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

Шестнадцатая конференция разработчиков свободных программ

Калуга, 27–29 сентября 2019 года

Тезисы докладов

Москва
Базальт СПО, МАКС Пресс
2019

УДК 004.91
ББК 32.97
П99

П99 Шестнадцатая конференция разработчиков свободных программ: Тезисы докладов. Калуга, 27–29 сентября 2019 года / отв. ред. Черный В. Л. — М.: Базальт СПО; МАКС Пресс, 2019. — 72 с. : ил.

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

ISBN 978-5-9908979-9-1
ISBN 978-5-317-06227-9 (МАКС Пресс)

Программа конференции

27 сентября, пятница

Дневное заседание
14.30–18.35

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

14.45–15.25 Голубев С. А.

15.25–16.05 Савченко А. А., ООО «Базальт СПО»

16.05–16.45 Чудов И. И, ООО «Базальт СПО»

16.45–17.05 Кофе-пауза

17.05–17.45 Ставцев Р. Г., АО «Байкал Электроникс»

17.45–18.25 Ермаков Н. О., ООО «Базальт СПО»

18.25–19.05 Левин Д. В., ООО «Базальт СПО»

28 сентября, суббота

Утреннее заседание
10.00–13.50

10.00–10.40 Михеев А. Г., ООО «Процессные технологии»

10.40–11.20 Румянцев М. Е., Хайдаров А. Р., МИЭТ

11.20–12.00 Синельников Е. А., ООО «Базальт СПО»

12.00–12.20 Кофе-пауза

12.20–13.00 Боковой А. Г., Red Hat

13.00–13.40 Мидюков А. П., ООО «Базальт СПО»

13.40–15.10 Перерыв на обед

Дневное заседание
15.10–18.30

29 сентября, воскресенье

Утреннее заседание
10.00–13.00

Сергей Голубев
Раменское

Как свободному проекту организовать работу со СМИ

Аннотация

В докладе рассматриваются некоторые аспекты информационной поддержки свободного проекта. Автор рассказывает о собственном опыте работы в данном направлении. Сделанные им выводы смогут помочь руководителям проектов эффективно спланировать PR-бюджет.

Коммерческие компании, занимающиеся разработкой или внедрением СПО, считают публикации в различных СМИ одной из важных составляющих продвижения продукта или услуги. Насколько эти ожидания оправданы? Какой эффект могут принести статьи? Как эффективно работать в данном направлении?

Зачем свободному проекту нужны публикации в СМИ

Один из главных мифов — публикации быстро и явно влияют на объёмы продаж. Это не так, особенно если речь идёт о контрактах с крупными компаниями или государственными учреждениями. Их руководители при принятии решения вряд ли руководствуются статьями в журналах. А СПО у России ориентированно именно на корпоративный и государственный сектора.

Какие же задачи свободного проекта можно решить при помощи публикаций или иных форм PR-деятельности?

Во-первых, публикации способствуют расширению сообщества. Как разработчиков, так и пользователей. Не исключено, что решаемая проектом задача покажется интересной кому-то ещё.

Во-вторых, публикации сокращают время обучения. Если хотя бы небольшая часть сотрудников компании знают о продукте заранее, то внедрение не встретит большого внутреннего сопротивления. А если удалось вовлечь их в сообщество, то задача облегчается радикально.

В-третьих, публикации могут дать ещё один канал обратной связи. Да-да, не все пользователи любят писать правильные баг-репорты на официальные баг-трекеры.

Средство массовой информации или система множества источников

Единственный формальный способ организовать управляемый поток публикаций в СМИ — договор с PR-агентством. Ещё можно регулярно создавать информационные поводы, но это стоит дороже, причём контроль за публикациями отсутствует.

Выход — создавать собственное СМИ. Только не как средство массовой информации, а как систему множества источников.

Соцсети люди читают значительно чаще, чем журналы — большинство опубликованных статей они находят по ссылкам из них. При этом совершенно не важно, именитое издание разместило материал или нет.

Свой блог: плюсы и минусы

Можно давать ссылки на заметки в собственном корпоративном блоге. Как это делает ЛК, причём вполне успешно. Пойдя традиционным путём она наверняка потратила бы больше ресурсов для достижения того же эффекта.

Безусловно, поддержание собственного блога в рабочем состоянии требует средств, но их потребуется несоизмеримо меньше, чем запросит PR-агентство. К тому же, все публикации будут контролироваться проектом, что гарантирует фильтрацию заведомо неграмотных статей.

Правда, ведение собственного блога — дополнительные организационные проблемы. К тому же, надеяться на одного-двух штатных авторов не стоит. Крайне желательно привлекать к написанию заметок других сотрудников, тексты которых могут потребовать редактирования.

Торопиться не надо

Сколько-нибудь заметный эффект от такой политики по самым оптимистичным прогнозам появляется на второй или даже третий год после начала его работы. Поэтому руководство проекта должно трезво оценивать свои возможности. Денег нужно относительно немного, но они нужны постоянно. Если нет уверенности в регулярном попол-

нении бюджета, то лучше отложить эту затею до более благополучных времён.

Андрей Савченко
Москва, Базальт СПО

Особенности портирования СПО на Эльбрус

Аннотация

Эльбрус является уникальной российской архитектурой с широким командным словом и расширенными возможностями обеспечения безопасности. Эти особенности обуславливают специфические проблемы при портировании программного обеспечения на данную платформу, ряд которых рассматривается в данной работе.

В отличие от большинства других архитектур Эльбрус (e2k) использует набор инструкций с широким командным словом (VLIW) и позволяет выполнять 25 инструкций общего назначения за такт (для 4-го поколения). Это свойство накладывает на компилятор существенно более серьёзные требования по оптимизации кода. Более того, с целью оптимизации использования площади кристалла все высокоуровневые функции, которые было возможно вынести из железа в компилятор, были перенесены туда. Грубо говоря, у Эльбрусов нет микрокода и все операции уровня микрокода выполняются на этапе компиляции, а не во время исполнения, как на семействе x86. Такой подход обеспечивает лучшую энергоэффективность и производительность на число транзисторов, но обратной стороной является то, что системный компилятор LCC является закрытым и, вероятно, останется таковым.

LCC старается соответствовать GCC по пользовательскому интерфейсу и поведению, но на этом общие черты заканчиваются. LCC основан на фронтенде EDG [1] и разработанном МЦСТ многоуровневом бэкенде.

Однако, свободное программное обеспечение также используется в тулчейне: libstdc++, libatomic, libgcc_s, C runtime и т.п. Было выполнено выделение свободных компонент и сборка их из исходных кодов, а также пакетирование в соответствии со стандартами Alt. На Эльбрусе реализован режим JIT компиляции x86/amd64 ассемблера, но

мы его не используем по соображениям как производительности, так и безопасности.

Большинство проблем портирования кода на уровне дистрибутива возникает из-за неполной совместимости LCC с GCC. Основной проблемой является фронтенд EDG, отвечающий за синтаксический анализ кода: используемые версии отстают от GCC по полноте поддержки стандартов и диалектов языков; применяемый ныне lcc-1.23 примерно соответствует лишь gcc-5.5. Особо ярко эта проблема проявляется на C++.

Однако, есть и иные проблемы:

- Иногда LCC замечает проблемы, пропускаемые GCC, например, неинициализированные переменные, что является особо надоедливой проблемой в комбинации с `-Werror`.
- lcc-1.23 реализует не все расширенные типы данных, что и gcc-5.5, например, нет `__uint128_t` и `_Decimal64`. Часто эти проблемы решаются с помощью макросов.
- Не все встроенные функции GCC поддерживаются, некоторые и не будут.
- Препроцессор LCC предполагает, что входные данные—это код C/C++ и заменяет табуляции выравнивания пробелами. Это ломает приложения, использующие препроцессор для обработки Makefile и иного чувствительного к табуляции кода.
- Ряд ПО использует недокументированные или сомнительные особенности компилятора: массивы переменной длины в структурах(VLAIS), вложенные функции, C++ без runtime. Они перестают работать как ожидается или вовсе не реализованы.

LCC поддерживает только C, C++ и Fortran, поэтому на данный момент на Эльбрусе нет поддержки языков, которым нужен собственный полноценный кодогенератор, например, Go, Rust, Haskell. В будущем эта проблема должна быть решена реализацией бэкенда LLVM.

Кроме особенностей компилятора, есть особенности самой архитектуры, помимо использования VLIW. Имеется три независимых аппаратных стека: для данных, функций и аргументов функций. Таким образом, система аппаратно защищена от целого класса атак. Но этому есть своя цена: необходима особая реализация для низкоуровневых операций с памятью вроде сборщиков мусора или переключения контекстов процесса.

Разумеется, системные вызовы и ioctl также отличаются и эту разницу следует учитывать, но это верно для любой новой архитектуры. Подроблее про особенности архитектуры, в т.ч. про тегированную память можно почитать в публичном официальном учебнике [2].

Внесение в апстримы патчей для e2k имеет свои ограничения, но всё же возможно. Основной проблемой является NDA с МЦСТ, не позволяющее публиковать патчи, полученные от МЦСТ. Тем не менее, собственные разработки публиковать *можно* без ограничений. Нами и иными разработчиками уже внесён вклад в ряд проектов (gimagereader, imake, lxc, ruby и др.), ознакомиться с которыми можно на нашей вики [22]. Апстримы, как и люди, все разные: кто-то с радостью принимает патчи, кто-то игнорирует, кто-то принципиально отказывается принимать, но продвижение кода всё же возможно. Есть надежда, что в будущем все патчи будут открыты и архитектура получит второе дыхание.

Литература

[1] Edison Design Group `https://www.edg.com/`

[2] Микропроцессоры и вычислительные комплексы семейства «Эльбрус» `http://www.mcst.ru/files/511cea/886487/1a8f40/000000/book_elbrus.pdf`

[3] Эльбрус upstream `https://www.altlinux.org/Эльбрус/upstream`

Игорь Чудов
Саратов, Базальт СПО

Проект: SBCL `http://www.sbcl.org/`

Проблемы портирования SBCL на новые аппаратные платформы

Аннотация

В докладе рассмотрена «проблема бутстрапа» реализации компилятора языка программирования Common LISP — SBCL (Steel Bank Common LISP) в приложении к архитектуре e2k. Рассмотрены составные элементы данного ПО, этапы бутстрапа, а также некоторые подробности низкоуровневого устройства. Данная работа призвана объединить разрозненные знания об устройстве проекта.

Проблематика

В целях портирования компилятора SBCL на архитектуру e2k было начато исследование проекта.

В процессе работы было обнаружено, что полноценное портирование на новые архитектуры выполняется достаточно редко, а портирование на архитектуры типа MIPS может выполняться по остаточному принципу в коде. В результате в новых версиях программы поддержка конкретной архитектуры может не гарантироваться.

Так как писать компилятор языка программирования на этом же языке программирования является достаточно популярной тенденцией — при возникновении новых архитектур возникает «проблема бутстрапа», которую иногда называют проблемой «курицы и яйца». SBCL также подвержен данной проблеме, но в относительно небольшой степени.

Устройство и этапы сборки

Бутстрап SBCL невозможен без существования интерпретатора Common LISP, но на начальном этапе достаточно иметь таковой на хостовой платформе. Для собственно инициализации сборки в таком случае необходим компилятор C, ассемблер и линкер для целевой платформы.

Простейший процесс сборки для новой платформы выглядит так:

1. Создание файлов конфигурации для сборки (`make-config.sh`);
2. Сборка кросс-компилятора Common LISP. Данная часть называется GENESIS 1;
3. Сборка компонентов целевого компилятора: эта часть собирается после GENESIS 1, чтобы получить файл `sbcl.h` для сборки части, написанной на C, и до сборки GENESIS 2, чтобы получить таблицу символов.
4. Запускается кросс-компилятор для получения объектных файлов для целевой операционной платформы.
5. Запускается GENESIS 2, чтобы получить `cold-sbcl.core`. Создаётся с помощью `src/compiler/generic/genesis.lisp`, который преобразует FASL файлы в образ LISP-системы на хосте.
6. Производится warm init — загрузка и сборка CLOS (Common LISP Object System), а также компонентов, от неё зависящих.

Для успешного прохождения всех этапов необходимо реализовать часть системы, которая зависит от целевой архитектуры.

Части системы

SBCL технически является компилятором — каждое поступающее на вход выражение сначала проходит этап компиляции в двоичный код, и только потом исполняется. Соответственно, для выполнения этой задачи в SBCL существует платформо-зависимая часть.

Платформо-зависимая часть состоит из определения регистров в файле `src/runtime/platformname-lispregs.h`, где `platformname` — название целевой архитектуры. Этот элемент необходим для описания доступных для работы регистров.

Также существует ассемблерная часть в файле `src/runtime/platformname-assem.S`, где определены всего две функции — `call_into_c` и `call_into_lisp`, которые выполняют переключение между C и SBCL ABI. Одна из задач этой части — переключение на C runtime, который отвечает за обработку сигналов, полученных от операционной системы. Для реализации таковых функций необходимо хорошее знание platform ABI целевой архитектуры, а также особенностей целевой операционной системы.

Самая объёмная и сложная платформо-зависимая часть системы находится в директориях `src/assembly/platformname` и `src/compiler/platformname` и содержит определения виртуальных операций (VOP) которые необходимы, чтобы преобразовать код на LISP в ассемблерный код, и используются в виртуальной машине SBCL в качестве низкоуровневых операций для в стандартных функциях ANSI Common LISP.

Одним из плюсов использования Virtual OPerations является возможность получать детализированные ассемблерные листинги при вызове дизассемблирования.

Проблемы при портировании на архитектуру e2k

В ситуации с портированием SBCL на архитектуру e2k в ОС ALT Linux были обнаружены следующие проблемы:

- Отсутствие кросс-компилятора C. Многие компиляторы, как SBCL, GHC и другие, не рассчитаны на сборку только на целевой платформе;

- Отсутствие детальной документации на C ABI для e2k. Данное знание необходимо для реализации функции переключения между SBCL ABI и C ABI, а также для эффективной реализации VOPs. Некоторые вещи приходится изучать методом исследования ассемблерных листингов;
- Частично утеряна документация по SBCL internals. К сожалению, лучшее средство изучения проекта — чтение кода и общение в списках рассылки;
- Сложность ручного просчёта блоков инструкций для широких слов. Ассемблер e2k плохо подходит для написания кода вручную;

Литература

[1] SBCL Internals CLiki site. https://web.archive.org/web/20120814000933/http://sbcl-internals.cliki.net/index

Роман Ставцев
Москва, Зеленоград, АО«БАЙКАЛ ЭЛЕКТРОНИКС»

Проект: Комплект средств разработки ПО (SDK) для BE-M1000
https://www.baikalelectronics.ru/products/

Процессор Байкал-М. Программное и аппаратное окружение

Аннотация
Краткий обзор СнК Байкал-М, тестовой платы DBM и комплекта средств разработки ПО SDK (BE–M1000) на основе СПО.

BE–M1000

Микропроцессор BE–M1000, другое название Байкал–М относится к типу Система—на—кристале. Микропроцессор имеет восемь ядер ARM®Cortex®–A57 с рабочей частотой 1.5 ГГц и оснащён когерентной кэш памятью L1, L2, и L3 уровня. Видеоподсистема включает два видеоконтроллера (LVDS и HDMI) и 4K видео декодер. Графический сопроцессор ARM®Mali^{TM}–T628 имеет восемь графических

ядер. СнК содержит два контроллера памяти DDR3/4 СнК поддерживает технологию безопасности ARM®TrustZone®. СнК обладает широким набором периферийных интерфейсов:

- PCIe Gen3;
- 10Gb Ethernet;
- 1Gb Ethernet;
- USB 3.0;
- USB 2.0;
- SATA 6G;
- eMMC/SD;
- I2S;
- eSPI/SPI;
- UART;
- I2C/SMBus;
- GPIO;

СнК производится по 28–нм технологии на Тайваньской фабрике TSMC. Рабочий диапазон температуры МП подтверждён испытаниями в пределах 0–70°C, по расчётным данным диапазон рабочих температур может достигать –45...70°C. СнК выпускается в 1521–выводном BGA корпусе размером 40×40 мм. шаг выводов 1 мм. Энергопотребление не более 28.5 Вт.

Блок-Схема BE-M1000

Тестовая плата DBM

Компанией для собственных нужд была разработана и произведена плата DBM, форматом 308×244 мм. На плате установлен СнК и реализован доступ ко всем низкоскоростным и высокоскоростным интерфейсам. В дальнейшем планируется выпустить отладочные платы/оценочные платы для заказчиков.

Комплект средств разработки ПО (SDK)

Комплект средств разработки программного обеспечения (далее SDK) полностью создан на базе СПО. SDK содержит кросс-компилятор языков C и C++, редактор связей, отладчик, утилиты

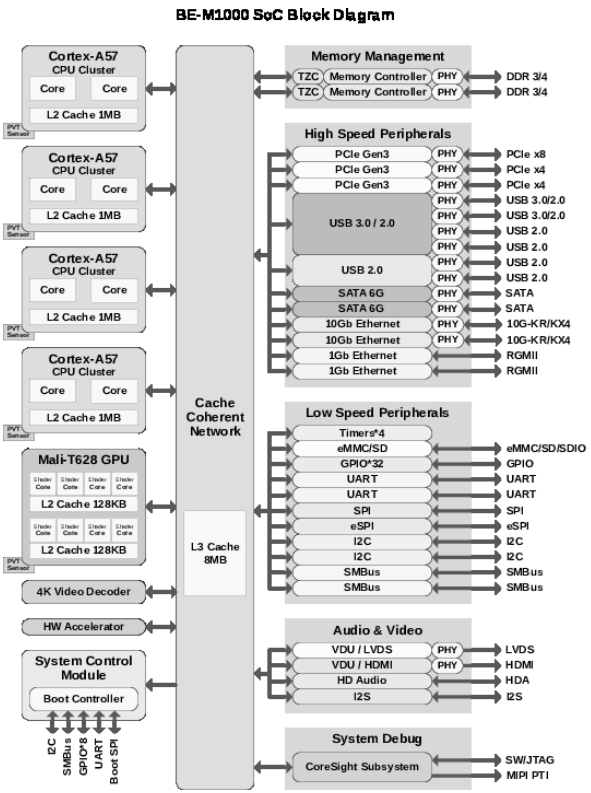

Рис. 1: Блок-Схема BE-M1000

и системные библиотеки, достаточные для разработки системного и прикладного ПО, исполняемого на микропроцессором. В состав SDK входит минимальная дистрибуция ОС Linux для целевой платформы с микропроцессором Байкал-М (на основе системной утилиты busybox). SDK поддерживает следующие целевые платформы:

– QEMU*
– Тестовая плата DBM
– Оценочная плата

Никита Ермаков
Москва, Базальт СПО

Архитектура RISC-V

Аннотация

RISC-V это гибкая, модульная архитектура набора команд (ISA) с открытой лицензией Creative Commons Attribution 4.0 International. Существуют как FPGA, так и ASIC (например HiFive Unleashed, Kendryte K210) реализации архитектуры. В данной работе приведён обзор архитектуры RISC-V, программной экосистемы, а также текущие результаты портирования ALT на RISC-V (rv64gc).

RISC-V [1] это новая, гибкая, модульная, little-endian архитектура набора команд (ISA), имеющая открытую лицензию Creative Commons Attribution 4.0 International (CC BY 4.0) [2]. Летом 2010 года была начата работа над первой версией ISA, которая уже осенью того же года использовалась в курсе по дизайну интегральных схем университета Беркли [3]. Первая версия ISA была опубликована под BSD лицензией и была предназначена для образовательных и академических целей. В августе 2014 года на конференции HotChips-26 [4] состоялась первая публичная демонстрация архитектуры RISC-V. На конференции были показаны ядра Rocket [5] и Hwacha [6]. Позже, в 2015 году, был основан фонд RISC-V Foundation [7], а в 2018 году Linux Foundation [8] и RISC-V Foundation объявили о сотрудничестве с целью ускорения open source разработки и адаптации RISC-V ISA. На данный момент RISC-V Foundation включает в себя более 325 участников [9].

Основу архитектуры RISC-V составляет базовая целочисленная (integer) ISA, которая должна присутствовать в любой реализации процессора плюс одно или несколько расширений. Базовая целочисленная ISA представляет собой минимальный набор инструкций необходимых для компиляторов, ассемблеров, компоновщиков и операционных систем [1] и включает в себя инструкции для записи, сохранения, переходов и целочисленной арифметики. Существует четыре варианта базовой целочисленной ISA: RV32I, RV64I, RV32E и RV128I. Каждый вариант характеризуется шириной целочисленных регистров и размером пользовательского адресного пространства. Первые два

варианта отвечают 32-х и 64-х битному адресному пространству соответственно. Аппаратные и программные (ОС) реализации могут обеспечивать как два этих типа ISA сразу (RV32I и RV64I), так и один из них. RV32E представляет собой уменьшенную копию RV32I разработанную для embedded систем. Было обнаружено, что в минимальных дизайнах ядер RV32I верхняя половина регистров (**x16-x31**) занимает четверть общей площади кристалла (за исключением памяти). Цена за один кристалл зависит от его площади, как $cost \sim f(A^2)$, где A – его площадь [10, 11]. Уменьшение числа регистров позволяет снизить стоимость ядра и его энергопотребление, что критично для микроконтроллеров и embedded систем. Основные отличия RV32E от RV32I:

- уменьшен размер регистров с 32 до 16;
- регистры счётчиков циклов, выполненных инструкций и таймера больше не являются обязательными для реализации.

RV128I является расширением RV64I на 128-ми битное адресное пространство так же, как и RV64I является расширением RV32I на 64-х битное адресное пространство. На данный момент RV128I не завершена и может быть изменена, когда появится первое реальное использование 128-ми битного, плоского, адресного пространства.

Помимо базовой целочисленной ISA, существуют различные расширения:

- **M** — умножение и деление целых чисел;
- **A** — атомарные операции;
- **F** — число с плавающей запятой одинарной точности;
- **D** — число с плавающей запятой двойной точности;
- **Q** — число с плавающей запятой четверной точности;
- **L** — десятичная плавающая точка;
- **C** — сжатые 16-битные инструкции;
- **V** — векторные инструкции;
- **B** — инструкции для манипуляций с битами;
- **J** — динамически транслируемые языки (например Java). ISA будет поддерживать динамические проверки и сборщик мусора;
- **T** — транзакционная память;

- **P** — упакованные SIMD инструкции;
- **N** — прерывания уровня пользователя.

На данный момент завершены следующие ISA: RV32/64I, M, A, F, D, Q, C. Набор расширений IMAFD для краткости обозначают одной буквой G (General). Варианты RISC-V со спецификацией RV32/64GC являются совместимыми с ОС на базе ядра Linux. Описанные выше модули относятся к пользовательскому уровню ISA. В произвольный момент времени, ядро RISC-V работает на определённом уровне привилегий. На данный момент определены три уровня привилегий в которых может находиться ядро RISC-V, в порядке увеличения привилегий: пользовательский (U), супервизора (S) и машинный (M). Уровни привилегий используются для защиты между разными компонентами программного стека. Попытка выполнения операций не предусмотренных данным уровнем привилегий приведёт к возникновению исключения. Машинный уровень (M) имеет наивысшие привилегии и является обязательным для реализации в любом ядре RISC-V [12]. Уровни пользователя и супервизора предназначены для пользовательских приложений и операционных систем соответственно. Текущий уровень привилегии ядра RISC-V кодируется двумя битами, что позволяет закодировать 4 значения уровня. Одно из значений зарезервировано для уровня гипервизора, который находятся в разработке.

Среди разработчиков программного обеспечения идёт активное освоение новой архитектуры. Ядро Linux [13] начиная с версии 4.15 имеет поддержку архитектуры RISC-V, а начиная с версии 4.19 содержит все необходимые драйверы для загрузки системы в QEMU [14]. Тулчейн получил поддержку RISC-V начиная с версий gcc 7.1 [15], binutils 2.28 [16], glibc 2.27 [17], gdb 8.3 [18]. Среди загрузчиков на данный момент доступны GRUB [19], U-Boot [20], Berkeley bootloader [21]. Разработчики операционных систем на базе ядра Linux активно портируют их на RISC-V. Существуют порты ALT [22], Debian [23], Fedora [24], Gentoo [25]. Порт ALT на RISC-V (rv64gc) построен вокруг репозитория Sisyphus [26]. На данный момент порт репозитория Sisyphus насчитывает более 5700 исходных RPM пакетов опубликованных в открытом доступе [27]. Сборка осуществляется в нативном режиме на плате HiFive Unleashed [28]. Для обеспечения воспроизводимой и безопасной сборки используется hasher [29]. Порт ALT на RISC-V имеет несколько вариантов образов системы:

- Минимальная система с systemd;

- Минимальная система с SysV;
- Минимальная система для разработчика (rpmbuild, hasher, gcc);
- Графическая система с XFCE (X11, VNC).

Каждый из вариантов доступен как для QEMU, так и для системы на кристалле (SoC) HiFive Unleashed.

Литература

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

[2] Creative Commons Attribution 4.0 International License `https://creativecommons.org/licenses/by/4.0/`

[3] Krste A. Instruction Sets Should Be Free: The Case For RISC-V / Krste Asanović и David A. Patterson // EECS Department, University of California, Berkeley. — 2014.

[4] RISC-V на HotChips-26, `https://riscv.org/2014/08/risc-v-at-hotchips-26/`

[5] Ядро Rocket RISC-V, `https://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/`

[6] Ядро Hwacha RISC-V, `http://hwacha.org/`

[7] RISC-V Foundation, `https://www.linuxfoundation.org/`

[8] Linux Foundation, `https://www.linuxfoundation.org/`

[9] Участники RISC-V Foundation, `https://riscv.org/members-at-a-glance/`

[10] John H. Computer Architecture: A Quantitative Approach / John L. Hennessy и David A. Patterson // ISBN: 978-0-12-383872-8. — 2012.

[11] David P, The RISC-V Reader: An Open Architecture Atlas / David A. Patterson и Andrew Waterman // ISBN: 978-0-99-924911-6. — 2017.

[12] The RISC-V Instruction Set Manual Volume II: Privileged Architecture, `https://riscv.org/specifications/privileged-isa/`

[13] Ядро Linux, `https://www.kernel.org/`

[14] Эмулятор процессора QEMU, `https://www.qemu.org/`

[15] The GNU Compiler Collection, `https://gcc.gnu.org/`

[16] GNU Binutils, `https://www.gnu.org/software/binutils/`

[17] The GNU C Library, `https://www.gnu.org/software/libc/`

[18] GDB: The GNU Project Debugger, `https://www.gnu.org/software/gdb/`

[19] GNU GRUB, https://www.gnu.org/software/grub/

[20] Das U-Boot — the Universal Boot Loader https://www.denx.de/wiki/U-Boot

[21] RISC-V Proxy Kernel and Boot Loader, https://github.com/riscv/riscv-pk

[22] Порт ALT на RISC-V, https://www.altlinux.org/Regular/riscv64

[23] Порт Debian на RISC-V, https://wiki.debian.org/RISC-V

[24] Порт Fedora на RISC-V, https://fedoraproject.org/wiki/Architectures/RISC-V

[25] Порт Gentoo на RISC-V, https://wiki.gentoo.org/wiki/Project:RISC-V

[26] Проект Sisyphus, https://www.altlinux.org/Sisyphus

[27] Порт репозитория Sisyphus на RISC-V, http://ftp.altlinux.org/pub/distributions/ALTLinux/ports/riscv64/Sisyphus/

[28] HiFive Unleashed, https://www.crowdsupply.com/sifive/hifive-unleashed

[29] Hasher, инструмент безопасной и воспроизводимой сборки пакетов, https://www.altlinux.org/Hasher

Дмитрий Левин
Москва, Базальт СПО
https://www.kernel.org/

linux kernel: история одного изъяна

Аннотация

Рассказ о том, как исправляются ошибки в архитектуре ядра linux, на примере одного изъяна в архитектуре ядра linux на x86-64, который просуществовал с 2001 по 2019 год и был исправлен в linux 5.3.

Введение

Архитектура x86-64 в ядре linux практически с самого начала поддерживала исполнение как традиционных инструкций x86, так и традиционных системных вызовов x86 с помощью прерывания `int 0x80` таким образом, что прерывания `int 0x80` могут вызывать как 64-битные, так и 32-битные процессы.

При этом ядро linux не предоставляло пользовательского интерфейса, позволяющего достоверно установить, какой именно системный вызов сделан, 64-битный или 32-битный, поэтому отладчики и трассировщики традиционно угадывали битность системного вызова по битности процесса.

Традиционный подход

Поддержка x86-64 в strace появилась 17 лет назад – в сентябре 2002 года, в ней битность системного вызова угадывалась по содержимому регистра `CS`, т.е. по битности процесса.

Первое известное мне упоминание [1] о проблеме этого подхода датируется 2008 годом, в нём приводится следующий пример:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("--------------------\n");
    __asm__("movl $2, %eax; int $0x80");
    printf("[i am %d]\n", getpid());
    return 0;
}
```

Обычный запуск этой программы на архитектуре x86-64 предсказуемо выводит

```
--------------------
[i am 29280]
[i am 29281]
```

Однако под strace происходит примерно следующее:

```
write(1, "--------------------\n", 21--------------------
)  = 21
open(NULL, O_RDONLY|O_CREAT|O_TRUNC|O_DIRECT|O_NOATIME|O_PATH|0x800020, 0154300) = 10873
getpid()                                = 10872
[i am 10873]
write(1, "[i am 10872]\n", 13[i am 10872]
)          = 13
exit_group(0)                           = ?
```

Альтернативный подход

Одновременно с поддержкой x86-64 в strace был добавлен закомментированный код, определявший битность системного вызова путём анализа инструкции, приведшей к системному вызову. Такой подход требует чтения памяти по адресу, на который указывает регистр `RIP`. Однако чтение памяти процесса для выяснения битности системного вызова – это не только дополнительный замедляющий системный вызов для отладчика, но и race condition, поскольку содержимое памяти, прочитанное отладчиком, может отличаться от фактически выполненного системного вызова.

Обсуждение

Отсутствие в ядре linux пользовательского интерфейса, позволяющего достоверно установить, какой именно системный вызов сделан, было осознано как проблема достаточно давно. Например, в начале 2012 года была живая дискуссия [2] на эту тему, в ней поучаствовали, в частности, Linus Torvalds, Andi Kleen, Andrew Lutomirski, Chris Evans, Denys Vlasenko, H. Peter Anvin, Indan Zupancic, Jamie Lokier, Oleg Nesterov, Pedro Alves, Roland McGrath, Will Drewry. Было высказано много идей разной степени экстравагантности, однако дальше дальше дискуссии дело не пошло.

Решение

Для того, чтобы перейти от общих рассуждений к решению, нужны были активно заинтересованные люди. В данном случае, к сожалению, их не было до до ноября 2018 года – именно тогда Эльвира Хабирова предложила к обсуждению первую редакцию решения [3].

Обсуждение привело к тому, что вторая редакция содержала уже 16 патчей, 15 из которых расширяло и упорядочивало внутренний API ядра на многочисленных архитектурах. В результате последующих итераций обсуждений патчсет вырос и стал затрагивать разные подсистемы и все архитектуры настолько, что было принято решение разделить его и апстримить по частям через разные деревья, поскольку никто не хотел брать патчсет целиком.

Результат

В конечном итоге цель была достигнута. Всего для реализации интерфейса PTRACE_GET_SYSCALL_INFO в ядре linux понадобилось 29 патчей, часть из них прошла 11 итераций обсуждения, а со дня отправки первой редакции патча [3] 7 ноября 2018 года до завершения приёма последнего патча [4] 17 июля 2019 года прошло немногим менее 9 месяцев.

Поддержка интерфейса PTRACE_GET_SYSCALL_INFO реализована в strace, начиная с версии 4.26, выпущенной в декабре 2018 года. strace переключается на использование этого интерфейса автоматически, если ядро linux его поддерживает.

Литература

[1] Debian Bug report #459820
https://bugs.debian.org/459820

[2] Re: Compat 32-bit syscall entry from 64-bit task!?
https://lore.kernel.org/lkml/CA+55aFzcSVmdDj9Lh_gdbz1OzHyEm6ZrGPBDAJnywm2LF_eVyg@mail.gmail.com/T/#u

[3] [RFC PATCH] ptrace: add PTRACE_GET_SYSCALL_INFO request
https://lore.kernel.org/lkml/20181107042751.3b519062@akathisia/

[4] linux kernel commit v5.3-rc1 65$\hat{2}$ 22
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=ac76de555d76b8cc7f8ef231692a3ad9cbd0ce63

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

Свободная система управления бизнес-процессами и административными регламентами RunaWFE Free. Версия 4.4.0

Аннотация

RunaWFE Free — свободная система управления бизнес-процессами и административными регламентами. В докладе описаны изменения, вошедшие в выпущенную в сентябре 2019 г. версию 4.4.0.

Система RunaWFE Free

RunaWFE Free — свободная система управления бизнес-процессами и административными регламентами. Система создает задания исполнителям в соответствии с положениями перемещающихся точек управления на схеме бизнес-процесса. Позволяет повысить производительность труда, прозрачность бизнеса, дает возможность быстрого изменения бизнес-процессов предприятия в ответ на изменение условий бизнеса.

Изменения версии 4.4.0 по сравнению с версией 4.3.0

Изменение главного меню системы

В меню появились следующие пункты (на Рисунке 1 они выделены рамками):

- Источники данных
- Скрипты
- Ошибки
- Критерии замещения
- Системные данные
- Задания сотрудников

В меню «Источники данных» группируются параметры, используя которые можно связать ботов с таблицами реляционных баз данных. В меню «Скрипты», «Ошибки» и «Системные данные» перемещены соответствующие разделы меню «Система» версии 4.3.0. В меню «Задания сотрудников» пользователь может посмотреть задания подчиненных сотрудников без возможности их выполнения.

Изменение системы аутентификации

В системе была существенно упрощена архитектура объектов системы безопасности. Теперь каждому классу сущностей соответствует свой список прав, которые применимы к данному классу (Эти права показываются в столбцах форм “Обладатели полномочий”). Что дает возможность легко вводить новые типы объектов безопасности и новые типы прав.

В частности, в системе появился новый тип прав на объекты безопасности «Полный доступ», который представлен на Рисунке 2.

Другие изменения

- Расширены настройки фильтров задач
- Расширены настройки фильтров экземпляров процессов
- Реализован просмотр задач других пользователей
- Добавлены элементы генерации и обработки событий
- Реализован "непрерывающий" режим работы обработки событий
- Реализовано чтение переменных из другого экземпляра бизнес-процесса

Рис. 1: Новые пункты меню.

- Добавлены глобальные роли
- Добавлены источники данных
- Расширены возможности бота внешнего хранилища

Ссылки

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

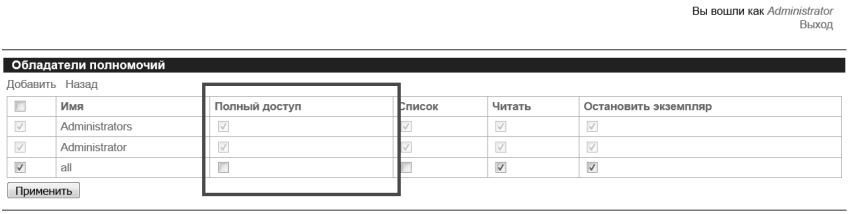

Рис. 2: Новый тип прав «Полный доступ».

Михаил Румянцев, Александр Хайдаров
Зеленоград, МИЭТ

Проект: RunaWFE-Free `http://www.runawfe.org/`

Решение проблемы организации быстрой связи между исполнителями при работе с бизнес-процессами, для внедрения в свободной системе RunaWFE

При работе с Бизнес процессами в среде RunaWFE, существует потребность в online взаимодействии пользователей. В докладе представляется решение данной задачи в виде программного модуля чата с использованием технологий WebSoket и Ajax, объясняется функционал и способ его реализации.

Системы управления бизнес-процессами генерируют задания исполнителям в соответствии с движением точек управления по схеме бизнес-процесса и таким образом координируют совместную работу сотрудников, а также информационных систем предприятия. При этом у исполнителей текущих заданий часто возникает потребность получения дополнительной информации от пользователей, выполнивших какие-то иные задания бизнес-процесса. Для обмена информацией между исполнителями заданий в системе управления бизнес-процессами RunaWFE, был разработан модуль online взаимодействия участников экземпляра бизнес-процесса в виде чата.

Ввиду того, что система RunaWFE изменяется под потребности конкретных пользователей (компаний) было решено выполнить чат в

Таблица 1: Таблицы БД чата

название таблицы	описание
CHAT_MESSAGE	хранение текста и автора сообщений
CHAT_MESSAGE_FILES	хранит файлы для контроля занимаемой памяти без удаления всего сообщения
CHATS_USER_INFO	для хранения индивидуальных настроек пользователя в конкретном чате и номера последнего непрочитанного им сообщения

виде полностью автономного модуля-тега, который будет легко встроить в любую html страницу. Для того чтобы окно чата не заслоняло нужный контент страницы он выполнен в виде кнопки, вызывающей открытие перемещаемого немодального окна поверх страницы, у которого есть 2 варианта размера.

Для администрирования/модерации чата была введена система chat_id, которая позволяет объединять чаты с разных страниц в общие чаты. Администратор может отслеживать происходящее во всех подконтрольных ему чатах одновременно, не отвлекаясь на постоянные переключения между окнами.

Ввиду того что бизнес-процесс может длиться долгое время, была реализована система цитирования сообщений, которая позволяет создавать многоуровневые вложения цитат, подгружаемые по мере необходимости и позволяющие пользователям просмотреть о чем шла речь в переписке без долгого поиска среди ленты старых сообщений.

Для передачи информации между чатом и сервером изначально применялись Ajax-запросы т.к. они уже применялись в системе повсеместно, однако ввиду необходимости инициирования получения информации сервером был добавлен web-socket для обмена сообщениями, а Ajax применяется только для подгрузки цитируемых сообщений с целью экономии трафика и улучшения производительности чата.

Хранение сообщений в базе данных выполнено с использованием библиотеки hibernate и разделено на таблицы (см. Табл. 1)

Функциональность текущего решения:

Кнопка «Открыть чат» — открывающая чат встраивается в html страницу (см. Рисунок 1)

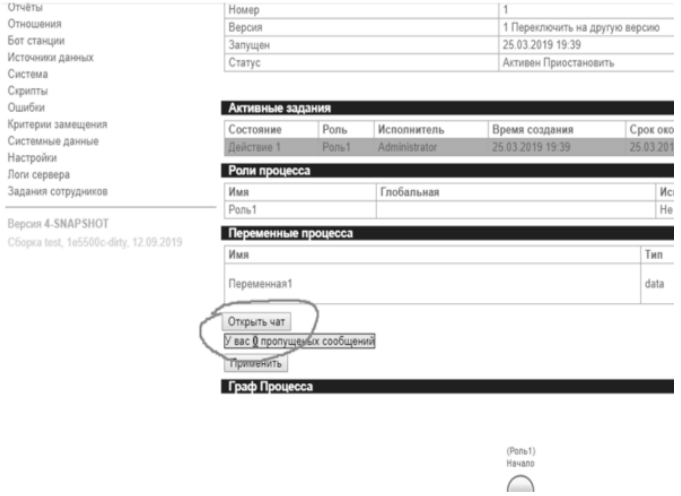

Рис. 1: Кнопка «Открыть чат»

Встроенный в код страницы тег чата:

На страницу добавляется три js-скрипта (в дальнейшем планируется их объединить), один css и тег с chat id (identifiableId) в желаемое местоположение кнопки открытия чата. (см. Рисунок 2)

- кнопка слева-сверху изменяет размер чата (на скриншете размер большой)
- кнопка «удалить» внизу сообщений присутствует только у адинистратора
- кнопка «ответить» добавляет цитирование сообщений, цитируемое сообщние появляется в нижней части чата, ограничение — около 100 единовременно цитируемых на 1 уровне вложения (т.е. не считая цитируемые в них сообщения), что является избыточным и позволяет пользователю об этом не беспокоится.

Для прикрепления файлов предусмотрено 2 способа:

1. кнопка «выбрать файлы»

Рис. 2: Внешний вид чата в развернутом виде

2. перетаскивание файла в область ввода сообщений (см. Рисунок 3)

Планы доработки чата:

- Ввести в чат систему вывода картинок, видео и таблиц в браузере (без их загрузки).
- Передача в чате ссылок на элементы бизнес-процесса для перехода к ним прямо из чата.
- Поправить смещения кнопки «ответить» при большом количестве развернутых уровней цитирования.

Литература

[1] Михеев А. Г. Системы управления бизнес-процессами и административными регламентами на примере свободной программы RunaWFE — 2-е изд. — М.:ДМК Пресс, 2016

[2] Илюшечкин В. М. Основы использования и проектирования баз данных

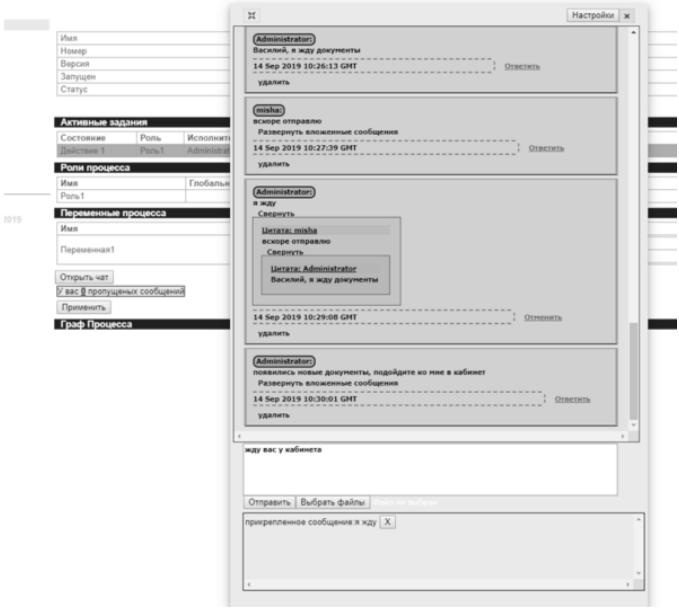

Рис. 3: Перетаскивание файла в область ввода сообщений

Таблица 2: Отладка и тестирование

Браузер	Результат
Google Chrome	работает нормально
Yandex browser	работает нормально
FireFox	работает нормально
Opera	работает нормально
Microsoft Edge (платформа Chromium)	работает нормально
Microsoft Edge (движок EdgeHTML)	имеется проблема с миганием при drag-and-drop файлов, альтернативный способ прикрепления кнопкой «обзор» работает. наблюдается смещение размеров сообщений (только визуальный баг)

Синельников Евгений
Саратов, ООО «Базальт СПО»

Проект: oddjob-gpupdate
http://git.altlinux.org/people/sin/packages/oddjob.git

Механизмы применения групповых политик в решениях на базе дистрибутивов ALT

Аннотация

Реализация групповых политик требует оформления набора правил и настроек как в виде декларативного описания, так и в виде набора инструментов их интерпретации и применения. Современные популярные средства управления конфигурациями под Linux обладают всеми возможностями для реализации механизмов применения настроек, но ограничены в интеграции инфраструктурой, а также не включают в себя инструментарий управления пользовательскими конфигурациями. Данный доклад посвящён реализации управления пользовательскими конфигурациями с помощью групповых политик Active Directory в решениях на базе дистрибутивов ALT и Sisyphus.

Современные средства управления конфигурациями ориентированы на управление состоянием компьютеров (удалённо управляемых узлов в сети), в то время как групповые политики в Active Directory ориентированы не только на динамическое управление состоянием рабочих станций и серверов, но и на применение состояния настроек пользователей, определяемых динамически в момент входа в систему. Такие конфигурации, представленные в виде декларативного набора обширного количества настроек, называются политиками.

Разделение политик и механизмов является одним из ключевых правил философии Unix [2]. В случае же с определением динамических настроек операционной системы и пользовательских настроек, под политиками подразумеваются варианты использования механизмов самой операционной системы, а не подход к их реализации, как готового решения, на основе одних и тех же механизмов ядра. В этом случае сама операционная система рассматривается уже не как итоговое решение, а как шаблон для реализации такого решения на основе заложенных в операционную систему возможностей.

В этом плане реализация групповых политик в Linux-решениях [1] требует существенной доработки механизмов, предоставляющих воз-

можность декларативной настройки политик Active Directory. Прежде всего на уровне встроенных механизмов их применения (исполняемых модулей, применяющих те или иные настройки), а также на уровне интеграции инструментов их применения в работу операционной системы.

Для применения декларативных пользовательских настроек в дистрибутивах ALT разработаны следующие механизмы для применения групповых политик:

- утилита `gpupdate` (находится в стадии разработки), заполняющий кеш GPO в каталоге `/var/cache/samba/gpo_cache`, для заданного пользователя, выгружать объединённый набор значений политик пользователя в кеш ключ/значение, запускать модуль `gpoa` («GPO Applier») для применения известных политик.

- модуль `oddjob_gpupdate`, позволяющий фиксировать вход в систему и выполнять утилиту `gpupdate` от имени администратора во время логина (предполагается, что `gpupdate` получает имя пользователя и номер его сессии из `/proc/self/sessionid` для отправки сигналов в программу «гриитер» через `dbus`);

- утилита `hreg` преобразующая pol-файлы (разбор `client side extension` пока не осуществлён) в дерево файлов в формате ключ/значение, который может применить модуль `gpoa`;

- в случае отсутствия доступа к сети ранее сформированное дерево файлов в формате ключ/значения применяется из кеша.

Кроме инструментария применения групповых политик, требуется исходный, управляемый вручную, локальный набор политик. Такой набор политик, относительно которого применяются и откатываются все остальные групповые политики. Локальный набор политик в решениях ALT также оформляется в виде дерева файлов в формате ключ/значения и подготавливается в виде отдельных пакетов под каждый вариант дистрибутива.

Использование утилиты gpupdate для применения декларативных настроек текущей машины осуществляется также, как и для пользовательских настроек, только по другим событиям — при загрузке операционный системы, а также явным запросом.

Полная интеграция механизмов применения политик на текущий момент требует:

- автоматизации встраивания модуля `pam_oddjob_gpupdate` в стек PAM-модулей, а именно в последний шаг настройки сессии в `common-login` после модулей `pam_loginuid.so` и `pam_systemd.so`;
- согласования локальных политик безопасности, настроек пакетов, по умолчанию, и ожидаемых политик в дистрибутивах;
- дополнения базовых настроек, доступных в виде применяемых политик после подключения машины к домену (например, подключение доступа по ssh через `GSSAPI` по умолчанию);
- отладки механизмов применения политик (модуля `gpoa`).

Литература

[1] ALT Linux Team, Групповые политики в решениях ALT, 2019, `https://www.altlinux.org/Групповые_политики`

[2] Eric S. Raymond, The Art of Unix Programming, 2003

Александр Боковой
Эспоо, Финляндия, Red Hat

Эволюция учета и аутентификации пользователей в Red Hat Enterprise Linux 8 и Fedora

Аннотация

В 2019 году мы празднуем 40 лет с момента появления в UNIX V7 вызовов семейства `getpwnam()`, `getgrnam()`, `getpwuid()` и `getgrgid()`, предназначенных для получения информации о зарегистрированных в системе пользователях и группах. За четыре десятка лет многое изменилось, но мы по-прежнему используем эти интерфейсы во всех POSIX-совместимых системах.

В 2019 году компания Red Hat выпустила Red Hat Enterprise Linux 8. В докладе рассматриваются изменения в системах учета и аутентификации пользователей в RHEL 8 в контексте изменений ИТ отрасли за последние четверть века.

Несмотря на существенные изменения на уровне протоколов, используемых для взаимодействия с различными системами учета пользователей, 40 лет мы используем API, семантика которого не претерпела существенных изменений. Вызовы семейства `getpwnam()`, `getgrnam()`, `getpwuid()` и `getgrgid()` можно считать одними из самых долгоживущих API в POSIX.

Чуть более четверти века назад (1993) появился интерфейс name service switch (NSS, `nsswitch.conf`), позволивший подключать различные модули, реализующие «наполнение» API `getpwnam()`, `getgrnam()`, `getpwuid()` и `getgrgid()`.

Наконец, созданный в 1995 и стандартизированный в 1997 как Single Sign-On Specification, интерфейс pluggable authentication modules (PAM) довершает набор интерфейсов, абстрагирующих аутентификацию и авторизацию на уровне операционной системы. Фактически, эти интерфейсы без изменений используются всеми POSIX-совместимыми ОС и сегодня.

Более того, фактически все сетевые протоколы, используемые в системах учета и аутентификации пользователей, были созданы и активно развивались в XX веке. Они до сих пор актуальны, но это накладывает отпечаток на реализации протоколов: проекты со свободными лицензиями, их реализующие, испытывают определенный недостаток «новой крови» — по многим причинам. За десятилетия написано множество компонент для PAM и модулей NSS, но некоторые из них уже фактически не поддерживаются: они либо заброшены своими авторами, либо потеряли актуальность из-за смены моделей использования тех или иных протоколов (NIS, Hesiod и другие), либо их внутренняя реализация не отвечает современным взглядам на проектирование системных компонент для встраивания в другие приложения (разделение привилегий, изоляция и т.д.).

С точки зрения разработчиков дистрибутивов операционных систем такие компоненты представляют проблему: отсутствие активного проекта означает сложную ситуацию с поддержкой в случае проблем безопасности. Вдобавок, не только адаптация к условиям эксплуатации в дистрибутиве, но и общее развитие проекта падает на плечи разработчиков дистрибутива. Даже если компонента используется во всех дистрибутивах GNU/Linux, зачастую совместной работы не получается — знания о реализации протокола этой компоненты не обязательно присутствуют у всех мейнтейнеров.

В 2007 году Red Hat инициировал проект System Security Services Daemon (SSSD), обеспечивающий единообразный доступ к сервисам учета, аутентификации и авторизации как локально, так и удаленно. За последнюю декаду было сделано многое для интеграции с существующими системами управления и учета пользователей (Active Directory, FreeIPA и другими). В SSSD был реализован функционал основных модулей работы с LDAP источниками, с Kerberos, добавлена поддержка авторизации с использованием смарт-карт и разнообразных токенов. В рамках интеграции с FreeIPA была разработана поддержка многофакторной аутентификации. Для FreeIPA и Active Directory добавлены средства авторизации, учитывающие специфические правила описания доступа (host-based access control в FreeIPA и Group Policies в AD). Всё это привело к дублированию функционала в рамках дистрибутива для целого ряда модулей. К тому же, во многих случаях эти модули не предполагали централизованное управление своими ресурсами и затрудняли поддержание единообразного представления систем доступа при эксплуатации внедрений с большим количеством одинаковых серверов.

В рамках Red Hat Enterprise Linux 8 была предпринята попытка «чистки» дистрибутива от устаревших компонент. Помимо самих компонент, модернизации подверглась и утилита настройки, использовавшаяся для поддержания конфигураций. За годы эксплуатации стало понятно, что на самом деле пользователи в основном реализуют ограниченый набор сценариев конфигурации и можно существенно упростить настройку и поддержку, если эти сценарии зафиксировать в готовых конфигурациях вместо модификации состояния конфигурационных файлов в любой момент времени. Последнее также позволяет упростить настройку систем в контейнерном исполнении, поскольку уменьшает размер дополнительных модификаций при создании контейнерного образа.

Как результат, модули `nss-pam-ldapd`, `pam_krb5`, `pam_pkcs11` были убраны из Red Hat Enterprise Linux 8. Утилита конфигурирования `authconfig` была заменена на `authselect`, а управление первоначальной настройкой для доменных сред унифицировано утилитой `realm`. Таким образом, Red Hat Enterprise Linux 8 поддерживает два основных варианта доступа к доменным средам: с использованием SSSD и с использованием winbindd из набора средств Samba. Для локального управления по умолчанию также используется демон SSSD, научившийся хранить информацию о локальных пользователях и группах.

Вместе с тем, сценарии использования операционных систем и предоставляемых ими сервисов расширились. Для многих задач доступ к данным о пользователях означает и доступ к свойствам, которые никогда не предоставлялись в рамках POSIX API. Например, многим приложениям требуется информация о почтовом адресе пользователя — они используют его для задач идентификации пользователя в рамках своих баз данных или для других целей. Другим приложениям необходимо уметь отображать сертификат, с помощью которого аутентифицировался пользователь, на самого пользователя. Для предоставления таких расширенных деталей о пользователях в SSSD был добавлен специальный механизм InfoPipe, прозрачно интегрирующий доступ к этой информации из всех поддерживаемых и настроенных источников, включая сложные комбинации вроде доверяемых отношений между FreeIPA и лесами Active Directory.

С целью изоляции ресурсов аутентификации еще в рамках Red Hat Enterprise Linux 7 был добавлен механизм прозрачного проксирования доступа к аутентификационным токенам в GSS-API. Этот механизм активно используется во FreeIPA и других проектах, а также для изоляции доступа к токенам аутентификации в веб-приложениях на основе Apache. В Red Hat Enterprise Linux 8 старые модули Apache (например, `mod_auth_kerb`) были убраны, их заменяет `mod_auth_gssapi`.

Аналогичная работа была проведена и с криптографическими библиотеками. Проект миграции всего свободного ПО на библиотеку NSS от Mozilla Foundation не удался, поэтому в Red Hat Enterprise Linux 8 большая часть приложения переведена на использование OpenSSL. NSS используется только в Firefox и системе организации удостоверяющего центра Dogtag, а также внутри программного токена SoftHSMv2. Остальные приложения были переписаны на OpenSSL и GnuTLS. Помимо решения проблем с унификацией, это позволит упростить сертификацию таких систем по государственным стандартам (FIPS 140-2 и подобным).

Одним из значимых изменений в Red Hat Enterprise Linux 8 стало устранение сервера LDAP от проекта OpenLDAP. Компания Red Hat изучила статистику обращений заказчиков в службу поддержки и выяснила, что существующее предложение не обеспечивает достаточный уровень поддержки в то время, как партнеры Red Hat, занимающиеся развитием OpenLDAP, могут предоставить более качественные услуги по его поддержке. В результате, было принято решение сфокусироваться на сервере LDAP из проекта 389-ds, который лежит в основе

двух активно поддерживаемых продуктов линейки Red Hat: RHEL IdM и RHDS. Аналогичное решение приняла и компания SUSE, заменившая OpenLDAP сервер в своих дистрибутивах на 389-ds и активно работающая над развитием 389-ds. При этом библиотеки OpenLDAP остались доступны и также активно развиваются.

Антон Мидюков
Прокопьевск, ООО «Базальт СПО»

Проект: mkimage-profiles
https://www.altlinux.org/Mkimage/Profiles/m-p

mkimage-profiles — гибкий инструмент сборки дистрибутивов для множества платформ

Аннотация

Необходимость сборки дистрибутивов для множества платформ стала вызовом для mkimage-profiles. С какими трудностями пришлось столкнуться при создании универсальных профилей дистрибутивов для нескольких архитектур? Как эти трудности были преодолены? Как адаптировать старый профиль под новые реалии? В докладе будут даны ответы на эти и другие немаловажные вопросы.

Почему возникли проблемы?

mkimage-profiles изначально был ориентирован на сборку дистрибутивов для PC.

При этом предполагались следующие типы целей:

1. distro — образы дисков iso с загрузчиком syslinux:
   a) Установочный дистрибутив installer;
   b) Живые диски live с возможностью установки и без;
   c) Два в одном: install + live;
2. ve/ — виртуальные окружения без ядра и загрузчика, запакованные в архивы tar, tar.gz, tar.xz;
3. vm/ — образы виртуальных машин: img, qcow2, vdi и т. д.

Изначально заложенных вариантов оказалось недостаточно.

При портировании профилей на новые платформы встали следующие задачи:

1. Необходимость упаковки архивов с ядром и загрузчиком, что в свою очередь заставило решать следующие задачи:
   a) Необходимо было вынести из tar2fs сборку initrd и подготовку конфига загрузчика;
   b) Принять решение порождать ли новый тип целей или расширить возможности одной из существующих: ve/ или vm/;
2. Использование специфичных загрузчиков для каждого устройства поставило вопрос, а как их всех охватить, сделав лишь одну сборку?
3. Для distro/ встала проблема перехода с syslinux на grub и его вариации;
4. В связи с необходимостью выпуска почти идентичных сборок не только в формате iso, но и в формате архивов и образов виртуальных машин, необходимо было выделить общую часть для них у каждой сборки;
5. Необходимо было также учесть доступность пакетов в списках для каждой поддерживаемой платформы.

Непростые решения

Наиболее сложным решением был выбор: порождать ли новый тип целей или расширить возможности одной из существующих. Было последовательно опробовано три варианта:

1. Использовать тип целей ve/ для упаковки архивов с ядром и загрузчиком. По началу выбор казался очевидным. Но вскоре стало ясно, что это создаёт некоторые неудобства. Как делать make-initrd? В отдельной feature под устройство? Так сначала и делалось, но это было неправильно, так как необходимо по максимуму выделять общее и сводить к нулю копипаст.
2. Создать новый тип целей mr/ (mashine rootfs), в которых бы явно проверялось наличие ядра, делался make-initrd. Но вскоре стало ясно, что mr/ есть не что иное, как vm/, но в виде тарбола, а потому был опробован третий вариант.
3. Адаптировать vm/, чтобы можно было создавать ещё и тарболы, а не только образы. Это позволило сократить число целей

в два раза, что сильно упростило жизнь для случая регулярных сборок и стартеркитов. Вот его и выбрали. Хотя это тоже спорное решение, так как не всегда состав архива файловой системы для одноплатного компьютера должен быть по составу такой же, как и образ для qemu. Но это решается использованием проверочного условия для `$(IMAGE_TYPE)`:

```
ifeq (,\$(filter-out qcow2 qcow2c,\$(IMAGE\_TYPE)))

...

endif
```

Остальные решения были просты и не мучительны.

Всё общее в сборках постепенно выделялось в отдельный конфиг mixin.mk. Изначально предполагалось, что цели mixin/ не должны зависеть друг от друга, но это предубеждение удалось преодолеть. И теперь эти цели можно рассматривать как кубики для получения почти одинаковых по составу сборок всех типов: distro/ ve/ vm/.

Ответ на вопрос, как сделать так, чтобы одну сборку можно было использовать для множества устройств, оказался прост. Необходимо переложить все специфические действия на пользователя. Для того, чтобы пользователю это не было делать обременительно, был написан скрипт alt-rootfs-installer. alt-rootfs-installer постепенно обрастает функционалом и скоро начнёт претендовать на место tar2fs.

Перевод сборок iso на grub осуществлён для aarch64 и ppc64le. Потребовалась правка mkimage, но это уже другая история. Предстоит ещё создание аналога feature syslinux для генерации конфига grub.

Как адаптировать старый профиль под новые реалии?

Теперь это сделать достаточно просто:

1. Выносим из профиля цели mixin/ с независимым от типа цели содержимым;
2. Специфичное для платформы огораживаем условиями например так:

```
ifeq (,\$(filter-out aarch64 armh,\$(ARCH)))

...
```

```
endif
```

3. Проходим по спискам пакетов и отмечаем архитектурно-зависимые пакеты указанием через знак «@» архитектуры, для которой он предназначен:
   grub-pc@X86
   grub-efi@X86
   grub-efi@aarch64
   grub-ieee1275@ppc64le
4. Создаём новые цели со спецификой для типа цели: vm/ или ve/

Итого: в mkimage-profiles появилась возможность делать почти одинаковые по составу сборки iso, tar, img и т. д. для нескольких архитектур.

Денис Силаков
Москва, Virtuozzo

Проект: OpenVZ `https://openvz.org/`

OpenVZ Customer Experience Program, или сбор данных о пользователях в OpenVZ 7

Аннотация

Разработчикам большинства приложений небезразлично, кто и как использует их продукт — как для «спортивного интереса», так и для определения дальнейших направлений развития. Для сбора подобной статистики нередко применяют автоматическую отсылку интересующих разработчиков данных с некоторой периодичностью; подобный подход применяется и в OpenVZ. Многие пользователи с недоверием относятся к любой попытке собрать какую-либо информацию об из системе, однако природа СПО позволяет нам открыть как сами инструменты сбора данных, так и результаты их работы. Данный доклад посвящен деталям реализации CEP в OpenVZ 7, различным аспектам юридического и морального характера, учтенными при разработке, а также пользе, которую приносит сбор данных в текущем виде.

Возможность сбора данных с серверов существует в OpenVZ уже много лет. Для предыдущей версии данные агрегируются на `https://`

stats.openvz.org/. В OpenVZ 7 вместе с кодом, принесенным из коммерческого Virtuozzo, был добавлен новый механизм сбора данных и появился новый сайт-агрегатор — https://stats7-web.openvz.org/.

Реализация сбора данных

Сбор и отправка данных осуществляются сервисом disp-helper, исходный код которого размещен на https://src.openvz.org/. Сервис состоит из движка, поочередно запускающего скрипты сбора данных. Все скрипты написаны на Python, так что можно изучать их непосредственно в установленной системе. Каждый скрипт собирает информацию о какой-то конкретной подсистеме:

- hardware.py — информация об аппаратном обеспечении;
- libvirt_stats.py — общая информация о libvirt;
- libvirt_guests.py — информация о виртуальных машинах;
- ves.py — данные о контейнерах;
- ... и так далее.

Помимо статичных данных (объема памяти, количества контейнеров и тому подобного), в OpenVZ 7 производится анализ производительности (в частности, выявляются операции, занимающие слишком много времени) и статистика использования подсистем наподобие PFCache, призванных повысить плотность контейнеров на машине.

Результаты работы каждого скрипта и финальный отчет можно найти в директории /vz/tmp/cep/output/.

По умолчанию, отсылка статистики производится раз в неделю. При этом движок запускает скрипты не одновременно, а равномерно распределяет моменты их старта на всю неделю.

Некоторые скрипты, присутствующие в исходном коде, непосредственно в OpenVZ не используются (например, license.py и backend_info.py) и включены только в коммерческих продуктах либо в комбинации OpenvVZ 7 и Virtuozzo Storage.

Приватность

Отсутствие приватной информации в отсылаемых отчетах — обязательное условие, без выполнения которого большинство пользователей откажутся от участия в сборе данных. При этом многие пользователи относят к приватным данные, формально не попадающие под

закон №152-ФЗ в России или GDPR в Европе. Например, далеко не все готовы раскрывать IP-адреса или MAC-адреса сетевых адаптеров.

Еще один камень преткновения — является ли легитимным какой-либо сбор данных о внутренностях виртуальных окружений. Ведь данные там принадлежат не провайдеру, а конечным пользователям, поэтому некоторые системные администраторы предпочитают совсем отключать анализ гостевых систем.

Во всех подобных случаях мы идем навстречу пользователям и либо совсем исключаем проблемные сведения из отчета, либо даем возможность их опционального удаления перед отсылкой.

Публичный веб-инетрфейс

По сравнению с OpenVZ 6, в новой версии продукта в CEP собирается гораздо больше информации — как минимум потому, что наряду с контейнерами в продукте поддерживаются виртуальные машины. Как следствие, формат отображения всей собранной статистики на одной странице стал слишком тяжеловесным и в новой версии веб-интерфейса данные разделены на секции. По умолчанию, статистика учитывает только отчеты, пришедшие за последние две недели.

Для большинства параметров доступны графики их изменений со временем. Также есть глобальный фильтр, позволяющий изучать статистику по состоянию на заданную дату.

На данный момент для OpenVZ 7 собрано уже достаточно много данных, позволяющих делать выводы об использовании продукта и о наиболее актуальных направлениях развития. Например:

- виртуальные машины используются, но далеко не так активно, как контейнеры (и как нам хотелось бы);
- наиболее популярными гостевыми системами, как и раньше, являются CentOS 7, Ubuntu и Debian;
- большинство машин имеют от силы несколько десятков контейнеров, но есть и такие, на которых размещено несколько сотен виртуальных окружений;
- ... и многое другое.

Антон Бондарев
Санкт-Петербург, Embox

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

Embox — ОС РВ позволяющая запускать Linux ПО на микроконтроллерах

Аннотация

Доклад посвящен опыту проекта Embox по использованию "тяжелого" прикладного обеспечения, например Qt, на микроконтроллерах. В докладе рассмотрены: преимущества подобного подхода, особенности микроконтроллеров и трудности связанные с использованием данного ПО на них, а также подход проекта Embox позволяющие не только преодолеть данные трудности, но и существенно снизить затраты на их преодоление.

Введение

На сегодняшний день от систем управления ожидают все более развитой функциональности. Характеристики аппаратуры, в том числе микроконтроллеров значительно выросли, что позволяет должным образом эту функциональность обеспечивать. Но программное обеспечение достаточно сильно отстает. В силу того, что исторически подобное ПО в целях оптимизации писалось под конкретную задачу. А переиспользование кода, которое является движущей силой развития функциональности ПО для универсальных систем, было достаточно ограниченным. С увеличением же функциональных требований к системам управления, уже невозможно создавать всю функциональность с нуля, игнорируя, уже разработанный и отлаженный код обеспечивающий данную функциональность для универсальных систем.

Особенности микроконтроллеров

Но оригинальное десктопное ПО имеет ряд особенностей, которые затрудняют его применение на небольший апаратных системах, в частности микроконтроллерах. Прежде всего это отсутствие в микроконтроллерах блока управления памятью (MMU). В микроконтроллерах используется блок защиты памяти (MPU) который не подразуме-

вает режим трансляции адресов. Соответственно возникают проблемы поскольку десктопное ПО традиционно запускается в отдельном адресном пространстве. Кроме того в десктопном ПО используется системный вызов fork() для создания процесса, являющегося экземпляром какого то прикладного приложения.

Вторым существенным ограничением является сильно меньшее количество ресурсов памяти доступных в системе. Универсальные системы разрабатывались по сути дела в условиях когда память доступная приложению ограничена только разрядностью адресного пространства. Возьмем например 32 разрядную систему, по сути дела приложение может использовать до трех гигабайт (а то и больше) памяти, ведь за даже если в системе нет такого количества физической памяти, с помощью механизма свопа, она может быть доступна, хотя это и значительно снижает производительность. В микроконтроллерах же с одной стороны память находится прямо на кристалле, а с другой стороны ее количество существенно меньше чем в десктопных системах.

Кроме того что в микроконтроллерах существенно меньше памяти, существует четкое разделение по типам памяти. Первое по типу доступа: на только для чтения ROM (read only memory) и на память с произвольным доступом то есть и на чтение и на запись RAM (random access memory). По типу расположения: память доступная прямо на кристалле и память доступная по внешней шине.

Выделение памяти только для чтения, оправданно с точки зрения увеличения производительности. Поскольку гарвардская архитектура применяемая в микроконтроллерах разделяет память для команд и память для данных, доступ происходит по разным шинам и следовательно общая пропускная способность памяти увеличивается. Но поскольку эта память не подразумевает произвольную запись, то загрузить код в эту память, как принято в десктопных приложениях, сложно. Соответственно еще одной важной особенностью десктопных систем является то что они не ориентированы на так называемых XiP (eXecute In Place) метод, а на обычную загрузку всех секций, включая, секцию кода, в основную память.

Опыт Embox

Embox — конфигурируемая ОС РВ позволяющая запускать прикладные приложения и библиотеки разработанные для Linux на мик-

роконтроллерах. Это достигается с помощью нескольких особенностей Embox.

Первое. Embox использует систему сборки Mybuild с собственным языком описания модулей и системы. Mybuild анализирует зависимости и генерирует различные артефакты, в том числе заголовочные файлы, линкер скрипты и Makefile -ы. Все это позволяет существенно уменьшить количество требуемой памяти и вообще не включать те части которые не используются.

Второе. Embox поддерживает POSIX. Это собственно позволяет использовать прикладное ПО фактически без изменений.

Третье. Статическая линковка используется чтобы избежать проблем с единым адресным пространством, уменьшить размер образа и включать только требуемые части. На разных стадиях собираются разные части системы, но окончательно все (ядро, все подсистемы, библиотеки и приложения) линкуется в единый образ. В нем есть вся информация обо всех символах в образе. И хотя в образе может быть много приложений со стандартными точками входа int main(..), но в результирующем образе эти символы будут заменены на другие, хранящие полное имя модуля и символ main .

Четвертое. Нам пришлось реализовать рантайм для C++, поскольку многие популярные приложения и библиотеки используют плюсы.

Пятое. Мы в систему сборки добавили возможность использовать ./configure; make; make install

Предварительно скачав откуда-то исходники и наложив необходимые патч.

Заключение

В результате Embox позволяет достаточно легко осуществить перенос ПО из Linux и запустить его на микроконтроллерах. Причем отладку можно произвести с помощью эмулятора qemu, а затем сконфигурировать образ под конкретную аппаратную платформу, в том числе микроконтроллер, что также существенно упрощает разработку конечной системы.

Применение десктопного ПО на микроконтроллерах дает выигрыш ведь с одной стороны, можно использовать более дешевую, надежную и менее потребляющую аппаратную платформу, а с другой

стороны существенно снижаются затраты на разработку функционального ПО.

Литература

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

Виталий Минко
Красногорск, ОАО «ИнфоТеКС»
Проект: Dscuss http://dscuss.org/

Dscuss — одноранговая сеть для публичного общения

Аннотация

Распространённые системы публичного общения (такие как Internet-форумы и социальные сети) не обеспечивают идентичных прав для всех собеседников. Владелец ресурса и модераторы всегда обладают расширенными правами. Злоупотребление этими правами может в итоге привести к расколу и вырождению сообщества. Для решения этой проблемы создана одноранговая сеть *Dscuss* [1].

Введение

Dscuss — это сеть для публичного общения, ключевыми свойствами которой являются равноправие всех участников сообщества и децентрализованный способ модерации сообщений. Каждому пользователю предоставляется возможность модерировать сообщения других пользователей. И каждый пользователь свободен выбирать себе модераторов, которых пожелает.

Для пояснения предпосылок к созданию новой сети и её ключевых свойств введена классификация средств публичного общения [2].

Принципы функционирования

Протокол Dscuss построен на объектах трёх типов: пользователи, сообщения и операции. Пользователи выполняют роль учётных записей и идентифицируются публичным ключом. Сообщения представ-

ляют информацию, опубликованную от имени пользователя в определённой рубрике. Пользователи подписываются на заинтересовавшие их рубрики и получают сообщения только из этих рубрик. Операции представляют определённое действие, выполненное от имени пользователя над объектами (например, блокировка пользователя или удаление сообщения). Объекты всех типов идентифицируются в общем адресном пространстве. Все объекты, попадающие под интересы пользователя, хранятся на локальной машине самого пользователя.

Безопасность

С точки зрения безопасности, Dscuss спроектирована для обеспечения следующих свойств данных: доступность, целостность и подлинность. Предусмотрена защита от *SPAM* и flood-атак. Модель нарушителя, принятая в Dscuss формализована [3].

Стоит отметить, что Dscuss не предназначен для обеспечения анонимности пользователей. Но пользователи могут добиться анонимности в сети с использованием сторонних средств (таких как *Tor*).

Протокол взаимодействия

Два узла сети связываются одним мультиплексируемым TCP-соединением, которое используется для передачи всех данных между узлами. Взаимодействие узлов происходит синхронно — отправитель пакета ждет ответа перед отправкой другого пакета. Пакеты передаются в текстовом виде (в формате *JSON*). Алгоритм взаимодействия узлов в сети детально описан в [4].

Реализация

На данный момент существует единственная реализация протокола, исходный код которой доступен под лицензией *GPLv3* [5]. Реализация выполнена на языке Go в виде набора пакетов. Архитектура решения описана в [6]. Реализация протокола имеет три сторонних зависимости: драйвер *SQLite*, реализация *scrypt* [7] и *Kademila DHT* [8].

Текущая стабильная версия реализации — *proof-of-concept*. В ней реализованы самые базовые функции: регистрация пользователей, соединение узлов, подписки на рубрики, публикация сообщений в рубриках и выполнение операций удаления сообщений и блокировки

пользователей. Для отладки и автоматического тестирования разработан клиент с интерфейсом командной строки. Он также позволяет на практике ознакомиться с концепцией сети.

Версия 0.1 находится в разработке. В ней уже реализована синхронизация данных между узлами, улучшена связанность сети (через поиск узлов в *DHT* [9]) и разработан клиент сети Dscuss с Web-интерфейсом.

Литература

[1] Dscuss — P2P network for public discussions, `http://dscuss.org/`

[2] Classification of the systems for public discussion, `http://vminko.org/dscuss/classification`

[3] Adversary model, `http://vminko.org/dscuss/adversary`

[4] Protocol description, `http://vminko.org/dscuss/protocol`

[5] dscuss — the master Dscuss repository, `http://dscuss.org/cgit/dscuss/`

[6] Dscuss service architecture, `http://vminko.org/dscuss/architecture`

[7] scrypt — Википедия, `https://ru.wikipedia.org/wiki/Scrypt`

[8] Kademlia — Википедия, `https://ru.wikipedia.org/wiki/Kademlia`

[9] Распределённая хеш-таблица — Википедия, `https://ru.wikipedia.org/wiki/`Распределённая_хеш-таблица

Звягинцев Максим Андреевич
Тула, ТГПУ им. Толстого

Улучшение качества открытого программного обеспечения с помощью инструментов анализа кода

Аннотация

Одним из способов внести свой вклад в развитие открытого программного обеспечения, является поиск и исправление различных программных ошибок и потенциальных уязвимостей. Достаточно просто, это можно осуществить, используя статические анализаторы кода. Это можно делать даже в том случае, если вы ещё мало знакомы с устройством работы проекта, но хотите внести вклад в его улучшение. Поговорим о методологии анализа исходного кода, бесплатных инструментах и автоматизации процессов проверки.

Представим достаточно большой проект на GitHub. Есть множество желающих поучаствовать в разработке, однако тестировщиков всегда недостаточно. Это серьезная проблема, так как без чётко выстроенной системы тестирования серьезные баги могут попасть в релиз. Для того, чтобы исправить эту ситуацию стоит рассмотреть различные способы автоматизации, такие как unit тесты, динамический и статический анализ и системы непрерывной интеграции.

Предлагаю рассмотреть статические и динамические анализаторы. Они отлично дополняют друг друга, так как позволяют одновременно выявить ошибки в исходном коде и проанализировать работоспособность во время выполнения.

На первый взгляд может показаться, что эти типы анализаторов достаточно похожи, однако это не так. Статический анализ работает с исходным кодом. Он занимается поиском различных аномалий, опечаток, потенциальных уязвимостей и многих других проблем, которые могут не проявлять себя во время работы программного обеспечения, но быть бомбой замедленного действия. Динамический анализ напротив смотрит не на исходный код, а на выполнение программы. Такой подход позволяет обнаружить такие серьезные проблемы, как например утечки памяти, выходы за границы массивов или гонки потоков. Разумеется, это не единственное, что умеют динамические анализаторы. Помимо поиска множества других проблем, они могут собрать различные метрики, например время выполнения частей программы и отдельных функций или покрытие кода тестами.

Вернемся к статическому анализу. В первый раз мы, конечно, проверим весь проект. И скорее всего получим внушительный список ошибок, так как анализатор ещё не настроен. Для такой ситуации существует подавление ложных срабатываний. Например, директива `#pragma` позволяет убрать ложные предупреждения компилятора. Да, компиляторы зачастую немного занимаются статическим анализом, однако они могут обнаружить лишь самые типовые ошибки из-за сложности реализации полноценного статического анализа.

Итак, мы разобрались со всеми предупреждениями, исправив ошибки и подавив ложные срабатывания. Однако, новые изменения кода могут принести в проект новые ошибки и новые потенциальные уязвимости, поэтому разумно запускать статический анализ достаточно часто, чтобы не накапливать технический долг.

Первое, что приходит в голову — делать проверки после каждого изменения исходного кода. Конечно, это сработает и будет выдавать все необходимые предупреждения, однако есть одна серьезная проблема — время анализа. Проверка большого проекта может быть весьма долгой, поэтому нужно каким-то образом ускорить её.

Например, в анализаторе PVS-Studio есть режим инкрементального анализа. Он позволяет проверить те файлы, которые были изменены с момента последней проверки. Это безусловно хороший вариант для локального анализа, однако это не подходит для ситуации, когда множество разработчиков делают вклад в общую кодовую базу.

Решением этой проблемы будет проверка кода при pull request. Например, у нас есть две ветки, которые нужно слить в одну. Для удобства назовем «главную» ветку master, а «ответвление» — hotfix. Проблема в том, что hotfix может принести в общую кодовую базу ошибки, так как никто не застрахован от опечаток. Поэтому было бы разумно проверять перед слиянием разницу между ветками. Это можно реализовать, например, при помощи связки сервиса непрерывной интеграции и анализаторов кода.

Панченко Иван Евгеньевич
Москва, Postgres Professional
https://postgrespro.ru/

Модель развития PostgreSQL как международного свободного продукта и сообщества

Аннотация

PostgreSQL является одним из наиболее успешных свободных проектов мирового масштаба. Он прошел путь от академической разработки до промышленной СУБД. Соответственно менялось и сообщество. Может ли PostgreSQL быть образцом для других? Как на развитие PostgreSQL влияет его лицензия? Мы проанализируем основные этапы истории развития PostgreSQL, как сообщества, так и продукта, и выделим положительно и отрицательно влияющие на их развитие факторы, и поймём, почему PostgreSQL именно такой.

Postgres возник в 1986 году как научная разработка профессора университета в Беркли М.Стоунбрейкера. В своей знаменитой статье [1] он закладывает основные концепции этой СУБД, в том числе расширяемость и версионный механизм обеспечения транзакционных свойств (ACID). Postgres того времени сильно отличается от современного. Например, в нём не было поддержки языка SQL, который в то время уже существовал [2], но только готовился к стандартизации [3].

В Open Source Postgres вышел в 1995 году под именем Postgres95, затем в 1996 году в связи с появлением поддержки языка SQL установилось название PostgreSQL [4,5]. Сформировалось сообщество разработчиков, членом которого мог быть любой желающий, и любой из них имел доступ в CVS-репозиторий. Рост объема кода, количества коммитов и необходимость эффективной работы с многочисленными ветвями кода привели к переходу на GIT в 2010 году [6,7]. Возможность коммита в репозиторий получило ограниченное количество наиболее авторитетных разработчиков — коммитеров [8]. Это выражает рост требований к качеству кода, обусловленный применением PostgreSQL во всё более ответственных задачах.

PostgreSQL стал использоваться в крупном бизнесе примерно с 2000 года. Вначале это были интернет-компании (Rambler, Skype, и др).

Приблизительно в 2008 году начали говорить о том, что PostgreSQL повернулся лицом к промышленным заказчикам (Enterprise-сектору). Это связано с появлением потоковой репликации, дающей возможность организовывать отказоустойчивые кластеры, и с существенным улучшением поддержки Microsoft Windows.

В наше время PostgreSQL и его производные используются в крупнейших БД России и мира. Соответственно величине и серьезности пользователей PostgreSQL, стало меняться и сообщество разработчиков. В 2004 году была создана первая компания, специализирующаяся на обслуживании и развитии PostgreSQL — британская 2nd Quadrant. В последующем году была основана американская Enterprise DB. Сейчас в мире действует 4 основных центра разработки PostgreSQL, которые способны реализовывать крупные проекты. Таким образом, от сообщества гиков-индивидуалов основной вес переместился к сообществу сотрудников компаний, получающих деньги за свою работу. Компании более эффективно представляют интересы крупных заказчиков, и это стимулирует разработку функций, нужных таким заказчикам, дающих высокую производительность, отказоустойчивость, простоту администрирования, и т. д.

Важную роль в развитии PostgreSQL играет его лицензия [9], разрешающая создание производных, в т.ч. коммерческих, закрытых продуктов. Парадоксально, но это не приводит к утечке идей из Open source. Наоборот, из-за того, коммерческие производные продукты (форки) нуждаются в постоянном, неизбежно трудоёмком мёрдже изменений из ванильного PostgreSQL, их производители становятся заинтересованы и в обратном потоке изменений. Таким образом идеи и их реализации, апробированные компаниями в коммерческих форках, возвращаются в open source. Это формирует устойчивую ситуацию взаимовыгодного сотрудничества коммерческих компаний и open source сообщества. Эта ситуация может оказаться более долговечной, чем отдельные коммерческие компании или открытые продукты [10].

Литература

[1] Stonebraker, M.; Rowe, L. A. (1986). The design of POSTGRES. ACM SIGMOD Record. 15 (2): 340. doi: `https://en.wikipedia.org/wiki/Digital_object_identifier` `https://doi.org/10.1145/16856.16888`.

[2] Chamberlin, Donald (2012). Early History of SQL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6359709`. IEEE Annals of the

History of Computing. 34 (4): 78–82. doi:https://doi.org/10.1109/MAHC.2012.61

[3] Database Language SQL/ANSI X3 135 (1986)

[4] История PostgreSQL. https://postgrespro.ru/docs/postgresql/11/history

[5] История PostgreSQL. https://www.2ndquadrant.com/en/postgresql/postgresql-story/

[6] Postgres: Hello GIT, goodbye CVS. https://www.endpoint.com/blog/2010/01/28/postgres-hello-git-goodbye-cvs

[7] Hagander, M. PostgreSQL — now on git. https://blog.hagander.net/postgresql-now-on-git-175/

[8] Список коммитеров PostgreSQL https://wiki.postgresql.org/wiki/Committers

[9] Лицензия PostgreSQL. https://www.postgresql.org/about/licence/

[10] Momjian, B. Will PostgreSQL live forever? http://momjian.us/main/writings/pgsql/forever.pdf

Селезнев Владимир
Москва, ALT Linux Team
Проект: Sisyphus

Бранчи и пакеты в Альте: от backport policy к disttag

Аннотация

Бранчи — репозитории пакетов программного обеспечения и сопутствующего материала. На основе бранчей создаются дистрибутивы Альта. Бранчи создаются путём форка Сизифа, нестабильного репозитория, или другого бранча. Среди задач, связанных с бранчами, можно выделить обновление версий пакетов в нём (и, как частный случай, бэкпортирование пакетов из других бранчей), и возможность обновления установленной системы с одного бранча на другой.

Пакеты

Пакеты делятся на два типа: исходные и бинарные. Исходные пакеты содержат в себе исходные тексты и материалы, патчи и дополнительные файлы и инструкции сборки их в бинарные пакеты. Бинарные пакеты как правило содержат уже откомпилированные программы, библиотеки и данные, и предназначены для установки на целевые системы. У каждого пакета есть уникальное *имя*, обычно включающее в себя апстримное[1] название проекта, *версия* (совпадает с версией исходного проекта) и *релиз*, обозначающий ревизию сборки данного проекта данной версии. Обычно релиз изменяют, когда надо произвести какие-то изменения в пакете этой же версии: исправить ошибку сборки или самого пакета, закрыть уязвимость и т.д.. Каждый пакет можно однозначно идентифицировать по этой твойке: %name-%version-%release[2].

Пакеты и бранчи

В процессе жизни бранча пакеты в нём могут обновляться сопровождающими пакетов или бранча, значение версии обновлённого пакета больше предыдущего (для простоты будем называть версией пакета связку {%epoch:}%version-%release, где %version имеет приоритет над %release, а %epoch над %version). В штатном режиме пакетный менеджер может обновлять пакеты на целевой системе, если кандидаты на обновления имеют более высокую версию, чем установленные.

Для решения задачи обновления установленной системы с одного бранча на более новый был введён запрет на сборку в нижележащие бранчи пакетов с версией, большей чем в вышележащие, а для задачи бэкпортирования пакетов из вышестоящих бранчей был введён backport policy, который справлялся с ней, пока бранчей было немного, и они были линейно упорядочены.

Современные проблемы бранчей

Со временем количество бранчей увеличилось, а также исчез линейный порядок. Помимо этого также существовала потребность со-

[1] Апстрим — команда или автор, предоставляющие оригинальные исходники
[2] Иногда перед версией может находиться *эпоха пакета*: суперверсия пакета

бирать пакеты из одних их тех же исходников в разные бранчи. Регулярно возникали случаи, когда необходимо было пересобрать пакет без фактического изменения исходников, что было невозможно ввиду требования уникальности %name-%epoch:%version-%release, однозначно определявшей сборку пакета, из-за чего приходилось увеличивать релиз. Различными участниками команды придумывались обходные пути для решения этих задач, которые плохо влияли на воспроизводимость сборки пакетов.

Строгие зависимости пакетов нового вида

Необходимо было решить задачу сборки бинарных пакетов в разные бранчи из одних и тех же исходных пакетов, а также пересборки пакетов без увеличения релиза. В качестве системного решения было предложено использовать более строгие зависимости пакетов с использованием build-id для различения бинарных сборок пакетов на основе хэшей от нефиктивной информации о пакетах в дополение к имени и версии пакета (через дополнительные предоставляемые и требуемые зависимости вида *.hash-%name-%version-%release*). Но от варианта с хэшом отказались из-за того, что он несовместим с возможностью сборок noarch- и arch-подпакетов из одного исходного пакета.

Вместо хэша было решено использовать информацию о целевом бранче и о сборочном задании, в котором собирется пакет. Эта информация для наглядности стала записываться в rpm тег *disttag'ов*. Предполагалось, что пакет будет предоставлять строгую зависимость вида *.disttag-%name-%version-%release*, как в схеме с хэшом, только со значением disttag'а на месте хэша. Но после её внедрения оказалось, что из-за особенностей алгоритма обработки apt'ом виртуальных пакетов начали возникать ошибки установки и обновления пакетов. Нужно было срочно переделывать схему на другую, лишённую такого недостатка.

В итоге выбор был сделан на зависимости, представляющие расширение традиционных для rpm-пакетов зависимостей вида

`%name = {%epoch:}%version-%release`

до

`%name = {%epoch:}%version-%release:%disttag`.

Предполагалось, что это позволит apt'у обрабатывать зависимости такого вида не как виртуальные пакеты. Ожидалось, что у пакетного

менеджера rpm предыдущих версий, которые не умеют обрабатывать зависимости такого виду, могут возникнуть проблемы с их обработкой, и наиболее ярко они проявились в пакетах, у которых есть конфликты на свои подпакеты отличных версий:

```
Conflicts: foo > %{EVR}
Conflicts: foo < %{EVR}
```

Решением всех проблем являлось обновление rpm на целевой системе.

Оставалось решить проблему обновления на новый бранч. Т.к. линейный порядок бранчей утрачен, решение о том, пакеты из какого бранча свежее, переложен на системного администратора. Рассматривались варианты с правкой apt_preferences и выставлением на нужный бранч высокого Pin-Priority, но особенности алгоритма вычисления весов в apt делали такой вариант не надёжным. В конце концов был введён новый макрос rpm %_priority_distbranch, значение которого обозначает наиболее приоритетный бранч для установки пакетов.

Ссылки

1. Backport policy
   `https://www.altlinux.org/Backports_Policy`
2. Binary package identity change
   `https://www.altlinux.org/Binary_package_identity_change`
3. Строгие зависимости и ошибки установки
   `https://bugzilla.altlinux.org/35580`
4. Transition to disttag
   `https://www.altlinux.org/Transition_to_disttag`

Волков А. А., Иванюк А. В., Черствов Т. В.
Москва, Московский Технический Университет Связи и Информатики

Проект: Разработка типового отечественного модуля изучения интернет-технологий

Обучение навыкам внедрения отечественных технических средств в национальную ИКТ-инфраструктуру

Существующая организация учебного процесса в ВУЗах не способствует решению важнейших задач подготовки специалистов в сфере развития национальной ИКТ — инфраструктуры с использованием отечественных технологий, а также обеспечения безопасности и доверия при её использовании. Такой мощный ресурс, как образование, используется недостаточно эффективно и целенаправленно для содействия решению задач обеспечения технологической независимости и информационной безопасности Российской Федерации. В докладе рассматривается опыт разработки и внедрения в образовательный процесс базовой кафедры Ассоциации документальной электросвязи в МТУСИ типового отечественного модуля изучения интернет-технологий. Ключевой особенностью модуля является использование при его создании российских программно — аппаратных средств и отечественной среды разработки.

- Несмотря на политику импортозамещения, продолжается повсеместное внедрение в учебный процесс продукции иностранных вендоров.
- Образование используется недостаточно эффективно и целенаправленно для содействия решению задач обеспечения технологической независимости и информационной безопасности Российской Федерации.
- Необходима подготовка специалистов в области интернет-технологий, обладающих компетенциями в разработке и использовании отечественных программных и аппаратных средств.
- Важным направлением решения этой проблемы является создание готовых унифицированных решений, которые могут быть

использованы при формировании и реализации образовательных программ. Руководствуясь этими соображениями, на базовой кафедре АДЭ в МТУСИ разработан и внедрен в образовательный процесс типовой отечественный модуль изучения интернет-технологий, который стал основой для формирования профессиональных знаний и компетенций в области построения, развития и управления сетью Интернет. Ключевой особенностью модуля является использование при его создании российских программно-аппаратных средств и отечественной среды разработки.

- Реализация проекта позволяет совместить обучение основам технологий сети Интернет с изучением возможностей отечественных технических средств и подготовкой кадров для их разработки, внедрения и использования.

Иван Волчецкий, Дмитрий Костюк, Павел Луцюк, Юрий Сойко
Брест, Брестский государственный технический университет
http://ostimeline.org

Построение документации с живыми иллюстрациями на основе встроенных виртуальных машин

Аннотация

Рассматривается подход к встраиванию демонстрационных версий программного обеспечения в документ для повышения наглядности и интерактивности. Приведены два варианта архитектуры, реализующей данный подход: проброс окон виртуальных машин в веб-документы с отрисовкой на canvas HTML5, и реализация доступа к виртуальной машине через плагин мультимедийной системы GStreamer, с расширением формата образов системы виртуализации QEMU для упрощения интеграции гостевых образов.

Встраивание в электронный документ фреймов, содержащих «живую» демонстрацию описываемого в документе программного обеспечения, лежит в русле увеличения медийной насыщенности и интерактивности обучающих и/или информационных материалов, что в свою

очередь эффективно сказывается на наглядности и усваиваемости материала. К сожалению, хотя вычислительные ресурсы современных компьютеров часто позволяют использовать подобное совмещение, существующие системы виртуализации оказались не готовы к подобному применению, и в результате в информационных материалах можно встретить лишь фрагментарные элементы «живой» демонстрации, а в основном — более традиционные мультимедийные средства: статическую графику, анимацию и видео.

Наиболее приближен к решению данной задачи ряд веб-проектов, демонстрирующих операционные системы и программное обеспечение прошлых лет на экране, встроенном в Интернет-страницу. Характерный пример – `http://oldweb.today`, посвященный истории старинных веб-браузеров и построенный на основе систем контейнерной виртуализации. Также существует ряд проектов, использующих для демонстрации старинного ПО для DOS эмулятор этой операционной системы, написанный на JavaScript. Однако все проекты подобного рода разработаны как специализированные системы, и несмотря на открытые коды, применение их наработок в качестве универсальной платформы для интеграции виртуализованного ПО в электронные документы по меньшей мере затруднено.

Аналогичная задача решалась нами в двух проектах, находящихся на разных стадиях завершенности:

- в системе демонстрации истории развития (таймлайна) графических интерфейсов пользователя на основе HTML-технологий – ostimeline;
- в программном решении для интеграции виртуальных машин в мультимедийный фреймворк GStreamer.

При этом обе системы получили сходные особенности. И одна и другая основаны на системе виртуализации QEMU, обе используют взаимодействие с системой виртуализации через сокет.

Система демонстрации виртуализованного ПО на основе HTML-технологий

Архитектура данной системы представлена на рис. 1).

Техническая инфраструктура проекта включает следующие компоненты:

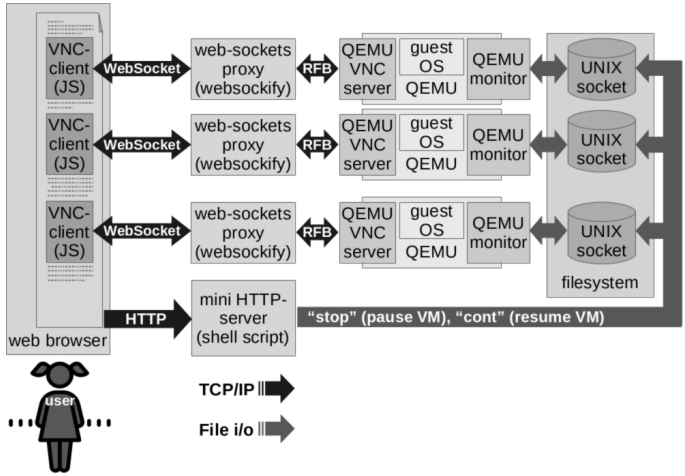

Рис. 1: Система на основе HTML-технологий

- система виртуализации QEMU (для гарантированной совместимости с ранее сохраненными снапшотами, мы используем задание архитектуры дополнительными параметрами строки запуска и локально собранный QEMU конкретных версий);
- клиент удалённого доступа на JavaScript, соединяющийся с виртуальной машиной по протоколу RFB (VNC) и выводящий изображение на canvas HTML5 (используется проект noVNC в связке с прокси websockify, выполняющим преобразование между RFB и WebSocket);
- JavaScript-фреймворк, отвечающий за отображение страниц документа и навигацию между ними (TimelineSetter для интерактивного таймлайна, reveal.js для презентационных материалов);
- специализированный HTTP-сервер в виде скрипта на shell, выполняющий приостановку и снятие с паузы виртуальных машин по мере перелистывания пользователем страниц информационного контента (реагирует на скрипт, привязанный к HTML-страницам).

Для развертывания и запуска ostimeline предусмотрена система скриптов, выполняющих скачивание и установку локально-использу-

емых зависимостей, включая требуемые версии QEMU, а также пересборку контента на основе структуры подкаталогов с образами виртуальных машин и страницами информационного контента.

На базе ostimeline развернуты демонстрационные материалы по истории графических настольных и мобильных ОС, виджет-тулкитов, дистрибутивов GNU/Linux, а также находящийся в активной разработке интерактивный обзор истории текстовых процессоров. Также одним из позитивных побочных эффектов, достигнутых при разработке, было включение в апстрим QEMU интеграции курсора мыши для старинных архитектур, не поддерживающих шину USB (на текущий момент такая опция не реализована ни одной другой системой виртуализации).

Система запуска виртуальных машин средствами GStreamer

Среди недостатков предыдущего подхода можно отметить отсутствие монолитности: контент с виртуализованными окружениями по сути представляет собой сложную систему, функционирующую как цельный интерактивный документ благодаря набору скриптов и самостоятельных программных компонентов. Однако ее развертывание и перенос на другие системы достаточно сложны, чтобы воспринимать такой «документ» как самостоятельную единицу, в сравнении с традиционными документами, содержащими нимации и видео.

Для преодоления данного недостатка был применен альтернативный подход, позволяющий трактовать образ виртуальной машины с гостевой ОС как «мультимедийный файл», проигрываемый «кодеком», использующим систему виртуализации (QEMU). На первый взгляд это решение может показаться неочевидным; однако интерактивные элементы присутствуют в медиа-файлах в течение долгого времени (например, меню DVD). Поэтому в архитектуру существующих систем воспроизведения видео заложены возможности в том или ином виде реагировать на действия пользователя.

Экспериментальная реализация была построена нами в виде плагина мультимедийного фреймворка GStreamer (рис. 2).

Разработанный плагин выполняет следующие функции:

- вывод изображения экрана гостевой ОС в медиа-плеер;
- подключение к QEMU через сокет;

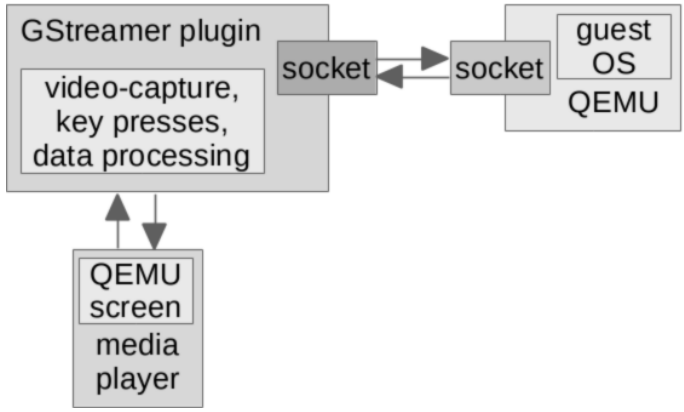

Рис. 2: Система на основе GStreamer

- регистрацию событий (движений мыши и нажатий клавиш) пользователя и передачу их в монитор QEMU.

При этом формат образов QEMU Qcow2 регистрируется в системе в качестве еще одного типа MIME (рис. 3).

Дополнительную сложность представляет хранение строки запуска виртуальной машины, которая в предыдущей реализации сохранялась в виде отдельных скриптов запуска. Очевидно, что к мультимедийному файлу скрипты запуска прилагаться не могут. Задача была решена применением собственного расширения формата образов QEMU.

Файл образа Qcow2 организован в виде кластеров фиксированного размера, как для мета-данных образа, так и для данных гостевой системы (разделов диска и снапшотов). В первом кластере находится файловый заголовок, содержащий после фиксированной части секцию расширений, в конец которой и было добавлено собственное расширение, хранящее строку запуска. Помимо плагина GStreamer, была также разработана служебная утилита для сохранения (обновления) строки запуска в образе и просмотра, при ее наличии.

Текущая реализация плагина является экспериментальной; в частности, на данный момент не решена проблема поддержки формата стандартными медиа-плеерами (без их перекомпиляции), а также

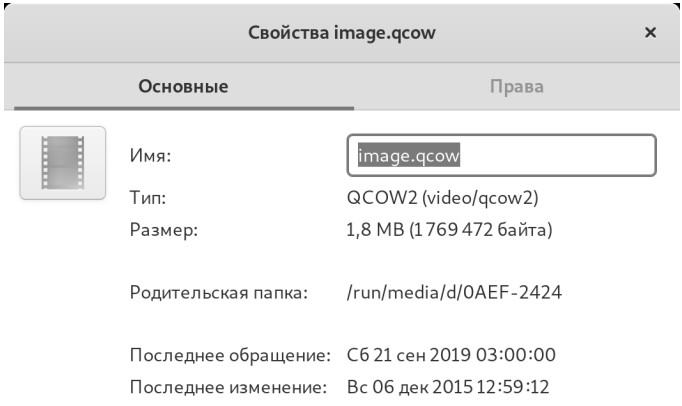

Рис. 3: Интерпретация образов виртуальной машины в качестве видео-контента

остается открытым вопрос включения Qcow2 в мультимедийные контейнеры (mkv).

В. В. Буслюк, Д. А. Костюк, Д. И. Кульбеда, А. А. Маркина, Н. Н. Терешкевич
Брест, Брестский государственный технический университет

«Умное зеркало» с функциями распознавания пользователя и персонализации контента на базе GNU/Linux

Аннотация
Рассмотрен проект встраиваемого в зеркало информационного табло на базе Raspberry-подобных одноплатных систем и GNU/Linux.

Представлена архитектура системы, персональная агрегация контента, а также средства идентификации пользователей на основе распознавания лиц и радио-меток ближнего действия. Рассмотрены варианты конфигурирования устройства с помощью мгновенных сообщений и QR-кодов.

Интеллектуальные информационные устройства — агрегаторы информационного контента с функцией автоматической идентификации пользователей — находятся в рамках технологии персональных цифровых помощников, которая получила дополнительное ускорение в последние годы, выйдя за пределы более традиционных наладонных компьютеров и смартфонов, с появлением стационарных домашних «виртуальных ассистентов» типа Amazon Alexa [1].

Основой предлагаемой разработки является встраивание такого виртуального ассистента в элементы интерьера, и идентификация пользователя, взаимодействующего с данными элементами интерьера, для выдачи ему персонализованных визуальных оповещений.

В качестве элемента интерьера для встраивания системы предлагается зеркало Гезелла [2]. Зеркала Гезелла в действительности представляют собой стёкла, покрытые тонким слоем металла — так, что часть падающего на поверхность стекла света отражается, а часть проходит насквозь. При этом насквозь свет проходит в обоих направлениях. Принцип действия зеркала и пример использования для системы вывода информации представлен на рисунке 1.

Использование первых полупрозрачных стёкол относят на счёт американского психолога А. Л. Гезелла. Впоследствии усовершенствованные полупрозрачные зеркальные стёкла получили его имя. Такие стёкла часто используются архитекторами в конструкции новых зданий. Также они активно применяются для оборудования переговорных комнат, комнат служб безопасности и др.

В основе устройства в нашем случае находится одноплатная система Raspberry Pi 3 с 7-дюймовым дисплеем, входившим в официальную поставку. Альтернативой является подключение большего дисплея по интерфейсу HDMI; однако, поскольку дисплей закрепляется с обратной стороны зеркала и обеспечивает собственно вывод информации для пользователя, ему необходим достаточный запас яркости, и наш опыт попыток применения для этой цели более дешевых дисплеев говорит о необходимости осторожного отношения к выбору данного компонента.

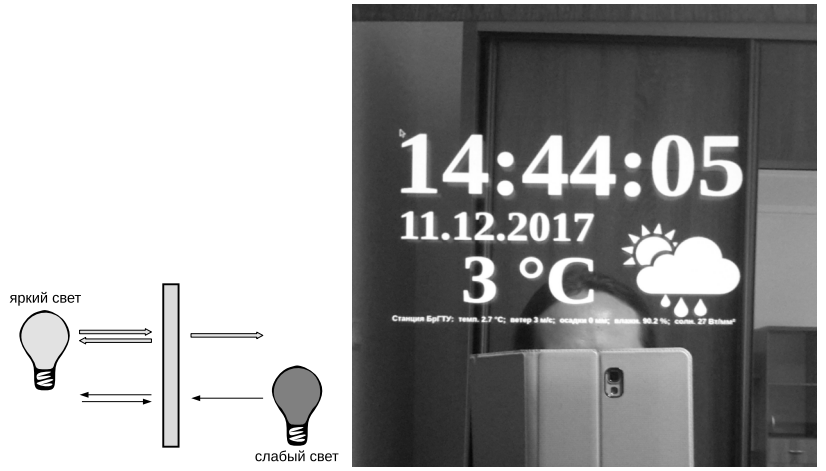

Рис. 1: Зеркало Гезелла: принцип действия (слева) и использование в качестве дисплея (справа)

В качестве первичного средства распознавания пользователя используется камера, которая передает непрерывный поток изображения для поиска пользователя и его идентификации. В нашем случае использовалась камера Raspberry Pi NoIR, отличие которой – отсутствие инфракрасного фильтра перед светочувствительной матрицей, что позволяет получать более информативные изображения лица в условиях слабой освещенности.

Доступ в сеть для обмена данными — загрузки персонализованного контента для конкретного пользователя – устройству обеспечивается посредством WiFi, а также в системе предусмотрена возможность подключения USB-модема с сим-картой.

В качестве программной платформы выбрана GNU/Linux (использован дистрибутив Raspbian) и библиотека Qt. Для определения лица в системе применяется функционал библиотеки OpenCV. На этапе обнаружения используется метод Виолы-Джонса, когда окно установленного размера движется по изображению, и для каждой области изображения, над которой проходит окно, рассчитывается признак Хаара. Наличие или отсутствие предмета в окне определяется разни-

цей между значением признака и обучаемым порогом [3,4]. Поскольку для описания объекта с достаточной точностью необходимо большее число признаков, в методе Виолы-Джонса признаки Хаара организованы в каскадный классификатор.

Кроме того, в проекте опробовано распознавание пользователя по Bluetooth-устройствам ближнего действия и по RFID-метке [5] (распознавание RFID-меток выполняется считывателем RFID RC522, подключенным к GPIO) (рис. 2).

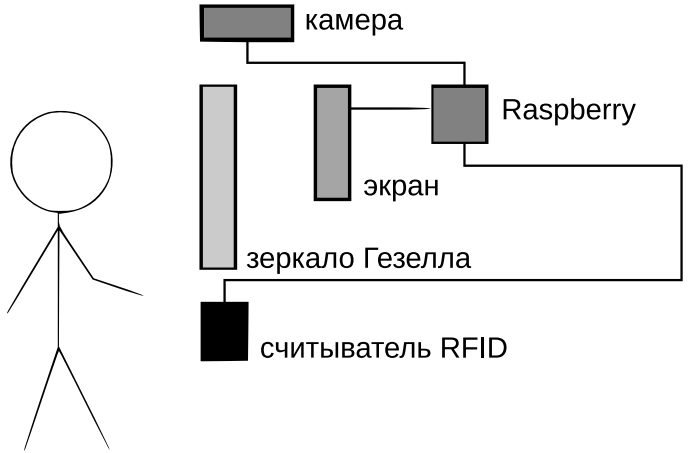

Рис. 2: Идентификация пользователя

Для решения задачи конфигурирования устройства в простейшем случае может использоваться подключаемая клавиатура; однако это плохо согласуется с концепцией встроенного информационного дисплея, поэтому были разработаны два варианта дистанционного управления:

- использование управляющих команд в приходящих на модем SMS-сообщениях (этот же канал, наряду с RSS, может служить источником персонализованных данных);
- применение QR-кодов, считываемых камерой устройства.

В случае статических команд, не требующих задания параметров, последний вариант предполагает использование печатного руководства с заранее сгенерированными страницами QR-кодов, кото-

рые можно «показать» зеркалу при необходимости. Более сложное управление может быть реализовано в виде Android-приложения в смартфоне, динамически генерирующего QR-код на основе выбранных пользователем опций настройки и выводящего его на экран для «показа».

Литература

[1] Echo & Alexa — Amazon Devices [Электронный ресурс] `https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011` — 8.04.2018.

[2] *Low J.* Two-Way Mirrors [Электронный ресурс] `http://www.jimloy.com/physics/mirror0.htm` — 11.07.2012.

[3] *P. Viola, M.J. Jones* Robust real-time face detection // International Journal of Computer Vision, vol. 57, no. 2, 2004., pp.137–154.

[4] *P. Viola, M.J. Jones* Rapid Object Detection using a Boosted Cascade of Simple Features // Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.

[5] *В. В. Буслюк, Д. А. Костюк, Д. И. Кульбеда, Н. А. Терешкевич, В. А.Юхно.* О возможностях применения в вузах аутентификации на основе персональных устройств // Тринадцатая конференция «Свободное программное обеспечение в высшей школе»: Материалы конференции / Переславль, 26–28 января 2018 года. М.: Базальт СПО, 2018. — С. 45–49.

Дмитрий Левин
Москва, Базальт СПО
`https://strace.io`

strace 2019

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 28 лет и широко применяется для диагностики, отладки и изучения поведения ПО. Многочисленные параметры управления фильтрацией дают возможность пользователю strace легко и гибко настраивать отображение системных вызовов и сигналов. С каждым выпуском strace таких возможностей становится больше, а точность отображения — выше.

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

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

seccomp-assisted system call filtering

В strace версии 5.3, выпущенной в сентябре 2019 года, реализована фильтрация системных вызовов с помощью интерфейса seccomp ядра linux. При указании параметра `-n` strace автоматически создаёт bpf-программу для фильтрации системных вызовов. Такой способ фильтрации снижает на два порядка накладные расходы обработки тех системных вызовов, которые не подлежат трассировке.

System call return status filtering

В strace версии 5.2, выпущенной в июле 2019 года, добавлена фильтрация системных вызовов по коду возврата:

- общий случай: `strace -e status=`*`set`*
- только завершившиеся успешно: `strace -z`
- только завершившиеся неудачно: `strace -Z`

Поддержка интерфейса PTRACE_GET_SYSCALL_INFO

В strace версии 4.26, выпущенной в декабре 2018 года, реализована поддержка нового интерфейса PTRACE_GET_SYSCALL_INFO

ядра linux, который позволяет надежно различать 64-битные и 32-битные системные вызовы на архитектуре x86-64. strace переключается на использование этого интерфейса автоматически, если ядро linux его поддерживает. Интерфейс PTRACE_GET_SYSCALL_INFO доступен в ядре linux начиная с версии 5.3.

Поддержка новых системных вызовов

Множество системных вызовов, для которых реализованы детальные парсеры, расширено и соответствует ядру linux версии 5.3, включая в себя новые системные вызовы:

5.3 : pidfd_open, clone3

5.2 : open_tree, move_mount, fsopen, fsconfig, fsmount, fspick

5.1 : clock_gettime64, clock_settime64, clock_adjtime64, clock_getres_time64, clock_nanosleep_time64, timer_gettime64, timer_settime64, timerfd_gettime64, timerfd_settime64, utimensat_time64, pselect6_time64, ppoll_time64, io_pgetevents_time64, recvmmsg_time64, mq_timedsend_time64, mq_timedreceive_time64, semtimedop_time64, rt_sigtimedwait_time64, futex_time64, sched_rr_get_interval_time64, pidfd_send_signal, io_uring_setup, io_uring_enter, io_uring_register

Лицензия

Условия распространения strace изменились с permissive на copyleft.

Так, исходный код тестов strace, с помощью которого, помимо всего прочего, было найдено множество ошибок в ядре linux и других проектах, теперь распространяется на условиях GNU General Public License v2+.

Остальная часть strace распространяется на условиях GNU Lesser General Public License v2.1+.

Литература

[1] *Дмитрий Левин*, Modern strace // Пятнадцатая конференция разработчиков свободных программ. Калуга, 21–23 сентября 2018 года. Тезисы докладов. Москва, Базальт СПО, 2018, стр. 18–22.

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

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

Научное издание

ШЕСТНАДЦАТАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ СВОБОДНЫХ ПРОГРАММ

Тезисы докладов

Ответственный редактор В.Л. Черный

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

Подписано в печать 23.09.2019 г.
Формат 60x90/16. Усл.печ.л. 3,38. Тираж 100 экз. Изд. № 200.
Издательство ООО «МАКС Пресс»
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.

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