Открытые технологии

Материалы двенадцатой международной конференции Linux Vacation / Eastern Europe 2016

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов двенадцатой
международной конференции
разработчиков и пользователей свободного
программного обеспечения
Linux Vacation / Eastern Europe 2016

(г. Гродно, 25–28 августа 2016 г.)

Брест
«Альтернатива»
2016

УДК 004.45(082)
ББК 32.973.26-018.2я43
О-83

Под общей редакцией **Д.А. Костюка**
Редакционная коллегия: **Д.А. Костюк, Н.Н. Маржан, Д.А. Пынькин, А.О. Шадура**

Рецензенты: кандидат технических наук, доцент, заведующий кафедрой электронных вычислительных машин Белорусского государственного университета информатики и радиоэлектроники **Д.И. Самаль;**
кандидат технических наук, доцент, заведующий кафедрой электронных вычислительных машин и систем Брестского государственного технического университета **С.С. Дереченник**

О-83 **Открытые технологии** : Сборник материалов двенадцатой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2016, Гродно, 25-28 августа 2016 г. / под общей ред. Д.А. Костюка. – Брест : Альтернатива. – 204 с.

ISBN 978-985-521-554-8.

В сборник вошли материалы двенадцатой международной конференциираразработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2016. Материалы докладов распространяются под лицензией Creative Commons Attribution-ShareAlike 3.0. Статьи посвящены новым технологиям и разработкам в сфере свободного (открытого) программного обеспечения. Тематические направления включают разработку, администрирование программных систем и создание аппаратного обеспечения под свободными лицензиями, а также правовые и организационные особенности их использования.

Сборник может быть интересен специалистам в области информационных технологий, занимающимся разработкой и сопровождением свободного программного обеспечения, а также подготовкой кадров высшей квалификации по профильным специальностям.

УДК 004.45(082)
ББК 32.973.26-018.2я43

ISBN 978-985-521-554-8

Содержание

Введение в ЯП Rust. Ключевые принципы и инженерные идеи*

Vitaly Shukela, Minsk, Belarus

Core ideas, features, engineering ideas, pros and cons of Mozilla's Rust programming language.

Введение

Rust — язык общего назначения для системного программирования, конкурент C++. Создан в том числе для того, чтобы снизить необходимый уровень квалификации для написания *правильного*, *надёжного* системного кода по сравнению с C++. В C++ легко написать работающий, но полагающийся на неопределённое поведение код. Rust предлагает разобраться со сложностями *безопасного* управления памятью и многопоточности *до* того, как программа начнёт работать.

Разработчики понимают, что язык сложный и с высоким порогом вхождения. Поэтому значительное внимание уделяется документации, сообщениям об ошибках и непосредственной помощи новичкам через Интернет.

ЯП Rust разрабатывается сообществом под началом Mozilla. Несмотря на то, что у языка нет одного центрального главного автора, ощущение эклектичности и «design by commitee» при знакомстве с языком меньше, что можно ожидать (хотя и присутствует).

Ключевые «X без Y» принципа ЯП Rust:

- безопасность по отношению к памяти без сборки мусора;
- поддержка многопоточности без состояний гонки (race condition);
- абстракция без накладных расходов;
- стабильность языка без стагнации.

Каждый из этих принципов к сожалению имеет и негативную сторону, усложняющую язык.

*vi0oss@gmail.com, http://lvee.org/ru/abstracts/189

Rust черпает идеи из многих других ЯП. Можно сказать, что утверждение «не придумывать своё, сделать правильно уже придуманное» взято как один из принципов разработки языка. Неполный список языков-«доноров»: Ocaml, C++, Haskell, Erlang, Swift, Scheme, C#.

Rust — императивный язык. Функциональное программирование на нём не очень популярно. Есть макросы и плагины к компилятору.

Несмотря на все достоинства ЯП Rust, перед его использованием в реальных проектах следует обратить внимание на недостатки:

- Сложность уровня C++. Высокий порог вхождения. «Была проблема, решил использовать Rust. Теперь у меня `&'a mut` Проблема`<'a, T>`, которую я не могу переместить из заимствованного контекста». Даже после некоторого знакомства с языком, следует ожидать двукратно более медленного программирование по сравнению с, например, C++.
- Молодой язык:
  - неполная поддержка IDE;
  - не все библиотеки написаны;
  - медленная компиляция, нереализованные оптимизации;
  - отсутствуют некоторые возможности языка.
- ABI нестабилизировано и несовместимо между версиями языка (как в C++, но не как в C).

Управление памятью в Rust

В Rust есть три режима доступа к объекту:

- Владение: `x`
- Доступ на запить: `&mut x`
- Доступ на чтение: `&x`

Эта тройственность бывает заметна в разных местах языка и стандартной библиотеки.

У каждого режима есть свои особенности:

- Владелец отвечает за освобождение памяти и выход деструктора. Может «раздавать» ссылки `\&` и `\&mut`.
- Доступ на запить означает, что объект можно изменять, а не только читать. Но после сеанса редактирования объект должен остаться на месте, и ссылку (заимствование) нужно «вернуть на место» владельцу.

– Доступ только на чтение. Это единственный режим с разделяемым доступом на уровне языка. В двух предыдущих режимах доступ монопольный.

Естественно, есть ещё специальный «небезопасный» режим с настоящими указателями в стиле C, без «приставленного к ним маленького милиционера, который следит за доступом». В этом спецрежиме («Unsafe Rust») реализовывается связь с библиотеками на других ЯП (в частности, на C) и структуры данных. Это позволяет реализовывать умные указатели со своими режимами доступа в библиотеках.

Вне этого спецрежима действуют гарантии языка по надёжности.

Следует обратить внимание на список ситуаций, которые *не* входят в эти гарантии:

– утечки памяти, невызовы деструкторов, нарушение RAII, например, из-за циклов в указателях с подсчётом ссылок;
– взаимоблокировки нитей;
– целочисленные переполнения (когда контроль переполнений отключен);
– переполнение стека (аварийное завершение программы, но без неопределённого поведения);
– вмешательсово в работу программы со стороны (отладчиком и т.д.);
– игнорирование некоторых труднообрабатываемых с RAII ошибок (системных вызов close).

Пример срабатывания контроля заимствований:

```
fn eat_box(boxed_int: Box<i32>) {
   println!("Объект, содержащий внутри {}\
             , освобождается из памяти", boxed_int);
}
fn peep_inside_box(borrowed_int: &i32) {
   println!("Заглянули в объект, внутри {}", borrowed_int);
}
fn main() {
   let boxed_int = Box::new(5);
   peep_inside_box(&boxed_int);
   peep_inside_box(&boxed_int);
   {   let _ref_to_int: &i32 = &boxed_int;
       eat_box(boxed_int); /* не компилируется */
```

```
    } // reference goes out of scope;
    eat_box(boxed_int);
}
```

У каждого заимствования (ссылки на объект) есть время жизни. Эти времена жизни, о которых иногда идёт речь и при описании других ЯП, выражены в Rust явно и входят в синтаксис языка (lifetimes). На этапе компиляции они проверяются. Функции, могущие оперировать со ссылками с разными временами жизни считаются обобщёнными (generic) и имеют специальный дополнительный параметр.

Пример:

```
fn choose<'a,'b>(j:&'a i32, k:&'b i32) -> &'a i32 { j }
```

Расшифровка примера приведена в таблице 1.

Прослеживая lifetime, компилятор Rust может рассуждать, действительно ли соблюдаются принципы работы с памятью:

- нет доступа на запить из нескольких мест к одному и тому же;
- нет чтения неинициализированной памяти;
- нет доступа к объекту, если он уже освобождён.

Это всё проверяется из типов данных и сигнатур функций. В частности, в приведённом выше примере невозможно было бы определить, какое время жизни у возвращаемой функцией choose ссылки, без «подглядывания» в реализацию «{ j }» (которая может быть в общем случае далеко от объявления).

Когда речь идёт о безопасности памяти и ссылках, Rust предпочитает быть скорее сложным, чем нестрогим. Можно частично избежать сложностей работы со ссылками в Rust путём использования доступных в стандартной библиотеке умных указателей `Rc`, `Arc` и `Cow`.

Так же как типы наследуются друг от друга в других ЯП, lifetime «наследуются» в Rust.

```
'a : 'b
```

Это означает, что `'a` шире `'b` (начинается не позднее начала, заканчивается не ранее конца `'b`). Значит, где требуется ссылка `&'b`, можно использовать и `&'a` (но не наоборот). Аналогично, из ссылки `&'a` можно сделать ссылку `&'b` (но не наоборот).

Таблица 1.

`fn`	Определяем функцию
`choose`	«choose»
`<`	с двумя generic-памаметрами:
`'a,`	время жизни `'a` и
`'b>`	время жизни `'b`;
`(`	с двумя аргументами:
`j:`	`j` —
`&`	ссылка,
`'a`	имеющая время жизни `'a`,
(пустота)	только для чтения
`i32,`	на 32-разнядное число со знаком;
`k:`	`k` —
`&'b i32`	ссылка только чтение на i32 с временем жизни `'b`,
`->`	возвращающая
`&'a i32`	ссылку только чтение на i32 с временем жизни `a`,
`{ j }`	а именно, свой первый аргумент.

```
зона действия 'a {
     ...
     зона действия 'b {
         ...
     }
     ...
 }
```

Интерфейсы (Traits), типы и generics.

Два пользовательских составных объекта в Rust — это структуры и перечисления. Они приблизительно соответствуют конструкциям C struct и union (с тегом).

Также можно задавать набор сигнатур функций (интерфейс, trait) который можно «привязывать» к типу данных.

Для trait'ов есть наследование, для обычных типов данных его нет.

И trait'ы, и типы данных могут быть generic, то есть определять семейство интерфейсов или типов в зависимости от набора типов-пареметров.

Реализации некоторых интерфейсов может предоставить сам компилятор:

```
#[derive(Debug, Eq, PartialEq, Ord, PartialOrd)]
 struct SomeEntry {
     pub q : String,
     w : i32,
 };

 #[derive(Copy)]
 enum Q {
     Variant1,
     Variant2(usize),
 }
```

В отличие от C++, реализации generic-функций проверяются на правильность до инстанцирования конкретными типами.

Пример использования интерфейса:

```
trait Qqq {
     fn a(&self) -> i32;
```

```
}

struct Www {
    g: isize;
}

impl Qqq for Www {
    fn a(&self) -> i32 { self.g }
}
```

Помимо generic-параметров «на входе», интерфейсы могут также давать типы «на выход»

```
trait MyTrait<T> {
    type Output;
    fn qqq(&self) -> Self::Output;
}

struct Lol;
struct LolOut;
impl MyTrait<u8> for Lol {
    type Output = LolOut;
    fn qqq(&self) -> LolOut { LolOut }
}
```

Библиотеки могут оставлять интерфейсы для реализации пользователям. При этом есть специальное правило: нельзя реализовывать (`impl`) чужой (из другого компонента) интерфейс для чужого типа. При помощи этого правила обеспечивается сочетаемость компонентов — каждая реализация «привязана» к компоненту типом и/или интерфейсом.

Прочие возможности ЯП Rust

Ниже приведен краткий перечень других возможностей языка с примерами их использования.

Rust поддерживает макросы.

```
macro_rules! o_O {
    (  $(
            $x:expr; [ $( $y:expr ),* ]
```

```
            );*   ) => {
            &[ $($( $x + $y ),*),* ]
        }
    }

    fn main() {
        let a: &[i32] =
            o_O!(10; [1, 2, 3];
                 20; [4, 5, 6]);

        assert_eq!(a, [11, 12, 13, 24, 25, 26]);
    }
```

В Rust широко используются итераторы:

```
let a = [1, 4, 2, 3, 8, 9, 6];
let sum: i32 = a.iter()
                .map(|x| *x)
                .inspect(|&x| println!("filtering {}", x))
                .filter(|&x| x % 2 == 0)
                .inspect(|&x| println!("{} seen", x))
                .fold(0, |sum, i| sum + i);
println!("{}", sum);
```

Тесты можно включать прямо в документацию к библиотеке:

```
   /// Clears the map, removing all values.
    ///
    /// # Examples
    ///
    /// <<‘
    /// use std::collections::BTreeMap;
    ///
    /// let mut a = BTreeMap::new();
    /// a.insert(1, "a");
    /// a.clear();
    /// assert!(a.is_empty());
    /// <<‘
```

Помимо дженериков можно использовать также и type erasure.

Также предусмотрены две системы обработки ошибок: `panic/unwind` и `Result`.

Заключение

Изучение ЯП Rust — хорошая идея для системных программистов независимо от использования или неиспользования языка в реальных проектах. Язык сравнивают с аппаратом Илизарова для небрежных программистов на C/C++ — после Rust «руки выпрямляются» и код получается более качественным в т.ч. за его пределами.

Литература

[1] Публичный протокол чата Rust. `https://botbot.me/mozilla/rust/`

[2] Тред «цитаты о Rust». `https://users.rust-lang.org/t/twir-quote-of-the-week/328/272`

[3] Презентация доклада. `https://vi-server.org/pub/rust.pdf`

strace*

Dmitry Levin, Moscow, Russian Federation

strace is a diagnostic, debugging and instructional userspace utility for monitoring interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. This paper gives an overview of strace development, lists the most noteworthy changes made in recent releases and the new features currently being worked on.

Немного цифр

На LVEE 2013 три года назад был представлен обзор проекта strace с 1991 года до середины 2013 года[1]. Нынешний обзор охватывает значительно более короткий период с середины 2013 года по настоящее время, что позволяет рассказать о некоторых значимых событиях более подробно.

За минувшие три года было выпущено пять версий strace. Периодичность выпусков постепенно сокращалась с одного раза в год до одного раза в два-три месяца, одновременно с выпусками ядра Linux.

Интенсивность разработки за эти три года по сравнению с предыдущим трехлетним периодом выросла в три раза, число авторов коммитов выросло в полтора раза, а число авторов 80% коммитов уменьшилось в два раза. Более подробно статистическая информация о коммитах и их авторах представлена в таблице.

*ldv@altlinux.org, http://lvee.org/ru/abstracts/227

Число коммитов и их авторов по версиям:

дата выпуска	номер версии	число коммитов		число авторов			число авторов
		всего	в год	прежних	всего	в год	80% коммитов
13.04.2010	4.5.20						
15.03.2011	4.6	112	122	5	12	13	4
02.05.2012	4.7	400	353	3	11	10	2
03.06.2013	4.8	237	218	4	17	16	3
всего за период		**749**	**238**		**33**	**11**	**2**
15.08.2014	4.9	247	206	4	22	18	3
06.03.2015	4.10	400	719	4	15	27	1
21.12.2015	4.11	586	737	5	15	19	1
31.05.2016	4.12	799	1804	5	16	36	1
26.07.2016	4.13	182	1182	4	6	39	1
всего за период		**2214**	**703**	**8**	**51**	**16**	**1**

Что нового

Из наиболее заметных изменений можно отметить следующие нововведения и улучшения.

Добавлены новые параметры:

-k (экспериментальный): печать стека вызовов функций трассируемых процессов после каждого трассируемого системного вызова;

-w сбор статистики по времени, проведенному трассируемыми процессами в системных вызовах;

-yy печать протокола и адреса, связанных с трассируемыми сокетами.

Реализована надежная трассировка процессов, использующих различные ABI ядра Linux, в частности, процессов x86 и x32 на x86_64.

Значительно расширена поддержка системного вызова ioctl.

Реализована поддержка всех системных вызовов, поддерживаемых ядром Linux 4.7.

Обнаружены и исправлены все известные на данный момент случаи не вполне корректного декодирования системных вызовов.

Система тестирования strace, в которой три года назад было 5 тестов, значительно доработана и на данный момент состоит из 314 тестов, покрывающих более 72% строк исходного кода.

В работе

В рамках GSoC 2016 завершается работа над следующими проектами:

Structured output : поддержка вывода в машиночитаемом виде в формате JSON;

Netlink socket parsers : расширяемый парсер данных, передаваемых через сокеты семейства netlink;

Fault injection : многофункциональный инжектор ошибок в системных вызовах.

Подробнее об этих проектах можно узнать на сайте GSoC 2016 strace projects[2].

Не strace'ом единым

В процессе доработки системы тестирования strace были обнаружены и исправлены ошибки в других свободных проектах, в частности,

Linux kernel :

- неправильная работа модуля unix_diag[3];
- неправильный номер системного вызова fgetxattr на архитектуре sh64[4];
- неправильный код возврата системного вызова personality на архитектуре sparc64[5];
- неправильный номер системного вызова restart_syscall для x32 ABI на архитектуре x86_64[6];
- неправильная реализация системных вызовов preadv2 и pwritev2 для x86 ABI на архитектуре x86_64[7];
- kernel panic на архитектуре parisc[8];
- kernel panic на ядрах RHEL5.

GNU libc : некорректная обработка кода возврата системного вызова personality[9];

musl libc : некорректное поведение модификатора формата %o в семействе функций printf[10].

Литература

[1] LVEE 2013. strace from upstream point of view
`https://lvee.org/ru/abstracts/94`

[2] GSoC 2016 strace projects
`https://summerofcode.withgoogle.com/organizations/5106770607341568/#projects`

[3] unix_diag: fix incorrect sign extension in unix_lookup_by_ino
`https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=b5f0549231ffb025337be5a625b0ff9f52b016f0`

[4] sh64: fix __NR_fgetxattr
`https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=2d33fa1059da4c8e816627a688d950b613ec0474`

[5] sparc64: fix incorrect sign extension in sys_sparc64_personality
`https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=525fd5a94e1be0776fa652df5c687697db508c91`

[6] x86/signal: Fix restart_syscall number for x32 tasks
`https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=22eab1108781eff09961ae7001704f7bd8fb1dce`

[7] x86: Use compat version for preadv2 and pwritev2
https://git.kernel.org/cgit/linux/kernel/
git/torvalds/linux.git/commit/?id=
9a7a076e8e4ffcfec05e3cafe4c4e31d41ddbaa0
[8] parisc: fix a bug when syscall number of tracee is __NR_Linux_syscalls
https://git.kernel.org/cgit/linux/kernel/
git/torvalds/linux.git/commit/?id=
9a7a076e8e4ffcfec05e3cafe4c4e31d41ddbaa0
[9] Fix linux personality syscall wrapper
https://sourceware.org/git/?p=glibc.git;a=commitdiff;
h=e0043e17dfc52fe1702746543127cb4a87232bcd
[10] fix printf regression with alt-form octal, default precision
http://www.openwall.com/lists/musl/2016/08/04/1

codename: Taurus*

Taras Svirinovski, Minsk, Belarus

Taurus is automation-friendly framework for Load Testing.

В статье рассматривается инструмент для простого доступа к различным средствам автоматизированного нагрузочного тестирования WEB-ресурсов.

Введение в нагрузочное тестирование

Нередко можно встретить снисходительное отношение к тестированию со стороны программистов, поэтому мы позволим себе объяснить важность предметной области и заодно провести маленький ликбез.

Вообразите себя владельцем интернет-магазина. Через месяц грядут большие распродажи и вы не знаете — ожидает вас рост объёмов или падение сервера и язвительные шутки конкурентов (впрочем, если вы известный монополист авиаперевозок — дальше можете не читать). Вот если бы эти пользователи набежали прямо сейчас, то, оценив нагрузку, вы бы добавили нужных ресурсов. Но каких именно? Памяти или процессоров? Или дискового кэша? А может всё и так хорошо? Определенность в данной ситуации — большое конкурентное преимущество для бизнеса и он ставит задачу: создать виртуальную нагрузку. Пусть два бота кладут товары в корзину, один пытается оплатить ещё один отменяет заказ и три в это время регистрируются, итого — семь. Или семь миллионов — как пожелает тот, кто создает нагрузку.

Здесь пора обозначить два понятия: *генератор нагрузки* (приложение, возможно много копий одного приложения в облаке, эмулирующие пользователей) и *тест-план* — сценарий поведения бота.

Существующие инструменты

Люди, имеющие отношение к обсуждаемой области, обязательно слышали про JMeter, Gatling, Selenium, более опытные могут

*grey.fenrir@gmail.com, http://lvee.org/ru/abstracts/190

упомянуть ещё с полдюжины средств генерации нагрузки. Итак, первая проблема — этих средств просто много и они совершенно разные. ctoНекоторые, как JMeter, для описания тест-плана требуют использования графического интерфейса, другие, как Gatling, опираются внутри на классические языки программирования (Scala, Java, Python). У них отличаются умолчания, способности распределенного выполнения и даже наличие такого базового функционала, как возможность поддерживать нагрузку определенное время. Эти инструменты имеют разную способность к автоматизации — некоторые, о ужас, возвращают успешный код завершения когда тест провалился. Люди, отвечающие за работу CI, понимают, что проверять систему подобными инструментами становится крайне неудобно. Впрочем, есть одна область, в которой традиционные инструменты нагрузочного тестирования достаточно близки: результаты их работы сложно увидеть, а увидев — почти невозможно понять.

Наш подход

Я прошу прощения если вы эксперт в данной области и описанные сложности кажутся вам надуманными. Вы, не просыпаясь, в уме разбираете километровые csv-файлы с timestamp'ами в UTC и ваш мощный скрипт рисует рисует графики, от которых в восхищении весь совет директоров. У меня для вас плохие новости: мы решили снизить планку входа в нагрузочное тестирование. Это не обязательно означает помочь вчерашним студентам начать карьеру в IT — нередко в нагрузочное тестирование приходят и опытные разработчики, с одной стороны, предпочитающие аскетичный внешний вид и ясные конфигурационные файлы а с другой, — активно использующие автоматизацию и нуждающиеся в свежих наработках индустрии. Мы в BlazeMeter проанализировали существующие инструменты и решили, что знаем, как упростить жизнь и новичкам, и экспертам.

Чтобы попробовать Taurus, нам понадобится:

1. установить инструмент (проще всего — из репозитория пакетов Python утилитой pip, но есть и нативные установщики для различных OS и даже образ для Docker'a)
2. создать конфигурационный файл для описания тест-плана:

```
execution:
- concurrency: 100
```

```
    ramp-up: 1m
    hold-for: 5m
    scenario: quick-test
scenarios:
  quick-test:
    requests:
```

1. запустить инструмент, передав ему параметром конфигурационный файл `$ bzt <test_file>`

Открывающийся после этого при настройках по умолчанию псевдографический мониторинг не только вызывает вау-эффект у настоящих гиков, но и неплохо позволяет анализировать тест в динамике:

Как это готовить правильно?

Конфигурационный файл может быть написан на YAML либо JSON. На вышеприведенном примере рассмотрим синтаксис первого из них. Файл должен начинаться с трех дефисов, отступы задают иерархию, основные строительные элементы — списки и словари, один элемент структуры на строку. Этих знаний достаточно, чтобы понимать и модифицировать образцы (например, с сайта gettaurus.org).

Приложение консольное, вопросов не задает и сходу встраивается в любую автоматизацию. На это хотелось бы обратить особое внимание: не просто совместимость, но удобство применения, скажем, для Continuous Integration — едва ли не основной принцип проекта. Добавлением нескольких строчек возможно:

– запустить одновременно несколько генераторов нагрузки;
– отложить тест на определенное время;
– настроить критерии успешности теста в том числе и на основании полученных http-пакетов;
– менять исполнителя — например, потребление ресурсов JMeter'ом вас не устраивает или нужны возможности Selenium'а;
– использовать разные типы http-запросов, добавлять заголовки и т.д.;
– получить web-отчет, ссылку на который можно отправить коллеге;
– перенести тест в облако и выполнить на десятках виртуальных машин (осторожно, это может быть платно, о чем я расскажу ниже).

Сейчас мы поддерживаем девять генераторов нагрузки — JMeter, Selenium, Gatling, Grinder, Locust, PBench, Siege, Apache Benchmark и Tsung.

Taurus преобразовывает конфигурационные параметры в опции, понятные конкретному генератору. Для JMeter генерируется XML (JMX), для Gatling — код на Scala (разумеется, выполнение уже готовых нативных тестовых скриптов также возможно). При необходимости Taurus скачивает инструмент из сети, запускает, следит за работой показывая промежуточные результаты (более или менее в реальном времени) и в конце теста выдает финальный отчет.

Отдельно нужно коснуться некоторых принципов выдачи результатов. Данные агрегируются по временным интервалам и выделяются настраиваемыми перцентилями — так, например, инженеру будет интересно, уложились ли 98% запросов в лимит времени. При этом результаты различных инструментов выглядят одинаково и могут сравниваться по основным метрикам.

Ещё пользователь может:

– выполнять мониторинг ресурсов как генерующей нагрузку системы, так и нагружаемой — очевидно, истощение ресурсов любой из них тестировщику следует знать;
– производить быструю настройку переопределяя элементы конфигурации из командной строки;
– структурировать тест-план используя включаемые файлы;
– определять команды операционной системы для выполнения на разных этапах теста.

Для специалистов замечу, что основным поддерживаемым инструментом является JMeter, так уж сложился спрос в индустрии.

Работа изнутри, перспективы

В данный момент разработка идет силами трех человек программистов. Исходный код публично на Github, поддержка пользователей идет посредством Google Groups. Для контроля качества кода изменения оформляются pullrequest'ами, проходящими обязательное review. Выделенных тестировщиков нету, их роль играет армия пользователей, на сообщения которых мы стараемся реагировать **безотлагательно**. Код основательно покрыт юнит-тестами (на данный момент покрытие составляет 92%).

Сборку и деплой пакета и сайта Taurus'а осуществляет Jenkins, выполняя перед этим функциональное и unit-тестирование. Для проверки commit'ов используются также Travis и Appveyor. JIR'ы у нас нету, и github'овский issue tracker отключен по вышеупомянутым соображениям — основная идея поддержки заключается в том, чтобы исправлять выявленные баги немедленно.

Можно уверенно говорить что мы уже стали полезны — в этом году нас почтил вниманием devops.com. Развитие идет достаточно активно, по крайней мере, новый функционал добавляется в планы быстрее, чем реализовывается.

BlazeMeter достаточно ненавязчиво продвигает некоторую интеграцию со своим сервисом — как я уже говорил, web-отчеты на их сайте, одна виртуальная машина бесплатно для облачного исполнения, некоторые сложные конвертации форматов.

Основная задача, которую ставит перед нами BlazeMeter — делать удобный продукт, который привлечет ощутимую часть сообщества, поэтому многие новые направления вырастают из хотелок пользователей. Из подобных глобальных вещей могу отметить скорый приход поддержки функционального тестирования, валидацию конфигурационных файлов, скоростную фотосъемку загрузки web-страницы, поддержку Selenium-тестов на .Net и JS.

Как мы дошли до такой жизни

Компания BlazeMeter, зарегистрированная в США и имеющая команду разработки в Израиле, уже около пяти лет предлагает

средства для тестирования формата «поставь галочки на сайте, а мы за тебя создадим нагрузку и покажем отчёт».

В поисках новых ниш компания обратила внимание на инженеров. Даже не принося деньги напрямую довольные технически грамотные люди создают очень полезный с точки зрения IT-бизнеса фон, недостижимый, например, рекламой или скидками. Этим клиентам не лень читать документацию, но им нужны очень мощные и гибкие инструменты, а ещё лучше — поддержка тех инструментов, которые они уже используют (много ли среди нас желающих переучиваться?). А то, что они используют, оставляет желать лучшего — значит, есть где приложить силы и получить хорошую репутацию.

Языком для проекта был выбран Python, отлично подходящий для работы со сторонними инструментами в командной строке на различных архитектурах, лицензией — Apache 2.0. Архитектура модульная, многие компоненты вполне могут быть переписаны программистом средней руки (поддержка новых генераторов нагрузки или специфический репортер, скажем, отправляющий SMS-сообщения).

Так работает Taurus — швейцарский нож для инструментов нагрузочного тестирования, дающий пользователям удобный интерфейс взаимодействия с ними (стараясь при этом не потерять некоторые изюминки отдельных генераторов).

Написание своей StdLibC*

Дмитрий Храбров, Гомель, Беларусь

This article shows how you can implement basic stdlib's functions. You can see here: string functions, formatted output, usage of *Nix kernel functions etc.

В последнее время на страницах информационных ресурсов можно увидеть статьи «Привет из свободного от libc мира» [1], «Минимальный Elf» [2] и пр. Все они показывают настройку получаемого исполняемого файла, стараются его уменьшить. Например, получен работающий Elf-файл, занимающий всего 45 байт дискового пространства. Однако за данной минимизацией забывается такое важное свойство, как юзабилити. Elf-файл в 45 байт умеет только запускаться и корректно завершаться. При этом если добавить хотя бы вывод классического сообщения «Hello World», то размер станет далеко не таким маленьким и от многих техник придётся отказаться.

Существует множество свободных реализаций стандартной библиотеки Си [3]. Минимальный размер имеет dietlibc, которая позволяет скомпилировать приложение размером всего 200 байт. Или 6 килобайт если используется функция printf. Однако значительная часть функционала там не реализована, или реализована не в соответствии со стандартом. Кроме того, dietlibc лицензирована под GPL2, что ставит ряд значительных ограничений (свои приложения придётся так же лицензировать под GPL2). Ещё одной хорошей альтернативой в плане размера является musl: 1800 байт минимум, 13 килобайт с printf. Библиотека musl лицензирована под MIT и качественно реализует стандарт. Однако именно из-за полного охвата стандарта Си и получаются 13 килобайт, которые хотелось бы уменьшить.

Целью данной работы является создание библиотеки, с помощью которой можно было бы более-менее привычно программировать, но в то же время минимизировать размер получаемого исполняемого файла, вообще не использовать стандартную библиотеку. Данная работа представляет больше исследовательский интерес, чем практическую ценность. И в конечном счёте приближает

*root@dexp.in, http://lvee.org/ru/abstracts/191

к пониманию, как именно работают те или иные функции из стандартной библиотеки языка Си. Предложенные реализации далеко не оптимальны или безопасны, однако просты для понимания.

В итоге необходимо получить корректное функционирование данного кода:

```
printf("Hello␣%s␣LVEE␣%d!\n", "Summer", 2016);
```

В этой простейшей, казалось бы, строке кода, поднимается сразу несколько вопросов: 1) непосредственно сама функция printf и её форматный вывод; 2) функции работы со строками; 3) конвертирование чисел в строку; 4) непосредственно вывод символов на экран. Всё это обычно реализуется средствами стандартной библиотеки, и всё придётся реализовать самостоятельно.

Syscall — вызов функций ядра Linux, для x86 реализуется через 0×80 ассемблерное прерывание. Полный код для функции с 3 аргументами (взят из [4]):

```
static long _syscall3(int number, unsigned long arg1,
    unsigned long arg2, unsigned long arg3){
  long result;
    __asm__ volatile ("int␣$0x80"
    : "=a" (result)
    : "0" (number), "b" (arg1), "c" (arg2), "d" (arg3)
    : "memory", "cc");
  return result;
}
```

Если вызываемой функции необходимо менее 3 аргументов, то завершающие аргументы можно передать нулевыми. Например, функция close принимает всего 1 аргумент — файловый дескриптор. Реализация может выглядеть следующим образом:

```
return (int)syscall3(__NR_close, fd, 0, 0);
```

Константы названий функций (NR_close и пр.) задаются для каждой архитектуры отдельно в файле unistd.h. Это заголовочный файл для доступа к API POSIX-совместимой операционной системы. В unistd.h можно посмотреть полный список функций ядра Linux. И не увидеть там ни строковых функций, ни потоков (thread), ни malloc/calloc/free, ни printf. Весь этот функционал реализуется на функциях более низкого уровня в стандартной библиотеке. Вывод символов на экран консоли осуществляется с помощью функции write:

```
return (ssize_t)syscall3(__NR_write, fd, buf, count);
```

Как видно, функция write использует 3 аргумента: файловый дескриптор, буфер и размер буфера в байтах. Однако бывают ситуации, когда размер буфера заранее неизвестен. Обычно в таких случаях используется функция получения длины строки — strlen. Реализуется она средствами чистого Си, без вызова функций ядра. Это является преимуществом, так как такой код может быть скомпилирован любым компилятором, реализующим стандарт Си. Начиная от свободных GCC, Clang, Tiny C Compiler, и заканчивая проприетарной Visual Studio. Пример реализации поиска длины строки:

```
len=0; while( str[len] != 0 ) len++;
```

Аналогично реализуются и многие другие функции работы со строками — происходит посимвольная обработка всей строки в цикле. В качестве примера обработки одного символа можно рассмотреть функцию перевода символа в верхний регистр. Если символ лежит в интервале от a до z, то от его кода отнимается «a» и прибавляется код «A»:

```
return symb - 'a' + 'A';
```

Конвертирование целых чисел в строку — деление на 10 до тех пор, пока не останется 0. Остаток от деления складывать в буфер. Существует множество способов сделать перевод из числа в строку, но они всё-равно опираются на какой-либо один базовый, который и необходимо реализовать.

Функция printf может принимать разное количество аргументов. Эта «перегрузка» параметров функции осуществляется средствами Си через списки варьирующихся аргументов: va_list, va_start, va_arg, va_end.

Пример реализации:

```
void Myprintf(char* format, ...){
  char* s;  // Буфер для вывода строк
  int i; // Счётчик текущей позиции в форматной строке
  va_list arg; // Список варьирующихся аргументов
  va_start(arg, format); // Инициализируем список
  for(i=0; i<strlen(format); i++){
    if( format[i] == '%' ) switch( format[i+1] ){
      case 'd': // вызов своей функции для чисел
```

```
        MyPutInt( va_arg(arg, int) ); i++; break;
      case 's': // получение строки и её вывод
        s = va_arg(arg, char*);
        MyWrite( s, strlen(s) ); i++; break;
      case 'c': // аргумент - в int
        MyPutchar( va_arg(arg, int) ); i++; break;
    } else MyPutchar( format[i] );
  } // вывод одного символа реализуется через write
  va_end(arg); // Необходимая очистка списка
}
```

Суть так же проста — перебираем символы форматной строки, если встречаем символ '%', то смотрим следующий за ним. По следующему символу узнаём тип аргумента (если символ «d», то нужно конвертировать в int). Далее просто берём следующий аргумент из списка варьируемых, сразу просим привести его к нужному типу. Нужно помнить, что нельзя сразу конвертировать во float, char или short — изначально должно быть преобразование в double или int.

Теперь остаётся только скомпилировать приложение с флагами nodefaultlibs и nostartfiles (подробнее про это можно почитать в статье «Привет из свободного от libc мира» [1]). Если всё собрано правильно, то при вызове ldd будет писать «not a dynamic executable». Это означает, что полученное приложение вообще не зависит ни от каких системных библиотек. Но в то же время оно написано на Си и использует функцию printf, что и требовалось получить.

Литература

[1] Hello from a libc-free world! `https://blogs.oracle.com/ksplice/entry/hello_from_a_libc_free`

[2] A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux. `http://www.muppetlabs.com/~{}breadbox/software/tiny/teensy.html`

[3] Comparison of C/POSIX standard library implementations for Linux. `http://www.etalabs.net/compare_libcs.html`

[4] Program in C without any C library. `https://github.com/fishilico/shared/tree/master/linux/nolibc`

«Параноидальный» офис — оборотная сторона безопасности*

Дмитрий Степанов, Минск, Беларусь

News about data breaches, hacking, discovered vulnerabilities are now typical. Common solutions deal with external threats along with international standards and best practices, not taking into account non-commercial threats of information leaks.

С каждым годом информационные ресурсы и системы развиваются, все глубже проникают в повседневную деятельность компаний и каждого человека в отдельности. Сейчас ни у кого не вызывает удивления новость об утечке данных пользователей того или иного сервиса, взломе системы, обнаружении уязвимости продукта. Многие руководители предприятий полагают, что это их не коснется, так как «это где-то там, а мы здесь». Решения, внедряемые на предприятиях, защищают в большинстве от внешних угроз, взломов, атак, они согласованы с международными стандартами и рекомендациями (cobit, itil и т. д.), и в большинстве своем оценивают риски потери информации с точки зрения убытков для предприятия, но не учитывают, что информация на предприятии может быть не только коммерческой но и чрезмерно конфиденциальной в ином плане.

Весь этот информационный поток порождает у руководителей частных предприятий вполне обоснованную паранойю о сохранности информации по деятельности предприятия.

Исходя из выше описанного мы и рассмотрим распространившуюся паранойю одного руководителя частной компании «X» и реализацию его желаний и потребностей.

На момент созревания «wish list» система компании «X» представляла собой филиальную сеть между двумя странами, граничащими друг с другом. В стране «A» находился отдел продажи и маркетинга (собственно, мы находились в стране «A»), а в стране «Б» — производство, логистика и небольшие продажи.

Инфраструктура была реализована на основе платного ПО (все лицензировано) и представляла следующую картину:

*admin@itg.by, http://lvee.org/ru/abstracts/206

– 80 ПК с установленными операционными системами Windows;
– 3 Сервера: 1. Шлюз на платном ПО; 2. Бухгалтерия на платном ПО; 3. Файловый сервер на Платном ПО.

Структура описана кратко и не учитывает наличие АТС и структуры сети (наличия коммутационного оборудования и Wi-fi точек).

Задача.

Была поставлена задача реализовать полное шифрование каналов связи с внешним миром, шифрование почтовой переписки, отслеживание сотрудников, максимально ограничить возможность утечки информации через сотрудников компании и в случае «случайной пропажи какого-либо сервера или компьютера с жесткими дисками из компании». Важнейшим условием стало уменьшение стоимости лицензирования ПО и то, что внедряемый проект в последующем будет распространен на остальные подразделения компании.

План создания.

Выбор ОС пал на дистрибутивы Linux по следующим причинам:

– низкий уровень владения и грамотности конечных пользователей в среде Linux;
– стоимость дистрибутива Linux;
– возможность реализации шифрования из коробки;
– наличие альтернативного ПО на замену проприетарным пакетам в среде Windows;
– компания занимается продажами — количество узкоспециализированного по незначительного;
– производительность ОС на разношерстном парке ПК.

В результате сервера были реализованы на Ubuntu Linux, а клиентские места на Linux Mint Xfce Edition.

Защита каналов и выбор шлюза.

Реализация защиты каналов была проведена посредством объединения филиалов Ipsec тоннелем, в качестве межсетевого экрана тестиррovались системы Zentyal и ZeroShell. В городе «А» был установлен ZeroShell благодаря его возможности работать с LiveCD. На

второй стороне в городе «Б» был установлен Zentyal — использовался как почтовый сервер и LDAP-сервер. Оба сервера работают под приложением OpenVPN, и таким образом реализация тоннеля не вызвала каких-либо существенных трудностей.

Маршруты были настроены так, что весь офис «А» выходит в интернет через офис «Б». Также был реализован отдельный канал IPSec с SIP-провайдером посредством установки маршрутизатора на стороне провайдера.

В последствии система несколько видоизменялась в связи с улучшением интеграции.

Почтовая переписка была переведена на почтовый клиент Thunderbird с реализованным модулем шифрования на основе PGP. На всех ПК был настроен Iptables для работы только в конкретной сети и с конкретными портами.

Шифрование серверов и офисного ПК.

Полное шифрование диска (Full Disk Encryption, FDE) стало доступнее для простых пользователей Linux. Так начиналась статья (откровенно говоря, не помню какого журнала и тем более автора), ставшая в свое время основой для неоднократной реализации «параноидальных офисов».

По очевидным причинам, мы не хотели стать жертвами термально-ректального криптоанализа со стороны злоумышленников, в результате чего выбрали Luks-шифрование с использованием ключей на основе сертификатов, причем сертификат находился за пределами границ государства офиса «А». Также были предприняты меры по уничтожению сертификата в случае необходимости сохранения информации в безопасности.

Контроль целостности файловой структуры был реализован посредством программного модуля Tripwire.

Таким образом мы получили полностью реализованную на GNU/Linux структуру с шифрованием дисков, на которых оставалась исключительно загрузочная область для получения сертификата и разблокирования диска.

Реализация отслеживания персонала.

Данный вопрос решился на удивление просто: у каждого сотрудника существовал корпоративный телефон на Android или iPhone,

который проходил перед каждой командировкой подготовку и чистку у отдела IT. На данные аппараты были установлены скрытые приложения для программного СПО Traccar. Это позволило убить сразу двух зайцев: контроль транспорта и командированных сотрудников, так как телефоны были заведены через VPN-канал на АТС для осуществления звонков через SIP и бесплатной связи с офисом, отключения производились редко, и в результате трек и местоположение были корректны и отчетливы.

Для реализации удаленной работы и подачи приложений (с почтой и иными приложениями) на случай необходимости был рассмотрен продукт Ulteo, недостатком которого было использование Java, но переход на HTML5 вопросы снял.

Подводя итоги:

Реализация данного проекта позволила в очередной раз пользователям с иной стороны рассмотреть СПО. Экономическая выгода, а так же явное преимущество «коробочного решения» по шифрованию, альтернативное офисное ПО и уровень защищенности платформы (благодаря низкой пользовательской популярности и своей структуре) от внешнего зловредного ПО делают гибридные интеграционные решения все более востребованными в современном мире.

Технология прозрачного сжатия графической памяти GPU*

Сергей Рогачев, Москва, Россия

Complexity of graphical user interfaces of mobile applications is growing rapidly. Much memory is needed to keep buffers for textures and graphical data. Modern mobile GPUs do not have built-in memory and use general memory to keep graphic buffers. A part of RAM is occupied by GPU and cannot be used by an operating system. According to this research most of such buffers are well compressible: 6-9 times. We propose a technology for transparent compression of graphic buffers in Linux kernel.

Каждый год анонсируются новые, все более совершенные гаджеты, использующие ядро Linux и открытые платформы: планшеты, смартфоны, телевизионные приставки, умные часы. Большинство из них оснащается мобильными GPU, используемыми для аппаратного ускорения 3D графики (OpenGL ES) и параллельных вычислений (OpenCL, RenderScript). Наиболее часто встречаются графические процессоры Mali, Adreno и PowerVR. В отличие от GPU, используемых в настольных системах, мобильные GPU не имеют встроенной памяти, а используют часть оперативной (рис. 6.1), предоставляемой ядром ОС — такая память становится недоступной ядру ОС и пользовательским программам. Будем называть память, используемую GPU, графической памятью. Оптимизация расхода такой памяти позволит достичь снижения цены устройств или повысить производительность за счет улучшения кешируемости приложений.

К графической памяти можно отнести GEM буферы, используемые для хранения результата рендеринга, и вспомогательные регионы (native буферы), хранящие цветовые буферы, поверхности, текстуры, различную вспомогательную информацию, относящуюся к элементам графической сцены.

Задача оптимизации использования графической памяти уже рассматривалась ранее [1], в статье упоминалась технология сжатия GEM буферов на стороне пользователя, также были отмечены высокие накладные расходы этого решения.

*rogachevsergei@gmail.com, http://lvee.org/ru/abstracts/216

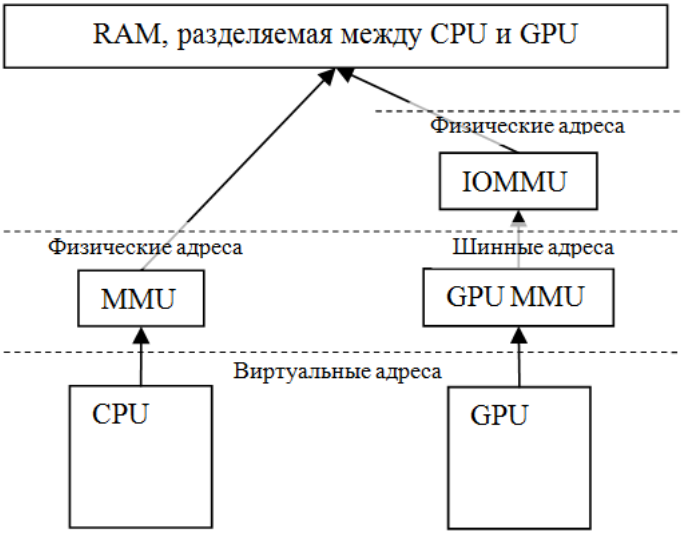


Рис. 6.1. Разделение памяти между CPU и GPU

В настоящей работе рассматривается исключительно вопрос оптимизации использования native буферов, поскольку современные композитные менеджеры уже способны эффективно управлять GEM буферами и дальнейшая работа в этом направлении не требуется.

Авторами было проведено исследование вопроса на примере GPU Mali серий Midgard и Utgard: за счет доработки драйверов ядра удалось реализовать прозрачную компрессию редко используемых native буферов графической памяти. В результате получилось добиться экономии более 100 Мб. оперативной памяти за счет высокого коэффициента сжатия графических буферов (6 — 9 раз), а также наличия множества регионов графической памяти, заполненных нулями.

По своей сути решение реализует механизм схожий с подкачкой страниц (swap), при этом есть ряд особенностей, характерных для графической памяти и мобильных устройств. Ранее подобный подход рассматривался как трудно реализуемый [2].

Для сжатия некоторого региона графической памяти нужна гарантия, что он не будет использоваться CPU или GPU в процессе

компрессии. Это гарантируется путем запрета доступа к страницам данного региона в таблицах страниц GPU, а также снятия отображений региона в адресные пространства процессов на CPU. После того, как соответствующие записи PTE и ATE (записи таблиц страниц CPU и GPU, соответственно) изменены, любое обращение к соответствующим адресам памяти должно вызвать исключение страничной адресации, известное как page fault или data abort.

После того как гарантировано отсутствие доступа к региону памяти, данные со страниц этого региона могут быть сжаты, а страницы освобождены. Для хранения сжатых данных графической памяти в работе применена подсистема GMC (graphical memory compression), созданная в качестве обобщенного слоя для разных драйверов GPU. GMC осуществляет управление буферами для хранения сжатых данных через подсистему ядра zpool (zsmalloc allocator [3]), сжатие данных с использованием crypto comp API, управление идентификаторами сжатых страниц, а также контролирует когерентность кешей. Страницы, данные которых плохо сжимаются, не принимаются для хранения и не освобождаются. Страницы, заполненные нулями, учитываются GMC и освобождаются.

При обращении к адресам страниц графической памяти, освобожденной ранее, генерируются исключения страничной адресации. Соответствующие данные извлекаются из хранилища GMC на вновь выделенные страницы физической памяти. Осуществляется отображение новых страниц по требуемым виртуальным адресам путем изменения записей PTE или ATE в соответствующих таблицах страниц.

Кроме уже описанного механизма важным является вопрос о ранжировании регионов графической памяти по признаку частоты доступа. Сжатие активно используемой памяти может привести к катастрофическому падению производительности за счет частых исключений страничной адресации. Для решения указанной проблемы используется информация о состоянии приложений. Специальный демон (resource daemon) следит за приложениями и информирует ядро о тех из них, которые переходят в фоновый режим (становятся невидимыми на экране). Графическая память таких приложений подлежит сжатию.

Взаимодействие всех компонент во время компрессии отражено на рис. 6.2. Основные шаги пронумерованы.

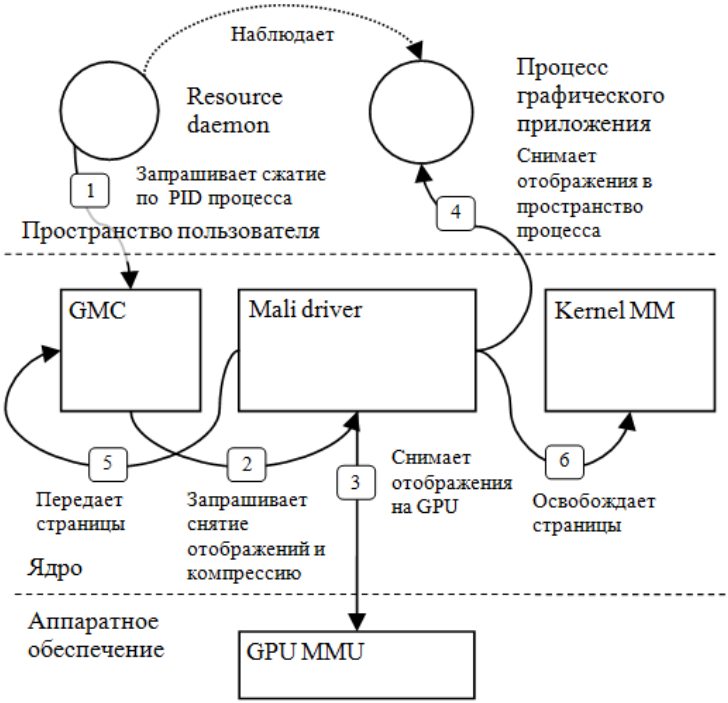


Рис. 6.2. Взаимодействие компонент системы

Рис. 6.3 иллюстрирует снятие отображений, компрессию и освобождение страниц с позиции данных, а не потока управления и дополняет рис. 6.2.

Тестирование решения на платформах Android и Tizen показало сокращение используемой памяти — более 100 мегабайт может быть освобождено при стандартном использовании устройства, когда в фоне оказывается достаточное количество кешированных приложений. Замечено большое количество регионов, заполненных нулями, а максимальная степень сжатия данных графических буферов составила 89%. Накладные расходы, вносимые решением минимальны, а пиковые не превышают 18% (Kindle App, Google Maps). В определенных ситуациях переключение между приложениями

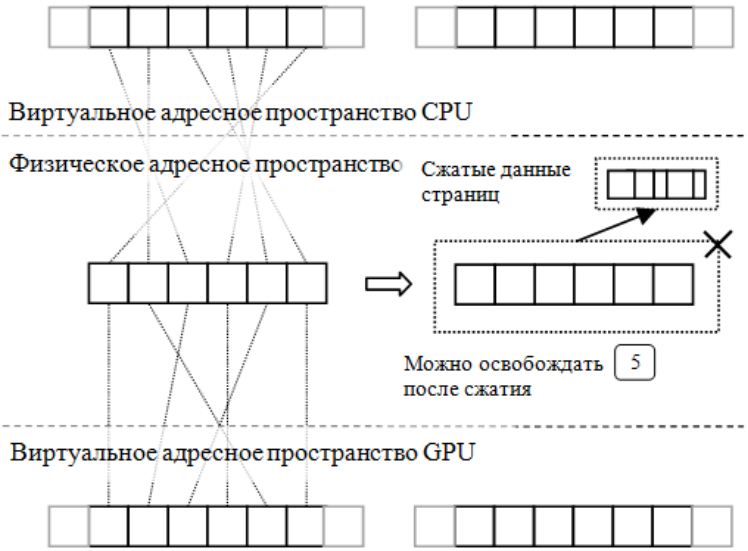


Рис. 6.3. Снятие отображений и компрессия

ускоряется до 40% за счет большего количества свободной памяти, а следовательно, лучшей кешируемости приложений.

Исходный код подсистемы GMC планируется опубликовать в LKML, а изменения в драйверах Mali через группы поддержки ARM.

Литература

[1] Kwon, S., Kim, S.-H., Kim, J.-S., and Jeong, J. Managing gpu buffers for caching more apps in mobile systems. Proceeding EMSOFT '15, 207–216 (2015)

[2] Carmack, J. GPU data paging, `http://media.armadilloaerospace.com/misc/gpuDataPaging.htm` (2010)

[3] Corbet J. The ZsmallocAllocator. Linux Weekly News (2012)

First steps into OpenStack*

Andrei Perapiolkin, Minsk, Belarus

This report aims to give understanding of what OpenStack is, moreover to guide on how to start to use and contribute to the OpenStack. I will describe OpenStack in the context of distributed computing; give short description of OpenStack services Nova, Neutron, Cinder, Swift, Glance, Horisont, Keystone; give short description of devstack, packstack, suseCloud, ubuntu juju. On the example of the devstack, I will give a short description of how you can deploy your own OpenStack cloud.

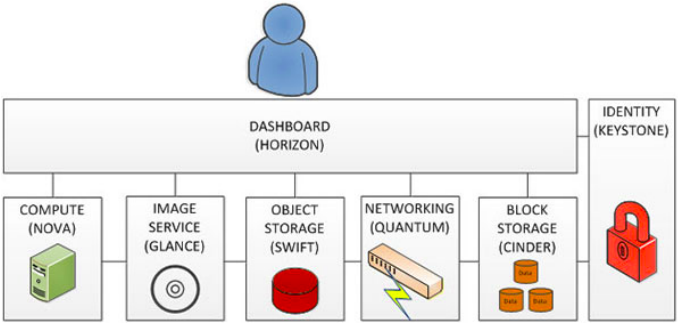
New technologies in the IT world mostly appear on the basis of existing knowledge approaches and infrastructure. It is also true for the cloud computing. We can see that new step of evolution of the grid and cluster systems is Cloud Computing systems. The reason of evolution is high demand for the available computing resources in the society. Rapidly changing modern world requires fast speed deployment of the IT infrastructure. This determines fast deployment of business infrastructure – fast accessibility to the computing resources, storage, ready software deployments, and others. From the perspective of the people who offer all of these resources it is extremely important to have orchestration tools to control and automate all of these tasks. So the merger of all of the requirements leaded to the appearance of the OpenStack.

OpenStack is a merge of some already existing technologies under the hood of Python automation. It includes such components like:

- **Keystone** provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
- **Nova** manages the computational resources by creating virtual computation environments.
- **Neutron** enables network connectivity for the OpenStack services. It allows to define networks, their topology, and attach instances.
- **Cinder** provides persistent block storage to running instances.
- **Glance** stores images for the Nova. Allowing it to make quick start from prepared image deployment.

*perepiolkin@itspartner.net, http://lvee.org/ru/abstracts/217

– **Swift** stores and retrieves arbitrary unstructured data objects via REST API.
– **Horison** service provides web-based interface, and allows to interact with OpenStack services like Cinder, Nova...



There are also such services like **Ceilometer** (provides telemetry), **Trove** (interaction with database), **Manila** (provides shared file system) and some others...

For instance, **Nova** which serves as a computational resource provider by itself is a management that aggregates emulation, visualization and containerization solutions under the unified API. Nova works with KVM, QEMU, Vmware ESXi, Microsoft Hyper-V, XenServer, Docker and others.

Core OpenStack service deployment process is a complex task. In bare OpenStack deployment it's get done by consistent configuration of each servicein in the manner that allows them to interact in between. This task can be simplified by using existing deployment automation tools like devstack and packstack. After installation and configuration admin can validate solution by running Tempest tests.

References

[1] OpenStack `https://www.openstack.org/`
[2] Nova `https://wiki.openstack.org/wiki/Nova`
[3] devstack `http://docs.openstack.org/developer/devstack/`
[4] RDO `https://www.rdoproject.org/`
[5] OpenStack diagram `http://www.ca.com/us/lpg/ca-technology-exchange/sunny-outlook-for-openstack-clouds.aspx`

Разглядывая атомы. Программное обеспечение для визуализации химического строения вещества*

Антон Літвіненка, Kyiv, Ukraine

Brief review of main types and uses of software for chemical structures visualization is presented in this abstract. Current situation with free chemical visualization software as well as some actual tasks and problems in chemical structure analysis are discussed.

Одни из важнейших объектов, которыми оперирует химия, — атомы и молекулы — практически недоступны для прямого экспериментального наблюдения. В то же время, взаимное расположение атомов в веществе несет важнейшую информацию о его строении и возможных свойствах. В настоящее время для анализа строения вещества на атомно-молекулярном уровне активно применяется специализированное программное обеспечение.

Целью работы является обзор существующих типов графического представления атомно-молекулярной структуры вещества, основных задач (решенных и актуальных) такого представления, и ситуации со свободным программным обеспечением для визуализации химических структур.

С помощью ПО для визуализации химических структур решаются следующие задачи:

- Подготовка рисунков для публикаций;
- Анализ параметров структуры (измерение расстояний между атомами, углов между связями, проверка наличия пустот и т.д.);
- Подготовка входных данных для других программ, выполняющих анализ или моделирование (квантовохимическое, молекулярно-механическое, поиск по базам данных и т.д.)
- Анализ результатов вычислений, выполненных другими программами.

*tenebrosus.scriptor@gmail.com, http://lvee.org/ru/abstracts/218

Большинство программ для визуализации предоставляет также широкие возможности для редактирования химических структур или создания их с нуля.

Основными элементами для отображения являются **атом** и **связь**.

2D и «2,5D» визуализаторы (редакторы).

Отображают исключительно связность атомов в химической структуре (как правило, органической молекуле), не отображая расположения атомов в пространстве друг относительно друга, а также длин связей, углов между связями и расстояний между атомами (рис. 8.1). Атомы обозначаются символами соответствующих элементов или вершинами геометрических фигур (вершина без символа обозначает атом углерода); атомы водорода, как правило, опускаются (они могут быть достроены в воображении зрителя, исходя из валентностей атомов, около которых находятся). Связи обозначаются ребрами геометрических фигур (стиль отрисовки линии обозначает тип связи — одинарная, двойная, тройная, координационная и т.д.). Кроме единичных формул, могут изображаться схемы реакций. Изображение плоское (2D) и, как правило, не требует цвета и сложных полутонов — таким образом, рисунки такого типа могут быть без адаптации использованы для печатных работ. В подавляющем большинстве случаев структуры сохраняются в специальных форматах, рисунок может быть экспортирован в виде растрового или векторного графического файла, а также (под Windows) с помощью OLE. Подписи, стрелки и другие элементы схем могут быть добавлены как в самом редакторе, так и при постобработке.

Исторически подобный способ изображения молекул (несколько отличающийся по используемым обозначениям) появился в середине XIX века [2]. Для нужд современной органической химии были добавлены условные обозначения и стандартные проекции, необходимые для указания конфигурации оптических изомеров (рис. 8.2) — таким образом, этот способ изображения молекул является чем-то средним между 2D и 3D.

Дополнительное использование — подготовка запросов для поиска по базам данных (по фрагменту структуры), а также для предсказания ЯМР-спектров.

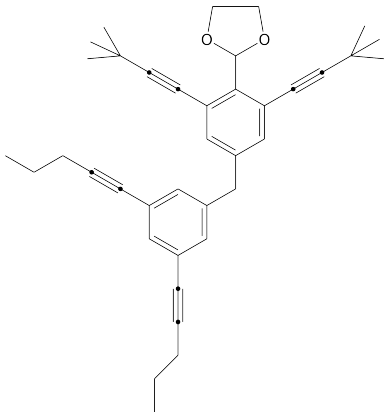


Рис. 8.1. Пример органической молекулы, нарисованный в GChemPaint (перерисовано автором по формуле, приведенной в работе [1]).

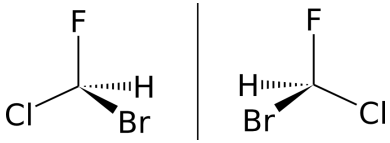


Рис. 8.2. Два оптических изомера бромфторхлорметана, отличающиющиеся только ориентацией заместителей. Связь, обозначенная сплошным треугольником, обозначает атом, находящийся перед плоскостью рисунка, штрихованым — за плоскостью рисунка.

Свободное ПО для 2D-визуализации (GChemPaint, BKChem) существует, однако имеет очень бедный функционал и проигрывает в сравнении с проприетарными аналогами (ACD/ChemSketch, ISIS/Draw). Шаблоны для LaTeX еще беднее по возможностям и используются крайне редко.

3D визуализаторы

Изображают координаты атомов в трехмерном пространстве, а также, опционально, связность.

Молекула может быть представлена как набор атомных ядер, вокруг которых расположены электроны (которые могут быть представлены в виде некоторого распределения электронной плотности, подчиняющегося уравнениям квантовой химии). Скорость движения ядер считается существенно более низкой, чем скорость движения электронов, что позволяет эти движения рассматривать независимо друг от друга (принцип Борна-Оппенгеймера). Таким образом, координаты атомов представляют собой координаты их центров (ядер), причем в зависимости от целей исследования может рассматриваться как некоторые статические координаты, так и анимация движения ядер. Что касается электронной плотности вокруг ядер, то ее полное представление как функции от пространственных координат очень сложно для наглядного построения и малоинформативно, потому применяются (рис. 8.3), в зависимости от задач, упрощенные модели (следует отметить, что понятие связи при таком подходе также является в значительной степени модельным):

– Каркас;
– Стержни;
– Шары и стержни;
– Сферы ван дер Ваальса.

Размеры изображений атомов и связей во всех моделях, кроме размеров сфер ван дер Ваальса, являются условными величинами, и их выбор определяется исключительно удобством представления — чем больше размеры, тем легче восприятие отдельных атомов, но тем больше атомы заслоняют друг друга. Для общего анализа геометрии сложной молекулы, состоящей из сотен атомов, больше подойдет изображение в виде каркаса, для подробного анализа более простых молекул — шаростержневая модель. Особое место занимает отображение атомов в виде сфер с радиусом, равным ван-дер-ваальсовским радиусам этих атомов, которое позволяет оценить пространство, доступное для других молекул.

В отличие от 2D-рисунков, использование меток и подписей в 3D-изображениях молекулярных структур ограничено, потому более интенсивно используются цветовые схемы (например, раскрашивание атомов разных элементов в разные цвета). Это ведет к получению более эффектных изображений, но усложняет подготовку рисунков для черно-белой печати.

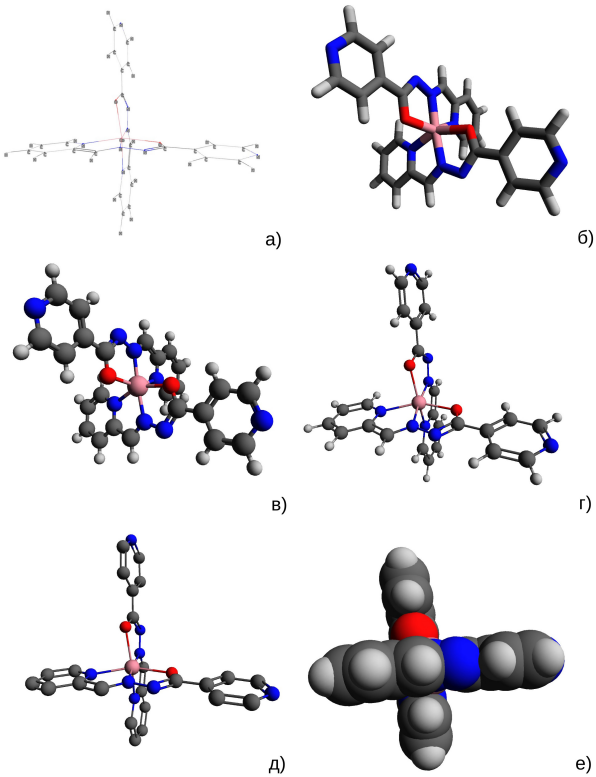


Рис. 8.3. Разные способы отображения 3D геометрии молекулы (оптимизированная путем квантовохимического моделирования геометрия комплексного соединения кобальта(II) с основанием Шиффа из 2-пиридилкарбальдегида и гидразида 4-пиридилкарбоновой кислоты, моделирование описано нами в работе [2]): а) каркас; б) стержни; в) шары и стержни; г) шары и стержни с уменьшенными радиусами; д) шары и стержни с уменьшенными радиусами без атомов водорода; е) сферы ван дер Ваальса. Построено с использованием программы Avogadro.

Примеры свободных программ: Avogadro, Gabedit, PyMOL. В целом, наиболее известные и продвинутые 3D визуализаторы относятся к свободному ПО.

3D-визуализаторы, как правило, имеют инструменты для вращения, приближения-удаления, перемещения изображаемой молекулы, а также для измерения расстояний между атомами, углов между тройками атомов, а также диэдральных углов между плоскостями, задаваемыми четырьмя атомами.

В ситуациях, когда координаты атомов в некоторой молекуле из эксперимента неизвестны, и необходимо получить приближенное строение молекулы, для которой известно только химическое строение (связность атомов), важным функционалом 3D-визуализаторов является возможность «рисования» молекул и первичной оптимизации их геометрии (последняя, как правило, заключается поиске геометрии, имеющей минимальную энергию, методом молекулярной механики, в ряде случаев с применениями ограничений на изменения координат). Примером программы, реализующей такую возможность, является Avogadro, основанная на библиотеке OpenBabel (которая, среди других вариантов, реализует силовое поле UFF [4], параметризованное для всех элементов от водорода до лоуренсия, что позволяет «рисование» как органических, так и неорганических, в том числе металлокомплексных, молекул).

3D-визуализаторы используются также как средство для анализа результатов квантовохимического моделирования, в том числе построения изображений молекулярных орбиталей (рис. 8.4), форм нормальных колебаний (в виде векторов или в виде анимации), траекторий молекулярной динамики и т.д.

Полученные изображения молекулярных структур сохраняются в специальных форматах (как правило, содержащих в том или ином виде координаты атомов и описание связей) и могут быть экспортированы в растровые или векторные изображения (в отличие от 2D-редакторов, возможности по добавлению текста или элементов схем в 3D редакторах намного беднее, однако, это можно сделать с помощью графических редакторов). Кроме того, ряд программ имеет возможность экспортировать сценарий для создания POV-Ray сцены (с ограниченными настройками, рис. 8.5).

Существуют плагины, позволяющие работу с молекулами в Blender, однако их функционал, равно как и размер импортируемых молекул, ограничены.

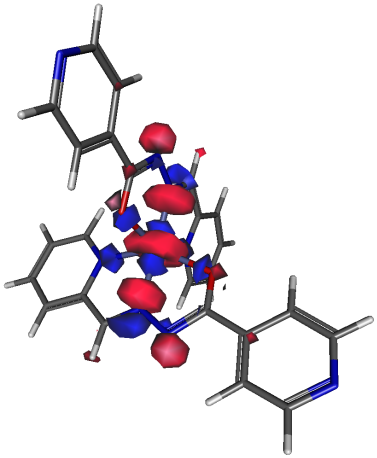


Рис. 8.4. 75% изоповерхность высшей занятой молекулярной орбитали (α) комплексного соединения кобальта(II) с основанием Шиффа из 2-пиридилкарбальдегида и гидразида 4-пиридилкарбоновой кислоты (моделирование описано нами в работе [3]), сгенерированная с помощью программы Gabedit.

Визуализаторы кристаллических структур.

Во многом похожи на 3D-визуализаторы, но отличаются полноценной возможностью работать с кристаллическими структурами — бесконечными периодическими системами (рис. 8.6). Соответственно, этот класс программ имеет возможность размножать элементы кристаллической решетки необходимое число раз операциями трансляции, ограничивать отображение отдельными цепочками или слоями с учетом трансляций, наращивать отображаемую структуру с учетом трансляций, ориентировать структуру вдоль осей элементарной ячейки и т.д. Дополнительными востребованными возможностями является анализ пористой структуры — объем (процент) пустот в кристалле, диаметр и форма пор, в том числе (в случае пор сложного строения) диаметр и форма сужений и расширений пор. Некоторые визуализаторы кристаллических структур (или сопряженные с ними программы) умеют также получать структуры путем обработки данных рентгеноструктурного анализа.

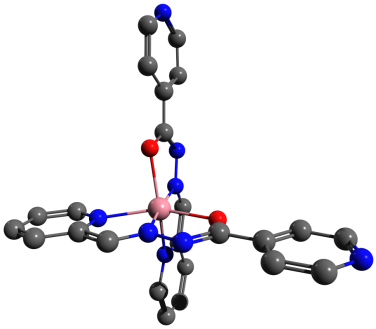


Рис. 8.5. 3D геометрия молекулы (оптимизированная путем квантовохимического моделирования геометрия комплексного соединения кобальта(II) с основанием Шиффа из 2-пиридилкарбальдегида и гидразида 4-пиридилкарбоновой кислоты, моделирование описано нами в работе [3]) в виде шаростержневой модели с уменьшенными радиусами без атомов водорода, сгенерированная с помощью POV-Ray, исходный файл для рендеринга создан программой Avogadro.

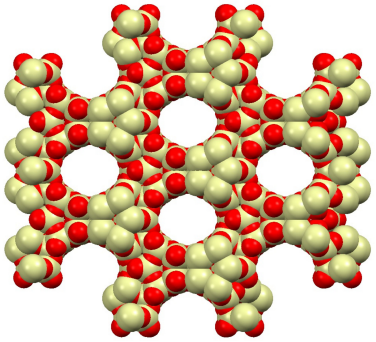


Рис. 8.6. Кристаллическая структура цеолита UTL, построенная в програме Mercury (proprietary freeware), модель — сферы ван дер Ваальса. На рисунке видны поры кристаллической структуры UTL.

К сожалению, в свободном ПО возможности работы с кристаллическими структурами крайне ограничены.

Текущие проблемы и задачи.

- Потребность в новых возможностях манипуляции кристаллическими структурами, в частности, подготовка входных данных и анализ результатов для квантовохимических расчетов с периодическими краевыми условиями (генерация суперячеек, преобразование в специфические форматы файлов, добавление гостевых молекул и т.д.);
- Продвинутый анализ микропористой (с диаметром пор до 2 нм) структуры: характеризация размеров и формы пор с учетом того, что размер поры сравним с размерами атомов (т.е. форма поры в значительной степени определяется расположением отдельных атомов);
- Поиск ответа на вопрос о возможности вхождения гостевой молекулы в микропору — геомерические критерии для сложных по форме пор и молекул, энергетические критерии (энергия активации диффузии молекулы в поре).

Для решения части этих задач нужен поиск или написание программного обеспечения, способного выполнять соответствующие действия, тогда как для последней задачи необходима разработка новых теоретических подходов. Некоторые работы на уровне *proof of concept* ведутся нами: так, доступность пор координационного полимера HKUST-1 (бензолтрикарбоксилата меди), катализирующего сочетание салицилового альдегида с нитрометаном с образованием *транс*-нитровинилфенола, для реагентов и продукта реакции была показана нами путем определения величины энергии активации диффузии молекул в поре HKUST-1 методом молекулярной механики [5].

Выводы.

Таким образом, программное обеспечение для визуализации химического строения вещества является важным инструментом в работе химика, применяемым для ряда связанных между собой задач: анализа строения вещества, подготовки и интерпретации результатов компьютерного моделирования химических веществ и процессов, а также подготовки материалов для публикаций. Свободное ПО такого типа развито недостаточно (за исключением 3D-визуализаторов молекулярных структур). Кроме того, в современной химии существуют задачи по анализу строения химических ве-

ществ, не решаемые или плохо решаемые с помощью существующих программ (в особенности свободных). Дальнейшее развитие визуализаторов химического строения вещества требует совместной работы специалистов в области химии и программирования и представляет собой важную задачу для развития современной химической науки.

Литература

[1] S. H. Chanteau, J. M. Tour, «Synthesis of Anthropomorphic Molecules: The NanoPutians», *J. Org. Chem.*, **2003**, 68(23):8750–8766.

[2] A. Crum Brown, «On the Theory of Isomeric Compounds», *Transactions of the Royal Society of Edinburgh*, **1864**, 23:707–719.

[3] A. S. Lytvynenko, S. V. Kolotilov, M. A. Kiskin, I. L. Eremenko, V. M. Novotortsev, «Modeling of catalytically active metal complex species and intermediates in reactions of organic halides electroreduction», *Phys. Chem. Chem. Phys*, **2015**, 17:5594–5605

[4] A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. Goddard III, W. M. Skiff, «UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations», *J. Am. Chem. Soc.*, **1992**, 114(25):10024–10035.

[5] S. A. Sotnik, K. S. Gavrilenko, A. S. Lytvynenko, S. V. Kolotilov, «Catalytic activity of copper(II) benzenetricarboxylate (HKUST-1) in reactions of aromatic aldehydes condensation with nitromethane: Kinetic and diffusion study», *Inorg. Chim. Acta*, **2015**, 426:119–125.

Enterprise Storage OS (ESOS)*

Александр Клыга, Минск, Беларусь

The Enterprise Storage OS (ESOS) is open-source software to build the Data Storages System within the SAN model. The ESOS is a compact Linux distribution and it is distributed under a GPL license. In my presentation the main components of the ESOS are considered including a loading method and the Data Storages System architecture.

Создание и обслуживание сетей хранения данных (Storage Area Network или сокращенно SAN) одна из интересных технических задач для технических специалистов обслуживающих системы хранения данных в различных компаниях. Прежде всего это обусловлено самой моделью SAN, и техническими требованиями предъявляемыми к системе хранения данных в целом, особенно с учетом широкого использования средств визуализации и необходимостью хранить большие объемы данных.

Очень часто техническим специалистам, сопровождающим крупные системы хранения данных, приходится решать трудные задачи, связанные с расширением объема доступного дискового пространства, надежностью хранения информации и интеграции с другими СХД, в частности с распределенными файловыми системами. Именно поэтому поиск оптимального решения подобных задач является критически важным, а оптимальный вариант должен обеспечивать должный уровень надежности, гибкости и возможности гибкого расширения без сложных структурных реорганизаций.

Поэтому вполне логично, что поиск решений для оптимизации и расширения существующей корпоративной архитектуры сети хранения данных начинается с решений на рынке свободного программного обеспечения (обзор некоторых из них был представлен мною в рамках доклада «Обзор решений на рынке открытого ПО для создания СХД» на зимней сессии LVEE в феврале этого года [1]). По совету коллег несколько лет назад я обратил внимание на один очень интересный проект Enterprise Storage OS (ESOS) [2].

*alex_kls@mail.ru, http://lvee.org/ru/abstracts/219

В ESOS основное внимание у меня привлекло возможность создания дисковых хранилищ для систем хранения данных модели SAN и возможности интеграции существующей SAN в объектное хранилище данных Ceph. С физической точки зрения отдельное дисковое хранилище в SAN представляет собой законченное техническое решение состоящее отдельных дисковых полок (с установленных в них дисками) подключенных к контроллеру управления, а для подключения к сетевой инфраструктуре SAN на нем устанавливаются HBA (host bus adapter) адаптеры. Управление оборудованием осуществляется через консоль или по локальной сети через специальное приложение установленное на компьютере или web – приложение.

Но прежде чем приступить к описанию всех достоинств использования ESOS в SAN, необходимо кратко рассмотреть ее физическую и программную модели.

Физическая модель SAN состоит из трех основных компонентов: серверной фермы, сетевой инфраструктуры и дискового массива. В состав серверной фермы SAN входят сервера SAN не имеющих собственной подсистемы дисковой памяти и подключаются к сетевой инфраструктуре хранилища через специальный адаптеры HBA (host bus adapter). В состав сетевой архитектуры SAN входит активное сетевое оборудование (концентраторы, коммутаторы, мосты, маршрутизаторы) и пассивное оборудование (кабели, коннекторы). В состав дискового массива входят специализированные дисковые стойки (устройства состоящие из десятков дисков управляемые контроллером или сервером) подключаемые к сетевой инфраструктуру через адаптеры HBA.

Программная модель взаимодействия основных компонентов SAN между собой базируется на клиент-серверной модели семейства протоколов Small Computer System Interface (SCSI). Согласно ее клиентские приложения устанавливаются на сервера серверной фермы SAN, сервер выступает в роли инициатора (Initiator) сессии взаимодействия с целевым устройством (Target) в дисковой подсистемой хранилища через сетевую инфраструктуру SAN. Target обрабатывает команды адресуемые ему от Initiator и формирует ответ.

При казалось бы видимой простоте программной модели следует отметить важные моменты:

– взаимодействия между Initiator и Target осуществляется блоками заданной длины (т. е. все устройства представляются в виде сетевых блочных устройств);
– Target это логическое устройство которому присвоен уникальный номер, и с одним Target могут взаимодействовать несколько Initiator;
– Initiator не взаимодействует напрямую с дисками в хранилище данных, а использует одно или несколько логических устройств с уникальным номером — LUN (logical unit number);
– LUN абстрагирован от физической архитектуры дискового хранилища и представляет собой виртуальный диск созданный посредством специального ПО.

Таким образом, становится очевидно, первое, что сервер SAN подключенный с сети хранения данных напрямую не управляет физическими дисками подключенными к нему, все манипуляции с ними осуществляет через LUN (один или несколько) на Target. Второе Target может быть как отдельное физическое хранилище имеющий свой уникальный физический номер, так и может выступать как программный интерфейс к распределенной файловой системе через блочный интерфейс доступа. При этом логический номер Target присваиваться в программном обеспечении контроллера (сервера) управляющего дисковым хранилищем (Data Storage) или сервера на котором установлено программное обеспечение интерфейса блочного доступа к распределенной файловой системе или другому хранилищу данных.

При этом важно отметить, что сервера серверной фермы SAN не имеют собственной подсистемы дисковой памяти и с точки зрения физической архитектуры, LUN выделенный на Target для сервера, представляет собой обычный физический диск на который будет установлена операционная система для развертывания клиентского ПО. Следовательно программное обеспечение с помощью которого создается LUN должно поддерживать эмуляцию большинства типов файловых систем в том числе гипервизоров систем виртуализации, например, VMware.

К моему удивлению все это было объединено в проекте ESOS. В его основе был положен проект SCST (Generic SCSI Target Subsystem for Linux) [3], дополненный всеми необходимыми инструментами управления дисками (включая поддержку RAID-массивов), пакетами драйверов поддержки инфраструктуры SAN (оборудования производства QLogic, Emulex и других вендоров), мониторинга

работы сервера и компонентами для поддержки пулов хранения в формате VMFS VMware ESX/ESXi, томов Windows NTFS и дисков Linux. Что дает возможность создавать хранилища данных (Data Storage) для SAN на базе серверов модели DAS [1]. Также в состав компонентов ESOS входят модули поддержки распределенной файловой системы Ceph (в частности используется Ceph RBD в качестве оконечных устройств хранения данных) и поддержки кластеризации (Pacemaker&Corosync и DRBD).

Исходные коды ESOS находятся на GitHub [4] и компилируется виде Linux-дистрибутива с очень компактным кодом и распространяется под лицензией GPL. Ключевой особенностью ESOS является его полное резидентное расположении в памяти, при этом загрузка осуществляется с USB-накопителя и не требует отдельного диска для установки дистрибутива. При этом реализованы механизмы восстановления работоспособности системы в случае сбоев.

Литература

[1] А. Клыга. Обзор решений на рынке открытого ПО для создания СХД // Материалы конференции LVEE Winter 2016, Минск, 12-14 ферваля 2016 г. `https://lvee.org/ru/abstracts/181`

[2] ESOS: ESOS – Enterprise Storage OS // ESOS `http://www.esos-project.com`

[3] SCST GENERIC SCSI TARGET SUBSYSTEM FOR LINUX // SCST Project `http://scst.sourceforge.net/`

[4] The project ESOS on GitHub // ESOS on GitHub `https://github.com/parodyne/esos`

Миграция виртуальной инфраструктуры на свободное ПО XenServer*

Антон Новиков, Минск, Беларусь

Migrate virtual infrastructure to open source XenServer. Real, complete project.

Введение

Виртуализация плотно вошла в «IT мир», закрепилась в ЦОДах крупного и среднего бизнеса, зачастую внедрение таких систем сопровождается множеством подсчетов и сравнений. Систем с каждым годом, если не сказать месяцем становится больше, времени на реализацию проектов все меньше. Данная статья о крупный игроке на рынке который отдал свою разработку в заботливые руки Open Source. Это — XenServer. Именно о нем как о представителе СПО этот доклад.

О системах Виртуализации

Хороших а главное гибких систем не так уж и много. Как правило выделяются всего пара таких проектов. В качестве ярчайших представителей можно уверенно рассматривать XenServer(СПО) и VmWare ESXi. Да существует огромное количество СПО систем виртуализации, которые бесспорно готовы во многом дать преимущество выше указанным гигантам, но в купе всех качеств в итоге проигрывают.

Почему был выбран именно XenServer

Помимо очевидного — «цены», простота, гибкость, открытость, функционал, сильное комьюнити. Это далеко не весь перечень, но хватает и минусов и к сожалению весьма досадных, вещи которые в других системах воспринимаются как должное в XenServer отсутствуют вовсе. Чего только, стоит вопрос с пробросом USB порта в виртуальную систему.

*arol90@gmail.com, http://lvee.org/ru/abstracts/220

После выбора системы виртуализации предстоял процесс внедрения, трудоемкий процесс и самое главное в данном мероприятии – время простоя бизнеса, и к сожалению в финансовой сфере оно равно нулю. Финансовые вложения тоже ограничены и нельзя просто построить рядом еще один ЦОД и мигрировать всё на него, процесс становится многослойным. Перейдем к нему немного детальнее.

Реальный проект перехода системы вирутализации и инфраструктуры крупной финансовой компании на СПО. Была проведена детальная проработка плана перехода, подготовка оборудования, и многие многие иные вопросы упавшие на плечи нашего IT подразделения, и самое главное все это подразумевало непрерывность бизнеса во время и после интеграции.

Был построен HA кластер на основе Blade серверов HP BL-460c Каким бы свободным не было ПО а серверное оборудования не получится скачать из репозитариев. Вложения в оборудование зачастую в разы выше вложений в персонал или ПО. Описываемый проект реализован на серверных шасси HP c7000 с использованием FiberChannel в качестве сети хранения данных и HP 3Par 7200 в качестве системы хранения.

Была произведена миграция с проприетарного программного обеспечения на СПО Xenserver.

По итогам проекта

Можно сделать вывод, что система справляется с возложенными на неё функциями, отлично управляется, не доставляет хлопот в обслуживании обновлении и администрировании, легко масштабируема. А главное пример доказывает, что СПО может и обязательно будет занимать своё заслуженное место в виртуализации не только мелкого\среднего но и крупного бизнеса.

Основные тенденции развития объектного хранилища данных Ceph*

Александр Клыга, Минск, Беларусь

The new criteria of development of the distributed object data storage Ceph in addition to description of its basic components and their interaction with the Linux kernel will be presented. An interface between Ceph and ESOS in Data Storages System will be discussed as well.

Концепция объектного хранилища данных Ceph [1] была разработана Сейджем Уэйлом в университете Калифорнии в Санта-Круз как часть его докторской диссертации в 2003 году. Научно-исследовательский период разработки этой новой системы хранения данных и ее основных компонентов в рамках университета продолжился до конца 2006 когда, когда под лицензией Lesser GNU Public License (LGPL) был открыт ее код, а к разработке подключились новые разработчики. Первый официальный релиз Ceph вышел в конце 2007 года, и до 2012 кода проект активно развивался и внедрялся на различных платформах, постепенно становясь ключевым решений для построения надежного хранилища данных.

Популярность проекту Ceph обеспечили несколько факторов. Первое, это открытость кода, что позволило быстро развивать проект привлекая в него новых разработчиков. Второе базовые принципы заложенные в Ceph позволяли создавать масштабируемое высоко отказоустойчивое хранилище данных больших объемов (более подробно об этом в моем прошлом докладе «Обзор решений на рынке открытого ПО для создания СХД» [2]). Третье были заключены партнерские отношения с Canonical, RedHat и SUSE, что позволило поддерживать развитие проекта и его поддержку в дистрибутивах Linux. Четвертое тесное сотрудничество с сообществами облачных решений таких как OpenStack, CloudStack и OpenNebula позволило стать Ceph центральным хранилищем данных для них.

В 2012 году Сейджем Уэйлом была основана компания Inktank основной целью которой был выход на рынок программного обеспечения корпоративных клиентов для широкого внедрения решений

*alex_kls@mail.ru, http://lvee.org/ru/abstracts/223

на базе Ceph. Однако, несмотря большую популяризацию Ceph среди программных решений для объектных хранилищ данных, его внедрение в области крупных корпоративных пользователей был относительно невысок. Прежде всего это обуславливалось сильной конкуренцией со стороны ведущих поставщиков программных решений на базе открытого кода и слабостью собственной службы технической поддержки. В конечном итоге в конце апреля 2014 года компания Red Hat поглотила компанию Inktank, а развитие объектного хранилища данных Ceph пошло по новому пути [3]. После того как компания Red Hat взяло под свой контроль все активы связанные с Ceph был разработан план развития системы, сформированы команды разработчиков и технической поддержки, а все наработки по Ceph включая панель управления Ceph CALAMARI [4] были переведены в open source, для того что бы ускорить процесс разработки.

План развития Ceph состоит из нескольких этапов по завершению которых должен быть представлен конечный продукт с длительным циклом поддержки. Основной тенденцией на первом этапе развития Ceph, как составляющей части продуктов компании RedHat, стала глубокое проработка структуры Ceph и ее компонентов и выработки обще концепции развития систем хранения данных компании и выпуск первого стабильного релиза.

В течении последующих двух лет все составляющие компоненты Ceph были детально доработаны и переписаны, особое внимание было уделено интеграции с различными облачными платформами и кластерными системами хранения данных. Итогом этой работы стал выход нового полностью обновленного релиза Ceph V10.2.0 JEWEL представленного компанией RedHat 21-го апреля 2016 года [5], который был объявлен как стабильный и станет основой для формирования новой ветки с длительным циклом поддержки (LTS).

Среди основных изменений в CephFS следует отметить:

- в данном выпуске POSIX-совместимая CephFS объявлена стабильной, однако, некоторые функции отключены по умолчанию, в том числе снапшоты и конфигурация с несколькими активными серверами метаданных;
- утилита по восстановлению работы ФС после сбоя функционирует по полной.
- включена экспериментальная поддержка нескольких CephFS в пределах одного кластера.

В RBD (Ceph Block Device):

– добавлена поддержка зеркалирования разделов (асинхронной репликации) с привлечением нескольких разных кластеров хранения;
– поддержка динамического управления включением или отключением таких возможностей как exclusive-lock, object-map, fast-diff, and journaling;
– добавлена возможность переименования снапшотов RBD;
– полностью переписан интерфейс командной строки, добавлена поддержка автодополнения ввода в bash.

В RADOS Gateway (RGW):

– переписана и перепроектирована система межкластерного взаимодействия;
– добавлена экспериментальная поддержка доступа к данным через NFS;
– реализована поддержка протокола AWS4 и API OpenStack Keystone v3. В RADOS (Reliable Autonomic Distributed Object Store):
– представлен новый более быстрый OSD-бэкенд BlueStore (Object Storage Device);
– файловая система ext4 не рекомендована к использованию в качестве основной для Ceph OSD.

В настоящий момент было выпущено два релиза обновления Jewel, последний датируется 15 июнем 2016 года с номером v10.2.2. На втором этапе развития Ceph основной тенденцией станет выпуск и развитие новой платформы системы хранения данных на базе стабильного релиза (в настоящее время это v10.2.0 Jewel). Ее презентация состоялась относительно недавно 22 июля 2016 года, а новая платформа получила название Red Hat Ceph Storage 2 [6]. Основным особенностями новой платформы являются:

– работа с глобальными объектными кластерами, поддерживающими единое пространство имен и синхронизацию данных между кластерами, развернутыми в разных регионах;
– повышенная защищенность за счет интеграции с такими популярными системами аутентификации, как Active Directory, LDAP и OpenStack Identity (Keystone) v3;
– тесная совместимость со средами Amazon S3 и OpenStack Object Storage (Swift), включая поддержку AWS v4 Client Signatures, версионность объектов и массовое их удаление.

Также в состав новой платформы хранения данных включена обновленная система мониторинга и управления хранилищем пришедшая на смену Ceph CALAMARI.

Литература

[1] Ceph: the future of storage // ceph.com http://ceph.com

[2] А. Клыга. Обзор решений на рынке открытого ПО для создания СХД // Материалы конференции LVEE Winter 2016, Минск, 12-14 ферваля 2016 г. https://lvee.org/ru/abstracts/181

[3] Red Hat Ceph Storage // Red Hat Ceph Storage https://www.redhat.com/en/technologies/storage/ceph

[4] CEPH CALAMARI // Dashboard CEPH CALAMARI http://ceph.com/community/ceph-calamari-goes-open-source/

[5] Ceph V10.2.0 JEWEL // Ceph V10.2.0 JEWEL http://ceph.com/releases/v10-2-0-jewel-released/

[6] Red Hat Ceph Storage 2 // Red Hat Ceph Storage 2 https://www.redhat.com/en/about/press-releases/red-hat-unveils-red-hat-ceph-storage-2-enhanced-object-storage-capabilities-improved-ease-use

Обобщенные деревья поиска: от теории к практике*

Андреев Юрий, Санкт-Петербург, РФ

This article describes the Generalized Search Tree (GiST), an index structure, which allows new data types to be indexed in a manner supporting queries natural to the types.

В статье дается краткий обзор обобщенных деревьев поиска (GiST), которые облегчают и унифицируют разработку методов доступа, оптимальных для конкретного типа данных.

Постановка задачи

За эффективный доступ к данным в современных Системах Управления Базами Данных (СУБД) отвечают специальные структуры, ***индексы***, *реализующие алгоритмы поиска и обеспечивающие поддержку надежности, конкурентности, восстановления.* Используя особенности конкретного типа данных, обычно удается найти алгоритмы, работающие эффективнее общих схем. Однако, реализация таких алгоритмов в индексе сопряжена с проблемами обеспечения должной поддержки. Решение заключается в том, чтобы реализовать в СУБД структуру индекса на более высоком уровне абстракции (GiST), оставив пользователю свободу в описании алгоритма через функциональный интерфейс.

Классической задачей, требующий специальный подход к индексированию, являются задачи с пространственными данными, *spatial searching*. Примером такой задачи может быть нахождение всех географических объектов, расположенных в определенной близости к заданному. Осложнением в данном случае являет то, что такие объекты часто не могут быть представлены лишь парой координат, а занимают некоторую площадь.

*andreev_yurij@mail.ru, http://lvee.org/ru/abstracts/232

Реализация

В качестве алгоритма для решения описанной задачи, задачи индексации многомерной информации, было предложено построение R-Tree дерева (*Rectangle* Tree). Хорошее представление об алгоритмах, работающих с различными типами данных, можно получить, познакомившись с такими структурами, как R-Tree, RD-Tree (*Russian Doll* Tree), KD-Tree (*k-Dimensional* Tree) ([1], [2], [3], Wikipedia).

Обобщенное дерево поиска, ***Generalized Search Tree (GiST)***, обеспечивает унифицированный подход для реализации алгоритмов [4]. (В частности, посредством GiST может быть реализовано R-Tree дерево.)

Для детального практического изучения индексирования, важно иметь свободную СУБД, с реализацией GiST. Также желательна хорошая расширяемость и документация. Примером такой СУБД может служить СУБД PostgreSQL, обладающая всеми перечисленными свойствами. PostgreSQL предоставляет семь интерфейсных функций, для работы со структурой GiST. Они, в свою очередь, манипулируют тремя базовыми методами дерева: search, insert, delete.

Пользователь может контролировать само построение дерева поиска с помощью функций `union` и `picksplit`, отвечающих за объединение и разделение узлов дерева. Функцией `penalty` определяется метод оценки сложности вставки новых данных. Минимизация «пенальти» является показателем эффективности индексации и учитывается при перестроении дерева.

Пользователь может контролировать поиск по дереву. Решение о переходе в один из дочерних узлов принимается с помощью функции `consistent`.

Реализация и примеры использования GiST в PostgreSQL подробно изложены в [5]. Наряду с GIST, в PostgreSQL реализованы и другие обобщенные структуры, такие как SP-GIST (*Space-Partitioned* GIST). Техническая документация также является хорошим и легко читаемым источником информации [6].

Литература

[1] Гектор Гарсиа-Молина, Джеффри Ульман, Дженнифер Уидом, «Системы Баз Данных», изд. «Вильямс» 2002; в частности, гл.14 «Многомерные и точечные индексы»

[2] Antonin Guttman, «R-Trees: A Dynamic Index Structure for Spatial Searching»
[3] Joseph M. Hellerstein, Avi Pfeffer, «The RD-Tree: An Index Structure for Sets»
[4] Joseph M. Hellerstein, Jeffrey F. Naughton, Avi Pfeffer, «Generalized Search Trees for Database Systems», In Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, 1995.
[5] Олег Бартунов, Федор Сигаев, «Написание расширений для PostgreSQL с использованием GiST», `http://citforum.ru/database/postgres/gist/`
[6] PostgreSQL Documentation, `https://www.postgresql.org/docs/`

Zabbix — контроль ситуации везде и всегда*

Тимофей Пирожник, Минск, Беларусь

With the development of infrastructure and the addition of services (third party products) it becomes too difficult or even impossible to keep track of the current state of systems and services. The most correct response on such challenge is to use the monitoring system. Zabbix project is a free/libre solution for such a task.

Введение

С развитием инфраструктуры и добавлением серверов и сервисов (сторонних продуктов) уследить за текущим состоянием информационной системы становится чересчур тяжело либо скорее всего невозможно. Наиболее правильный выход из сложившейся ситуации — ввод в эксплуатацию системы мониторинга.

Когда мы столкнулись с подобной идеей, из представленных программных продуктов была сделана простейшая выборка, которая исключала коммерческие продукты (в т.ч. таковые с бесплатным пробным периодом); также были исключены системы мониторинга, использующие в своей непосредственной работе Java либо Javascript; исключены системы, работающие исключительно с *nix (на момент внедрения в организации использовались различные ОС). В результате выбор пал на Zabbix.

Обзор возможностей/ Технический обзор

Согласно документации, Zabbix следит за «узлами» сети. Под узлами подразумевается все, что можно опросить и получить удобочитаемый/внятный ответ об их состоянии. Очевидный объект для опроса — физические (гипервизоры) и виртуальные сервера. Приложив определённые усилия, можно наблюдать за рабочими станциями (это актуально для тех, кто работает на аутсорсе). Также

*ghrono@gmail.com, http://lvee.org/ru/abstracts/222

доступно наблюдение за состоянием определенных сервисов (приложения, службы, ресурсы, веб-мониторинг). И конечно же, доступен мониторинг для сети и сетевого оборудования.

Минимальные требования по аппаратной части для сервера мониторинга вариативны из-за разного количества узлов и наблюдаемых параметров. Например, два разных сервера Zabbix с приблизительно одинаковым количеством узлов и одинаковой архитектурой (на одном сервере установлен сам Zabbix и база данных), работающие через агент и SNMP, потребовали совершенно разные ресурсы.

Способы мониторинга также разнообразны. «Из коробки» предлагаются такие варианты, как web-мониторинг — проверка наличия в доступе станицы (сайта) плюс имитация действий пользователя, статистические данные (время ответа). Протокол SNMP — один из основных способов мониторинга сетевого оборудования, и его поддерживает подавляющее количество устройств. Также используется Zabbix-агент — клиентская часть для серверов и рабочих станций.

Набор действий при появлении событий на первый взгляд скромен: можно либо отправить сообщение о событии, либо выполнить некое действие. Варианты оповещения достаточно вариативны, и выбор протокола остается за администратором. Действия, в контексте Zabbix, это выполнение скриптов, и таки образом способы реагировать на события расширяются до границ фантазии администратора.

Опыт использования в реальном проекте

Zabbix был использован нами для построения системы мониторинга за сервисом «клиент-банк» крупной финансовой компании. Он показал себя как отличный пример решения open source, которое работает «из коробки», хотя и требует определенной грамотности и здравого смысла при развёртывании. По нашему опыту можно сказать, что в большинстве ситуаций внедрение не вызывает сложностей, но есть незадокументированые особенности, выявление которых происходит только во время эксплуатации.

Нестандартное применение Zabbix

С завидным постоянством (вероятно, в результате развития информационных технологий) появляются задачи, которые по сути не

относятся к мониторингу, но могут быть реализованы средствами системы. Система мониторинга как таковая ограничивается только фантазией администратора, и потому при необходимости легко обеспечивает выход за рамки стандартных проверок серверов и сервисов.

Заключение

После ввода системы мониторинга в нашем случае сократилось время простоя основных сервисов, предоставляемых организацией (соответственно начался спад недовольных клиентов), появилось больше времени на оптимизацию сервисов и улучшение их качества.

Using Hadoop stack to build a cloud VAT declarations revising service*

Alex Chistyakov, Saint-Petersburg, Russian Federation

Recent changes in Russian federal law require organizations to submit tax information in electronic form. Federal Tax Service of Russia revises this information and sends automatically generated claims. «Kontur.NDS+» project helps businesses to perform this revising in the cloud before tax documents are submitted to the Federal Tax Service. «Kontur.NDS+» is built on Hadoop stack (which is quite common), sharded and replicated Solr (which is quite common too, but required some tuning) and Perl (which is not common at all). The project evolved the last 1.5 years and this evolution is a subject of current presentation.

Hadoop is a free/libre software project, which includes tools, libraries and framework to build distributed applications, hosted on clusters of handreds and thousands of nodes. Hadoop is used as the basis of search engines in many well-known high load websites, including Facebook.

Our own hadoop-based project presented here was started more than 1.5 years ago and was basically a startup at that time. We did not get brand new servers and therefore had to design a fully fault-tolerant system without a SPOF (single point of failure). A standard Hadoop stack used by everybody is not fault tolerant right out of the box, so we had to carefully plan and implement a quorum-based system.

Since our project was the only cloud-based solution for tax documents revising, it quickly began to dominate on the market. Our team had to use advanced performance measurement and optimization techniques to fulfill customer expectations. We tried many single-node and cluster-based monitoring tools before we finally settled on stack, built of Graphite [1], Whisper [2] and Grafana [3].

We record and analyze many Hadoop- and HBase-related metrics, JVM (memory and GC-related) metrics and our custom business metrics. We use flamegraphs (a technique popularized by Brendan Gregg [4] to record and analyze Perl stack frames. We also had to

*alexclear@gmail.com, http://lvee.org/ru/abstracts/229

dump Solr [5] memory to better understand why it needed so much RAM.

We have also tailored our software development process to meet performance requirements. Number of requests and overall load is not the same during the year, maximums of load are at the quarter ends, so we were able to plan and perform thorough stress testing. We maintain this stress testing procedure routinely every quarter.

Perl in particular and dynamically typed languages in general do not seem to be a safe choice for developing a lot of business logic, but we were able to overcome limitations of the dynamically typed language. We utilize a thorough code review process and a set of assertions and cross checks to ensure our implementation is valid.

Every quarter means a lot of new documents for us, so we are constantly seeking the right tool to do our job better. We are testing the Phoenix extension to HBase right now. Phoenix [6] is basically an SQL-like engine which does not use Hadoop YARN resource management to schedule and run queries. Phoenix will allow us to experiment with ad-hoc OLAP & OLTP — online analytical and transactions processing queries.

And, of course, we use and love PostgreSQL, but it is for not-so-big data, as known.

References

[1] Graphite monitoring tool `https://github.com/graphite-project/graphite-web`

[2] Whisper Databse `https://github.com/graphite-project/whisper`

[3] Grafana data visualizer `http://grafana.org/`

[4] B. Gregg. CPU Flame Graphs `http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html`

[5] Solr search platform `http://lucene.apache.org/solr/`

[6] Apache Phoenix `http://phoenix.apache.org/`

О создании bartop arcade автомата*

Alexandr Sorokin, Minsk, Belarus

Experience share of building the bartop arcade cabinet from ground up, based on open source technologies.

Введение

Аркады были очень популярны в 70-80е годы. Для некоторых людей это существенная часть детства/юности. Аркадные игры поражают своей увлекательностью и простотой. Несмотря на то, что современные игры стали реалистичнее, красочнее, и «онлайновее», играть в старые аркады по-прежнему интересно. Аркадный автомат становится центром притяжения в веселой компании или в рабочем офисе. Есть несколько вариантов добыть себе такой автомат:

Моя история. Начало

Я выбрал вариант самостоятельной сборки игрового автомата с нуля. Целью было набить руку в столярном деле, сборке электроники. И, можно отметить, что даже в таком небольшом проекте

*a_sorokin@wargaming.net, http://lvee.org/ru/abstracts/187

удалось добить поставленной цели, получив массу опыта. Начать было нелегко, работа требовала пространства, времени и инструментов. В Минске я узнал о хакерспейсе, который Сложности, мешающие начать работу над проектом, удалось решить благодаря помощи Минского хакерспейса.

Эмуляторы

Сейчас доступно огромное количество эмуляторов для десятков разнообразных платформ, включая игровые. Весь этот зоопарк эмуляторов организован в системы с фронтендами и даже дистрибутивами.

Наиболее передовые эмуляторы — LibRetro и M.A.M.E (M.A.M.E. — идеологический отщепенец). Наиболее известные фронтенды — emulation station, RetroArch, FB Alpha. Популярные дистрибутивы — RetroPie, Lakka.

Работа над аппаратом. выбор железа. список покупок

Существенная доля составных частей аппарата — легко доступные материалы и детали. Но также требуется и немного специализированных частей, которые лучше приобрести заранее. Самое необходимое — это набор аркадных кнопок и джойстиков. Все остальное более-менее взаимозаменяемо.

Работа над аппаратом. Первые шаги.

Моделирование корпуса проводилось в OpenScad — открытом пакете для моделирования путем написания программ на специальном скриптовом языке. Проектирование отталкивалось от идеи, что аппарат должен быть легко разборным, с возможностью заменять отдельные части. Для начала был изготовлен прототип из картона:

Прототип — важная стадия проекта, и учет результатов и выводов этого этапа значительно облегчает выполнение последующих этапов.

Работа над аппаратом. Корпус

Для копруса может быть использован любой легко обрабатываемый материал. Я выбрал клееный деревянный щит для боковин и фанеру 10мм для остальных частей корпуса. Удалось сделать разборными части с монитором и консоль с джойстиками, как и планировалось на этапе проектирования. Тем не менее, был допущен ряд просчетов, которые сказались на внешнем виде аппарата.

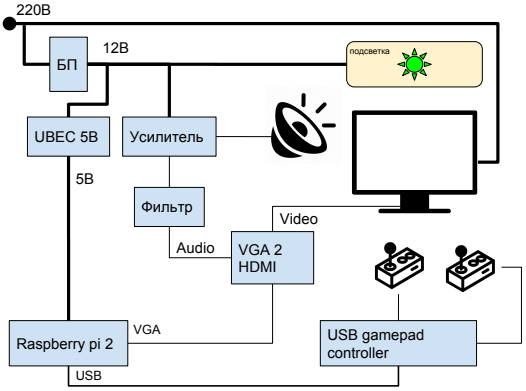
Работа над аппаратом. Электроника

Внутри аппарата была необходимость использовать цепи с напряжением 5V и 12V. Изначальной задумкой было использовать интегрированный блок питания на два напряжения. Но в итоге пришлось остановиться на блоке 12V с дополнительным преобразователем DC-DC 12V-5V.

Основной проблемой в части электроники стала аналоговая часть, а именно шумы в колонках. Этому было несколько причин. Проблемы были как с усилителями китайского производства, так и с эффектом ground loop. Последняя проблема решилась только добавлением дополнительного фильтра, хотя по сути ее можно было бы решить отдельными блоками питания AC-DC для разных цепей.

В качестве микроконтроллерной платформы для аппарата был использован Raspberry Pi. Это привнесло свою специфику, поскольку Pi чувствителен к помехам по питанию и подключениям в USB. Кроме того, в заводском варианте не предусмотрена кнопка RESET, и ее тоже пришлось делать.

Освещение верхней панели сделано при помощи двух LED-модулей. Благодаря матовой рассеивающей пленке освещение практически равномерно.



Работа над аппаратом. Программная часть

Благодаря готовому дистрибутиву RetroPie настройка ПО прошла практически безболезненно. Но конечно не обошлось без загвоздок. Потребовалось настраивать канал вывода звука (типовая проблема Pi), а также распознавание джойстиков, которые по умолчанию определялись как USB-мышь.

Дополнительного времени требует настройка M.A.M.E.: требуется скачивать BIOS-образы платформ, а также конвертировать образы игр. На этом же этапе обнаружилось, что количество реализованных в аппарате кнопок для многих игр M.A.M.E. маловато.

Литература

[1] Фронтэнд FB Alpha `http://www.fbalpha.com`
[2] Эмулятор M.A.M.E. `http://mamedev.org`
[3] Дистрибутив RetroPie `https://retropie.org.uk`
[4] Дистрибутив Lakka `http://www.lakka.tv`
[5] Фронтэнд RetroArch `http://www.libretro.com/index.php/retroarch-2/`
[6] Эмулятор LibRetro `http://www.libretro.com`
[7] САПР OpenScad `http://openscad.net`
[8] Образы ROM `https://duckduckgo.com/?q=arcade+roms`
[9] Описание проекта в персональном блоге `http://alexsorokin.ru/retro-cabinet/`
[10] Опубликованные исходные коды `https://github.com/Gromina/bartop_arcade`

Криптографический алгоритм DersCrypt*

Sergey Derevyago, Minsk, Belarus

An original block symmetric cryptography algorithm DersCrypt is presented. Specifics of implementation is discussed, as far as reference and production implementations.

DersCrypt — это блочный симметричный криптографический алгоритм, построенный на не использовавшихся ранее принципах. Суть работы алгоритма состоит в переводе числа из одной системы счисления в другую, перестановке «цифр» и обратном переводе в исходную систему счисления.

DersCrypt обладает следующими достоинствами:

- Нет ограничений на длину ключа;
- Криптоалгоритм предельно прост для понимания и реализации;
- Криптоалгоритм является открытым и свободным для использования.

Впервые криптоалгоритм был представлен в феврале 2004 года в виде сообщения в профильные ньюсгруппы UseNet [1]. Наиболее ценное обсуждение произошло в конференции fido7.ru.crypt сети FidoNet. В апреле 2005 года увидела свет первая версия DersCrypt, заметно отличавшаяся от того, что было представлено в самом начале [2]. Текущей является версия алгоритма 1.1.

Особенности алгоритма

Базовый алгоритм DersCrypt состоит из следующих шагов:

1. Текст для шифрования рассматривается как неотрицательное число A_1 в некоторой системе счисления S_1 по основанию B_1.
2. Число A_1 переводится в другую систему счисления S_2 по основанию B_2. Число B_2 является ключом.
3. Цифры числа A_1 в системе счисления S_2 переставляются, в результате чего получается число A_2. Перестановка задается значением ключа.
4. Число A_2 переводится в исходную систему счисления S_1.

*derevyago@yahoo.com, http://lvee.org/ru/abstracts/193

5. Число A_2 в системе счисления S_1 и является результатом шифрования.

Сам по себе базовый алгоритм DersCrypt непригоден к практическому применению в силу существования следующих эффективных способов атаки:

1. Если зашифровать два числа, отличающихся только значением младшего бита, то разность зашифрованных чисел, очевидно, будет равна B_2^n для некоторого n.
2. Существует и более эффективный способ атаки: Max Alekseyev указал на то, что имея число A_1 в системе счисления по основанию B_2 и число A_2, полученное из A_1 произвольной перестановкой цифр, разность $A_1 - A_2$ всегда делится на $B_2 - 1$.

Таким образом, DersCrypt не должен сводиться к простой перестановке цифр в некоторой системе счисления. Решение достаточно очевидно: перед применением базового алгоритма, число A_1 превращается в некоторое число A_1' посредством приписывания к A_1 некоторого количества случайных байт слева и справа. После дешифрования приписанные байты достаточно просто отбросить, получив исходное число A_1.

В силу того, что действительно случайные байты довольно сложно получить на практике, в DersCrypt используется предварительное шифрование числа A_1 базовым алгоритмом и извлечение из полученного числа A_2 последовательности бит нужной длины. На самом деле, для учета значения предыдущего зашифрованного блока текста, процедура формирования псевдослучайной битовой последовательности чуть более усложнена. Ознакомиться с ней можно непосредственно по коду эталонной реализации DersCrypt, которая, благодаря детальным комментариям [3], определяет сам алгоритм и предназначена для практического исследования его свойств.

Реализации алгоритма

Языком эталонной реализации выбрана Java, а исходный код оформлен исходя из упрощения понимания, а не производительности и/или простоты включения во внешние проекты.

Также имеются реализации на Java [4] и C++ [5], предназначенные для практического использования. Об их сравнительной производительности можно судить по следующей таблице, в которой приведена скорость шифрования в килобайтах в секунду на 256-битном ключе для двух компьютеров разной архитектуры:

	Эталонная реализация	Java-реализация	C++-реализация универсальная	C++-реализация оптимизированная
Pentium4	52.5 K/s	74.2 K/s	351.8 K/s	612.7 K/s
Athlon	86.4 K/s	127.2 K/s	1149.2 K/s	1404.6 K/s

Как можно видеть, оптимизированная C++ реализация в 8-11 раз быстрее Java реализации, которая, в свою очередь, на 40-50 процентов быстрее эталонной. Судя по всему, неприемлемой производительностью Java реализации обязаны стандартному классу java.math.BigInteger, чей код никак нельзя признать оптимальным.

Исходный код реализаций алгоритма свободно распространяется с правом копирования, использования, модификации, продажи и распространения при условии сохранения сведений об авторе [4] [5].

Литература

[1] Анонс алгоритма `http://ders.stml.net/crypt/derscrypt/orig.html`

[2] С. Деревяго. Криптографический алгоритм DersCrypt `http://ders.angen.net/crypt`

[3] Комментарии к эталонной реализации алгоритма `http://ders.stml.net/crypt/derscrypt/derscrypt.html#3.3`

[4] Java-реализация алгоритма `http://ders.stml.net/crypt/derscrypt/java/src.zip`

[5] Реализация алгоритма на C++ `http://ders.stml.net/crypt/derscrypt/cpp/src.zip`

Hidden powers of screen and xterm: multiple windows inside of a terminal*

Andrew Savchenko, Moscow, Russian Federation

Xterm and screen are both well known software, but they are more powerful than they look like at a first glance. This talk reveals some of their not so widely used features which may increase productivity, improve user experience or are just fun. Some problems and their solutions are also discussed.

XTerm

Default xterm looks unpleasantly in most distributions: small, barely readable fonts, no multiple tabs or CJK [1] support. Thus many users disregard it, though it is one of the most powerful applications if tuned properly.

All numerous features of XTerm can be tuned via X Resources. Ways of providing them depend on WM and distribution, but portable way is to run (file name may be arbitrary): `xrdb ~/.Xresources`

Fonts

Arbitrary TTF font of your liking may be set using following resources:

```
xterm*faceName: DejaVu Sans Mono:style=Bold
xterm*faceSize: 15
```

On some fonts and sizes symbols like «_» may be unreadable, to correct this use: `xterm*scaleHeight: 1.01`

Let's add some CJK love (double size glyphs CJK font and 4-byte width symbols):

```
XTerm*combiningChars: 4
XTerm*faceNameDoublesize: mikachan
```

Of course, CJK must be properly configured, e.g.:

*bircoph@gmail.com, http://lvee.org/ru/abstracts/226

```
export GTK_IM_MODULE=uim
export QT_IM_MODULE=uim
export XMODIFIERS=\@im=uim
uim-xim --engine=anthy-utf8 &
```

Colors

256-color palette can be enabled: `xterm*termName: xterm-256color`

Custom RGB colors configuration:

```
xterm*color1:    rgb:b2/18/18
xterm*color2:    rgb:18/b2/18
...
```

It is possible to replace bold and underline attributes by color:

```
xterm*colorBDMode: True
xterm*colorBD: magenta3
xterm*colorULMode: True
xterm*colorUL: cyan3
```

Limitation: if locale is not UTF-8 and wide chars (CJK) are used, 256 colors are not available due to limited number of XResources. *Solution:* use UTF-8!

Dynamic colors change (for already printed text) can be done using escape sequence: `\033]<target>;#RRGGBB\007`

E.g. to set background to #1818B2 (marine blue): `echo -ne "\033]10;#1818B2\007"`

For details about possible targets see [2], example scripts are at [3].

Screen

GNU screen is a terminal manager, you may think of it like a window manager for terminals.

Why do you need that? There are so many terminals with multiple tabs and various features available... Because of its client–server architecture! It is possible to continue your work remotely, to survive broken ssh link like it was nothing and so on.

Screen server is started and stopped automatically and manages terminal sessions which can be attached and detached arbitrary by the user. This allows to connect both locally and via ssh from different hosts or interrupted connections.

Just start *screen*, you'll have an ordinary terminal (in some distributions you may see information info). It works like an ordinary terminal. All screen controls can be accessed via *C-a-** (^A) sequence, e.g. *C-a-c* will create new terminal, *C-a-n* will switch to next one, *C-a-p* will switch to previous terminal, C-a-"shows list of terminal windows available.

Screens can be detached from underlying terminal via *screen -d* or *C-a-d* and reattached using *screen -r* or *screen -RD* (with forced reattach first).

Configuration

Tired of repeating *C-a-** here and there? Let's configure it to something else. Screen's configuration lies in ~/.screenrc (though any file may be provided using -c option). The following option will bind *C-a* (command sequence) to *menu* key (on most common keyboards):

```
bindkey ^[[29~ command
```

But how to get that *^[[29~* escape sequence? This little trick will help: `cat > /dev/null` Now press any hotkeys you like and if they are not intercepted by other software (e.g. VM or DE), you will see their escape sequences.

This way 256 colors and bce (background color erase) can be enabled in screen:

```
term screen-256color-bce-s
defbce on
```

In screen each terminal window has its own history. It can be access via copy mode (*C-a-ESC* or *C-a-[*). It is possible to search within it (*C-S* for forward search, *C-R* for backward direction), select and copy using *space*, paste using *C-a-]*. But default history depth is just 100 lines per windown. This can be fixed via: `defscrollback 3000`

Status line

It is possible to configure status line showing date, time, list of windows, names of current directories per window and highlight an active one (and more information available for screen, see its man page for details).

This will create a simple status line with green on black [date(blue) time] in the right corner: `hardstatus string '%{=kG}%= [%{B}%d/%m %{G}%0c:%s]'` and always enable it an the bottom line: `hardstatus alwayslastline`

Here things became more complicated: `hardstatus string '%{ =kG}[%= %{= kw}%?%-Lw%?%{m}(%{W}%n*%f%t%?(%u)%?%{m})%{w}%?% +Lw%?%?%= %{G}][%{B}%d/%m %{G}%0c]'` To the right of date/time from previous example list of windows is added, active window is highlighted and window title is being shown.

PS1 variable can be modified to truncate window name to 8 chars (or whatever you like) and collapse \$HOME to ~, so put something alike in bashrc:

```
case \${TERM} in
...
        screen*)
                PROMPT_COMMAND='tmp="\${PWD/\$HOME
                    /\~}";␣tmp=\${tmp##*/};␣[[␣-z␣\$tmp
                    ␣]]␣&&␣tmp=/;␣echo␣-ne␣"\033k\${tmp
                    :0:8}\033\\";␣unset␣tmp'
                ;;
esac
```

Now we have something yummy like shown on fig.1.

Moving around windows using *menu-p*, *menu-n* and *menu-"* is still tiring. How about using arrows (*S-right* and *S-left*) for moving between windows and *C-S-right*, *C-S-left* to move windows themselves? The following lines does this:

```
bindkey ^[[1;2D prev
bindkey ^[[1;2C next
bindkey ^[[1;6D bumpleft
bindkey ^[[1;6C bumpright
```

Troubleshooting

Sometimes after su, sudo or ssh 256 color depth or bce are lost. This happens because \$TERM **and** \$TERMCAP environment variables are not passed. To fix this either reconfigure your software to pass *and* accept these variables (look for ssh_config SendEnv, sshd_config AcceptEnv and so on) or set \$TERM properly and set *bce on* in screen session.

References

[1] CJK is a collective term for the Chinese, Japanese, and Korean languages. `https://en.wikipedIA.org/wiki/CJK_characters`

[2] fn2. `http://rtfm.etla.org/xterm/ctlseq.html`

[3] fn3. `http://rcr.io/words/dynamic-xterm-colors.html`

Пастильда — открытый аппаратный менеджер паролей*

Иван Ларионов, Санкт-Петербург, Россия

Pastilda - simple and cheap opensource device that allows a hardware safekeeping and entering usernames/passwords on all types of devices

Введение

Хранение паролей — актуальный вопрос, волнующий пользователей и руководство IT отделов организаций. Существующие программные и аппаратные средства хранения паролей имеют ряд недостатков:

- закрытый код снижает доверие и вероятность оперативного устранения уязвимостей;
- для автозаполнения нужно ставить дополнительный софт;
- после ввода мастер-пароля вся база открыта и доступна, в том числе для вредоносного ПО, что особенно актуально на недоверенных устройствах;
- использование мобильных приложений для хранения паролей все равно подразумевает ручной ввод с клавиатуры, например когда требуется залогиниться на стационарном ПК;
- автозаполнение невозможно в некоторых случаях (в bios, консоли, вход в систему)

Пастильда

Устройство Пастильда призвано решить задачи безопасного хранения и ввода паролей на любых устройствах без указанных недостатков. В нем реализованы две прорывные идеи:

- Пастильда является умным USB-USB переходником для клавиатуры. В пассивном режиме команды с клавиатуры транслируются насквозь, в активном — интерпретируются устройством.

*i.larionov@thirdpin.ru, http://lvee.org/ru/abstracts/228

– Меню устройства может быть вызвано в любой строке ввода, где расположен курсор, при переходе в активный режим.

Хранение паролей

Для хранения и управления учетными записями с паролями используется база KeePass `http://keepass.info/`, хранящаяся на встроенном накопителе. С базой можно работать стандартными средствами из любой ОС, обновлять через облака. Для доступа к базе используется мастер-пароль и PIN-код. PIN-код позволяет при переходе в активный режим не вводить мастер-пароль, что усложняет взлом базы удаленно с подглядыванием. Мастер-пароль никогда не вводится непосредственно в ПК, а ОС не получает доступа ко всей базе.

USB

Пастильда для ОС представляет собой составное USB-устройство, содержащее MSD (функция USB-флеш для непосредственной работы с файлом базы данных через файловую систему) и HID (функция клавиатуры для транслирования команд в пассивном и активном режимах). Для клавиатуры Пастильда является хостом.

Меню

Для отображения однострочного меню Пастильда вводит и стирает символы в активном поле ввода. Поле ввода может быть в браузере, командной строке, блокноте и так далее. Меню позволяет использовать функции: добавление записи в базу паролей, использование существующих пар логин-пароль. Для быстрого выбора среди множества записей планируется реализовать подсказки при вводе.

Open source

Исходники опубликованы на GitHub `https://github.com/thirdpin/pastilda`, причем как софт, так и аппаратная часть. Проектом активно интересуются частные лица и компании. Планируется развитие софта, разработка новой аппаратной версии.

Связанные открытые данные в университете*

Ирина Радченко, Михаил Навроцкий,

Санкт-Петербург, Россия

LinkedUniversities.org is one of important projects in the field of Linked Open Data in European universities. ITMO University joined linked open data in universities initiative in 2014. ISST Labs is an organization responsible for developing LOD-IFMO project. I am going to give a talk on project progress and outcomes.

Уже несколько лет существует европейский проект связанных университетов LinkedUniversities.org, в рамках которого несколько европейских университетов выкладывают наборы связанных открытых данных.

В 2014 году в лаборатории ISST университета ИТМО [1] был запущен проект LOD-IFMO [2] — проект связанных открытых данных в российском университете. Этот проект был инициирован совместно с коллегами из исследовательской группы Agile Knowledge Engineering and Semantic Web (AKSW) Лейпцигского университета [3].

Одной из важных целей этого проекта было присоединение к европейской инициативе университетских связанных открытых данных. Для этого университетские открытые данные (данные по учебным курсам, публикациям, исследовательским проектам, профессорско-преподавательскому составу и зданиям университета) необходимо сделать связанными открытыми данными, а точнее — перевести в форматы представления данных по модели RDF (Resource Description Framework) при помощи таких онтологий, как FOAF, BIBO, VIVO, AIISO и др. и связать их со связанными открытыми данными европейских университетов. Для перевода данных из университетских баз данных в RDF-форматы представления данных в проекте LOD-IFMO используется kOre framework (kOre: Lin*k*ed Data for *O*penScience Info*r*mation Int*e*gration).

*iradche@gmail.com, http://lvee.org/ru/abstracts/230

Серверная часть системы поддерживается при помощи платформы управления данными Virtuoso Universal Server [4]. Это платформа с гибридной серверной архитектурой, позволяющая управлять как реляционными базами данных, так и системами связанных открытых данных, осуществлять управление контентом и веб-приложениями.

Фронт-энд для развертывания интерфейса доступа SPARQL-endpoints разрабатывался при помощи Pubby [5], который позволяет организовать интерфейс связанных данных на основе SPARQL-endpoints. Исходный код проекта выложен на GitHub [6].

Помимо технологической сложности развертывания подобных проектов (проектов с использованием связанных открытых данных) существует сложность в организации работы пользователей с данными, представленными по модели RDF. Для решения этой задачи необходимо разработать учебный курс по работе со связанными открытыми данными, а также систему понятной документации.

К сожалению, на настоящий момент времени русскоязычная документация на подобные системы почти не представлена. В силу этой причины участниками проекта было принято решение о создании раздела сайта университета ИТМО, посвященного особенностям использования связанных открытых данных в университете. В этом разделе планируется также выкладывать инструкции пользователя, словари данных, описание интерфейса доступа SPARQL-endpoints, документация на систему и тд. Предварительные макеты проекта расположены по следующему адресу: luitmo.herokuapp.com `https://lu-itmo.herokuapp.com/`.

Литература

[1] Лаборатория ISST университета ИТМО `http://semantics.ifmo.ru/`semantics.ifmo.ru.

[2] Проект LOD-IFMO `http://lod.ifmo.ru`.

[3] Группа Agile Knowledge Engineering and Semantic Web Лейпцигского университета `http://aksw.org/About.html`.

[4] Virtuoso Universal Server `http://virtuoso.openlinksw.com/`.

[5] Pubby `http://wifo5-03.informatik.uni-mannheim.de/pubby`.

[6] Исходный код проекта на GitHub `https://github.com/AKSW/itmolod`.

Досвед апрацоўкі відэа з дакладаў ці лекцый з дапамогай kdenlive*

Андрэй Захарэвіч, Мінск, Беларусь

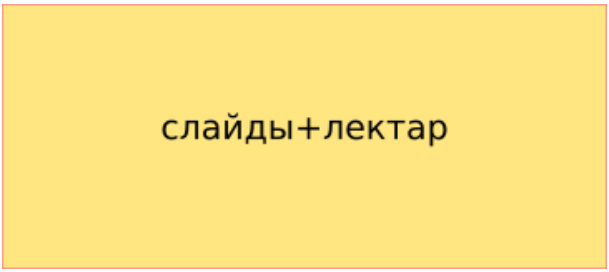
This thesis contains some background ideas for my workshop. Workshop will demonstrates one of the ways of video processing some content from presentations, lectures and so one using Kdenlive, the open-source multitrack video editor.

Прадмова

Тэзісы ніжэй фактычна фіксуюць некалькі думак і назіранняў, тлумачаць чаму для мантажа відэа для лінукосвак я выбраў гэты інструмент (kdenlive) і менавіта такі фармат. Для таго каб зразумець воркшоп не абавязковыя, але могуць быць карысныя тым, хто шукае свае інструменты і падыходы для працы з гэткім кантэнтам.

З чаго пачыналася

Пачыналі мы, як і большасць, я думаю, з таго што пачалі запісваць відэа. То бок ёсць праекцыйная дошка, побач з якой умоўна кажучы лектар і гэта ўсё фіксуецца камерай [1].



На гэтым жа месцы і сустрэлі першыя граблі: калі не ставіць спецыяльнае святло, то лектар значна больш слаба падсвечваецца

*andrej@zahar.ws, http://lvee.org/ru/abstracts/234

чым экран, на які праектар паказвае слайды. І тут альбо зыркі экран у чорнай рамцы, дзе вяла варушыцца цень лектрара, альбо лектар, які стаіць побач з невыносна зыркім экранам, глядзець які немагчыма. Часцей за ўсё нешта невыноснай якасці паміж гэтымі крайнасцямі. А калі дадаць усякія іншыя эфекты такой разнасці святла ў кадры, кшталту шума і іншага, то ясна: тут не тое што лектара ўбачыць, тут хаця б самыя вялікія літаркі на слайдзе прачытаць бы.

Аднойчы Wargaming прапанаваў нам здымаць даклады на лінуксоўках сваёй (прафесійнай) тэхнікай, і каб гэты матэрыял апрацоўвалі спецыялісты. Адрозненні пачаліся ад пачатку: прафесіяналы вырашылі праблему з тым што нешта нечытаецца не толькі камерай лепшай якасці, але і правільна пастаўленым святлом. Выніковае відэа атрымалася значна лепш, усё чыталася, разгубленыя твары дакладчыкаў было добра бачна ў кадры.

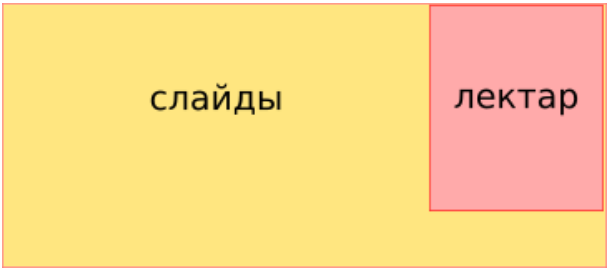
Чаму разгубленыя? А гэта другая праблема падыхода "здымаем слайды і дакладчыка адной камерай", для тых хто прайшоў першы ўзровень квеста і выставіў святло: дакладчык не бачыць аўдыторыі. Для яго няма слухачоў, ён бачыць толькі святло ў твар і цемру за лямпай.

Гэта можа не быць праблемай, калі вы здымаеце нейкую акадэмічную канферэнцыю ці паточную лекцыю, дзе узаемадзеяння паміж слухачамі і лектарам не прадугледжана. І, тым не меньш, большасць знаёмых лектараў і дакладчыкаў хочуць бачыць рэакцыю аўдыторыі на свае словы. Не кажучы пра тое, што для дакладаў у фармаце лінуксовак, калі можна задаваць пытанні практычна ў любы момант ці дапаўняць дакладчыка, гэта сапраўдная катастрофа.

Які фармат больш падыходзіць?

Адказ відавочны: слайды павінны займаць большую частку плошчы кадра і побач было б неблага каб мітусіўся лектар (для таго каб крыху больш цікава было глядзець). Слайды павінны быць у такой якасці, каб весь без выключэння тэкст чытаўся [2]. Гэта асноўнае, усялякія дадаткі пра тое што яшчэ можа быць — крыху ніжэй.

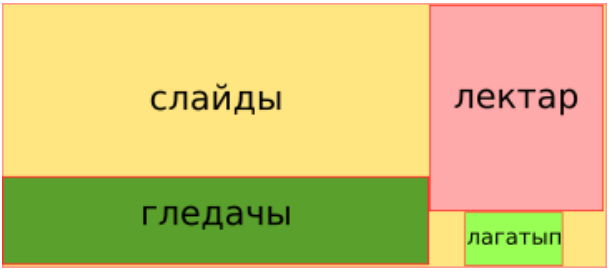
Я сустракаў такое відэа, але мне здавалася, што гэта неверагодна складана і робіцца не нейкіх пакетах, якія вывучыць за пару



вечароў неўважлівага гартання даведкі і спроб немагчыма. Трэба быць прафесіяналам і лепш яшчэ і са стажам, вышэйшай адукацыяй і заробкам, які група энтузіястаў проста не ў стане будзе сабраць.

Крокам наперад стала калі я пазнаёміўся з распрацоўкай Стафа Фаміна (Масква, РФ) якая называецца SeminarAssembler [3]. Вельмі раю азнаёміцца з відэакіраўніцтвамі і іншымі матэрыяламі па спасылцы, яны моцна прачышчаюць мозг і падштурхваюць да разумення што галоўнае.

А галоўнае, вядома ж, слайды, або дэманстрацыя чагосьці праз праектар і яны павінны займаць асноўнае месца у кадры. Неблага, калі побач са слайдамі бачна дакладчыка. Бо гэта атрымліваецца неяк больш асабіста і цікава. Дапамагае не заснуць падчас лекцыі, што ў любым навучальным працэсе безумоўны плюс. Каб дадаць інтэрактыву можна ставіць недзе побач яшчэ і аўдыторыю, каб слухач адчуваў сябе часткай групы. Зноўку ж, дапамагае засяродзіцца. Хаця і ужо зусім неабавязкова.



З неабавязковых, але прыемных рэчаў можна яшчэ успомніць, напрыклад, лагатып, які зойме пустое месца на вольнай ад сэнсу частцы кадра фінальнага відэа. І калі дакладчык не паклапаціўся

ўставіць гэта ў слайды, то небгала на пачатку даць назву даклада і імя (а можа і кантакты) дакладчыка.

Як атрымаць патрэбнае

Усё вышэйузгаданае SeminarAssembler выдатна умее, больш за тое: апрацоўка скрыптуецца, дзяленне на часткі задаецца зручна, плюс разумныя патрабванні да жалеза (яно не аджырае жалеза пад свае задачы, пакідаючы на відэаапрацоўку, што вельмі карысна), выкарыстоўвае добра вядомыя сродкі апрацоўкі, не ствараючы сваіх веласіпедаў і нават мае свабодныя зыходнікі [4], плюс дакументацыя і відэакіраўніцтва [3].

Здавалася б, вось яно: ідэальны сродак для апрацоўкі відэа існуе. Але ёсць адзін дробны мінус, які для мяне асабіста ламае ўсё гэтае шчасце: гэта праграма працуе выключна пад Windows. Ставіць віртуалку для такой прагнай да рэсурсаў задачы — глупства. Трымаць асобны кампьютэр пад адну задачу — немагчыма. Можна, канешне, зрабіць дуалбут, але: крыху шкада грошы на ліцэнзійную Windows і мне падаецца крыху дурнаватым трымаць асобную сістэму пад адну адзіную задачу, якая будзе рабіцца ў лепшым выпадку раз на месяц.

Таму я паспрабаваў некалькі розных відэарэдактараў з адкрытымі зыходнікамі (уключна, нават, з Blender), але найменьш праблем з засваеннем функцый пры магчымасці выканання ўсіх патрэбных аперацый я знайшоў толькі ў Kdenlive [5].

Для таго каб не займацца ручной сінхранізацыяй слайдаў я выправіў памылку ў прыкладзе скрыпта для захопу скрынкастаў, які прапанаваў Стас Фамін [6]. Маю версію вы можаце адшукаць на Github [7].

Калі вы патрапілі ў цяжкія ўмовы, калі трэба рабіраць PDF і самастойна сінхранізаваць кожны слайд з відэа ці аўдыёзапісам лектара (шчыра спачуваю!), то можа прыдацца яшчэ такая утыліта, як pdftoppm [8] з пакету Poppler [9] (poppler-utils у Debian). Яна найпрасцей разбірае PDF на прыстойнай якасці малюнкі прыблізна такім чынам:

```
pdftoppm -jpeg slides.pdf prefix
```

Літаратура

[1] Прыклад відэа ў старым фармаце: Майская лінуксоўка 2013 https://www.youtube.com/playlist?list=PLXNPIJ8clv69tRLxpl-CO8CFBzvpXG4lv

[2] Прыклад новага мантажу лінуксоўкі: Жнівеньская лінуксоўка 2015 https://www.youtube.com/playlist?list=PLj7ezu7K_27VZYUfkUEZLw4C0WIRU3LSr

[3] SeminarAssembler http://wiki.4intra.net/SeminarAssembler

[4] Исходники SeminarAssembler https://abf.io/belonesox/seminar-assembler/

[5] Kdenlive https://kdenlive.org/

[6] Скрыпт для запісу скрынкастаў ад Стаса Фаміна http://wiki.4intra.net/LinuxScreencasting

[7] Мая версія скрыпта https://github.com/measles/mlug_screencast_sound

[8] pdftoppm http://linux.die.net/man/1/pdftoppm

[9] Афіцыйная старонка Poppler https://poppler.freedesktop.org/

[10] fm3. http://rcr.io/words/dynamic-xterm-colors.html

Восьмая платформа BaseALT*

Михаил Шигорин, Москва, Россия

BaseALT p8 Platform has been released Summer 2016 following the three-years-old ALT Linux p7 Platform; let's look into the news of the release and the corresponding distributions with spins including starterkits, ALT Workstation, and Education.

Летом 2016 года выпущена восьмая платформа BaseALT (p8) как следующая за седьмой платформой ALT Linux (p7), опубликованной три года тому; доклад посвящён новостям в подходе к выпуску и публикации дистрибутивов и иных сборок, включая стартовые наборы, Альт Рабочая станция, комплект для образования.

Репозиторий

BaseALT p8 — новая стабильная инфраструктура репозиториев пакетов СПО, созданная и развиваемая в рамках проекта Sisyphus командой ALT Linux Team (инструментарий сам является свободным ПО). Стандартными архитектурами пакетов являются i586 и x86_64; кроме того, добавлен репозиторий для архитектуры AArch64 (64-битный ARM), а к осени планируется выпуск репозитория для аппаратной платформ «Эльбрус».

Стартовые наборы

Мы предлагаем пользователям, предпочитающим самостоятельно определять состав и оформление системы, загрузочные образы стартеркитов для архитектур i586 и x86_64. Поддерживается 13 вариантов окружений рабочего стола (Cinnamon, Enlightenment, GNOME, GNUstep, IceWM, KDE4, KDE5, LXDE, LXQt, MATE, TDE, WindowMaker, Xfce), а также 9 специализированных вариантов дистрибутива (rescue для восстановления систем, основной server, jeos с минималистичным инсталятором, builder для воспроизводимой сборки пакетов, vm-net для виртуальных машин, cloud с cloud-init для облачных виртуальных машин, ovz-generic для

*mike@altlinux.org, http://lvee.org/ru/abstracts/225

OpenVZ, server-openstack с OpenStack и server-pve с Proxmox). Среди пакетов, ориентированных на корпоративное применение, также сервер контроллера домена Active Directory (Samba-DC) и сервер групповой работы с функциональностью Microsoft Exchange (SOGo).

При этом стартеркиты являются стартовыми наборами, а не дистрибутивами в прямом смысле этого слова, т.е. не представляют из себя законченное решение.

Дистрибутивы

Дистрибутивы, существующие в рамках Восьмой платформы, частично основаны на представленных в стартовых наборах наработках и на сегодняшний день их насчитывается три: Альт Сервер, Базальт Рабочая станция и Альт Образование.

Альт Сервер 8.0 — многофункциональный дистрибутив для серверов с удобным пользовательским интерфейсом для настройки и с возможностью использования в качестве рабочей станции разработчика. Дистрибутив построен на основе графической среды MATE со стандартным набором пользвоательских приложений. Основу его функциональности составляет комплект «BaseALT-домен»: взаимосвязанные серверы LDAP, Kerberos, DNS, Samba, DHCP, Postfix, Dovecot, сервер сетевой загрузки, сервер обновлений. Предусмотрена возможность развернуть только определённые службы или использовать их отдельно, без Alterator.

Базальт Рабочая станция 8.0 включает ПО для типовых пользовательских задач: электронная почта, работа с документами и презентациями, прослушивание аудиофайлов и просмотр видео, работа в сети Интернет. На этапе установки пользователю предоставляется выбор разворачиваемых решений по области применения (например, виртуализация, мультимедиа и т.д.). Основу пользовательского окружения составляют рабочая среда MATE, офисный пакет LibreOffice версии 5.1 и браузер Firefox.

Альт Образование 8.0 — это простая в установке система, ориентированная на учреждения общего и среднего образования. В состав входят средства управление классом, возможность stateless-использования (с восстановлением состояния рабочего места после завершения сеанса), средства централизованной аутентификации по сети (через Active Directory и LDAP/Kerberos) с графическими средствами настройки, а также стандартные средства синхро-

низации времени, управления пользователями, группами и т.д. В качестве рабочей среды предоставляется XFCE 4.12 (опционально KDE 5.7.1), также предусмотрен стандартный набор пакетов работы в сети Интернет, с документами, графикой, анимацией, для обработки звука и видео, разработки ПО. Предусмотрено ПО для образования (Moodle, РУЖЕЛЬ, Mediawiki и др.), WINE и средства антивирусной защиты.

LXD*

Denis Pynkin, Minsk, Belarus

Введение

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

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

В докладе рассказывается о настройке и использовании LXD, основываясь на опыте пакетирования проекта для ОС ALTLinux и портирования среды разработки коммерческого проекта в контейнер LXC.

Состав проекта

Проект состоит из 4 основных частей:

- LXC — ядро системы, обеспечивающее низкоуровневое управление контейнерами;
- LXD — «высокоуровневая» часть системы управления, предоставляющая единообразное API управления, а также утилиты командной строки;
- `CGManager` — демон для управления контролем групп, позволяющий создавать и использовать непривилегированные контейнеры;

*`denis_pynkin@epam.com`, `http://lvee.org/ru/abstracts/237` (онлайн доступна расширенная версия материала)

– `lxcfs` — файловая система на базе FUSE, «скрывающая» реальные подсистемы `procfs`, `sysfs` и `cgroupfs`, изолируя доступ к ним от программ, работающих внутри контейнеров.

Установка LXD

Установка LXD:

```
apt-get install lxd
```

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

Вместе с LXC будут установлены все необходимые подпроекты:
– cgmanager
– lxcfs
– LXC

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

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

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

Непривилегированный root

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

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

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

Дисковое пространство

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

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

Сеть

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

Такой мост можно как создать и сконфигурировать самостоятельно, так и положиться на встроенные демон LXD — в обоих случаях достаточно добавить несколько строчек в системный файл конфигурации.

Пользовательская конфигурация

Локальная конфигурация пользователя находится в:

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

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

Системные образы

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

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

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

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

Типы образов:

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

Конфигурация контейнера

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

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

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

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

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

Свой уласны Dropbox, з календаром, кантактамі і RSS*

Андрэй Захарэвіч, Мінск, Беларусь

This is a story of attempt to get some open source and provider independent solutions, namely ownCloud, for tasks we usually allow to perform companies and corporations like Dropbox Inc. or Google Inc.

Папярэджанне

Ад пачатку хацеў бы заўважыць, што гэта не спроба разглядзець усе магчымасці ownCloud. Гэта расказ пра прымяненне яго як інструмента для задавальнення пэўных (маіх) патрэб такім чынам, каб гэты было зручна мне.

Мэты і задачы

Пачнем, вядома ж, з вызначэння задач і таго, наколькі абраны інструмент дазваляе іх выканаць.

Файлаабменнік

Задачы:

- Магчымасць падзяліцца спасылкай на нейкі файл з кім заўгодна (з нейкім механізмам абмежавання доступа);
- Магчымасць стварыць агульны набор рэсурсаў для некалькіх карытальнікаў
- Звычайныя магчымасці па абнаўленню файлаў з адной крыніцы на ўсе месцы, дзе ляжыць яго копія
- Кліент пад Linux для сінхранізацыі ўсяго акаўнта ці асобных яго папак. Пажадана, каб дазваляў дадаваць некалькі акаўнтаў.

*andrej@zahar.ws, http://lvee.org/ru/abstracts/169

Кантакты і календар

- Цэнтралізаванае сховішча для календара і кантактаў
- З магчымасцю сінхранізацыі на прылады Android. У ідэале каб сінханізацыя адбывалася гэтак жа, як са звычайнымі правайдэрамі, то бок как не абмяжоўваць спіс праграм, якія потым з гэтымі кантактамі ці календарнымі запісамі змогуць працаваць
- Зручны (хаця б мінімальна) інтэрфейс для кампьютэра з магчымасцю аб'ядноўваць кантакты ў групы.
- Групы кантактаў павінны быць бачныя і даступныя для любых аперацый як з кампа, так і з тэлефона.
- Магчымасць імпарту кантактаў і календара ў Thunderbird
- Магчымасць экспарту у файл і пераносу на іншы сервер

RSS

- Магчымасць чытаць як з кампьютэра, так і з Android-прылад
- Магчымасць дадаваць з кампьютэра і андроід-прылад
- Магчымасць сінхранізаваць стан (прачытаны, пазначаны як цікавы) для розных крыніц
- У ідэале, магчымасць атырмаць доступ праз вэб-інтэрфейс
- Магчымасць экспарту у файл і пераносу на іншы сервер

Спасылкі

- Магчымасць чытаць як з кампьютэра, так і з Android-прылад
- Магчымасць дадаваць з кампьютэра і андроід-прылад
- Групіроўка па групах ці тэгах, даступная для ўсіх кліентаў
- Магчымасць экспарту у файл і пераносу на іншы сервер

Як атрымаць уласнае воблака ад пачатку да канца

Тут я паспрабую караценька расказаць, як можна атырмаць усё што я шукаў ва ўласным воблачным сховішчы.

Як паставіць ownCloud

Самы просты варыянт атрымання інстансу ownCloud — гэта паставіць яго, карыстаючыся парадамі з афіцыйнага кіраўніцтва да бягучай стабільнай версіі [1].

Калі караценька: узяць гатовы пакет да большасці папулярных дыстрыбутываў, ці можна нават прапісаць адпаведную крыніцы для інсталяцыі пакетаў. Другі спосаб дазваляе даволі зручна абнаўляцца. Я такім чынам пражыў ужо нават апдэйт паміж мажорнымі версіямі.

Парады па наладцы

Працягваючы слаўную традыцыю, прапаную звярнуцца да парад па інсталяцыі [1] і настройцы базы дадзеных [4].

Хаця сам я спрабаваў сёе-тое з прыведзеных парад, але вымушаны заўважыць, што на серверы з не надта вялікім аб'ёмам памяці кэш можа не паскараць працу, а замаруджваць яе, насуперак парадам з афіцыйнага кіраўніцтва. А неправільная настройка PHP-кэшавання можа зрабіць вашае воблака бясконца бяспечным (то бок, недаступным нікому).

Даволі неблага працуе выкананне задач па раскладзе з дапамогай cron-задач, хаця, нажаль, некаторыя задачы могуць павісаць. У выпадку з RSS калі працэс не здолее атрымаць абнаўленні за пэўны час і перастане працаваць не знішчыўшчы lock-файл, то абнаўленне будзе немагчымае без таго, каб гэты файл знішчыць. Гэта магчыма зрабіць рукамі, альбо натравіць асобны скрыпт, які будзе па раскладзе маніторыць стан працэсу.

Калі вы плануеце карыстацца вэб-інтэрфейсам каб дадаваць файлы ў воблака, то варта будзе яшчэ выканаць дадатковыя настройкі для гэтага [6]. Паколькі мне дастаткова якіх 150–250 мегабайтаў, то хапіла проста настройкі адпаведных зменных у настройках PHP.

Што сапраўды варта зрабіць, гэта настройка бяспекі. Зноўку ж, варта звярнуцца да адпаведнага раздзелу афіцыйнага кіраўніцтва [3]. Вельмі раю адключыць доступ не па HTTPS і выканаць іншыя парады. З дапамогай гэтых мер я змог падняць сайт па рэйтынгу SSL Server Test [5] ад T(\C) да T(A).

Але з любымі мерамі бяспекі варта помніць: калі вы нешта публікуеце ў інтэрнэце, то даць стопрацэнтную гарантыю таго, што

інфармацыя будзе даступная толькі і выключна вам ужо немагчыма. Але не варта панікаваць.

Перанос дадзеных ва ўласнае воблака

Прасцей за ўсё з файламі: перанос можна пачаць адразу ж, выкарыстоўваючы вэб-інтэрфейс. Ці дэсктопны кліент [7]. Калі вы вышкталцоны аматар рэдкіх вычварэнняў і надзвычайных прыгод, то можна скарыстацца і мабільным кліентам [8]. Уласна, карыстацца можна з дапамогай таго ж заапарку. Дэсктопныя кліенты з нядаўніх часоў дазваляюць сінхранізацыю з некалькімі акаўнтамі.

Кантакты я пераносіў шляхом экспарту з Google Contacts і паўторным імпартам ужо ў ownCloud. Калі вы ўсё яшчэ карыстаеццеся кірыліцай і іншым састарэлым юнікодам па-за межамі ASCII, то я б параіў выбіраць фармат vCard, інакш будуць праблемы з кадзіроўкаю. Ну і каб закрыць пытанні з кадзіроўкаю: выкарыстанне знакаў у імёнах груп кантактаў недапушчальнае — у выніку дзе-небудзь пасля сінхранізацыі імёны будуць ламацца. Давялоса міграваць групу «Сям'я» у групу «Сваякі».

Калі вы — як і я — прасцей запамінаеце твары асоб чым імёны, то для вас благая навіна: выявы для кантактаў вы страціце. А вось для аматараў усё акуратна раскладаць па папачках і каталагізаваць у мяне добрая навіна: кантакты фактычна не належаць да нейкай групы, яны проста маюць адпаведны тэг. То бок, калі у вас сябар, з якім вы разам працавалі на нейкую кампанію, займаліся спортам і падзяляеце захапленне Linux, прычым для кожнай прыгаданай прыкметы ў вас ёсць асобная група, то гэты ваш знаёмы можа патрапіць у чатыры ці болей груп. Пацешце ўнутранага бюракрата!

На гэтым месцы лепш за ўсё настроіць двухбаковую сінхранізацыю з нейкім Android-смартфонам ці планшэтам. Добрая навіна: пачынаючы з версіі 4.0 Андроід дазваляе дадаваць правайдэры кантактаў і календара. Я спрабаваў карыстацца рознымі праграмамі, але DAVdroid – CalDAV/CardDAV Sync паказаў сябе лепш за ўсё: сіхранізацыя працуе хутка і надзейна. Дазваляе сінхранізаваць адразу кантакты і календар. Перажыў некалькі абнаўленняў як воблака, так і сваіх уласных.

З календаром было крыху больш складана: я не знайшоў спосабу штатным чынам экспартаваць яго з гуглаўскага календара. Я скарыстаўся метадам з блога Man and Keyboard [9]. Калі каранька:

1. Выдаліць непатрэбныя падзеі
2. Настроіць двухбаковую сінхранізацыю календара паміж Android-прыладай і ownCloud
3. Скрыстацца праграмай iCal Import/Export CalDAV [10] і перанесці сінхранізаваць календар паміж аблокамі

Ну і раз ужо усё гэта так удала сінхранізавалася, то можна сінхранізаваць з дэсктопам [12, 13].

RSS-чыталка пакуль мне лепш за ўсё спадабалася адна [14], якая дазваляе зручна чытаць і сінхранізаваць. Нажаль, у сэнсе кіравання падпіскамі па-за межамі дадаць новую крыніцу лепш карыстацца веб-інтэрфейсам. Але, з іншага боку, карыстацца веб-інтэрфейсам для чытання не так зручна.

Са спасылкамі прасцей за ўсё: карыстацца праз веб-інтэрфейс досыць зручна. Ёсць неблагі кліент Андроід [15]. Нажаль, кліент не дазваляе дадаваць групы для спасылак і рэдагаваць спасылкі. Ва ўсім астатнім кліент досыць зручны (і невялікі).

Усе функцыі былі рэалізаваныя праз штатныя плагіны.

Літаратура

[1] ownCloud 9.0 Server Administration Manual: Installation. https://doc.owncloud.org/server/8.2/admin_manual/installation/index.html

[2] ownCloud Server Administration Manual:Server Tuning & Performance Tips. https://doc.owncloud.org/server/8.2/admin_manual/configuration_server/performance_tuning.html

[3] ownCloud Server Administration Manual: Hardening and Security Guidance. https://doc.owncloud.org/server/8.2/admin_manual/configuration_server/harden_server.html?highlight=security

[4] ownCloud Server Administration Manual: Database Configuration. https://doc.owncloud.org/server/8.2/admin_manual/configuration_database/linux_database_configuration.html

[5] Qualys SSL Labs SSL Server Test. https://www.ssllabs.com/ssltest/

[6] ownCloud Server Administration Manual: Uploading big files. https://doc.owncloud.org/server/8.2/admin_manual/configuration_files/big_file_upload_configuration.html

[7] ownCloud Desktop Clients. https://owncloud.com/products/desktop-clients/
[8] ownCloud client for Android. https://play.google.com/store/apps/details?id=com.owncloud.android
[9] Man and Keyboard: Google to Owncloud, Contacts and Calendar. http://manandkeyboard.tk/2015/02/26/google-to-owncloud-contacts-and-calendar/
[10] iCal Import/Export CalDAV. https://play.google.com/store/apps/details?id=tk.drlue.icalimportexport
[11] DAVdroid – CalDAV/CardDAV Sync. https://play.google.com/store/apps/details?id=at.bitfire.davdroid
[12] Moving your Contacts and Calendar Away from Google. http://flailingmonkey.com/moving-contacts-calendar-google/
[13] Birthday Calendar with ownCloud via CalDAV. http://blog.mehl.mx/2014/birthday-calendar-with-owncloud-via-caldav/
[14] ownCloud News Reader. https://play.google.com/store/apps/details?id=de.luhmer.owncloudnewsreader
[15] ownCloud Bookmarks. https://play.google.com/store/apps/details?id=cz.nethar.owncloudbookmarks

Обзор решений на рынке открытого ПО для создания СХД*

Александр Клыга, Минск, Беларусь

The survey of possible data storage solutions based on Open Source programs is presented. We focus on the ZFS file system and available solutions for data storage along with the direct-attached storage model. In addition, solutions to form data storage by using network-attached storage and storage area networks are outlined. Eventually, a review of cluster data storage on the basis of the storage object files system is considered.

Создание эффективного хранилища данных — одна из сложных задач для системного архитектора и системного администратора, которому предстоит им управлять, и поэтому неудивительно, что очень часто эти две разные роли объединяются в одном лице. Именно ему приходится выбирать решение, способное эффективно решать поставленные задачи, и когда ограниченность ресурсов не дает возможности приобретения дорогих решений от ведущих вендоров в производстве систем хранения данных, выходом из подобной ситуации зачастую является использование решений с открытым программным обеспечением на базе операционных систем Linux.

Операционная система Linux с ее открытым кодом в сочетании с широкими возможностями поддержи различных аппаратных платформ и программных продуктов Open Source идеально подходит для создания эффективных систем хранения данных СХД любой сложности. Ниже представлен обзор нескольких таких решений.

Первое решение — программный продукт на базе дистрибутива Debian OpenMediaVault (официальный сайт проекта `openmediavault.org`), ориентированный на создание СХД модели NAS. Включает в себя программный RAID (0, 1, 5, 6, 10, JBOD), почтовый клиент, SSH, FTP, SMB/CIFS, NFS, RSYNC, средства управления пользователями и мониторинга работы. Возможности OMV могут быть расширены плагинами, например, такими как DAAP-медиа-сервер, ISCSI, BitTorrent-клиент и другими. В настоящий момент разработка и развитие OMV версии 2 приостановлена

*alex_kls@mail.ru, http://lvee.org/ru/abstracts/181

(последняя стабильная версия 2.1), и полным ходом идет работа над новой версией OMV 3.0 с кодовым названием "Erasmus" (данная версия в качестве beta доступна для скачивания и установки). Ключевой особенностью 3-й версии OMV станет адаптация проекта под новую версию Debian 8 и расширение возможностей за счет переработки старых и выпуска новых плагинов. В качестве ключевых особенностей данного дистрибутива также можно отметить многоязычность (включая поддержку русского языка), поддержку GPT и файловых систем ext4/XFS/JFS, неплохой web-интерфейс. Руководит проектом Волкер Тейле (Volker Theile), бывший основной разработчик FreeNAS.

Следующее решение в обзоре — проект openfiler (официальный сайт проекта `www.openfiler.com`), основанный на интересном дистрибутиве rPATH Linux. В настоящий момент для скачивания доступна версия 2.99. Ключевой особенностью данного проекта является возможность создания СХД моделей NAS/SAN с поддержкой интерфейсов FC (Fibre Channel) и iSCSI большой емкости (более 60TB). Openfiler поддерживает ссоздание и обслуживание программных RAID-массивов различных уровней (0,1,5,6 и 10), позволяет создавать и управлять кластерами. Данное решение включает широкий круг возможностей для предоставления и гибкого управления сетевыми сервисами (NFS, CIFS, HTTP/DAV, FTP, rsync). Также следует отметить, что openfiler поддерживает сетевые каталоги NIS, LDAP, Active Directory (в обычном и совмещенном режиме), контроллер домена Windows NT 4 и Hesio. Подключение по web-интерфейсу осуществляется по защищенному протоколу https. Все исходные коды проекта openfiler открыты и могут быть использованы для разработки и модификации собственных проектов.

При реализации DAS (direct-attached storage) модели СХД одним из решений может быть использование функционала файловой системы ZFS (в отличие от других типов файловых систем, ZFS это 128-битная ФС). Основным преимуществом ZFS для СХД является возможность строить хранилище данных большой емкости, с высокой отказоустойчивостью. Изначально файловая система ZFS была разработана в компании Sun для операционной системы Solaris, но в настоящее время ZFS портирована на большинство *NIX-систем (FreeBSD, MacOS), а также Linux (официальный сайт проекта `zfsonlinux.org`). Большинство облачных решений используют файловую систему ZFS как базовую для СХД, а в качестве базового дистрибутива Linux 64-битный дистрибутив Debian (вер-

сия 7 или 8). Также стоит отметить наиболее популярные решения СХД на базе файловой системы ZFS: это FreeNAS, NAS4Free, ZFSGuru (построены на базе дистрибутива FreeBSD) и NexentaOS (на базе проекта OpenSolaris).

Кластерные решения СХД базируются на возможностях распределенной файловой системы для построения высокопроизводительных и отказоустойчивых сетей хранения данных; как правило, данные решения наиболее востребованы при создании облачной инфраструктуры или организации централизованного хранилища данных в организациях. GlusterFS является наиболее популярным решением распределенной файловой системы для построения кластерных СХД. Кроме того, существует еще одно очень интересное решение, Ceph – распределенная файловая система, предназначенная для построения единого пространства СХД на базе отдельных узлов или аппаратных решений (официальный сайт проекта `ceph.com`).

Решение Ceph относится к открытым распределенным файловым системам и предназначено для создания отказоустойчивого распределенного хранилища данных, работающего по протоколу TCP. Одно из базовых свойств Ceph — масштабируемость до петабайтных размеров. РФС Ceph предоставляет на выбор три различных абстракции для работы с хранилищем: объектного хранилища (RADOS Gateway), блочного устройства (RADOS Block Device) или POSIX-совместимой файловой системы (CephFS). Главным отличием Ceph является абстрагирование от уровня ядра: вся обработка ведется в пользовательском пространстве, что позволяет исключить зависимость от аппаратной платформы. Как правило, данное решение используется при создании облачного хостинга или внутреннего единого корпоративного СХД.

Как перестать страдать и начать использовать Let's Encrypt*

Aliaksandr Kharkevich, Volha Kharkevich, Gomel, Belarus

This article describes several types of SSL certificates, CA with issuing certificates for free, ACME protocol and Let's Encrypt as Certificate Authority.

Глобальное движение в сторону HTTPS

HTTPS — это обычный HTTP, работающий через шифрованные транспортные механизмы SSL и TLS. Он обеспечивает защиту от атак, основанных на прослушивании сетевого соединения.

Глобальный переход на SSL стал трендом, и разработчики двух популярных браузеров, Mozilla Firefox [1] и Chromium [2] планируют помечать http-ресурсы как небезопасные (Non-secure).

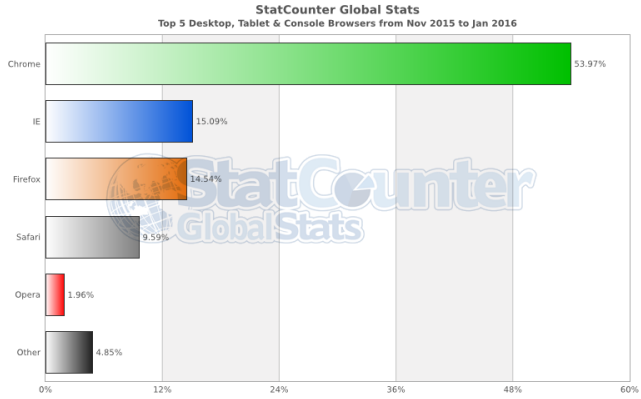
При этом в Google уже с 2014 года предпочтение отдается сайтам с https [3].

Виды SSL-сертификатов

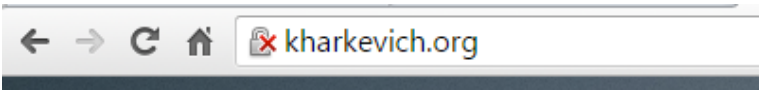
Можно выделить следующие виды сертификатов:

- Domain Validation — сертификат, подтверждающий только доменное имя.
- Organization Validation — сертификат, подтверждающий домен и организацию.
- Extended Validation — сертификат с расширенной проверкой.

*other.bigmouse@gmail.com, http://lvee.org/ru/abstracts/177



StatCounter — статистика за 2015 год



Пример небезопаснго веб-сайта

Если Organization Validation и Extended Validation продаются только за деньги, то Domain Validation бывает и **бесплатный**.

В дальнейшем будем рассматривать только сертификаты Domain Validation.

Варианты получения SSL-сертификатов

В реальной жизни для получения сертификатов существуют следующие варианты:

- Самоподписанные сертификаты (self-signed-certificate) и сертификаты, подписанные собственным центром сертификации.
  - Из плюсов — возможность самостоятельно создавать неограниченное количество SSL-сертификатов; отсутствие денежных затрат; быстрота в создании; в случае собственного удостоверяющего центра — доверие к данному сертификату.
  - Из мнусов — необходимость добавления удостоверяющего центра в список доверенных ЦС вручную либо анало-

гичной процедуры для самоподписанных сертификатов; в случае использования собственного удостоверяющего центра — дополнительные требования по размещению и защите.

- Платные сертификаты от известных центров сертификации.
  - Из их плюсов — внешний гарант подлинности.
  - Из минусов — нужно платить деньги.
- Коммерческие триальные SSL-сертификаты в качестве бесплатных.
  - Из их плюсов — бесплатность.
  - Из минусов — малый срок действия и невозможность заказать повторно.
- Бесплатные сертификаты от StartSSL.
  - Из их плюсов — бесплатность.
  - Из минусов — ручная установка сертификатов на сервер; сертификат бесплатен только для некоммерческого использования; сложности с отзывом сертификата.
- Бесплатные сертификаты от WoSign.
  - Из их плюсов — сертификат действительно бесплатный; у них появился англоязычный интерфейс; поддержка до пяти поддоменов в запросе на сертификат.

WoSign на китайском языке

  - Из минусов — ручная установка сертификатов на сервер; OCSP в Китае; для поддоменов постоянно урезаются ли-

миты; потенциальный MitM в связи с наличием The Golden Shield Project.

- Бесплатные сертификаты от Let's Encrypt.
  - Из минусов — новый игрок на рынке (требуется обновить списки CA); еще не вышел из бета-версии; только DV-сертификаты; есть некоторые лимиты на выпуск/перевыпуск сертификатов.
  - Из плюсов — нет ограничения по применению сертификатов; может использоваться в коммерческих проектах; ACME — полностью автоматический процесс выдачи сертификата; срок действия сертификата до 90 дней.

Let's Encrypt – наш выбор

Шаги по установлению глобального доверия

Для подписания пользовательских сертификатов Let's Encrypt использует промежуточные сертификаты, которые кроме корневой подписи имеют перекрестную подпись от IdenTrust. Такая конфигурация позволила как минимум взлететь на тех браузерах, в которых еще ничего не было сказано про корневой сертификат Let's Encrypt [4].

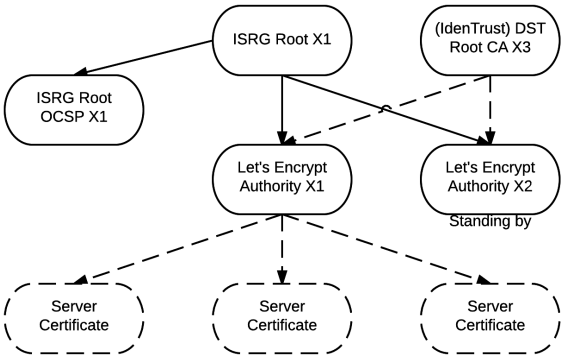
Как работает ACME (Automatic Certificate Management Environment)

Для получения сертификата необходимо доказать центру сертификации, что домен, для которого пришел запрос сертификата, принадлежит запросившей сертификат стороне.

Это происходит в несколько шагов.

В первый раз взаимодействуя с Let's Encrypt CA, агент сгенерирует пару ключей, которые будет использовать для доказательства наличия прав на доменное имя.

Агент отправляет запрос в Let's Encrypt CA с указанием домена, который необходимо подтвердить.

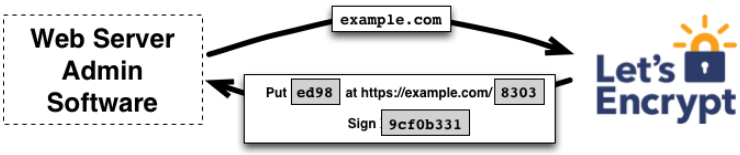


Перекрестная подпись промежуточных сертификатов

Let's Encrypt CA оценивает домен и выдает клиенту как минимум одну задачу на доказательство владения доменом:

– Создание DNS-записи в доменной зоне для запрошенного домена.
– Создание ресурса, доступного по HTTP на известном URI для запрошенного домена.

Наряду с решением вышеописанной задачи, Let's Encrypt CA создает одноразовый ключ, который агент должен подписать своим закрытым ключом для домена, чтобы доказать, что он контролирует ключевую пару.

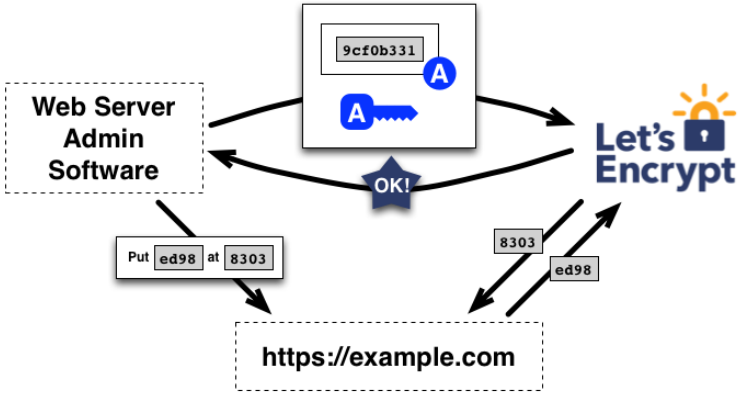


Агент выполняет выбранную задачу и уведомляет сервер о готовности пройти валидацию.

Let's Encrypt CA проверяет электронную цифровую подпись и доступность файлов или записей в DNS.

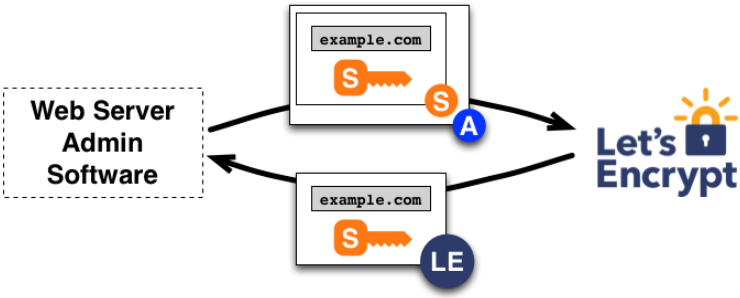
В случае, если цифровая подпись верна и выбранная задача решена верно, Let's Encrypt CA считает, что агент имеет право на управление сертификатами для запрошенного домена.

Ключевая пара, использованная агентом, становится «авторизованной парой ключей» для запрошенного домена.

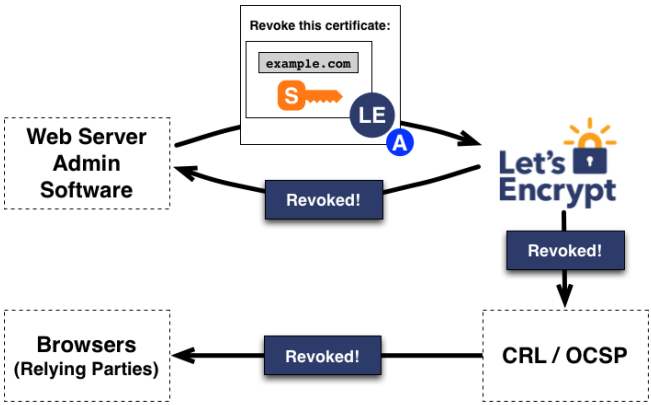


Успешное выполнение всех задач от Let's Encrypt

После авторизации агент может запросить, обновить или отозвать сертификаты для своего домена. Сообщения должны быть подписаны авторизованной парой ключей.



Выпуск сертификата



Отзыв сертификата

Клиенты/плагины для Let's Encrypt

Для Let's Encrypt существует как официальная реализация клиентской части, так и множество других реализаций протокола ACME (написаны на Python, Go, C#, Ruby, Bash).

Официальный клиент поддерживает расширение функциональности за счет плагинов. На текущий момент, реализованные плагины представлены на странице github [5].

Написать свой плагин достаточно просто: нужно всег лишь прочитать документацию [6] и написать его.

Примеры из реальной жизни и немного статистики

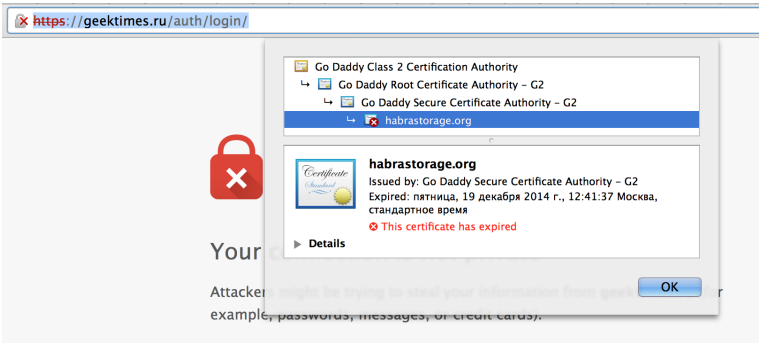
Судя по crt.sh — Let's Encrypt выпустил громадное количество сертификатов и с отрывом лидирует над остальными.

Забытые обновления сертификатов

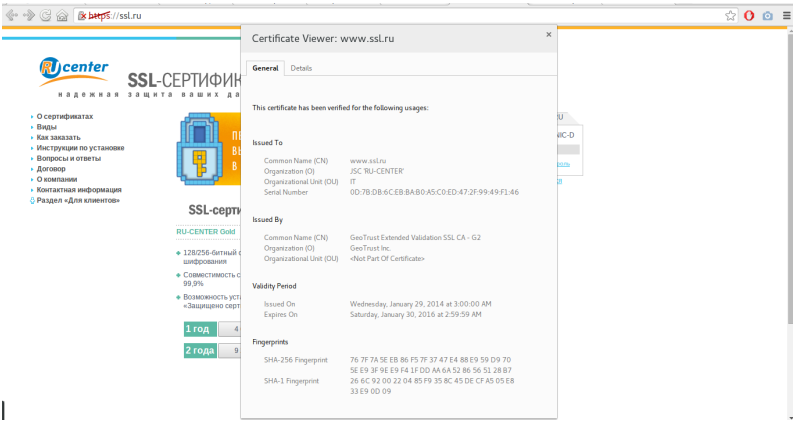
В заключение — несколько случаев пропущенных обновлений сертификатов:

– Ростелеком в 2010 году [7]

– Хабр и habrastorage.org в 2014 году [8]



– RU-CENTER (ssl.ru) в 2016 году



Литература

[1] Mozilla Security Blog: Deprecating Non-Secure HTTP. https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/

[2] The Chromium Projects proposal: Marking HTTP As Non-Secure. https://www.chromium.org/Home/chromium-security/marking-http-as-non-secure
[3] Google Webmaster Central Blog: HTTPS as a ranking signal. https://googlewebmastercentral.blogspot.com.by/2014/08/https-as-ranking-signal.html
[4] Let's Encrypt certificates. https://letsencrypt.org/certificates/
[5] Letsencrypt officially supported plugins. https://github.com/letsencrypt/letsencrypt/wiki/Plugins
[6] Writing your own letsencrypt plugin. https://letsencrypt.readthedocs.org/en/latest/contributing.html#dev-plugin
[7] Geektimes: Ростелеком забыл обновить ssl-сертификат gosuslugi.ru. https://geektimes.ru/post/102984/
[8] Geektimes: Пора обновить сертификат на Habrastorage.org. https://geektimes.ru/post/243231/
[9] Поиск по сертификатам. https://crt.sh/
[10] Тестирование конфигурации SSL. https://www.ssllabs.com/ssltest/
[11] Генератор SSL конфигурационных файлов от Mozilla. https://mozilla.github.io/server-side-tls/ssl-config-generator/
[12] Анализ веб-трафика. http://gs.statcounter.com/
[13] Let's Encrypt how it works. https://letsencrypt.org/howitworks/technology/

Comparative Review of FLOSS Testing Frameworks for Embedded C++*

Алексей Хлебников, Oslo, Norway

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Automated tests is the best way to increase the effectiveness, efficiency and coverage of your software testing. This comparative review evaluates several C++ testing frameworks with focus on usage in modern embedded systems, like Android and iOS.

Requirements

It turns out that most automated testing frameworks for C++ were designed for Desktop, as opposed to Embedded devices. They usually report test results to standard output, many frameworks even supply main() function. On Embedded platforms we usually don't have stdout and applications can not use main() function, generated by the frameworks. We need to capture testing report into memory and then either output it to the log, or to some nice window on the device.

Thus, our requirements for testing frameworks are as follows:

- Cross-platform, i.e. support for testing on Android, iOS, Linux, MacOS X, Windows.
- Support for custom test runner, because we don't have main() on Android or iOS.
- Support for custom outputter, because we don't have stdout on Android and iOS.
- Support for fixtures, i.e. setup/teardown.
- Support for testsuites, i.e. test grouping.

Non-mandatory, but desired features:

- Easy and pleasant to use: Sensible and logical design, good documentation, sensible syntax, terminology and keywords.

*alexei.khlebnikov@gmail.com, http://lvee.org/ru/abstracts/171

– The framework should be supported and mature.
– The less boilerplating — the better. For example, automatic test registration.
– Support for running test selectively, preferably by regex match on test names.
– Support for listing available tests.

Testing framework list

The following testing frameworks were evaluated:

– Bandit
– Boost.Test
– CATCH
– CppUnit
– CxxTest
– Google Test
– Igloo
– Lest
– TUT
– UnitTest++

More about each framework

CppUnit

C++'s classics of the classics. The first xUnit-like testing framework for C++. Powerful and feature-rich. Has good documentation. Unfortunately, test registration is not automatic and developer will need to retype test class and function names several times. xUnit-like frameworks in other languages, like Java and Ruby, do not require such manual test registration, because they, unlike C++, have reflection mechanisms, which are used for automatic test discovery.

Code example:

```
class MyTestSuite : public CppUnit::TestFixture
{
public:
    void setUp()
    {
        m_num = 2;
    }
```

```
    void tearDown()
    {
        m_num = 0;
    }

    void testOneThing()
    {
        CPPUNIT_ASSERT(m_num == 2);
    }

    void testAnotherThing()
    {
        CPPUNIT_ASSERT_EQUAL(m_num * m_num, 4);
    }
    ...
};

...

auto* suite = new CppUnit::TestSuite("MyTestSuite");

suite->addTest(
    new CppUnit::TestCaller <MyTestSuite> (
        "testOneThing",
        &MyTestSuite::testOneThing
    )
);

suite->addTest(
    new CppUnit::TestCaller <MyTestSuite> (
        "testAnotherThing",
        &MyTestSuite::testAnotherThing
    )
);

CppUnit::TextUi::TestRunner runner;

runner.addTest(suite);
```

Supports:

- Custom runner.
- Custom outputter by subclassing class TextOutputter and overriding 1 function.
- Fixtures and testsuites.
- Listing available tests.
- Selective running of one particular test.
- Unlike most other frameworks, defining and registering test both using C++-only code and using macros.

Downsides:

- Test autoregistration is almost non-existing. There are helper macros, but they do not help enough.
- This framework requires much boilerplating. People on the Internet complain about it a lot.
- The framework is supported, but not very actively. Currently it is supported by LibreOffice team. On the other hand, the framework already has so many features, that it does not need much further development. Sure, there is a room for improvement with boilerplating, but such work suggests so much refactoring that it is easier to just take another testing framework.

Verdict:

CppUnit requires too much boilerplating. It was probably OK 15 years ago, when C++ developers did not have much choice, but for 2016 it is too bad.

Google Test

Powerful framework with lots of features and complicated syntax. Has good documentation. Unfortunately, it is hard to redirect output. It is one of the first testing frameworks, featuring automatic test registration. C++ still does not have reflection, but Google Test overcomes this obstacle by using macros, that both define and register tests.

Supports:

- Custom runner.
- Fixtures and testsuites.
- Test autoregistration.
- Listing available tests.
- Running subset of tests, including and excluding them by path-like wildcards.

Downsides:

– Custom outputter is not supported. The framework uses C file descriptors for output. The best that can be done is redirecting output to the file.

Verdict:

Google Test is not good enough for us, because custom outputter is not supported.

Boost.Test

Powerful framework with lots of features and complicated syntax. Has good documentation. Did not support autoregistration before, but supports now. Can be compiled as static or dynamic library or used as header-only library. People on the Internet report that in case of header-only library compilation takes quite long time. Typical for a Boost library.

Code example:

```
struct MyFixtureStructure
{
    MyFixtureStructure()  { m_num = 2; }
    ~MyFixtureStructure() { m_num = 0; }
    ...
};

BOOST_FIXTURE_TEST_SUITE( MyTestSuite, MyFixtureStructure )

    BOOST_AUTO_TEST_CASE( test_one_thing )
    {
        BOOST_REQUIRE(m_num == 2);
    }

    BOOST_AUTO_TEST_CASE( test_another_thing )
    {
        BOOST_CHECK_EQUAL(m_num * m_num, 4);
    }

BOOST_AUTO_TEST_SUITE_END()
```

Supports:

– Custom runner.
– Custom outputter by subclassing std::ostream.
– Fixtures and testsuites.
– Test autoregistration.
– Listing available tests.
– Running subset of tests, selecting by path-like wildcards and tags.

Verdict:
Good candidate, supports all our requirements. Can we find better?

CxxTest

Lightweight framework with good design and syntax. Implements automatic testcase registration by running a Python script, instead of clumsy macros, used by other testing frameworks. Each testsuite is a class with (optional) setUp/tearDown functions, each test is a function starting with «test». As a result, a testsuite looks like a nice C++ class. The framework has good, even though not very long, documentation and easily readable source code otherwise.

Code example:

```
class MyTestSuite : public CxxTest::TestSuite
{
public:
    void setUp()
    {
        m_num = 2;
    }

    void tearDown()
    {
        m_num = 0;
    }

    void testOneThing()
    {
        TS_ASSERT(m_num == 2);
    }

    void testAnotherThing()
    {
```

```
        TS_ASSERT_EQUALS(m_num * m_num, 4);
    }
    ...
};
```

Supports:

- Custom runner.
- Custom outputter by subclassing class OutputStream and overriding 3 functions.
- Fixtures and testsuites.
- Test autoregistration without macros.
- Listing available tests.
- Selective running of one particular test or testsuite.

Downsides:

- Introduces dependency on Python for a C++ project.
- The Python script, used for testing code processing, has simplified C++ parser. Thus testing code must be kept parser-friendly. Fortunately, it is not hard.

Verdict:

Good candidate, supports all our requirements. Avoids macros, code looks cleaner. Can we still find better?

CATCH

New-generation testing framework, supporting both usual xUnit-style tests and TDD/BDD-style tests with SECTIONS and SCENARIOS. SECTIONS is a killer feature, allowing to save a lot of code on fixtures. SCENARIOS is development of the SECTIONS idea, more descriptive, but requires more typing.

Code example:

```
TEST_CASE( "My test suite name", "[my_tag]" )
{
    num = 2;

    SECTION( "increment" )
    {
        num++;
        REQUIRE(num == 3);
    }
```

```
    SECTION( "decrement" )
    {
        num--;

        SECTION( "increment after decrement" )
        {
            num++;
            REQUIRE(num == 2);
        }

        SECTION( "2 decrements" )
        {
            num--;
            REQUIRE(num == 0);
        }
    }
}
```

Using SECTIONS, it is possible to combine fixture code with testing code. The above code is equivalent to 3 tests in xUnit model:

```
TEST_CASE( "increment" )
{
    num = 2;
    num++;
    REQUIRE(num == 3);
}

TEST_CASE( "increment after decrement" )
{
    num = 2;
    num--;
    num++;
    REQUIRE(num == 2);
}

TEST_CASE( "2 decrements" )
{
    num = 2;
    num--;
```

```
    num--;
    REQUIRE(num == 0);
}
```

As we can see, SECTIONS provide a way to compactly describe several tests in a tree-like manner.

Supports:

- Custom runner.
- Custom outputter by subclassing std::ostream.
- Fixtures and testsuites.
- SECTIONS and SCENARIOS!
- Test autoregistration.
- Listing available tests.
- Running subset of tests, selecting by path-like wildcards and tags.

Verdict:

Surprisingly good young contender. Supports all our requirements and in addition has SECTIONS killer feature. We have a winner!

Evaluation of other testing frameworks follows for completeness of the review.

Lest

Aims to be C++11-fied version of CATCH. Seems to be less powerful than CATCH so far, and less mature. The first commit on GitHub for lest was in June 2013, vs November 2010 for CATCH. According to lest homepage, lest takes much more time to compile, probably because of excessive use of C++11 features.

Igloo

BDD-style framework with unclear syntax and bad documentation. Seems to be inactively supported: the last commit on GitHub is from July 2015, but the last release was in 2013. Seems like the author devotes his attention to his another testing framework, Igloo.

Bandit

Another BDD-style framework with unclear syntax and bad documentation. C++11-fied version of Igloo framework from the same author. C++11 syntax was supposed to make the syntax better, but, I believe, the opposite happened: Bandit syntax is even worse than Igloo

syntax, despite that the author calls it «Human friendly unit testing for C++11».

TUT

Old framework with awful syntax, relying on C++ templates instead of C++ macros. Unsupported: the last release was in 2013, commit rate is approximately 2 commits per year.

UnitTest++

Minimalistic framework with «usual» syntax with C++ macros TEST/SUITE/CHECK/etc. Documentation is quite brief. The framework seems to be supported, though not much development is being done, probably because the intention is to keep the framework minimalistic. The last release and the last commit was in November 2015.

Supports:

- Custom runner.
- Custom outputter by subclassing class TestReporter and overriding 4 functions.
- Fixtures and testsuites.
- Test autoregistration.

Downsides:

- No support for listing available tests.
- No support for selective test running.

Verdict:

Good choice, if you want very light-weight and minimalistic framework. If you want more features — choose something else.

Conclusion

Considering upsides and downsides of different testing frameworks, I am giving top 3 places to these frameworks that satisfy all our requirements:

1. CATCH, for supporting SECTIONS.
2. CxxTest, for avoiding clumsy macros, easy implementation of custom outputter and generally good design.
3. Boost.Test, for being feature-rich, mature and quality product.

OpenBSD изнутри*

Vadim Zhukov, Svetlana Savina, Moscow, Russia

The article would bring some light to the dark side of OpenBSD development process. You will know better: 1) about OpenBSD developers hierarchy; 2) what formal and informal rules are used in development process; 3) how OpenBSD development is sponsored nowadays; 4) what OpenBSD team building events AKA hackathons do look like; 5) how Ted Unangst became a verb; 6) and, finally, why do OpenBSD developers hate Australia.

Проект OpenBSD отличается от многих других проектов, занимающихся разработкой операционных систем. Здесь нет демократии, ни «по западному», ни по какому-либо ещё типу. Не существует ни одной организации, которая могла бы с полным правом заявить, что контролирует проект, полностью или частично. Даже OpenBSD Foundation, будучи основанным несколькими членами сообщества, формально дистанцирован от проекта. Во многом такое положение дел является результатом основателя и главного дирижёра проекта — Тео де Раадта, человека крайне принципиального.

Разработку в OpenBSD можно условно разделить на две части: базовая система (включая xenocara, адаптированная для OpenBSD сборка X Window System) и порты. Разделение это во многом связано с разницей в объёмах и темпах обновления «обслуживаемого» исходного текста: в то время как базовая система насчитывает порядка 25 миллионов строк кода, в портах счёт идёт на сотни миллионов. При этом скорость выхода релизов у многих портов заметно больше, чем у OpenBSD; особенно это заметно на портах, находящихся на острие web-технологий: Web-движки и построенные на их базе браузеры. Как результат, имеются две основные чат-комнаты: «для всех» и, отдельная, для работающих над портами. Как правило, вторые редко пишут в первой, и наоборот.

В чатах общаются практически исключительно разработчики, посторонних там нет. Это не от стремления скрыть что-то важное и страшное от мира, а, скорее, наоборот: в чатах общение часто весьма неформальное, часто идёт обмен довольно личной информацией, выносить которую в списки рассылки не хочется.

*zhuk@openbsd.org, http://lvee.org/ru/abstracts/170

О списках рассылки. Их на данный момент можно пересчитать по пальцам одной руки:

- `tech@` — для технических обсуждений: как правило, именно здесь предлагаются патчи и происходит их ревью.
- `misc@` — для обсуждения всего подряд, включая жалобы на баги.
- `announce@` — важные анонсы происходят через этот адрес.
- а также список рассылка под кодовым названием «`h`», предназначенный для внутреннего пользования; причины закрытости те же, что и выше. Впрочем, иногда на нём также происходит ревью патчей, когда оным требуется особое внимание: письма с этого списка рассылки считаются наиболее приоритетными.

Существует также несколько узкоспециализированных, зачастую временных, ящиков для общения отдельных небольших команд «по интересам». Часть из них — публичные, часть — нет.

Сам процесс разработки подразумевает ревью практически всего вносимого кода. Если попытаться формализовать список исключений, он окажется составлен примерно таким образом:

- Обновление портов, производимое его мейнтейнером, или по его просьбе.
- Внесение изменений в userland-код их (со-)автором.
- Внесение изменений в ещё не стабилизированный, недавно внесённый код, производимое его (со-)автором до подключения данного кода в сборку.

Довольно часто как в базовой системе, так и в портах производится массовый неглубокий рефакторинг. В таком случае изменения обязательно тестируются кем-либо ещё, после чего и даётся «okay». Согласия по каждому порту или каждой составляющей базовой системы при этом никто не спрашивает. Для внесения изменений в ядро и критичные системы обычно требуется минимум два «okay»; исключение составляют малофункциональные коммиты вроде добавления идентификаторов оборудования в таблицы PCI- и USB-устройств.

Стоит отметить, что web-сайт OpenBSD практически полностью находится в CVS-репозитории, а изменения обычно не требуют «okay».

Как показывает опыт, в ходе разработки намного больше шансов имеет получить коммит, больше удаляющий, чем добавляющий.

В общем-то, это почти все правила. В целом процесс разработки больше завязан на людях, чем на правилах. Отчасти странно, что сообщество на редкость доброжелательных и полных энтузиазма людей заслужило славу агрессивно-враждебного. По всей видимости, виной тому сравнительно высокий уровень требований к желающим стать частью сообщества: чтобы стать разработчиком нужно «всего лишь» замучать разработчиков достаточно качественной работой, чтобы проще оказалось дать права на коммит, чем самому заниматься интеграцией.

Попасть в число разработчиков OpenBSD можно только по приглашению. Сначала приглашают пообщаться, и, если не поступает заметных возражений, приглашающий радует приглашённого предложением пообщаться с Тео. После этого обычно следует также приглашение посетить один из ближайших хакатонов.

Хакатоны — важная составляющая процесса разработки. В год проходит несколько хакатонов, в разных уголках света: это не только возможность встретиться и вживую решить многие накопившиеся вопросы, но ещё и познакомиться с другой страной и другими людьми. В частности, ежегодно проходит большой хакатон, где стараются собраться все разработчики (несколько десятков). В последние годы место большого хакатона стараются по разные стороны Атлантики: чётные в Европе, нечётные — в Америке.

Традиционно хакатон проходит около недели. Организаторы берут на себя помещение и напитки. Размещение и отдельные работы, если требуется, в последние годы оплачивает OpenBSD Foundation. Билеты же участники покупают сами. Так как даты хакатонов планируются сильно заранее (порой чуть ли не за год), то достать сравнительно недорогие билеты при желании труда не составляет.

Помимо хакатонов, OpenBSD Foundation оплачивает счета, связанные с поддержкой инфраструктуры проекта, помогает приобретать оборудование отдельным разработчикам, а также в течение последних нескольких лет курирует проекты в рамках программы Google Summer of Code.

OpenBSD 5.8 & 5.9*

Vadim Zhukov, Moscow, Russia

At the time of LVEE Winter, the OpenBSD repositories will be in pre-release lock, so it's a perfect time to take a deep look at changes happened. In particular: a) pledge(2): simple tool for improving security of your own programs; b) cloud-ready OpenBSD: native OpenBSD hypervisor, Xen support, improvements in service management; c) new oldies: file(1), doas(1); d) UTF-8 support for everybody.

В последний и грядущий релизы OpenBSD (5.8 и 5.9, соответственно) вошло немало изменений. Не пытаясь перечислить их все, остановимся на самых любопытных и неоднозначных.

pledge(2)

Новый защитный фреймворк. Изначально анонсирован как tame(2). Идея ограничения системных вызовов, используемых приложением, не нова. В том же OpenBSD уже давно имеется systrace, а в ядре Linux имеется SELinux, разработанный когда-то АНБ (строго говоря, в нём больше акцент на файловые дескрипторы, но всё же). Главное отличие pledge(2) — семантический подход.

В pledge(2) системные вызовы сгруппированы в соответствии с типовыми профилями их использования. Более того, контроль в отдельных случаях ведётся на уровне аргументов системных вызовов: например, область «unix» разрешает работу с сокетами, но только в домене AF_UNIX (они же AF_LOCAL). Попытка создать или подключиться к сокету, скажем, по IPv4 приведёт к немедленной, неконтролируемой смерти программы.

Вызывать pledge(2) можно несколько раз во время выполнения программы, постепенно уменьшая число требуемых привилегий. Собственно, идея pledge(2) во многом опирается на двухэтапную организацию процесса работы программы: сначала инициализация (возможно, требующая расширенных прав), а после — основной рабочий цикл программы, в котором программа занимается в основном обработкой данных и, скажем, не открывает новые сокеты.

*zhuk@openbsd.org, http://lvee.org/ru/abstracts/175

Такой подход, конечно, конфликтует с ПО, которое позволяет переконфигурировать себя в ходе работы и не использует при этом разделение привилегий (priviledge separation). Однако стоит отметить, что за счёт такой «гибкости» такое ПО одновременно становится куда более уязвимым.

На данный момент под контроль pledge(2) переведена большая часть базовой системы OpenBSD (порядка 90% всех приложений). Также поддержка pledge(2) постепенно добавляется в стороннее ПО, например: p7zip, mutt, Chromium/Iridium... Сверх того, для данного API уже реализованы обвязки под ряд языков программирования, от Python до Haskell.

Поддержка хост-режима виртуализации и Xen

OpenBSD уже несколько лет является добротной платформой для облачной инфраструктуры, но только в качестве гостевой системы. Хост-режим до недавнего времени был доступен только на платформе sparc64.

По ряду причин Reyk Floeter сотоварищи решили реализовывать гипервизор самостоятельно. На данный момент он умеет грузить только OpenBSD, поддержку других ОС стоит ожидать после релиза 5.9. Гипервизор не планируется настолько же богатым по возможностям, как, скажем, у VMWare. Конфигурационный файл vmd(8) легко читаем и схож по синтаксису с другими разработками OpenBSD, вроде httpd(8).

Параллельно с этим Михаил Белопухов реализовал поддержку паравиртуализации Xen. Благодаря проделанной им работе OpenBSD теперь может полноценно взаимодействовать с Xen-хостом, включая корректную инициализацию виртуальных устройств.

Для поддержки различных гипервизоров ещё в OpenBSD 5.8 была добавлена специальная шина драйверов, pvbus(4). На данный момент она обеспечивает поддержку следующих гипервизоров: Hyper-V, KVM, VMware, Xen, а также, конечно, вышеупомянутого родного гипервизора.

file(1) и doas(1)

Два сравнительно небольших изменения, отлично иллюстрирующих вектор развития OpenBSD.

– file(1) и связанная с ней libmagic используются довольно часто. При этом и в описании форматов, и в самом коде file(1) нередко находят ошибки, уже не раз приводившие к опасным уязвимостям. При этом самой утилите, несмотря на сложную внутренню логику, практически не требуется какой-либо доступ к окружающей системе, а фактически используемый набор функций и запрашиваемых сведений о файлах весьма мал... В итоге file(1) была переписана Nicholas Marriott (к слову, автором tmux) с использованием техники разделения привилегий: фактический разбор входного потока происходит в предельно ограниченном процессе, который от получает от мастер-процесса файловые дескрипторы для анализа.
– doas(1) — по сути, сильно упрощённая альтернатива sudo(8) с чуть более удобным синтаксисом. Общий код умещается в 822 строчки, включая комментарии и заголовочный файл. Для сравнения, в sudo 1.8.15 содержится 83596 строк. Из заметных «потерь» — sudoedit и явный проброс переменных окружения в командной строке.

UTF-8 для всех и каждого

Свершилось то, чего давно ждали: OpenBSD уходит от однобайтных кодировок в сторону UTF-8. Хотя данный переход считают прозрачным, на самом деле существует целая россыпь подводных камней, прежде всего, в работе с терминальными приложениями: начиная от расчёта ширины строки и заканчивая, как ни удивительно, вопросами безопасности: если программа пытается честно, но не слишком внимательно, интерпретировать UTF-8, злонамеренный пользователь может заставить её исказить обрабатываемые данные или их отображение для других пользователей.

Литература

[1] Презентация pledge. `http://www.openbsd.org/papers/hackfest2015-pledge/mgp00001.html`

[2] Отчёт о хакатоне n2k15, включая рассказ о vmd. `http://undeadly.org/cgi?action=article&sid=20151217134417`

[3] Анонс поддержки Xen. `http://undeadly.org/cgi?action=article&sid=20160114113445`

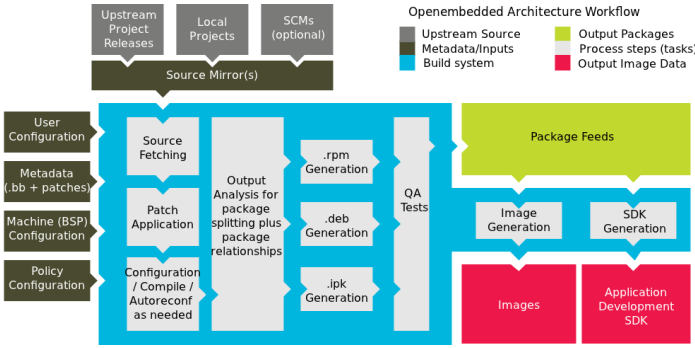
Yocto and OpenEmbedded at Collabora*

Andrew Shadura, Bratislava, Slovakia

How the use of Yocto and OpenEmbedded helps corporations migrate to free software

There's a certain confusion existing even among people closely working with Yocto Project, on what exactly Yocto is. First of all, Yocto isn't a Linux distribution. In fact, Yocto Project is an umbrella organisation that takes care of a bunch of embedded Linux technologies, including OpenEmbedded Core, BitBake, Poky and others. These and others technologies Yocto Project provides allow users to build custom Linux distributions suited to their own needs.

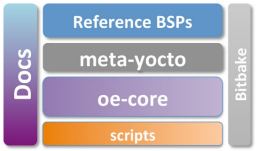
One might ask, if Yocto isn't a distribution, how does one make a distribution using its technologies?



The current Yocto technology stack has evolved from its roots in previously separate OpenEmbedded Project. Since the merger of OpenEmbedded and Yocto, OpenEmbedded has introduced a layers system allowing vendors and users to have their bits separate yet plugging into each other. There's a number of layers Yocto Project provides (`oe-core`, `meta-yocto`, `meta-yocto-bsp`) which form so-called «reference distribution», Poky. Poky contains foundation package recipes (from OpenEmbedded Core), distribution policy

*`andrew@shadura.me`, `http://lvee.org/ru/abstracts/182`

configuration, reference BSPs, build tools and documentation. Normally, Poky is what you start from when creating your own distribution.



Package recipes are written in a language similar to both Make and shell, and to certain extent resemble Gentoo's ebuilds. These recipes are being used by BitBake to build binary packages, and provide necessary information on where sources are found, and how to build them. BitBake recipes support includes, inheritance and overrides making it easy to change or extend the behaviour of an existing recipe without touching its code.

Fig: Example of a BitBake recipe:

```
SUMMARY = "The␣canonical␣example␣of␣init␣scripts"
SECTION = "base"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://${WORKDIR}/COPYRIGHT;md5=349
    c872e0066155e1818b786938876a4"

SRC_URI = "file://skeleton␣\
␣␣␣␣␣␣␣␣␣␣␣file://skeleton_test.c␣\
␣␣␣␣␣␣␣␣␣␣␣file://COPYRIGHT␣\
␣␣␣␣␣␣␣␣␣␣␣"

do_compile () {
           ${CC} ${WORKDIR}/skeleton_test.c -o ${
               WORKDIR}/skeleton-test
}

do_install () {
           install -d ${D}${sysconfdir}/init.d
           cat ${WORKDIR}/skeleton | \
             sed -e 's,/etc,${sysconfdir},g' \
                 -e 's,/usr/sbin,${sbindir},g' \
                 -e 's,/var,${localstatedir},g' \
                 -e 's,/usr/bin,${bindir},g' \
                 -e 's,/usr,${prefix},g' > ${D}${
                     sysconfdir}/init.d/skeleton
```

```
            chmod a+x ${D}${sysconfdir}/init.d/skeleton

            install -d ${D}${sbindir}
            install -m 0755 ${WORKDIR}/skeleton-test ${D
                }${sbindir}/
}

RDEPENDS_${PN} = "initscripts"
CONFFILES_${PN} += "${sysconfdir}/init.d/skeleton"
```

Fig: Example of a BitBake recipe with inheritance:

```
SUMMARY = "Example␣of␣how␣to␣build␣an␣external␣Linux␣
    kernel␣module"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12
    f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

SRC_URI = "file://Makefile␣\
␣␣␣␣␣␣␣␣␣␣␣file://hello.c␣\
␣␣␣␣␣␣␣␣␣␣␣file://COPYING␣\
␣␣␣␣␣␣␣␣␣␣"

S = "${WORKDIR}"
```

Fig: Example of .bbappend:

```
RDEPENDS_${PN}_append = "␣systemd"
```

Collabora is a company that provides consultancy to companies who are deploying open source technologies in their products, by providing its own open source based products and through knowledge sharing activities such as training. Collabora employs many free software developers who are experts or major developers in such areas as multimedia (GStreamer), graphics (Wayland, Weston), Linux kernel, productivity software (LibreOffice) and others.

At Collabora, we use Yocto on a project for a manufacturer of medical equipment, who use a Linux-based operating system in their products. Currently, their production devices are using a very custom Buildroot-based system, which runs quite an old (3.x-something) version of Linux kernel with lots of custom proprietary daemons and APIs. At some point they realised that it's quite a difficult task to support that system, and they decided they need help of experts, us.

For the project, the customer have decided to eliminate as much as possible custom proprietary libraries, take as much work as possible upstream, migrate to and integrate systemd as the init system. They also wanted us to help them with development methodology.

Thanks to the layer system Yocto Project uses, building your own distribution based on it is quite an easy job, once you know what you need from it. It all started as a layer with three files: the layer definition itself, distribution configuration file, and a machine definition.

Fig: Layer definition file (`conf/layer.conf`)

```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
 ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "distro-core"
BBFILE_PATTERN_distro-core = "^${LAYERDIR}/"
BBFILE_PRIORITY_distro-core = "10"
```

Fig: Distribution definition file (`conf/distro/badger.conf`)

```
require conf/distro/poky.conf

DISTRO = "distro-core"
DISTRO_NAME = "Core Distro Platform"
DISTRO_VERSION = "2.0"
DISTRO_CODENAME = "badger"

DISTRO_FEATURES_append = " systemd wayland xwayland
    xattr pam apparmor"
DISTRO_FEATURES_remove = "x11"

# Enable systemd as init
VIRTUAL-RUNTIME_init_manager = "systemd"

# Disable sysv init and prevent any init scripts in the
    images
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_initscripts = ""

PREFERRED_PROVIDER_jpeg = "jpeg"
PREFERRED_PROVIDER_jpeg-native = "jpeg-native"
```

Fig: Machine definition file (`conf/machine/vexpressa9.conf`)

```
#@NAME: vexpress-a9 machine
#@DESCRIPTION: Machine configuration for the vexpress
    a9 board

PREFERRED_PROVIDER_virtual/xserver = "xserver-xorg"

# Ship all kernel modules by default
MACHINE_EXTRA_RRECOMMENDS = " kernel-modules"

# Allow for MMC booting (required by the NAND-less)
EXTRA_IMAGEDEPENDS += ""

# Uncomment the following line to enable the hard
    floating point abi. Note that
# this breaks some binary libraries and 3D (neither of
    which ship with
# meta-yocto). For maximum compatibility, leave this
    disabled.
#DEFAULTTUNE ?= "cortexa8hf-neon"
include conf/machine/include/tune-cortexa9.inc

#IMAGE_CLASSES += "sdcard_image"

#IMAGE_FSTYPES += "tar.bz2 ext3 vexpressa9-sdimg"
IMAGE_FSTYPES += "tar.bz2 ext3"

#EXTRA_IMAGECMD_jffs2 = "-lnp "

# 2.6.37 and later kernels use OMAP_SERIAL, ttyO2
# earlier kernels use ttyS2
SERIAL_CONSOLE = "115200 ttyO2"

PREFERRED_PROVIDER_virtual/kernel ?= "linux-yocto"

KERNEL_IMAGETYPE = "zImage"
UBOOT_MACHINE = "ca9x4_ct_vxp_config"
UBOOT_ENTRYPOINT = "0x80008000"
UBOOT_LOADADDRESS = "0x80008000"
KERNEL_EXTRA_ARGS += "LOADADDR=${UBOOT_ENTRYPOINT}"

MACHINE_FEATURES = "kernel26 apm usbgadget usbhost vfat
    alsa"
```

These files define where recipe files are to be found, exactly what features (systemd, wayland, apparmor) we're using and what we don't (x11, sysvinit), and what processor architectures we're building for, what types of images we need to generate and so on.

For architecture-dependent parts, we first used a layer Freescale provided (meta-fsl-arm, meta-fsl-arm-extra), so we wouldn't need to write our own image generation routines, or tuning the cross-compiler features by hand. Later, we removed that dependency by bundling greatly simplified and customised versions of Freescale's recipes, but until we needed that it was a great help that we could reuse already working code.

When the initial phase of the project was completed, and we had a working image booting on a hardware prototype with Weston shell running, it was decided to split our layer in two, thus separating the platform itself and the application layer. Into the application layer went customer's proprietary software that will remain proprietary — at least, for now, and supporting daemons and libraries it depends on. The platform layer is almost entirely free software, with the exception of a few legacy hardware-related daemons which will be at some point replaced with their free software counterparts.

One might ask, why do we need anything in the platform layer apart from the distro configuration, if it's free software anyway? The answer is that, we're using quite some bleeding edge technology packages, but at the same time we want our platform to be based on the stable branch of Poky, the Yocto meta-distribution. That means, from time to time we need to import recipes for newer versions of software from the development branch. This is especially true when recipes coming from Yocto need to be improved: while BitBake allows extending existing recipes with use of .bbappend files, we mostly use that for distribution-specific things only. If we need some generic change that can be upstreamed, like user sessions support in systemd and dbus, we copy the latest version of the recipe from upstream, and patch it locally, so that the fixes can be easily submitted upstream.

This brings a benefit of needing minimal edits to the patches before they're submitted; otherwise we'd need a complete rewrite of the feature we need.

Apart from software updates and patches we also carry recipes for some free software which isn't release-ready, some custom configuration and some temporary workarounds for kernel bugs we don't currently have capacity to fix properly.

So far, since the project began, our team has contributed to the community at least the following:

- lots of fixes to various OE Core package recipes, including patches enabling systemd integration, merged /usr
- patches to Weston and Linux kernel enabling accelerated graphics on our hardware
- patches to ifupdown adding inheritance feature similar to what BitBake has
- lot more

Kernel support for the customer's hardware is all being upstreamed by the customer, as the customer believes their hardware should run mainline kernel.

Even though the project is still in progress, even at its current stage it clearly demonstrates how is possible to migrate a big project to an open platform, reducing the maintenance cost and at the same time helping everyone else, and to a great extent that is possible thanks to Yocto.

An introduction to post-quantum cryptography*

Andrew Savchenko, Moscow, Russia

These days cryptography faces a new type of threat: quantum computing. An overview of quantum computing and how it works in the context of cryptography is presented. It is discussed when it is dangerous and when it is not for commonly used algorithms. There are well known, but not commonly used algorithms, which are resilient to the quantum computing approach and free software is available to use them in a manner similar to GnuPG.

Preface

Post-quantum cryptography is a way to preserve data security in the world of quantum computers: it is a set of algorithms and protocols resilient to cryptanalysis using quantum computing [1]. It has nothing to do with quantum cryptography, which is a science of using quantum mechanical properties for cryptographic tasks (e.g. secure key distribution based on quantum entanglement or data copy protection based on wave function collapse).

Quantum computing

In classical electronics each bit can be in exactly one state: 0 or 1, e.g. byte of 8 bits can encode 256 states, but can be in only one state at once. Quantum bit (known as qubit) can also encode 0 or 1 state, but can be in superposition of both states at once, so 8 qubits describe 256 states simultaneously. Qubits can be operated by quantum gates the same way as bits are operated by microelectronics gates; quantum gates are very different in nature and properties from classical gates, but can also implement a full set of logical operations. The main power of quantum computing is that single quantum gate can operate on 2^n states for n qubits available.

Quantum computers are probabilistic by the nature: 2+2 = 4 only sometimes, it may be 5 or -10 as well, but with different probabilities,

*bircoph@gmail.com, http://lvee.org/ru/abstracts/184

so results must be verified or repeated many times to give acceptable error probability. Thus quantum computer is in no way replacement for classical computations, it is a dedicated machine that can augment classical computations, but never replace them.

Quantum algorithms

There are many quantum algorithms available, but two of them are of the most importance for crypto analysis

Shor's algorithm Shor's algorithm [2] allows for fast integer factorization, thus breaking prime multiplication and elliptic curve algorithms (RSA, ECDSA, ED25519 and so on). It is based on the idea of transforming factorization problem into function period finding problem (this is implemented using classical computing), finding period of a function using quantum Fourier transform. Then using a classical computations for continued fraction expansion prime can be obtained. Of course verification is needed.

As a result, search time is reduced from exponential to polynomial.

Grover's algorithm Grover's algorithm [3] is a "brute force" algorithm which reduces search complexity from O(N) to O(sqrt(N)), thus halving key strength, e.g. AES-256 will be reduced to AES-128.

Impact on cryptographic algorithms.

- Symmetric key cryptography
  - key strength is halved, this is a danger but solvable;
- Asymmetric key cryptography
  - RSA/DSA/El-Gamal — dead;
  - elliptic curves cryptography — dead;
  - hash-based — halve reduced;
  - code-based — halve reduced;
  - lattice-based — halve reduced;
  - and so on...

There are plenty of asymmetric key algorithms which can't be reduced to factorization problem. So why they are not used? The answer is: fast and effective implementation is also needed, but with modern CPUs it is not such a big problem.

Free software solutions

Encryption Free software implementing algorithms resistible to quantum computing exists! See codecrypt [4] (LGPL-3). Main features:

- functionality similar to GnuPG:
  - key pair generation;
  - signature;
  - asymmetric encryption;
  - symmetric encryption;
  - import/export/recipients and so on
- many algorithms are implemented

Of course it is still new and don't rely on it blindly, better combine with GnuPG or other schemes.

Upstream is very dynamic and responsive!

Programming For those interested in quantum programming, you can try QCL [5] (GPL-2): it is a quantum computing language with an emulator of a quantum computer!

References

[1] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen (editors). Post-quantum cryptography. Springer, Berlin, 2009. ISBN 978-3-540-88701-0. https://pqcrypto.org/www.springer.com/cda/content/document/cda_downloaddocument/9783540887010-c1.pdf

[2] https://en.wikipedia.org/wiki/Shor's_algorithm

[3] https://en.wikipedia.org/wiki/Grover's_algorithm

[4] http://e-x-a.org/codecrypt/

[5] http://tph.tuwien.ac.at/~{}oemer/qcl.html

Создание робота для Roborace*

Dmitry Sklipus, Brest, Belarus

Author shares his development experience of Arduino-based robots targeted at Roborace competitions. Specifics of the competition is explained as far as some details of the hardware platform and the contol algorythm.

Специфика Roborace

Roborace — это состязания, в которых соревнуются роботы-автомобили на специальной кольцевой трассе. Можно провести некоторую аналогию между Roborace и гонками Формулы 1, за исключением двух моментов.

- Во-первых, вместо полномасштабных гоночных болидов участвуют уменьшенные модели авто и оригинальные конструкции с габаритными и весовыми ограничениями (максимальные ШхД=25x50 см и вес до 3 кг).
- Во-вторых, вместо пилотов автомобилем управляет бортовой компьютер, который анализирует показания различных датчиков и ориентирует автомобиль на трассе, выбирает скорость движения, предотвращает столкновения с препятствиями и соперниками. Собственно "поведение" авто на трассе определяется управляющей программой бортового компьютера.

Roborace проводится в виде чемпионата, состоящего из этапов, которые организуются в различных городах Беларуси и за рубежом. Участие в чемпионате принимают как конструкции начального уровня (например, на базе конструктора типа LEGO), так и сложные робототехнические устройства. Регламенты соревнований формируются таким образом, чтобы охватить как можно более широкий спектр характеристик и возможностей робототехнических конструкций.

Рассмотрим трассу, изображенную на рисунке 31.1, по которой предстоит перемещаться роботу. Обязательными элементами являются черные линии и стенки. Исходя из этого можно строить стратегию движения робота по трассе: например, оборудовать робота

*sklipus@gmail.com, http://lvee.org/ru/abstracts/163

Рис. 31.1. Трасса для гонок роботов

датчиками черной линии и использовать линии трассы для навигации, или установить дальномеры для обнаружения препятствий и двигаться вдоль стенок.

В рамках данной статьи представлен один из роботов, разработанный мной для Roborace по второй стратегия (движение на основе показаний дальномеров).

Этапы создания робота

Создание робота для Roborace начинается с выбора шасси. Сейчас магазины предлагают большой выбор гусеничных и колесных платформ. Я рекомендую остановиться на классической схеме, когда задние колеса приводятся в движение электродвигателем, а передние управляются сервоприводом [1]. На рисунке 31.2 изображен робот для Roborace, построенный по подобной схеме.

Мне повезло за небольшие деньги приобрести модель в местном клубе радио-моделистов (читатели могут попытаться сделать то же самое в своем регионе: обычно у них много устаревших моделей). Так как в приобретенной модели не было тягового электродвигате-

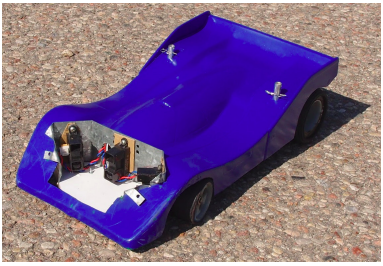


Рис. 31.2. Робот

ля, на нее был установлен установил купленный 12-вольтный мотор. Можно было также использовать обычную игрушку: они обычно довольно живучи, и требуется только модифицировать рулевое управление.

Так как в моем случае сервопривод уже был установлен, с ним проблем не возникло.

Следующий этап — выбор платы управления. Тут есть множество вариантов. Я выбрал Arduino как самый простой вариант. То же можно порекомендовать и читателю, особенно при недостатке опыта. Исходя из моего достаточно большого опыта, для таких роботов достаточно обычных 8-битных микроконтроллеров. Поэтому если не планируется использовать для отслеживания движений робота камеру, не стоит усложнять его более мощным процессором.

Сервопривод можно напрямую подключить к Arduino — например, через sensor shield, изображенный на рисунке 31.3. К нему также удобно подключать датчики.

Мотор подключить к Arduino напрямую не получится. Нужно использовать специальные Motor Driver. Сейчас из достаточно много в продаже, и есть инструкции по подключению. Я использовал Motor Driver, разработанный в нашей лаборатории (рис. 31.4).

В соревнованиях роботов приходится много внимания уделять батареям. Я использую литий-полимерные аккумуляторы. Они очень хорошо себя зарекомендовали. Один из хаков, которые я применяю в своем роботе, касается преобразователя напряжения. Штатный преобразователь в Arduino не очень хорош, поэтому для экономии энергии аккумулятора неплохо использовать Step-Down регулятор [2]. Конечно, можно использовать и обычный линейный преобразователь.

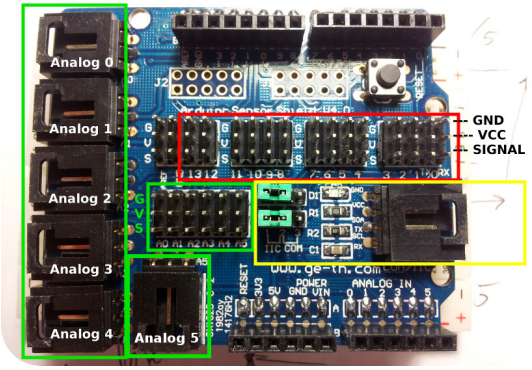


Рис. 31.3. Sensor shield v4

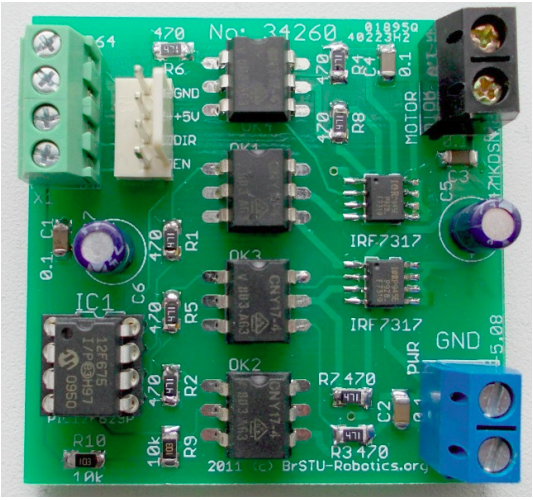


Рис. 31.4. Motor driver

Самая главная часть робота это датчики — то, что обеспечивает его информацией об окружающем мире, о препятствиях и о других роботах. В средней ценовой категории мы можем выбирать из *ультразвуковых* и *инфракрасных* датчиков. В своем роботе я использую инфракрасные датчики GP2Y0A02YK0F. Мне не нравятся

ультразвуковые датчики из-за того, что может происходить зашумление одного датчика другим. Например, у меня возникали такие ситуации: правый датчик посылал сигнал, а левый его принимал. Я все ещё работаю над правильным размещением ультразвуковых датчиков и над управлением ими. Надежду их запустить постоянно подпитывает их цена.

На представленной здесь модели робота установлено три инфракрасных датчика. Датчики можно увидеть на рисунке 31.2. Они установлены в глубь корпуса по двум причинам:

1. для уменьшения мертвой зоны датчика, которая у данной модели составляет 20 см;
2. корпус робота защищает датчики от механических повреждений во время столкновений с другими роботами.

Боковые датчики установлены под углом 45 градусов. Хорошо, если в конструкции робота предусмотрена регулировка угла их установки. Общую схему робота можно посмотреть на рисунке 31.5.

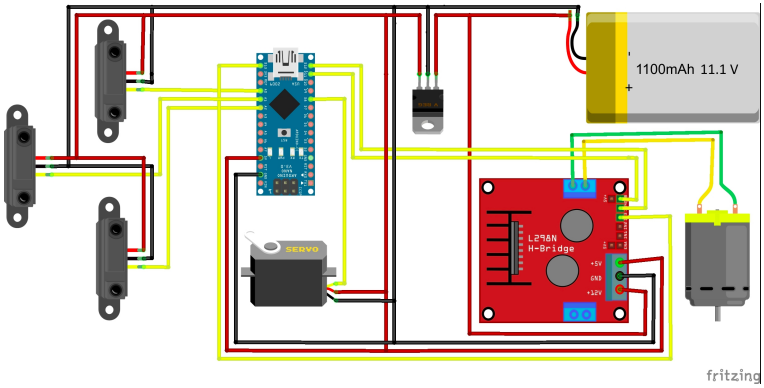


Рис. 31.5. Общая схема робота

Программирование робота

Так как на роботе используется Arduino, то программирование выполняется с использованием Arduino IDE. Программа робота представляет собой замкнутый цикл, который состоит из следующих блоков:

1. Фильтрация показаний датчиков;
2. Вычисление угла и скорости движения робота;
3. Передача управляющих сигналов на механизмы.

В данной структуре отсутствует блок получения информации от датчиков. Так как датчики возвращают аналоговый сигнал, в Arduino IDE есть функция analogWrite(). Данная функция замечательно работает, если не важна скорость измерения. Но так как робот разрабатывался для соревнований, было принято решение вынести обработку датчиков в прерывание.

Все платы Arduino, построенные на микроконтроллере ATmega, имеют возможность проводить измерения АЦП в автоматическом режиме. Нужно один раз настроить этот режим, а потом пользоваться полученными значениями. В результате контроллер постоянно проверяет датчики, не тратя на это процессорное время. Фильтрация показаний датчиков осуществляется медианным фильтром с окном в три элемента.

Для движения по трассе был выработан следующий алгоритм. Робот сравнивает расстояния до правой и левой стенки, и в соответствии с этим поворачивает колеса в нужное направление. Если впереди робота нет препятствий, скорость увеличивается, но также уменьшается максимально возможный угол поворота колес. Это нужно для того, чтобы на прямых участках робот ехал более прямо. При обнаружении препятствия угол поворота колес увеличивается, и робот притормаживает.

Есть конечно и нерешенные проблемы. Например, робот не знает кривизну поворота, поэтому тормозит перед каждым поворотом.

Посмотреть код проекта можно на GitHub [3].

Литература

[1] https://en.wikipedia.org/wiki/Servo_%28radio_control%29

[2] https://www.google.com/url?q=https%3A%2F%2Fsites.google.com%2Fsite%2Fsklipusrobotsystems%2Fhardware%2Fpower-of-robot&sa=D&sntz=1&usg=AFQjCNHdVp_jzQoLmZ8pIL3PrrfXRarkuA

[3] https://github.com/sklipus/roborace/tree/master/robots/FreshSRDEDITION

Использование архитектуры EAV в Opensource-проектах*

Виталий Сороко, Гродно, Беларусь

Entity–attribute–value model (EAV) is a very flexible data model used in various open source projects. This data model has many advantages and disadvantages so it is not suitable for wide application. But sometimes it is highly recommended to use EAV in design and development process.

Cases are decribed, in which it is preferable to use EAV data model. EAV is completely compared with the standard relational data model and some examples of successfully integration EAV in ready-to-use open source solutions are given.

Entity–attribute–value model (EAV) – это модель данных, которая благодаря использованию сущностей, атрибутов и их значений обеспечивает большую гибкость при создании новых данных. Это означает, что при создании нового типа товара в интернет магазине нет никакой необходимости создавать новые таблицы и поля в базе данных. Более того благодаря использованию этой модели данных можно с лёгкостью создать один товар из нескольких уже существующих (например «подарочный набор»). Тем не менее использование модели EAV не ограничивается только интернет магазинами: некоммерческая организация Apache Software Foundation использует её в проекте Apache UIMA, а некоторые системы баз данных, такие как InfinityDB, изначально имеют поддержку модели данных EAV.

Тем не менее у модели данных EAV, как и у всех других моделей данных, есть свои преимущества и недостатки и это в некоторых случаях ограничивает возможность использования данной модели.

Главными плюсами модели данных EAV являются:

1. Гибкая и универсальная структура данных (можно менять количество свойств без изменения полей в таблицах);
2. Относительная простота при добавлении или изменении характеристики товара;

*vitalyok@tut.by, http://lvee.org/ru/abstracts/172

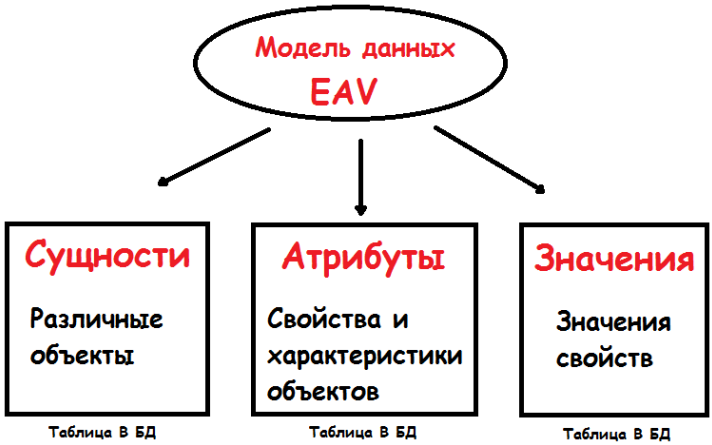


Рис. 32.1. Общая схема EAV в виде трёх таблиц

3. При правильной реализации избыточность данных почти отсутствует.

Основные минусы EAV:

1. Для вставки данных обычно используется несколько запросов;
2. Не очень быстрая выборка данных в случае реализации с большим количеством таблиц и высокая нагрузка на сервер при больших объемах данных в связи с невозможностью применения стандартных способов индексации;
3. Иногда возникают трудности в обеспечении целостности данных.

В настоящее время модель данных EAV используется в таких Opensource проектах как Magento Community Edition, EAV-Django и др. Необходимо отметить, что модель данных EAV не получила очень широкого распространения не только из-за своих технических минусов, но также в результате её неполного понимания со стороны разработчиков. Как правило только те, кто разобрался с принципом работы этой модели и попробовал её в реальной среде, могут объективно судить о её преимуществах и недостатках, а также рекомендовать её к использованию в той или иной ситуации и именно поэтому, очень важно полностью понять основные принципы организации её работы. В упрощённом виде реализация

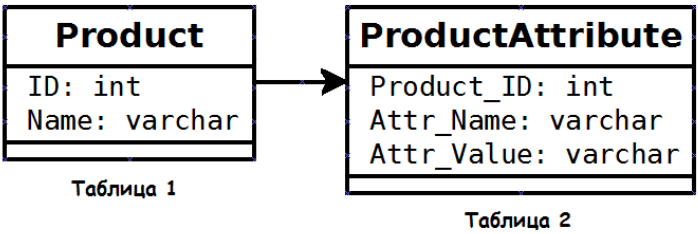


Рис. 32.2. Простейшая реализация EAV в виде двух таблиц

модели EAV может быть в виде двух или трех таблиц (см. рис. 32.1 и рис. 32.2).

Однако в реальных проектах реализация обычно сложнее, и на то есть определенные причины, такие как оптимизация скорости обработки запросов и уменьшение общего количества записей в таблицах.Так, например, в Magento CMS CE для реализации данной модели используется больше десятка таблиц. Как вы могли узнать из описание недостатков выше, иногда гибкость может трансформироваться в падение производительности всей системы. Тем не менее, в настоящее время уже разработаны различные способы решения этой проблемы. Итого, главная рекомендация по применению следующая: EAV рекомендуется использовать только исходя из плюсов, а именно в случаях если характеристики для товаров очень динамичны и часто изменяются и вы не ограничены в серверных ресурсах.

Опыт использования СПО в учебном процессе УРТК им. А.С. Попова*

Anton Uymin, Arseny Meshkov, Yekaterinburg, Russia

The article describes experience of adopting free software in a secondary vocational educational institution - a college. List of profession specialties and specialized classrooms that use free software is presented. Advantages and disadvantages of using free software in the educational process are discussed.

В настоящее время в Российском образовании складывается сложная ситуация. С одной стороны, идет активная реформа по преобразованию учебных заведений, их укрупнению и консолидации ресурсов, с другой стороны среднее профессиональное образование и высшее образование уже не дополняют друг друга, как это было ранее, а являются прямыми конкурентами на рынке образовательных услуг. При этом уровень финансирования СПО и ВПО отличается на порядки. Сегодня образовательное учреждение не может выжить, если будет предоставлять парты и доски, необходим качественно новый уровень технического и методического обеспечения. Каждый студент может получить полный объём информации в интернете, но он не может её качественно классифицировать и, зачастую, у него нет хорошей базы оборудования для изучения современных технологий. Поэтому ОУ сейчас не столько должны нести новые знания, сколько предоставлять платформу для образовательной деятельности.

«СПО для нищебродов!» — с этой фразы одного из представителей сообщества СПО в Екатеринбурге хотелось бы начать данную статью. Уральский радиотехнический колледж им. А.С. Попова находится в Екатеринбурге. У нас реализуется всего 12 специальностей. Из них 6 — по IT-направлению. До 2008 года в нашем учебном заведении из СПО были шлюз и сайт на FreeBSD. В учебном процессе СПО не использовалось никак. В 2008 году мы начали сотрудничать с отделом «К», консультировались у них, отправляли к ним на практику студентов. В этом же году начался перевод учебных материалов на Linux.

*au-mail@ya.ru, mirex08@ya.ru, http://lvee.org/ru/abstracts/173

В колледже 19 компьютерных лабораторий. Парк ПК насчитывает около 500 шт. Парк серверов 20 шт. Обслуживанием лабораторий занимаются заведующие лабораториями, которые являются и преподавателями спец. дисциплин.

Первой лабораторией, переведенной на Linux, стала лаборатория 104 «Программно-аппаратной защиты объектов сетевой инфраструктуры». Выбор дистрибутива, который окажется штатным в лаборатории был долгим и сложным. Мы прошли путь от FreeBSD > PC-BSD > Debian > Fedora > Alt Linux. У Alt Linux подкупила простота начального вхождения, качество документирования и форум, на котором действительно помогают. Мы начали с Alt Linux Школьный Новый Легкий, сейчас работаем на Alt Linux Centaurus P7 и с нетерпением ждем выхода 8 платформы.

В таблице 2 приведён перечень лабораторий и указан год перехода на СПО. Во всех лабораториях расположено по 12 рабочих мест студентов, кроме 304, где 27 раб. мест.

Таблица 2. Перечень лабораторий

№	**Название лаборатории**	**Год перехода**	**Преподаются дисциплины связанные с**	**На рабочем месте студента**
101	Сборки, монтажа и эксплуатации средств вычислительной техники	2012	аппаратным обеспечением ПЭВМ. Установка, настройка системного ПО, развертывание служб и сервисов. Техническое обслуживание и ремонт СВТ.	ПЭВМ, принтер, сканер
104	Программно-аппаратной защиты объектов сетевой инфраструктуры	2009	сетевыми операционными системами, информационной безопасностью.	2 ПЭВМ, на которых студенты развертывают несколько сетей в системах виртуализации.

№ (продолжение)	**Название лаборатории** (продолжение)	**Год перехода** (продолжение)	**Преподаются дисциплины связанные с** (продолжение)	**На рабочем месте студента** (продолжение)
114	Сетевая академия Cisco.	2013	сетевыми технологиями в рамках Cisco CCNA.	ПЭВМ
117	Лаборатория беспроводных технологий	2014	сетевыми технологиями: беспроводные сети.	ПЭВМ, с дискретными беспроводными и проводными адаптерами и видеокартой Radeon для анализа защищенности беспроводных соединений. В стенд входят два беспроводных маршрутизатора и ip-камера.
304	Информати - ки. Программирования и баз данных	2015	информатикой и программированием.	ПЭВМ
305	Программно - го обеспечения компьютерных сетей. Авторизованный центр Cisco Academy	2015	основами компьютерной грамотности.	ПЭВМ

Есть участки, на которых нет возможности перейти на СПО и приходится использовать Windows. В связи с тем, что мы обучаем для конкретных предприятий и организаций, то прикладное ПО диктуют они. Используются такие программные пакеты как Altium Designer, Autodesk Autocad, MS Office, 1C и т.д.

Стандартным пакетом дополнительного ПО в лабораториях является: Remmina, Firefox, Geany, iTALC, OpenSSH, PuTTY, VirtualBox.

К нестандартному относятся такие пакеты как Cisco Packet Tracer, LinSSID, Aircrack-ng, Wireshark, Metasploit Framework.

Сервисы на базе СПО, используемые для обслуживания образовательного процесса, приведены в таблице 3.

Таблица 3. Используемые сервисы

Сервис	**На чем настроен**	**Назначение**
Система видеонаблюдения	Debian 7 + MotionEye	Обеспечение безопасности функционирования лабораторий, отслеживание инцидентов
Локальное зеркало репозиториев Debian	Debian 7 + apt-mirror	Проведение лабораторных работ
Локальное зеркало репозиториев Alt Linux	Alt Linux P7 + sisyphus-mirror	Обслуживание ПЭВМ
Сервер сетевой загрузки	Debian 7 + tftpd-hpa + isc-dhcp-server + syslinux + nfs-kernel-server + smb + apache2	Обслуживание ПЭВМ
Вики-энциклопедия	Alt Linux P7 + Mediawiki	Создание и хранение документации

Сервис (продолжение)	**На чем настроен** (продолжение)	**Назначение** (продолжение)
Сервер SNMP-мониторинга	Debian 8 + Zabbix	Мониторинг сетевого оборудования
Система дистанционного обучения	FreeBSD 7 + Moodle	СДО

Сервер сетевой загрузки позволяет производить загрузку следующих образов ОС – Debian 8, Alt Linux P7, Windows 7/8.1/2012R2, а так же системных и служебных утилит HDT, memtest86+, MS DaRT, MHDD. В 2016 году планируется развёртывание LDAP-домена на базе Alt Linux с целью централизованного управления правами студентов в лабораториях и обеспечения непрерывной рабочей среды, так же мы начали сотрудничество с компанией РусБИТех, запланирована организация учебных мест на базе Astra Linux SE.

УРТК им. А.С. Попова пытается стать образовательной площадкой, на базе которой студенты имеют возможность поработать на современном сетевом оборудовании — (учебные классы таких вендоров как Cisco, D-Link, TP-Link) с современным программным и аппаратным обеспечением — (ПЭВМ i7/16Gb/1Tb для развертывания виртуальных машин – в зависимости от лабораторной работы от 1 до 8 шт. для каждого студента). Кроме этого, мы стараемся держать инфраструктуру колледжа в актуальном состоянии, т.е. систематически добавлять новые интересные сервисы в процесс технического обслуживания образовательного процесса. СПО нам в этом неоценимо помогает, т.к. позволяет активно изучать интересные пакеты и сервисы, оптимально по денежным вложениям и нетребовательно по аппаратным ресурсам, а самое главное, доступно для различных дополнений и модификаций под конкретные задачи.

Вместо вывода:

1. Свободное программное обеспечение позволяет будущим специалистам более глубоко изучать технологии: если студент смог настроить DNS на Linux, то у него не возникнет проблем при аналогичных настройках на Windows или сетевом оборудовании.

2. Кризис и санкции стимулировали внедрение СПО в СПО. Например, до 2015 года руководство особого интереса не проявляло, а в 2015 году мы выступали перед советом директоров ССУЗов Свердловской области с докладом о практике внедрения свободного программного обеспечения в нашем колледже.
3. Свободное программное обеспечение сложнее в развёртывании, настройке и обслуживании. Руководство не готово оплачивать ни полноценную техническую поддержку ни доплачивать за обслуживание СПО, что влияет на мотивацию технического персонала.
4. Так как у Linux отсутствует маркетинговая поддержка, то затруднительно агитировать и продвигать свободные технологии в массы. Например, колледж проводит много различных мероприятий: областные и международные олимпиады. С одной стороны, поддержку данным мероприятиям готовы оказать только люди, интересующиеся СПО в частном порядке. Компании и вендоры не готовы оказывать поддержку. С другой стороны, большое количество учебных заведений испытывают трудности при подготовке студентов к заданиям с использованием СПО.
5. К сожалению, мы «нищеброды», которые пытаются готовить профессионалов, в чём СПО нам активно помогает.

Практическая индивидуальная настройка клавиатуры в GNU/Linux*

Александра Игоревна Кононова, Алексей Владиславович

Городилов, Олег Олегович Кондрашов

Москва, г. Зеленоград, РФ

xkb (X Keyboard Extension) possibilities are not limited to keyboard layout selection and switching from predefined list. With xkb it is possible to create custom layouts with necessary symbols and modifiers, assign up to 8 symbols to a single key. It is also possible to set up non-cyclic layouts switching, re-assign keys of auxiliary keyboards (e. g. gaming mouse keyboard), change layouts on-the-fly, combine symbol entry and layout switch in one key, etc.

В настоящее время наиболее распространённой оконной системой для построения графического интерфейса пользователя в UNIX-подобных ОС является X Window System. Для работы с клавиатурой предназначена одна из подсистем X Window System — xkb. Чаще всего пользователь X Window System встречается с xkb при настройке поддержки русского ввода. Но её возможности не ограничиваются выбором раскладки и переключателя раскладки из предопределённого разработчиком дистрибутива списка вариантов. Подробное описание настройки xkb доступно на сайте Ивана Паскаля [1]. Рассмотрим практическое применение некоторых возможностей этой подсистемы.

Ввод символов

Часто возникает необходимость вставить в текст символ, отсутствующий в используемых раскладках, в частности, тире и кавычки, соответствующие правилам русской типографики. Подсистема xkb предлагает четыре основных способа набора таких символов.

*illinc@bk.ru, http://lvee.org/ru/abstracts/176

1. Указание кода символа в Unicode (в частности, в соответствии с ISO 14755). Позволяет ввести любой существующий символ, но для этого требуется помнить этот код, что не очень удобно. Кроме того, способ задания кода может различаться для разных приложений.
2. Compose-последовательности. Чтобы ввести символ, нажимается специальная клавиша Compose и вводится цепочка символов.
3. Использование существующей раскладки с типографскими символами. Типографское расширение раскладки в xkb включает так называемый третий уровень, что позволяет набрать дополнительные символы, нажав одновременно с клавишей модификатор третьего уровня.
4. Модификация используемой раскладки. Можно изменить существующую раскладку или создать новую, содержащую необходимые для пользователя символы.

Раскладки и настройки

Раскладки в xkb — файлы настроек специального вида, которые описывают символы, генерируемые клавишами. Каждый вариант раскладки — отдельный блок. Внутри блока описаны связанные с клавишей данные [1]. Для большинства клавиш это символы, которые выдаются по нажатию клавиши на различных уровнях (shift levels). Каждому уровню может соответствовать символ, задаваемый кодом Unicode (например, U2190) или специальной константой (например, leftarrow). Файлы раскладки поддерживают комментарии в стиле C++.

Кроме печатаемых символов (цифр, букв, иных символов Unicode), специальных констант VoidSymbol и NoSymbol, а также невизуальных символов, таких как F1-F10, Multi_key (Compose), XF86Back, XF86Forward, SunFront, SunProps и т. д., клавишам могут соответствовать символы и модификаторы, влияющие на состояние клавиатуры. В частности, это символы, изменяющие текущую раскладку (группу символов). Чтобы набрать символ второго или более высокого уровня, при нажатии на клавишу, соответствующую этому символу, зажать ещё одну или несколько клавиш — модификаторов соответствующего уровня. Модификатором второго уровня является Shift. Различные модификаторы третьего уровня (а также, в блоке modifier_mapping, добавление

виртуального модификатора «LVL3» в группу Mod5) описаны в /usr/share/X11/xkb/symbols/level3. Аналогичным образом можно назначить модификатором третьего уровня любую другую клавишу. Символ четвёртого уровня можно получить, зажав модификатор третьего уровня и Shift вместе.

Различные модификаторы пятого уровня (и добавление модификатора «MDSW» в группу Mod3, что необходимо для правильной работы) описаны в файле /usr/share/X11/xkb/symbols/level5. Способом, аналогичным описанному в этом файле, можно назначить и любую другую клавишу. Иногда требуется набрать на некоторой раскладке только один символ (чаще всего это символ латиницы в кириллическом тексте). Готового символа для временного включения конкретной раскладки найти пока не удалось, но эту проблему можно решить с помощью действий (actions). Номер текущей раскладки (группы) определяется суммой трёх переменных — base group, latched group и locked group. Их можно изменить соответствующими действиями [1]:

```
replace key <RALT> \{
    actions[Group1]=[ ],
    actions[Group2]=[ SetGroup(group=-1) ],
    actions[Group3]=[ SetGroup(group=-2) ],
    actions[Group4]=[ SetGroup(group=-3) ]
\};
```

Если включена первая раскладка, при нажатии правого Alt не делается ничего, если вторая — переменная base group уменьшается на единицу на время нажатия и т. д. Постоянное включение первой раскладки также возможно с помощью действий, для этого необходимо изменить переменную locked group.

Некоторые из описанных дополнений к обычной раскладке работают не только в X Window System, но и в «чистой» консоли.

Файлы и утилиты

Основной файл настройки клавиатуры — /etc/default/keyboard, задающий набор раскладок и модификаторов для всех пользователей компьютера и всех клавиатур, причём не только для xkb, но, по умолчанию, и для консоли.

Список файлов используемых раскладок (файлы должны содержаться в каталоге /usr/share/X11/xkb/symbols/) указывается в пе-

ременной
XKBLAYOUT, список блоков — переменной XKBVARIANT. Дополнительные модификаторы и настройки, такие, как поведение светодиодов на клавиатуре, перечисляются в переменной XKBOPTIONS.

Изменить раскладку для текущего сеанса позволяет утилита setxkbmap. Дополнительные настройки могут быть заданы через ключ -option, сам список аналогичен XKBOPTIONS.

Утилита setxkbmap позволяет задать различные раскладки для различных устройств, указав ключ -device [2]. Это может быть полезно, в частности, при настройке игровой мыши, имеющей на боку цифровую клавиатуру. Список всех устройств ввода, в частности, клавиатур, можно получить командой xinput с ключом list. Так как идентификаторы устройств могут меняться от сеанса к сеансу, лучше проверять этот список каждый раз при назначении раскладки.

В том случае, если желаемой конфигурации не получается добиться, комбинируя стандартные файлы, можно изменить одну из стандартных раскладок или создать в каталоге /usr/share/X11/xkb/symbols/ новую по образцу имеющихся. Создание собственного файла раскладки обычно предпочтительнее, так как стандартные файлы могут быть перезаписаны при обновлении.

Файл раскладки должен содержать по крайней мере один блок, описывающий поведение алфавитно-цифровых клавиш. Кроме того, можно переопределить поведение таких клавиш, как пробел и «стрелки» при нажатых модификаторах третьего и пятого уровней. Используемые модификаторы (включатели раскладки, модификаторы третьего и пятого уровня и т. д.) также могут быть описаны непосредственно в файле раскладки.

Таким образом, возможности xkb позволяют, не используя стороннее ПО, легко набирать любые необходимые символы, а также задавать индивидуальные раскладки для различных устройств.

Литература

[1] Иван Паскаль. X Keyboard Extension `http://pascal.tsu.ru/other/xkb/`

[2] XKB remapping `http://www.pixelbeat.org/docs/xkb_remap/`

Что такое SMR-диски и как их готовить*

Yauhen Kharuzhy, Minsk, Belarus

Brief review of Shingled Magnetic Recording (SMR) drives and status of their support in Linux is presented including projects which are currently in development supported by Seagate and HGST companies.

Введение

Плотность записи на магнитные диски растёт, но расти она может не бесконечно. Текущая применяемая в накопителях технология, перпендикулярная магнитная запись, имеет теоретический предел плотности 1 Тбит/кв.дюйм.

Один из вариантов преодоления этого предела — так называемая черепичная магнитная запись, SMR (Shingled Magnetic Recording). В этой технологии используется тот факт, что ширина читающей головки заметно меньше, чем ширина записывающей головки. Соседние дорожки при записи располагаются с перекрытием, каждая следующая перезаписывает большую часть предыдущей, оставляя небольшую полоску нетронутых данных для чтения.

В первых же версиях SMR дисков, выпущенных Seagate и HGST, удалось повысить плотность записи на 25%.

Черепичное расположение дорожек приводит к тому, что запись на них должна производиться последовательно, иначе данные будут потеряны из-за перезаписи «вышележащей» дорожки «нижележащей». Для того, чтобы работа с диском не превращалась в работу с «магнитной лентой» без возможности перемотки, диск делится на зоны. Внутри каждой зоны запись должна производиться последовательно, но работа с зонами может вестись в любом порядке.

Особенности работы SMR дисков

Как уже было сказано, накопители, использующие технологию SMR, разделены на зоны. Для удобства работы и возможности ор-

*jekhor@gmail.com, http://lvee.org/ru/abstracts/179

ганизации файловых систем на таких дисках, зоны делаются разных типов:

- Conventional, «обычная», — зона, внутри которой возможны произвольные запись и чтение, как на классических магнитных дисках;
- Sequential Write, последовательной записи — должна производиться только последовательно, либо в которой произвольная запись эмулируется встроенным ПО накопителя (Sequential Write Required и Sequential Write Preferred, соответственно).

В зонах последовательной записи есть понятие «указателя записи», Write Pointer (WP) — переменной, хранящейся на диске, которая указывает на место, где должна производиться следующая операция записи, чтобы не нарушить последовательность. В случае чтения за пределами записанной области или записи в сектор, отличный от значения WP, прошивка диска либо прерывает выполнение команды с ошибкой, либо эмулирует произвольные чтение-запись, в зависимости от типа зоны.

Также каждая зона может находиться в различных состояниях:

- Empty
- Implicit Open
- Explicit Open
- Closed
- Full
- Read Only
- Offline

Для некоторых дисков количество зон, одновременно находящихся в открытом состоянии, может быть ограничено.

DM, HA, HM диски

В зависимости от того, на какой стороне реализуется работа с зонами последовательной записи, накопители делятся на:

- Device Managed (DM) — накопители, в которых все действия, специфичные для SMR-технологии, спрятаны от контроллера и реализуются встроенным ПО. Для ОС они представляются, как обычные диски, аналогично уже привычным нам SSD, в которых внутренняя «кухня» точно так же скрыта за стандартными командами. Минус подхода — потери в быстродействии.

– Host Aware (HA) — диски, в которых произвольные чтение-запись также эмулируются встроенным ПО, но при этом присутствует набор команд, который позволяет операционной системе получить информацию о зонах и работать с ними;
– Host Managed (HM) — накопители, встроенное ПО которых никак не обрабатывает случаи произвольных чтения-записи в зоны последовательной записи, т.е., операционная система обязана обеспечивать корректность операций чтения-записи самостоятельно.

Текущий статус стандартов

Интерфейс работы с SMR дисками на данный момент ещё не стандартизован, но уже существуют черновики стандартов, определяющие модель работы накопителей и набор соответствующих команд:

– SCSI ZBC, Zoned Block Device — описывает набор зональных команд для накопителей с интерфейсом SCSI;
– ZAC, Zoned Device ATA Command Set — описывает набор зональных команд для ATA-устройств.

Поскольку черновики стандартов «пишутся на ходу», то уже выпущенные производителями диски не всегда им соответствуют, местами отличия достаточно существенные (например, у HGST в некоторых командах параметры передаются в другом порядке), что необходимо учитывать при написании драйверов.

Linux:

уровень SCSI-устройства, libata:

Поддержка есть в основной ветке, Host-Managed диски определяются не как диски, но как SCSI устройства, что позволяет работать с ними через интерфейс SG_IO.

уровень блочного устройства:

Существует набор патчей от Dr. Hannes Reinecke (SUSE), которые реализуют интерфейс стандартного диска для SMR устройств и поддерживает работу с зонами. Эту же реализацию в немного изменённом виде использует Seagate в своём проекте портирования Ext4.

device mapper:

Существует две реализации ZDM (zoned device mapper): от Dr. Hannes Reinecke и от Seagate. Первая обеспечивает эмуляцию произвольных чтения-записи с помощью простого кэширования и перезаписи содержимого зон. Вторая — более сложный вариант, использующий динамическое отображение секторов и зон, подобно тому, как это делают контроллеры SSD дисков.

файловые системы:

Специализированных ФС нет. Есть наработки Seagate по портированию Ext4, также компанией HGST сейчас ведётся работа по добавлению поддержки SMR в btrfs. Btrfs в данном применении обещает быть более перспективной, чем Ext4, поскольку в ней изначально используется подход Copy on Write, за счёт чего необходимость перезаписи произвольных секторов сведена к минимуму.

Литература

[1] H. Reinecke. Strategies for running unmodified filesystems on SMR drives `http://www.snia.org/sites/default/files/SDC15_presentations/smr/HannesReinecke_Strategies_for_running_unmodified_FS_SMR.pdf`

[2] A. Palmer. SMR in Linux Systems `http://events.linuxfoundation.org/sites/events/files/slides/SMR%20in%20Linux%20Systems%20-%20Vault.pdf`

[3] `https://github.com/Seagate/ZDM-Device-Mapper`

[4] `https://github.com/Seagate/SMR_FS-EXT4`

[5] Zoned Device ATA Command Set (ZAC). Working Draft, r05. T13 Technical Committee of Accredited Standards Committee INCITS

[6] Zoned Block Commands (ZBC). Working Draft, r4b. T10 Technical Committee of Accredited Standards Committee INCITS

[7] HGST Ultrastar Archive Ha10. Hard disk drive specifications. Revision 1.0.

Альт на «Эльбрусе»*

Михаил Шигорин, Москва, Россия

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

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

Я работаю в компании «Базальт СПО», которая участвует в разработке репозитория ALT Linux Sisyphus. Как только у нас появился доступ на машину с процессором «Эльбрус-4С», возникло вполне естественное желание портировать туда нашу пакетную базу. Первым этапом стало портирование пакетного менеджера (RPM версии ALT Linux, он же ALT-RPM). Когда заработал rpm, следующим этапом стал запуск hasher — инструмента, с помощью которого собираются пакеты Sisyphus (hasher спроектирован так, чтобы не допускать влияния собираемого пакета на хост-систему, а также взаимного влияния собирающихся пакетов).

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

На время написания тезисов доступна базовая сборочная среда ALT для сборки в автоматически создаваемом силами hasher чруте, за исключением архитектурнозависимых пакетов (binutils,

*mike@altlinux.org, http://lvee.org/ru/abstracts/180

glibc, компилятор), которые пока alien'изированы из предоставленных разработчиком системы deb-пакетов — примерно 230 исходных пакетов.

Основные пройденные стадии сборки:

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

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

В целом, работа позволила оценить достоинства и недостатки:

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

Литература

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

c_uglify — семантический фильтр GNU C для компиляции программ с помощью не-GCC*

Ivan Zakharyaschev, Moscow, Russia

A converter of C code is presented, which overwrites some GNU extensions, making it possible to compile gcc-oriented FOSS software with othe compilers — clang, etc. Converter is based on «language-c»: http://hackage.haskell.org/package/language-c Haskell library. Draft GNU extensions classification and tasks of the distribution porting to an alien platform are reviewed as far as stage of the project and current results of its usage.

Темой представленной работы является преобразователь C-кода, который бы переписывал некоторые GNU extensions. Это необходимо для того, чтобы такие программы можно было компилировать компилятором, который GNU extensions не поддерживает — например, clang или др. Преобразователь реализуется на Haskell-библиотеке language-c http://hackage.haskell.org/package/language-c.

Введение. Мы за или против GNU extensions?

Автор считает, что использование GNU extensions при программировании на C — это не плохо. На это же намекает и название проекта «C uglify»: захочет ли человек видеть обезображенный код (ugly), который получается в результате избавления от GNU extensions, т.е. переписывания программы с языка чуть более высокого уровня на язык чуть более низкого уровня?

Пример из ALT-овского rpm http://git.altlinux.org/people/ldv/packages/?p=rpm.git;a=blob;f=build/interdep.c;h=57b802cfcfb1355234dffefa938f94bf6cdf405f;hb=1ccc182f71c741f4af9c8e72d8cd1a4d35a921fa#l495:

```
// prune src dups from pkg1 and add dependency on pkg2
static
```

*imz@altlinux.org, http://lvee.org/ru/abstracts/185

```
void pruneSrc1(struct Req *r, Package pkg1, Package
    pkg2)
{
    TFI_t fi1 = pkg1->cpioList;
    const TFI_t fi2 = pkg2->cpioList;
    if (!fi1 || !fi2) return;
    struct {
            unsigned int n;
            char list[fi1->fc];
    } pruned;
    bzero(&pruned, sizeof(pruned));
    fiIntersect(fi1, fi2, fiIntersect_cb, &pruned);
    if (pruned.n == 0)
        return;
    addDeps1(r, pkg1, pkg2);
    rpmMessage(RPMMESS_NORMAL, "Removing␣from␣%s␣%d␣
        sources␣provided␣by␣%s\n",
            pkgName(pkg1), pruned.n, pkgName(pkg2));
    fiPrune(fi1, pruned.list);
}
```

Здесь было использовано GNU extension «variable-length array at the end of struct». А после избавления от него вручную стало так:

```
// prune src dups from pkg1 and add dependency on pkg2
static
void pruneSrc1(struct Req *r, Package pkg1, Package
    pkg2)
{
    TFI_t fi1 = pkg1->cpioList;
    const TFI_t fi2 = pkg2->cpioList;
    if (!fi1 || !fi2) return;
    struct {
            unsigned int n;
            char *list;
    } pruned;
    pruned.n = 0;
    pruned.list = alloca(fi1->fc);
    memset(pruned.list, 0, fi1->fc);
    fiIntersect(fi1, fi2, fiIntersect_cb, &pruned);
    if (pruned.n == 0)
        return;
    addDeps1(r, pkg1, pkg2);
```

```
    rpmMessage(RPMMESS_NORMAL, "Removing␣from␣%s␣%d␣
        sources␣provided␣by␣%s\n",
            pkgName(pkg1), pruned.n, pkgName(pkg2));
    fiPrune(fi1, pruned.list);
}
```

Ответ: человеку с таким кодом лучше не иметь дело — при условии, что есть возможность написать его короче и понятнее (как было).

(Любопытно, что использование VLAs at the end of struct появилось в свою очередь в этом коде в результате избавления от другого GNU C extension «nested functions»; оно в ту пору было произведено[1] ещё не столько в целях портируемости кода, сколько по отдельным причинам[1]).

Таким образом, cuglify предполагается использовать не для того, чтобы контрибьютить некрасивые патчи в проекты, которые не удаётся скомпилировать компилятором, отличным от GCC (например, clang), а чтобы сделать возможной выполнение такой компиляции гладко и незаметно для компилирующего (продолжение этой темы — в разделе «Часть большей задачи»).

Существуют и другие примеры «некрасивых» и не всегда компактных патчей, которые были предложены по этой причине в некоторые FOSS-проекты. Можно предположить, что их написание (помимо того, что они делают код менее понятным) потребовало немало человеческих сил (например, патчи на elfutils были растянуты во времени), а проверка на отсутствие ошибок должна была потребовать много человеческого внимания:

- elfutils — неполное решение `https://lists.fedorahosted.org/archives/list/elfutils-devel@lists.fedorahosted.org/thread/MLVCKB43BRADKY3JCJQ4MKOWRYXUFNSB/` (28 insertions(+), 19 deletions(-); 15 Sep 2015)
- elfutils — полное решение для версии 0.159, нет в upstream `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=758064` (1849 insertions(+), 1608 deletions(-)[3]; 13 Aug 2014)

[1]Вложенными функциями, на которые берётся указатель, мы вообще пока заниматься не будем. Во-первых, потому что избавление от них compile-time потребует совсем иного подхода, чем в cuglify, если вообще возможно, а именно более глобального, затрагивающего весь дистрибутив, а не отдельную программу; во-вторых, из-за того, что их run-time реализация через executable stack считается нежелательной хотя бы с точки зрения безопасности, есть надежда, что их и так стараются избегать. Среди пакетов ALT Sisyphus есть около 30 специфических пакетов с executable stack.

– gpm `http://lists.linux.it/pipermail/gpm/2011-April/001122.html` (38 insertions(+), 31 deletions(-)[4]; 3 Apr 2011)
– ...

Также в случае, если патч не принят в upstream, придётся регулярно повторять эти человеческие усилия для переноса на новые версии. (Ср. отставание патча на elfutils от текущей версии upstream-а: 0.165.)

С чем надо справляться (рабочая классификация GNU extensions)

Одним из заметных GNU C extensions являются nested functions. (С полным списком классов проблем, обнаруженных при пересборке Debian clang-ом, можно ознакомиться в [2]). Они были замечены в исходном коде базовых инструментов для сборки дистрибутивов ALT (упомянутый выше случай elfutils и неупомянутые ещё случаи nested functions в rpm).

В первую очередь в cuglify мы занялись переписыванием nested functions.

Набросок классификации GNU C extensions в наших рабочих планах на текущий момент выглядит так:

– GNU C extensions:
  – VLAs;
  – nested functions:
    * на которые берётся указатель;
    * r/w;
    * reader-only;
    * «pure»;
  – ...
  – другие;
  – ...

Пояснение названий приведенных классов вложенных функций: «reader-only» — те, которые только читают переменные из local scope функции, в которую они вложены, но не пишут в них; «r/w» — и пишут, и читают; «pure» — ни то, ни другое.

Классификация nested functions сделана в порядке убывания сложности задачи их переписывания.

Вложенными функциями, на которые берётся указатель, было решено пока не заниматься. Это принципиально иной, более сложный случай, при этом предположительно нечастый[2].

Последние три класса не отличаются на принципиальном уровне, но ожидается, что технически реализация переписывания «r/w» вложенных функций займёт больше времени, т.е. появится в более поздней версии cuglify.

Часть большей задачи. Обобщение задачи семантического фильтра

На самом деле, разработка cuglify — часть большей практической задачи. Мы хотим научиться применять cuglify в работе с некой «инородной» платформой FOO/Linux, а точнее, в работе по созданию порта Sisyphus на эту платформу (т.е. порта репозитория пакетов свободного ПО, из которого создаются дистрибутивы ОС). Здесь FOO/Linux — какая-то платформа без GCC, где есть свой foo-cc, в чём-то похожий на GCC, а в чём-то нет (и foo-cc патчить мы не можем).

Исходя из этого (в ходе работы) возникла такая более общая **формулировка задачи cuglify, как фильтра, работающего с семантикой**.

cuglify является фильтром, цель которого — поддержка задуманной программистом (и мейнтейнером пакета) семантики C-кода с использованием в качестве backend-а не GCC, а foo-cc.

Задуманная семантика может выражаться не только в C-коде (со своими особыми конструкциями вроде nested functions, которые мы переписываем из-за ограничений в backend), но и в опциях gcc. Опции, работа которых в foo-cc нас не устраивает, мы будем исполнять по возможности сами, а опции для backend-а переписывать (хотя в простом случае всего лишь передаём их без изменений). Знание о семантике некоторых опций GCC и особенностях их реализации в foo-cc закладывается в cuglify.

[2]Вложенными функциями, на которые берётся указатель, мы вообще пока заниматься не будем. Во-первых, потому что избавление от них compile-time потребует совсем иного подхода, чем в cuglify, если вообще возможно, а именно более глобального, затрагивающего весь дистрибутив, а не отдельную программу; во-вторых, из-за того, что их run-time реализация через executable stack считается нежелательной хотя бы с точки зрения безопасности, есть надежда, что их и так стараются избегать. Среди пакетов ALT Sisyphus есть около 30 специфических пакетов с executable stack.

В результате незаметно для собирающего под FOO/Linux убирается часть массовых проблем со сборкой пакетов. Не потребуется в них влезать.

Особенности текущей реализации

Исходники на текущий момент расположены по адресу `http://hub.darcs.net/imz/language-c_WIP`. Свежесобранный cuglify под платформу x86 с последним набором работающего функционала может быть взят у автора лично или по инструкции в списке рассылки devel ALTLinux[5].

Есть несколько вариантов, как начать применять cuglify (на инородной платформе). Один вариант — «автоматический нативный» — более основательный и ценный для будущего; второй вариант — «ручной» — уже опробован на практике (при компиляции ALT-ового пакета rpm на Эльбрусе), проще и позволяет быстрее увидеть результат компиляции.

И третий вариант, который прорабатывается в данный момент и который достижим быстрее, чем первый основательный — основан на клиент-серверной модели взаимодействия между сборочницей на FOO/Linux и cuglify на x86. Эта схема использования напоминает ccache или ещё больше distcc, с тем небольшим техническим усложнением, что cuglify должна решать на основании переданных ей C-исходников и GCC-подобных опций, какую команду foo-cc надо выполнить, и попросить сделать это сборочницу.

Благодарности

Автор выражает благодарность за обсуждения в ходе этого проекта участникам ALT Linux Team: Г.Фонтенгауэру-Малиновскому, Д.Левину, Г.Курячему, А.Новодворскому, М.Шигорину, А.Турбину, А.Гладкову. Отдельная благодарность за написание вручную и отыскивание патчей, избавляющих от GNU C extensions в пакетах из Sisyphus, Г.Фонтенгауэру-Малиновскому, Д.Левину, М.Шигорину (и тем самым предоставление примеров, а также за обнаружение на практике опций GCC, которые не так понимаются «другим компилятором», а именно lcc на Эльбрусе). И, конечно же, энтузиасту сборки Debian clang-ом `http://clang.debian.net` Sylvestre Ledru. А также всем участникам создания свободного ПО.

Автор благодарен редакторам и организаторам LVEE2016 Winter за замечания и помощь в редактировании этих тезисов.

Литература

[1] http://git.altlinux.org/people/ldv/packages/?p=rpm.git;a=commit;h=1ccc182f71c741f4af9c8e72d8cd1a4d35a921fa

[2] http://clang.debian.net

[3] 1849 insertions(+), 1608 deletions(-) in elfutils http://git.altlinux.org/people/imz/packages/?p=elfutils.git;a=commitdiff;h=4591378e29823115744c75bde15daa9f9b3539c4

[4] 38 insertions(+), 31 deletions(-) in gpm http://git.altlinux.org/people/imz/packages/?p=gpm.git;a=commit;h=9526f29a486037f15a7fe9c5891ae8a3f39703ba

[5] Вот последняя на момент публикации инструкция, отправленная в devel https://lists.altlinux.org/pipermail/devel/2016-February/200850.html (Её копия сохранена как ann1_4 http://hub.darcs.net/imz/cuglify/browse/ann1_4.md.)

Ключевые риски, связанные с использованием свободных лицензий

Anzhelika Sakhipgareeva, Moscow, Russia[‡]

The article is about risks associated with the use of free software in the company or in the development of its own software products.

Ключевые риски, связанные с использованием свободных лицензий, можно разделить на две большие группы:

- Риски, связанные с использованием свободного программного обеспечения в деятельности компании;
- Риски, связанные с использованием свободного программного обеспечения при разработке собственных программных продуктов.

Риски, связанные с использованием свободного программного обеспечения в деятельности компании

Как ни странно, гражданско-правовой точки зрения особых рисков в данном случае нет. Все виды свободных лицензий предоставляют пользователю свободного ПО неограниченную свободу использования программы (ограничение появляется при ее модификации, но это обычно выходит за рамки обычного использования программы в офисе).

Однако в свое время существовали определенные риски публично-правового характера. В начале 2000-х, когда свободное ПО только начали устанавливать в организациях, Интернет-форумы были затоплены разного рода «страшилками» на тему того, как однажды оперативники Управления К или другие сотрудники правоохранительных органов пришли в гости и, не найдя заветной наклейки на системном блоке, обвиняли в пиратстве и опечатывали компьютеры.

С тех пор имело место немало обсуждений по этому поводу, однако в настоящее время их можно считать во многом устаревшими.

[‡]Sar.3472@yandex.ru, http://lvee.org/ru/abstracts/186

Причин тому несколько. Во-первых, прошло достаточно времени, open source и свободное ПО перестало восприниматься как что-то маргинальное. Термин «свободное ПО» стали широко использовать в различных правовых актах. Во-вторых, появилось официальное разъяснение Министерства экономического развития РФ, которое «благословило» использование свободных лицензий и GPL в частности. Речь идет о письме Министерства экономического развития РФ от 5 мая 2009 г. N Д05-2235 «Об использовании свободного программного обеспечения», в котором было прямо сказано, что «использование свободного программного обеспечения не может быть основанием для применения санкций и препятствий в осуществлении предпринимательской деятельности при контроле за соблюдением авторских прав».

Однако, при наличии подозрений на «заказ» со стороны конкурентов или других заинтересованных лиц, целесообразно подготовиться, и кроме распечатки указанного письма озаботиться распечаткой и нотариальным удостоверением перевода текста лицензионных соглашений на свободное ПО, которое используется в организации (операционные системы, офисные пакеты и т.п.). Печать нотариуса с гербом обычно оказывает успокаивающее действие на правоохранительные органы.

В том случае, когда свободное ПО покупалось в виде коробочных версий (а не скачивалось в сети Интернет), сопутствующая «бумажная» документация (накладные, счета-фактуры, договоры) также способна иметь большое «легализующее» значение.

Риски, связанные с использованием свободного программного обеспечения при разработке собственных программных продуктов

Именно в этом случае следует особенно осторожно подходить к анализу условий соответствующей свободной лицензии для оценки возможности использования кода, лицензируемого на ее условиях, при создании собственного продукта.

Основная проблема заключается в том, что при наличии около 70 свободных лицензий, многие из них несовместимы между собой. Другими словами, сочетание фрагментов кода, распространяемого на условиях лицензий GPL, BSD, MPL и Apache, может оказаться юридически невозможным. Сам факт использования GPL-кода потребует применения к итоговой программе лицензии GPL, в то

время как это будет противоречить лицензии MPL, будет требовать применения MPL к изменившимся файлам. В условиях, когда какой-нибудь фрагмент кода программы используется с нарушением условий его лицензирования, весь результирующий программный продукт приобретает сомнительный правовой статус. Это может иметь значение при дальнейшей коммерциализации программы, особенно, если она будет осуществляться с привлечением зарубежных партнеров, которые традиционно уделяют большое внимание юридической чистоте продукта. Но еще большее значение это будет иметь в тех случаях, когда крупная иностранная компания заинтересуется разработчиком программы и захочет его приобрести. Сопутствующий M & A процесс due diligence включает в себя анализ исходного кода программы, в том числе с привлечением специализированных организаций (например, компании Black Duck и ей подобных, которые могут просканировать код и определить его происхождение). Наличие нарушений лицензионного соглашения при разработке программы может послужить основанием для существенного снижения цены или даже отказа от сделки. Так что разного рода стартапам, которые рассчитывают на интерес со стороны крупных игроков рынка IT, следует особое внимание уделить качеству кода своих программных продуктов.

Разумеется, существует и формальный риск предъявления претензий со стороны правообладателя (автора GPL-программы) или его уполномоченного представителя (для лицензий GPL — это FSF). Существует ряд судебных споров (пока только зарубежных), которые обычно заканчивались капитуляцией нарушителя. См. подробнее: `http://gpl-violations.org`. Даже если в результате и не будет присуждена значительная компенсация, ущерб репутации в сообществе open source будет весьма значительным.

Основным средством минимизации рисков в данной ситуации является знание того, из каких компонентов состоит программный продукт и на каких условиях они могут быть использованы.

Для достижения указанной цели целесообразно внедрение в компании политики использования open source. В частности, обязанность программистов фиксировать происхождения кода, а также загружать вместе с ним те лицензии, которые его сопровождают. Она также может содержать перечни лицензий, которые можно использовать без согласования с юристами (как правило, это все либеральные лицензии) и лицензий, которые требуют такого согласования.

В некоторых случаях целесообразно проведение анализа (сканирование) программного кода с помощью автоматизированных средств (в т.ч. с привлечением специализированных компаний, например Black Duck Software Inc.; Palamida, Inc.).

В любом случае, руководство компании должно донести до сведения сотрудников простую истину (и обеспечить понимание): выбор и использование сотрудниками программного кода является не только их личным делом, а непосредственно влияет на качество результата с точки зрения его юридической оборотоспособности. Следовательно, участие юриста в данном процессе должно рассматриваться не как досадная помеха, а как одно из условий обеспечения дальнейшего коммерческого успеха того, что разрабатывается.

Голос спонсора: EPAM Systems

Голос спонсора: SaM Solutions

Голос спонсора: ITS Partner

Голос спонсора: mycloud.by

Интервью с участниками

По традиции в сборник материалов входят интервью, в которых активные участники сообщества open source делятся своим мнением о свободном ПО, открытых технологиях, роли и месте свободных лицензий, рассказывают, как видят проблематику свободных проектов. В этот раз мы решили выбрать в качестве тематического русла использование свободных лицензий в сфере, отличной от программного обеспечения: в таких областях, как свободный контент, свободное аппаратное обеспечение и криптовалюты.

1 Евгений Хоружий — Минск, Беларусь

LVEE: Для начала, скажи несколько слов о себе в контексте разработки встраиваемых систем.

Евгений Хоружий: Ну и вопрос... Я связан с этой сферой с 2005 года как разработчик, в разной степени приближенности и к аппаратной, и к программной части, но в основном работал с системным ПО. Но и паяльник с 3 класса в руках держал, поэтому когда пошел в embedded, хорошо представлял, что у всех этих устройств внутри.

L: Удачное сочетание одного с другим.

E: Да, к сожалению почему-то в наше время это редко встречается, и очень больно смотреть на нынешних программистов, занимающихся embedded.

L: Как возник интерес к open hardware как таковому? От программного обеспечения, или как?

E: Честно говоря, я как-то не занимался разработкой именно железа, не участвовал в аппаратных открытых проектах, участвовал в программных... И вообще больше в области встраиваемоего ПО работал. Но в принципе, не вижу здесь большой разницы. Другое дело, что меня мучила ностальгия радиокружка, когда в одном месте собиралось много людей, которым было интересно с чем-то разбираться, которые друг другу помогали, подтрунивали, мешали и так далее. И вот тогда, собственно говоря, идея хакерспейса начала кристаллизоваться.

L: То есть просто то, с чем тебе приходилось работать, периодически было родственно open hardware или свободным программным проектам, связанным с embedded.

E: Скорее, проектам, связанным с embedded. Например, был проект OpenInkpot, прошивки для электронных книг. Мы аппаратные электронные книги не делали, это именно прошивка, достаточно низкоуровневая, в общем-то — и да, это был открытый проект.

Ну и свои проекты если делаю, то обычно выкладываю на github, когда не стыдно что-то показать. И, понятно, что если делаю что-то в хакерспейсе, то всегда рассказываю что и как.

L: Кстати, как вообще возник Минский Хакерспейс? С чего это началось?

E: С моего выступления на LVEE кажется... Так, не скромничая :)

На самом деле, где-то в году 12 я закинул эту идею, я уже не помню, на LVEE или на линуксовке. Идея, понятное дело, не моя, хакерспейсы существуют во многих странах достаточно давно. И мне тоже давно такого хотелось, и хотелось верить, что не мне одному.

L: Так интерес начался с того, что ты побывал в действующем хакерспейсе или что-то о них прочитал?

E: Скорее я читал: периодически натыкался на материалы, что в таком-то хакерспейсе что-то сделали. Ресурс `http://hackaday.com` время от времени публикует всякие забавные устройства, которые как раз в этих хакерспесах и делаются.

Честно говоря, не помню, где я увидел это своими глазами, но нужно учитывать еще и школьный радиокружок. Я прекрасно понимал, как это работает; конечно, в отличие от школьных времен, чуть-чтуь другая форма, но разница незначительная.

L: От идеи до возникновения действующего хакерспейса прошло много времени?

E: Много. Даже не помню сколько — наверное год, не меньше. Может, немного быстрее. Сначала мы нашли вариант в БГУ, помещение в здании юридического колледжа. Пока договаривались, узнали, что там еще по вечерам курсы английского будут проходить, и только потом можно будет его полностью использовать.

Это была первая попытка, и она показала, что все-таки очень важно чувство собственного помещения, что оно должно быть в полном распоряжении, чтобы доступ был в любое время. Активность в том первом варианте была нулевой, в том числе и из-за режима доступа.

L: Да, я помню... И невозможность попасть туда на выходных, кажется...

E: Да-да, множество проблем. Некоторые вроде бы были преодолимы, но все равно в сумме это давало фактически ноль. Поэтому пришлось оттуда уйти. И опять же, у нас и у пригласившей нас стороны оказались разные представления о том, что такое хакерспейс. Они надеялись, что мы будем площадкой, из которой вырастают публичные проекты. А по нашему определению хакерспейс — это площадка для каких-то проектов типа хобби, для людей, которые в первую очередь просто хотят чем-то заниматься. Если это вырастает в какой-то проект — это отлично, но не обязательно.

Потом мы пытались что-то делать в ME100, когда это был заводской цех: большая площадка для мероприятий, коворкинги — что там только ни пытались организовать. Но мы снова столкнулись с тем, что находиться внутри чужой площадки очень сложно. Особенно учитывая, что она не была отремонтирована, и мы сами участвовали в ремонте. А к тому времени, когда ремонт стал приближаться к той точке, когда помещением стало можно пользоваться, ME100 вдруг стало ЦЭХом, художественной галереей. И тут уже возникли опять вопросы: оказалось, что находиться внутри галереи совсем неудобно. В общем, оттуда тоже пришлось уйти.

Но к счастью нам попалось текущее помещение. Для хакерспейса оно годится почти идеально — кроме местоположения, пожалуй. Во всяком случае, оно куда лучше предыдущих. Отдельный вход, никаких коворкингов, готовое подвальное помещение, в котором не требуется никакого ремонта. И оно работает.

L: В общем, одиссея пока закончилась.

E: Да, если по срокам сказать, два года назад приблизительно мы заселились.

L: Вопрос, который часто возникает в интервью по поводу хакерспейсов, он наверняка всем имеющим отношение к хакерспейсам надоел, но все равно он и тут прозвучит тоже. Что ты можешь сказать по поводу техники безопасности? Как-нибудь специально прорабатывали эту тему?

E: Несколько раз конечно поднимался этот вопрос, но пока людей у нас все-таки не так много, и удается все это проговаривать в неформальном русле. Но тем не менее, с новым участником оговариваем... И стараемся следить конечно, когда видим что это надо. Но такого уж совсем критичного оборудования у нас нет — только электробезопасность. Сейчас, вот, зарегистрировали организацию и

будем переоформляться на юридическое лицо. Действительно, есть вопросы, кто будет ответственным, но тем не менее, эти вопросы проблем не создают.

L: Мы плавно перешли к официальному статусу. Не так давно все-таки состоялась регистрация.

E: Да, эти перипетии тянулись чуть ли не дольше чем поиск места для хакерспейса. Когда начали его создавать, было понятно, что нужен легальный статус, чтобы иметь возможность собирать членские взносы, официально, арендовать помещение на организацию. Проще всего зарегистрировать коммерческую организацию, или некоммерческкую, но учрежденную не коллективно, а самим собственником. Но лучше всего фактической структуре хакерспейса соответствует, конечно, общественное объединение. Есть у нас в законодательстве такое понятие; к этой категории относятся, например, разные клубы.

L: Опыт Минского велосипедного общества, наверное, здорово пригодился при оформлении.

E: Естественно, что у меня уже был опыт, я долгое время был в этом обществе председателем правления, хорошо знал, как все это делается — и внутреннюю кухню, и требования.

Мы начали пытаться регистрировать организацию в разных вариантах, в том числе и как республиканское общественное объединение. По-моему, 5 или 6 попыток было. Последний вариант не совсем удобный, но по крайней мере работающий: это первичная организация Белорусского общества изобретателей и рационализаторов. Общество существует с 90-х годов, а может и раньше, с советских времен. У него есть даже первичные ячейки на заводах.

L: Нельзя не отметить, вообще говоря, что тематика родственная.

E: Да, устав практически полностью имел все, что нам нужно. Плюс, теоретически, будучи их подразделением, можно получить льготы на аренду государственной собственности. Но главное — получилось зарегистрироваться. И первичная организация заинтересована в нас как в ячейке, у них сейчас не очень большая активность. . .

L: Последний вопрос наверное, хоть и большой. Насколько активность хакерспейса, его участников, связана с понятием свободного программного и аппаратного обеспечения? Можно ли сказать, например, что примерно столько-

то участников Минского Хакерспейса имеет представление о свободных лицензиях?

E: На самом деле, как раз те, кто приходит к нам, уже имеют представление о свободном ПО. Да, конечно, в хакерспейсе есть разные люди, но тех, кто связан с чисто аппаратными проектами, на самом деле почти нет. Приходят именно те, кто либо имел до этого опыт с программированием, либо уже немного владеет темой и знает достаточно про open source. Сказывается, что инициировалось все в Linux-ориентированной среде, и среди участников достаточно много тех, кто связан с Linux достаточно долго. Но есть и пара человек чисто по аппаратной части.

L: Кстати, тему открытого аппаратного обеспечения ведь разрекламировали и протолкнули в массы несколько проектов типа Arduino, на которых достаточно легко создаются разные аппаратные устройства?

E: В мире — да, и Arduino как раз интересный пример, потому что вообще-то ничего особенно революционного в нем нет. Это обычный микроконтроллер, почти без обвязки, но просто разработчики дали возможность любому человеку очень легко стартовать свой проект — что-то написать под него, загрузить, запустить... Увидеть, что лампочка мигает :)

L: И на удивление каким-то образом быстро собралась база проектов, почти готовых, совсем готовых, совсем не готовых — но тем не менее выложенных в открытый доступ. Как-то получилось, что их оказалось достаточно много, чтобы еще больше упростить начальный этап.

E: Да, сначала авторы Arduino сами выкладывали некоторые библиотеки, примеры проектов, а потом уже сформировалось достаточно большое сообщество. Профессиональные «железячники» сначала над этим посмеивались. Там ведь такой подход, что иногда это как стрельба из пушки по воробьям... Но тем не менее, сейчас многие производители каких-то электронных компонентов сразу же выкладывают готовые проекты и примеры, как их подключить к Arduino, протестировать...

L: Доходит до того, что если в коммерческих проектах внутри Arduino, это уже не считается чем-то несолидным.

E: Ну да, в нынешнем мире скорость разработки очень важна, а Arduino как раз позволяет ее сильно увеличить.

Arduino — такая платформа, которая сделала программирование микроконтроллеров массовым. Раньше-то это все требовало достаточно глубокого погружения.

L: Вспоминаешь свой опыт.

E: Да, конечно. Ну а потом следующим таким же успешным проектом стал Raspberry Pi.

L: Который тоже, надо сказать, с аппаратной точки зрения ничего выдающегося собой не представлял в сравнении с конкурентами.

E: В общем-то да. Система на чипе, минимум обвязки, но зато стоимость в пределах $50. Были разные версии, и опять-таки, вставив SD-карту, ты сразу получаешь «живую» систему с возможностью подключения самой разной аппаратуры. Что очень важно, потому что есть много именно аппаратных проектов с его использованием. Кстати, если посмотреть на трамвайные табло, которые в Минске висят по городу — когда их включают, там виден логотип Raspberry Pi.

L: Замечательный пример.

E: Наши матерые разработчики тоже прочувствовали и поняли, что нет смысла изобретать велосипед и делать эти модули самостоятельно или покупать какие-то дорогие промышленные, когда есть дешевая массовая платформа.

L: В общем, подводя итог: получается, что секрет успеха открытого аппаратного обеспечения — низкий порог вхождения и скорость разработки.

E: Да. И доступ к готовым наработкам тоже играет достаточно значительную роль.

2 Міхаіл Волчак — Мінск, Беларусь

LVEE: Ты прымаў удзел у некалькіх папярэдніх LVEE ў досыць розных амплуа: ад прэзентацыі беларускай партыі піратаў да воркшопа па меш-сетках. Як бы ты ідэнтыфікаваў сябе ў першую чаргу?

Міхаіл Волчак: Перш за ўсё для мне цікавыя такія праекты, у якіх удзельнічае супольнасць, таму што, супольнасць дае разнастайнасць, а я лічу што разнастайнасць — гэта такая базавая ўмова для развіцця. Гэта датычыцца розных праектаў, у тым ліку і развіцця размеркаваных меш-сетак. Але вядома я — пірат.

L: Які менавіта тэрмін ты ўкладаеш у слова «пірат»?
M: Піраты для мяне — гэта пэўная ідэалогія, калі Інтэрнэт на сваю карысць выкарыстоўваюць людзі, а не «крывавы карпарэйшн», напрыклад... Гэта пірацкая палітычная ідэалогія, якая зараз узнікае, яе тэзіс, калі лаканічна, дэмакратыя з выкарыстаннем лічбавых сродкаў, праграмных і апаратных.
L: І гэтая твая цікавасць неяк перасякаецца з вольнымі ліцэнзіямі і кантэнтам, да якіх ты маеш непасрэднае дачыненне?
M: Свабодны кантэнт у нашу эпоху ведаў, пост-інфармацыйную эпоху...
L: А пост-інфармацыйная, гэта?..
M: Усе кажуць што мы ў інфармацыйнай, а насамрэч зараз важна пост-інфармацыя, калі мы не толькі ўспрымаем дадзеныя, мы іх ўмеем пераўтвараць у веды. Для спажыўцоў — інфармацыйная. Але інфармацыя гэта не мэта, а сродак або інструмент.
L: І як гэта і пірацтва звязана са свабодным кантэнтам?
M: А як пірацтва звязана с кантэнтам? Кантэнт можна не толькі спажываць, а яго можна яшчэ змяняць, і гэта асноўны падыход, які мне блізкі. Яго можа зменяць кожны, калі мы заходзім у сеціва і маем там вікі ў сваіх свабодных праектах, і такім часам мы не проста спажыўцы, а сустваральнікі. Сустварэнне — гэта такая важная штука для любога апенсорса і ўвогуле любой супольнасці, якая хоча развівацца. Ўключаць новае ў свой дыскурс.

Першы кампанент — гэта сумесны ўдзел і магчымасць змяняць кантэнт, і другі кампанент — абараніць гэты кантэнт у эпоху, калі яго могуць прыватызаваць. Першапачатковую маёмасць супольнасці часам могуць прыватызаваць праз такія інструменты, як інтэлектуальная ўласнасць, гэта цэлы набор законаў, які дазваляе эканамічным суб'ектам, фірмам там, вялікім, малым — прысвойваць. Другі бок — гэта юрыдычнае спараджэнне свабоднага кантэнта, так званыя свабодныя ліцэнзіі, якія мы маем. GNU, Creative Commons (Творчыя Суполкі), яшчэ шэраг. Пад свабодай кантэнту мы маем тое ж самае, што з сафтом, тыя знакамітыя рычардавскія свабоды: свабода вывучаць, змяняць і распаўсюджваць. Калі ў кантэнту адна свабода знікае — знікае і цэласнае разуменне гэтай свабоды. Палітычныя піраты на сёння гэта разумеюць. Напрыклад, з забаронай распаўсюджвання знікае магчымасць арганічнага развіцця супольнасці. Карпаратыўнае развіццё можа мець месца, але тады арганічнасць знікае...

L: Аднак, твая цікавасць да свабодных ліцэнзіях з'явілася яшчэ раней? Як гэта было?

M: Упершыню я пачуў пра гэта дзесьці ў годзе 2009-2010. У той перыяд, калі я пачаў больш глыбока разумець вікі-тэнхалогію і пачаў працаваць. Гэта прыйшло ад сафта. Я доўга мучыўся ад дыхатаміі Windows-Linux, першы мой Linux быў у 1999 годзе, я яго роўна на паўгадзіны устанавіў...

L: Які дыстрыбутыў?

M: Asp Linux, там было некалькі дыскаў, якія я засоўваў і высоўваў — проста набыў дыскі і пачаў з імі іграцца... Але потым была Kubuntu дзесь у 2010 годзе, у мяне стабільна сталі дзве сістэмы. Другі крок гэта былі web-application, Drupal. Я актыўна удзельнічаў у Drupal-супольнасці, актыўна вывучаў Drupal і пасля пачаў чытаць ліцэнзійныя дамовы GNU GPL, разоў пяць пачынаў чытаць і не заканчваў, але мне спадабаўся канцэпт. Канцэпт не з узору легальнасці насамрэч — тое, што ён выяўляўся не для вырашэння задач камерцыі, а для вырашэння задач супольнасці. Я ў Drupal-супольнасць пагрузіўся не як спажывец праграмы, ці там кода, а як удзельнік, ну і тады я пачаў вывучаць пакеты, з якімі яны працуюць, і тады пакаціласа.

Ліцэнзійныя прынцыпы рэфлексуюць прынцыпы супольнасці, і калі ты адымаеш ад супольнасці штосьці з гэтых прынцыпаў, атрымліваецца нейкая карпаратыўная культура, а калі ты працуеш з трыма свабодамі, атрымліваецца супольнасць. А потым з Creative Commons платнічок пайшоў ў 2012 годзе. Я чытаў шмат кніг пра капірайт, і так арганічна гэта пайшло туды...

L: Лоўрэнс Лессіг распіярыў?

M: Ведаеш, магчыма ад яго пайшлі некаторыя словы. Яго «Свабодная культура» дала мне нейкі стартавы слоўнікавы запас, на які можна было абапірацца далей.

L: А раскажы трохі аб Вікіпедыі. Пра сваё знаёмства з самой тэхналогіяй, падыходам, калі кам'юніці рэгулюе артыкулы.

M: Знаёмства адбылося практычна. Напісаў артыкул, яго праз некалькі хвілін выдалілі, а я на яго патраціў тры дні. Тады я зрабіў паўзу, таму што не разумеў, як працуе гэта супольнасць... Вядома, я выкарыстоўваў той жа падыход, які выкарыстоўваў у Drupal-супольнасці, але яно не спрацавала. І хутчэй па культурна-палітычным плане...

L: Як гэта?

M: Я напісаў артыкул, які не адпавядаў крытэрыям значнасці для рускай вікіпедыі... Потым я зноў пачаў пісаць, напісаў артыкул у беларускай вікіпедыі, яго заапрувілі і жыццё пайшло... І вось у гэты момант, калі заапрувілі (быў статус «на вычытку» і раптам змяніўся) — мне стала цікава, як адбываецца самарэгуляцыя, як кантэнт застаецца рэлевантным.

L: У нейкім сэнсе самарэгуляцыю ты адчуў на сабе, пры першай спробе...

M: Не зусім так. Спачатку мне здавалася, што гэта самарэгуляцыя, але потым прыйшло разуменне. Моўныя раздзелы, якія маюць 100-200 тысяч артыкулаў, выкарыстоўваюць стратэгію «набрацъ колькасць», калі прымаюцца любыя артыкулы, якія маюць 2-3 крыніцы. Калі раздзел перасякае рысу 500-600 тысяч, яны трохі змяняюць стратэгію,але вось гэтая свабода дадавання колькасці існуе. А калі яны перасякаюць мільён, або 800-900 тысяч, яны кардынальна мяняюць стратэгію і павялічваюць крытэрый значнасці. Такую стратэгію спачатку выкарыстоўвала і руская вікіпедыя, але пасля гэтага парога яны пачалі касіць усё, што нязначна для Расіі і выкарыстоўваюць тую ж стратэгію для беларускіх артыкулаў. Іншымі словамі, для "свадобных" ведаў у рувікі выкарыстоўваюцца геаграфічныя і культурна-палітычныя межы.

L: Патлумач гэты момант.

M: Ну, мы ж з большасці рускамоўныя, таму і пішам там.

У вікі-супольнасці ёсць 5 прынцыпаў асноўных, адзін з іх — крытэрый значнасці, notable sources, і ў многіх вялікіх раздзелах яна дэфармавалася ў крытэрый мега-якасці, які стварае такі парог уваходу, што ў невялікіх супольнасцяў не знойдзецца столькі спецыялістаў, якія будуць пісаць на адмысловую тэму. Атрымліваецца што невялікая супольнасць, напрыклад беларуская, якая прымае такія ж стандарты, якія існуюць у польскай або ў рускай wiki (у польскай мякчэй) — узнікае такі парог уваходу, што пачаткоўца, які захоча нешта напісаць, выкідваюць з удзелу.

L: А чаму такі падыход не назіраецца ў ангельскай?

M: Да, з ангельскай вікі такія праблемы не ўзнікалі, я публікаваў нават такі эксперыментальны матэрыял, які быў бы з рускай выключаны з тымі крытэрыямі. Кожная супольнасць, нават і анлайнавая, рэфлексуе, як бы ты не хацеў называць яе анлайнавай, пэўную мадэль лакацыі... Ангельская — выключэнне, таму што гэта вельмі глабальная супольнасць, яна больш інтэрнацыянальная, там і Еўропа, і Амерыка, і Азія. Ўсюды англійскае камьюніці, і таму

там культурнага рэлятывізму або нейкай палітычнай скіраванасці не існуе.

L: І напрыканцы, як ты ўяўляеш сабе перспектывы адкрытых ліцэнзій ў вобласці, не звязанай з софтам: у open media, open hardware?

M: Асноўная праблема кантэнтных і сафтовых ліцэнзій — тое, што яны не арыгінальныя для трох чвэрцей гэтай планеты. У іх вельмі класная задума, як хакнуць залішнія абмежаванні, напрыклад, капірайта, але яны вельмі арганічна глядзяцца ў краіне, дзе капірайт найбольш развіты і стаў часткай штодзеннага жыцця для многіх. Для Беларусі гэта нехарактэрна, Беларусь ніколі не была краінай, дзе капірайт вельмі прывіваўся... Ад Францыска Скарыны, які быў піратам і правозіў кантрабанднае абсталяванне праз польскую мяжу для распаўсюду свабодных ведаў у Вялікім Княстве Літоўскім.

L: ?!

M: Так, яго двойчы затрымлівалі, і толькі на трэці раз яму ўдалося правезці гэта hardware, і толькі пасля гэтага ён змог заняцца тут кнігадрукаваннем... Умоўна кажучы, гэтая ліцэнзійная фішка павінна перасэнсавацца нашым грамадствам, і ўсімі не Western Europe і амерыканскімі супольнасцямі. Заходняя Еўропа — гэта індывідуалістычны падыход да ўсяго, а open source — гэта community ownership, traditional knowledge, традыцыйныя веды, калі усё "племя" нешта ведае, і ніхто не робіць гэтыя веды прапрыятарнымі, каб нажыцца. Сеткавыя супольнасці зараз працуюць па гэтым жа прынцыпе як плямёны там... з Лацінскай Амерыкі ці Аўстраліі, і таму тут пытанне з гэтымі ліцэнзіямі, якія індывідуальна накіраваныя.

L: Але яны досыць паспяхова прымяняюцца?

M: Не зусім так. Спачатку мне здавалася, што гэта самарэгуляцыя, але потым прыйшло разуменне. Моўныя раздзелы, якія маюць 100-200 тысяч артыкулаў, выкарыстоўваюць стратэгію «набраць колькасць», калі прымаюцца любыя артыкулы, якія маюць 2-3 крыніцы. Калі раздзел перасякае рысу 500-600 тысяч, яны трохі змяняюць стратэгію,але вось гэтая свабода дадавання колькасці існуе. А калі яны перасякаюць мільён, або 800-900 тысяч, яны кардынальна мяняюць стратэгію і павялічваюць крытэрый значнасці. Такую стратэгію спачатку выкарыстоўвала і руская вікіпедыя, але пасля гэтага парога яны пачалі касіць усё, што нязначна для Расіі і выкарыстоўваюць тую ж стратэгію для беларускіх артыкулаў. Ін-

шымі словамі, для “свадобных” ведаў у рувікі выкарыстоўваюцца геаграфічныя і культурна-палітычныя межы.

L: Тут выразна чутны голас пірацкай партыі :-)

M: Я пакуль спрабую перасэнсаваць гэтае пытанне... Я за тое, каб гэтыя ліцэнзіі прысутнічалі, я сам прасоўваю ліцензіі Creative Commons ў Беларусі. Але калі займаешся іх прамоўшнам... Так, пад вікіпедыяй ёсць Creative Commons, але 90% вікіпедыстаў разумеюць гэтую ліцэнзію хутчэй праз паводзіны ў супольнасці, а не ў юрыдычным плане. Паовдзіны супольнасці фарміруюць правілы — змест гэтых ліцэнзій, а не наадварот. Гэта для мяне важный момант...

А ліцэнзіі на жалеза — гэта больш накшталт патэнтаў. У патэнтаў ёсць адзін плюс... У іх ёсць шмат мінусаў, але адзін плюс, якому пакуль не прыдумана альтэрнатыва. Яны ўсё ўключаюць у адну базу дадзеных, якую чалавек можа праглядаць. Вось калі open source community, якое працуе з hardware, выпрацуе нейкі падобны механізм з класіфікатарам, катэгарызацыяй — тады гэта можа быць годнай альтэрнатывай, але пакуль такога механізму няма. Гэта выклік — ці могуць актывісты open hardware такое зрабіць, але такое трэба. Першая ўмова росту папулярнасці вольных ліцэнзій для hardware — гэта стварыць сістэму, якая будзе больш эфектыўная, больш адкрытая, чым патэнтнае бюро, патэнтныя офісы па розных краінах.

3 Даниэль Надь — Будапешт, Венгрия

LVEE: Поскольку нынешние интервью связаны с комьюнити-ориентированными и свободными технологиями за пределами непосредственно свободного программного обеспечения — поговорим о криптовалютах. Как возник твой интерес к этой сфере?

Даниэль Надь: Это уходит корнями в глубокое детство. В 1984 году была выпущена такая компьютерная игра, Elite.... . .

L: Ух ты! Всё началось с неё?

Д: Вот, ты её тоже помнишь. Я в неё вложил — т. е. закопал — многие тысячи часов, наверное. И когда уже появился Интернет, у меня была такая мечта — сделать что-то похожее, но для большого количества игроков. Сейчас уже такие игры существуют, а когда я занимался этим своим юношеским проектом — тогда возникали две

проблемы. Во-первых, достаточно быстро стало ясно, что на централизованном сервере хранить все это очень затратно. Хотелось использовать вычислительные мощности клиентов, сделать какую-то распределенную систему. А во-вторых, я заметил, что в Elite наступал такой момент, когда у игрока становилось очень много денег, и игра из-за этого делалась несколько скучной. Я задумался, почему так не случается в жизни, начал читать про денежную систему — как это работает, что такое деньги вообще, почему, например, в игре никогда не случается так, что у меня есть какой-то товар, а у покупателей на космической станции нет денег...

L: Действительно :)

Д: ...а в реальном мире это достаточно частая ситуация. В общем, я начал вчитываться в суть денег, потом наступил крах доткомов 2001 года, когда у многих появился интерес к этой теме, и появилось с кем об этом поговорить. Сначала я начал вчитываться в то, как можно вести распределенный бухгалтерский учет для большой онлайн-игры, а через некоторое время задумался: зачем это делать в игрушечном варианте, когда чего-то подобного миру не хватает по-настоящему.

L: Как обстояли на тот момент дела с электронными платёжными системами?

Д: В 1998 году в русскоязычном пространстве появился Webmoney, а за несколько лет до этого был DigiCash, так что ростки криптовалют — они появлялись в конце 90-х, а теоретические основы (академические статьи на эту тему) начали появляться ещё в 80-х.

L: А в распределенном, децентрализованном виде?

Д: Всем хотелось децентрализации. Это был такой Святой Грааль, который всем хотелось сделать. И, естественно, была проблема византийского консенсуса. Если описать простыми словами — это задача, как обеспечить, чтобы состояние бухгалтерского учета было для всех одинаково, с какой точки ни смотри на систему, т. е. чтобы всегда сохранялась целостность. Это была достаточно сложная проблема, и хорошего практического решения она не имела, пока вдруг из ниоткуда не появился Bitcoin.

L:А блокчейн?

Д: Он не существовал до Bitcoin.

L: Насколько понимаю, распределенная децентрализованная система по идеологии и принципам достаточно близка открытому ПО. И что, Bitcoin оказался первой реально

успешной системой электронных денег, по своим принципам родственной свободному ПО?

Д: Не сказал бы, что первой. Были свободные проекты до Bitcoin. И ePoint ведь тоже был основан на открытом ПО, и еще было много разных проектов. Это в начале двухтысячных была достаточно горячая тема, многие ею занимались, были разные эксперименты, но они были не очень успешны именно из-за той нерешенной проблемы, о которой мы только что говорили.

L: Кто первый сумел ее решить, тот завоевал мир?

Д: Да. Хотя я этот параметр и не очень высоко ценю, но тем не менее рыночная капитализация Bitcoin в разы превосходит рыночную капитализацию остальных криптовалют вместе взятых.

L: С технической точки зрения, со времён появления Bitcoin криптовалюты как-то развивались?

Д: Конечно. Есть разные направления развития. Кстати не все они положительные. Да и направление развития Bitcoin не совсем соответствует тем ожиданиям и предсказаниям, которые ему сопутствовали в 2009 году. На сегодняшний день децентрализация Bitcoin во многом условна. На самом деле обработка транзакций происходит на достаточно маленьком количестве узлов, которое к тому же еще и сокращается.

L: И территориально, насколько я помню, есть проблема, что почти все они расположены там, где быстро делают микросхемы, и...

Д: ...и где легко воровать электричество, называя вещи своими именами.

Когда эти проблемы Bitcoin стали более-менее очевидными, многие взялись их решать. Первая более-менее успешная альтернатива — Litecoin, базируется на коде Bitcoin, и тут как раз видно преимущество открытого ПО, потому что можно было не реализовывать всё с нуля, а просто взять Bitcoin и поправить там некоторые вещи, чтобы достичь тех целей, которые разработчики себе поставили. В случае Litecoin основной задачей была децентрализация майнинга. В отличие от ожиданий Сатоши Накамото, оказалось что идеальный майнер в Bitcoin — это не универсальный компьютер, а некая микросхема, созданная именно под эту задачу. И создатели Litecoin задались целью создать такой алгоритм майнинга, который лучше всего выполняется на компьютере общего назначения. Это им в какой-то степени удалось.

А еще в тот момент, когда запустили Litecoin, люди уже были знакомы с Bitcoin, поэтому у этой системы не было ещё одной проблемы. Около трети общего количества биткоинов находится в руках максимум 18 юридических или физических лиц. Эта очень большая централизация денежной массы случилась из-за того, что в самом начале мало кто знал о Bitcoin, и ранние майнеры очень много себе намайнили. С Litecoin такой проблемы не было, т. к. когда его объявили, очень многие бросились заниматься майнингом. И сам майнинг там децентрализован благодаря тому, что очень долго не было (а по-моему, и до сих пор нет) специализированных микросхем.

Это было первое развитие темы Bitcoin — достаточно успешный проект, у которого тоже существенная рыночная капитализация, и кроме того есть ещё некоторые преимущества перед Bitcoin. Например, существенно быстрее время блоков.

L: Думаю, это нелишним будет пояснить.

Д: Для тех кто не знаком с основными принципами блокчейна, можно сказать, что Bitcoin — это такая большая публичная бухгалтерия, куда записываются все транзакции, и страница в этой бухгалтерской книге — это блок.

L: Копии всех страниц должны быть на каждом узле, и благодаря системе указателей с цифровыми подписями ни у кого нет возможности незаметно модифицировать часть блокчейна.

Д: Да. И пока транзакция не попала в эту общую книгу, она теоретически может быть ещё изменена или удалена. Транзакция становится точной и необратимой, когда попадает в этот децентрализованный распределенный блокчейн. В случае Bitcoin это занимает в среднем около 10 минут, а в случае Litecoin — 2.5 минуты.

L: После этого были ещё похожие системы?

Д: Очень много. Многие пытались что-то улучшить. С технической точки зрения им это может быть и удалось, но в случае с криптовалютами очень силён эффект сети. Даже если какая-то криптовалюта по параметрам лучше Bitcoin, для того, чтобы у нее появилась существенная база пользователей, она должна найти для себя нишу, потому что сам тот факт, что огромное количество людей и бизнесов пользуется Bitcoin, делает остальные альтернативы непривлекательными.

L: Кстати, по поводу блокчейна, который присутствует на всех узлах. Вроде бы криптовалюты считаются такими ано-

нимными, но при этом все твои транзакции сохраняются навсегда в публичном доступе. Они вроде бы пока не привязаны к твоей персоне, но никогда нет гарантии что...

Д: ...что кто-нибудь потом не привяжет.

L: Да. Так насколько корректно считать это способом анонимнизации платежей?

Д: По-моему, совершенно некорректно. Bitcoin я бы анонимным не назвал. Есть отдельные анонимизаторы, такие сервисы, принимающие грязные биткоины, а возвращающие отмытые. Потом есть, скажем так, теоретически разработанные технологии, при помощи которых можно создать действительно анонимную криптовалюту. Этим сейчас тоже занимаются люди, и думаю, это тоже одно из направлений, куда в будущем будут развиваться технологии.

L: По поводу анонимности. Получается, если какие-то правительства косо смотрят на Bitcoin по поводу возможности сокрытия доходов — то это скорее ошибочная точка зрения?

Д: Нет, с этим уже не соглашусь. Несмотря на то, что биткоины не очень сложно привязать к какому-то конкретному имени, их очень сложно отнять и очень сложно воспрепятствовать их движению. Само по себе использование Bitcoin еще не спасает в случае чего, но если у человека значительная часть сбережений в биткоинах — он, грубо говоря, бежит значительно быстрее в случае каких-то потрясений. Когда случаются какие-то финансовые катаклизмы, массовые неплатежи или финансовый кризис в банковской системе, тогда бывает паника, люди пытаются перемещать свои капиталы, но многим это не удаётся или удаётся только частично. И вот в этом случае Bitcoin очень сильно помогает — это было продемонстрировано в случае кипрского кризиса, потом греческого кризиса, последнее время в Восточной Европе тоже, в Украине, и в Беларуси тоже, по-моему, те кто сделал ставку на Bitcoin, не ошиблись.

Но есть ещё и такое преимущество Bitcoin и вообще систем распределённого бухгалтерского учета: если надо, они достаточно хорошо сочетаются с системой налогообложения, контроля. Например, я сейчас получаю значительную часть моей прибыли в криптовалютах, получаю от фонда, фонд должен держать свою бухгалтерию открытой, тот факт что это получаю я, а не кто-то другой — это не секрет, я с этих денег конечно же плачу налоги. В налоговой было достаточно просто показать, что вот этот счет, вот транзакция — в общем, из-за этого не возникает подозрений, что где-то тут

мошенничество или что я что-то скрываю. Т. е. если я хочу играть с открытыми картами — я имею такую возможность.

Подводя итог, такой подход в значительной степени отдаёт контроль над средствами индивиду или организации, даёт возможность решать, какую часть своих операций держать в какой юрисдикции. Это ставит юрисдикции в положение конкуренции друг с другом.

L: Еще вопрос про применение блокчейна за пределами платёжных систем. Насколько понимаю, с тех пор появились и такие проекты?

Д: Да. Я работаю как раз над таким проектом. Сейчас наиболее известен и успешен среди таких проектов Etherium — проект, который действительно превратил распределенный бухгалтерский учет Bitcoin в распределенную базу данных общего назначения. Там можно хранить в блокчейне любые данные и записывать условия изменения этих данных на тьюринг-полном языке. То есть на этих данных можно проводить практически любые вычисления так, что все могут убедиться: действительно результатом этих вычислений стало то-то и то-то. Здесь открываются такие возможности, о которых даже трудно говорить. Это как пытаться в начале 80-х годов рассказать, что такое персональный компьютер. Подавляющее большинство будущих применений еще не видны из нашей перспективы, потому что их еще не изобрели.

L: Можешь привести хотя бы парочку примеров, случайно выбранных?

Д: Попытаюсь. Например, можно сделать страховку без страховой компании. Если какие-то люди хотят распределить между собой какие-то риски, они могут записать условие в умный контракт: тот, кто регулярно делал какие-то взносы и с ним что-то случилось, получает определённые деньги. Это записывается в контракт, и таким образом страхование становится распределенным и децентрализованным: нет страховой компании, есть только люди, которые между собой поделили риски.

Или, что меня больше всего интересует — это общее пользование капиталом. Самое простое, то над чем я сейчас конкретно работаю — чтобы любой человек мог сдавать в аренду свои вычислительные ресурсы. В данном случае это хранение и передача данных, такое распределенное хранилище, к которому любой человек может подключить свои накопители и зарабатывать на том, что его накопителями пользуются другие люди.

L: То есть мы говорим не строго о вычислительной мощности процессора, мы понимаем более широко.

Д: Да, более широко, но и процессор тоже. В будущем, я думаю, можно будет сдавать в аренду и процессор. Есть какая-то программа, которая записывается в блокчейн, и потом данные можно рассылать разным участникам, они эти данные будут обрабатывать. Конечно, должна быть некоторая избыточность. Если те же самые данные два узла обработали иначе, автоматически без участия человека можно выяснить, кто из них обработал данные неправильно, и исключить этот узел. Можно, например, распределенно рендерить фильмы. Но это, мне кажется, только первые ласточки, это достаточно просто. А вот, например, есть такая проблема, что огромное количество капитала человечества существует в виде припаркованных автомобилей, которые ничего не делают, а просто загораживают дорогу.

L: Неожиданно.

Д: Из-за того, что мы друг другу не доверяем, я не могу подойти к какому-то припаркованному автомобилю, быстро доказать владельцу, что я надёжный и хороший человек и у меня есть достаточно сбережений, чтобы компенсировать ему ущерб, если я его машину сломаю. Несмотря на то что это так, я не могу легко и просто это доказать. Но при помощи блокчейн-технологий это все становится доказуемо: как репутация, так и финансовое положение. Можно сделать такую автоматическую систему, когда к любому автомобилю можно подойти, быстро доказать, что мне его можно дать в аренду, заплатить, поехать куда-то и там его оставить для следующего желающего. Таким образом, количество машин, которое должно существовать, оказывается значительно меньшим. Многим не нужно будет иметь собственный автомобиль. И мне кажется что в итоге от этого человечество станет значительно богаче: 90% автомобилей не нужно будет производить, и эти мощности можно будет использовать для чего-то другого.

L: По существу, блокчейн — такой достаточно мощный механизм, который «перетащит» комьюнити-технолгоии в осязаемую сферу?

Д: Да. Научная фантастика на эту тему существует, мы ее читали, и действительно сказку сейчас делаем былью, в том смысле, что, как мы считаем, он перевернет мир так же, как в свое время это сделал доступный компьютер. Это очень амбициозный проект, и над ним очень приятно работать :)

sam solutions

23+	600+	50+	9.5/10
Founded in 1993	Employees	Active clients	Satisfaction score from annual client assessement

COMPETENCIES

- SOFTWARE ENGINEERING
- MOBILE AND EMBEDDED
- E-COMMERCE SOLUTIONS (hybris, Magento)
- OUTSOURCING & NEARSHORING

INDUSTRIES

High-Tech · Smart Products · Retail · Intellectual Property · Telecommunications · Automotive

SELECTED CLIENTS

abat · Atos · Europäisches Patentamt / European Patent Office / Office européen des brevets · FUJITSU · HAUFE. LEXWARE · KASPERSKY lab · LBi · PAYBACK · SIEMENS

SaM Solutions GmbH & Co. KG
Am Bahnhof 4a
82205, Gilching
Germany

Tel.: +49 81 057 7890
Fax: +49 81 057 78920
info@sam-solutions.com
www.sam-solutions.com

Reliable software engineering partner with 20 years of experience

Nearshore capacities for cost-efficiency

Lean & Agile project methodologies

Advanced tools for efficient communication for clear requirements

Real-time dashboards for project quality management

Partnerships with leading technical universities

RECOGNITIONS & PARTNERSHIPS

Microsoft Partner
Gold Application Development
Silver Application Lifecycle Management

CERTIFIED ISO 9001

ORACLE PARTNER

The Global Outsourcing 100 IAOP

SOFTWARE 500

GLOBAL 100 SERVICES

ИТС Партнёр – **кто мы такие**?

Мы – белорусская компания, на рынке с 2009 года, резидент Парка Высоких Технологий

Разрабатываем:

- встроенное ПО для систем хранения данных и роутеров;
- веб-сервисы и приложения;
- мобильные приложения;

60+ сотрудников и растем, 70% разработчиков с опытом более 5 лет.

Более 1 миллиона устройств с разработанным нами встроенным ПО продаётся ежегодно.

Самый крупный проект – более 25 000 человеко-часов.

Стабильная команда профессионалов.

Чем интересны?

Наши проекты на C++:

Ready NAS OS – встроенное ПО для устройств (класс SMB) хранения данных

Ready DATA – встроенное ПО для устройств (класс Enterprise) хранения данных: платформа для единого хранения данных (NAS/SAN) на базе технологий корпоративного класса для бизнеса любого уровня.

ARLO – умная система домашнего видеонаблюдения. Определение и отслеживание движения, обучение и распознавание образов, запись видео и фото, удалённый доступ к записям со смартфона или из веб-клиента.

SportStation – мобильный программно-аппаратный комплекс для организации спортивных состязаний по бегу.

Плюшки и бонусы:

- Возможность карьерного роста
- Работа в распределённых командах
- Командировки в Европу и США
- Оплата профессиональной литературы, конференций, и других вариантов повышения квалификации
- Бесплатное обучение английскому в офисе
- Компенсация стоимости спортивных занятий
- Чай / кофе / сливки / печенье / фрукты
- Интересные корпоративные праздники
- Подарки детям сотрудников на Новый год
- И, конечно же, традиционный Pizza Day: каждую 2ю пятницу месяца заказываем пиццу или суши или еще какую-нибудь вкусняшку за счет компании + готовим интересную тему для обсуждения и здорово проводим вместе время.

А кто *вы*?

Пишите: info@itspartner.net

Звоните: +375 25 6922319

Будем рады знакомству с вами!

Научное издание

ОТКРЫТЫЕ ТЕХНОЛОГИИ

Сборник материалов двенадцатой международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2016

(Гродно, 25–28 июня 2016 г.)

Фото на обложке Арсения Случевского

Ответственный редактор Д.А. Костюк
Компьютерная верстка А.А. Маркина

Подписано в печать 16.08.2016.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная.
Ризография. Усл. печ. л. 10,7. Уч.-изд. л. 6,8.
Тираж 170 экз. Заказ 3404.

Издатель и полиграфическое исполнение:
частное производственно-торговое унитарное предприятие
«Издательство “Альтернатива”».
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий
№ 1/193 от 19.02.2014.
№ 2/47 от 20.02.2014.
Пр. Машерова, 75/1, к. 312, 224013, Брест.