейства токенов.

Видя, что сама архитектура инсталлятора предполагает использование отдельных `.desktop`-файлов для каждого шага, мы решили, что именно через эти файлы вполне уместно передавать параметры конфигурации по умолчанию. Реализация передачи параметров через URI была продиктована широко распространённым стандартом.

О типовых задачах и вариантах настройки аутентификации

Теперь перейдём к вопросу о том, какие варианты настройки доступны в нашем базовом дистрибутиве, каким образом, следовательно, можно настроить аутентификацию с помощью предлагаемых средств и почему это так.

Первое, что мы постарались обеспечить, это возможность включить такой режим, при котором наличие токена было бы обязательным для нормального использования системы. По-сути, это — возможность шире задействовать потенциал первого из двух факторов аутентификации. Исключение было сделано для пользователей с повышенными правами (администраторов) — для них мы предусмотрительно оставили возможность войти в систему без использования токена, по одному только паролю. Совершение входа по токену, тем не менее, делает обязательным его наличие во время сеанса работы и для этой категории пользователей. Извлечение токена сопровождается временной блокировкой сеанса. Для возобновления работы необходимо подключить токен и ввести верный PIN-код.

Блокировка и разблокировка сеанса по токену хорошо себя показывает тогда, когда за данным рабочим местом работает всегда один и тот же пользователь. В случае же, когда пользователей несколько, дополнительные действия по переключению сеанса, вводу или выбору имени пользователя оказываются ничем не оправданными и претендуют на то, чтобы их автоматизировать. Действительно, аутентифи-

кация основывается на том, что на токене записано имя пользователя либо информация, которая может быть поставлена в однозначное соответствие с ним.

Несколько слов о том, какие способы идентификации пользователей доступны. Прежде всего это сличение по сертификату. Оно основано на поиске записанного на токене сертификата в содержимом файлов `.ssh/authorized_keys` в домашних директориях пользователей. И хотя ясно, что прямой поиск работает неоптимально, но на мой взгляд, этот вариант вполне оправдан для систем с небольшим количеством пользователей, поскольку очень прост в настройке. А иногда даже не требует дополнительной настройки, поскольку файл `authorized_keys` часто уже присутствует и содержит нужные данные. Аналогичная процедура поиска выполняется также для файлов в директории `.eid/` и подходит для случая, когда удалённый доступ по сертификату к системе не предоставляется.

Кроме сличения сертификатов существует много, я бы сказал, прямых, способов идентификации пользователей. Общей чертой их является то, что имя пользователя берётся непосредственно из того или другого поля сертификата. В самом простом случае это поле `CN` (common name). Среди других вариантов можно указать поле `subject` и адрес электронной почты. Однако я должен сказать, что кажущаяся простота использования этих способов обманчива: на самом деле использование этих вариантов связано с большими рисками и должно сопровождаться повышенным вниманием к фактической достоверности записанной на токене информации, а значит, очень высокими требованиями к надёжности её источника. Это означает, что записанное, например, в поле `CN` имя пользователя должно фактически соответствовать одному, конкретному человеку (или группе людей). В противном случае мы рискуем предоставить постороннему человеку доступ с правами иного лица.

Кроме перечисленных вариантов установления соответствия, поддерживаемых в исходной версии пакета, мы разработали ещё один, специальный, применяемый совместно с механизмом автоматической регистрации учётных записей. О нём будет рассказано далее.

Вернёмся теперь к вопросу об автоматическом переключении сеанса. Для этого, во-первых, требуется повторять процедуру определения имени пользователя каждый раз, когда токен подключается к компьютеру. Во-вторых, после того, как имя пользователя было определено, следует решить какое действие необходимо выполнить:

возобновить приостановленный сеанс, либо предложить пользователю пройти аутентификацию и начать новый сеанс.

Предложение начать новый сеанс должно, конечно, сопровождаться автоматической установкой имени пользователя. Реализация этого действия потребовала определённой доработки программы-гритера — диалогового окна входа в систему. Благодаря этой доработке, процедура аутентификации по токену теперь сводится только ко вводу PIN-кода.

Предполагаю, что по политике блокировки сеанса могут быть вопросы. Можно, например, предположить случай, когда предпочтительнее было бы не блокировать, а завершать сеанс. Однако мы посчитали, что проектировать подобное поведение, не отталкиваясь при этом от конкретной задачи, вряд ли оправдано — можно легко промахнуться с результатом. Так, например, безусловное завершение сеанса при отключении токена может стать неожиданным и неприятным сюрпризом для пользователя. Вместо этого более безопасным решением кажется блокировка сеанса с последующим его автоматическим завершением по истечении таймаута. Реализация завершения сеанса по таймауту, к тому же, непосредственно не связана с аутентификацией по токенам.

Наименее проработанной частью системы настройки, на сегодняшний день, является настройка криптографической части в отношении проверки сертификатов и доверия удостоверяющим центрам. Частично об этом уже было сказано ранее. В дополнение к сказанному, нужно добавить несколько слов о настройке шифров, а также рассмотреть перспективы по улучшению ситуации.

Под настройкой шифров я имею в виду включение и выключение поддержки алгоритмов ГОСТ. Пока это единственный, относящийся к шифрам параметр, который у нас реализован. Главный же недостаток реализации заключается в том, что включение и выключение производится через общесистемную конфигурацию, хотя конфигурационный модуль имеет вполне ясное назначение, связанное с аутентификацией. Причина такого недостатка состоит в том, что модуль аутентификации использует библиотеку `crypto` OpenSSL с настройками по умолчанию. Я думаю, что мы должны исправить этот момент[1].

Второй недостаток касается работы с набором доверенных сертификатов. Здесь ситуация следующая: автоматическое добавление сер-

[1] Исправлено в `pam_pkcs 0.6.9-alt16`

тификатов в число доверенных реализовано только для набора корневых сертификатов ГОСТ, и при этом, оно выполняется *однократно* — при включении поддержки шифрования по ГОСТ. Механизма обновления списка доверенных сертификатов пока не предусмотрено. Также, для добавления прочих сертификатов, не входящих в набор корневых сертификатов ГОСТ, никаких инструментов не предусмотрено (вручную это сделать, конечно, можно). Предложения по улучшению ситуации приветствуются.

Автоматическая регистрация учётных записей

Перехожу к последнему вопросу, касающемуся такого новшества в нашей системе, как автоматическая регистрация учётных записей.

Как указывалось ранее, процедура аутентификации основывается на том, что на токене записана информация, которая может быть поставлена в однозначное соответствие с именем пользователя. Алгоритмов для установления соответствия можно придумать много. И в части их, соответствие будет иметь *потенциальный* характер: я имею в виду случаи, когда имя пользователя каким-либо образом *вычисляется*, а в системе такого имени может и не быть ещё зарегистрировано. В этом случае, если мы в достаточной степени доверяем записанной на токене информации, мы можем это вычисленное, принесённое на токене имя зарегистрировать. Можно сказать, что здесь имеет место перенос данных из какой-то более общей базы данных учётных записей в локальную базу данных, причём запись переносится на отдельном физическом носителе, а её достоверность обеспечивается средствами криптографии.

Регистрация учётной записи производится прозрачно для пользователя, в ходе самой аутентификации. Говоря более строго: после аутентификации, но до запуска сеанса работы. Запись регистрируется один раз, в том случае, когда она отсутствует, а затем доступ к ней предоставляется обычным образом.

Для надёжности описанной процедуры есть одно условие: уникальность записи. Для этого, во-первых, доверие к источнику информации должно касаться не только формальной подлинности записанных на токене данных, но и фактической их подлинности, т. е. соответствия одному, конкретному человеку (или группе людей). Об этом я уже упоминал, когда рассказывал о прямых способах идентификации. И тот факт, что вычисление имени пользователя может быть

весьма «непрямым», не отменяет повышенных требований к уровню доверия. Оно, повторяю, должно быть очень высоким. Второе, что мы должны обеспечить — это уникальное соответствие результата вычисления исходным, идентифицирующим пользователя, данным. Теперь уже для того, чтобы всегда предоставлять одному и тому же человеку доступ именно к его данным в локальной системе, а не к пустой, вновь созданной записи. Иными словами, мы не должны ошибиться в бóльшую сторону, считая, что каждый новый сертификат — это всегда новый пользователь. Пользователь, обычно, имеет право обновить свой сертификат и, в ряде случаев, обязан периодически делать это.

Требование уникальности имени пользователя, однако, может входить и входит в противоречие с конфиденциальностью личных данных пользователя. Именно с таким вариантом мы столкнулись, когда выбрали в качестве основы имени пользователя поле СНИЛС сертификата (имеется в виду сертификат ГОСТ). Ясно, что даже если этот конкретный СНИЛС ничего не говорит лично мне, то я всё равно имею возможность использовать его не по назначению, просто потому, что знаю, что это — действительный номер. Это нехорошо. И поэтому мы приняли решение прятать действительный номер за хеш-суммой. Конечно эта мера в известной степени нарушает уникальность имени пользователя, и значит, мы рискуем открыть доступ постороннему. Но на сегодняшний день, ничего лучше для апробации механизма автоматической регистрации учётных записей мы не придумали.

По меткому выражению одного из моих коллег, механизм автоматической регистрации учётных записей ликвидирует имя пользователя — пользователю больше не нужно его придумывать или запоминать. Конечно, это подходит не для всех пользователей и систем. Но техническая возможность это сделать теперь есть.

Заключение

Работа по внедрению аутентификации по аппаратным токенам в дистрибутивы Альт далеко ещё не закончена. Но уже теперь она вызывает живой интерес как со стороны производителей токенов, так и со стороны некоторых наших заказчиков. При этом, однако, данное направление в разработке никак нельзя назвать исключительно заказным: здесь мы пытаемся работать на опережение, заранее реализуя заложенные в технологию аппаратных токенов возможности.

Относительно перспективности данного направления хочу сказать следующее: в очень сильной степени оно зависит от пути, по которому будет развиваться аппаратная часть. Чем удобнее и эффективнее можно будет пользоваться токенами, тем шире, очевидно, они будут распространяться. Но при этом нельзя забывать о том, что удобство и, особенно, эффективность использования, в свою очередь, зависят от уровня поддержки аппаратных решений программным обеспечением, и в частности, от степени *интеграции* таких решений в операционную систему. Последнее имеет к нам, как к разработчикам операционных систем, самое прямое отношение. Значит, работать нам нужно сообща.

Дмитрий Белявский
Москва, Технический Центр Интернет

Проект: OpenSSL, OpenSSL GOST engine http://www.openssl.org/, https://github.com/gost-engine/

ГОСТ в OpenSSL: 12 лет международного взаимодействия

Аннотация

Начиная с 2005 года мои коллеги и я занимаемся внедрением поддержки российской специфики в OpenSSL.

Российская специфика не сводится к криптографии, хотя начиналось всё именно с неё. Не меньшее значение имеет и корректный вывод русских букв, и поддержка расширений X.509-сертификатов, и многое другое.

На первом этапе работа сводилась прежде всего к выделению структур, предназначенных реализовать более-менее любой алгоритм электронной подписи в OpenSSL. Этот этап был закончен к выходу версии 1.0, когда эти структуры были специфицированы, и в состав проекта OpenSSL вошла реализация алгоритмов шифрования, хеширования и электронной подписи по ГОСТ. Тогда поддержка была добавлена за счёт спонсорства моего работодателя на тот момент, фирмы «Криптоком».

На тот момент в OpenSSL не было отлаженных процедур работы с запросами извне, и если запрос не был интересен кому-нибудь из

core team или не был спонсирован, с большими шансами обращение, не приводящее к уязвимости, игнорировалось.

Внутренние процедуры в OpenSSL существенно изменились после Heartbleed, когда стало понятно, что такой механизм развития продукта чреват большими неприятностями. Тогда были получены средства, выстроены процедуры и составлен план развития продукта.

К этому моменту было принято новое семейство стандартов ГОСТ (хеш и электронная подпись), поддержка которых существовала в виде патчей у того же «Криптокома». Было желание включить это в апстрим, но не было понимания, как именно.

Следующий этап наступил в результате моего знакомства с Ричем Сальзом (Rich Salz) на одной из конференций IETF. Там мне удалось договориться о включении поддержки ГОСТ в ядро системы в части протоколов и о вынесении ГОСТ engine в отдельный проект. Это было сделано в 2015 году, и начиная с версии 1.1.0 код для поддержки ГОСТ в TLS, PKCS12, PKCS8 и кое-где по мелочи входит в апстрим. Реализация алгоритмов живёт отдельно.

Потом присылались отдельные патчи, исправлявшие вывод русских букв, утечки памяти и прочие мелочи.

Общий вывод: успех в добавлении кода в апстрим зависит от наличия у апстрима процедур и личных знакомств с представителями основной команды, настроенных на взаимодействие.

Борис Макаренко
Москва, ФССП России

Проект: token-manager, gost-crypto-gui
https://github.com/bmakarenko/token-manager

Криптографические операции в окружении рабочего стола ОС «Гослинукс»

Аннотация

В ФССП России активно используется свободное ПО, в том числе разработанный для нужд Службы дистрибутив ОС «Гослинукс». Используя проприетарные средства криптозащиты информации, ФССП России столкнулось с проблемой отсутствия в них удобного для пользователя интерфейса. Для решения данной проблемы было разработано два приложения: token-manager и gost-crypto-gui.

token-manager служит для работы с сертификатами и ключевыми носителями. gost-crypto-gui позволяет выполнять криптографические операции над файлами.

Федеральной службой судебных приставов (далее — ФССП России) разработана и внедрена операционная система типового дистрибутива автоматизированной информационной системы ФССП России (далее — ОС «Гослинукс»). В настоящее время данная ОС используется на более 30 тыс. рабочих станций и серверов Службы, что составляет 74% от общего числа.

В ОС «Гослинукс» в качестве штатного средства криптозащиты информации (далее — СКЗИ) используется КриптоПро CSP, которое содержится в перечне средств защиты информации, сертифицированных ФСБ России, соответствует требованиям ГОСТ 28147-89, ГОСТ Р 34.11-94, ГОСТ Р 34.10-2001, требованиям ФСБ России к шифровальным (криптографическим) средствам класса КС1 и требованиям ФСБ России к средствам электронной подписи установленным для класса КС1, и может использоваться для криптографической защиты информации, не содержащей сведений, составляющих государственную тайну, а также для формирования электронной подписи в соответствии с Федеральным законом «Об электронной подписи» от 06.04.2011 N 63-ФЗ. Однако, версия данного СКЗИ для операционных систем GNU/Linux не имеет графического интерфейса пользователя. В поставку включены только библиотеки и утилиты с интерфейсом командной строки. Все операции предполагается производить через них. В том числе, отсутствует графический интерфейс, позволяющий пользователю самостоятельно производить криптографические операции над файлами, такие как: подпись, проверка подписи, отсоединение подписи, шифрование и расшифровка.

Приложение token-manager

Процессе внедрения ОС «Гослинукс» специалисты по информатизации ФССП России столкнулись с проблемой отсутствия в ней удобного пользовательского интерфейса, позволяющего работать с носителями ключевой информации, такими как ruToken, и сертификатами электронной подписи. Использование утилит, входящих в состав СКЗИ представлялось трудоёмкой , что существенно замедляло внедрение ОС «Гослинукс» в территориальных органах ФССП России.

Для решения данной проблемы было разработано приложение token-manager, представляющее собой графическую обёртку над консольными утилитами, входящие в состав СКЗИ КриптоПро CSP и пакет opensc. Данное приложение позволяет устанавливать, просматривать и удалять сертификаты в корневом и личном хранилищах, просматривать и обновлять списки отозванных сертификатов, работать с контейнерами ключевой информации, устанавливать и просматривать лицензионный ключ для СКЗИ КриптоПро CSP.

Приложение gost-crypto-gui

Несмотря на то, что средства электронной подписи были уже встроены в различные подсистемы автоматизированной информационной системы ФССП России, в ОС «Гослинукс» все ещё отсутствовала возможность применения электронной подписи и шифрования файлов непосредственно в окружении рабочего стола пользователя.

С целью обеспечения пользователей ОС «Гослинукс» такой возможностью было разработано приложение gost-crypto-gui. Данное приложение позволяет пользователям выполнять криптографические операции над файлами, в том числе прямо из файлового менеджера Nautilus. Поддерживаются пять операций: подпись, проверка подписи, отсоединение подписи, шифрование и расшифровка. Получающиеся в результате файлы соответствуют PKCS#7 и совместимы с аналогичными программами, в том числе с наиболее популярной на данный момент КриптоАРМ для MS Windows.

В версиях ОС «Гослинукс» с RTM до ИК-4 используется окружение рабочего стола GNOME2, поэтому интеграция приложения пока доступна только для данного окружения. Однако, в планируемом релизе ОС «Гослинукс» ИК-5 будут использоваться окружения MATE и Cinnamon, и будущем в gost-crypto-gui также будет реализована поддержка файловых менеджеров данных окружений.

Заключение

Оба приложения распространяются по лицензии свободного программного обеспечения MIT, не ограничивающей пользователей в их использовании, а разработчиков — в возможности встраивания в свои продукты. Разработка приложений ведется открыто. Публичные git-репозитории располагаются на GitHub по адресам

https://github.com/bmakarenko/gost-crypto-gui и
https://github.com/bmakarenko/gost-crypto-gui,
https://github.com/bmakarenko/token-manager

В настоящее время установочные пакеты token-manager и gost-crypto-gui размещены в репозитории ОС «Гослинукс» и репозитории Sisyphus компании «Базальт СПО».

Матвеев Дмитрий Андреевич
Москва, «Электронные Офисные Системы»

Система криптографического обеспечения «КАРМА» — Универсальная система для криптографической защиты информации и применения электронной подписи (ЭП) для любых приложений

Отсутствие стандартизованного интерфейса приложений для работы с криптопровайдерами существенно затрудняло применение электронной подписи в различных приложениях, например, применение ЭП пользователями электронных торговых площадок. Кроме того, далеко не все распространенные средства СКЗИ обладают готовым интерфейсом для работы в средах ОС Linux.

Система криптографической защиты «КАРМА» –инструментальное средство, обеспечивающие возможность встраивания и использования средств криптографической защиты информации практически в любую прикладную систему. Система «КАРМА» поддерживает работу со всеми распространенными средствами СКЗИ.

Применение система «КАРМА», как промежуточного слоя между прикладным ПО и средствами криптозащиты позволяет разработчикам решений создавать универсальные продукты, не зависящие от используемой среды и наличия/отсутствия веб-браузеров.

В результате, разработчики прикладного ПО получают возможность применения стандартизированного высокоуровневого интерфейса для работы с ЭП, что существенно упрощает процесс встраивания средств электронной подписи и шифрования.

Селедкин Андрей Евгеньевич
г. Йошкар-Ола, ООО «Цифровые технологии»
https://trustedplus.github.io/

Разработка безопасных приложений на платформе Trusted.API

Аннотация

Доклад о регулирующих технологиях, об открытой платформе для создания безопасных приложений Trusted.API и о примерах созданных приложений на базе Trusted.API.

Регтех или регулирующие технологии

В финансовых сервисах главное — юридическая безопасность. Добиваться ее можно разными способами, но все большее признание среди банков получает относительно молодая отрасль технологий — регулирующие технологии или регтех.

Регтех уже называют одним из самых быстрорастущих секторов интернет-экономики. В США оборот индустрии регтеха по данным Reuters[1] составит $120 млрд уже к 2020 году.

По словам председателя Банка России Эльвиры Набиуллиной в России в ближайшие пять лет регулирование в сфере финансовых информационных технологий станет приоритетной задачей для Центробанка России.

Законодательные требования в банковской сфере настолько сложны и многослойны, что кажется, будто за ними вообще невозможно уследить. Учитывая как быстро может меняться правовой климат, порой бывает сложно за всем уследить, даже имея большой штат юристов.

Поэтому технологии, призванные автоматизировать процессы приведения банковских инструментов, будь то клиентское приложение или онлайн сервис, в соответствие с законодательством, позволяют финансовым компаниям направить ценные ресурсы на развитие бизнеса и обслуживание клиентов.

[1] http://www.reuters.com/article/us-asia-fintech-regulations-idUSKBN1360UQ

Мы в компании «Цифровые технологии» на протяжении 15 лет занимаемся по сути тем же самым регтехом. Со своей, конечно, специализацией. Это защита данных, безопасная передача данных, электронная подпись, многофакторная аутентификация, все в соответствии с требованиями отечественных регуляторов.

Trusted API — открытая платформа для создания безопасных приложений

В 2014–2015 годах наше государство на волне принятых нашими западными коллегами санкций выбрало курс на активное импортозамещение во всех ключевых отраслях экономики. Применительно к ИТ сфере был создан Единый реестр российских программ для ЭВМ и баз данных, постановлением правительства №96 от 01.02.2015 г. было запрещены государственные закупки ПО, не включенного в данный реестр.

Тогда же нам стало понятно, что в ближайшие 5-10 лет, отечественным разработчикам понадобится создать большое количество ПО по самым разным направлениям. И эти решения должны будут соответствовать требованиям действующего законодательства, требованиям регуляторов ФСБ , ФСТЭК, ЦБ РФ по безопасности.

Поэтому, мы в свое время начали создавать собственную платформу для разработчиков софта, на которой они смогут легко и с минимальными затратами создать собственные безопасные веб-приложения, в котором уже будут внедрены все необходимые технологии защиты информации с соблюдением российского законодательства

Мы назвали платформу Trusted.API, она представляет из себя несколько основных компонент, большинство из которых полностью открыты.

Это актуальные сборки Node.js и Electron, позволяющие создать как веб-сервер, так и абсолютно любые приложения. Это известная криптографическая библиотека OpenSSL, модифицированная нами для поддержки российских криптографических алгоритмов, криптопровайдеров и требований к ключам и сертификатам электронной подписи.

В Trusted.API присутствуют отдельные модули, такие как CMS, PKI, XML Crypto, PKCS#11, реализованные в соответствии с российскими стандартами.

Решение открытое, оно сейчас уже доступно на github, и его можно использовать свободно без какой либо платы. Хотя конечно мы не отказываемся от монетизации, и у Trusted.API существуют и коммерческие версии.

Trusted eSign — пример приложения созданного на платформе Trusted.API

Работая над платформой Trusted.API мы не могли, как говорится, сами не попробовать. И мы начали делать Trusted eSign.

Trusted это кроссплатформенное приложение для создания и проверки электронной подписи и шифрования файлов. Она поддерживает работу с отечественными криптопровайдерами, например СКЗИ КриптоПро CSP 4.0, поддерживает все стандарты подписи и шифрования, поддерживает работу с ключами усиленной квалифицированной электронной подписи, выдаваемыми аккредитованными удостоверяющими центрами.

Плюс у приложения простой и красивый пользовательский интерфейс, а за счет возможностей, которые дает применение Electron, нам не сложно менять его в любую сторону по пожеланиям наших пользователей.

Сейчас мы активно ведем работу по сотрудничеству с компаниями, выпускающими российские операционные системы, проводим совместное тестирование, договариваемся о включении программы в список приложений по умолчанию. Параллельно работаем над сертификацией решения, но это долгий процесс. Отмечу только, что Trusted eSign и Trusted.API были включены в Единый реестр российских программ для ЭВМ и баз данных.

Недавно мы также создали прототип приложения для ГИС ЖКХ, позволяющее отправлять запросы в ГИС ЖКХ по требованиям данной системы. Дело в том, что в ней используются запросы в формате xml, они должны соответствовать строго определенному виду каноникализации, подписываться УКЭП, и передаваться по зашифрованному каналу с использованием ГОСТ 28147-89. Все это у нас опять же создавалось на платформе Trusted.API.

Леонид Юрьев
Москва, Positive Technologies

Проект: «Fast Positive Hash»
https://github.com/PositiveTechnologies/t1ha

t1ha — самая быстрая, переносимая, 64-битная хэш-функция

Аннотация

В докладе представляется некриптографическая хеш-функция t1ha — переносимая и чрезвычайно быстрая на современных процессорах. Кроме базового переносимого варианта также предлагается несколько платформо-зависимых вариантов, способных выполнять хеширование на скоростях близких к пропускной способности памяти.

При этом базовый вариант t1ha имеет регулярную структуру, в статистических тестах качества не уступает никому из своей весовой категории, а по субъективным оценкам лучше конкурентов. Все функции проходят все тесты SMHasher без каких-либо замечаний.

Опустим определение хэш-функций вместе с детальным перечислением свойств и требований для их криптографического применения, предполагая что читатель либо владеет необходимым минимумов знаний, либо восполнит их из открытых источников, включая Википедию. Также условимся, что здесь и далее мы подразумеваем некриптографические (криптографически не стойкие) хэш-функции, если явно не указывается иное.

Хэширование применяется в массе алгоритмов, при этом практически всегда требуется максимально эффективная (быстрая) обработка данных, одновременно с соблюдением некоторого минимального уровня качества хеширования. Причём под «качеством», прежде всего, понимается «условная случайность» (стохастичность) результата относительно исходных данных. Несколько реже предъявляются дополнительные требования: устойчивость к преднамеренной генерации коллизий или необратимость.

Для стройности изложения необходимо определить понятия «качества» хэш-функции и остальные требования чуть более детально:

- Базовое качество: Изменение одного или более произвольных бит в произвольном наборе исходных данных, приводит к изменению каждого бита результата с вероятностью $\frac{1}{2}$.
- Необратимость (стойкость к восстановлению прообраза): Невозможность получения исходных данных или отдельных битов по результату хеширования.
- Устойчивость к подбору хэша (стойкость к коллизиям первого рода): Сложность поиска/подбора исходного набора данных с целью получения заданного результата или даже отдельных его битов.
- Устойчивость к подбору сообщений (стойкость к коллизиям второго рода): Сложность поиска/подбора двух разных наборов данных, которые давали бы одинаковый результат или совпадение отдельных битов.

Опуская цитирование доказательств и прочие выкладки можно констатировать:

- Надлежащее выполнение всех пунктов, одновременно с обеспечением производительности, является достаточно трудной задачей, решение которой даёт хорошую криптографическую хэш-функцию.
- Обеспечение базового качества требует достаточно большого количества операций АЛУ. Проще говоря, качество всегда конфликтует со скоростью.
- Получение качественного результата с разрядностью больше разрядности операций АЛУ требует более чем кратного увеличения количества перемешиваний, а следовательно базовых операций АЛУ.
- В целом, *создание быстрой кэш-функции предполагает достижения взвешенного компромисса между скоростью, качеством и разрядностью результата.*

Исходя из вышесказанного, можно сказать, что t1ha появилась в результате поиска компромисса между качеством и скоростью, одновременно с учетом возможностей современных процессоров и уже найденных способов (арифметико-логических комбинаций) перемешивания и распространения зависимостей (лавинного эффекта).

Базовый вариант t1ha является (самой) быстрой переносимой хэш-функцией для построения хэш-таблиц и других родственных применений. Поэтому базовый вариант t1ha ориентирован на 64-битные little-endian архитектуры, принимает 64-битное подсаливающее значение (seed) и выдает 64-битный результат, который включает усиление длиной ключа. Стоит отметить, t1ha намеренно сконструирована так, чтобы возвращать 0 при нулевых входных данных (ключ нулевого размера и нулевой seed).

Оценить качество хэш-функции во всех аспектах достаточно сложно. Можно идти аналитическим путем, либо проводить различные статистические испытания. К сожалению, аналитический подход малоэффективен для оценки хэш-функций с компромиссом между качеством и скоростью. Причем сравнительная аналитическая оценка таких функций стремиться к субъективной.

Напротив, для статистических испытаний легко получить прозрачные количественные оценки. При этом есть хорошо зарекомендовавшие себя тестовые пакеты, например SMHasher. Для t1ha результаты просты — все варианты t1ha проходят все тесты без каких-либо замечаний. С другой стороны, не следует считать, что у t1ha есть какие-либо свойства сверх тех, что необходимы для целевого применения (построение хэш-таблиц).

Бенчмарки

Стоит пояснить наличие в заголовке словосочетания «самая быстрая». Действительно, крайне маловероятно, что существует хэш-функция, которая будет полезной и одновременно самой быстрой на всех платформах/архитектурах. На разных процессорах доступны разные наборы инструкций, а схожие инструкции выполняются с разной эффективностью. Очевидно, что «всеобщая самая быстрая» функция, скорее всего, не может быть создана. Однако, представляется допустимым использовать «самая быстрая» для функции, которая является переносимой и одновременно самой быстрой как минимум на самой распространенной платформе (x86_64), при этом имея мало шансов проиграть на любом современном процессоре с достойным оптимизирующим компилятором.

В состав исходных текстов проекта входить тест, который проверяет как корректность результата, так и замеряет скорость работы каждого реализованного варианта При этом на x86, в зависимости

от возможностей процессора (и компилятора) могут проверяться дополнительные варианты функций, а замеры производится в тактах процессора.

Кроме этого, на сайте проекта приведены таблицы с результатам замеров производительности посредством доработанной версии SMHasher от Reini Urban. Соответственно, все цифры можно перепроверить и/или получить результаты на конкретном процессоре при использовании конкретного компилятора.

Здесь же можно привести сопоставление с некоторыми ближайшими конкурентами t1ha.

Хэширование коротких ключей (среднее для 1.31 байта, см. Табл. 1). Смотрим на правую колонку «Cycles/Hash» (чем меньше значение, тем быстрее):

Таблица 1: Хэширование коротких ключей

	Mib/Second	**Cycles/Hash**
th1a	12228.0	35.55
fasthash64	5578.06	43.42
City64	11041.72	51.77
xxHash64	11123.15	56.17
metrohash	11808.92	46.33

Хэширование длинных ключей (256 Кб, см. Табл. 2). Смотрим на среднюю колонку «MiB/Second» (чем больше значение, тем быстрее):

Таблица 2: Хэширование длинных ключей

	Mib/Second	**Cycles/Hash**
th1a	12228.0	35.55
FarmHash64	12145.36	60.12
City64	11041.72	51.77
xxHash64	11123.15	56.17
Spooky64	11820.2	60.39

Варианты t1ha

Разработка t1ha преследовала сугубо практические цели. Первой такой целью было получение быстрой переносимой и достаточно качественной функции для построения хеш-таблиц.

Затем потребовалась максимально быстрый вариант хэш-функции, который давал-бы сравнимый по качеству результат, но был максимально адаптирован на целевую платформу. Например, базовый вариант t1ha работает с little-endian порядком байт, из-за чего на big-endian архитектурах требуется конвертация с неизбежной потерей производительности. Так почему-бы не избавиться от лишних операций на конкретной целевой платформе? Таким же образом было добавлено ещё несколько вариантов:

- Упрощенный вариант для 32-битных платформ, как little, так и big-endian.
- Вариант с использованием инструкций AES-NI для процессоров без AVX.
- Два варианта с использованием инструкций AES-NI с использованием AVX.

Чуть позже стало понятно что потребуются ещё варианты, сконструированные для различных применений, включая разную разрядность результата, требования к качеству и стойкости. Такое многообразие потребовало наведения порядка. Что выразилось в смене схемы именования, в которой цифровой суффикс обозначает «уровень» функции:

t1ha0() — максимально быстрый вариант для текущего процессора.

t1ha1() — базовый переносимый 64-битный вариант t1ha.

t1ha2() — переносимый 64-битный вариант с чуть большей заботой о качестве.

t1ha3() — быстрый переносимый 128-битный вариант для получения отпечатков.

и т.д.

В этой схеме предполагается, что t1ha0() является диспетчером, который реализует перенаправление в зависимости от платформы и возможностей текущего процессора. Кроме этого, не исключается использование суффиксов «_le» и «_be» для явного выбора между little-endian и big-endian вариантами.

Таким образом, под «вывеской» t1ha сейчас находиться несколько хеш-функций и это семейство будет пополняться, в том числе с прицелом на отечественный E2K «Эльбрус».

Представление о текущем наборе функций и их свойствах можно получить из вывода теста. Стоит лишь отметить, что все функции проходят все тесты SMHasher, а производительность вариантов AES-NI сильно варьируется в зависимости от модели процессора:

```
Simple bench for x86 (large keys, 262144 bytes):
    t1ha1_64le:   47151 ticks,  0.1799 clk/byte,  16.679 Gb/s @3GHz
    t1ha1_64be:   61602 ticks,  0.2350 clk/byte,  12.766 Gb/s @3GHz
    t1ha0_32le:   94101 ticks,  0.3590 clk/byte,   8.357 Gb/s @3GHz
    t1ha0_32be:   99804 ticks,  0.3807 clk/byte,   7.880 Gb/s @3GHz
Simple bench for x86 (small keys, 31 bytes):
    t1ha1_64le:      39 ticks,  1.2581 clk/byte,   2.385 Gb/s @3GHz
    t1ha1_64be:      42 ticks,  1.3548 clk/byte,   2.214 Gb/s @3GHz
    t1ha0_32le:      51 ticks,  1.6452 clk/byte,   1.824 Gb/s @3GHz
    t1ha0_32be:      54 ticks,  1.7419 clk/byte,   1.722 Gb/s @3GHz
Simple bench for AES-NI (medium keys, 127 bytes):
    t1ha0_ia32aes_noavx:  72 ticks,  0.5669 clk/byte,   5.292 Gb/s @3GHz
    t1ha0_ia32aes_avx:    78 ticks,  0.6142 clk/byte,   4.885 Gb/s @3GHz
    t1ha0_ia32aes_avx2:   78 ticks,  0.6142 clk/byte,   4.885 Gb/s @3GHz
Simple bench for AES-NI (large keys, 262144 bytes):
    t1ha0_ia32aes_noavx: 38607 ticks,  0.1473 clk/byte,  20.370 Gb/s @3GHz
    t1ha0_ia32aes_avx:   38595 ticks,  0.1472 clk/byte,  20.377 Gb/s @3GHz
    t1ha0_ia32aes_avx2:  19881 ticks,  0.0758 clk/byte,  39.557 Gb/s @3GHz
```

Чуть подробнее о внутреннем устройстве

Если говорить чуть более детально, то t1ha построена по схеме Меркла-Дамгарда (энтропийная губка) с упрочнением от размера данных и подсаливающего значения. Внутри основного сжимающего цикла используется 256-битное состояние, с аналогичным размером входного блока. Причем для каждого операнда данных реализуется две точки инъекции с перекрестным опылением. По завершению сжимающего цикла выполняется сжатие 256-битного состояния до 128 бит.

При выполнении описанных действий используются 64-битные операции, комбинирующие миксеры ARX (Add-Rotate-Xor) и MUX/-MRX (Mul-Rotate-Xor). Немаловажно, что все эти вычисления выстроены так, чтобы обеспечить возможность параллельного выполнения большинства операция и плотной укладки u-ops как в конвейер, так и в исполняющие устройства x86_64. За счет этого достигается достаточно хорошее качество при практически предельной скорости хэширования длинных ключей.

Стоит отметить, что сжимающий цикл запускается только для блоков достаточного размера. Если же данных меньше, то промежуточное 128-битное состояние будет состоять только из размера ключа и подсаливающего значения.

Далее, оставшийся хвост данных порциями по 64 бита подмешивается попеременно к половинам 128-битного состояния. В заключении выполняется перемешивание состояния одновременно со сжатием до 64-битного результата. Немаловажной особенностью t1ha здесь является использование миксера на базе широкого умножения (128-битное произведение двух 64-битных множителей). Это позволяет реализовать качественно перемешивание с хорошим лавинным эффектом за меньшее количество операций. Несмотря на то, что широкое умножение относительно дорогая операция, меньшее количество операций позволяет t1ha обрабатывать короткие ключи за рекордно малое количество тактов процессора.

Следует отметить, что используемый миксер на основе широкого умножения и исключающего ИЛИ не идеален. Несмотря на то, что t1ha проходит все тесты SMHasher, у автора есть представление о последствиях неинъективности. Тем не менее, результирующее качество представляется рационально-достаточным, а в планах развития линейки t1ha уже отражено намерение предоставить чуть более качественный вариант.

Леонид Юрьев
Москва, Positive Technologies

Проект: libmdbx https://github.com/leo-yuriev/libmdbx

libmdbx key-value storage — кандидат в лучшие из встраиваемых

Аннотация

В докладе делается обзор высокопроизводительного движка хранения пар ключ-значения (key-value storage). Сейчас в проекте происходят огромные изменения, в результате которых MDBX (libmdbx) приобретает новые, недосягаемые ранее, свойства. В докладе рассказывается о целях инициированной революции, уже доступных и ожидаемых результатах.

Стоит вынести главный утверждающий тезис доклада — уже сейчас libmdbx является одним из самых быстрых key-value хранилищ, а в следующем году избавится от ряда унаследованных недостатков и перейдет в класс технологических лидеров.

Хранилища ключ-значения (key-value) уже давно стали трендом индустрии и фактически вытеснили реляционные СУБД из применений требующих высокой производительности или экономии ресурсов. При этом, несмотря на крайне широкий спектр уже доступных технологий идёт битва и поиск новых решений, в том числе более оптимальных в некоторых сферах применения или специфических условиях.

MDBX (libmdbx) — это классический встраиваемый ультра-быстрый движок хранения пар ключ-значение (key-value), но со специфическим набором свойств и возможностей, ориентированный на создание решений с предельной производительностью и/или контролируемым (прозрачным) поведением.

Следует ответить, что libmdbx действительно является чемпионом во многих открытых бенчмарках, в том числе по таким параметрам как: равномерность времени выполнения запросов, масштабируемость чтения, используемое место, потребление процессорного времени и т.д.

С другой стороны, libmdbx объективно будет отставать от конкурентов в некоторых сценариях использования. Поэтому к результатам сравнительных тестов следует относиться взвешенно, обязательно

принимая во внимание все детали, особенно предоставляемые гарантии сохранности данных, объём потерь и длительность восстановления после аварий.

История libmdbx

libmdbx является трёхгодичным форком Lightning Memory-Mapped Database (LMDB). Но только с 2017 года развитие проекта не сдерживается необходимостью совместимости с прародителем, в том числе по формату файлов БД.

Уместно сказать, что LMDB широко известен «в узких кругах» и используется во многих общеизвестных проектах, например: InfluxDB, Postfix, Cyrus SALS, Knot DNS, Power DNS, libpaxos, Monero и т. д. Родиной LMDB является проект OpenLDAP, где в 2010 году потребовалась замена для технологически и морально устаревшей Berkeley DB.

История libmdbx начинается осенью 2014 года. Изначально работа велась в составе проекта ReOpenLDAP, который был отпочкован от OpenLDAP из-за отказа основных разработчиков принимать часть изменений, которые были жизненно необходимы для «починки» (тушения пожара) и успешной эксплуатации в инфраструктуре ПАО МегаФон с полной проектной нагрузкой. К осени 2015 сделанные доработки приобрели самостоятельную ценность и исходный код движка хранения был выделен в отдельный проект libmdbx.

В 2017 году libmdbx получил новый импульс развития. С одной стороны, были выполнены обязательства по сохранению совместимости. С другой стороны, libmdbx был задействован в «Позитивных таблицах» компании Positive Technologies.

Основные свойства libmdbx

Как уже было пояснено, libmdbx является достаточно глубокой переработкой и развитием LMDB. Поэтому, с одной стороны, libmdbx наследует все характеристики и уникальные свойства LMDB, в том числе проблемы и недостатки. С другой стороны, libmdbx предлагает ряд успешных «заплаток» для всех проблем LMDB, а также предлагает дополнительные возможности.

Хотелось-бы избежать нахваливания и подробного описания libmdbx, в том числе LMDB как прародителя. Подобная информация доступна

в сети, в том числе на сайте проекта. Тем не менее, всё же необходимо тезисно перечислить основные свойства libmdbx:

1. Данные отображаются в память и используются совместно всеми процессами работающими с экземпляром БД.
2. В качестве модели данных предлагаются упорядоченные ассоциативные массивы с поддержкой курсоров и выборкой по диапазонам.
3. Запросы чтения выполняются параллельно без использования дорогих атомарных операций, в том числе никак не блокируются (lockfree).
4. Изменения выполняются строго последовательно, но не блокируются запросами чтения при наличии места в БД.
5. Чтение и обновление данных происходят только в рамках транзакций с полными гарантиями ACID, которые обеспечиваются посредством MVCC (параллельный доступ с помощью многоверсионности) и COW (копированием при изменении).
6. Амортизационная стоимость любой операции Olog(N), в том числе Write Amplification Factor и Read Amplification Factor.
7. Нет журнала транзакций или журнала опережающей записи. Не требуется восстановление после аварий. Не требуется какое-либо периодическое обслуживание. Поддерживается резервное копирование «на лету».
8. Поддерживаются ключи с множественными значениями. При этом ключи хранятся в одном экземпляре, а значения помещаются в аналоги «вложенных таблиц», для которых также доступны курсоры.
9. Код движка компактен (менее 64K для x86_64). Не имеет собственных механизмов управления памятью и кешированием. Не нуждается в дополнительных thread для собственных нужд (за исключением планируемого механизма асинхронной фиксации).

Сценарии использования libmdbx

Исходя из свойств и внутренних механизмов libmdbx можно выделить три группы областей применений или сценариев использования libmdbx: подходящие, пограничные, противопоказанные. Однако,

практика показывает что такое подразделение достаточно условно и правильнее оценивать сумму факторов, каждый из которых вытекает из отдельного свойства libmdbx и может быть как положительным, так и отрицательным в каждом конкретном случае. Кратко рассмотрим самые важные свойства и связанные с ними плюсы/минусы:

Встраиваемый движок:

Встраиваемость становится весомым плюсом когда отдельный серверный процесс не требуется, тем более если этого хочется избежать.

С другой стороны, при встраивании, со стороны остального кода, есть неизбежный риск повреждения внутренних структур данных движка. Что может приводить к повреждению БД. Характерным примером может служить «удачная» запись нуля по некорректному указателю, которая заставит движок сделать запись в нулевую (управляющую) мета-страницу БД.

Данные отображены в ОЗУ:

Отображение в память позволяет кардинально снизить накладные расходы на доступ к данным и не иметь какой-либо собственной инфраструктуры кеширования (всё необходимое выполняется ядром ОС). В частности, в libmdbx можно легко избежать лишнего копирования данных (zero copy) получая непосредственно указатель на read-only отображение в памяти.

С другой стороны, могут быть затруднения если объём данных превышает размер ОЗУ или механизмы управления виртуальной памятью неверно настроены и/или работаю не оптимально (Windows 8).

Кроме этого, в сравнении с хранилищами основанными на LSM-деревьях (Log-Structured Merge-tree), в libmdbx новые и старые данные равны между собой. Поэтому на больших объёмах libmdbx может проигрывать в сценариях с частым доступом к недавно добавленным данных. Однако, стоит отметить, что на практике ситуация зависит от массы факторов и может быть кардинально обратной. Например, «горячие» и/или недавно добавленные данные с большой вероятностью НЕ будут вытесняться из ОЗУ и кэша процессора, и тогда libmdbx может быть на порядок более эффективнее LSM-деревьев, что подтверждается соответствующими тестами.

Отсутствие журнала транзакций:

Пожалуй это один из самых принципиальных моментов, который, в зависимости от требований, может становиться как радикальным плюсом, так и большим минусом. Вопрос сложен и неразрывно связан

с гарантиями сохранности данных и производительностью для операций изменяющих данные. Но давайте попробуем тезисно разобраться (далее предполагается что читатель знаком или ознакомится с терминами «журнал опережающей записи» и «журнализация транзакций» в Википедии).

- При низком темпе изменений данных, отсутствие или наличие журнала не окажет значимой разницы. Скорее всего мы предпочтем полностью фиксировать данные на диске после каждой транзакции. Например, такой простой сценарий является классическим для LDAP.

  В этом случае libmdbx вероятно будет абсолютным чемпионом, так как позволяет выполнять читающие запросы сколь угодно параллельно, без блокировок и с минимальными накладными расходами.

- При высоком темпе изменений, точнее при фиксации этих изменений, узким местом становится диск. Чтобы записывать меньше данных, и преимущественно последовательно, придётся вести журнал и тратить на него ресурсы, в том числе что-то делать дважды: сначала делать промежуточное «быстрое» фиксирование в журнале, а позже ещё раз внутри БД.

  Однако, важнее то, что использование журнала потребует восстановления БД после аварии. Такое восстановление может быть длительным, и даже потребовать участия человека (например, при нехватке места), что в ряде применений может быть неприемлемым. Поэтому отсутствие фазы восстановления в libmdbx может стать решающим плюсом.

- Несомненно использование журнала дает выигрыш в производительности, особенно на шпиндельных дисках (HDD). Но нередко упускается тот факт, что для гарантии сохранности данных требуется фиксировать изменение самого журнала после каждой транзакции, что крайне драматически сказывается на итоговой производительности.

  Например, журнал обычно располагается в файловой системе, нередко на одном физическом устройстве с телом БД или другими изменяемыми сущностями. При этом реальная фиксация данных требует выполнения носителем всей очереди команд и выдачи подтверждения (наличие нескольких логических очередей не означает что драйвер и ОС будут ими пользоваться, а

также не означает что встроенный контроллер носителя будет действительно независимо их обслуживать). В результате, фиксация транзакции всё также «стреляет по диску дробью», как если-бы журнала не было. Причём в случае SSD ситуация во многом схожая — нет затрат на перемещение головок, зато данные пишутся большими секторами в несколько шагов, что при синхронном выполнении с подтверждением занимает достаточно времени.

Поэтому, практически всегда приходится жертвовать гарантиями сохранности данных. Проще говоря, если при аварии мы готовы потерять некоторое количество последних изменений, то можем фиксировать журнал реже или как-получится, тем самым подняв производительность ещё на порядок или более.

В целом, здесь речь всегда идёт о компромиссе между производительностью и гарантиями сохранности данных в терминах объема потерь при аварии. В libmdbx предлагается несколько вариантов подобного компромисса:
— «простая» отложенная запись в файлы ядром ОС;
— автоматическая фиксация по объёму изменений и/или по времени;
— асинхронная «догоняющая» фиксация (в разработке);
— полностью асинхронная фиксация по усмотрению ядра ОС;
— явная «ручная» фиксация через API.

По оценкам автора, за счёт предлагаемых компромиссов и возможностей, libmdbx может более чем успешно конкурировать в подавляющем большинстве сценариев. Тем не менее, следует понимать что в абсолютных цифрах WAF (Write Amplification Factor) в libmdbx, скорее всего и почти всегда, будет больше чем в движках с хорошо реализованным WAL (write-ahead log).

Проблема долгих чтений в MVCC:

Подобная проблема так или иначе существует во всех СУБД, которые обеспечивают конкурентную обработку читающих и пишущих запросов с соблюдением ACID (aka MVCC bloat). Суть в том, что для любой читающей транзакции, на всё время жизни такой транзакции, СУБД должна обеспечивать консистентный набор данных, который не меняется с точки зрения читателя. Если при этом происходит интенсивное изменение данных, то все данные могут быть изменены и для поддержания долгой читающей транзакции может потребовать-

ся дополнительное место, равное объёму данных на момент старта читающей транзакции.

В случае libmdbx проблема проявляется более остро и на это есть две веские причины:

- Ради производительности libmdbx отслеживает использование страниц через линейную проекцию истории их изменения. Поэтому сборка мусора крайне эффективна (дешева), но может происходить только последовательно от старых версий снимков данных к новым. В результате сборка мусора вынужденно приостанавливается на самой старой (дальней) используемой/читаемой точки в истории данных.

  Другими словами, одного «застрявшего» читателя достаточно чтобы приостановить сборку мусора и вызвать переполнение БД.

- Высокая производительность позволяет израсходовать всё свободное место относительно быстро.

Для решения проблемы текущая версия libmdbx предлагает использовать обратный вызов, который, в частности, может либо подождать «застрявшего» читателя, либо прервать его транзакцию.

В будущих версиях libmdbx планируется реализовать механизм аварийной сборки мусора, который ценой относительно большого объёма операций будет искать неиспользуемые страницы глядя «сквозь» всю линейную историю, не останавливаясь на последней используемой версии данных. Можно сказать, что этот механизм будет потребованию выполнять аналог VACUUM из PostreSQL.

Причины и цели революционных изменений

Уместно напомнить, что libmdbx отпочковался от LMDB вынужденно. В частности, из-за того что часть необходимых доработок не была принята. Более того, вся первоначальная активность была ради устранения производственно-эксплуатационных проблем возникших при эксплуатации OpenLDAP в ПАО МегаФон.

Это обстоятельство требовало «ехать, а не искать шашечки», в том числе нужно было устранять проблемы, а не улучшать движок. По той-же причине было невозможно нарушать совместимость с LMDB по формату файлов, а изменения API требовали немедленного отражения в исходном коде ReOpenLDAP.

Когда же в 2016 удалось добиться достаточно стабильной работы ReOpenLDAP, включа libmdbx и механизм репликации, активная доработка движка потеряла смысл. С одной стороны, просились радикальные изменения, но их реализация неизбежно увеличивала риски новых дефектов, и было-бы крайне нерационально привносить эти риски в промышленно эксплуатируемый сервис. С другой стороны, параллельная доработка без реальной живой потребности в ней, чревато «работой в стол» и усложнением поддержки эксплуатируемой версии.

Не углубляясь в детали, можно уверенно сказать, что в LMDB есть масса проблем и недостатков, устранить которые можно только кардинальной переработкой API и ещё более существенной переработкой формата файлов БД. В LMDB сделать подобное крайне затруднительно, так как фактически это будет означать создание нового движка и взорвет все зависящие проекты. В случае libmdbx подобного барьера не было, изменения напрашивались с самого начала, «зудели и кровоточили». Однако, по совокупности вышеназванных причин задуманные доработки «стояли на паузе» до 2017. Кроме этого, необходимость изменений заставляла требовала не рекламировать libmdbx, чтобы в итоге создать неудобства меньшему количеству пользователей.

Начиная же с 2017 года, понемногу, реализовывались не-кардинальные, но необходимые доработки, с постепенным увеличением масштаба и амплитуды. Так в частности, была возвращена поддержка Windows, а также решены проблемы с завершением потоков из-за архитектурных дефектов в thread-local-storage destructors в GNU libc.

Завершенные доработки

Здесь и далее подразумевают доработки и новые возможности относительно LMDB, как прародителя MDBX (libmdbx). Текущая версия (ветка master) содержит более 25 различных доработок и все они приемлемо-подробно описаны в README. Поэтому здесь приведем лишь некоторые ключевые:

- Поддержка LIFO порядка при «переработке мусора». Из-за чего цикл использования страниц памяти и секторов диска становился минимальным, что снижает эффект вымывание кэша и на порядок увеличивает эффективность очередей обратной записи;

- Поддержка пользовательского обработчика для случая исчерпания свободного места в БД из-за блокирования сборщика мусора долгими транзакциями чтения. Это позволяет управлять ситуацией, а в случае ReOpenLDAP позволило предотвратить «случайные» отказы сервиса;
- Полная переработка пути записи на диск, что дает гарантию сохранности БД. Проще говоря, libmdbx в отличие от LMDB в ряде режимов при аварии потеряет последние изменения, но не разрушит всю базу.

В 2017 году был начат процесс реализации задуманных доработок, включая изменения формата файлов БД. При этом внезапно образовалась неожиданная проблема — очень давно и очень многое хотелось изменить или переделать. Так давно, что многое уже забылось, а глаза привыкли к странностям кода и перестали их замечать. И так много, что начиная что-то менять неудержимо втягиваешься в переделку всего, в изменение каждой строчки кода...

Тем не менее, в этом году уже удалось сделать и довести до эксплуатации большие и важные доработки:

- Унификация формата файлов и структур для 32-битных и 64-битных сборок, включая переход на 32-битные номера страниц.
- Поддержку трех мета-страниц вместо двух, что на самом деле означает возможность прозрачно и произвольно комбинировать «сильные» и «слабые» точки фиксации, одновременно гарантированно и надежно соблюдая MVCC как для всех читателей, так и на случай любых аварий.
- Управление геометрией, включая выбор размера страниц, контролируемое увеличение и уменьшение размера БД.
- Автоматическую компактификацию БД без дополнительных затрат, с последующим автоматическим освобождением места на диске. Стоит пояснить, что в LMDB это большая боль, так как однажды увеличенный файл БД может быть уменьшен только путем копирования с компактификацией.

Планируемые доработки и новшества

- Переработка API с ликвидация хрупкости.

Если говорить кратко, то унаследованное API имеет много сложно связанных опций/флажков и соглашений, нарушение которых может приводить к сложно обнаруживаемым ошибкам.

Отдельного упоминания заслуживает путаница с термином «database» и странности API открытия/закрытия ассоциативных массивов. Из-за чего совершенно нетривиально эффективно организовать координацию читающих MVCC-транзакций с созданием и/или удалением таких массивов.

- Асинхронная догоняющая фиксация данных на диске.

  Технически libmdbx позволит запускать дополнительный thread, который будет заниматься фоновой фиксацией данных и формированием «сильных» точек фиксации.

  С точки зрения пользователя это добавит режимы «ленивой» и «догоняющей» фиксации данных, что с точки зрения обеспечения сохранности данных аналогично ведению журнала с асинхронной его фиксацией.

- Рефакторинг структур для уменьшения накладных расходов.

  Кроме изменения заголовков страниц будет переработан формат хранения списков.

- Возможность размещения больших страниц в отдельном файле.

  Технически эта большая переработка, затрагивающая все механизмы движка, которая позволит вынести хранение «больших записей» на отдельный носитель. В результате становится возможной гибридная модель хранения (например SSD + HDD), при которой индексы и короткие значения хранятся на быстром носителе, а большие данные вынесены на дешевый.

  С учетом внутренних механизмов libmdbx такая модель может быть предельно эффективной по показателю цена/производительность: для поиска данных используется быстрый носитель, а при доступе к большим записям происходит адресное последовательное чтение с дешевого носителя.

- Контрольные суммы страниц.

Контроль содержимого страниц на основе CRC64 или t1ha позволит обнаруживать ошибочное изменение данных. Выбор функции будет производиться при создании БД, причём для t1ha будет возможность выбрать вариант допускающие аппаратную акселерацию (AES-NI).

- Контроль консистентности БД посредством Merkle Tree.

  Технически эта большая переработка, требующая эффективной реализации «обратной волны» при обновлении страниц.

  С точки зрения пользователя появиться возможность верифицировать состояние БД, также как git проверяет совокупную целостность всех файлов в большом иерархическом проекте.

- Аварийный сборщик мусора.

  Позволит решить архитектурно-обусловленную проблему переполнения БД (исчерпания свободных страниц) из-за остановки сборки мусора на долгой читающей транзакции. Технически доработка сводиться к выявлению свободных страниц путем обхода B+Tree дерева для каждого используемого/читаемого снимка БД.

  С точки зрения пользователя, в результате эффективность использования места в БД, даже в проблемных ситуациях, сравняется с таким СУБД как Oracle. DB2, MSSQL, PostgreSQL и т.р.