ALT LINUX TEAM

OSSDEVCONF-2023

XIX

конференция разработчиков свободных программ

29 сентября – 1 октября 2023, Переславль-Залесский

ТЕЗИСЫ ДОКЛАДОВ

Научное издание

Организаторы конференции

ООО «Базальт СПО»
Институт Программных Систем РАН

Девятнадцатая конференция разработчиков свободных программ

Переславль-Залесский, 29 сентября — 1 октября 2023 года

Сборник материалов конференции
Научное издание

МАКС ПРЕСС

МОСКВА – 2023

УДК 004.91
ББК 32.97
Д25

https://elibrary.ru/rtrbfn

Д25 **Девятнадцатая конференция разработчиков свободных программ** : материалы конференции / Переславль-Залесский, 29 сентября – 1 октября 2023 г. / отв. ред. Чёрный В.Л. – Москва : МАКС Пресс, 2023. — 144 с.

ISBN 978-5-317-07050-2

DOI: https://doi.org/10.29003/m3562.978-5-317-07050-2

В книге собраны материалы конференции, одобренные Программным комитетом девятнадцатой конференции разработчиков свободных программ.

Ключевые слова: свободное программное обеспечение, разработка свободного программного обеспечения, безопасность программного продукта, Linux, репозиторий Sisyphus, сообщество разработчиков.

УДК 004.91
ББК 32.97

Nineteenth Free Software Developers Conference : conference materials / Pereslavl-Zalessky, September 29 – October 1, 2023 / resp. ed. Cherny V.L. – M. : MAKS Press, 2023. – 144 p.

ISBN 978-5-317-07050-2

DOI: https://doi.org/10.29003/m3562.978-5-317-07050-2

The book contains conference proceedings approved by the Program Committee of the Nineteenth Free Software Developers Conference.

Keywords: free software, free software development, software product security, Linux, Sysiphus repository, developer community.

ISBN 978-5-317-07050-2

Программа конференции

29 сентября, пятница

12.00-14.30 Заселение, обед, регистрация участников

13:50 Автобус от гостиницы «Переславль»

Дневное заседание
14.30–18.20

14.30 Приветственное слово организаторов

16.20–16.40 Кофе-пауза

30 сентября, суббота

09:30 Автобус от гостиницы «Переславль»

Утреннее заседание
10.00–13.00

13:00 Автобус в гостиницу «Переславль»

13.00–15.00 Перерыв на обед

14:40 Автобус от гостиницы «Переславль»

Вечернее заседание
15.00–19.00

16.40–17.00 Кофе-пауза

1 октября, воскресенье

09:30 Автобус от гостиницы «Переславль»

Утреннее заседание
10.00–13.20

Вне программы

Анатолий DOC Якушин
Москва, Независимый эксперт

Мифологемы мира программного обеспечения

Аннотация

Мифологемы приводят к существенному искажению реальности, навязывают свою точку зрения, выгодную определённым кругам. Массовый приход в СПО в течение последнего времени новых групп пользователей, приносящих чуждые движению мифологемы, приводит к терминологическому размыванию, нечёткости формулировок и недопониманию между пользователями.

Современное массовое сознание всё чаще отказывает своим носителям в критическом мышлении, заменяя его сконструированной реальностью. Для подобных элементов отражения реальности в литературе обычно используют термин «мифологема». В современной науке понятие «мифологема» остаётся одним из дискутабельных, что обусловлено междисциплинарным характером термина и наличием разных его трактовок в пределах одной области знания. В данной работе под мифологемой понимается некий популярный, социально значимый искажённый образ действительности, активно участвующий в формировании мировоззрения и поведения отдельных членов социума.

Отличие мифологемы от реальности увеличивается в тех областях, где отсутствуют устоявшиеся представления о предмете обсуждения, причём зачастую именно мифологемы и создают массовое суждение о подобных областях. Информационные технологии, к сожалению, стали одной из первых жертв подобной искажённой популяризации. Это объясняется относительной новизной данной области человеческого знания, отсутствием доступной и ясной неспециалисту информации о предмете обсуждения, а зачастую и навязанным превратным представлением в интересах определённых групп и отдельных компаний.

В качестве наверное наиболее распространённого примера подобных ложных массовых представлений можно привести мифологему «покупка программы». Обычно под «покупкой программы» подразумевается платёж владельцу авторского права на некое программное

средство, в обмен на который заплативший получает некое неисключительное право использовать копию программы. Согласитесь, что подобная передача прав имеет мало общего с реальной покупкой. Однако стойко внедрённое в массовое сознание мифологема о том, что программные средства «покупают» и «продают» ведёт к порождению таких популярных мифологем, как «воровство программ», «пиратство». Вал подобных ложных представлений, перенесённых на любой цифровой контент, породил по всему миру целый ряд абсурдных законопроектов по защите цифровых копий любого вида, за который многие и многие получили вполне серьёзные судебные приговоры, зачастую более суровые, чем уголовные наказания за реальные преступления.

Сообщество разработчиков и пользователей свободного программного обеспечения (СПО) за сорок лет своего развития неоднократно испытывало массовое увеличение своего состава, обусловленное в разные годы разными причинами. Подобные колебания носят как глобальный, так и локальный характер. События последних лет в России вызвали небывалый интерес к СПО и, как следствие этого, значительное количественное увеличение сообщества. К сожалению, вновь пришедшие зачастую поверхностно относятся к знакомству с традициями и терминологией мира СПО и привносят привычные им мифологемы мира программного обеспечения. Это неизбежно приводит к отсутствию взаимопонимания, порождает конфликты и в конечном итоге снижает качество сообщества в целом.

Рассмотрим некоторые мифологемы, сопровождающие свободное программное обеспечение долгие годы.

Свободное программное обеспечение не является коммерческим. Данное ложное утверждение основано на мнении, что единственным возможным способом извлечения прибыли является продажа лицензий на ПО. Однако анализ рынка показывает, что от 70 до 80 процентов прибыли на рынке программного обеспечения получается за счёт продажи различных услуг по эксплуатации, а не от продажи лицензий. Следует помнить, что коммерческое программное обеспечение — это программное обеспечение, разрабатываемое и поддерживаемое в рамках соответствующей специализации бизнеса на заказ для конкретного конечного пользователя, либо с расчётом на заранее не определённый круг конечных пользователей с целью извлечения прибыли. Коммерческое программное обеспечение следует отличать от так называемой «домашней» (ad home) разработки программ (разработки

силой самой организации — конечного пользователя) и от разработки программ вне коммерческого контекста вообще (в ходе научного экспериментирования, учебных проектов, как хобби и т. п.).

Свободное программное обеспечение редко используется. Обычно эту мифологему любят использовать те, кто мало знаком с положением дел в IT-отрасли и в основном использует в повседневной жизни офисные пакеты и игры. Действительно, в ряде областей СПО пока ещё используется нечасто, но в инфраструктурных областях, например в функционировании Интернета, доля СПО весьма заметна. Всё зависит от точки зрения и характера подсчёта.

Свободное программное обеспечение ничего не стоит. Эта мифологема прямо порождена мифологемой «покупка программы». Из того, что пользователь не платит за лицензию, следует вывод о бесплатности вообще. Однако реальная стоимость программного продукта определяется не ценой лицензии, а совокупной стоимостью владения, куда входит стоимость аппаратно-программного комплекса, его эксплуатации, апгрейда, стоимость обучения персонала и пользователей и т. д. Если конечный пользователь дома не учитывает эти затраты, хотя и несёт их, то в корпоративной среде подобные расходы учитываются и составляют весьма значительные суммы. Однако действительно, совокупная стоимость владения СПО всегда ниже проприетарного на стоимость лицензий.

Свободное программное обеспечение хуже по своему качеству, чем проприетарное. (Также существует встречная мифологема о том, что СПО по качеству лучше проприетарного). Многочисленные исследования однозначно показывают, что не метод разработки, ни лицензия не влияют на качество программного продукта.

Таким образом, сегодня мы знаем, что:

- СПО — это коммерческое программное обеспечение, разрабатываемое и поддерживаемое в рамках соответствующей специализации бизнеса на заказ для конкретного конечного пользователя, либо с расчётом на заранее не определённый круг конечных пользователей с целью извлечения прибыли.
- Совокупная стоимость владения (TCO) свободного программного продукта всегда меньше, чем проприетарного, как минимум на стоимость проприетарной лицензии.
- Существует значительное количество проверенных временем и рынком методов монетизации СПО.

- СПО ни в коей мере не противоречит авторскому и патентному праву, а базируясь на них, предоставляет пользователю наименее обременительную реализацию прав автора произведения.
- СПО сегодня является единственным эффективным средством межкорпоративного обмена передовыми технологиями.

Александр Аксельрод
Москва, «Тинькофф банк»
Проект: OrgMode `https://orgmode.org`

Инженерный способ делать менеджмент

Аннотация

Для управления разработкой традиционно используют Jira, Mantis, Redmine. Для хранения знаний используют Confluence. Для управления личной эффективностью популярным инструментом является Notion. Доклад посвящён возможности замены всех этих инструментов Emacs'ом при условии инвестирования времени инженерной команды в освоение этого инструмента.

1. Основные возможности `OrgMode`.

- Язык разметки. Среди прочего, предоставляет функциональность, аналогичную Markdown/Latex, включает раскраску текста, организацию заголовков и таблиц.
- Экспорт статьи в различные форматы. Действительно богатый функционал экспорта — помимо традиционных PDF/Tex/Odt поддерживает экзотические экспорты, такие как в формате статьи, в презентации в Beam или страницы в Hugo. Последнее позволяет вести статические Web-сайты в OrgMode и публиковать их, не выходя из Emacs.
- Среда выполнения кода. Является аналогом Jupiter Notebook, позволяет в тексте статьи размещать код, выполняемый при экспорте. Результат выполнения подставляется вместо блока исходного кода и является частью результата экспорта.
- Планировщик. Предоставляет функционал управления ToDo-листами. Сами ToDo могут размещаться во всех Org-файлах.

По задачам доступна возможность поиска, фильтрации по тегам. При создании ToDo можно указать периодичность повторения напоминания, дедлайн, к которому должна быть сделана таска. Также можно задать набор различных статусов, в которых может быть таска.

- Агенда. Можно сконфигурировать Дешборд, который отобразит в удобном виде важную информацию, собранную из всех ToDo, размещённых в разных файлах. По умолчанию отображает таски, запланированные на неделю или с истекающим/истёкшим дедлайном. При этом с помощью конфигурации можно расширить набор отображаемой информации — например, в отдельном разделе отобразить высокоприоритетные таски или все таски в определённом статусе.

2. `EXWM` как правильный способ использования `Emacs`.

`Emacs` – это не просто текстовый редактор, это комбайн, в котором можно читать новости, слушать подкасты, проверять почту, общаться в `Telegram`, программировать и многое другое. Среди прочего, есть возможность сделать из `Emacs` средство управления окнами. Проект называется `EXWM` и представляет собой тайловый оконный менеджер, в котором команды `Emacs` являются `First-Class-Citizen`.

3. Использование `OrgMode + EXWM` для личной эффективности.

Объединяя `OrgMode` и `EXWM`, мы встраиваем `OrgMode` в повседневную жизнь, максимально упростив доступ к работе и таскам. Не нужно переключаться между окнами, чтобы посмотреть повестку на сегодня/создать новую ToDo. Из любого окна (браузера/конференции) можно создать таску с помощью комбинации C-c C t t или открыть текущую агенду.

4. Измерение эффективности процесса разработки с помощью `Org-mode + Jira`.

Для управления разработкой часто используется `Scrum`. Отличительной особенностью Scrum является работа по спринтам — команда планирует 2-недельную итерацию и принимает обязательство выполнить весь запланированный на итерацию объём работы. Основной метрикой измерения предсказуемости разработки в `Scrum` является `Scope-Drop` — соотношение не выполненных задач из изначального комита спринта к общему объёму запланированной работы. Для контроля работы команд разработки, работающих по `Scrum`, хорошо подходит ритуал `OpReview`. На `Op-Review` среди прочего проводит-

ся анализ `Scope-Drop`, а также ход выполнения спринта. С помощью `OrgMode` можно написать скрипт, который по API получает из `Jira` данные о последнем спринте, считает `Scope-Drop` и представляет это в виде удобного отчёта.

Таким образом, к `Op-Review` получается автоматически сгенерированный отчёт в формате `Org`-файла, который можно проанализировать совместно с тимлидом команды. Ключевым отличием от аналогичных отчётов в `Jira`/прочих инструментах (некоторые используют для этого `Metabase`) является то, что отчёт можно редактировать в процессе `Op-Review`, записывая туда полученные выводы/обязательства до следующего `Op-Review`. Конечно же, делать это можно в формате тех же ToDo. Это представляет ключевое преимущество такого подхода.

5. Убираем `Jira`: настройка `Scrum`-процесса с использованием `Org-Mode`.

Развитием этого подхода является переход из `Jira` в `OrgMode`. Поскольку в `Org` можно вставлять куски кода, создавать таски, в т. ч. на основе заранее определённого шаблона, то мы можем перенести ведение задач из `Jira` в `OrgMode`. Для этого нужно проделать следующую работу:

- сконфигурировать Агенду в формате Scrum-доски;
- научить dev-team использовать Emacs. Для любителей Vim доступен Doom Emacs;
- договориться о формате описания спринта — создать шаблон задач/шаблон спринта/создать свойства старта спринта;
- написать скрипты, позволяющие запуск спринт/ завершать спринт/ генерировать отчёт для `Op Review`.

Ключевыми преимуществами этого подхода являются:

- всё в `Git` — таски и анализ хода разработки хранится там же, где исходный код;
- информация о задачах разработки всегда под рукой — фактически, встроена в оконный менеджер, нет необходимости открывать браузер, дедлайны всегда под рукой;
- нет лицензионных костов, нет необходимости в централизованной инсталляции;
- высокая степень кастомизации процесса.

Вениамин Арефьев, Никита Барабанов, Георгий Курячий
Москва, Базальт СПО, ВМК МГУ

Проект: HWorker `https://github.com/FrBrGeorge/HWorker`

Организация рабочего процесса разработки системы проверки домашних заданий

Аннотация

В докладе описывается опыт разработки проекта со свободной лицензией, предназначенного для автоматической проверки домашних заданий и публикации результатов. Разбираются особенности организации рабочего процесса в условиях сжатых временных рамок и ограниченных человеческих ресурсов, а также проблемы, с которыми можно встретиться в процессе разработки и варианты их решения.

Постановка задачи

Основным назначением системы является сопровождение учебных курсов «Язык программирования Python», «Совместная разработка на Python» и «Практические аспекты сетевых протоколов в Linux», которые читаются на кафедре АСВК факультета ВМК МГУ, а также любых других курсов со схожей структурой. А именно: сдача заданий по электронной почте или путём выкладывания в открытых репозиториях, оформление тестов в виде «входные данные – эталонный вывод».

Рабочий процесс и его особенности

- **Ограничения:** Данный проект имел строгие временные рамки, начать его можно было не раньше 1 июля, а закончиться он мог не позже 31 августа. То есть на всё планирование, разработку и тестирование отводилось не больше двух месяцев.
- **Precious source:** Разработка велась в одном главном публичном репозитории, в который принимались только (почти) пулл–реквесты. Ревью не проводилось ввиду крайне малого размера команды, процесс «вливания» изменений увеличился бы в кратном размере.

- **Обсуждение архитектуры:** Первая неделя работы была полностью посвящена определению основных функциональных требования к проекту, которые, как и в любом другом проекте, по мере разработки дополнялись, изменялись и уточнялись.
- **Пятиминутки:** В начале каждого рабочего дня проводились общие звонки, на которых каждый описывал проделанную работу, возникшие проблемы и планы на день (в идеальном варианте по пять минут на человека, но идеал на то и идеал). Раз в неделю устраивались очные (во возможности) встречи с более широким обсуждением вопросов и проблем, возникших при разработке.
- **Wiki:** Основная информация о проекте, причём эта вики не является обычным генератом от того же `sphinx`. В вики проекта были описаны формальные и неформальные требования к каждому модулю системы и их подмодулям, а также различные предложения по тому, как могут быть реализованы данные модули предлагались на соответствующих страницах.
- **API:** Для создания межмодульного взаимодействия были выделены специальные публичные функции, которые могут вызывать конечные пользователи модуля. Поэтому эти функции были выделены в отдельную страничку `API` на вики проекта. На этой страничке описаны все публичные функции всех модулей, которыми могут пользоваться другие модули. Также во время разработки эта страница использовалась для создания запросов на написание таких функций.
- **Стиль программирования и документирование:** Все публичные функции, описанные в `API` должны иметь полное документирование, то есть иметь описание и типизацию всех аргументов и возвращаемого значения. При создании `Pull request`'а в `Precious source` весь исходный текст должен быть обработан с помощью форматтера `black` [1]. Данные требования были введены исключительно в интересах упрощения взаимодействия между разработчиками, то есть унификации результатов их работы, которое способствовало пониманию кода друг друга.
- **Тесты и CI:** В самом начале разработки было принято решение о том, что весь написанный публичный функционал модулей стоит покрывать тестами с помощью `pytest` [2], если это

представляется возможным сделать за разумные сроки (меньше чем писался сам модуль). В последние две недели разработки было выяснено, что проект работает по–разному на разных операционным системах, поэтому был внедрён `Github Continuous Integration` [3], который при каждом измерении репозитория запускает тесты, позволяя убедиться в том, что функционал работает корректно.

Результаты

По итогу совместной работы система успешно функционирует на корпусе решений прошлогодних курсов, и далее планируется введение в полную эксплуатацию и соответствующее сопровождение проекта. На данный момент `HWorker` умеет:

- Скачивать выполненные студентами домашние задания.
- Разбивать их на решения, тесты и ссылки на тесты других студентов (с возможностью для преподавателя указывать свои тесты отдельно).
- Запускать ассоциированные тесты.
- Оценивать результаты при помощи функций, написанных преподавателем.
- Публиковать эти результаты в удобном для восприятия виде.

Анализ эффективности

В команде присутствовал один тимлид и два разработчика. Изначально планировалось, что в команде будет три разработчика, первые два были найдены достаточно быстро (они работали со схожими системами до этого), а с поиском последнего возникли трудности. После первого месяца разработки стало понятно, что найти человека и ввести его в курс дела не представляется возможным, поэтому для успешного завершения проекта было принято решение об увеличении времени работы для уже работающих над проектом. В итоге для такого небольшого и быстрого проекта уменьшение команды оказало положительное влияние. Также несмотря на заранее оговорённые области работы в конце возникли некоторые проблемы с эшелонированием задач, один не мог работать без срочных изменений от другого.

Благодаря тому, что на формирование и обсуждение архитектуры было выделено достаточно времени, получилось придумать и реализовать модули системы сравнительно независимо. Это очень помогло исправить серьёзный «костыль» в логике работы с домашними заданиями в последние дни работы над проектом, не приводя к накоплению технического долга.

Отношение к пятиминуткам в нашей команде полярное. С одной стороны «двухчасовики» полезны потому, что можно обсудить все тонкости планируемого функционала и убедиться в том, что все стороны правильно понимают мысли друг друга, а с другой стороны, такие долгие обсуждения очень сильно утомляют и могут сбить рабочий настрой.

В проект достаточно поздно была внедрена практика использования непрерывная интеграция, что не оказало серьёзного негативного влияния на процесс разработки. `Continuous Integration` помог бы исправить проблемы взаимодействия разработчиков использующих разные операционные системы в процессе работы.

Литература

[1] Black: the uncompromising code formatter [Электронный ресурс]: `https://black.readthedocs.io/en/stable/`

[2] Pytest: helps you write better programs [Электронный ресурс]: `https://docs.pytest.org/en/7.4.x/`

[3] GitHub Actions documentation [Электронный ресурс]: `https://docs.github.com/en/actions`

Антон Бондарев
Санкт-Петербург

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

СПО в процессах безопасной разработки на примере ОС РВ Embox

Аннотация

Основная критика использования Open source в объектах критической информационной инфраструктуры заключается в том, что поддержка и сопровождение подобных решений имеет ряд трудностей, в результате итоговые системы небезопасны. В статье представлен ряд методов для увеличения безопасности программного обеспечения, в том числе тестирование, и использование безопасной модели разработки. Обозначено место в этом процессе Open Source-проектов и показано, что использование модели СПО не только не ухудшает безопасность конечных систем, но может улучшить её.

В докладе также будут рассмотрены процессы, по которым на базе открытой ОС РВ Embox могут быть построены надёжные и безопасные системы.

По данным Gartner, более 95% ИТ-компаний в мире сейчас используют Open source-решения. Среди остальных компаний не менее 40% предпочитают в работе свободное ПО. И если в системах общего назначения, где цена ошибки не очень велика, правильность применения СПО не вызывает сомнений, то, поскольку при создании критически важных вычислительных систем на первый план выходит не скорость и стоимость разработки, а качество и надёжность полученной системы, применение Open source-решений порой резко критикуется.

Основная критика заключается в том, что ПО для подобных систем должно быть протестировано, верифицировано и сертифицировано до попадания на реальный объект. Причём сделано это должно быть целиком, и кто-то должен обеспечить конечную целостность, надёжность и безопасность. Иными словами, существует мнение, что использование Open source-проектов в подобных системах недопустимо, поскольку непонятно, кто отвечает за внесённый код.

В данной работе представлен анализ методов обеспечения надёжности и безопасности вычислительных систем. В том числе, рассматривается влияние использование Open source в процессах, обеспечивающих безопасную разработку.

Любое программное обеспечение содержит ошибки. Например, одним из параметров, характеризующих качество ПО, является количество ошибок на 1000 строк кода.

Также существует ряд общепризнанных методов, которые повышают надёжность и безопасность ПО, в том числе и системного. Можно выделить такие группы: тестирование, переиспользование кода, следование рекомендациям безопасного кодирования (стандартам), применение статических анализаторов и других других средств анализа кода, использование подходящей архитектуры ПО, применение подходящих процессов разработки.

Тестирование — наиболее понятный способ повышения надёжности и безопасности ПО. Можно выделить несколько типов тестирования: unit-тестирование (тестирование модулей), интеграционное, системное. Также необходимо отметить такие типы тестирования как:

- функциональное — тестирование на соответствие заявленному функционалу;
- стресс- и нагрузочное тестирование — тестирование на предельных значениях нагрузки;
- тестирование безопасности — тестирование на надёжность работы при различных внешних воздействиях.

Важным параметром при тестировании является уровень покрытия тестами функциональности всей системы и отдельных её частей. В ходе тестирования ПО на предмет безопасности и надёжности необходима проверка как граничных значений передаваемых, так и средних (допустимых и не допустимых), передаваемых в объект тестирования. Одним из эффективных способов тестирования, увеличивающим, в том числе, безопасность ПО, является фаззинг-тестирование. Вариант тестирования, когда на вход отдельных функций и системы в целом подаются случайные (псевдослучайные данные).

Тестирование можно проводить в ручном и автоматическом режиме. Для построения систем с требуемым уровнем безопасности необходимо иметь документ, который содержит описание процесса разработки с указанными типами тестирования для каждой из стадий.

Неотъемлемым элементом для повышения качества кода является применения анализаторов кода, а также использование заложенных в компиляторах возможностей по выявлению потенциально опасных мест. Подобные средства хоть и обладают довольно большим числом ложных срабатываний, но сильно повышают качество кода в целом.

Но эффект от подобных практик, как и в случае с тестированием, может быть достигнут только при их применении их на регулярной основе. В качестве примера можно привести использование максимально возможного уровня сообщения об ошибках во флагах компилятора и применение флага интерпретации всех предупреждений как ошибки. В некоторых случаях этого трудно добиться, но все подобные случаи должны быть рассмотрены подробнейшим образом, и уменьшать уровень предупреждений нужно на как можно меньшем элементе кода и только для конкретного случая.

Другими эффективными способами использования средств повышения качества кода, заложенным в компиляторы, являются:

- использование санитайзеров;
- использование различных компиляторов;
- использование различных способов оптимизации кода.

Увеличения эффекта можно добиться, если сочетать различные виды тестирования с разными флагами оптимизации и санитайзерами.

Использование статических анализаторов является очень эффективным способом улучшения качества кода. Но в отличие от использования флагов компилятора работу со статическими анализаторами необходимо включать в процесс разработки. Кроме того, статический анализ существенно улучшает качество кода, но не может проверить логику работы системы.

На конечную функциональность очень сильно влияет применение различных подходов к архитектуре ПО, в том числе, и архитектуре ОС. Например, применение микроядра, может увеличить надёжность системы, но вместе с тем может существенно ухудшить её производительность.

Для построения эффективных вычислительных систем необходимо ещё на этапе проектирования учитывать те задачи, которые должна в итоге решать конечная система. Иными словами, специализированная система по своим характеристикам будет превосходить универсальную при решении конкретных задач. Следовательно, если ещё

на этапе проектирования системы описать требования к её характеристикам на каком-то формальном языке и в дальнейшем использовать это описание на разных стадиях жизненного цикла системы, в том числе при разработке базовых характеристик ОС, то это позволит не только создать систему, эффективно решающую задачи, описанные в требованиях, но и существенно сократить время разработки конечной системы.

Для классификации систем по уровню безопасности и надёжности существует принятый в России международный стандарт «Общие критерии оценки защищённости информационных технологий, Общие критерии», (англ. Common Criteria for Information Technology Security Evaluation, Common Criteria, CC). По данному стандарту существует 7 уровней Evaluation Assurance Level (EAL) (Вычислимых уровней надёжности):

- EAL1: Functionally Tested (функционально протестированная система);
- EAL2: Structurally Tested (структурно протестированная система);
- EAL3: Methodically Tested and Checked (протестировано по описанной методике);
- EAL4: Methodically Designed, Tested, and Reviewed (существует методика для разных этапов жизненного цикла продукта);
- EAL5: Semiformally Designed and Tested (существуют инструменты, с помощью которых формально описывается и ведётся процесс разработки);
- EAL6: Semiformally Verified Design and Tested (существуют инструменты для формального определения соответствия функциональности заявленным требованиям);
- EAL7: Formally Verified Design and Tested (существуют наборы инструментов, позволяющих осуществить проверку на полное соответствие полученной функциональности системы заявленным требованиям).

Таким образом, видно, что ключевым фактором для построения надёжных систем является процесс разработки системы, а не применение тех или иных инструментов или техник программирования. Все эти техники и методологии должны быть включены в описанный

процесс разработки, который будет приводить к созданию систем с требуемым уровнем надёжности и безопасности. Видно, что с повышением уровня безопасности системы увеличивается и автоматизация процесса, а начиная с уровня 5 необходимо применение формального языка, описывающего характеристики проектируемой системы.

Такой процесс описан в серии стандартов КТ-178 (перевод международного DO-178B). DO-178B требует внедрения в процесс наработки критически важного ПО так называемой V-образной модели. В упрощённом виде она представлена на Рис. 1.

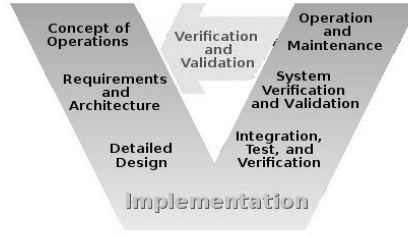

Рис. 1: V-образная модель разработки ПО

Данная модель, в отличие от водопадной, предполагает наличие обратной связи между этапами на спадающей и восходящей ветках разработки. Спадающая ветка включает этапы, предшествующие стадии реализации (собственно, написания кода). При этом по временной шкале этапы располагаются с постепенным увеличением детализации требований и архитектуры. На восходящей ветке располагаются этапы, отвечающие за проверку соответствующих требований. На каждом этапе из восходящей ветки в случае несоответствия предъявляемым требованиям процесс может вернуться на уровень, в котором были выявлены нарушения. Причём последним по времени стоит этап проверки на соответствие самым общим требованиям, то есть предъявляемым на начальном этапе работ.

При этом идеология СПО не только не противоречит данной модели, но и в значительной степени улучшает её. Ведь если рассмотреть использование Open source как базовых проектов, то есть частей, из которых собирается итоговая безопасная система, и соответствие

описываем требованиям, которые к этим частям предъявляются (в том числе описываем набор тестов на соответствие), то ухудшения характеристикне происходит, поскольку выявленное несоответствие проявится на поздних этапах создания системы. При этом будет произведена коррекция на более ранних стадиях. Улучшение же проявляется в том, что СПО-проекты используются, а, следовательно, тестируются в различных ситуациях, и таким образом неявно проходят этап дополнительного фаззинга.

Стандарт DO-178B требует, чтобы результатом каждого жизненного цикла стали различного рода артефакты, позволяющие провести формальную проверку требований, предъявляемых к данному этапу разработки. Такими артефактами могут являться документы, спецификации, тесты, диаграммы описания архитектуры и непосредственно программный код.

Естественно, важнейшую роль в данном процессе занимают спецификации. Причём это касается как спецификаций на требования по функциональности, так и спецификаций на методы проверки этих требований. Если требования формулируются на естественном языке, то следить за корректностью их выполнения довольно тяжело. Решением может быть описание требований на формальном языке и применение этого языка на всех этапах жизненного цикла ПО, автоматизирующее процесс разработки системы.

При применении формального языка описания требований для системы можно составить модель, которая может быть представлена графом предикатов, и, следовательно, может быть проверена на целостность и непротиворечивость.

На первом этапе граф состоит из формально описанных требований верхнего уровня системы. В ходе разработки системы граф дополняется различными деталями, которые представляют собой требуемые на данном уровне артефакты, также описанные на формальном языке с помощью DSL. При этом важным остаётся сохранение непротиворечивости и целостности графа на каждом этапе разработки системы.

Выводы

Уровень безопасность информационной системы зависит в первую очередь от процессов, которые применялись при её создании. Существует стандарт DO-178B, который описывает модель разработки для

увеличения надёжности. В частности, в её основе лежит так называемая V-образная модель или циклическая модель, которая в отличие от водопадной модели подразумевает возврат к более ранним стадиям проектирования в случае обнаружения каких то несоответствий заявленным характеристикам.

На разных стадиях разработки могут применяться более локальные средства увеличения надёжности систем, такие как различные виды тестирования, статический анализ и так далее.

Большую роль в увеличении надёжности конечных систем играет применение автоматизированных архитектурных методов, таких как задание требований и автоматическое формирование архитектуры и состава конечной системы на основе формальных описаний.

Применение СПО-проектов в качестве элементов, включённых в конечную систему, увеличивает её надёжность, поскольку увеличивается коэффициент покрытия в различных ситуациях. С другой стороны, применение СПО-проектов возможно только при встраивании их в конечную модель разработки, за что отвечает отдельный разработчик.

ОС РВ Embox содержит ряд средств, с помощью которых можно построить модель разработки конечной системы с очень высоким уровнем безопасности.

Арсений Краснов
Москва, SberDevices

Проект: Разработка virtio/vsock подсистемы ядра Linux
https://github.com/torvalds/linux

VSOCK: коммуникация между гостем и хостом с минимальными затратами

Аннотация

В рамках работы над собственной операционной системой Salute OS для умных устройств мы в SberDevices активно проводили исследования по использованию виртуализации в целях повышения безопасности и использовали технологию VSOCK, о которой пойдёт речь в данном докладе. Я являюсь одним из контрибьюторов в данную технологию в ядре Linux. VSOCK-сокеты — это семейство сокетов, которые предназначены для создания канала передачи данных между

приложением в гостевой ОС и приложением в хосте. Для работы с ними используется POSIX API-сокетов. В докладе рассказывается про отличия VSOCK сокетов от привычных нам сокетов (например, TCP), описывается внутреннее устройство этой подсистемы в ядре Linux, описываются сценарии использования таких сокетов, их текущий статус в ядре Linux и планы по дальнейшему развитию.

VSOCK — это акроним, который означает Virtual machine SOCKet. VSOCK-сокеты представляют собой POSIX-совместимый интерфейс сокетов, аналогичный TCP или UNIX-сокетам. Доступны те же самые вызовы, которые используются для работы с другими типами сокетов и описаны в POSIX: `socket()`, `connect()`, `listen()`, `bind()`, `accept()` и т. д. Разница с TCP, например, заключается в адресации: вместо IP-адреса используется так называемый CID (context ID). Некоторые значения CID являются зарезервированными: 1 — loopback (передача данных в пределах одной ОС, аналогично localhost), 2 — host, 3 и выше — для гостевых ОС. Порт же имеет тот же самый смысл, что и в TCP-сокетах. Таким образом, можно легко переделывать существующие приложения на данный тип сокетов. Также при создании VSOCK-сокета в вызов `socket()` нужно передавать параметр `AF_VSOCK` в отличие от TCP-сокетов, где используется `AF_INET`.

В чём преимущество использования таких сокетов, в отличие, например, от виртуальной сетевой карты и обычных сетевых протоколов? Основное отличие: VSOCK-коммуникация не требует никаких настроек — сразу после загрузки драйвера можно начинать обмен данными. Не нужно стартовать сетевой интерфейс, назначать ему адрес и другие параметры сети. CID хоста назначается автоматически и всегда равен 2 — таким образом, хост всегда доступен из гостя по данному адресу. CID же гостевых систем назначаются вручную либо автоматически при их запуске, и начинаются с 3.

На данный момент VSOCK используют следующие решения: `Qemu guest agent, Kata container agent, Android debug bridge`.

VSOCK подсистему можно представить как высокоуровневую реализацию `AF_VSOCK` сокетов и лежащую под ней транспортную часть, которая обеспечивает протокол передачи и саму передачу данных между гостем и хостом. Транспортная часть зависит от гипервизора. На данный момент реализованы четыре транспорта: `virtio, Hyper-V(Microsoft), VMCI (VMWare)` и `loopback`.

Поддерживаются три типа сокетов: `SOCK_STREAM` (на всех трёх транспортах), `SOCK_DGRAM` (только на VMCI), `SOCK_SEQPACKET` (только на virtio).

Остановимся подробнее на virtio-транспорте. Как видно из названия, для передачи данных здесь используются virtio-очереди. Одна очередь для приёма данных из гостя, вторая для передачи данных в гостя. В госте запускается virtio драйвер, со стороны хоста же работает vhost драйвер. Также реализуется внутренний протокол, который служит для двух целей:

1. Установка и разрыв соединения (аналогично например протоколу TCP). Это нужно для поддержки `SOCK_STREAM` и `SOCK_SEQPACKET` сокетов, так как они должны поддерживать установку соединения.

2. Контроль за объёмом передаваемых данных. Это позволяет ограничивать объём передаваемых данных принимающей стороне, чтобы не вызвать большой расход памяти, т.е. когда пакеты копятся на принимающей стороне, но не вычитываются из сокета и, таким образом, не могут быть освобождены.

VSOCK-сокеты поддерживаются в следующих утилитах для работы с сокетами/сетью: `wireshark`, `iproute2`, `tcpdump`, `nmap`, `iperf-vsock`, `socat-vsock`.

Также имеется поддержка в следующих языках: `C (glibc), python (3.7), Golang, Rust.`

Дальнейшие планы по развитию VSOCK:

1. `SOCK_DGRAM` на virtio транспорте (code review)

2. Передача в режиме zerocopy (code review). Используется флаг `MSG_ZEROCOPY` в вызове `send()/sendto()/sendmsg()`. В данном режиме память с данными, которые хочет передать пользовательское приложение, начинает использоваться как virtio-буфер и таким образом избегается копирование между userspace и ядром.

3. Приём в режиме zerocopy (RFC). В данном режиме пользователь создаёт маппинг виртуальной памяти с помощью вызова `mmap()`, и далее ядро при получении данных будет отображать в данный маппинг страницы приёмных virtio-буферов. Таким образом снова избегается копирование между userspace и ядром.

В заключение стоит отметить преимущества данного типа сокетов:

1. Легко адаптировать уже существующие сетевые приложения под данный тип сокетов.
2. Канал данных не требует никакой настройки по сравнению с виртуальной сетевой картой или механизмом обмена данными между гостем и хостом.
3. Уже имеется поддержка в библиотеках популярных языков программирования.

Алексей Костарев, Александр Степченко
Пермь, Москва

Проект: PODSEC (Podman Security) https://github.com/alt-cloud/podsec

Реализация rootless kubernetes в рамках ALTLinux-пакетов

Аннотация

В докладе рассматриваются вопросы одной из первых реализаций rootless kubernetes в рамках ALTLinux-пакетов защиты контейнерных решений podsec (Podman Security). Сравнение rootless и rootfull-решения. Особенности реализации rootless-решения. Механизмы защиты от несанкционированного доступа и использования уязвимостей.

Базовое решение Usernetes

Реализация `rootless kubernetes` была сделана на основе проекта Usernetes[1].

Отличие данного решения от стандартного `rootfull`-решения:

- Основные программы (`kube-apiserver`, `kube-controller`, `kube-scheduler`, `kube-proxy`, `coredns`, `etcd`, ...) запускаются в виде сервисов, а не в виде контейнеров.
- Сертификаты генерируются сторонней программой (`cfssl`) и помещаются в тома, доступные узлам кластера.

Достоинства данного решения:

- Позволяет разворачивать rootless kubernetes в одноузловом или кластерном варианте.

[1] `https://github.com/rootless-containers/usernetes`

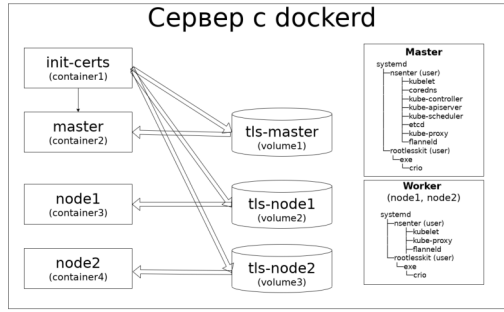

Рис. 1: Схема разворачивания Usernetes rootless kubernetes.

- Обеспечивает быстрое тестирование решения путём разворачивания `rootless kubernetes` в виде стека сервисов (`docker-compose`) в рамках одного сервера.

В то же время это решение носит экспериментальный характер, что предопределило следующие недостатки:

- Генерация сертификатов производится программой `cfssl`, отсутствующей в репозитории пакетов `ALTLinux`.
- Для создания и доступа к сертификатам необходимо использовать либо `docker-тома,` либо `разделяемую файловую систему`.
- Решение создаётся без использования стандартной команды разворачивания кластера `kubeadm`, что ограничивает варианты разворачивания решения в виде кластера.
- Решение поднимается либо в виде стека сервисов (`docker-compose`) в рамках одного сервера, либо требует создание `разделяемого тома` в рамках кластера для доступа к сертификатам.
- Процедуры добавления, удаления узлов в кластер, обновления сертификатов нестандартные и требуют дополнительной квалификации от системного администратора.
- Реализация основных компонент кластера в виде сервисов, а не в виде контейнеров (`POD`'ов) несёт потенциальные риски при обновлении версий `kubernetes`, так как требует анализа возможных изменений в образах новых версий `kubernetes`.

Стандартная схема разворачивания rootfull kubernetes кластера через команду kubeadm

Для снятия основных недостатков необходимо было реализовать на основе `Usernetes` вариант разворачивания `rootless kubernetes`-кластера через стандартную программу `kubernetes` — `kubeadm`.

Процесс разворачивания `rootfull-кластера` через `kubeadm` выглядит следующим образом:

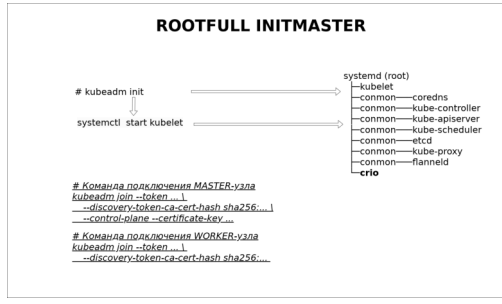

Рис. 2: Схема инициализации rootfull master сервера.

При запуске на начальном (`Init`) `Master`(`ControlPlane`) узле `kubeadm init` производит следующие основные действия:

- загружает с регистратора все необходимые (`kube-apiserver`, `kube-controller`, `kube-scheduler`, `kube-proxy`, `coredns`, `etcd`, ...) `kubernetes`-образы;
- запускает через `systemctl start kubelet` сервис `kubelet`;
- генерирует сертификаты и kubernet-манифесты в каталоге `/etc/kubernetes/manifests/`;
- поднимает через `kubelet` основные контейнеры (`POD`'ы);
- конфигурирует одноузловой кластер и записывает его конфигурацию в `etcd`;
- генерирует строки подключения (`kubeadm join ...`) к кластеру `ControlPlane` и `Worker` узлов.

Для подключения дополнительных узлов на них запускается команда `kubeadm join` с указанными параметрами.

Разворачивание rootless-кластера через команду kubeadm

Реализованный в рамках ALTLinux пакет `podsec-k8s`[2] — это набор скриптов, который позволяет разворачивать rootless-кластер стандартной немодифицированной бинарной командой `kubeadm`.

Все разворачиваемые процессы в `rootless kubernetes` кластере запускаются в `user namespace` системного пользователя `u7s-admin`, обладающего обычными (непривилегированными) правами. Таким образом, все `POD`'ы имеют права обычного непривилегированного пользователя и не могут при взломе повлиять на функционирование узла.

Разворачивание Master(ControlPlane) rootless -узла

Скрипты пакета `podsec-k8s` находятся в каталоге `/usr/libexec/podsec/u7s/bin`.

Часть из них (`kubeadm`, `systemctl`) являются оболочками над стандартными командами системы с аналогичным именем. За счёт установки переменной `PATH`

```
export PATH=/usr/libexec/podsec/u7s/bin/:$PATH
```

они вызываются до вызова основных системных команд `kubeadm` и `systemctl`, и после выполнения настройки среды вызывают основные системные команды.

При запуске на начальном (`Init`) `Master`(`ControlPlane`) узле скрипт `kubeadm init` производит следующие основные действия:

- В окружении пользователя `u7s-admin` запускает сервис `rootlesskit`, который позволяет запускать процессы с правами «псевдоroot». Процессы в рамках `user namespace` пользователя `u7s-admin` имеют права `root` (`UID=0`), могут создавать сетевые интерфейсы, настраивать правила маршрутизации `iptables` и выполнять другие системные действия. Но в рамках `HOST-системы` все производимые действия изолируются в `user namespace` пользователя `u7s-admin` и никак не взаимосвязаны с системными ресурсами HOST-системы. В процессе `rootlesskit` в рамках `user namespace` пользователя `u7s-admin` запускаются:

1. а) подпроцесс `crio`, обеспечивающий работу с контейнерами;

[2] `https://github.com/alt-cloud/podsec`

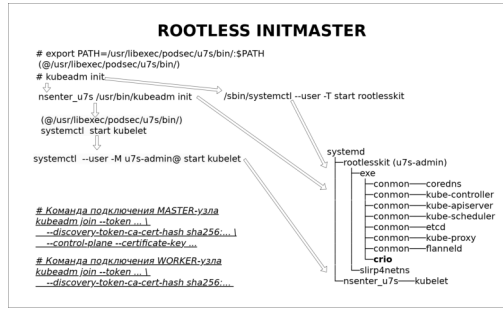

Рис. 3: Схема инициализации rootless master сервера.

b) подпроцесс `slirp4net`, обеспечивающий создание сетевых интерфейсов и работу с ними.

- В рамках `user namespace` пользователя `u7s-admin` запускается стандартная бинарная команда `kubeadm`.
- В рамках этого `namespace` команда `kubeadm` имеет права `псевдо-root`:

1. a) Загружает с регистратора все необходимые (`kube-apiserver`, `kube-controller`, `kube-scheduler`, `kube-proxy`, `coredns`, `etcd`, ...) `kubernetes`-образы.
   b) Запускает через `systemctl start kubelet` сервис `kubelet`.
   c) Генерирует сертификаты и kubernet-манифесты в каталоге `/etc/kubernetes/manifests/`.
   d) Поднимает через `kubelet` основные контейнеры (`POD`'ы).
   e) Конфигурирует одноузловой кластер и записывает его конфигурацию в `etcd`.
   f) Генерирует строки подключения (`kubeadm join ...`) к кластеру `ControlPlane` и `Worker`-узлов.

- Производит настройку интерфейсов и правил `iptables` в HOST-системе и `user namespace` пользователя `u7s-admin`
- Производит настройку файлов конфигурации (`.kube/config`) пользователей `root` и `u7s-admin`.

Таким образом, при запуске скрипта `kubeadm` пакета `podsec-k8s` производится полноценная настройка `Master` (`ControlPlane`) узла `rootless кластера`.

Подключение дополнительных ControlPlane и Worker-узлов

Подключение дополнительных ControlPlane и Worker узлов производится аналогичным образом:

- Устанавливается пакет `podsec-k8s`.
- Устанавливается переменная среды `PATH`:
- Запускается скрипт 'kubeadm' пакета `podsec-k8s` в режиме `kubeadm join ...` с параметрами, сгенерированными при установке начального `Master` (`ControlPlane`) узла.

Группа пакетов podsec (Podman Security)

Пакет `podsec-k8s` входит в состав группы пакетов:

- `podsec`
  - Разворачивание локального регистратора (`regitry.local`) и сервера подписей (`sigstore.local`).
  - Создание пользователей группы `podsec-dev`, имеющих права на создание docker-образов, их подписывание и размещение в локальном регистраторе.
  - Создание пользователей группы `podsec`, имеющих возможность запуска подписанных образов с локального регистратора. Работа с образами из других источников не допускается.
  - Настройка политик доступа и работы с образами для различных групп пользователей.
  - Загрузка и обновление новых версий `kubernetes` с регистратора `registry.altlinux.org`, их архивирование, подпись и размещение их на локальном регистраторе.
- `podsec-k8s-rbac`:
  - Создание удалённых рабочих мест с генерацией сертификатов для доступа к `rootfull` или `rootless kubernetes` кластеру.

  - Создание привязки (`bind`) пользователя к обычной или кластерной роли.
  - Просмотр списка связанных с пользователем ролей.
  - Удаление привязки (`bind`) пользователя к обычной или кластерной роли.
- `podsec-inotify`:
  - Автоматический мониторинг политик безопасности на узлах кластера и рабочих местах пользователей.
  - Контроль несанкционированного доступа к `API kubernetes` кластера.
  - Мониторинг docker-образов узла сканером безопасности `trivy`.

Литература

[1] PODSEC (Podman Security) — https://github.com/alt-cloud/podsec.

[2] Rootless kubernetes — `https://www.altlinux.org/Rootless_kubernetes`.

[3] Kubernetes — https://www.altlinux.org/Kubernetes.

[4] Usernetes: Kubernetes without the root privileges — https://github.com/rootless-containers/usernetes.

[5] slirp4netns: User-mode networking for unprivileged network namespaces — `https://github.com/rootless-containers/slirp4netns`.

Иванов Сергей
Обнинск, ООО «Базальт СПО»
`https://gitlab.basealt.space/snowmix/openqa-via-pve`

Интеграция OpenQA с Proxmox Virtual Environment

Аннотация

Доклад посвящён интеграции фреймворка автоматического тестирования OpenQA с системой виртуализации Proxmox Virtual Environment. В докладе рассматривается как сам процесс интеграции, так и работа данных компонентов на примерах тестирования дистрибутивов и пакетов.

Доклад посвящён интеграции фреймворка автоматического тестирования OpenQA с системой виртуализации Proxmox Virtual Environment. В докладе рассматривается как сам процесс интеграции, так и работа данных компонентов на примерах тестирования дистрибутивов и пакетов.

Отдел тестирования компании «Базальт СПО» занимается различными видами проверок, в том числе тестированием дистрибутивов и сборочных заданий для стабильных репозиториев. Тестовые сценарии постоянно улучшаются, их количество разрастается, расширяется тестовое покрытие. Этот процесс ведёт к увеличению трудозатрат на тестирование и повышает вероятность появления ошибки со стороны тестировщика (так называемый «человеческий фактор»). Самым оптимальным решением в данной ситуации является автоматизация части рутинных тестовых сценариев.

В отделе тестирования уже разработана инфраструктура для автоматической проверки сборочных заданий, что помогает сократить время выполнения задач. О данном решении было рассказано на прошлой конференции в рамках доклада «Автоматизация процессов в рамках тестирования сборочных заданий для стабильных репозиториев ОС ALT Linux». Однако в разработанной инфраструктуре присутствует недостаток: система не способна тестировать графические приложения. Автоматизировано лишь тестирование приложений, работу которых можно проверить через консоль.

Данный недостаток решается использованием такого инструмента как OpenQA — открытого фреймворка, позволяющего в полностью автоматическом режиме проводить тестирование различных компонентов операционной системы. Данный фреймворк уже используется в отделе тестирования для проверки установки дистрибутивов (про это было также рассказано на конференции в 2022 году). Кроме того, OpenQA задействуется для проверки сборочных заданий, содержащих пакеты, связанные с установщиком системы. В настоящий момент активно идёт внедрение возможностей OpenQA для тестирования остального функционала сборочных заданий, а не только компонентов, касающихся установки.

OpenQA состоит из нескольких компонентов, одним из который является OpenQA-Worker. По умолчанию OpenQA запускает тестируемый образ на виртуальных машинах qemu, на которых выполняются различные тесты.

В свою очередь, каждая виртуальная машина имеет свои характеристики, (по умолчанию это 25GB HDD, 2GB RAM).

Запуск большого количества параллельно работающих тестов вынуждает использовать под OpenQA-Worker отдельный высокопроизводительный сервер.

Кроме того, в случае возникновения каких-либо ошибок в процессе выполнения OpenQA-тестов виртуальные машины, на которых упали тесты, удаляются. Это вынуждает вручную разворачивать машины для воспроизведения / исправления возникших ошибок.

Обозначенные проблемы можно решить использованием системы виртуализации Proxmox Virtual Environment вместо отдельного сервера. В OpenQA, помимо бекендов, работающих напрямую со средствами виртуализации (qemu, libvirt, hyperv, virtualbox), существуют бекенды, подключающиеся по VNC / SSH к уже существующим виртуальным / реальным машинам, но не осуществляющие их менеджмент (создание, удаление, настройка и т. п.).

Вследствие этого необходима разработка программной обвязки, осуществляющая создание виртуальных машин на PVE и запуск OpenQA-тестов, которые будут подключаться к созданным машинам и работать с ними, так, как если бы использовался бекенд qemu.

Подробности данной интеграции будет рассмотрены в текущем докладе.

Литература

[1] Документация Proxmox VE API, `https://pve.proxmox.com/pve-docs/api-viewer`

[2] Документация openQA, `https://open.qa/docs/`

[3] Документация os-autoinst, `https://github.com/os-autoinst/os-autoinst/tree/master/doc`

Михаил Шигорин
Москва, ООО «Базальт СПО»

Вид с «Эльбруса»

Аннотация

Обзор состояния аппаратной и программной части платформы «Эльбрус», – включая пять лет повседневного применения по работе и год – дома. Достигнутое в настоящем, виды и планы на будущее, виртуальные и реальные проблемы и возможности.

Железо

Как известно, МЦСТ начинали со взаимодействия с Sun Microsystems и соучастия в разработке SPARC; эта линейка микропроцессоров продолжается и по сей день в виде R2000.

Основное же внимание уделим именно отечественной архитектуре «Эльбрус» — именно поддержкой её в ОС Альт и занимаюсь с 2015 года.

Софт

В девяностых годах на SPARC-процессорах естественным образом стали применять ОС Solaris (2.5.1 на e90), но уже в ранних нулевых начали работы по переносу ядра Linux 2.4, что в итоге вылилось в наработанный к «Эльбрус-2С+» технологический задел на Linux 2.6 [1] и создание либо перенос ряда линуксовых дистрибутивов.

Опять же, по своей деятельности говорить буду про «Альт».

Люди

Создатели микропроцессора «Эльбрус» — МЦСТ [2]; в работах по вычислительным комплексам принимает участие ИНЭУМ [3]; в переносе критического системного ПО, затрагивающего кодогенерацию — Унипро [4]; во внедрениях — Эльбрус-2000 [5]. Впоследствии приняли участие и многие другие компании, но эти — изначальные.

Поскольку как начали «Эльбрус», так и спасли от IBM в шестидесятых-семидесятых годах да вытянули уже в девяностых-нулевых военные, соответствующие интересы и особенности неизбежно оказыва-

ют влияние до сих пор на более или менее все аспекты, связанные с данной разработкой.

Сообщество же пошло формироваться, пожалуй, «от новостей». Сперва люди заинтересовывались публикациями о разработке отечественного процессора и первых практических результатах (пусть и недоступных на предмет «пощупать руками»). Затем начали появляться профильные вопросы на форумах по программированию. И с 2014–2015 года, когда организациям стало возможно приобрести рабочую станцию «Эльбрус 401-PC», не будучи в контуре ОПК — пошли первые результаты тестов [7], обсуждения применимости, патчи в апстримы и т. д.

Сообщество

В первую очередь, при таких вводных, естественно, стало формироваться русскоязычное сообщество: поскольку и новости выходили в оригинале по-русски, и гражданские организации, где начали появляться машины – тоже российские. При этом первой известной мне публичной «точкой кристаллизации» оказался Яндекс-музей, где усилиями одного из энтузиастов началось освоение 801-PC с целью переноса и запуска игрушек.

Наблюдается интерес и из-за рубежа — с вопросами от возможности приобретения (пока увы) до соучастия в переносе программ (а вот с этим уже получше).

Взаимодействие столь же естественно начало складываться двусторонним — наработанные патчи стали отправлять в соответствующие проекты [8].

Песочница

Пожалуй, наиболее удачным экспериментом в области собирания сообщества пользователей и разработчиков ПО на архитектуре «Эльбрус» стоит признать доступный желающим стенд ИНЭУМ, созданный усилиями Игоря Молчанова и на сентябрь 2023 года насчитывающий около полутысячи аккаунтов; для обсуждения (желательно технического) и анонсов работ по инфраструктуре (включая и важные обновления системного ПО, например, новые версии компилятора) применяется чат в telegram.

Альтовое

Попытки формирования сообщества пользователей e2k-alt-linux успехом скорее не увенчались в силу специфики организаций и недостаточности уделяемого вопросу внимания: как правило, сообщения в непубличную рассылку принимаются к сведению без обсуждения там же; основным полезным направлением оказался тематический раздел в рамках Альт-вики [9].

OpenE2K

Потихоньку собираются люди около проекта переноса эмулятора qemu на платформу e2k [10], который не стал ограничиваться наработанным qemu-user-e2k, но пошёл «вширь».

Домашнее

Также интересны случаи применения систем «Эльбрус» в качестве домашнего компьютера, каковых известно уже больше десятка. Мой начался в 2020 году («одомашнивание» стенда), а в штатный режим перешёл в 2022 [11]. На данный момент можно обобщить так: владельцы домашних «Эльбрусов» — достаточно опытные администраторы/разработчики, основной же проблемой является браузер (обычно это стареющий firefox).

Виды

Железо

Судя по публикуемым госзакупкам, на «Эльбрус-2С3» надеяться вполне можно. Это достаточно мощный процессор текущей версии микроархитектуры для ознакомления и нетяжёлых задач (два ядра по два гигагерца), поддерживающий виртуализацию и PCIe gen3 [12]. Более мощные многопроцессорные e2k-системы сейчас простому человеку доступны скорее через стенд ИНЭУМ.

Софт

Существующие инициативы «снизу» сами по себе хороши, но разрознены и упираются в фундаментальные ограничения вроде фор-

мальной закрытости системы команд даже при наличии хотя бы удалённого доступа.

При возможности перевода проекта в открытый режим взаимодействия хотелось бы избежать типичных проблем:

1. «выкидывание через забор»;
2. архивы вместо гитов;
3. отсутствие обратной связи;
4. форки вместо соучастия.

И, соответственно, прийти к известной обкатанной технической инфраструктуре [13]:

1. репозитории разработки кода (git);
2. чат для быстрых обсуждений (telegram);
3. почтовые рассылки для обсуждения по существу (mailman);
4. отслеживание конкретных ошибок (bugzilla);
5. накопление и систематизация опыта (wiki).

с последующим выходом на масштабируемый рабочий режим из хорошо знакомого «пилу точить некогда, пилить надо».

Люди

Несмотря на многолетнюю и подчас некорректную критику проекта со стороны — людей, заинтересованных в их развитии, больше, чем проекты разработки способны «переварить» в плане приёма в команду и становления на крыло. Готовиться к этой проблеме также стоит загодя, сейчас эту роль по большей части выполняют опубликованная документация разработчика [15] и общественно доступные стенды [16].

Но мы точно можем лучше сообща [17][18][19].

Литература

[1] [Электр. ресурс]: `http://osday.ru/semenihin.html`

[2] [Электр. ресурс]: `http://mcst.ru`

[3] [Электр. ресурс]: `http://ineum.ru`

[4] [Электр. ресурс]: `http://unipro.ru`

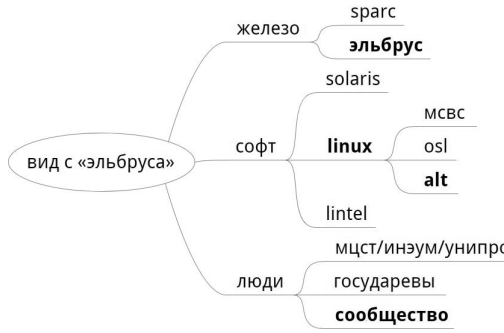

Рис. 1: Вид с «Эльбруса»

[5] [Электр. ресурс]: http://elbrus2000.ru

[6] [Электр. ресурс]: http://t.me/e2k_chat

[7] [Электр. ресурс]: http://altlinux.org/Эльбрус/тесты

[8] [Электр. ресурс]: http://altlinux.org/Эльбрус/upstream

[9] [Электр. ресурс]: http://altlinux.org/elbrus

[10] [Электр. ресурс]: http://git.mentality.rip/OpenE2K/qemu-e2k

[11] [Электр. ресурс]: http://sdelanounas.ru/blogs/149691/

[12] [Электр. ресурс]: http://mcst.ru/elbrus-2c3

[13] [Электр. ресурс]: http://producingoss.com/ru/technical-infrastructure.html

[14] [Электр. ресурс]: http://0x1.tv/2a9-lets-cross-forks-shigorin

[15] [Электр. ресурс]: http://mcst.ru/elbrus_prog

[16] [Электр. ресурс]: http://elbrus.kurisa.ch

[17] [Электр. ресурс]: http://github.com/ilyakurdyukov/e2k-ports

[18] [Электр. ресурс]: http://gitflic.ru/user/erthink

[19] [Электр. ресурс]: http://gitflic.ru/project/e2khome/lccrt

Игорь Молчанов
Москва, МФТИ (НИУ), ИНЭУМ им. И. С. Брука

Проект: meson, kexec-e2k и др. https://github.com/mesonbuild/meson, https://github.com/makise-homura/kexec-e2k, и др.

Разработка, портирование и тестирование СПО для платформы «Эльбрус»

Аннотация

В работе описывается текущая ситуация с наличием СПО для платформы «Эльбрус», приводятся примеры таких проектов, обсуждаются проблемы портирования ПО и поддержки его инфраструктуры, а также взаимодействие с апстримами проектов и роль энтузиастов в данных процессах.

В последнее десятилетие в мире наблюдается тенденция повышения популярности свободного программного обеспечения (СПО). В частности, с января 2013 по январь 2023 года доля GNU/Linux среди операционных систем настольных компьютеров выросла с 0,88% до 2,91% [1]. Эта тенденция особенно важна для систем с архитектурами процессора, отличными от x86_64: ассортимент доступного программного обеспечения для них значительно меньше, чем для x86_64, в особенности для архитектур, не известных широко во всём мире. Поскольку возможность портирования проприетарного ПО на такие архитектуры разработчиком этого ПО маловероятна. А разработчиком архитектуры — как правило, вообще невозможна. Однако возможность использования ПО с открытым кодом, в частности, СПО, на таких архитектурах принципиальна.

При этом для стабильного развития культуры портирования СПО важно при работе с разработчиками и энтузиастами уходить от сложившейся в России практики нелицензионного использования такого СПО [2] (в частности, невыдачи исходного кода производного проекта по запросу пользователя или линковки свободных библиотек с несвободным кодом). В работе обсуждаются некоторые такие проекты, портированные автором на платформу «Эльбрус», а также вспомогательные проекты, разработанные (полностью или частично) автором под данную платформу и не только, которые доступны под свободными лицензиями, и описываются проблемы, возникшие при разработке/портировании и способы их преодоления, в том числе с

помощью тестирования, а также сложности интеграции описанного ПО в дистрибутивы существующих для платформы операционных систем.

В частности, в работе обсуждаются:

- CMake (`https://gitlab.kitware.com/cmake/cmake/`), meson (`https://github.com/mesonbuild/meson`) — системы сборки;
- Taisei Project (`https://github.com/taisei-project/taisei`), GZDoom (`https://github.com/ZDoom/gzdoom/`), wadtools (`https://github.com/makise-homura/wadtools`) и др. — игры, игровые движки и утилиты;
- LAL Suite (`https://git.ligo.org/lscsoft/lalsuite/`), BOINC (`https://github.com/BOINC/boinc`), Milkyway@home (`https://github.com/Milkyway-at-home/milkywayathome\_client`) — научное ПО;
- Koishi (`https://github.com/taisei-project/koishi`), SIMDe (`https://github.com/simd-everywhere/simde/`) — вспомогательные библиотеки;
- REIMU (`https://github.com/makise-homura/openbmc/`), mcst-fruid (`https://github.com/makise-homura/mcst-fruid`), kexec-e2k (`https://github.com/makise-homura/kexec-e2k`) — программы управления;
- nemuno (`https://github.com/makise-homura/nemuno-bot`) — система организации ssh-аккаунтов на серверной инфраструктуре;
- и другие утилиты, разработанные или адаптированные автором.

Также рассмотрены дополнительные вопросы, позволяющие разработчикам получить доступ к машинам на базе платформы «Эльбрус» и участвовать в разработке и портировании ПО.

Литература

[1] Desktop Operating System Market Share Worldwide Jan 2013 - Jan 2023 / StatCounter GlobalStats // StatCounter.com, 2023 [Электронный ресурс] URL: `https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-201301-202301`, дата обращения: 5.09.2023

[2] Артём Сафьянников. Правовые проблемы использования открытого программного обеспечения (open source) / Журнал Суда по интеллектуальным правам // Garant.ru, 2022 [Электронный ресурс] URL: https://www.garant.ru/article/1555428/, дата обращения: 5.09.2023

Игорь Воронин, Евгений Коцюба, Ростислав Воронин
Шатура, Моск. обл., ИПЛИТ РАН, ИПЛИТ РАН, МГТУ им. Баумана

Использование СПО Home Assistant в среде Alt Linux для автоматизации хозяйственной деятельности организации

Аннотация

В докладе разбираются варианты использования СПО Home Assistant в различных конфигурациях установки для решения задач автоматизации учёта и мониторинга коммунальных услуг промышленных предприятий, научных организаций и частных домов.

Home Assistant (HA) — это одна из самых популярных систем организации, автоматизации и управления устройствами умного дома и интернета вещей, объединяющая в себе автоматический сбор данных от устройств большого количества различных брендов. Система написана на Python и имеет открытый исходный код [1].

Возможности HA

Отличительными особенностями HA являются:

- наличие большого набора готовых вариантов визуализации данных;
- удобный конструктор карточек для визуализации;
- возможность интеграции большого количества датчиков и устройств, реализованная через дополнения, устанавливаемые как из официального магазина дополнений, так и созданных сообществом;
- возможность и удобство интеграции в HA датчиков собственной разработки и устройств посредством использования протокола MQTT;

- наличие постоянно обновляемой официальной документации;
- активное сообщество в Телеграм-каналах и форумах, помогающее решить возникающие вопросы;
- возможность работы на разнообразных аппаратных платформах;
- свободная лицензия и отсутствие затрат на закупку ПО.

Указанные особенности позволяют адаптировать и использовать HA не только для систем автоматизации умного дома, но и для автоматизации хозяйственной деятельности организаций.

Установка HA

Возможны различные варианты установки HA.

- Core — самый минимальный функционал, все приложения, дополнения и интеграции нужно устанавливать отдельно, вручную.
- Supervised — в этом варианте установки из WEB-интерфейса приложения возможно установить максимальный набор приложений.
- HAOS — система устанавливается на чистое железо, как ОС, включая и загрузчик.
- Возможна установка в окружении Python — vnev.
- Возможна установка в контейнере Docker или Docker-composer.
- Установка HAOS в виртуальной машине VirtualBox.

Как выглядит реально работающая система в режиме CORE, можно посмотреть по ссылке [2] логин/пароль для входа гостем: guest/guest.

Кроме процессора AMD/INTEL HA можно устанавливать также и на одноплатный компьютер с процессором архитектуры ARM, разбор нюансов установки произведён в статье WIKI [3].

Особенности установки HA в VirtualBox

Установка в VirtualBox на AltLinux имеет некоторые особенности. На странице загрузки HAOS предлагается для скачивания образ диска в формате VirtualBox (.vdi). Однако версия VirtualBox из репозитория AltLinux немного отстаёт от актуальной версии от Oracle, что

выражается в том, что образ диска .vdi не воспринимается данной версией VirtualBox. Однако, если использовать инструменты VirtualBox на компьютере с установленной актуальной версией (например, под Windows или Ubuntu), то можно сконвертировать виртуальный носитель из формата .vdi в формат .vdd (Файл — Инструменты — менеджер виртуальных носителей), перенести .vdd файл на AltLinux и далее из альтлинуксового VirtualBox'а с помощью такого же инструмента сконвертировать .vdd в .vdi.

Пример использования HA для автоматизации хозяйственной деятельности организации

В хозяйственной деятельности организации требуется контролировать и вести учёт ряда коммунальных ресурсов. Обычно это электроэнергия, тепло и водоснабжение. Датчики, которые учитывают эти ресурсы, могут иметь подключение к локальной сети передачи данных, однако значительное число таких датчиков имеют простой выход на последовательный порт, и в этом случае для передачи данных необходимо подключение отдельного контроллера, который может служить шлюзом для передачи данных от устройства до сервера сбора данных.

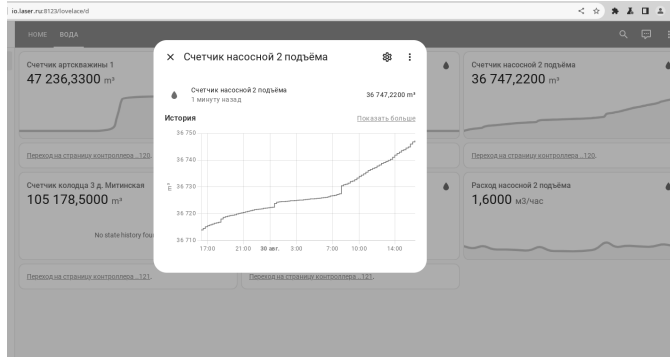

Рис. 1: Счетчик насоса подъёма воды.

В качестве примера автоматизации рассмотрим подключение счётчиков воды от артезианской скважины, имеющейся на территории

организации. Счётчики используются как для учёта расхода воды внутри организации, в том числе для своевременного обнаружения возможных протечек, так и для организации расчётов с внешними потребителями и для отчётов в контролирующие органы.

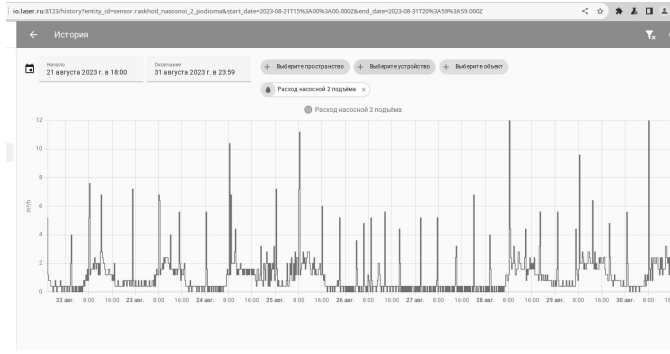

Рис. 2: Посуточный расход воды за 10 дней августа.

В данном случае сбор данных организован по локальной сети, путём отправки периодических запросов от сервера с ОС Альт на контроллеры счётчиков с WEB интерфейсом. Данные собираются запросом CURL с заданной периодичностью и отправляются с использованием mosquitto клиентом в заданные топики HA. Пример использованных команд приведён ниже:

Контроллер счётчиков находится по адресу: `http://192.168.1.120`, хост с HA: `http://192.168.1.2`

По запросу `http://192.168.1.120/inf.cgi` контроллер отдаёт страницу такого содержания:

```
info...
UART0 Ok
47341.130 46683.580 36868.720 1.800
```

На этой странице ответа в третьей строке указаны значения 4-х счётчиков.

Забираем показания счётчиков командой:

```
$ curl -s -m 15 --connect-timeout 12 http://192.168.1.120/inf.cgi>/tmp/120.txt
```

Далее скриптом bash с использованием команды awk из файла /tmp/120.txt парсим значение прокачанной воды для двух счётчиков, это 47341.130 и 46683.580.

Передаём эти значения в топики mosquitto HA:

```
$ mosquitto_pub -h 192.168.1.2  -u user -P pwd -t water/s0  -m 47341.130
$ mosquitto_pub -h 192.168.1.2  -u user -P pwd -t water/s1  -m 46683.580
```

Для приёма данных на стороне HA необходимо выполнить интеграцию с mosquitto, а в конфигурационном файле HA достаточно указать раздел сенсора mqtt:

```
- sensor:
    name: "Счетчик артскважины 1"
    unique_id: water0s
    device_class: water
    state_topic: "water/s0"
    unit_of_measurement: "m3"
```

Аналогично можно организовать сбор и отображение данных с датчиков температуры, влажности, теплосчетчиков, счётчиков электроэнергии, управление освещением, что может быть полезным и актуальным, например, для агротепличного хозяйства.

Если жилой дом снабжён котлом газового отопление с протоколом OpenTherm, то установка контроллера с Home Assistant позволит мониторить затраты на отопление в реальном времени и получать мгновенные сообщения о возникновении нештатных ситуаций.

Литература

[1] Официальный сайт Home Assistant, [Электронный ресурс]: `https://www.home-assistant.io/`

[2] Демонстрация работы (вход гостем guest/guest), [Электронный ресурс]: `http://wiki.umki-kit.ru:8123/`

[3] Установка приложения, [Электронный ресурс]: `http://wiki.umki-kit.ru/wiki/index.php/Установка_Home_Assistant_на_AltLinux`

[4] телеграм-канал сообщества, [Электронный ресурс]: `https://t.me/homassistant`

Александр Кардаполов
Екатеринбург, ООО «Real Time Intelligence»

Ad-hoc мониторинг в режиме реального времени

Аннотация

Рассматриваются подходы к решению задач мониторинга информационных систем с использованием Low-code, хранения данных в колоночной базе данных и продвинутой визуализацией метрик в режиме реального времени с помощью специализированного приложения.

Проблема

Развитие информационных систем в XXI веке характеризуется ростом сложности приложений и инфраструктуры, объёмов обрабатываемых данных и повышенными требованиями к скорости получения результата:

- Программный и инфраструктурный код, который используется для построения программно-аппаратных комплексов, усложняется, появляются всё новые абстракции, которые используются без понимания того, как оно устроено внутри, в виде «черного ящика». Это приводит к росту количества человеческих ошибок при разработке и сопровождении программного обеспечения (багов). Например, широкое распространение получила практика управления сложностью путём декомпозиции сложных систем (монолитов) на более простые части (микросервисы). Но и она не решает всех проблем — управлять системой из десятков, сотен и даже тысяч небольших компонентов тоже становится непростой задачей.
- С каждым годом, информационные системы создают всё больший поток данных, которые необходимы для принятия решений в различных сферах человеческой деятельности. Для примера можно взять рост безналичных платежей за последние 10 лет. Больше данных теперь требуется и для обеспечения мониторинга программно-аппаратных комплексов, сбора трассировок и журналов, мониторинга безопасности, анализа бизнес-метрик и т. п. Всё это требует особой культуры работы с данными, когда управленческие решения принимаются на базе цифр и расчётов,

а не на интуиции и личном опыте — так называемый подход «Data-driven».

- Хорошо знакомое «Time To Market» в последнее время повышает требования и к информационными технологиям, когда поставка информации для принятия решения требуется прямо сейчас, в режиме реального времени. В нашем случае скорость получения результата — «Time To Result», становится критически важным, чтобы оперативно и точно реагировать на изменившиеся условия, снижать время реакции на инциденты и т. п. Для примера, как один из принципов манифеста гибкой разработки «Agile» — частая поставка работающего приложения.

Решение

В связи с вышеизложенным возникают сложности с общим пониманием того, как устроена система в целом, а не на уровне отдельного сегмента, и как они взаимосвязаны. Большое количество различных компонентов систем, выполняющих специализированные функции, требуют значительного времени на понимание принципов работы программно-аппаратных комплексов.

Широкое распространение получают практики объединения навыков и компетенций смежных специальностей. Например, на стыке разработки, администрирования (DevOps) и безопасности (DevSecOps).

Всё это приводит к росту когнитивной нагрузки на инженеров сопровождения, разработки, тестирования и аналитики, и как следствие — повышенной вероятности совершить ошибку.

Поэтому актуальным становится использование инструментов, которые позволят оперативно получить максимально полные данные о работе системы без необходимости погружаться в рутину по сбору данных, их обработке и компактному хранению, настройке визуализации и анализу.

Подобные приложения востребованы, когда нужна высокая скорость получения информации о собранных данных в удобном для пользователя виде и в режиме реального времени. Когда есть возможность гибкой настройки сбора данных по большинству ключевых показателей, на базе которых можно провести не только real-time анализ, но и получить ретроспективную аналитику за предыдущие периоды наблюдений.

Реализация

Real Time Intelligence Desktop — это настольное Java приложение, предназначенное для сбора, хранения, визуализации и анализа данных в режиме реального времени.

- *Сбор данных* — реализован безагентный сбор данных с использованием собственной системы pipeline-ов. Поддерживается сбор данных как самим приложением, так и получение данных, собранных внешним агентом на стороне БД. Настраивается интервал сбора данных с внешнего источника, SQL-запрос и указывается поле данных для хранения меток времени (timestamp) в локальной базе данных.
- *Хранение и обработка данных* – используется встроенная БД блочно-колоночного типа для хранения данных временных рядов FBASE. Для оптимизации хранения данных используется концепция локального индексирования, когда решение об использовании того или иного типа хранения (RAW, ENUM или HISTOGRAM) принимается по распределению данных в собираемых наборах данных в автоматическом режиме.
- *Визуализация данных* – используется открытый пакет JFreeChart для отображения графиков и отчётов. Доступен просмотр исторических данных, реализован быстрый доступ к визуализации по выбранной метрике и временному диапазону, доступ к истории выполнения запросов, возможность визуального представления результатов поиска по подстроке с группировкой по столбцам таблицы, постраничное получение данных для raw-данных.

Михаил Чекан
Иркутск, ООО «Полюс-НТ»

Проект: Lapki IDE https://github.com/kruzhok-team/lapki-client

Среда визуального программирования машин состояний

Аннотация

Доклад посвящён разработке среды программирования, создаваемой в рамках национальной киберфизической платформы. Целью среды является снижение входного порога разработки киберфизических систем за счёт применения парадигмы иерархических машин состояний. На текущий момент позволяет разрабатывать схемы для платы Arduino Uno, ведётся работа по созданию API для других платформ.

Киберфизическая система — это комплексная система физических элементов и их цифровых двойников в вычислительном слое управления, которая постоянно получает данные из окружающей их среды и использует их для оптимизации процессов достижения установленных целей [1]. Так, киберфизика охватывает собой широкий спектр задач от программирования отдельных электронных устройств до проектирования «умных» инфраструктур. Формирование в России национальной киберфизической платформы (НКФП) вызвано прежде всего потребностью в инженерных кадрах, способных проектировать и создавать микроэлектронные и киберфизические системы [2]. Участники НКФП решают эту задачу через популяризацию технического творчества и технологического образования в сфере микроэлектроники и киберфизики, создавая траекторию от школьной программы и технологических кружков до профильных курсов и решения промышленных задач. Первым массовым решением в рамках НКФП стала платформа «Берлога» [3], представляющая собой набор образовательных игр, затрагивающих программирование, существующих в едином сеттинге и интегрированных в реальную жизнь и образование через платформу «Талант» «Кружкового движения». В технологической плоскости НКФП разрабатываются новые стандарты и инструменты, призванные поднять качество создаваемых систем и доступность их создания более широкому кругу пользователей. Одним из таких инструментов положена графическая среда программирования киберфизических систем, основанная на диаграммах иерархических

машин состояний. Эта среда под названием Lapki IDE активно разрабатывается студенческой командой с мая 2023 года под руководством автора. В конце августа 2023 года опубликована альфа-версия среды под лицензией GNU GPL версии 3.

На текущий момент Lapki IDE рассчитана на разработку схем поведения юнитов для игры «Защита пасеки» в жанре Tower Defence на платформе «Берлога», а также прошивок для платы Arduino Uno. В дальнейшем планируется расширение списка платформ, в частности, основанными на STM32 и nRF52833. Также запланирована поддержка симулятора космических аппаратов «Орбита» [4], используемого в профиле «Спутниковые системы» Национальной технологической олимпиады. В этих целях ведётся работа по формированию API плагинов для модуля компилятора.

Основополагающей чертой Lapki IDE является визуальная парадигма программирования на основе расширенных иерархических машин состояний (РИМС). Это промышленный стандарт программирования встраиваемых систем, основанный на UML и широко применяемый в устройствах от бытовой до космической сферы. За рубежом парадигма РИМС распространена благодаря М. Самеку [5] и его компании Quantum Leaps. В отечественной среде существует родственная парадигма автоматного программирования [6]. Применение РИМС обеспечивает низкий порог входа в создание систем для начинающих разработчиков, предоставляя естественную для аппаратных систем событийно-реактивную логику управления. Продвинутым же пользователям такая парадигма даёт возможность создавать сложнейшие алгоритмы, сохраняя простую и прозрачную визуальную структуру.

В проекте Lapki IDE закладывается ряд принципов, способствующих более быстрому входу в разработку, и при этом повышению осознанности и понимания процесса разработки. Так, принцип «открытого капота» декларирует прозрачность разработки на всех этапах. Это достигается вынесением операций над платформой (будь то этапы компиляции или прошивка устройства) в поле прямого доступа пользователя, а также снабжением интерфейсов настройки, выходных журналов и артефактов «переводом» на более доступный неспециалистам язык. Принцип «разработка в состоянии потока» обеспечивает комфортное пребывание в desktop-среде за счёт тщательной проработки пользовательского опыта с целью сократить число лишних и контринтуитивных действий. Принцип «модульность и трансоблачность» определяет архитектуру IDE как клиентское приложение

и набор сопровождающих его модулей, размещаемых вместе с клиентом или на удалённом сервере. Размещение в облаке модуля компиляции позволит распределённой команде производить сборку выходного файла прошивки в одинаковых воспроизводимых условиях, а также снизить требования к рабочей станции. Вынесение модуля загрузчика также позволяет облегчить процесс разработки, убирая необходимость каждому разработчику владеть прошиваемым устройством.

В настоящий момент опубликованы альфа-версии четырёх компонентов: клиент с редактором РИМС; модуль компилятора, выполняющий сборку прошивок под Arduino Uno и трансляцию схем в формат игры «Защита пасеки»; загрузчик прошивок для Arduino Uno и оснастка для сервера документации. Ведётся документирование и доработка модулей, формируется программный интерфейс для подключения новых целевых платформ.

Литература

[1] Громаков Е. И., Сидорова А. А., Современные технологии. Киберфизические системы, Изд-во Томского политехнического университета, 2021

[2] Кружковое движение, Владимир Путин поддержал запуск Национальной киберфизической платформы «Восток», [Электронный ресурс]: `https://team.kruzhok.org/news/post/vladimir-putin-podderzhal`

[3] Национальная киберфизическая платформа,[Электронный ресурс]: `https://platform.kruzhok.org`

[4] Образовательная игра «Орбита», [Электронный ресурс]: `http://orbitagame.ru`

[5] Samek M., Practical UML statecharts in C/C++: event-driven programming for embedded systems, Elsevier Inc., 2009

[6] Поликарпова Н. И., Шалыто А. А., Автоматное программирование, Изд-во «Питер», 2009

Алексей Федорчук
Москва, Независимый автор

Последний Linux сочинителя

Книга написана для себя — в ознаменование двадцати пяти лет использования СПО, основанного на ОС Linux, при сочинении и редподготовке нарративных текстов научного и технического содержания, а также публицистики, беллетристики, мемуаристики и прочей «лирики». Она разделяется на три части.

Первая часть посвящена тенденциям современного Linux'а, затрагивающим сочинителя: позитивным, негативным и нейтральным. Первых — мало, и о них редко вспоминают. Вторых — не больше, но они вызывают много разговоров. Третьих — подавляющее большинство, но их обычно считают либо (редко) позитивными, либо (чаще) негативными. Хотя на самом деле они сочинителю до... забора дверца.

Поэтому важно различать тенденции позитивные, дабы употреблять их себе на пользу, и негативные, чтобы по возможности им противостоять, а всем прочим просто не забивать себе мозги. На этом и основывается выбор трёх основных составляющих рабочей среды сочинителя — графического окружения, оболочки CLI, дистрибутива. Причём именно в этом порядке:

- с дистрибутивом сочинитель имеет дело непосредственно при установке системы, её первичной настройке и комплектации пакетами, в дальнейшем лишь следит за целостностью и обновлением;
- оболочка CLI необходима, когда требуется быстрота выполнения, и лишь при должном навыке — в отсутствии оного эффективней использовать GUI-эквиваленты;
- в графическом окружении же, сиречь десктопе, сочинитель целиком проводит своё рабочее время — и этим всё сказано.

Выбор десктопа (KDE) был предопределён (с этого два года назад и начался данный проект), оболочке Zsh я не видел альтернатив 20 лет, так что в основном время ушло на знакомство с достижениями современного «дистростроения». Описанию этого процесса в тезисах не место, поэтому итог выбора, которым и заканчивается первая часть

книги: KDE-редакция дистрибутива MX Linux, основанного на стабильной ветке Debian'a. Ему и посвящена вторая часть книги.

Она начинается с общего обзора дистрибутива, в том числе его истории: MX Linux происходит от Mepis'a, активно развивавшегося в нулевых годах XX века, и генетически связан с греческим дистрибутивом antiX (откуда и его имя). В настоящее время он существует в трёх редакциях: с десктопами KDE (только 64-битная версия, ориентированная на мощные машины) и Xfce — для машин «стандартных», и с оконным менеджером Fluxbox — для маломощных конфигураций.

Далее во второй части описывается установка KDE-редакции. В отличие от многочисленных ныне «близнецов-братьев во Calamares'е», инсталлятор MX Linux — собственный и выглядит несколько архаично, но он очень гибок. Во-первых, поддерживается как UEFI BIOS, так и BIOS Legacy, а init-система — как SysV (по умолчанию), так и systemd. Во-вторых, функционал его не оставляет желать лучшего. Так, на стадии инсталляции можно настроить раскладку клавиатуры вплоть до любого её варианта — и она будет унаследована установленной системой. В-третьих, и главных, вне инсталлятора, непосредственно в Live-сеансе, можно выполнить любые настройки и провести полную перекомплектацию прикладных пакетов — и всё это сохранится в установленной системе: особенность, уникальная для современных дистрибутивов. В тех системах, где время от времени появлялись подобные «фишки» (Fedora, openSUSE), они очень быстро и бесследно куда-то исчезали.

Разумеется, любую кастомизацию можно выполнить и в установленной системе, чему посвящена вторая половина второй части. Для чего, кроме репозиториев stable-ветки Debian'a, имеются репозитории проекта, в том числе AHS с пакетами поддержки нового оборудования. Которыми, впрочем, насильно никто пользоваться не заставляет...

Третья часть книги отдана под пользовательские приложения и утилиты. На эту тему можно говорить или очень много, или... ничего. Я избрал компромиссный вариант, ограничившись приложениями, важными для сочинителя. Из которых архиважнейшими в сочинительском ремесле являются браузер и текстовый редактор.

Роль браузера определяется тем, что любому сочинителю, прежде чем стать «чукчей-писателем», надо некоторое время побыть «чукчей-читателем». А поскольку читают нынче в сети, то ему в качестве инструмента для чтения предлагается браузер Vivaldi. На

возражение, что это не СПО, я не скажу ничего, даже не спрошу — «вам в шашечки, или ехать?» А просто пошлю... не туда, куда подумали испорченные мальчики, а почитать программный документ участников проекта: Почему Vivaldi не Open Source проект[1]: кое-что станет яснее.

Тем более что и с шашечками у Vivaldi всё хорошо: супротив него остальные браузеры — что плотник супротив столяра. Это касается и функционала, и его настраиваемости.

А о текстовом редакторе, в амплуа которого выступает Kate[2], в размещённых на сайте отрывках сказано (пока) достаточно.

Андрей Черепанов
Москва, «Базальт СПО»

Вызовы и перспективные направления развития образовательного программного обеспечения

Аннотация

Доклад посвящён актуальным вызовам сопровождения и развития свободного программного обеспечения для образования.

Актуальные вызовы:

- портирование программ на новые языки:
  - переписывание и разработка новых приложений на `Rust`, `Go` и `NodeJS`: можно ли жить без Интернета? Можно, но тяжело. Разработчики на этих платформах подразумевают, что на время сборки должен быть доступ в Интернет для выкачивания модулей. При этом само скачивание часто явно прописано в скриптах и системах сборки (например, в `anki2`). Для обеспечения воспроизводимой сборки в Hasher сеть недоступна и требуется в пакет включать все требуемые модули (забандливать).
  - проблемы забандливания и неявной сборки во время этой операции. Для `Rust`, `Go` и `NodeJS` требуется забандливать

[1] `https://vivaldi.com/ru/blog/why-isnt-vivaldi-browser-open-source-ru/`
[2] `https://www.alvstory.ru/tag/kate/`

по 500-600 МБ библиотек, часть из которых может начинать собираться во время установки забандленного на хостовую машину перед сборкой в изолированном окружении (например, `pgadmin4`). При этом не факт, что забандленные модули соберутся в изолированной среде.

  - `C#` — есть ли перспективы? `Mono` фактически не поддерживается, а `dotnet` предоставляет только базовые библиотеки без GUI. Проекты типа `Avalonia UI` находятся в зачаточном состоянии и не позволяют создавать полноценные графические приложения. Разработчики на `C#` в этой ситуации начинают делать веб-интерфейсы, превращаемые в десктопные приложения с помощью `Electron` с диким оверхедом.

- политические и юридические аспекты:
  - ограничения доступа к исходному коду и возможности его изменения не позволяют организовать совместную работу и вовлечение отечественных разработчиков в апстримные проекты. Слабое развитие местных проектов СПО.
  - соблюдение местного законодательства в части картографической информации и часовых поясов требует доработку программ, связанных с выбором часовых поясов и показа карт.

Перспективные направления развития:

- увеличение влияния робототехники в процессе обучения, симуляции и управления устройствами. Повышение спроса на `gz-sim`, `webots`, `arduino`. Активное использование программного обеспечения для 3D-печати, ограниченное доступностью устройств;
- опциональная установка LibreOffice для возможности использования вместо него WPS Office, OnlyOffice, МойОфис и R7-Офис;
- упрощение использования стороннего ПО. Доработка GUI для новых версий `epm play`, разработка плагина к `epm play` для `gnome-software` и, возможно, `discover`.
- разворачивание сетевых служб: расширение перечня служб при разворачивании локальным `ansible`, возможность не только

развернуть, но и донастроить (например, сменить пароль администратора, генерируемый при установке автоматически) и провести процедуру удаления экземпляра службы.

Полина Петруша, Иван Хахаев
Санкт-Петербург, СПб ГЭТУ «ЛЭТИ»

Свободное ПО для образования: реконструкция мышления, репозиционирование, редизайн

Аннотация

Изменения рынка российского программного обеспечения, связанные с необходимостью массового перехода на отечественное программное обеспечение, включая свободное, открывает новые перспективы для российских компаний, разработчиков и интеграторов свободного ПО и ставит перед ними серьёзные задачи. В первую очередь, требуется изменить мышление поставщиков продуктов с технологически сфокусированного (technical-centered) на ориентированное на пользователя (user-centered) и сфокусированное на задачи (solution-focused). Из изменения мышления логически следует изменение позиционирования продуктов свободного ПО, а также изменение их продуктового дизайна. Мы рассмотрим весьма перспективный с точки зрения покупательской ёмкости рынок — рынок свободного ПО для образовательных организаций как пример реконструкции мышления, репозиционирования и редизайна решений на базе свободного ПО.

Барьеры для свободного ПО в образовании

При работе с рынком образования (как и другими «клиентскими» рынками в России) следует исходить из того, что решение о закупке решений для образовательных организаций принимают не столько технические специалисты ИТ-служб (технические заказчики), сколько заказчики от различных подразделений (образовательных, административных, финансовых служб — бизнес-заказчики) исходя из того, насколько данное программное обеспечение решает их задачи. Даже если непосредственно закупкой занимаются технические заказчики, то действуют они исходя из потребностей и заявок бизнес-заказчиков.

Это требует кардинального пересмотра представления информации о возможностях дистрибутивов или пакетов свободного ПО, а именно — переход от технических деталей и продажи дистрибутивов к продаже решений для определённой группы пользователей.

Анализ информации, представленной на сайтах основных российских дистрибутивов в настоящее время, показывает ориентацию на продажу дистрибутивов. Из-за этого формируются барьеры, затрудняющие принятие решений о покупке российского свободного ПО в качестве основного ПО для работы образовательных учреждений.

Барьер 1. Решение для «всех и сразу». Не определены разные группы целевой аудитории (вузы, школы, СПО, дошкольные учреждения), соответственно, не сформулированы предложения, преимущества и функциональный состав продукта для каждой из групп целевой аудитории. В случае дистрибутивов для нескольких групп пользователей отсутствует фокус на то, что данное решение подходит для этих групп. В результате бизнес-пользователям крайне сложно понять, подходит для них данный программный продукт или нет.

Барьер 2. Техноориентированность. В презентации продуктов в первую очередь показаны технические характеристики, но крайне редко и обще («безопасно», «удобно», «легко») показаны выгоды для функциональных специалистов (сотрудников учебного департамента, офиса по цифровизации, проектных офисов). Соответственно, в описании продукта идёт концентрация на технической информации, в ущерб информации о функциях продукта, важных для лиц принимающих решения. Таким образом, техническим специалистам становится крайне сложно обосновать необходимость закупки именно этого продукта.

Барьер 3. Труднодоступность информации о составе ПО. Информация о составе компонентов продукта для разных групп целевой аудитории, о функциональных возможностях компонентов продукта расположена крайне неочевидно и труднодоступна. Отсутствует информация о типовых use-case для разных компонентов одного функционального назначения, что тоже усложняет принятие и обоснование решения для покупки как у технических специалистов, так и у лиц принимающих решения.

Барьер 4. Непрозрачное ценообразование. Нет возможности быстро получить информацию о цене дистрибутива для корпоративных пользователей. Помимо дополнительных усилий по поиску этой

информации, затрудняется ещё и мониторинг со стороны отделов закупок.

Предлагаемые решения по преодолению барьеров

Для того, чтобы облегчить принятие решений о приобретении со стороны бизнес-пользователей предлагаются следующие рекомендации по репозиционированию и редизайну программных решений на базе свободного ПО для образовательных учреждений.

- **По Барьеру 1:** Определить группы пользователей и сфокусировать решения под их задачи.

  Создать экосистему продуктов для различных уровней образования с чётким разделением продуктов/пакетов: Детский сад, Школа (без разделения на начальную, общую), Колледж, Университет. По каждому продукту сделать отдельный презентационный материал (страницы сайта, презентации, листовки) и там перечислить состав продукта, решаемые им задачи, все достоинства и ограничения.

- **По Барьеру 2:** Предоставить информацию, понятную для лиц принимающих/влияющих на принятия решения.

  В презентационных материалах рекомендуется описывать не технические детали, а решения для пользователей. Не «Поддержка периферийного оборудования, в том числе интерактивного», а «Возможность работы с интерактивными досками в классе», не «Возможность выбора рабочего окружения (KDE или XFCE)», а «Выбор рабочей среды в зависимости от предпочтений пользователя».

- **По Барьеру 3:** Расположить информацию о составе и функциях пакетов очевидным для пользователя, понятным, читаемым, легко доступным и легко индексируемым образом.

  Создать описание «от решений». Например: решения для инженерного проектирования, решения для оснащения мультимедийного класса под ключ. В каждом решении добавить краткое описание продуктов и их различие (если в пакете 2 одинаковых продукта). Буквально по 1–2 строчки на продукт.

- **По Барьеру 4:** Обозначить ориентировочные, базовые цены рядом с описанием продукта. В презентациях, на страницах программных пакетов рекомендуется разместить информацию по

ценам на пакеты (стоимость за лицензию, пакет, или иные варианты цены). Это облегчит мониторинг служб закупок, а также позволить в процессе подготовки технического задания бизнес- и техническими заказчиками обосновать целесообразность закупки у того или иного поставщика.

Алексей Федосеев
Москва, Ассоциация участников технологических кружков

Свободный симулятор космических аппаратов «Орбита». Опыт организации школьных и студенческих стажировок в целях развития проекта

Аннотация

С 2022 года в России ежегодно проводятся стажировки «Код для всех», в которых школьники и студенты включаются в существующие команды и проекты по созданию свободных программ. Одним из проектов СПО в рамках стажировки этого года стал космический симулятор «Орбита», который уже 10 лет используется в кружках по космонавтике, но был опубликован под свободной лицензией только в 2023 году. Симулятор «Орбита» позволяет спроектировать космический аппарат, который должен решить поставленную прикладную задачу на орбите. Космический аппарат конструируется в виде набора общих параметров и выбора подсистем, включая также программу полёта, написанную на языке Python или с использованием диаграмм иерархических машин состояний. В рамках стажировки «Код для всех» в развитие симулятора включились 5 школьников и студентов, работа которых была направлена на создание графического интерфейса симулятора, развитие его миссий и доработку документации. В статье рассматриваются опыт и результаты стажировки, особенности включения молодёжи в развитие проекта СПО.

Код для всех

С мая 2022 года в России стартовала инициатива по запуску стажировок для участников сообщества разработчиков свободного и открытого ПО «Код для всех» [1] — в качестве отечественной альтернативы программы «Google Summer of Code». В первый год запуска программы в отборе поучаствовало 540 школьников и студентов, из

которых было отобрано отобрано 22 человека, ставших участниками программ стажировки в командах PostgresPro, Belsoft, CyberOK и Яндекс. Стажёры получили не только ценный опыт, но и стипендию или вознаграждение от партнёров программы [2]. В 2023 года на стажировки «Код для всех» подали заявки уже более 1200 человек.

Одной из заявленных тематик стажировки стало участие в развитии симулятора космических аппаратов «Орбита» [3], который был создан более 10 лет назад и опубликован в начале 2023 года под лицензией GPLv3. В отборе по данной теме участвовало 142 школьника и студента, в итоге пятеро молодых людей, имеющих опыт разработки, достойное портфолио в GitHub и историю достижений в мероприятиях «Кружкового движения», приступили к работе над задачами по развитию проекта в рамках стажировки.

Симулятор «Орбита»

Симулятор «Орбита» был создан для погружения школьников в задачи системной инженерии на примере разработки и запуска космических аппаратов. Симулятор позволяет спроектировать космический аппарат, который должен решить поставленную прикладную задачу на орбите. Космический аппарат конструируется в виде набора общих параметров и выбора подсистем, которые должны соответствовать общим требованиям и решаемой задаче, и программы полёта, составленной из таблицы команд, программы на языке Python или диаграммы с иерархической машиной состояний. Симулятор позволяет рассчитать параметры космического аппарата в течение всего полёта и отображает переданную на Землю телеметрию и результат полёта.

Симулятор был впервые разработан для проведения одноимённых турниров для школьников [4], а затем стал частью Олимпиады Национальной технологической инициативы (ныне — Национальная технологическая олимпиада) [5]. Симулятор используется в кружках по инженерии и космонавтике, в Кванториумах и профильных школах.

Важным отличием «Орбиты» от популярных игровых симуляторов космических аппаратов вроде Kerbal Space Program [6] является открытость и проработанность физической модели, позволяющей использовать программу для погружения не только в инженерию, но и в преподавание школьной и даже университетской физики с использованием сюжета космических полётов. Руководства к симулятору

включают в себя полное описание моделей и последовательно усложняющихся миссий, позволяющих постепенно увеличивать сложность для учащихся. Симулятор включает в себя две последовательно появившиеся версии, рассматривающие следующие ситуации: посадку на планеты солнечной системы и выполнение миссий на околоземной орбите. Каждая версия симулятора содержит набор физических моделей: механики, теплообмена, радиосвязи и др. В общей сложности симулятор содержит более 10 прикладных миссий, для которых можно конструировать космические аппараты и предлагать их программы полёта.

Симулятор «Орбита» полностью написан на `Python` с использованием нескольких библиотек. Используется система сборки на `Makefile` и документация на LaTeX. Для исполнения программ полёта на языке `Python` используется запуск дочерних процессов с передачей сообщений и изоляцией. Важным расширением симулятора стала возможность задания программ полёта с помощью диаграмм иерархических машин состояний [8], которые могут создаваться в популярных редакторах и транслироваться в исполняемый код с использованием библиотеки `PySM` [9]. Симулятор «Орбита» включает в себя свободно распространяемый код универсального транслятора диаграмм машин состояний в код на `Python`.

Стажировки в «Орбите»

В рамках стажировки «Код для всех» были приняты следующие задачи по развитию симулятора:

- графический интерфейс пользователя на базе кроссплатформенной библиотеки Qt, позволяющий конструировать космический аппарат, анализировать результаты запуска симулятора — графики и данные телеметрии;
- расширение физической модели симулятора, позволяющей создавать миссии с несколькими взаимодействующими аппаратами;
- развитие системы управления полётом аппаратов с помощью иерархических машин состояний, перевод всех программ полёта для существующих миссий на язык диаграмм;

- адаптация симулятора (код, документация), изначально разработанного под GNU/Linux к популярным операционным системам — Windows и Mac OS X;
- перевод документации к симулятору на английский язык.

Результаты работы участников, выполняемые в отдельных репозиториях, были протестированы и включены в upstream. Несмотря на то, что не все запланированные работы были выполнены в полном объёме, результат стажировки и качество созданного кода и документаций превзошли ожидания.

Выводы

Включение школьников и студентов в разработку проекта СПО предполагает ряд преимуществ и сложностей. К безусловному преимуществу можно отнести высокий интерес к получению такого живого опыта у ориентированных на программную разработку школьников и студентов. Участие в проекте СПО достаточно требовательно к качеству кода и оставляет реальный след в портфолио, что создаёт хорошую мотивацию для участников.

Сложностями таких стажировок становятся ограничения удалённой коммуникации, неравномерная скорость работы участников с разными умениями и опытом, свободных характер рабочих отношений и гибкий режим работы, слабое знакомство школьников и студентов с базовой культурой работы в проектах с открытым кодом (см. [10]).

Тем не менее, рост популярности конкурсов и стажировок по теме СПО даёт надежду, что данный механизм со временем станет важной составляющей каждого развивающегося проекта.

Литература

[1] Сайт программы «Код для всех» — `https://foss.kruzhok.org/code-for-all`

[2] А.И. Федосеев Всероссийский конкурс open source проектов: опыт проведения и перспективы // Восемнадцатая конференция. Свободное программное обеспечение в высшей школе : материалы конференции / Переславль-Залесский, 27–29 января 2023 г. – Москва : МАКС Пресс, 2023.

[3] Симулятор космических аппаратов «Орбита» — `https://github.com/dralex/orbita-simulator`

[4] В рамках Всероссийского фестиваля науки состоялся турнир по конструированию космических аппаратов — https://asi.ru/news/22815/

[5] Национальная технологическая олимпиада — https://ntcontest.ru

[6] Kerbal Space Program — https://www.kerbalspaceprogram.com

[7] Кружковое движение Национальной технологической инициативы — https://kruzhok.org

[8] Samek, M. Practical UML statecharts in C/C++: event-driven programming for embedded systems. CRC Press, 2008.

[9] Python State Machine — https://pysm.readthedocs.io/

[10] Raymond, Eric S. The art of Unix programming. Addison-Wesley Professional, 2003.

Александра Панюкова, Антон Моисеенко, Сергей Тарасов
Москва, ООО «ШЭРИКС», ГАПОУ КП №11

Проект: ShariX Open https://git.sharix-app.org/ShariX_Open

Об использовании ShariX Open в студенческих проектах

Аннотация

Рассказывается о том, какие студенческие проекты разрабатывались на основе ShariX Open, как разработка этих проектов повлияла на развитие основного продукта

ShariX Open — модули шаблона для создания сервисов оказания произвольных услуг (шэринговых сервисов), совместимых с иными сервисами ShariX.

Шэринговые сервисы — сервисы, позволяющие более эффективно управлять человеческими и материальными ресурсами за счёт возможности использования данных ресурсов различными заказчиками с помощью специальных программных продуктов [1].

ООО «ШЭРИКС» — стартап в области информационных технологий, компания-разработчик платформы ShariX, активно привлекающий к разработке продукта с открытым исходным кодом ShariX Open студентов, формирующий коммьюнити из студентов и выпускников

вокруг проекта. За последний год более 80 студентов взаимодействовало с проектом, при этом более 10 — активно.

На основе ShariX Open можно создавать собственные сервисы, которые можно запускать как совместно с основным продуктом компании при его доступности, так и независимо.

В процессе разработки дочерних решений предусмотрено развитие исходного проекта.

Примеры дочерних решений, которые студенты брали себе в качестве дипломных проектов — сервис поиска гидов, сервис поиска напарников для спортивной тренировки, сервис поиска личных водителей, сервис помощи студентам и другие [2].

В ходе работы над дочерними решениями утверждена структура репозиториев решения ShariX Open, особенности их лицензирования, выстроена структура работы таким образом, чтобы новые изменения уходили как в дочерние проекты, так и обратно. Фактически именно активная разработка новых сервисов сформировала кодовую базу ShariX Open, а отдельным продуктом, пригодным для переиспользования, он стал благодаря спроектированной заранее структуре и принятым подходом к разработке.

Структура репозиториев ShariX Open (`https://git.sharix-app.org/ShariX_Open`):

- sharix-open-webapp-base (AGPL v3) — базовый модуль сборки проекта: `https://git.sharix-app.org/ShariX_Open/sharix-open-webapp-base`
- sharix-open-tickets (AGPL v3) — система для обработки заявок, веб-интерфейс отслеживания заказов: `https://git.sharix-app.org/ShariX_Open/sharix-open-tickets`;
- sharix-open-backend (AGPL v3) — бэкэнд-часть, обеспечивающая совместимость сервисов между собой: `https://git.sharix-app.org/ShariX_Open/sharix-open-backend`;
- sharix-open-webservice-running (GPL v3) — шаблон для построения бэкэнда уникальной части сервиса: `https://git.sharix-app.org/ShariX_Open/sharix-open-webservice-running`;
- sharix-open-webadmin (GPL v3) — расширение модуля администрирования: `https://git.sharix-app.org/ShariX_Open/sharix-open-webadmin`;
- иные репозитории, согласно структуре [3].

В данный момент ведётся активная разработка, готовится релиз первой версии.

Литература

[1] Шэринговые сервисы (sharing-сервисы), 2023, https://wiki.sharix-app.org/doku.php/sheringovie_servisi

[2] ShariX Open Docs, 2023, https://wiki.sharix-app.org/doku.php/start#osnovnye_dejstvujuschie_servisy

[3] Рекомендуемая структура репозиториев сервисов на основе ShariX Open, 2023, https://wiki.sharix-app.org/doku.php/open/tech/dev/repository_structure

Иван Мельников
Саратов, ООО «Базальт СПО»

Проект: «Сизиф» https://www.altlinux.org/Sisyphus

Опыт поддержки догоняющей сборки «Сизифа»

Аннотация

«Сизиф» как репозиторий свободного ПО поддерживает несколько аппаратных архитектур, часть из которых являются основными и обновляются синхронно, а часть догоняющими (не блокируют сборку на основные архитектуры). В докладе рассмотрены технические и философские особенности организации сборки на такие догоняющие архитектуры.

«Сизиф» как репозиторий свободного ПО поддерживает несколько аппаратных архитектур. Каждое изменение репозитория оформляется в виде задания (task), в которое могут входить сборка или удаление одного или нескольких пакетов. Такие задания исполняются сразу для нескольких архитектур, и изменения в репозитории для этих архитектур происходят синхронно. Такие архитектуры мы называем основными.

Помимо основных архитектур, существуют порты «Сизифа» на другие архитектуры, среди которых mipsel (MIPS32 little endian, o32 ABI), riscv64 (RV64GC, lp64d ABI), «Эльбрусы», а также недавно созданный ООО «Базальт СПО» новый порт на архитектуру loongarch64 (new world).

Автор несколько лет поддерживает порты «Сизифа» на mipsel и riscv64. Именно о них пойдёт речь в этом докладе.

Инфраструктура сборки пакетов для mipsel и riscv64 аналогична используемой для основных архитектур: используется отдельный экземпляр girar[1] и отдельный репозиторий (включая собственный компонент noarch). После того, как задание для основного «Сизифа» успешно завершено, специальный робот[2] создаёт аналогичное задание в «догоняющий» girar. После одобрения мейнтенером порта задания собираются.

Существует несколько факторов, которые могут не позволять порту стать основной архитектурой. Не претендуя на полноту, отмечу некоторые из них.

Производительность и доступность оборудования. Пакеты для каждой из этих архитектур собираются «нативно» — сборочный узел должен иметь ту же архитектуру, что и целевая платформа пакета, или совместимую с ней. Однако зачастую достаточно производительное оборудование под целевую архитектуру оказывается крайне редким, плохо поддерживаемым или вообще не существует, так что сборка может проходить в десятки раз медленнее, чем на популярных архитектурах.

Готовность репозитория. В догоняющие репозитории обычно собрано меньше пакетов, чем в основной «Сизиф». Даже пакеты, казалось бы, не требующие портирования, часто требуют адаптации для сборке на ещё одной архитектуре. Также в «Сизифе» распространены циклы из сборочных зависимостей, которые при первой сборке таких пакетов на новую архитектуру приходится вручную разрывать. Поэтому развитие порта — это достаточно долгая работа. При этом репозиторий становится полезным задолго до её завершения — например, можно выпускать полезные образы ОС, имея в репозитории 3–4 тысячи пакетов из собранных в «Сизиф» примерно 18 тысяч (имеются ввиду source-пакеты).

Поддержка сторонних патчей. При сборке на сравнительно новую и/или малопопулярную архитектуру часто приходится вносить в код какие-либо изменения. Эти изменения нужно поддерживать при выходе новых версий. Не всегда возможно и не всегда правильно перекладывать эту работу на мейнтейнера пакета в основном «Сизифе»;

[1] `https://www.altlinux.org/Girar`

[2] `https://www.altlinux.org/Girar/task-rerunner_&_recycler`

также не всегда удаётся передать изменения разработчикам. Мейнтейнер «догоняющего» порта в такой ситуации может поддерживать свой «мини-форк», что в основном «Сизифе» было бы организационно сложнее.

Хочется также отметить несколько особенностей, к которым приводит «догоняющая» сборка пакетов.

Пакеты в догоняющих портах обновляются не сразу — как минимум, в то время, когда собранный пакет уже доступен в основном репозитории, в догоняющий он ещё должен быть собран. Если при сборке возникают проблемы или требуется дополнительное тестирование, отставание может ещё увеличится.

Возникает соблазн и возможность «придержать» некоторые обновления, особенно те, про которые точно известно, что они ломают сборку многих пакетов. Это оправдано на раннем этапе развития порта, когда в приоритете расширение пакетного состава репозитория. Тогда приходится особенно много собирать пакетов, которые в основной «Сизиф» были собраны значительно раньше, в других условиях.

Стоит отметить, что и достаточно зрелые порты продолжают развиваться. Например, в этом году порт riscv64 вырос более чем на 1300 source-пакетов. Это означает, что задача собрать пакет, собранный в «Сизиф» когда-то давно, типична для мейнтейнера порта, и на исправление сборки пакетов в основном «Сизифе» уходит достаточно много сил и времени.

Однако в целом имеет смысл стараться сводить отличия от основного «Сизифа» к минимуму. Во-первых, потому что концептуально порт — это не форк; во-вторых, потому, что решать проблемы, которые затрагивают и основной «Сизиф», помогая тем самым всему сообществу, важнее и интереснее, чем бороться со специфичными проблемами, вызванными отличиями от основного репозитория.

Павел Волнейкин
Санкт-Петербург, ООО «Базальт СПО»

Проект: depcalc
http://git.altlinux.org/people/manowar/packages/depcalc.git

Быстрый расчёт сборочного окружения пакета

Аннотация

Во время выпуска и сопровождения дистрибутивов на базе Sisyphus и его бранчей часто возникает задача пересборки большого количества пакетов. Порядок следования сборочных заданий и их состав играет в этом случае немаловажную роль, поскольку неудачный порядок или состав создаёт паразитную нагрузку на сборочный сервер, снижает его КПД. Для того чтобы снизить её, состав и порядок сборочных заданий должны быть близки к оптимальному. Одним из шагов в этом направлении является оценка влияния результатов завершённого сборочного задания на весь репозиторий, то есть оценка изменений в условиях сборки пакетов, которые вызвало данное задание. Для того чтобы снизить время и ресурсы, необходимые для таких вычислений, потребовалось внести ряд изменений в программу работы со сборочными окружениями на базе Hasher. Обзору этих изменений и новых возможностей посвящён настоящий доклад.

Что такое сборочное окружение исходного SRPM-пакета? Это множество всех пакетов, требуемых для полной его сборки. Множество формируется из двух частей. Первая часть — это базовое сборочное окружение, единое для всех пакетов. Вторая часть — это дополнительное подмножество пакетов, формируемое на основе информации, указанной в spec-файле исходного пакета (тег `BuildRequires`). Получается, что полное сборочное окружение пакета известно заранее? Ведь первая часть — константа, а вторая указывается в самом пакете? Не совсем так. В spec-файле перечисляются имена лишь тех дополнительных пакетов, которые непосредственно нужны для сборки. Но сами эти дополнительные пакеты тоже имеют свои зависимости, поэтому окончательное множество пакетов зависит от текущего состава всего репозитория. К тому же, иногда указание на дополнительные пакеты в spec-файле имеет не вполне однозначный характер. В этом случае спрогнозировать результат ещё сложнее.

До недавнего времени самым простым способом определить сборочное окружение пакета был такой: запустить сборку интересующего

пакета, и после того, как сборочное окружение будет готово, прервать её. Полученное окружение можно было исследовать, в том числе и определить его состав. Те, кто вникал в эту задачу глубже, наверняка нашли, что используемая для сборки пакета команда `hsh-rebuild` имеет специальную опцию `-install-only`, позволяющую автоматически прерывать сборку пакета после установки всех требуемых зависимостей. Однако и в этом случае происходило фактическое получение файлов пакетов и их установка в сборочное окружение. А ведь задача определения состава сборочного окружения состоит лишь в том, чтобы получить сведения о входящих в него пакетах. Кажется, что её можно решить без фактической обработки файлов этих пакетов. И это действительно так. Тем более, что механизм вывода информации о подготовленном к установке наборе пакетов давно реализован в `apt` — программе, управляющей процессом установки пакетов в дистрибутивах ALT.

Таким образом, первый шаг состоял в том, чтобы задействовать этот механизм — команду `apt-get -print-uris` во время работы Hasher по подготовке сборочного окружения. Соответствующая опция `-print-only` была добавлена в скрипт `hsh-install`[1]. Действие её аналогично опции `-install-only`, но фактической обработки файлов пакетов не происходит. Вместо этого выводится список с подробной информацией о пакетах, которые должны были бы быть установлены.

Как выглядит процедура определения состава сборочного окружения пакета при использовании `hsh-install -print-only`? Сперва разворачивается базовое сборочное окружение и выводится информация о входящих в него пакетах. После этого вместо фактической установки указанных в spec-файле дополнительных пакетов выводятся сведения о них и обо всех других пакетах, которые понадобились бы для их установки. Наконец, полученные сведения о составе базового сборочного окружения и обо всех дополнительных пакетах объединяются в один список.

Достоинство этой процедуры в том, что её очень просто оптимизировать. Действительно, если мы знаем, что состав репозитория не менялся, то, следовательно, нет и необходимости повторно разворачивать базовую сборочную среду — ведь в результате работы в режиме

[1]Здесь и далее речь идёт об экспериментальных изменениях, не вошедших пока в основную ветку развития проекта Hasher. Ознакомиться с ними можно по адресу: `https://git.altlinux.org/people/manowar/packages/?p=hasher.git;a=shortlog;h=refs/heads/depcalc`.

`-print-only` она не будет изменена. Это значит, что однажды развёрнутая базовая сборочная среда может быть использована повторно с различными исходными пакетами для вычисления соответствующих им наборов сборочных зависимостей.

Но это ещё не всё. Вторая очевидная оптимизация заключалась в том, чтобы отказаться от копирования исходного пакета в сборочную среду с последующей его распаковкой и вместо этого использовать только spec-файл, который довольно просто извлечь из исходного пакета заранее. В Сизифе нашлось лишь 12 пакетов, для которых данная оптимизация оказалась невозможной ввиду использования директивы `%include`.

С учётом этих двух оптимизаций вышеописанная процедура была реализована в виде скрипта под названием `hsh-print-req`.

Кроме очевидных оптимизаций, данная процедура имеет и одно серьёзное ограничение, касающееся не всех, но существенной части имеющихся в репозитории SRPM-пакетов. Дело в том, что часть исходных пакетов имеет так называемые предварительные сборочные зависимости. Их spec-файлы содержат директиву `BuildRequires(pre)`. В Сизифе около 40% таких пакетов. По правилам, такие spec-файлы должны анализироваться в среде, где установлены все перечисленные в этой директиве пакеты. Иными словами, анализ spec-файла с директивой `BuildRequires(pre)` в базовом сборочном окружении потенциально может привести к ошибке. Это значит, что перед расчётом сборочных зависимостей в режиме `-print-only` в базовую сборочную среду должен быть доустановлен ряд пакетов. Сделать это можно. Но фактическая доустановка пакетов означает, что базовая сборочная среда изменяется. Получается, что 40% пакетов исключают возможность повторного использования сборочной среды? Это не совсем так.

Во-первых, исходным условием для вычисления состава сборочного окружения по-прежнему является базовая сборочная среда. Отличие состоит в том, что в одном случае в ходе вычисления эта среда не изменяется, а в другом — изменяется. Отсюда возникает задача тиражирования заранее подготовленной сборочной среды. Смысл её в том, чтобы растиражировать однажды построенную базовую сборочную среду и затем сэкономить время и ресурсы, используя готовые экземпляры при вычислениях.

Во-вторых, изменённая базовая сборочная среда тоже может быть использована повторно — для тех исходных пакетов, которые имеют

соответствующий ей набор предварительных зависимостей (то есть зависимостей, указанных в `BuildRequires(pre)`). Простые исследования показали, что примерно на 7400 исходных пакетов приходится всего около 700 уникальных списков предварительных зависимостей. Это значит, что в среднем, каждое сборочное окружение, полученное доустановкой пакетов в базовое, может быть использовано повторно около 10 раз. Последующие исследования показали, что для некоторых популярных наборов предварительных зависимостей, это число превышает 200 раз.

Оба вышеуказанных замечания, если рассматривать их вместе, приводят к идее тиражирования не только базового, а любого заранее сформированного окружения. Как будет выглядеть процедура определения состава сборочного окружения пакета, если учесть всё сказанное? Примерно так:

1. берём экземпляр базового сборочного окружения и spec-файл;
2. определяем, какие предварительные зависимости имеет пакет;
3. определив их, ищем экземпляр сборочного окружения, который им соответствует;
4. если такого нет, то создаём его на базе экземпляра базового окружения;
5. на базе подходящего сборочного окружения определяем конечный набор сборочных зависимостей (установленные в окружение пакеты + пакеты, которые должны были бы быть установлены).

Другой, потенциально более оптимальный вариант, состоит в том, чтобы выбрав и подготовив окружение, провести на нём вычисления для всех пакетов, которые требуют именно данного окружения. Практика покажет, какой вариант будет лучше. Но как в одном, так и в другом случае, присутствует задача простой идентификации состава сборочного окружения. С одной стороны, такой идентификатор нужен для хранения и поиска заранее подготовленных окружений. С другой стороны, тот же самый идентификатор должен использоваться и по отношению к пакетам, раз мы хотим установить взаимно однозначные соответствия между окружениями и пакетами, которые требуют данных окружений.

Идентификации по составу сборочного окружения была реализована на основе двух скриптов: модифицированной версии уже упоми-

навшегося `hsh-print-req` и нового скрипта, получившего название `hsh-print-pkgs`.

Модификация скрипта `hsh-print-req` состояла в добавлении двух опций: `-pre` и `-hash`. Работа скрипта при указании их обоих состоит в том, что сперва определяется состав предварительного сборочного окружения, а затем вычисляется SHA1-хеш от этих данных. Состав предварительного сборочного окружения определяется точно так же, как и конечный набор сборочных зависимостей пакета — в режиме `-print-only`, но вместо списка пакетов из тега `BuildRequires` в этом случае используется список из тега `BuildRequires(pre)`. Важно отметить, что данная операция тоже производится в базовом сборочном окружении, и это окружение не меняется в ходе неё. Это значит, что потенциально идентификатор предварительного сборочного окружения может быть получен для всех пакетов на базе одного и того же экземпляра базового окружения.

Новый скрипт `hsh-print-pkgs` попросту выводит состав заданного сборочного окружения, а с добавлением опции `-hash` — вычисляет SHA1-хеш от этих данных.

Для целей тиражирования сборочных окружений была усовершенствована процедура их кеширования, уже реализованная в Hasher. Усовершенствование заключается в более удобном использовании данной функции в том случае, когда архив рабочей директории Hasher сохраняется в произвольном месте, а также при восстановлении рабочей директории из такого архива. Дополнительные скрипты получили названия `hsh-create-workdir`, `hsh-save-workdir` и `hsh-rm-workdir`. Теперь сохранение состояния директории можно выполнить в любой момент, а не только совместно с подготовкой базового сборочного окружения.

Ещё одно нововведение заключается в возможности использовать один и тот же экземпляр рабочего окружения `apt` с несколькими сборочными окружениями, то есть с несколькими экземплярами рабочих директорий Hasher. Что это даёт? Всё это время мы исходили из того, что состояние репозитория не изменяется, пока идут вычисления. Иными словами, если мы поставили задачу определить состав сборочных окружений всех интересующих нас исходных пакетов, то это имеет смысл делать на одном и том же фиксированном состоянии репозитория. Это так. Но как можно упростить данную процедуру на следующей её итерации? Можно ли как-то выгодно повторить её на новом состоянии репозитория?

Здесь возможны два крайних случая: в первом случае изменения состава репозитория таковы, что они не затрагивают сборочную среду ни одного пакета. В такой ситуации нет необходимости отбрасывать заранее подготовленные сборочные окружения — их можно продолжать использовать повторно. В противоположном случае, наоборот, изменения затрагивают базовую сборочную среду и, следовательно, все заранее подготовленные и сохранённые сборочные окружения необходимо считать негодными и отбросить.

Может возникнуть вопрос: нужно ли вообще проводить новую итерацию по определению состава сборочных окружений пакетов в том случае, если известно, что изменения в пакетной базе не затрагивают ни базовое сборочное окружение, ни одно из заранее подготовленных предварительных окружений? Ответ: да, это имеет смысл. Существует вероятность, что несмотря на неизменный состав базовых окружений, вычисление окончательного состава сборочных зависимостей, которую выполняет `apt`, даст в новых условиях другой результат.

Как проверить, что изменения в пакетной базе не затрагивают того или иного сборочного окружения? Можно поступить так же, как и в обычной системе: сперва обновить информацию о доступных пакетах командой `apt-get update`, а затем определить состав очередного обновления командой `apt-get dist-upgrade`. Отсутствие обновлений покажет, что изменения в пакетной базе не затрагивают данное сборочное окружение. На чём здесь можно сэкономить? Очевидно, на процедуре обновления информации о доступных пакетах. Именно для этой цели служит модификация Hasher, получившая название shared aptbox. Благодаря ей `apt` в каждой из нескольких рабочих директорий Hasher будет работать со свежими сведениями о пакетах после однократного выполнения команды `apt-get update`. Это касается и проверки `dist-upgrade`. Технически это реализовано так: файлы с информацией о доступных пакетах хранятся в общей директории, а в конфигурационных файлах `apt.conf` в каждой из рабочих директорий Hasher даются ссылки на эту директорию.

На базе всех вышеописанных модификаций Hasher была написана утилита под названием `depcalc`, комплексно реализующая процедуру определения состава сборочного окружения пакета. При этом утилита автоматически создаёт и использует кеш сборочных окружений и имеет отдельные опции для обслуживания этого кеша. Другой набор опций служит для обновления информации о доступных пакетах с использованием возможностей shared aptbox.

Количественные характеристики работы этой утилиты я надеюсь представить в своём докладе на конференции.

Виталий Липатов
Санкт-Петербург, Этерсофт
Проект: EPM https://eepm.ru

Реализация перепаковки сторонних пакетов в `epm play`

Аннотация

Поддержка в `epm play` установки пакетов из различных сторонних источников и форматов потребовала реализации трёхступенчатого преобразования исходного пакета. Стремление сделать правила перепаковки едиными для различных систем приводит к использованию дистроуниверсальных зависимостей. Переносимые пакеты собираются отдельным rpmbuild, не привязанным к системе.

Концепция `epm play` добавляет недостающее звено (или улучшает его) между производителями дистрибутивов на основе GNU/Linux и производителями программного обеспечения (ISV), решая задачу совместимости для бинарно поставляемого ПО.

Рассмотрим применение `epm play` на платформе ALT.

Как правило, поставщик ПО предоставляет своё ПО в одном из следующих вариантов:

- tar-архив с исполняемыми файлами;
- AppImage, snap;
- deb-пакет для Debian/Ubuntu;
- rpm-пакет для родственных Fedora систем;
- rpm-пакет для конкретного дистрибутива (например, для основанного на бранче ALT p10).

Результатом работы `epm play` должен быть rpm-пакет, устанавливаемый в систему (и удаляемый) штатными средствами.

Для адаптации стороннего ПО в epm реализовано несколько механизмов, которые могут быть использованы по отдельности:

- `epm play` [--download-only] — загрузить и установить приложение (указанной версии);

- `epm pack [--install]` — преобразовать архив от поставщика в файловую структуру, пригодную для сборки пакета (или же просто извлечь пакет из архива);
- `epm repack [--install]` — перепаковать пакет для целевой платформы.

Помимо бинарной совместимости исполняемого кода и формата пакетов приходится контролировать ещё:

- зависимости пакетов (как правило, формируются заново);
- скрипты, выполняемые после установки / перед удалением пакета (выключаются);
- репозитории, откуда производится установка (не добавляются в sources.list).

В общем случае просто выполняется перепаковка в `rpm`. Поскольку штатный `rpmbuild` имеет свою специфику (тянет среду сборки C-программ (ALT bug 34308), имеет сложный автопоиск зависимостей, уникален для платформы ALT), было принято решение собрать апстримный rpmbuild в пакет `eepm-rpm-build`. В перспективе это позволит обеспечить идентичность сборки независимых от дистрибутива пакетов на различных системах.

В последней версии `epm` автоматический поиск зависимостей по умолчанию выключен, и применяется простой скрипт, вычитающий из списка требуемых soname список предоставляемых пакетов и результат добавляющий в качестве Requires.

Получилось перейти к использованию дистрибутиво-независимых зависимостей для библиотек и команд (вида `libX11.so.6()(64bit)` и `/usr/bin/less`) при ручном указании зависимостей пакета, что позволяет использовать одни и те же правила перепаковки для всех rpm-based систем.

Также потребовалось добавить проверку доступной в дистрибутиве версии `glibc` и `libstdc++`, поскольку некоторые программы перешли на версию `glibc` выше, чем в p10. Для них устанавливается последняя совместимая версия.

Есть перспективы облегчить установку поддержки принтеров и сканеров. Как правило, вручную требуется достаточно много манипуляций, описываемых на вики. Для проверки концепции добавлены

```
cnrdrvcups-ufr2-uk
epsonscan2
hplip-plugin
kyodialog
sane-panakvs
pantum
```

и получены положительные отзывы.

Для каждой версии epm для каждой платформы выполняется автоматическая проверка устанавливаемости приложений через epm play, после чего проверенная версия записывается в список рекомендуемых к обновлению.

Это позволяет пользователю, выполняя `epm play --update all`, обновиться до последних проверенных версий. Что не мешает, выполнив `epm play <приложение>`, установить самую последнюю версию.

На данный момент `epm play` только упрощает скачивание ПО с официальных сайтов. Планируется добавить проверку контрольных сумм, чтобы ввести независимый контроль (предварительное тестирование) устанавливаемых бинарников.

За прошедший год в `epm play` добавлена поддержка межпланетной файловой системы IPFS (`epm play --ipfs`), которая обеспечивает адресацию файла по хэшу его содержимого. Уже сейчас это позволяет обходить запреты на доступ к иностранным сайтам, а при дальнейшем развитии позволит реализовать защиту от подмены содержимого.

EPM и `epm play` это свободные проекты, необходимые сценарии можно как присылать в виде pull request на `https://github.com/Etersoft/eepm` или иным способом, так и паковать в отдельные пакеты.

Литература

[1] OSDAY-2021 — Дистрибутив ROSA и стороннее ПО — Михаил Новосёлов `https://www.youtube.com/watch?v=6tCL2MPnS-o`

[2] OSDAY-2023 — Дистрибутивоуниверсальные пакеты — Михаил Новосёлов `https://vk.com/video-172037486_456239033`

[3] Выступление на OSDAY-2023. Как сделать дистрибутивоуниверсальный пакет — Михаил Новосёлов `https://nixtux.ru/1268`

Данила Загайнов, Георгий Курячий, Дмитрий Волканов
Москва, ВМК МГУ, «Базальт СПО», ВМК МГУ

Проект: py3dephell git://git.altlinux.org/gears/p/py3dephell.git

Построение и исследования графа python3-зависимостей в репозитории Sisyphus

Аннотация

В работе рассматриваются зависимости и provides-ы python3-пакетов, а также выделение подпакета из исходного python3-пакета.

py3dephell

Данный пакет предоставляет модули для работы с зависимостями и `provides`-ами `python3`-пакетов.

Его компонент `py3prov` на основе переданных имён файлов и каталогов определяет `provides`-ы рассматриваемого пакета с учётом `.pth` файлов и корректности получаемых имен [1].

Для выявления же зависимостей используется `py3req`. Данный модуль читает `AST`-дерево переданного скрипта, написанного на `python3`, или же парсит заголовок `ELF`-файла, для выявления соответствующих зависимостей [2]. Для определения `self-provides` `py3req` использует `py3prov`.

Компоненты `rpm-build-python3` уже используют данный пакет для формирования зависимостей и `provides`-ов `rpm`-пакетов в репозитории `Sisyphus`. В ходе внедрения `py3dephell` было выявлено множество (свыше 400) ложных `provides`-ов у `rpm`-пакетов, а также ошибки в упаковке ряда пакетов, вследствие чего, например, порождались зависимости между основным пакетом и подпакетом с тестами.

Также данные модули могут автоматически формировать requirements.txt, пригодного для использования утилитой `pip` [3].

В то же время была рассмотрена задача вынесения нежелательных зависимостей из пакета. Предполагается, что пользователь, задав набор ключевых модулей, сможет выделить из исходного пакета подпакеты, содержащие избыточные зависимости. Примером таких зависимостей могут служить зависимости на тесты, графический интерфейс, сеть и т. д.

Был предложен следующий алгоритм:

1. Для заданного набора основных модулей формируется подпакет, содержащий их и их зависимости. Если в полученный подпакет попала одна из избыточных зависимостей, то распиливание для заданных наборов невозможно.
2. Для вынесенных зависимостей строятся подпакеты, содержащие каждую из избыточных зависимостей, и модули, зависящие от них.
3. Для образовавшихся подпакетов берутся их зависимости, которые ещё не попали ни в 1 из подпакетов. Если несколько подпакетов хотят один и тот же модуль, то его выносят в отдельный подпакет.

Литература

[1] Site-specific configuration hook [Электронный ресурс]: `https://docs.python.org/3/library/site.html`

[2] Abstract Syntax Trees [Электронный ресурс]: `https://docs.python.org/3/library/ast.html`

[3] The PyPA recommended tool for installing Python packages [Электронный ресурс]: `https://pypi.org/project/pip/`

Стас Фомин
Москва, ИСП РАН
`http://github.com/belonesox/terrarium-assembler`,
`http://github.com/belonesox/terrarium-assembler-win`,
`http://github.com/belonesox/terrarium-adapter`

Terrarium Assembler — кроссплатформенные Python/Go приложения с аудитом сборки

Python-стек общепринят для прототипирования и разработки высокотехнологических приложений (оптимизация, машинное обучение, симуляции, распознавание текста, . . . , любая научная магия). Однако нетривиально превратить прототип в кроссплатформенное приложение, работающее на всех Windows- и Linux-системах, включая сертифицированные российские Линуксы, не используя запрещённые регулятором приёмы типа классических Linux-контейнеров.

Ещё сложнее пройти аудит и сертификацию, избавившись от интерпретаторов в поставке и предъявив воспроизводимый способ сборки каждого бинарного файла, имеющегося в дистрибутиве, из исходных кодов. Даже если брать только Linux, то пересборка «всего лишь Python-библиотек» с произвольным выбором версий сложнее «полной пересборки Linux», где эти версии зафиксированы, ведь при этом всё равно приходится сутками пересобирать базовый уровень библиотек, от gcc до библиотек линейной алгебры.

Важно обеспечить, чтобы технические проблемы сертификационной сборки не тормозили разработку или экспериментирование с выбором библиотек и их версий.

Для решения этих проблем мы разработали серию open-source проектов «Terrarium Assembler» (TA).

Python уже давно самый удобный язык для прототипирования и разработки высокотехнологических приложений, ибо сейчас в нём есть «батарейки» практически для всех областей научной разработки (криптография, символическая алгебра, алгоритмы оптимизации, машинное зрение, AI/ML), и при этом самый короткий путь от набросков алгоритмов в Jupyter-ноутбуках до работающего для пользователей web-сервиса.

Однако обычным путём такие приложения нельзя сделать кроссплатформенными и сертифицированными российскими регуляторами — требуется и отсутствие интерпретируемого кода, и контролируемая сборка из исходников всех бинарных файлов (см., например. ДСП-шные руководства ФСТЭК).

Нельзя (или, скажем, неразумно до безумия) использовать:

- Python-деплой в виде как-то опакеченных виртуальных окружений — нет, не кроссплатформенно (виртуальные окружения привязаны к версиям python, не говоря уж про версии GCC), нельзя тащить что-то на интерпретируемых языках (регуляторы), в любом случае надо пересобирать каждый бинарник.
- Системные пакеты самих ОС — сертифицированные Линуксы — это только базовые пакеты (несколько тысяч пакетов по сравнению с сотней тысяч пакетов «нормального» Linux, и там нет ничего высоконаучно-питонового. Плюс необходимость собирать и тестировать отдельные версии под каждую версию целевой системы, с разной пакетной базой, политиками, форматами пакетов и даже версиями GCC. «Десятки сертифицированных Ли-

нуксов» с анемичной пакетной базой, и оригинальными доработками — это не достижение российской индустрии, а вечная боль всех поставщиков прикладных решений, минное поле для тестирования ошибок в поддержке стандартов и протоколов. Увы, шанс в обозримом будущем дождаться «единый сертифицированный российский Линукс» с пакетной базой уровня Debian/Fedora — пренебрежимо мал.

- Линукс-контейнеры docker/podman — могли бы быть (и может ещё будут) решением, но пока они, с одной стороны, отсутствуют в большей части сертифицированных дистрибутивов, а с другой стороны, являются настолько запрещёнными, что даже наличие где-то среди исходников файла формата сборки Docker-контейнера действует как красный флаг для сертификационной лаборатории. Аналогично с подходами типа snap/flatpack/appimage.

И это не говоря уже про Windows, где совсем отдельный мир для сборки и публикации приложений, хотя там гораздо лучше с бинарной совместимостью.

Наш подход, обеспечиваемый серией проектов «Terrarium Assembler» (TA), следующий:

- Разработчики совершенно свободно используют удобные им дистрибутивы для разработки будь то Windows или Linux, разрабатывая кроссплатформенный Python-код, используя все нужные им библиотеки.

- Достаточно быстро с помощью TA можно собрать «отладочную версию», проверив, что различные проекты имеют совместимые зависимости по библиотекам, что проходят тесты и т. п.

- Если тесты в порядке, то более длинный путь собирает скомпилированный дистрибутив, где Python-код компилируется в Си-шный код с помощью Nuitka, затем компилируется в бинарный код, и всё это собирается в «переносимую папку со всеми зависимостями и ld.so загрузчиком», работающую под любыми Линуксами. Рекомендуемый вендорами российских Линуксов путь «ладно, берите нужные библиотеки, но glibc-то у нас годная» `https://0x1.tv/20230623H` — увы, не работает, когда нужна поддержка старых дистрибутивов, поэтому надо брать всё, включая «ld.so».

- Аналогично, и чуть легче с некой заменой исходников, можно собрать проект и для Windows, тоже «переносимая папка».
- Опакечивается в любой требуемый формат (DEB/RPM/ISO/MSI/EXE с инсталлятором), и запускаются финальные тесты с инсталляцией на стенды.
- И самый длинный путь, если вышеперечисленной сборки прошли тесты — это сборка для аудита, когда дополнительно происходит
  - Сборка «Системной платформы» (может длиться сутки) — перекомпиляция RPM-спеков всех системных библиотек (от libc до питоновых), так или иначе попадающих в дистрибутив.
    * При этом можно собрать не только библиотеки, но и все нужные сервисы и утилиты (включая базы данных, вебсерверы и т.п.).
    * Также при этом можно собрать отсутствующие системные RPM-пакеты на основе своих SRPM.
    * Для Windows используются спеки сборки системы CONAN.
  - Сборка «Python-библиотек» нужных версий из исходников. Это приходится делать отдельно, т. к. нужных версий скорее всего нет в «Системной платформе».
- Кроме Python-проектов можно собирать также и Go и C++ проекты и модули, но со сборкой Go-проектов и так нет проблем («go download/go build»), и в целом проект ориентирован именно на решение выкатки Python-решений.

В результате, удаётся добиться и быстрой разработки, и честного аудита воспроизводимой сборки, выполняемого из фиксированных исходников в «чистой комнате» без доступа интернету и к каким-либо хранилищам артефактов, так, что про каждый бинарный файл известно, из какого исходного кода он получен.

Система была разработана несколько лет назад, презентована `https://0x1.tv/20210617C` на конференции OSSDEVCONF-2021 («Terrarium Assembler — эффективные сборка и распространение высокотехнологичных Python-приложений под различные операционные системы»), но с тех пор постоянно развивалась, в частности:

- Контейнеризирована вся сборка, что позволяет на одной сборочнице собирать разные проекты без риска «переопыления», и теперь Linux-версией ТА можно пользоваться на любом современном Линуксе.
- Поддерживается сборка пакетов всех типов — RPM/DEB/ISO/MSI.
- Радикально улучшен режим аудит-сборки, по опыту взаимодействия с разными лабораториями сертификации («под Минобороны», «под ФСТЭК», ...).
- Автоматизирована система минимизации дистрибутива используя strace-анализ выполняемых тестов.
- Моделирование «монорепозитория» в виде отдельных операций над всеми репозиториями отдельных библиотек проекта.
- Системно решено множество частных проблемных ситуаций.

В отличие от прошлого доклада, поверхностно посвящённого решаемым проблемам и преимуществам подхода, в этом мы постараемся кратко и концептуально, не дублируя документацию (`https://github.com/belonesox/terrarium_assembler/blob/master/README_RU.md`), рассказать «как это работает» на архитектурно-техническом уровне, с подсветкой наверное наиболее важной для российской аудитории части — как использование проекта помогает проходить сертификации.

Александр Певзнер
Москва, независимый разработчик
`https://github.com/alexpevzner/sane-airscan`,
`https://github.com/OpenPrinting/ipp-usb`

Стек «бездрайверного» сканирования и печати ОС Linux

Аннотация

Описываются пакеты sane-airscan и ipp-usb. Sane-airscan — это драйвер SANE, реализующий поддержку «бездрайверного» сканирования. Драйвер поддерживает два наиболее популярных протокола: eSCL (часть Apple Airprint) и Microsoft WDS. Ipp-usb — это системный демон,

обеспечивающий поддержку «бездрайверного» сканирования и печати для USB устройств без сетевого подключения. Даётся обзор технологии «бездрайверного» сканирования и печати, которая существенно упрощает администрирование и настройку принтеров и сканеров, делая её доступной даже неквалифицированным пользователям.

Что такое бездрайверное сканирование и печать?

Традиционно любое внешнее устройство нуждается в своём специфическом драйвере. Поиск, установка и настройка драйверов — не всегда простая задача. Особенно если речь идёт не о Windows.

В то же время, мы привыкли, что любая флешка в любом компьютере под управлением любой ОС «просто работает» и в настройке не нуждается.

Это стало возможным потому, что все флешки поддерживают единый, стандартный протокол. На этом примере мы видим смену парадигмы: теперь не операционная система подстраивается (с помощью драйвера) под аппаратуру, а аппаратура и ОС подстраиваются под стандартный, не зависящий от производителей, протокол.

Аналогичная технология в отношении принтеров и сканеров называется «бездрайверной» печатью и сканированием — в том смысле, что общий, единый для всех устройств драйвер становится частью ОС, а не поставляется вместе с аппаратурой.

Стандарты и протоколы

Существует два набора стандартов для «бездрайверного» сканирования и печати.

Первый набор активно продвигается (и частично разработан) компанией Apple. Он основан на таких открытых стандартах IETF, как DNS-SD [1] для поиска устройств в сети и IPP [2][3] для печати.

Спецификация eSCL (протокол для сканирования) сначала была доступна только на условиях NDA, но позже была опубликована Mopria Alliance [4]. Тем самым, в настоящий момент весь стек протоколов является открытым.

Маркетинговые названия, под которыми Apple продвигает этот набор протоколов (Bonjour Printing и AirPrint. AirPrint) для сканеров неофициально называется AirScan.

Другой набор протоколов, близкий семантически, но совершенно не совместимый на уровне форматов запросов и ответов, разработан и придвигается компанией Microsoft и называется Web Services for Devices (WSD) и состоит из WS-Discovery, протокола поиска устройств, и WS-Scan/WS-Print, протоколов сканирования/печати. Спецификация опубликована в [5]

Для поддержки USB-устройств без сетевого подключения существует протокол IPP over USB [6], который вернее было бы назвать HTTP over USB. Его идея заключается в замене TCP-транспорта USB-транспортом в HTTP, и тем самым — обеспечения поддержки протоколов IPP и eSCL, основанных на HTTP. Даже встроенная в принтер конфигурационная веб-консоль работает через IPP over USB.

При этом протоколы WSD через IPP over USB не поддерживаются.

Основные компоненты стека

Бездрайверное сканирование и печать работает с устройствами, которые подключаются по сети (Ethernet или WiFi) или через USB.

Настройка устройств максимально автоматизирована и в большинстве случаев сводится к выбору нужного устройства из предложенного списка.

Стек бездрайверного сканирования и печати состоит из нескольких компонент, обеспечивающих следующие функции:

1. Автоматическое обнаружение устройств в сети.
2. Поддержку протокола печати.
3. Поддержку протокола сканирования.
4. Поддержку IPP over USB.

Автором был разработан драйвер сканера sane-airscan, реализующий оба семейства протоколов сканирования в одном драйвере и программа (системный демон) ipp-usb, реализующая поддержку IPP over USB.

sane-airscan

sane-airscan [7] — это драйвер сканера, совместимый с проектом SANE (но не входящий в него, независимый). Он написан на языке Си.

sane-airscan реализует оба протокола сканирования, eSCL и WSD.

Поиск eSCL-устройств полагается на DNS-SD демона Avahi, но поиск WSD-устройств пришлось реализовать в драйвере самостоятельно из-за отсутствия в Linux стандартной реализации WS-Discovery.

При разработке sane-airscan ставилась задача обеспечить максимальный комфорт для пользователя. Это потребовало решения ряда интересных технических задач.

1. Сведение вместе результатов поиска одного и того же устройства. Одно и то же устройство может быть найдено сразу в нескольких экземплярах: eSCL/WSD, IPv4/IPv6, HTTP/HTTPS и т.п. sane-airscan сводит все такие «находки» вместе, предлагая пользователю выбирать из списка фактически присутствующих физических устройств. Это оказалось достаточно неочевидной задачей, и её решение занимает примерно 1/4 кодовой базы драйвера.

2. Оптимизация скорости поиска WSD. Поиск DNS-SD через Avahi работает быстро, т.к. фактически Avahi сканирует сеть всё время и по запросу возвращает свой кэш. А вот работающему без демона-помощника WS-Discovery, для надёжного обнаружения устройств требуется время на сканирование сети.

Для ускорения сканирования используется тот факт, что большинство WSD-сканеров являются МФУ, совмещёнными с IPP-принтерами: сканирование можно прекратить, как только прояснится статус всех устройств, которые видны через Avahi как IPP-принтеры.

3. Корректировка параметров изображения. Интерфейс SANE предполагает, что процесс сканирования осуществляется в фоновом режиме, поскольку он может быть очень долгим.

Однако SANE требует, чтобы сразу после запуска сканирования драйвер мог вернуть точные фактические параметры изображения (которые у ряда устройств отличаются от запрошенных).

Для решения этой проблемы sane-airscan спекулятивно возвращает параметры изображения, совпадающие с запрошенными, а при распаковке изображения приводит его в соответствие с заявленными параметрами.

ipp-usb

ipp-usb [8] — это демон, реализующий поддержку IPP over USB. Он написан на языке Go.

Ранняя реализация для Linux, ippusbxd [9], была очень простая. Эта программа принимала TCP-соединения, принятые из TCP-сокета, данные отправляла в USB, пришедший из USB ответ отправляла назад в TCP.

Работало это не очень хорошо. При некорректном поведении клиента синхронизация между хостом и устройством терялась, и это нарушало обработку последующих запросов. Причина в том, что в отличии от TCP, в USB нет механизма, позволяющего восстановить синхронизацию, просто закрыв соединение.

ipp-usb понимает протокол HTTP и является полноценным HTTP-proxy, гарантирующим, что при любом поведении клиентов, с точки зрения устройства HTTP-протокол будет полностью соблюдён.

Текущее состояние и перспективы развития

Оба проекта несколько лет находятся в активной эксплуатации и входят во все основные дистрибутивы Linux. Усилиями пользователей sane-airscan перенесён на все основные ветки BSD. Ipp-usb недавно был перенесён на FreeBSD. Sane-airscan входит в состав Google ChromeOS. Реализация IPP over USB из ChromeOS является разработкой Google, но основана на наработках и идеях, позаимствованных из ipp-usb [10].

Список устройств, протестированных (усилиями пользователей) на совместимость с sane-airscan, состоит более чем из 170-и записей и постоянно продолжает расти.

Можно ожидать, что в перспективе нескольких лет практически все новые принтеры и сканеры будут «бездрайверными».

Опыт продвижения проектов в дистрибутивы Linux

Ключевыми моментами успешного продвижения проектов в дистрибутивы Linux явились, по мнению автора, следующие:

1. Востребованность самих проектов.
2. Участие автора в дискуссиях на площадках зарубежных форумов коллективной поддержки популярных дистрибутивов Linux.
3. Качественная реализация.

4. Распространение программ не только в исходных текстах, но и в виде пригодных для инсталляции пакетов для многих дистрибутивов.
5. Быстрая реакция на сообщения пользователей о проблемах и ошибках.

Для сборки пакетов использовалась площадка OpenSUSE Build Service [11], поддерживающая, кроме OpenSUSE, множество других дистрибутивов. Использование этой площадки бесплатно для проектов с открытыми исходными текстами.

Литература

[1] S. Cheshire, M. Krochmal, DNS-Based Service Discovery, RFC-6763, 2013

[2] M. Sweet, Internet Printing Protocol/1.1: Encoding and Transport, RFC-8010, 2017

[3] M. Sweet, Internet Printing Protocol/1.1: Model and Semantics, RFC-8011, 2017

[4] Mopria eSCL Specification, `https://mopria.org/spec-download`

[5] Web Services on Devices Reference, `https://learn.microsoft.com/en-us/windows-hardware/drivers/image/web-services-on-devices-reference`

[6] USB Print Interface Class IPP Protocol Specification, `https://www.usb.org/document-library/ipp-protocol-10`

[7] Проект sane-airscan, `https://github.com/alexpevzner/sane-airscan`

[8] Проект ipp-usb, `https://github.com/OpenPrinting/ipp-usb`

[9] Проект ippusbxd, `https://github.com/OpenPrinting/ippusbxd`

[10] Ippusb Bridge, `https://chromium.googlesource.com/chromiumos/platform2/+/HEAD/ippusb_bridge`

[11] openSUSE Build Service, `https://build.opensuse.org/`

Антон Абрамов
Саратов, ООО «Базальт СПО»
https://github.com/altlinux/gpupdate

Расширение возможностей администрирования ОС«Альт» через групповые политики

Аннотация

Групповые политики в Linux-дистрибутивах ОС «Альт» управляют многочисленными параметрами системы — более 1100 наименований. Утилита применения групповых политик в дистрибутиве ОС «Альт» — gpupdate — позволяет гибко использовать внутренние механизмы обработки загружаемых параметров. Устройство административных шаблонов групповых политик — admx-файлов — позволяет удобно и быстро задавать параметры и сохранять их в каталог Sysvol контроллера домена. В совокупности связка admx-шаблонов и gpupdate от «Базальт СПО» дают системным администраторам гибкий механизм управления доменом. В докладе рассматриваются способы увеличения охвата параметров в системе, управляемых через групповые политики в Linux-дистрибутивах »Альт». Описываются механизмы для расширения групповых политик через admx-шаблоны и утилиту gpupdate. Приводятся примеры практической реализации.

Управление доменом со службой каталогов на базе Samba DC либо MS Active Directory поддерживается в ОС «Альт» инструментом групповых политик. Исполнение групповых политик в дистрибутивах «Альт» включает: графические инструменты управления службой каталогов, редактирования параметров политик, утилиту применения параметров на клиентских компьютерах. GUI-редактор параметров политик GPUI поддерживает стандарт admx-шаблонов. XML-структура административных шаблонов групповых политик (admx-файлов) подробно документирована и позволяет в простой форме задавать системные настройки для компьютера, пользователя, с поддержкой любого языка. Редактор GPUI позволяет моментально загрузить корректно подготовленный admx-шаблон и применять в работе — наполнять параметрами файлы каталогов групповых политик (GPT — Group Policy Template). Таким образом, системные администраторы сторонних организаций могли бы самостоятельно готовить шаблоны политик под внутренние запросы. Но необходима поддержка второго элемента конструкции — утилиты, применяющей параметры в системе клиента.

Механизмы применения gpupdate

Применение параметров групповых политик в системе для компьютера или пользователя в ОС «Альт» осуществляет утилита gpupdate. Утилита задействует ряд механизмов применения, которые позволяют также самостоятельно расширить функционал вручную. Механизмы применения параметров, которые позволяют создавать собственные политики: polkit, gsettings (dconf), control, systemd, скрипты (logon/logoff/startup/shutdown). Механизмы с коротким описанием:

- Polkit. Возможность добавить новые действия для правил polkit;
- Gsettings. Управление настройками системной базы реестра dconf;
- Systemd. Управление запуском служб Systemd;
- Control. Управление скриптами control в настройке системы;
- Скрипты загрузки системы и авторизации пользователя.

Подробный метод действия

Библиотека Polkit выполняет в операционной системе роль ограничителя действий приложений, преимущественно работающих на шине D-Bus. Polkit позволяет установить степень ограничений для непривилегированного процесса при обращении к привилегированному. Например, ограничить доступ пользователя к настройкам сети или монтированию блочного устройства (USB-накопителя) только для тех пользователей, которые знают пароль суперпользователя. Действия приложений, для которых заложены Polkit-ограничения, называются «actions». На основе указанных действий «actions» возможно формирование правил «polkit-rules», согласно которым в операционной системе и определится степень ограничений для программ и пользователей. Механизм gpupdate умеет автоматически формировать polkit-правила. Благодаря этому возможно добавление новых групповых политик для polkit-правил (см. Рис.1).

Свободная среда рабочего стола для UNIX-подобных операционных систем под названием Gnome породила большую серию ответвлений прочих популярных окружений рабочего стола (desktop environment — DE). Системные настройки в GNOME хранятся в формате dconf-файлов, для управления которыми применяется утилита

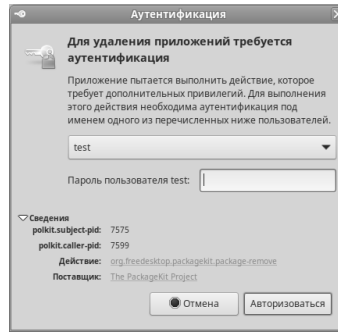

Рис. 1: Получение прав.

GSettings. Ключи в базе данных dconf напоминают реестр операционной системы Windows. В ОС «Альт Рабочая станция» применяется окружение Mate, ставшее ответвлением от GNOME 2. Конфигурация DE Mate хранится в базе dconf, ключами которой управляет консольное приложение GSettings. Механизм gpupdate управляет ключами Gsettings и через них редактирует ключи dconf. На текущий день через групповые политики можно настраивать целый ряд параметров графической среды Mate — от фона рабочего стола до хранителя экрана. Возможно добавить новые политики для незадействованных ключей Gsettings:

```
org.mate.peripherals-touchpad vertical-edge-scrolling true
org.mate.peripherals-touchpad tap-button-three-finger 2
org.mate.peripherals-touchpad natural-scroll false
org.mate.peripherals-touchpad left-handed 'mouse'
org.mate.peripherals-touchpad tap-button-one-finger 1
org.mate.peripherals-touchpad horizontal-two-finger-scrolling false
org.mate.peripherals-touchpad two-finger-click 1
org.mate.peripherals-touchpad motion-acceleration -1.0
org.mate.peripherals-touchpad tap-button-two-finger 3
```

Подсистема инициализации и управления службами Systemd в групповых политиках «Альт» может запускать или останавливать службы. Systemd является гибким и функциональным инструментом, позволяет задавать расписание запуска служб, выполнять монтирование и многое другое. Через групповые политики «Альт» возможно управление самостоятельно созданной пользовательской службой.

Подсистема control в Linux-дистрибутивах является интерфейсом управления конфигурацией системы. Например, в перечне по умолчанию control в дистрибутивах «Альт» заложена конфигурация прав суперпользователя, включение в группу wheel, настройки

Samba-сервера и другое. Возможно написание собственных control-интерфейсов для изменения системных или прикладных настроек, управляемых в дальнейшем через групповые политики:

```
su            wheelonly (public wheel wheelonly restricted)
sudo          wheelonly (public wheelonly restricted)
sudoers       strict    (strict relaxed)
sudopw        default   (default root target runas)
sudoreplay    wheelonly (public wheelonly restricted)
sudowheel     disabled  (disabled enabled)
system-auth   local     (krb5 krb5_ccreds ldap local multi pkcs11 pkcs11_strict sss winbind)
system-policy local     (local remote)
```

Кроме перечисленных механизмов утилита gpupdate поддерживает работу со сценариями запуска скриптов при входе пользователя в систему, выходе из системы, загрузке и выключении компьютера. Logon/logoff/startup/shutdown сценарии формируются через раздел настроек системы («Preferences» в MS RSAT). В отличие от предыдущих примеров, этот случай не связан с подготовкой admx-файлов. Однако скрипты также позволяют шире использовать возможности дистрибутивов «Альт».

Вывод

Системные администраторы через групповые политики получают возможность добавлять правила Polkit, менять значения параметров базы реестра Dconf, запускать службы Systemd, управлять интерфейсами Control, либо запускать выполнение скриптов (поддерживаемых в системе). Также пользователи могут самостоятельно подготовить шаблоны admx-настроек для задействования графического редактора политик. В итоге получаем простой инструмент расширения возможностей групповых политик в системе.

Литература

[1] Групповые политики [ОС «Альт»] Руководство пользователя [Электронный ресурс]. URL: https://docs.altlinux.org/ru-RU/domain/10.1/html/group-policy/index.html

[2] «Альт Рабочая станция» 10.1 Руководство пользователя [Электронный ресурс]. URL: https://docs.altlinux.org/ru-RU/alt-workstation/10.1/html/alt-workstation/index.html

[3] «Альт Сервер» 10.1 Руководство пользователя [Электронный ресурс]. URL: https://docs.altlinux.org/ru-RU/alt-server/10.1/html/alt-server/index.html

[4] ALT Linux Wiki: ADMC [Электронный ресурс]. URL: https://www.altlinux.org/ADMC

[5] ALT Linux Wiki: GPUI [Электронный ресурс]. URL: https://www.altlinux.org/Групповые_политики/GPUI

[6] ALT Linux Team [Электронный ресурс]. URL: https://github.com/altlinux

Станислав Михалкович
Ростов-на-Дону, Институт математики, механики и компьютерных наук им. И.И. Воровича

Проект: PascalABC.NET https://pascalabc.net

Опыт разработки системы программирования PascalABC.NET как свободного ПО

Аннотация

В докладе рассматриваются опыт 15-летней разработки отечественной системы программирования PascalABC.NET, широко используемой в школьном образовании и для обучения современному программированию. Рассказывается об основных расширениях языка Паскаль, реализованных в языке PascalABC.NET, а также об использовании указанных конструкций в процессе обучения школьников и студентов. Анализируется опыт развития PascalABC.NET как свободного ПО и опыт интеграции PascalABC.NET в российские операционные системы. Особо отмечается поддержка PascalABC.NET в ОС «ALT» как первой российской операционной системе, включившей российскую систему программирования PascalABC.NET в основной репозиторий.

Система программирования PascalABC.NET создавалась и развивалась на мехмате Южного федерального университета на протяжении более 15 лет с 2007 г. [1–6]. Основная цель её создания — обучение современному программированию школьников и студентов. За основу был выбран популярный в то время язык Delphi Pascal, который уже на тот момент не устраивал преподавателей мехмата ЮФУ в силу несовременности языковых конструкций. В качестве платформы для реализации компилятора и среды была выбрана .NET Framework, что давало языку хорошие стандартные .NET–библиотеки, относительную простоту генерации кода и опору на популярный язык программирования C# и его технологии.

Система программирования PascalABC.NET развивалась как большой студенческий проект. К настоящему времени в проекте участвовали более 50 студентов мехмата ЮФУ, сам проект насчитывает более 1 миллиона строк кода. С 2015 года исходные коды PascalABC.NET выложены на Github под свободной лицензией LGPL v3. С этого момента исправлены около 2500 Issue, вышло около 20 Release-версий PascalABC.NET от 3.0 до 3.9.0.

Оболочка IDE PascalABC.NET — одновременно простая и мощная, обеспечивающая развитую систему Intellisense: всплывающие подсказки по точке, по скобке и при наведению на имя, а также автозавершение кода, автоформатирование кода и автоматическую оценку здоровья кода. Кроме того, создано ядро PascalABC.NET для Jupyter Notebook, что позволяет писать множество программ на одной странице с текстовыми пояснениями на Markdown.

Мощность языка PascalABC.NET существенно превосходит Delphi и приближается по языковым конструкциям к языку C#. В частности, в языке реализованы автовывод типов, кортежи, срезы массивов и строк, многомерные срезы, обработка исключений, обобщённые классы и подпрограммы, лямбда-выражения с захватом переменных из различных контекстов, последовательности как новый тип данных, генераторы последовательностей с использованием оператора yield, распаковка кортежей и последовательностей в переменные, ковариантные параметры обобщённых типов и многое другое.

Следует отметить, что система программирования PascalABC.NET пользуется популярностью в российских школах. Наряду с Python и C++ это третий по популярности язык на российских школьных олимпиадах по программированию, он также широко используется на компьютерном ЕГЭ по информатике.

В последние годы разработчики языка потратили много усилий на внедрение языковых и библиотечных конструкций PascalABC.NET, дающих возможность писать простой, лаконичный и понятный код, приближающийся по лаконичности к Python. Всё это позволило существенно изменить стиль программирования на PascalABC.NET, сделав его существенно более современным и выразительным. Благодаря грамотно подобранным стандартным функциям и методам расширения многие учебные задачи стали решаться в одну строку.

Изменился также спектр задач, предлагаемых обучающимся: они стали более высокоуровневыми. Например, ранее предлагаемая на олимпиадах задача «вывести первые по количеству населения 10

стран Азии, считав информацию о них из файла» теперь является одной из простейших в теме «массивы классов» и имеет компактное и ясное решение с использованием лямбда-выражений с захватом переменных и цепочек методов, причём, это решение легко понимается и воспроизводится школьниками.

Значительным изменениям подвергся также курс «Основы программирования» для студентов первого курса направления Фундаментальная информатика мехмата ЮФУ. Примерно со второго месяца обучения в этом курсе используются последовательности и лямбда-выражения, что позволяет сочетать императивный, объектный и функциональный стили программирования при обучении. Центральной идеей при обучении в первом семестре является лозунг «всё есть последовательность», что приводит к способности писать более простые и модифицируемые алгоритмы.

Примерно в 2016 году был модифицирован консольный компилятор `PascalABC.NET` так, чтобы он мог запускаться на Linux в среде Mono. Следует отметить, что для его реализации использовалась кросс-компиляция: сам компилятор компилировался под Windows, а запускался под Linux.

Однако перенос графической среды `PascalABC.NET` под Linux откладывался по техническим причинам. Основная проблема переноса заключалась в отсутствии технологии WPF под Linux, а также в том, что оболочка `PascalABC.NET` содержала сочетание технологий Windows Forms, WPF и некоторые нестандартные компоненты, которые были плохо адаптированы под Mono.

В 2022 году была предпринята успешная попытка переноса графической среды `PascalABC.NET` под Linux. Основным направлением усилий было избавиться в коде от используемых вызовов библиотеки WPF. Кроме того, выяснилось, что технология Windows Forms реализована под Mono не полностью — отсутствует несколько важных событий, в числе которых было событие OnActivate для оконных компонент. Поскольку без этого события не работала подсистема Intellisense `PascalABC.NET`, пришлось переписывать цепочку событий, эмулируя по существу событие OnActivate и несколько родственных событий.

В результате указанных усилий была в итоге реализована графическая среда `PascalABC.NET`, позволяющая под Linux работать ученикам в знакомом по Windows интерфейсе. Графическая среда `PascalABC.NET` была оттестирована в ряде Linux-систем, в частности, в российских Линуксах: Alt Linux, Astra Linux, Ред ОС, Rosa Linux и

МОС-Linux. Был исправлен ряд мелких ошибок и неточностей, приводящих к неправильному отображению элементов интерфейса в этих ОС.

Хочется особо отметить огромную помощь разработчиков Alt Linux, которые выявили ряд первоначальных багов, впоследствии устранённых. Кроме того, разработчики Alt Linux собрали rpm-пакет `PascalABC.NET` и включили `PascalABC.NET` в репозиторий в комплект стандартного предустановленного ПО в «Альт Образование» 10.1.

Литература

[1] *Михалкович С. С.* Учебная система программирования Pascal ABC / Научная конференция «Современные информационные технологии в образовании: Южный Федеральный округ». Материалы конференции. Ростов-на-Дону, 2004. С. 128–132.

[2] *Водолазов Н. Н., Михалкович С. С., Ткачук А. В.* Опыт разработки учебного языка программирования для платформы.NET//СИТО-2007. С. 71–73

[3] *Михалкович С. С.* Курс «Основы программирования» на базе системы `PascalABC.NET` / Научная конференция «Современные информационные технологии и ИТ-образование». Материалы конференции. Москва, 2009. № 5, С. 385.

[4] *Бондарев И. В., Михалкович С. С.* Система программирования `PascalABC.NET`: новые возможности 2015–16 гг. / Труды XXIII Научно-методической конференции «Современные информационные технологии: тенденции и перспективы развития». Ростов-на-Дону: Изд-во ЮФУ, 2016. С. 69–71.

[5] *Бондарев И. В., Михалкович С. С.* Система программирования `PascalABC.NET`: 15 лет развития / XXV Научная конференция «Современные информационные технологии: тенденции и перспективы развития». Материалы конференции. Ростов-на-Дону, 2018. С. 31–34.

[6] *Михалкович С. С.* Система программирования `PascalABC.NET`: 20 лет развития / XXX Научная конференция «Современные информационные технологии: тенденции и перспективы развития». Материалы конференции. Ростов-на-Дону, 2023. С. 287–290.

Кирилл Изместьев
Киров, ООО «Базальт СПО»

Свободное программное обеспечение, как инструмент изучения и развития языков народов России

Аннотация

Если вы представитель малого народа Росии, использующий операционную систему с закрытым исходным кодом, то маловероятно, что вам удастся добиться перевода операционной системы на родной язык, потому что крупным корпорациям это не интересно. Но благодаря открытости инструментов свободного ПО это возможно.

Все начинается с добавления локали

Локаль — это набор параметров, определяющий региональные настройки пользовательского интерфейса, такие как язык, страна, часовой пояс, набор символов, формат вывода даты, времени, используемая денежная единица и пр.

Установка региональных параметров является частью большого процесса локализации. При этом изменение региональных настроек часто доступно конечному пользователю системы без перепрограммирования программного обеспечения (ПО), в отличие от перевода сообщений пользовательского интерфейса, который часто требует изменения программного кода разработчиками ПО. Обычно код языка — это 2 символа. Но если код языка 3-хсимвольный, то могут быть проблемы (в установщике есть регулярные выражения, которые рассчитаны на 2 символа: выделяется первые два и опускается 3-й символ).

Уже есть локаль для: лугового марийского, чувашского, татарского и других... Создана локаль для языка коми. Планируется для горно-марийского языка. В перспективе есть лингвисты, заинтересованные в том, чтобы их язык тоже был в виде локали. Формат переменной:

```
LANGUAGE="mhr_RU:ru_RU.UTF-8"
```

Смысл в том, что можно в переменной задать не один язык, а два и более. Если нет перевода для первого языка, выбирается следующий. Достаточно сделать «недоперевод», ведь народы России хорошо знают русский язык. Компьютерные термины на своём языке могут

отсутствовать, и если в этом случае не будет перевода на родной язык, то это не только не критично, но и в отдельных случаях может быть даже плюсом.

Подбор фраз для перевода

Ресурсы на перевод обычно ограничены, поэтому перевести надо в первую очередь то, что пользователь чаще всего видит. Как определить список этих надписей? Для этого необходимо пропатчить glibc, чтобы выполнялось логирование при каждом переводе. Есть функция gettext в коде, и в эту функцию передаётся строка на английском языке. Потом в зависимости от переменной language выполняется поиск в таблице соответствий на том языке, на котором надо эту надпись отобразить. Можно в эту функцию вставить запись в файл и собрать профиль. У каждого пользователя профиль будет свой (список надписей, которые отображаются), но можно получить некоторый усреднённый список.

Как организовать перевод?

1. Автоматический перевод по словарю В большинстве программ есть, например, слова «открыть», «закрыть», «справка». Нет смысла переводить каждую — контекст совпадает, смысл не меняется. Так небольшими силами можно получить заметный эффект, который можно продемонстрировать. Программа автоматического перевода была написана на основе исходного кода проекта gettext.

2. Перевод через weblate.org Свободный сайт для удобного перевода можно установить на свой сервер и начать перевод. weblate.marsu.ru — на сайте Марийского государственного университета. Перевод планируется сделать силами студентов-филологов, есть договорённость с университетом.

Как внедрить перевод

До момента появления перевода в апстриме файлы .mo будут в отдельном пакете.

Национальная раскладка клавиатуры. Добавление в систему по умолчанию.

Проверка орфографии — должно быть по умолчанию. Для марийской орфографии уже есть пакет в репозитории ОС «Альт» — hunspell-mhr

Пункты главного меню (переводятся иначе)

В xfce эти пункты переводятся с английского языка на национальный без учёта переменной language. Был пропатчен пакет libxfce4util, так как использовался Simply Linux. Для других DE это будет отличаться. Если какой-то из пунктов не переведён на марийский язык, то он должен отображаться на русском (без патча будет английский язык). В идеале, патч должен проверять переменную language, делить её на части (через: указаны локали, переменную надо поделить на список локалей - выделить в массив) и желательно, чтобы это было принято в апстриме. В планах доработать.

Сборка установочного образа:

Изменение установщика: в установщик добавить свой язык. В итоге получается отдельный установочный образ с включёнными по умолчанию локалью, словарями проверки орфографии, файлами переводов .mo.

В основном приложения gtk. QT приложения локализуются отдельно, иным способом.

Своя спецификация локализации у LibreOffice и Firefox.

Опыт можно тиражировать на другие языки.

Надежда Кострюкова, Антон Руфф
Москва, АНО «Открытый код»

Проект: Проект АНО «Открытый код» `https://russiaos.ru`

Российский open source: community, бизнес, государство

Аннотация

Доклад посвящён направлениям потенциального взаимодействия community open source, бизнеса и государства. Уделено внимание текущим потребностям ОСПО-сообщества и возможностям их интеграции в повестку поддержки со стороны государства.

1. Эффективное развитие open source — это прежде всего конкретные разработчики, сообщества и бизнес. При этом многие часто развивают проекты просто на собственном энтузиазме, чтобы сделать полезный вклад в понравившийся проект.

2. Настоящий момент характеризуется, с одной стороны, тем, что российские разработчики давно и активно участвуют в зарубежных community и проектах. По данным JetBrains, на РФ приходится 7% контрибьюторов международных проектов с открытым исходным кодом, это 5 место в мире после США, Китая, Индии и Японии. В том числе развивают свои решения на основе форков наиболее популярных open source-проектов.

3. С другой стороны, с каждым годом растут риски использования зарубежной инфраструктуры для развития собственных open source-проектов российских разработчиков. Всем известны случаи блокировки, перевода в архив репозиториев российских разработчиков, включения в программу вредоносных кодов и др.

4. В этой ситуации очевидна потребность поддержки отечественных разработчиков со стороны государства. Такая поддержка в первую очередь должна включать создание отечественной инфраструктуры, защиту интересов разработчиков и community, формирование культуры работы с открытым ПО, в том числе через образование и проекты по развитию кадров, совершенствование нормативного и методического регулирования сферы open source.

5. Фактически open source является одним из важных направлений развития цифровой экономики государства. В связи с чем требуется комплексный подход к его развитию на государственном уровне.

6. Заинтересованность государства в развитии отечественного open source подтверждается опубликованным 03.09.2023 г. перечнем поручений Президента Российской Федерации по итогам встречи с учёными и пленарного заседания Форума будущих технологий, прошедших 13.07.2023 года. В числе поручений Правительству Российской Федерации обозначена необходимость создания до 01.06.2024 г. «отечественных хранилищ кода (платформ и сервисов), необходимых для совместной работы российских и иностранных программистов».

7. Комплексное развитие open source может быть реализовано через утверждение на государственном уровне «Стратегии развития программного обеспечения с открытым кодом в России». Но содержание Стратегии не должно навязываться государством —

инициатива должна исходить от ИТ-сообщества и отражать реальные интересы разработчиков и бизнеса в сфере open source.

8. Первоначально такой проект Стратегии был разработан сообществом ещё в конце 2021 года. В рамках работы Экспертного совета АНО «Открытый код» проект Стратегии был усовершенствован с учётом мнения ведущих специалистов отрасли.

9. Проект Стратегии содержит все перечисленные ранее направления, а также обеспечение системной координации реализации Стратегии, создание экосистемы разработки открытого ПО, стимулирование и поддержку развития сферы open source.

10. Ключевыми вехами реализации Стратегии являются:

- Создание Национального репозитория и включение его в экосистему репозиториев (государственных, публичных, частных), реализованных на территории России.
- Определение ответственной организации за взаимодействие и работы с сообществом.
- Реализация мероприятий по формированию культуры работы с открытым и свободным ПО.
- Снятие барьеров и рисков, содержащихся в российском законодательстве.
- Запуск целевых мер финансовой и методической поддержки разработчиков и open source-проектов.

1. Для реализации каждого из направлений требуется решение определённого круга задач, при условии поддержки конкретных шагов сообществом.

2. Так, например, функционал Национального репозитория должен не только стремиться стать аналогом GitHub, но и решать конкретные задачи российских разработчиков. Важным является обеспечение зеркалирования зарубежных репозиториев.

3. Мировой опыт подтверждает необходимость вклада большого числа представителей сообщества и бизнеса в координацию усилий и популяризацию open source, что обуславливает необходимость наличия организации для координации усилий сообщества.

4. При совершенствовании законодательства необходимо опираться на практику ведения проектов, чтобы не допускать появления таких же рисков, как в случае с принятием поправок в Федеральный закон об НКО.
5. По проведённому опросу среди членов ЭС более 71% респондентов заявили, что существующих мер поддержки open source-проектов недостаточно. Важными направлениями поддержки могут являться поддержка в соблюдении требований лицензирования, стимулирование участие зарубежных разработчиков из дружественных стран в российских open source-проектах, внедрение практики регулярного анализа состояния эффективности мер поддержки развития сферы open source.
6. В части развития образования и подготовки кадров можно отметить, что на сегодня отсутствует специализированная подготовка ВУЗами программистов в сфере open source. Специалисты учатся либо на практике в реальных проектах, либо проходя немногочисленные специализированные курсы.

Андрей Михеев, Андрей Потапов
ООО «Процессные технологии» и Финансовый университет, ООО «Процессные технологии»
http://www.runawfe.org/

Свободный документооборот на основе платформы RunaWFE Free

Аннотация

В докладе показана разработанная в последних версиях платформы RunaWFE Free функциональность «Бизнес-процессы электронного документооборота», относящаяся к внутреннему электронному документообороту предприятия. Выпуск неквалифицированной ЭЦП. Подписание документа присоединённой ЭЦП и проверка подписи документа различными пользователями.

Платформа RunaWFE Free и процессный подход

RunaWFE Free основана на процессном подходе к управлению предприятием. В соответствии с этим подходом деятельность пред-

приятия представляется в виде исполняющихся экземпляров бизнес-процессов. Каждый экземпляр содержит схему бизнес-процесса, по которой перемещаются точки управления. В момент прихода точек управления в определённые узлы схемы генерируют задания исполнителям. Схемы бизнес-процессов должны разрабатываться так, чтобы задания исполнялись скоординировано и в нужном порядке. Процессный подход позволяет повысить производительность труда, прозрачность бизнеса, даёт возможность быстрого изменения бизнес-процессов предприятия в ответ на изменение условий бизнеса.

Новая функциональность RunaWFE Free, связанная с электронно-цифровой подписью

Платформа была интегрирована с библиотеками Bouncy Castle (для работы с ЭЦП) и Pdfbox (для работы с pdf-файлами). Взаимодействие с библиотеками реализовано через графические элементы форм и задачи-сценарии (объекты, располагаемые на схеме бизнес-процесса).

В установленной системе генерируется корневой сертификат ЭЦП. Его свойства можно посмотреть в меню «Система» (см. Рисунок 1).

Рис. 1: Информация о корневом сертификате в меню «Система».

Далее для любого пользователя средствами платформы можно сгенерировать сертификат и закрытый ключ ЭЦП. Информация об ЭЦП пользователя добавлена в его свойства (см. Рисунок 2).

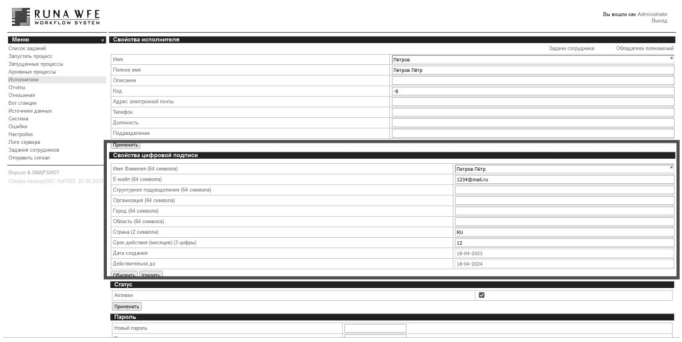

Рис. 2: Информация об ЭЦП пользователя.

Подписание документа производится в обработчике задачи-сценария, который обращается к контейнеру PKCS#12 пользователя. Проверка подписи производится в другом обработчике задачи-сценария, тоже с обращением к контейнеру PKCS#12 пользователя. Закрытые ключи пользователей и корневого сертификата хранятся в базе данных в виде PKCS#12 контейнеров сериализованных в массив байтов.

В настоящее время платформа позволяет подписывать только PDF-документы.

Реализация документооборота на платформе RunaWFE Free

В системе разрабатывается бизнес-процесс, содержащий узлы подписания и проверки электронной подписи. (см. Рисунок 3).

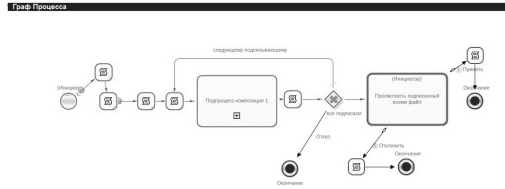

Рис. 3: Бизнес-процесс согласования документа.

После подписания PDF-файла кроме самой ЭЦП в его текст вносится информация (см. Рисунок 4). Подпись является присоединённой к документу. Каждая последующая подпись подписывает документ вместе со всеми предыдущими подписями.

Рис. 4: Документ, подписанный ЭЦП.

На рисунках 5 и 6 представлена стартовая форма бизнес-процесса согласования документа и форма одного из его заданий. Задание позволяет пользователю как подписать, так и отклонить документ, а также содержит информацию о других участниках документооборота.

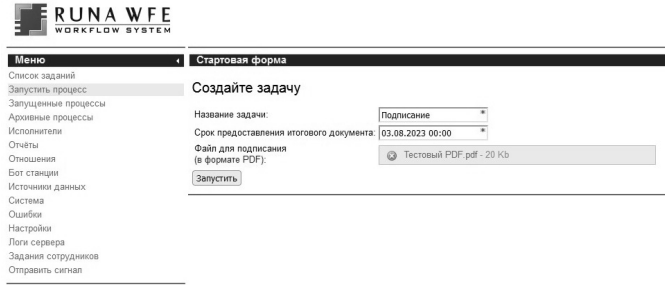

Рис. 5: Стартовая форма бизнес-процесса согласования документа.

Также возможен режим использования RunaWFE Free, при котором криптографические средства не встраиваются в систему, а используются независимо. Система при этом только генерирует задания

Рис. 6: Форма одного из заданий бизнес-процесса согласования документа.

на подписание документов и хранит подписанные файлы. В качестве криптографической программы для этого можно использовать, например, графическую утилиту Kleopatra. Однако это будет неудобно, т. к. файлы для проверки и подписания надо будет всё время передавать в Kleopatra через файловую систему.

Михаил Смирнов
Москва, НИУ ВШЭ, ООО «Процессные технологии»

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

Проект RunaWFE Free. Разработка элемента «событийный подпроцесс»

Аннотация

Для расширения возможностей реагирования на события в рамках процессного управления был разработан элемент «событийный подпроцесс» для свободного ПО RunaWFE Free. Элемент соответствует стандарту BPMN 2.0 и может быть использован для описания асинхронных задач. Событийный подпроцесс стартует каждый раз при возникновении события или по истечению времени таймера и не блокирует выполнение основного процесса.

События в процессном управлении

Процессный подход рассматривает предприятие как множество выполняющихся экземпляров бизнес-процессов. Процессный подход

может эффективно применяться в компаниях или государственных органах, где работники выполняют множество повторяющихся в известной последовательности операций. Выполнение процесса в компьютерной среде позволяет повысить скорость взаимодействия сотрудников, а также даёт предприятиям ряд дополнительных преимуществ [1]. Программы, в которых создаются и исполняются бизнес-процессы, называются системами управления бизнес-процессами. Одной из таких программ является российское свободное ПО с отрытым исходным кодом RunaWFE Free [2].

Для взаимодействия экземпляров бизнес-процесов между собой в системах управления бизнес-процессами используются события. При таком взаимодействии одни экземпляры генерируют события, а другие их обрабатывают. В RunaWFE Free межпроцессное взаимодействие реализовано на основе Java Message Service. До настоящей работы оно выполнялось при помощи элементов двух типов: «Генерация события» и «Обработка события». Элементы располагаются на схеме бизнес-процесса. При прохождении точки управления через элемент «Генерация события» элемент генерирует события. При приходе точки управления в узел «Обработка события» она остаётся в этом узле до наступления соответствующего события, а после его наступления — переходит дальше по схеме бизнес-процесса.

Однако существуют ситуации, в которых определённые события должны быть обработаны без привязки к элементу, находящемуся на схеме основной ветки бизнес-процесса. Для этого в стандарте BPMN 2.0 [3] используется событийный подпроцесс, который не связан переходами с узлом — началом бизнес-процесса, и может быть активирован по сигналу в любой момент при условии, что экземпляр бизнес-процесса активен (не завершён). В данной работе для платформы RunaWFE Free был реализован такой элемент. Реализован только непрерывающий вариант событийного подпроцесса, не останавливающий ход основного процесса при наступлении стартового события для подпроцесса.

Реализация элемента «событийный подпроцесс» в редакторе бизнес-процессов RunaWFE Free

Рассмотрим пример «Подготовка конференции> (Рис. 1 и Рис. 2). В основном процессе выполняются задачи, необходимые для прове-

дения конференции. Параллельно могут приходить заявки от слушателей и докладчиков. Заявки будут приниматься до тех пор, пока процесс не завершится, т. е. пока не наступит конец регистрации. Событийные подпроцессы, в отличие от всех остальных элементов, не имеют входящих или исходящих переходов. На схеме они обведены пунктирной рамкой. Граница начального узла в событийном подпроцессе также пунктирная.

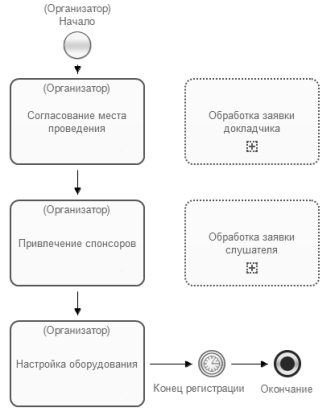

Рис. 1: Процесс подготовки конференции с событийным процессом

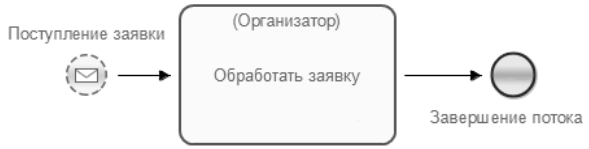

Рис. 2: Схема событийного подпроцесса обработки заявки докладчика

За событийный подпроцесс отвечают две сущности: схема подпроцесса и элемент на схеме. Так как редактор основан на Eclipse, все элементы описываются в plugin.xml. Для каждого элемента определяются название, иконка и класс, который его представляет. Эле-

мент событийного подпроцесса реализован как новый класс, потомок подпроцесса-композиции, который не содержит никаких новых полей и методов. Такой пустой класс удобен, так как архитектурно предполагается, что все элементы имеют разные классы. Типы элементов проверяются через «Java Reflection».

Начало событийного подпроцесса должно содержать обработку события или таймер. При сохранении процесса с начальным узлом без события редактор сообщает об ошибке (Рис. 3).

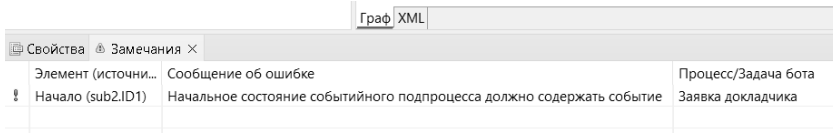

Рис. 3: Ошибка отсутствия события в начальном элементе

Реализация элемента «событийный подпроцесс» на сервере RunaWFE Free

Событийный подпроцесс принимает сообщения только когда активен родительский процесс (то есть в нём есть хотя бы одна активная точка управления). Были рассмотрены 2 варианта реализации:

- Событийный подпроцесс соединяется невидимым переходом с началом «родителя», что делает его обычным элементом на схеме. Его стартовый узел после обработки события не перемещает точку управления, а генерирует новую. Когда точка управления попадает в завершение потока родительского процесса, осуществляется проверка активности. Если «родитель» не активен, то все точки управления из стартовых узлов событийных подпроцессов удаляются.
- Создаётся таблица с триггерами. При старте процесса в таблицу добавляется триггер на каждый стартовый узел событийного подпроцесса. Обработчик событий выбирает триггеры, подходящие по селектору сообщения, а среди них те, которые должны принимать сообщения. В узлах этих триггеров генерируются точки управления.

Был выбран второй вариант, так как фиктивные точки управления из первого варианта имеют побочные эффекты. Более того, триггеры не влияют на процессы, в которых нет событийных подпроцессов.

Следует обратить внимание, что события принимаются только когда активен непосредственный «родитель» событийного подпроцесса. Таким образом, если сам процесс «подготовка конференции» является подпроцессом-композицией в другом процессе, то логика принятия заявок не будет нарушена.

Таблица триггеров добавляется в список миграций, чтобы существующие базы могли обновится до новой версии. Она содержит только идентификатор процесса, название узла и селектор сообщений. Селектор определяет правила маршрутизации сообщений, а остальные столбцы необходимы для генерации точки управления.

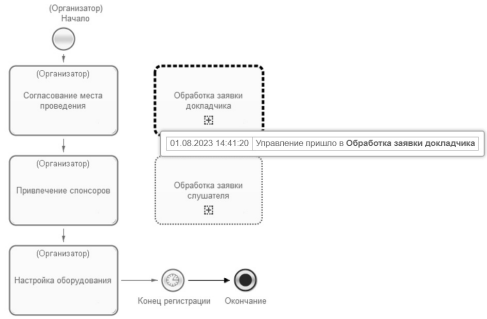

Рис. 4: Активный событийный подпроцесс

Активный событийный подпроцесс выделяется пунктирной чёрной рамкой (Рис. 4), выполненный — зелёной рамкой (Рис. 5). На сервере рамки около элементов рисуются поверх изображения процесса из редактора. Для определения состояния элемента используется журнал переходов точки управления. В начале всем элементам процесса присвоено состояние «не активный». Журнал просматривается в хронологическом порядке, и каждая его запись меняет состояние инцидентных переходу элементов. Начало перехода меняет состояние на «выполненный», а конец на «активный». В результате получаем актуальные состояния всех элементов. В алгоритме используется журнал,

так как различить состояния «неактивный» и «выполненный», зная только текущее состояние процесса, затруднительно.

Событийный подпроцесс не имеет входящих переходов, поэтому в журнале не существует записи о переходе, в котором он является концом. Для проверки активности событийного подпроцесса используется специальный NodeEnterLog, в который добавляется запись сразу, как только сгенерированная точка управления появляется в стартовом узле событийного подпроцесса.

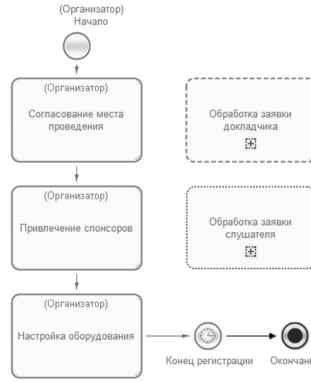

Рис. 5: Выполненный событийный подпроцесс

Заключение

Для поддержки элемента стандарта BPMN 2.0 «событийный подпроцесс» в данной работе были реализованы изменения в клиентской и серверной частях платформы RunaWFE Free. Изменения загружены на портал github. Серверная часть доступна по ссылке `https://github.com/processtech/runawfe-free-server`. Клиентская часть (редактор бизнес-процессов) — `https://github.com/processtech/runawfe-free-devstudio`.

Литература

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

[2] Сайт проекта RunaWFE Free [Электронный ресурс]: http://runawfe.org/

[3] BPMN версия 2.0, стр. 176-178, [Электронный ресурс]: https://www.omg.org/spec/BPMN/2.0/PDF

Владислав Сизов
Москва, НИУ ВШЭ, ООО «Процессные технологии»
Проект: RunaWFE Free https://runawfe.org

Реализация работы бота с внутренним хранилищем в свободной системе управления бизнес-процессами RunaWFE Free

Аннотация

В системе RunaWFE Free работать с внутренним хранилищем данных можно как путём использования элементов нотации BPMN «задача-сценарий», так и при помощи ботов (автоматических исполнителей заданий). Для задач-сценариев в системе RunaWFE Free существует удобный способ работы с таблицами внутреннего хранилища и переменными бизнес-процессов. В данной работе аналогичный способ работы с данными внутреннего хранилища был реализован для ботов. В отличие от задачи-сценария, которая является частью определённого бизнес-процесса, бот не может обращаться к переменным бизнес-процесса напрямую. Поэтому для бота работы с внутренним хранилищем были добавлены глобальные разделы, содержащие переменные и типы данных.

Введение

По сравнению с задачами-сценариями у ботов в системе RunaWFE Free [1] есть два преимущества:

1. Один бот (с одной настроенной конфигурацией) может работать с разными бизнес-процессами. Если для этого использовать задачи-сценарии, то в каждом бизнес-процессе их конфигурации придётся дублировать, что приводит к сложностям при сопровождении и изменении работы системы.
2. Боты, в отличие от задач-сценариев, при работе с данными могут поддерживать транзакции, основанные на принципах процессного управления [2].

В случае задач-сценариев для работы с таблицами внутреннего хранилища в бизнес-процессе для каждой используемой таблицы создаётся пользовательский тип переменной, подтипы которого соответствуют типам полей таблицы. Далее при выполнении действий с таблицами (Select, Insert, Delete, Update) в специальных формах из списков выбираются и сопоставляются поля таблиц, соответствующие переменные бизнес-процесса и логические операции между ними. В отличие от задачи-сценария бот не является частью бизнес-процесса и не может обращаться к переменным бизнес-процесса напрямую, т. к. бот может работать с разными бизнес-процессами, в которых переменные могут иметь различные названия. Поэтому в данной работе для бота была добавлена возможность использовать глобальные разделы, из которых в конфигурациях действий с таблицами выбираются соответствующие переменные и в дальнейшем используются как формальные параметры задач ботов. А при назначении задач ботам в бизнес-процессах этим формальным параметрам ставятся в соответствие переменные бизнес-процессов (т. е. фактические параметры).

Реализация

Для решения проблемы в системе RunaWFE Free был создан обработчик для бота для взаимодействия с внутренним хранилищем:

1. В качестве таблицы БД выбирается любой тип данных из внутреннего хранилища.
2. На основе переменных таблицы создаются переменные бота
3. На основе переменных бота создаётся глобальный раздел, который доступен во всех бизнес-процессах проекта.

Работу с внутренним хранилищем было решено реализовать следующим образом:

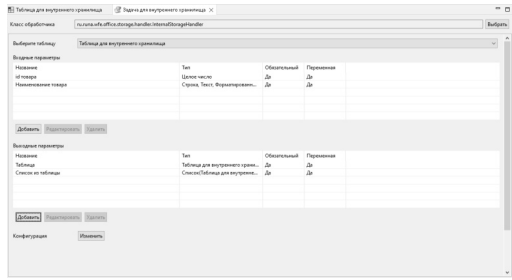

Рис. 1: Конфигурация задачи бота.

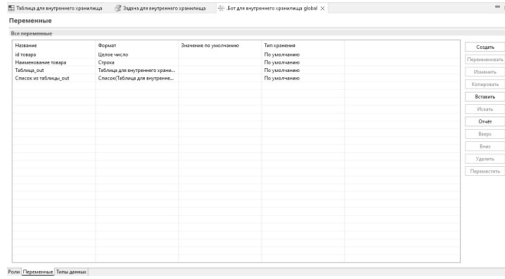

Рис. 2:

1. Входные параметры бота должны состоять из переменных таблиц.
2. Выходным параметром должна быть сама таблица в случае операций INSERT, DELETE, UPDATE. В случае операции SELECT – список хранящий тип данных таблицы.

При реализации описанной функциональности были использованы следующие технологии:

- Java 8;
- Spring 3.1.2;
- Lombok 1.18;
- Dom4j 1.6.1.

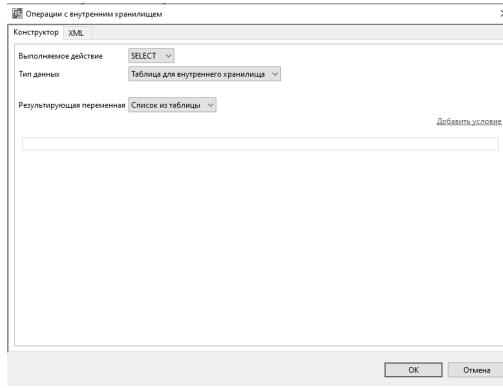

Рис. 3: Конфигурация обработчика в задаче бота.

Результаты

На иллюстрациях 3–4 представлены скриншоты проекта, позволяющего использовать бота для работы с внутренним хранилищем. Структура конфигураций сильно упрощена за счёт автоматического создания глобального раздела бота, который будет использоваться для передачи данных в конфигурацию задачи бота.

Заключение

В результате создания глобального раздела бота пользователям стало гораздо удобнее настраивать бота для работы с внутренним хранилищем системы.

Литература

[1] Михеев А. Г., Орлов М. В. Система управления бизнес-процессами и административными регламентами. // Программные продукты и системы, № 3, 2011

[2] Михеев А. Г. Проект RunaWFE Free. Реализация транзакций для курса процессного управления предприятием — в кн.: Четырнадцатая конференция Свободное программное обеспечение в высшей школе. Матери-

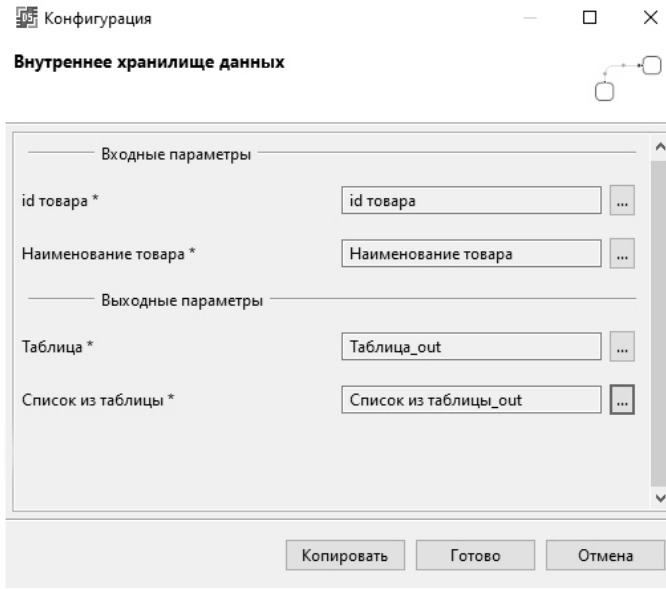

Рис. 4: Схема конфигурации задачи бота в бизнес-процессе.

Рис. 5: Схема бизнес-процесса.

алы конференции / Переславль, 25–27 января 2019 года. М.: МАКС Пресс., 2019.

[3] Сайт проекта RunaWFE Free: `http://runawfe.org/`

Андрей Савченко
Москва, ООО «Базальт СПО»
https://www.altlinux.org/ALT_Mobile

ALT Mobile

Аннотация

В докладе делается обзор проекта ALT Mobile, позволяющего использовать смартфон на СПО на базе Sisyphus, без зависимости от проприетарных сервисов и драйверов. Обсуждаются требования к оборудованию, функциональные возможности, проблемы и направления развития.

Введение

В то время как почти вся мобильная экосистема захвачена проприетарной дуополией, существенно ограничивающей свободу пользователей на распоряжение собственными устройствами и ПО на них, всё большее развитие получают альтернативные проекты, позволяющие использовать современный телефон без проприетарного ПО, за исключением прошивок некоторых периферийных устройств.

Команда ALT Linux Team присоединилась к этому начинанию в рамках проекта ALT Mobile[1].

Аппаратная платформа

Основным требованием к совместимым устройствам является требование необходимости и достаточности свободного ПО для базовой функциональности устройства. Таким образом, вся базовая функциональность основной операционной системы должна реализовываться с помощью исключительно свободного ПО, включая драйвера устройств.

Это жёсткие ограничения, но именно они позволяют обеспечить свободу и безопасность пользователей. На данный момент поддерживаются устройства на базе СнК rk3399s, но ведётся работа и над поддержкой других СнК.

В рамках проекта доступны образы[2] как для pinephone pro, так и для виртуальных машин архитектур x86_64, aarch64 и riscv.

Графический интерфейс

На текущем этапе развития проекта основным графическим окружением выбрана оболочка Phosh (phone shell), основанная на библиотеках Gnome shell.

В ходе портирования возникли нетривиальные проблемы, связанные с использованием в ОС «Альт» механизма tcb[3] вместо обычного shadow, которые были успешно решены путём доработки[4] phosh.

Прикладное ПО

Представлен широкий спектр прикладного ПО как для связи (gnome calls, chatty, gnome contacts, telegram), так и для повседневных задач: карты (gnome maps), мультимедиа (mpv, totem, amberol, lollypop), работа с камерой (megapixels, cheese), браузеры (chromium, firefox), почта (geary), календари, калькулятор, погодный виджет и т.п.

При необходимости можно использовать механизм контейнерной изоляции Waydroid для запуска Android приложений. Но этот механизм не рекомендуется вследствие худшего контроля над ним и закрытости типовых приложений. Грубо говоря, ситуация аналогична использованию wine для запуска приложений Windows в Linux.

Проблемы

В основном, проблемы разделяются на два класса: недостаточная адаптация приложений для мобильного интерфейса и проблемы с энергосбережением. Некоторые приложения не адаптированы вовсе (например, офисные пакеты). Работы в этих направлениях ведутся. Следует отметить, что подобные проблемы характерны для всех дистрибутивов Linux на мобильный устройствах с фокусом на СПО и решение их достигается совместными действиями.

Так же есть системная проблема, когда системно значимые приложения существуют только в закрытом виде, например, клиенты для работы с Системой Быстрых Платежей, да и то лишь в виде Android приложений. Здесь требуются уже не только технические, но и административные решения.

Планы

Рассматривается возможность поддержки других оболочек в будущем (kde plasma mobile, swmo, gnome mobile). Возможна поддержка устройств на другом оборудовании. Требуется взаимодействие с индустрией по её переориентации на создание свободных приложений под Linux.

Литература

[1] ALT Mobile
`https://www.altlinux.org/ALT_Mobile`

[2] Образы ALT Mobile
`http://beta.altlinux.org/mobile/latest/`

[3] TCB authentification system
`https://www.openwall.com/tcb/`

[4] Проблема разблокировки экрана в phosh
`https://bugzilla.altlinux.org/46389`

Евгений Синельников
Саратов, ООО «Базальт СПО»

Проект: Конфигурирование операционной системы `https://gitlab.basealt.space/alt/alterator-manager`, `https://gitlab.basealt.space/alt/alterator-module-executor`, `http://git.altlinux.org/people/sav/packages/alt-diagnostic-tool.git`, `https://www.altlinux.org/Alterator` (будет актуализировано до конца 2023 года)

Концептуальный подход к развитию новой версии Альтератор

Аннотация

Доклад посвящён анализу текущего состояния разработки новой версии конфигуратора для операционных систем семейства «Альт», а также концептуальным особенностям и техническим возможностям его реализации на базе шины Dbus. Раскрываются планы разработки на ближайшее будущее, включающие смену стека технологий бекенда

и предоставление возможности реализации фронтендов на различных языках программирования с различными графическими и консольными фрейворками, которым предоставлен гибкий, расширяемых набор программных интерфейсов.

Проект Альтератор создавался, как «новое поколение платформ для разработки сложных систем». Оригинальная реализация проекта Альтератор написана в рамках модульного подхода «по большей части Scheme (диалект языка Lisp, в реализации Guile), C и старый добрый sh+awk». До последнего времени является основным базовым компонентом для построения дистрибутивных решений «Альт» («используется как инсталлятор и конфигуратор системы»). Оригинальная архитектура проекта предполагает возможность парного расширения модулей на языке программирования Scheme для «фронтендов» (под графический интерфейс на базе Qt и под Web-интерфейс на базе встроенного, отдельно устанавливаемого web-сервера) и на языке Bash для «бэкендов» (со взаимодействием через собственную шину передачи сообщений — woo-bus).

Основными причинами необходимости перевода проекта Альтератор на новый технологический стек на базе шины Dbus явились:

- необходимость спецификации интерфейсов «бэкендов» для создания «фронтендов» в различных графических и консольных приложениях;
- возможность защищённого, контролируемого выполнения привилегированных действий из-под непривилегированного окружения (команд из-под root'а в графических приложениях без необходимости ввода только «root'ового пароля», что определяется настройками);
- интеграция шины dbus со всеми системными компонентами в современных Linux-дистрибутивах (начиная от systemd и заканчивая всеми графическими окружениями для управления, например, такими сервисами как UDisks2, NetworkManager, а также отдельными сервисами systemd);
- расширение числа заинтересованных разработчиков в развитии проекта, позволяющего конфигурировать операционные системы «Альт»;
- предоставление разработчикам широких возможностей в виде «фреймфорка» для создания собственных приложений, управляемых как локально, так и по сети;

- обобщение механизмов, используемых как для локального, так и удалённого управления, в том числе и через инструменты конфигурациями (такие как ssh, ansible и групповые политики развиваемые в рамках проекта «Альт Домен»).

Архитектура новой реализации проекта Альтератор опирается на опыт конфигурирования операционных систем «Альт» и концептуальный подход, заложенный в нём. В первую очередь этот подход позволяет сохранить при использовании конфигуратора возможность традиционного управления в консоли путём ручного редактирования конфигурационных файлов. То есть конфигурирование обеспечивается не путём создания специальной базы данных, а путём разбора и автоматизированного редактирования ровно тех же файлов, которые редактируются системным администратором вручную. Кроме того, расширение модулей спроектировано, в большей степени, под декларативный сценарий, обеспечивающий возможность привлечения к разработке и расширению возможностей конфигурирования наибольшего количества заинтересованных «мейнтейнеров» (участников команды ALT).

В целом, развитие возможностей протокола и «фреймворков» работающих через шину Dbus позволяет полностью покрыть весь набор необходимых для реализации такого конфигуратора возможностей. А стандартизация использования этой шины позволяет использовать любые современные языки программирования без необходимости создания и сопровождения специальных связующих компонент, поскольку все необходимые «батарейки» для шины Dbus встроены во все современные дистрибутивы. Что дополнительно открывает широкие возможности для применения новой реализации проекта Альтератор в различных дистрибутивных решениях.

Архитектура проекта Альтератор на Dbus

Ключевым звеном старого проекта Alterator является служба alteratord, реализующая как взаимодействие через шину woo-bus, так и запуск бэкендов в виде отдельных процессов, обрабатывающих запросы на шине. Ключевым же звеном новой реализации является служба alterator-manager, которая обеспечивает подключение к шине Dbus, разбор декларативных описаний интерфейсов для модулей Аль-

тератора и загрузку бинарных модулей, обрабатывающих запросы на объектах Dbus для загруженных из описаний интерфейсов.

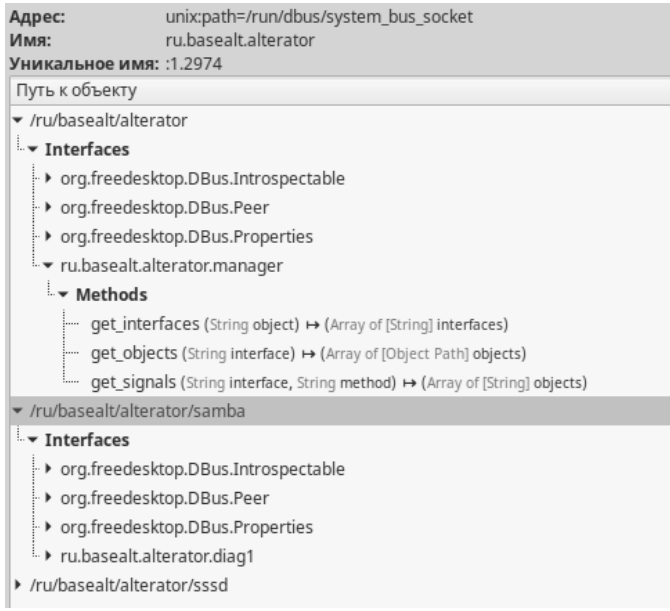

Рис. 1: Интроспекция объектов и интерфейсов

Основной набор модулей новой версии Альтератор:

- **executor** — ключевой модуль, который обеспечивает декларативное расширение перечня объектов и интерфейсов, позволяющих через отдельные методы запускать заданные приложения с заданными параметрами (переводит «Unixway» на уровень шины dbus);
- **fileio** — обеспечивает возможность контролируемого доступа на чтение и запись к системным файлам и каталогам за счёт возможности передачи открытых файловых дескрипторов. Работает только в локальном режиме (в разработке);
- **registry** — обеспечивает обобщённый доступ и хранение конфигурационных параметров в рамках единого пространства имён — аналога реестра Windows (в разработке);

- **remote** — обеспечивает удалённое управление узлами, на которых установлен новый Альтератор с транспортом по протоколу ssh и возможностью аутентификации через kerberos (в разработке).

На текущий момент базовый каркас в виде пакетов `alterator-manager` и `alterator-module-executor` опубликован и собран в репозиторий «Сизиф». Данный каркас обеспечивает следующий набор возможностей:

- в рамках базового интерфейса ru.basealt.alterator.manager — поиск любых объектов Альтератор, предоставляющих заданный интерфейс, а также вывод перечня интерфейсов предоставляемых заданным объектом Альтератор;
- в рамках расширенного модуля executor — возможность создания любого количества объектов, реализующих заданные интерфейсы через «Unixway» на Dbus;
- для всех создаваемых интерфейсов — генерировать базовый ограниченный набор правил PolicyKit;
- контролировать соответствие интерфейсов создаваемых для объектов заданному шаблону (имена методов и их сигнатуры).

В качестве доказательства работы концепта реализовано два клиентских приложения:

- alt-diagnostic-tool (adt) — графическая утилита, обеспечивающая поиск и загрузку гибко расширяемого набора диагностических инструментов;
- новая реализация alterator-control-center (в разработке), которая должна обеспечить прозрачный запуск как графических модулей Альтератора старой версии, так и новых графических интерфейсов, обращающихся к интерфейсам на Dbus и управляемых через групповые политики.

Таким образом, представленный набор компонент предлагается рассмотреть с целью постепенной замены сначала графической части старой версии конфигуратора, затем графической части инсталлятора. При этом возможность удалённого управления модулями Альтератора позволяет обеспечить возможность конфигурирования без установки браузера и использования для конфигурирования web-протоколов. Дальнейшее развитие этого решения также может быть

представлено в виде отдельного proxy-сервиса для написания полноценного, расширяемого web-интерфейса. Ключевым вопросом для конфигурирования через web остаётся делегирование расширенных полномочий web-серверу, делающее его дополнительной точкой уязвимости.

Важным шагом для расширения количества разрабатываемых модулей и клиентских приложений, а также повышения общей зрелости предлагаемой к рассмотрению реализации, является подробное документирование как архитектурных особенностей, так и возможностей для разработчика.

Иван Савин
Саратов, ООО «Базальт СПО»

Проект: alterator-manager, alterator-module-executor
`https://git.altlinux.org/gears/a/alterator-manager.git`,
`https://git.altlinux.org/gears/a/alterator-module-executor.git`

Alterator-manager

Аннотация

Доклад посвящён модульному сервису alterator-manager и пока единственному его модулю — executor. Назначение сервиса — управление операционной системой, её настройками. Модуль executor предназначен для запуска программ и скриптов, его можно сравнить с sudo. Далее немного подробнее о реализации и использовании.

Alterator-manager — это сервис для управления операционной системой через D-Bus. Его функционал реализуется в модулях, а интерфейс описывается в специальных конфигурационных файлах.

Как работает alterator-manager

Alterator-manager может быть запущен в двух режимах — обычном и пользовательском:

```
$systemctl start alterator-manager.service
```

или

```
$systemctl --user start alterator-manager.service
```

В первом случае сервис запускается с полномочиями суперпользователя (root) и регистрирует имя (`ru.basealt.alterator`) на системной шине D-Bus. Во втором случае сервис запускается от обычного пользователя и регистрирует имя на сессионной шине D-Bus. После запуска происходит считывание конфигурационных файлов из `/usr/share/alterator/backends/` и загрузка модулей из `/usr/libexec/alterator/`. Конфигурационные файлы — это glib-овские key-value файлы, они должны иметь расширение `.backend` для обычного режима работы и `.user.backend` для пользовательского. На их основе формируется интроспекция для D-Bus и правила polkit (только в обычном режиме).

Формат конфигурационных файлов

```
[Manager]
module_name = executor
node_name = example
interface_name = example
thread_limit = 5
[make_ls2]
execute = ls -al {param}
[make_ls3]
execute = ls -al {param}
[example]
execute = ls -al {param}
stdout_strings = enabled
stdout_bytes = enabled
stderr_strings = enabled
stdout_signal_name = stdout_signal_example
stderr_signal_name = stderr_signal_example
stdout_byte_limit = 200000
stdout_strings_limit = 200000
stderr_strings_limit = 200000
thread_limit = 3
action_id = method-name
```

Первая секция `Manager` обязательная, она имеет следующие пары ключ-значение:

- `module_name` — имя модуля который будет обрабатывать запросы;
- `node_name` — имя узла в поддереве D-Bus;
- `interface_name` — имя интерфейса D-Bus;
- `thread_limit` — максимальное число тредов для данного интерфейса.

К `interface_name` автоматически добавляется префикс «`ru.basealt.alterator.`». Имя интерфейса может содержать только `[A-Z][a-z][0-9]_`. Если оно состоит из нескольких секций, то они разделяются точкой. Пустые секции не допускаются. Из имени интерфейса формируется `action_id` (polkit) по умолчанию для методов интерфейса, для чего подчёркивания заменяются на тире.

Опция `thread_limit` не обязательна. Если её явно не задать, то её значение будет равно 10 (максимальное число методов, выполняемых одновременно).

Каждая следующая секция — это метод. Имя секции — имя метода. Имя метода может состоять только из латинских букв, цифр и символа подчёркивания. Пробелы не допускаются. Содержимое секции зависит от модуля, который будет обрабатывать запрос.

В секции `example` приведены возможные опции метода для модуля `executor`. Поле `execute` — обязательное, остальные — нет. Поле `execute` содержит строку для `bash` с исполняемым файлом. Внутри фигурных скобок могут задаваться параметры для метода D-Bus. Параметров может не быть. Имя параметра может содержать только латинские буквы, цифры и символ подчёркивания. В примере выше метод `make_ls2` будет выполнять команду `ls -al` с параметром `param`, в который можно будет передать имя директории или файла.

С помощью полей `stdout_strings`, `stdout_bytes` и `stderr_strings` можно включить возможность возврата массива строк или массива байт из `stdout` и массива строк из `stderr`. Для этого полю нужно задать значение `enabled`. Если включён возврат массива байт, то возврат строк из `stdout` отключается (в том числе сигналами). Максимальный размер в байтах для соответствующего массива можно задать с помощью полей `stdout_byte_limit`, `stdout_strings_limit` и `stderr_strings_limit`. Значение по умолчанию — 524288. Допустимый диапазон — от 0 до 2147483647. Вывод из `stdout` и `stderr` можно также получать с помощью сигналов. Имя

сигнала получается путём конкатенации sender'а и значения соответствующего поля (stdout_signal_name или stderr_signal_name). В sender'е «:» и «.» заменяются на «_».

Значения полей stdout_signal_name и stderr_signal_name могут содержать только латинские буквы, цифры и символ подчёркивания. Сигналы для stdout и stderr включены, если соответствующему полю задано значение. С помощью поля thread_limit можно задать ограничение на количество тредов для метода, т.е. сколько экземпляров этого метода можно запустить параллельно. Значение по умолчанию — 1. В поле action_id можно задать action_id polkit для метода. Поле может содержать только следующие символы [A-Z][a-z][0-9].-. К значению этого поля будет автоматически добавлен префикс сформированный из имени интерфейса.

Пример

Создадим example.backend:

```
[Manager]
module_name = executor
node_name = example
interface_name = example
[hexdump]
execute = hexdump -C {param}
stdout_strings = enabled
stdout_strings_limit = 7000000
thread_limit = 3
```

В этом примере мы получаем вывод (stdout) запускаемой команды массивом строк, но, как уже писалось выше, вывод мы можем получать и в виде массива байт.

Запустим наш сервис и посмотрим на дерево D-Bus (см. Рис. 1).

Объект /ru/basealt/alterator/example

Интерфейс ru.basealt.alterator.example

Метод hexdump возвращает массив строк из stdout.

Выполним метод hexdump (см. Рис. 2).

Таким образом, можно сформировать backend для какого-либо приложения с API на D-Bus.

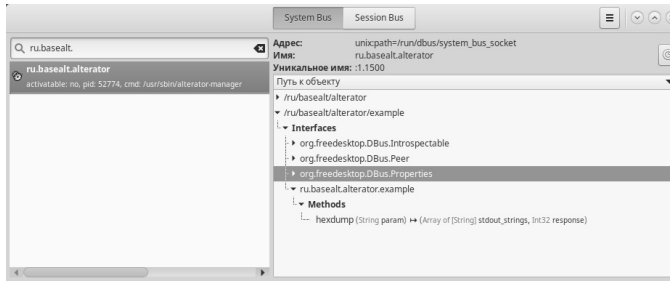

Рис. 1: Дерево D-Bus для ru.basealt.alterator

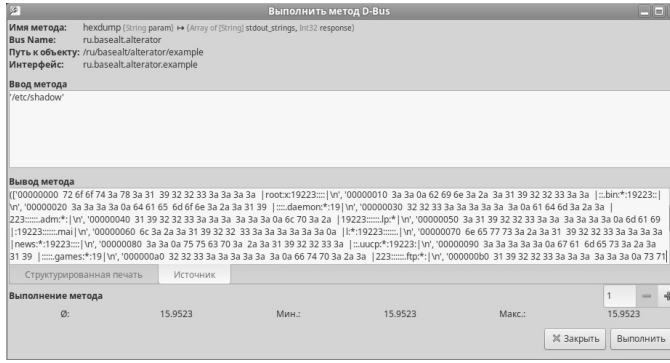

Рис. 2: Выполнение метода hexdump

Алексей Сапрунов
Саратов, ООО «Базальт СПО»

Проект: Alt Diagnostic Tool
https://github.com/AlexSPO/alt-diagnostic-tool/

Утилита диагностики системы Alt Diagnoctic Tool (ADT)

Аннотация

Своевременная диагностика системы — важная составляющая эксплуатации Linux-дистрибутива, необходимая разным специалистам, от разработчика до системного администратора. ADT — инструмент, позволяющий упростить диагностику, самостоятельно разработать тесты для анализа распространённых проблем. ADT представлен в графическом интерфейсе и интерфейсе командной строки.

Что такое Alt Diagnostic Tool (ADT)?

ADT представляет собой инструмент для запуска тестов в терминале или в графическом интерфейсе. В текущей реализации графический интерфейс ADT имеет вид:

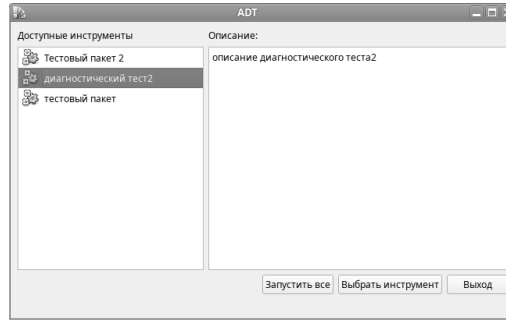

Рис. 1:

Цель создания ADT — упрощение проведения диагностики и анализа работы системы. Работа ADT по поиску доступных тестов, их

запуску и получению результатов работы теста осуществляется через D-Bus с использованием Alterator Manager и его модуля Executor. Благодаря подобной схеме реализации утилита диагностики системы получает доступ к интерфейсу взаимодействия на шине D-Bus. С использованием Alterator Manager и D-Bus возвращается информация о результатах выполнения теста.

Схема работы ADT:

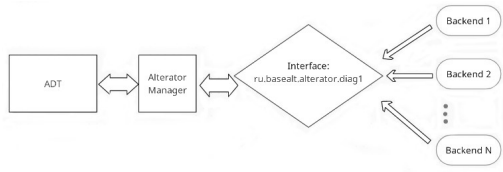

Рис. 2:

Для разработки инструментов мы опирались на интерфейс `ru.basealt.alterator.diag1`. ADT группирует тесты по инструментам, один пакет — один инструмент с произвольным количеством тестов. Это позволяет создавать удобные группы запуска множества тестов при диагностике какой-либо проблемы. Например, инструмент domain-diag содержит в себе 28 тестов, которые дают необходимую информацию о работоспособности клиента в домене. Тесты инструмента domain-diag:

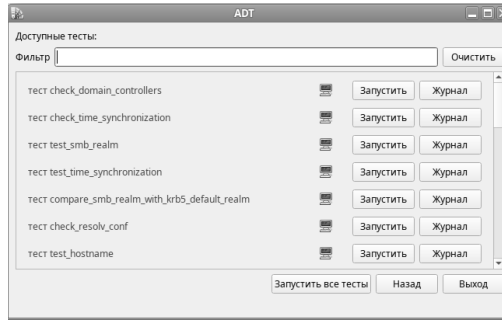

Рис. 3:

Получив информацию, собранную ADT при выполнении тестов инструмента domain-diag, оператор программы получает возможность выдать конкретные инструкции по устранению проблем рабочей станции в домене.

Самостоятельная разработка тестов достаточно проста. В наиболее простом варианте тест может быть bash-скриптом. Такой скрипт реализует интерфейс «ru.basealt.alterator.diag1», «.backend-файл» для alterator manager и «.desktop-файл». backend-файл позволяет запускать тест, .desktop-файл содержит текстовое описание инструмента вместе с включёнными в инструмент тестами. Самостоятельная разработка инструментов позволяет эффективно решать следующие задачи диагностики:

- диагностика в пределах закрытых контуров;
- запуск графического инструмента диагностики неквалифицированным пользователем с последующим предоставлением информации специалисту для решения выявленных проблем;
- увеличение эффективности выявления типовых и часто встречающихся проблем с разным ПО на множестве рабочих станций; ведение журнала тестирования.

Примерный вид журнала работы теста:

Вывод

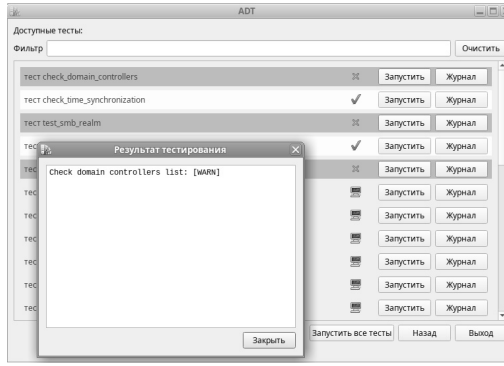

Рис. 4:

Специалисты компании «Базальт СПО» планируют использовать ADT для диагностики проблем функционирования доменов, проблем применения групповых политик, а также для выявления и устранения проблем в закрытых контурах. Гибкий подход разработки тестов позволяет существенно сократить время анализа проблем. Возможность запуска тестирования в графической среде и возможность быстрого создания необходимых тестов позволяет быстрее выявлять и решать «типовые» проблемы. Графическая оболочка запуска тестов снижает требования к квалификации системных администраторов, позволяя быстрее обучать технический персонал.

Литература

[1] Савин Иван — Документация по Alterator manager `https://gitlab.basealt.space/alt/alterator-manager/-/blob/master/docs/README-ru.md`.

[2] Pennington Havoc, Carlsson Anders, Larson Alexander, Herzberg Sven, McVittie Simon, Zeuthen David — D-Bus Specification `https://dbus.freedesktop.org/doc/dbus-specification.html`.

Алексей Бережок
Москва, ИНФЕРИТ
Проект: ОС МСВСфера

Как мы интегрировали GNOME Online Accounts с сервисами Yandex в российской ОС МСВСфера

GNOME Online Accounts, или сокращённо GOA, предоставляет пользователям простой способ входа в онлайн-учётные записи различных сервисов (например Google, Yahoo, Nextcloud и т. д) в среде рабочего стола GNOME.

В окне настройки учётных записей в Gnome видно, что отечественных сервисов в списке не предусмотрено[1]. Но при этом тот же Yandex имеет функционал, который может быть использован в GOA. В российском сегменте подобных интеграций или доработок не обнаружено, поэтому было решено добавить поддержку новых сервисов.

Для взаимодействия с сервисами Yandex было решено использовать Yandex Auth через OAuth токен[2].

Так как процесс получения токена у Yandex в определённом смысле похож на получение токена в Google[3][4], то за основу был взят исходный код goagoogleprovider.c[5], но с заменой путей url, характерных для Yandex.

Идентификатор пользователя извлекается из default_email, возвращаемый при запросе страницы `https://login.yandex.ru/info`.

Список доступа к приложениям Yandex сформирован таким образом:

```
login:email login:info mail:imap_full mail:imap_ro mail:smtp
calendar:all yadisk:disk cloud_api:disk.write
cloud_api:disk.read
```

При добавлении функционала он будет расширяться.

В итоге доработок в списке GOA появился новый GOA-провайдер — Yandex. Иконка Yandex отображается в стандартной теме GNOME. При этом в списке GOA-провайдеров Yandex был вынесен на первое место при отображении.

Так выглядит регистрация:

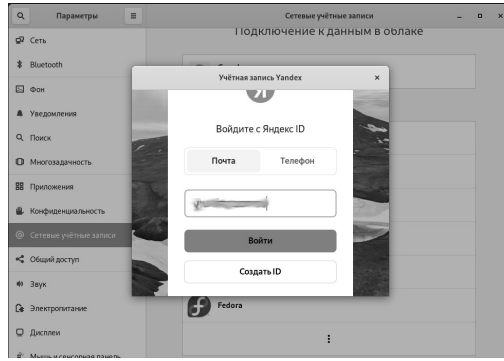

Рис. 1: Окно регистрации аккаунта.

Evolution имеет интеграцию с GOA, поэтому для первичной проверки был выбран именно он. После запуска он показал, что он не

знает нового провайдера. Чтобы он мог с ним работать, был доработан пакет Evolution-data-server. В него добавлен новый провайдер — Yandex. Evolution способен извлекать из GOA токен и параметры учётной записи, и, используя функции провайдера, взаимодействовать с необходимыми сервисами.

Интеграция с почтой оказалась самой простой. Её выполнили по аналогии с Google Mail, после чего она заработала без каких-либо сложностей и нареканий.

Сложности возникли с календарём Yandex, так как у него есть свои особенности. Работа с ним отличается от работы с календарём Google: в протоколе отличий нет, но есть отличие в реализации аутентификации. Google-провайдер передаёт заголовок

```
Authorization: Bearer <authorization-parameters>
```

согласно спецификации[6], в то время как Яндекс-сервер ожидает `OAuth` вместо `Bearer`. Небольшая замена в исходных кодах программного пакета позволила также подключить и календарь от Yandex сервисов.

Для подключения Яндекс Диска в наличии имелись два варианта: Yandex REST API[7] и Yandex WebDAV API[8]. Основной пакет, который реализует подключение виртуальных файловых систем в Gnome — это gvfs. В данном пакете уже имеется инструментарий по работе с WebDAV протоколом (реализован для NextCloud). Поэтому в первую очередь заинтересовал именно WebDAV протокол от Yandex. Для определения типа авторизации используется заголовок www-authenticate[6], который от Yandex сервера возвращает строку:

```
WWW-Authenticate: Basic realm="Yandex.Disk"
```

где аутентификация идентифицирует себя как Basic, но в качестве realm указывается Yandex.Disk. Эта маленькая деталь отделяла от работоспособности сервиса в gvfs пакете, т. к. Yandex требовал «OAuth», а не «Basic». После добавления такого нестандартного Basic, который возвращает в заголовке OAuth, функционал также заработал.

Таким образом, малыми усилиями удалось интегрировать Яндекс сервисы с Gnome окружением. Для подключения Yandex сервисов, достаточно пройти одну авторизацию и получить в различных программах, поддерживающих Gnome Online Accounts работоспособные почту, календарь и диск.

Литература

[1] Запрос на добавление Yandex сервисов в Gnome, https://gitlab.gnome.org/GNOME/gnome-online-accounts/-/issues/30

[2] https://yandex.ru/dev/id/doc/ru/concepts/ya-oauth-intro

[3] Using OAuth 2.0 for Web Server Applications, https://developers.google.com/identity/protocols/oauth2/web-server?hl=en#httprest

[4] Yandex, Получение кода подтверждения из URL, https://yandex.ru/dev/id/doc/ru/codes/code-url

[5] Исходный код Gnome Online Accounts, https://github.com/GNOME/gnome-online-accounts/blob/master/src/goabackend/goagoogleprovider.c

[6] Mozilla.org, Описание HTTP заголовка Authorization, https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

[7] Yandex, Описание Яндекс REST API, https://yandex.ru/dev/disk/rest/

[8] Yandex, Описание Яндекс WebDAV API, https://yandex.ru/dev/disk/webdav/

Ярослав Клементьев
Москва, АО «Е-Паблиш», студент 1-го курса МТИ

Система журналирования и хранения онлайн-конференций

Аннотация

В докладе описывается система журналирования, хранения и доступа к онлайн-конференциям (далее Система). На примере популярной видеоплатформы BigBlueButton (BBB) рассматриваются возможности Системы, включая организацию, управление и доступ к видеозаписям конференций. Работу Системы в рамках интеграции с BBB можно описать так: после завершения конференции происходит конвертация в формат MP4; файл MP4 архивируется в формат ZIP, защищается паролем и сохраняется на сервере. Производится описание владельцем конференции. Часть полей описания портируется из конференции автоматически. Предоставляется удобный интерфейс для воспроизведения

и скачивания записей. В результате сохраняется и накапливается история конференций, доступен поиск по ключевым словам и возможность воспроизведения.

Онлайн-конференции приобретают всё большую популярность, параллельно с этим увеличивается их число. В режиме конференций проводятся совещания, вебинары, учебные занятия, онлайн-консультации. В связи с этим возникает потребность в инструменте для поиска интересующих записей, скачивания и воспроизведения. Для решения этой задачи была разработана Система журналирования и хранения онлайн-конференций, которая интегрирована с BigBlueButton, но может быть адаптирована для работы с аналогичными системами.

Система реализована на языке программирования `Python` с использованием фреймворка `Django` и обладает обширным функционалом. После проведения конференции её запись сохраняется на сервере и обрабатывается Системой. Описание конференции, участников, спикеров, тип извлекается из файла событий и добавляется в базу данных. После появления записи в базе данных становится доступным поиск по ключевым словам, скачивание и ряд других функций. Также Система умеет определять, к какому типу источника относится запись.

Пользователи могут редактировать информацию о конференции: название, описание, участники, приватность, дата удаления. Предусмотрены настройки:

- настройка приватности определяет, кто имеет доступ к записи;
- настройка даты удаления определяет, когда запись будет автоматически удалена из системы.

Дополнительно Система предоставляет возможность добавления раздаточного материала, такого как презентации и доклады.

В Системе реализованы несколько ролей:

1. *Администратор* — имеет доступ ко всем проведённым конференциям, может предоставлять доступ к записям пользователей с ролью «Проводящий конференцию» и редактировать всю информацию.
2. *Ведущий конференцию* — имеет доступ к записям, открытым для него, может редактировать только общую информацию и открывать доступ к конференции пользователям с ролью «Зритель» или по ссылке.

3. *Зритель* — имеет доступ к записям, открытым для него, и может только просматривать/скачивать записи и раздаточный материал.

Разработанная система предоставляет широкий функционал и позволяет эффективно работать с записями. Благодаря разделению на роли достигается дифференцированный доступ и управление информацией. В настоящее время дорабатываются новые функции, такие как вызов абонентов, встраивание в другие системы по `API` и другие.

Получить доступ к репозиторию с исходным кодом можно, обратившись в АО «Е-Паблиш».

Литература

[1] Документация Django Framework., [Электронный ресурс]: `https://docs.djangoproject.com/en/4.2/`.

[2] Документация BigBlueButton., [Электронный ресурс]: `https://docs.bigbluebutton.org/development/api/`.

Сергей Мартишин, Марина Храпченко
Москва, Институт системного программирования им. В.П. Иванникова РАН

Разработка СПО приложения для загрузки на облако зашифрованных данных из электронных таблиц формата ODS (OpenDocument Spreadsheet)

Аннотация

Рассматриваются принципы и инструменты создания студентами СПО веб-приложения, которое позволяет шифровать данные электронных таблиц формата ODS и размещать их на облаке. Приведена ссылка на код приложения, что позволяет загружать приложение, при необходимости модифицировать его и использовать на практике.

В процессе обучения по специальностям, связанным с изучением и использованием СПО (свободного программного обеспечения), студенты выполняют собственные проекты, которые также являются СПО. Во многих случаях эти проекты оказываются востребованными и находят практическое применение. Особенно проекты, связанные с защитой информации различных типов данных.

Электронные таблицы позволяют хранить и организовывать вычисления для данных, представленных в табличной форме. Автоматизация вычислений позволяет уменьшить трудоёмкость обработки информации, осуществлять математическое моделирование и наблюдать за влиянием отдельных данных на результат вычислений. Всё это делает электронные таблицы удобным инструментом для решения экономических и статистических задач. Кроме того, программа для работы с электронными таблицами является одним из основных компонентов пакета офисных приложений, например, `LibreOffice` или `OpenOffice`.

На практике часто возникает ситуация, когда необходимо организовать совместную работу с такого рода электронными таблицами, занимающими существенный объём. Как правило, для хранения и вычислений над такого рода данными используются облачные сервисы. Однако часто в электронных таблицах может содержаться конфиденциальная информация. Поэтому при работе с ними следует учитывать, что одним из участников протоколов обеспечения безопасности доступа к облачным хранилищам данных может быть противник.

Одним из наиболее распространённых способов защиты данных, находящихся в общем доступе, является шифрование на стороне клиента. Очевидно, что вычислительных ресурсов для шифрования на стороне клиента достаточно при использовании встроенного в браузер языка `JavaScript`. При этом серверная часть освобождается от значительной вычислительной нагрузки.

В качестве алгоритма шифрования используется **AES** (Advanced Encryption Standard или Rijndael) — симметричный алгоритм блочного шифрования. То есть AES использует один и тот же ключ для шифрования и расшифрования данных (размер блока 128 бит, ключ 128/192/256 бит). Достоинством алгоритма является то, что один исходный текст при шифровании преобразуется в разные шифротексты, что затрудняет работу противника.

Проект, реализованный студентами в рамках изучения безопасного хранения информации на облаке, заключался в шифровании содержимого электронных таблиц (одного или нескольких файлов) и передачи их на облако.

Для реализации проекта студенты использовали СПО. В качестве инструмента построения электронных таблиц был использован Calc `LibreOffice`, для создания приложения использован язык `JavaScript`, для создания фронтэнд (front-end) частей сайтов и веб-приложений использован Bootstrap и nmp пакеты.

`JavaScript` является реализацией спецификации ECMAScript (стандарт ECMA-262) [1].

- `JavaScript` — язык свободной формы, форматирование не требуется;
- `JavaScript` — язык без строгого контроля типов, то есть типы данных переменных объявлять явно не нужно. Кроме того, во многих случаях `JavaScript` выполняет преобразования автоматически, когда они необходимы. Например, при сложении строки и числа, число будет преобразовано в строку;
- `JavaScript` — интерпретируемый язык, то есть обрабатывается отдельной программой — интерпретатором;
- `JavaScript` — не зависит от аппаратного обеспечения, то есть может использоваться на любой платформе.

Для создания сценариев `JavaScript` необходимы две программы: во-первых, простой текстовый редактор, например, блокнот, в котором

набирается код. Во-вторых, браузер, при помощи которого можно оценить результат выполнения сценария.

Bootstrap [2] включает в себя различные шаблоны для создания форм, кнопок, меню и иных компонентов web-интерфейса. Большим удобством при использовании фреймворков является возможность создавать кроссбраузерные приложения, включая приложения для экранов с различным разрешением (от самых маленьких экранов смартфонов до полноразмерных экранов).

Установка пакетов, необходимых для работы, производится при помощи менеджера пакетов `npm` (Node Package Manager) [3]. Реестр npm представляет собой большую общедоступную базу данных программного обеспечения `JavaScript`.

Был установлен пакет `SheetJS Community Edition` [4] для извлечения данных из электронной таблицы и создания новых электронных таблиц. Команда для установки:

```
npm i xlsx
```

Также был установлен пакет из библиотеки `CryptoJS` [5]. Данная библиотека `JavaScript` содержит реализации стандартных и безопасных криптографических алгоритмов. Команда для установки:

```
npm i crypto-js
```

Диспетчер пакетов производит установку полностью, то есть установит также все необходимые модули.

В корневом каталоге разрабатываемого приложения при первой инсталляции любого пакета создаётся файл под названием package.json, в котором будут находиться метаданные, описывающие приложение и зависимости пакетов, необходимые для запуска приложений. Также будет создана директория `node_modules`, содержащая инсталированные и необходимые для работы с инсталированными пакетами другие пакеты.

Таким образом, в процессе выполнения данного проекта студенты изучают возможности шифрования информации электронных таблиц, получают навыки работы с широко распространённым СПО (`Calc LibreOffice`, `JavaScript`, `Bootstrap`, `npm`). К достоинствам данного способа создания проекта относится простота реализации. Студентам не требуется углублённого изучения Node.js и создание файла app.js для Node.js,

В результате выполнения проекта студентами было создано веб-приложение, которое загружает локально в браузер один или несколько файлов формата ODS, выполняет локально шифрование и далее зашифрованные данные могут быть отправлены на облако. Приложение является СПО и может быть использовано на практике.

Код проекта доступен по адресу `https://github.com/otd13isp/Pereslavl2023_2`.

Литература

[1] ECMA [Электронный ресурс]: — URL: `https://www.ecma-international.org/publications-and-standards/standards/ecma-262` — Яз. англ. — Дата обращения: 15.08.2023.

[2] Bootstrap [Электронный ресурс]: — URL: `http://getbootstrap.com` – Яз. англ. — Дата обращения: 15.08.2023.

[3] Npm [Электронный ресурс]. — URL: `https://www.npmjs.com` — Яз. англ. — Дата обращения: 15.08.2023,

[4] SheetJS Community Edition [Электронный ресурс]: — URL: `https://docs.sheetjs.com/docs` — Яз. англ. — Дата обращения: 15.08.2023.

[5] CryptoJS [Электронный ресурс]: — URL: `https://cryptojs.gitbook.io/docs` — Яз. англ. — Дата обращения: 15.08.2023.

Научное издание

ДЕВЯТНАДЦАТАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ СВОБОДНЫХ ПРОГРАММ

Сборник материалов конференции

Переславль-Залесский,
29 сентября – 1 октября 2023 года

Оформление обложки: *А.С.Осмоловская*
Вёрстка: *В.Л.Чёрный*
Редактура: *В.Л.Чёрный*

Отпечатано с готового оригинал-макета
Подписано в печать 21.09.2023 г.
Формат 60x90 1/16. Усл. печ. л. 9.
Тираж 220 экз. Изд. № 138.

Издательство ООО «МАКС Пресс»
Лицензия ИД N 00510 от 01.12.99 г.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В.Ломоносова,
2-й учебный корпус, 527 к
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3893

Отпечатано в полном соответствии с качеством
предоставленных материалов в ООО «Фотоэксперт»
109316, г. Москва, Волгоградский проспект, д. 42,
корп. 5, эт. 1, пом. I, ком. 6.3-23Н

Информационные партнёры

COMNEWS

TADVISER
Государство. Бизнес. ИТ

Научная Россия

3DNEWS
Daily Digital Digest

servernews
Все самое свежее из мира больших мощностей

CYBER MEDIA

RSpectr.com

AM
ANTI-MALWARE

InformationSecurity
ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

iXBT.com

itWeek

ТЕЛЕСПУТНИК
TGROUP

MSKIT.ru

ITSZ.ru

NNIT.ru

ICTONLINE

ICT2GO

SPBIT.RU

ОТКРЫТЫЕ СИСТЕМЫ
Open Systems Publications

ГЛОБАЛМЕДИА
СЕТЬ РЕГИОНАЛЬНЫХ БИЗНЕС ПОРТАЛОВ

АТР
Мир IT с Антоном Павленко

АРПП
Отечественный софт

РУС СОФТ

АССОЦИАЦИЯ ДОКУМЕНТАЛЬНОЙ ЭЛЕКТРОСВЯЗИ

4CIO

elibrary.ru

basealt.ru

basealt.ru/
19dev-conf