Вторая

международная конференция

разработчиков свободных программ

на Протве

Тезисы докладов

Обнинск

25-27 июля 2005

Оформление и дизайн: Илья Кравец
Верстка: Илья Машкин

В сборнике представлены тезисы докладов, одобренных Программным комитетом Второй международной конференции разработчиков свободных программ на Протве. Другие материалы конференции можно найти по http://conf.altlinux.ru

Программа конференции

24 июля

25 июля

10.00 – 12.30: Регистрация в холле гостиницы.
12.00 – 12.30: Кофе

Дневное заседание
12.30 – 14.30

12:30-13:00: Открытие конференции
13.00 – 13.45: Дмитрий Тараканов

13.45 – 14.30: Вартан Хачатуров

Вечернее заседание
(15.30 - 19.00)

15.30 – 16.15: Станислав Иевлев

16.15 – 17.00: Дмитрий Тараканов

17.30 – 18.15: Федор Зуев

18.15 – 19.00: Александр Боковой

26 июля.

Утреннее заседание
(9.30 – 14.00)

Дневное заседание
(14.45 - 17.00)

27 июля

24 июля

18.00 – 22.00: Регистрация в холле гостиницы

25 июля

10.00 – 12.30: Регистрация в холле гостиницы.
12.00 – 12.30: Кофе

Дневное заседание

12.30 - 14.30

12.30 – 13.00: Открытие конференции.

13.00 - 13.45

Дмитрий Тараканов
Москва, Intel

Программа Intel Software Network и инструменты компании для разработки программного обеспечения.

В докладе представлен краткий обзор новой программы для разработчиков Intel Software Network и инструментов для разработки программного обеспечения под ОС Linux*. Подробно рассказывается о возможностях новых версий инструментов Интел.

13.45 – 14.30

Вартан Хачатуров
Siemens

Linux в компании Siemens: история успеха.

Доклад посвящён истории внедрения операционной системы Linux в качестве перспективной платформы для разработки встраиваемых устройств. Будут освещены задачи, которые требовалось решить созданному центру компетенции Embedded Linux, политика руководства компании по отношению к Linux, приведены некоторые примеры успешных продуктов на основе
Linux.
Во второй логической части доклада будут представлены технологические особенности Linux и FOSS community, которые помогают компании Siemens в создании новых продуктов, и те аспекты, которые часто являются камнем преткновения для заказчиков.

14.30 – 15.30: Обеденный перерыв

Вечернее заседание

(15.30 - 19.00)

15.30 – 16.15

Станислав Иевлев
Москва, ALT Linux

Проект: Sisyphus

Alterator как платформа разработки.

Alterator - это платформа для разработки приложений, связанных с организацией интерфейса управления типовым решением на основе Unix-подобной ОС и её служб. Задача проекта - упростить процедуру разработки таких решений, а также последующее их сопровождение и эксплуатацию.

С момента прошлогоднего доклада alterator сильно изменился, появилась и "отболела детскими болезнями" первая реализация, были уточнены многие моменты, появились новые наработки и идеи.

Alterator существенно отличается от других имеющихся проектов создания архитектур для построения конфигуратора.

Во-первых, они имеет не классическую двузвенную модель интерфейс-бакенд, а трёхзвенную: интерфейс-модель-бакенд. Наличие ещё одного

промежуточного слоя позволяет быстро создавать новые решения из готовых компонентов и не закладываться в низкоуровневой части (бакенд) на особенности того или иного создаваемого решения.

Во-вторых, имеется принципиальная возможность адаптации любой степени сложности уже готового, поставляемого решения без модификации компонент, поставляемых из коробки, что позволяет избежать проблем при проведении экстренных обновлений системы.

В-третьих, при решении той или иной задачи вы пишете тот или иной компонент на произвольном, наиболее подходящем для решения задачи языке программировании без каких-либо биндигов (привязок) к несущей среде.

Единственное исключение (которое исчезнет в будущем), это единый, регулярный язык описания интерфейсов. В качестве базового языка был выбран - Scheme. Его размеры (это один из самых маленьких универсальных языков программирования), а также особенности синтаксиса позволяют сделать описания интерфейсов легко читаемыми и одновременно избежать изучения дополнительных языков разметки. При традиционном подходе (XUL) вам надо изучать отдельно: язык разметки (в данном случае xml), который зачастую обладает собственным достаточно нерегулярным синтаксисом, язык описания динамики виджетов (javascript). Также, ещё часто требуется некий вариант препроцессора, для создания первоначального заполнения виджетов данными, взятыми у backend. В случае alterator - для всего этого надо будет изучить минимум одного единственного языка - Scheme.

Вот небольшой пример описания интерфейса:

```
(hbox
  (id 'my-label (label "Some Label"))
  (button "Some Button"
          (on-click
                  (my-label text "Button clicked"))))
```

Как видно он совершенно не уступает по читаемости своему XML-аналогу и имеет регулярный синтаксис. Данный конкретный пример к тому же не использует не одну из конструкций языка Scheme, хотя производит достаточно сложные действия.

Хорошей демонстрацией использования alterator является реализации системы установки и конфигурирования дистрибутива ALT Linux 3.0. Инсталлятор системы состоит из трёх стадий, где две последних являются приложениями Alterator (в текущем ALT Linux 3.0 Compact вторая стадия ещё является самостоятельной программой). Интересно, что использование полноценного языка программирования для описания интерфейса позволило максимально использовать одни и те же диалоги конфигурования в трёх вариантах: как один из шагов инсталлятора, как один из модулей в Control Center, и как отдельное самостоятельно приложение.

16.15 – 17.00

Дмитрий Тараканов
Москва, Intel

Оптимизация производительности сервера MySQL*.

Рассказ о том, как инженеры компании Интел и MySQL* AB работают над увеличением производительность сервера MySQL*. Приводится анализ производительности сервера MySQL*, рассказывается о применении инструментов Интел для увеличения производительности, приводятся результаты работы.

17.00 – 17.30: Кофе

17.30 - 18.15

Федор Зуев.

GNU GPL как юридический вездеход

1) GNU GPL - жемчужина хакерского искусства

> hacker, n. [originally, someone who makes furniture with an axe]
> <...>
> 1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum

necessary.
<...>
6. An expert or enthusiast of any kind. One might be an astronomy hacker, for example.
(Jargon File (4.4.4, 14 Aug 2003))

GNU GPL важна для сообщества свободного софта во многих отношениях. И как “конституция свободного софта”, и как талантливый пропагандистский памфлет, и как юридическая защита разработчиков свободных программ.

И пример того, как хакерская культура и традиция порождают значительные результаты и за пределами собственно программирования. Можно сказать, что GNU GPL - программа, исполняемая на человеческом обществе. Успех GPL породил серию попыток использовать дозированную передачу прав как инструмент социального конструирования. С весьма скромными успехами.

В особенности это заметно в делах международных. GPL успешно функционирует в десятках стран с самыми различными юридическими системами. В от время как проект Creative Commons плодит тучи несовместимых между собой лицензий для каждой страны.

Поучительно поэтому изучить используемые GPL приемы. Не для создания собственных публичных лицензий - эта требуется нечасто, и чем реже, тем лучше - но чтобы уметь оценить качество многочисленных подражаний.

GPL избегает указаний на конкретные законы и юридические термины. Законы меняются от страны к

стране и от эпохе к эпохе, а GPL рассчитывает действовать везде и всегда. Поэтому все специфически юридические термины оттуда исключены, кроме единственного упоминания “derivative work under copyright law”. Вместо этого GPL излагает волю лицензиара в возможно более простых и общеупотребительных словах.

GPL щепетильно точно следует духу закона. Никакие попытки обойти закон, воспользоваться неясностью или крючкотворством, чтобы настоять на своем, здесь неприемлемы. *Лицензия строится на основе фундаментальных и неизменны принципов права, ограничиваясь требованиями, которые можно привязать к таким принципам.*

GPL по возможности избегает создания чисто договорных отношений между лицензиатом и лицензиаром. Каждое требование к лицензиату по возможности основывается не только на желании лицензиара, но и подкрепляется нормой закона.

2) GPL в контексте российского законодательства

В англосаксонских странах периодически вспыхивают жаркие дискуссии о том, является GPL “контрактом” или “лицензией” - различные правовые англо-американские формы. На самом деле GPL намеренно построена так, что может интерпретироваться как тем, так и другим образом.

В российском праве нет ничего похожего на американскую “лицензию”, так что GPL несомненно является договором. Точнее, *офертой* , предложением заключить договор, которое принимается пользователем в момент, когда он

станет производить действия, причисленные законом к исключительным авторским правам.

В интерпретации GPL по российским законам есть и ряд тонкостей. Например: каким образом приобретает силу “подпись” лицензиара? В отношении компьютерных программ здесь существует специальная норма относительно права на “особый порядок заключения договоров путем изложения типовых условий договора передаваемых экземплярах программ”. Однако для общего случая приходится пользоваться более общей нормой ГК о публичной оферте. Получается, что не-программы, распространяющиеся не-публично выпадают из обоих норм и вынуждены полагаться только на аналогию закона, то есть зависит от усмотрения и настроения судей.

При этом изготовители отдельных экземпляров свободных программ выступают в роли представителей лицензиара при заключении такого договора.

3) Надуманные и реальные проблемы GPL в России

Существует ряд расхожих мифов относительно проблем, возникающих у GPL под российской юрисдикцией. Большинство из них не имеют под собой реальных оснований.

“Российский закон о правах потребителей запрещает отказ от гарантийных обязательств” . Или еще какие-либо утверждения со ссылками на этот закон. На самом деле предметом регулирования являются “ возникающие между потребителями и

изготовителями, исполнителями, продавцами при продаже товаров (выполнении работ, оказании услуг)". Передача авторских прав, которая является предметом GPL, не принадлежит ни к одной из этих категорий.

"Договор в России должен быть на русском языке". На самом деле российское законодательство не говорит ничего о языке договора и вообще не требует использования естественного языка. Кассовый чек в магазине содержит строчки маловразумительных цифровых сокращений - но тем не менее является наиболее распространенной формой письменного договора.

С другой стороны, в российском законодательстве существует ряд действительных, скажем осторожно, шероховатостей в отношении GPL.

Закон РФ об авторском праве не предусматривает безвозмездную передачу прав. Согласно букве закона любая передача авторских прав предполагает вознаграждение автора, причем иногда для вознаграждения установлен нижний предел. Сочетание авторского договора с договором дарения осложняется тем, что право дарения между коммерческими организациями весьма ограничено.

Выглядит внушительно, но на самом деле от безвозмездных авторских договоров зависит настолько широкий сложившейся практике (Интернет, научная и периодическая печать, реклама) что фактическое запрещение их маловероятно.

Закон РФ требует явного перечисления каждого способа использования, права на которое

передаются и запрещает передачу прав на способы использования, неизвестные при заключении договора. Это значит, что спустя длительное время после опубликования произведения, распространение его через новые, неизвестные на момент публикации каналы может быть поставлено под сомнение.

Цели GPL вступают в конфликт с существованием и деятельностью "обществ по коллективному управлению авторскими правами". Правда, касается это пока в основном музыкальных произведений и фонограмм, которые под GPL лицензируются редко.

18.15 – 19.00

Александр Боковой
Москва, IBM LTC

Samba 4 - состояние, перспективы, реальность. Практическая демистификация

В 2003 году Samba Team начала работу над амбициозным проектом создания свободной полноценной реализации технологий Active Directory. Каково состояние проекта? Что следует ожидать в краткосрочной и долгосрочной перспективах?

Доклад представляет собой демонстрацию развертывания Samba 4 в качестве сервера ADS и интеграцию Windows XP в свободную реализацию Active Directory.

26 июля.

Утреннее заседание (9.30 - 14.00)

9.55 – 10.20

Александр Ковтушенко
МГТУ

Инструмент для визуализации трассы выполнения параллельной программы - TV 2.0

Оправданием затрат, понесенных при разработке параллельной программы, является достигнутое повышение скорости счета нашей задачи. При разработке параллельной программы используются методики предварительной оценки скорости параллельного счета. Сложность, «многофакторность» поведения проектируемого объекта ограничивают точность предсказания. Средством для нахождения узких мест, «доводки» параллельной программы является получение и анализ профиля выполнения параллельной программы. Одним из ценных средств компонент инструментальной среды, применяемых для этой цели, является связка: библиотека для автоматической (точнее - «почти автоматической») фиксации событий времени выполнения параллельной программы + диалоговая среда для просмотра/анализа накопленных данных.

Отдельным вопросом является выбор инструментальной среды для разработки параллельной программы. Критериями являются:

- Доступность инструментального программного обеспечения

- Предварительная оценка эффективности счета на доступном параллельном вычислителе нашей прикладной задачи в данной инструментальной среде,

- Трудоемкость кодирования и отладки

Архитектура параллельной ЭВМ определяет доступный набор инструментальных средств. Архитектура кластерных мульти-ЭВМ в настоящее время доминирует: раз в полгода публикуется отчет о 500 наиболее производительных ЭВМ (http://www.top500.org/), на данный момент последний 25-ый релиз состоит из кластеров на 60,8%. Важнейшим достоинством кластеров является их относительная дешевизна (на единицу пиковой производительности). В нашем Отчестве они представляют почти весь парк параллельных ЭВМ.

Кластер как правило содержит два уровня для организации параллельных вычислений:

Коммутатор, объединяющий вычислительные узлы, каждый из которых является серийно выпускаемым компьютером,

В каждом вычислительном узле работают несколько (как правило- два) процессора. Каким программным

обеспечением поддерживается работа кластера и разработка прикладного программного обеспечения? Основными инструментальными средами являются MPI и OpenMP.

Среда MPI (Message Passing Interface) основывается на открытом стандарте, разработанным открытым образом (общедоступен не только сам стандарт, комментарии-рекомендации к нему, но и рабочие документы MPI-Форума, протоколы постатейного голосования, обсуждавшиеся замечания, возражения). MPI предоставляет средство взаимодействия задач, т. е. вычислений выполняющихся в разных адресных пространствах. Стандарт исключает какое-либо скрытое, не включенное в предоставленный программисту API взаимодействие ветви параллельного процесса со средой MPI. Это позволяет эффективно переносить программы с одной реализации MPI на другую. Базовой реализацией MPI является MPICH, опубликованная под своей собственной открытой лицензией (не накладывается никаких ограничений). При этом широко распространены коммерческие продукты, ориентированные на аппаратные особенности коммутаторов. Например:

• шведская компания SCALI является интегратором, т. е. поставляет доработанную среду MPI, а также средства администрирования кластера.

• американская компания Myricom поставляет аппаратную часть (коммутатор) и адаптированную реализацию MPI.

Коммерческие продукты являются, как правило необходимым дополнением специализированного

коммутатора, управляемого непосредственно (без слоя TCP/IP).

Среда OpenMP основывается на открытом стандарте, разрабатываемом OpenMP Architecture Review Board. Разработка стандарта продолжается – версия OpenMP 2.5 от мая 2005 г. При этом OpenMP предоставляется средство выполнения параллельной работы над данными в общей памяти. В текст программы C/C++/Fortran вставляются «прагмы», обрабатываемые компилятором. Получаемый таким образом результат может быть заменен ручным кодированием параллельных нитей/легковесных процессов (за исключением каких-то аппаратнозависимых оптимизаций). Обработка OpenMP является как правило одной из опций компилятора, поддерживается последними компиляторами Intell, HP, SGI, IBM, Fujitsu, однако к сожалению не GCC. Однако, на основе университетских проектов доступны открытые решения:

- http://www.odinmp.com/

- http://phase.hpcc.jp/Omni/home.html

Разрабатываемый нами визуализатор является частью инструментальной среды MPI. Он позволяет выявить недостатки проектируемого параллельного алгоритма, выявить особенности функционирования коммутатора. Полученный опыт показывает, что латентность коммутатора существенно зависит от предшествовавших пересылок. Основным функциональным расширением TV 2.0 (текущая альфа версия) является сетевой режим, т. е. возможность удаленного просмотра трассы

параллельной программы. Интеграция его со свободными средствами среды OpenMP является нашей следующей задачей.

10.20 – 10.45

Сергей Гонтарев
Институт океанологии РАН

Необитаемый аппарат для подводных исследований.

При разработке морских исследовательских приборов в силу неполной определенности среды исследований невозможно заранее описать весь комплекс необходимых функций. В большинстве случаев набор функций, выполняемых прибором, уточняется и дорабатывается в процессе эксплуатации прибора. Требования к набору функций подводного аппарата могут существенно изменяться при работе на различных типах объектов. Ввиду сложности имитации полного набора факторов, воздействующих на подводный аппарат окончательная отладка должна проводиться только в реальной среде.

Один из вариантов построения комплекса для проведения подводных исследований необитаемым подводным аппаратом представлен в докладе. Отличительной особенностью данного аппарата является его специализация на проведении исследований. При проведении измерений для

получения максимального количества информации необходимо иметь возможность изменять условия проведения эксперимента, как в процессе отдельного эксперимента, так и при переходе от эксперимента к эксперименту. При этом каждый исследователь должен иметь возможность индивидуально настраивать свой комплекс приборов. Возможен режим совместного использования отдельных приборов. Повышение эффективности проведения исследований также может быть получено введением интеллектуального режима работы приборов. Система также должна позволять проводить быстрый экспресс анализ полученных данных, с целью своевременной коррекции условий проведения измерений. Для исследователя методы работы в системе не должны сильно отличаться от привычных.

Подводные аппараты используются, как правило, в двух конфигурациях. Первая для небольших глубин и малых скоростей течений. Вторая для глубин и течений, на которых становиться существенной парусность питающего кабеля. В первом варианте кабель опускается непосредственно с борта судна. Во втором варианте используется промежуточный бокс для спуска и подъема подводного аппарата. Бортовая часть, располагающаяся на судне, состоит системы питания, системы диагностики и тестирования, системы управления и системы сбора и отображения информации. При использовании промежуточного бокса в нем могут находиться часть систем питания и системы управления. Оборудование, располагающееся на подводном аппарате, состоит из системы питания (вторичные источники питания), системы управления приводами движителей, системы подсветки, системы видеокамер, системы внутренних датчиков аппарата и системы исследовательских

датчиков. Такой набор функций требует для своей реализации организации параллельного функционирования нескольких задач с одновременной синхронизацией режимов их работы. Возможны несколько подходов в построении аппаратной части подводного аппарата. Аппарат может быть построен с использованием микропроцессорной реализации отдельных функций с последующим объединением каналов управления в единый канал передачи данных. Его недостатком является необходимость написания значительного объема сервисных программ и сложность объединения такой системы со средствами получения и отображения данных. Построение на базе закрытых коммерческих продуктов, как правило, имеет ограниченную функциональность и сложности с изменением продукта под конкретный набор требований. Минимальная стоимость проекта получается при построении аппаратная части на широко используемом сетевом оборудовании. Систему управления подводным аппаратом проектируется как локальная сеть с соответствующей аппаратурой и программным обеспечением.

Программная реализация системы управления построена на программном обеспечении с открытым кодом. Функции, не изменяющиеся в процессе эксплуатации, могут быть написаны в виде отдельно запускаемых процессов. Интерфейс работы с часто изменяемыми датчиками может быть реализован как в виде отдельного процесса, так и в виде WWW приложений. Открытость кода в данном случае позволяет проводить доработку, отладку и модернизацию функций непосредственно на месте эксплуатации, что значительно сокращает сроки работ, повышает качество и информативность

исследований за счет более удобной эксплуатации прибора.

Подводный аппарат поставляет исследователю (группе исследователей) значительный объем информации. Наиболее удобным представляется использование привычных для исследователя средств первичной обработки информации. Как правило, в силу сложившихся привычек такие средства первичной обработки используются в среде Windows. Система управления, построенная на основе стандартных сетевых средств и программного обеспечения, обеспечивает возможность подключения в качестве рабочих мест машин, работающих под управлением Windows или Unix.

Подводный аппарат может быть оборудован датчиками для проведения нескольких экспериментов в одном погружении. Для удобства работы с информацией система должна обеспечивать возможность независимого получения данных и индивидуальной организации их обработки и отображения каждым исследователем, что требует возможности организации нескольких рабочих мест. Использование реализации в виде WWW приложений позволяет организовать просмотр их содержимого стандартными средствами просмотра, организовывать произвольное количество рабочих мест и изменять информацию, отображаемую на рабочем месте в соответствии с потребностями оператора. Средства отображения информации также как и датчики должны допускать легкую трансформацию в зависимости от выполняемых задач.

Поддержка системой управления аппарата сетевых

архитектур обеспечивает совместимость приборов и возможность их использования в составе информационных систем верхнего уровня. Например, возможно подключение к корабельной информационной сети или объединение через канал связи с береговой сетью.

10.45 – 11.10

Вадим Житников
ООО "Компания Скид"

Портирование свободного программного обеспечения на платформу Windows CE

Windows CE является одной из самых распространённых ОС на КПК, и такая ситуация, по-видимому, сохранится в обозримом будущем. Поэтому проблема портирования свободного программного обеспечения на эту платформу является актуальной.

Программирование для платформы Windows CE весьма похоже на программирование для Win32 - доступно подмножество MFC, сквозная поддержка unicode. Вместе с тем имеются определенные ограничения:

- Не полная стандартная C run-time библиотека
- Отсутствует понятие текущей директории и переменных окружения

- Отсутствует командная строка

- Отсутствует терминальный ввод-вывод с возможностью перенаправления

Эти ограничения серьезным образом усложняют перенос на Windows CE свободных программ, большинство из которых изначально предназначены для UNIX. Требуется создание дополнительных библиотек и консольных API.

Среди проектов портирования свободного программного обеспечения обеспечения на Windows CE в первую очередь следует отметить работу Rainer Keuchel (http://www.wince-devel.org). Проект основан на библиотеке собственной разработки celib.dll и консольной программы w32console. Переменные окружения эмулируются с помощью реестра.

Используется cross-компилятор gcc 3.0.3 с целевыми архитектурами arm, mips и sh3, и binutils 2.11.2. В рамках данного проекта осуществлен успешный порт многих программ:

Vim, Emacs, Perl, Tcl/Tk, GCL, Maxima, gnuplot, rsync, Apache, XFree86 и др. Несмотря на то, что самые поздние сборки были выполнены в 2001-02 гг. для Windows CE 2.11 и 3.0 большинство программ вполне работоспособны даже в среде Windows Mobile 2003 SE (Windows CE 4.2). Существует активный форум http://groups.yahoo.com/group/wince-devel/ посвященный данному проекту и смежным вопросам.

В более поздних проектах широко используются библиотека newlib (http://sourceware.org/newlib/) и консольная программа PocketConsole/PocketCMD

(http://www.symbolictools.de/public/pocketconsole/). Newlib C библиотека, ориентированная на использование во встроенных системах. PocketConsole предоставляет API для терминальных програм, а PocketCMD реализованная на этой основе командная оболочка, близкая по возможностям к Windows NT cmd.exe.

Проект Voxware (http://win-ce.voxware.com:28575/Development% 20Tools/gnuwince.html) включает Linux cross-компилятор gcc 3.3, binutils 2.13.91, newlib 1.11, имеется bash-подобная командная оболочка и набор утилит ps, kill, mkdir, cp, mv и т.д. Сервер rlogind позволяет удалённо использовать ush с любого rlogin клиента, если КПК подключен к сети.

PocketGCC Виталия Пронкина (http://pocketgcc.sourceforge.net/) является native-компилятором gcc 3.2 и binutils 2.13. Другой проект данного автора Pocket C# - порт C# компилятора DotGNU (http://pocketgcc.sourceforge.net/pcsharp/).

Проект Mamaich (http://mamaich.kasone.com/fr_pocket.htm) включает cygwin cross-компилятор gcc 3.3.3 и native-compiler, binutils 2.13.2.1, gdb, newlib 1.9, pthereads, SDL. Используется PocketConsole.

Source Forge проект GNUDE (GNU Development Environment, http://gnude.sourceforge.net/) ставит целью создание полного набора GCC/binutils, включая C, C++, FORTRAN, Java cross-компиляторов, отладчиков GDB/Insight и сопутствующих утилит для разработки приложений для архитектуры ARM. В данный момент доступны

Windows (cygwin) и Mac OS X cross-компиляторы.

Отметим, что ряд проектов специально предоставляют сборки своих программ для платформы Windows CE. Например: Perl CE (http://perlce.sourceforge.net/), TclKit Mobile

(http://wiki.tcl.tk/TclkitMobile), wxWidgets

(http://www.koansoftware.com/it/prd_svil_wxwince.htm)

Несмотря на видимое обилие проектов, ситуацию с портированием свободного программного обеспечения на платформу Windows CE нельзя признать удовлетворительной. Некоторые проекты оставлены разработчиками, статус других не вполне ясен, общее число разработчиков невелико.

11.10 – 11.35

Антон Качалов
Москва, ALT Linux

Многоплатформенность в ALT Sisyphus

Многие Linux-ориентированные компании и дистрибутиво-строители осуществляют выпуск и поддержку решений и дистрибутивов под платформы, отличные от ix86. До недавнего времени, Сизиф и его производные могли функционировать только на платформах семейства Pentium и старше. Ситуация переломилась с появлением платформы x86_64. Это 64-х битная платформа, имеющая обратную совместимость с 32-х битной Intel-архитектурой. То есть перед нами представитель так

называемой BiArch-архитектуры.

Этапы внедрения новой платформы.

Первым этапом рождения новой платформы - это формирование минимального инструментария для сборки ядра и системных библиотек. Для этого необходимо собрать программы для кросс-компиляции. Традиционно, сюда входят: binutils, gcc, glibc. Это так называемый Toolchain. Причём, gcc собирается несколько раз - сначала только с поддержкой C, а потом, после сборки glibc, с поддержкой C++. Предпочтительнее всего собрать минимум, необходимый для загрузки системы, а дальше расширять набор программ в нативном окружении, если это позволяет конечная платформа.

Вторым этапом идёт сборка RPM, а далее минимального списка пакетов, необходимых для дальнейшей сборки в hasher'e.

Третий этап - этап добавления патчей, налаживание автоматизированной пересборки новых пакетов и синхронизация по отставшим пакетам с Сизифом.

Проблемы и их решения.

С BiArch-архитектурами возникает больше проблем в виду того, что необходимо поддерживать работу как и 64-х битных программ, так и 32-х битных в одной системе. Первый шаг по разрешению данной проблемы был добавление семейства директорий для 64-х битных библиотек и архитектурно-зависимых файлов. Как правило, это: /lib64, /usr/lib64, /usr/X11R6/lib64 и т. д. Данное решение сопряжено со множеством проблем, возникающих при пересборке

пакетов под данную архитектуру, так как многие библиотеки и программы не расчитаны на использование lib64. Во многом это решается небольшими исправлениями в Makefile'ax и configure-скриптах. Реже требуется исправление исходного кода программ.

Кросс-компиляция несёт в себе множество неприятных сюрпризов. Проблемы с кроссом в случае x86_64 начинали возникать ещё на этапе сборки полноценного toolchain, что только ещё больше подталкивало на перенос всей сборки в нативную среду.

Сборка RPM-пакетов несёт в себе массу проблем по сборочным зависимостям. Многие пакеты имеют кольцевую зависимость. И зачастую приходится собирать пакеты нечестным образом, отключая те или иные части пакета, или вмешиваясь в сам процесс сборки, подменяя или редактируя что-либо. Но эти проблемы существуют ровно до тех пор, пока не будет создана минимальная самодостаточная пакетная база.

11.35 – 11.50: Кофе

11.50 – 12.15

Алексей Гладков
Москва, ALT Linux

Новые технологии в проекте Sisyphus

В докладе рассказывается об изменениях произошедших за прошедший год,а также о дальнейших планах по автоматизации работы.

Проект incominger

Год назад на этой же конференции была анонсирована автоматическая система проверки пакетов направляемых в репозиторий сизифа. За этот год система пережила несколько перерождений. Были изменены основные алгоритмы, расширен функционал.

Проект sisyphus растёт и увеличивается количество пакетов каждый день приходящих в incoming. Последовательная проверка такого пакетов занимает очень много времени. Чтобы решить эту проблему incominger был перепроектирован так, чтобы все независимые операции производились параллельно на нескольких серверах.

Для того чтобы иметь возможность проверять пересобираемость пакетов был написан новый алгоритм для создания очереди пересборки. Этот алгоритм основывается на сборочных зависимостях исходных пакетов. Также учитываются пакеты находящиеся в incoming, но еще не попавшие в репозиторий.

Основная задача этого алгоритма - это формирование списка пакетов из числа не пересобранных пакетов, которые можно попытаться собрать и они не зависят друг от друга.

Прежде чем описать алгоритм, нужно сформулировать несколько условий, которые накладывает rpm:

- До момента сборки невозможно узнать (очень трудно, если хотите) какие бинарные пакеты получатся из данного исходного.

- Для сборки исходного пакета в сборочной системе должны быть все бинарные пакеты, указанные в сборочных зависимостях (BuildRequires).

Исходя их этих условий становится ясно, что построить граф зависимостей по исходным пакетам нельзя.

Такая ситуация очень печальна, но из нее есть выход. Если нам не удастся определить порядок заранее, можно решать проблемы по мере их поступления. Основная идея в том, чтобы определять какие исходные пакеты можно пересобрать в данных момент времени. Для этого помимо репозитория sisyphus, нам понадобится еще один (внутренний) репозиторий. В нем будут находится пересобранные пакеты.

Алгоритм вычисления очереди выполняется в цикле после каждой очередной сборки пакетов до тех пор, пока на очередной итерации не соберется ни один пакет. Это означает, что мы достигли "стабильного" состояния и оставшиеся пакеты нельзя собрать для

этих репозиториев (внутреннего и внешнего).

Минус такого алгоритма состоит в том, что мы не можем предвидеть ход сборки более чем на один шаг вперед. Это затрудняет отслеживание циклических зависимостей.

Начиная с версии 0.0.8 в incominger добавлена возможность параллельной проверки пакетов под несколько платформ одновременно.

Для удобства разработчиков проекта sisyphus incominger хранит и публикует все логи проверок. Логи от пакетов разделены на две группы:

- Логи от пакетов, прошедших все проверки и добавленные в репозиторий;

- Логи от пакетов, не прошедших в сизиф.

Планы

Планируется добавить проверку пакетов на создание неудовлетворенностей в репозитории. Также планируется добавить алгоритмы направленные на автоматическое устранение неудовлетворенностей. К сожалению не всегда можно установить виновника создания неудовлетворенностей и тем более автоматически устранить проблему.

Планируется добавить возможность создания веток (branch) репозитория sisyphus. Это позволит формировать срезы проекта основывающиеся на определенном критерии. Как частный случай можно рассматривать создание дистрибутива.

Также планируется создание интеграция incominger с системой отслеживания ошибок. К сожалению, эта интеграция требует принятия некоторых соглашений среди членов команды.

Проект incominger-dude

Этот проект находится в стадии разработки и предназначен для облегчения и ускорения работы членов команды разработчиков.

Основная цель этого проекта - это предоставление простого и удобного интерфейса к incominger. На данный момент планируется создание только mail-интерфейса.

Через этот интерфейс мантейнеры могут вносить изменения в свои пакеты на стороне сервера (incoming). Например, будут доступны следующие команды для incominger:

- Пересобрать пакет;
- Изменить версию, релиз и changelog и пересобрать пакет;
- Собрать пакет X и если он соберется, то пересобрать все пакеты кому он нужен.

Таким образом, будет возможность задавать некоторую простую логику действий.

12.15 – 12.40

Петр Савельев
JSC Eltel

GNU RAD/Linux как пример разработки дистрибутива на базе ALT Sisyphus

1. Введение

Разнообразие оборудования и программных средств, с которыми приходится иметь дело системным и сетевым администраторам Internet-провайдеров часто создаёт сложности в работе. Одним из возможных решений является использование сетевых устройств одного производителя. Другим решением может быть создание решений с однотипным интерфейсом.

Данная работа является иллюстрацией второго подхода. Решение должно предоставлять базовые возможности маршрутизации, фильтрации и учёта трафика, а также авторизованного доступа клиентов к сети. Также планируется использовать решение для организации виртуального хостинга.

2. Решения и поддержка

В качестве операционной системы была выбрана ОС GNU/Linux, как из-за лицензирования так и из-за удобства разработки специализированных решений на её базе. Значительную роль в выборе сыграли также возможности ОС GNU/Linux в области маршрутизации и коммутации трафика.

Разработка и поддержка такого рода решения подразумевает использование широкого круга ПО, которое, будучи доступным по условиям лицензии GPL, тем не менее, требует усилий по отслеживанию новых версий и исправлений. Возможным вариантом

является сборка с опробованными версиями программ и отказ от модернизации решения. Такой вариант хорош тем, что работоспособность решения протестирована, решение является типовым и никаких неожиданностей в его работе в штатных ситуациях не предвидится. Минусом является потенциальное наличие в ПО уязвимостей и ошибок, неизвестных на момент сборки. Позднейшее появление информации по таким уязвимостям поставит под удар все инсталляции решения, а приём «security by obscurity» в данном случае неприемлем как из-за использования открытого ПО, так и в силу небольшой эффективности данного подхода.

Другой вариант – самостоятельная сборка новых версий и тестирование их на совместимость с уже используемым ПО. Однако, такие задачи и решаются мантейнерами дистрибутивов и сообществами пользователей GNU/Linux, обычно в рамках политики того или иного дистрибутива. Было бы неразумно не воспользоваться результатами их работы и их опытом . И это создаёт предпосылки для третьего варианта. То есть, для разработки «дочерних дистрибутивов» на базе уже собранных и протестированных пакетов.

Использование разнообразного ПО в рамках одного решения обычно подразумевает наличие зависимостей между отдельными программными пакетами. Среди инструментов по отслеживанию подобных зависимостей одним из старейших и удачных является apt (Advanced Package Tool), разработанный командой Debian. Работа apt требует наличия репозитариев (хранилищ) собранных пакетов, и на данный момент такие репозитарии есть для многих крупных дистрибутивов GNU/Linux.

3. Коротко о самом проекте

На данный момент RAD GNU/Linux представляет

собой небольшое монолитное решение. Большинство базовых утилит предоставляет пакет Busybox. Шелл (rt-shell) написан с использованием GNU awk и пакета rlwrap, в качестве примера был принят удачный подход с контекстной помощью при автодополнении, используемый в устройствах Cisco (r) и Juniper (r). Настройка RAD GNU/Linux осуществляется утилитой rt-networkc помощью файла (ов) конфигурации, также имеющем Cisco-like формат. Обе утилиты написаны в рамках проекта RAD GNU/Linux, однако могут использоваться и вне его.

Одним из нечасто встречающихся решений является использование vserver для управления штатными сервисами. Для каждого сервиса (ntpd, httpd, dhcpd и так далее) создаётся свой контекст выполнения. Это является дополнительным фактором в обеспечении безопасности, так как с помощью vserver можно ограничивать очень многие параметры, начиная от привилегий и кончая квотами на дисковое пространство и вычислительные ресурсы для каждого контекста. Также это сильно облегчает управление сервисами. Например, для остановки сервиса не нужно иметь корректный pid-файл и не нужно вычислять всех его потомков. Достаточно остановить все процессы заданного контекста.

4. Заключение

В качестве базы для сборки решения был выбран ALT Sisyphus. Основными доводами стали большой выбор, большая оперативность по обновлению пакетов и довольно высокое качество тестирования. Особое внимание команда ALT уделяет безопасности ПО, что также часто является плюсом.

Однако существуют ситуации, когда представленного в Sisyphus ПО недостаточно, что требует создания собственного репозитария. И такой репозитарий в

рамках ALT сделать также легко, а специфика сборки rpm-пакетов делает создание собственных пакетов не сильно обременительным. Сборка решения осуществляется скриптом на базе проекта separator Антона Фарыгина.
В заключение хочется отметить, что на данный момент решения ALT могут служить удобной базой для создания собственных решений, предоставляя всё необходимое для сборки и облегчая задачи по обновлению ПО.
5. Ссылки
ALT Sisyphus –
http://www.altlinux.ru/index.php?module=sisyphus
Busybox – http://www.busybox.net/
Cisco (r) – http://www.cisco.com/
Juniper (r) – http://www.juniper.net/

12.40 – 13.05

АнатолийЯкушин

Раиль Алиев
Инфраресурс

Проект: OpenOffice.ru

OpenOffice.org 2.0 - новая версия, новые возможности

Доклад посвящен особенностям новой версии свободного офисного пакета OpenOffice.org и новому свободному формату электронного документа.

13.05 – 13.30

Михаил Пожидаев
Томский Госуниверситет

Обзор систем для работы в среде GNU/Linux без зрительного контроля

Аннотация:

Доклад является обзором средств для работы в системе GNU/Linux людей с ослабленным зрением. Рассматривается применение подобного программного обеспечения зарубежом, а также состояние дел и существующие проблемы для пользователей России. Большое внимание уделено синтезаторам речи, пригодных для использования в среде GNU/Linux.

Всё программное обеспечение, которое может понадобиться человеку с ослабленным зрением, пригодное для использования в среде GNU/Linux, делится на 3 группы, каждая из которых заслуживает отдельного рассмотрения:

1. пакеты для снятия экранной информации, так называемые, screen reader'ы;

2. речевые синтезаторы;

3. пакеты для взаимодействия screen reader'ов и речевых синтезаторов.

Ветераном среди пакетов для снятия экранной информации является пакет emacspeak. Он на сегодняшний день - самая развитая среда для использования незрячим пользователем. По своей сути это дополнение в оболочке GNU Emacs, написанное на языке lisp. В некоторой мере его популярность объясняется универсальностью среды GNU Emacs.

В последнее время появилось два проекта, предназначенных для перехвата всей информации, выводимой на экран в терминальном режиме.

Это пакеты Speak Up и YASR. Пакет Speak Up выполнен в виде патча к ядру Linux. Он статически встраивается в операционную систему и перенаправляет текстовую информацию в специально зарегистрированное устройство для дальнейшей обработки.

Пакет YASR (Yet Another Screen Reader) сделан как маленькая, хорошо переносимая программа, порождающая виртуальный терминал и посылающая весь появляющийся в нем текст в речевой синтезатор. Такие программы удобны для работы с командной строкой, но совершенно непригодны, например, для редактирования текста.

Долгое время работе пользователя в графических средах без зрительного контроля не уделялось внимания. Но недавно был открыт проект Gnopernicus, предназначенный для работы в среде GNOME, и уже достигший заметных результатов.

Поддержка среды KDE пока не выходит за рамки общих рассуждений о потенциальной возможности работы незрячего пользователя.

Речевых синтезаторов, предназначенных для функционирования в среде GNU/Linux, и способных генерировать англоязычную речь не так мало, как это может показаться на первый взгляд. Здесь ситуация осложняется тем, что что их применение ограничено условиями распространения и использования. Среди синтезаторов, распространяемых на условиях GPL, нужно отметить системы Festival и Flite. Речевой синтезатор Festival изначально разрабатывался группой программистов из университета в Эдинбурге. Помимо самого синтезатора, ими был разработан целый пакет для работы с речью, но приложение получилось довольно неповоротливым и неудобным для практического использования. В настоящее время разработка заброшена. Другой, свободно распространяемый синтезатор Flite, значительно гибче предыдущего, но к его недостаткам относится плохое качество генерируемой речи.

Самым крупным и удачным пакетом для синтеза речи является синтезатор Via Voice компании IBM. Долгое время этот синтезатор был свободен для использования, но два года назад компания IBM запретила его открытое распространение и использование.

Одним из средств, подающим большие надежды конечному российскому пользователю, является связка FreeSpeech + Mbrola. По своей сути это два проекта. FreeSpeech занимается разложением текста на фонетические составляющие, а пакет Mbrola

производит связывание звуков речи. FreeSpeech полностью открыт, и его использование ничем не ограничено, но у Mbrola нет исходных текстов, и вопрос об его распространении в составе дистрибутивов нужно оговаривать отдельно с разработчиками. В общем же случае, он свободно может быть скачан с сайта разработчиков.

Такое не очень утешительное положение вещей в мире синтеза речи имеет свои причины. В западных странах и США широко распространены аппаратные синтезаторы речи, которые представляют собой отдельное устройство, соединённое с компьютером через внешний порт. Зачастую у зарубежных пользователей нет никакой потребности в программном синтезаторе.

Пакеты YASR, Speak Up перенаправляют речевую информацию напрямую в порт синтезатора.

Пакет Emacspeak подразумевает наличие отдельного компонента для обработки речи – речевого сервера, в который передаётся текстовая информация. Сам пакет Emacspeak обработкой речи не занимается.

Для российских пользователей встаёт вопрос о разработке специального речевого сервера, поскольку необходимо различать обработку английской и русской речи. Единственным примером синтезатора для обработки русской речи является синтезатор ru_tts, который существует в варианте “как есть”, т. е. без наличия исходных текстов и каких-либо комментариев относительно условий его распространения.

В начале 2001 года был распространён диск

Slackspeak, автором которого был Игорь Порецкий. Этот диск представлял собой вариант дистрибутива Slackware 7.0 с подготовленными для работы пакетами Emacspeak, FreeSpeech, Mbrola и ru_tts, но также без комментариев относительно легальности распространения синтезатора ru_tts и возможности его дальнейшего использования.

13.30 – 13.55

Георгий Курячий
Москва, ALT Linux

Средства разработки "типовых решений": утопия и реальность

Конфигуратор - это средство быстрой (типовой, автоматической, непрофессиональной и т. п.) настройки операционной системы. Подобные средства есть во всех ОС, однако способ организации дистрибутивов Linux предъявляет к конфигуратору особенные требования. Полнофункциональные конфигураторы появились в Linux не сразу, а только после того, как настройка системы и служб перестала быть разовой высокопрофессиональной работой и перешла в руки **операторов**, а не администраторов системы.

Первыми возникли установщики - программы, позволяющие установить и первоначально настроить систему (такая работа всегда была кропотливой, а кроме того, большинство параметров настройки можно вычислить автоматически). После того, как в работе информационных систем начали складываться

стереотипы - "типовые" службы, решающие "типовые" задачи, понадобились и конфигураторы, понимаемые, с одной стороны, как готовые решения этих задач, а с другой стороны - как инструмент разработки таких решений. Наконец, с увеличением компьютерного парка возникла необходимость **масштабирования** действие по настройке системы (например, однократная настройка и перенастройка профиля всех рабочих станций машинного зала).

В то время, как UNIX-системы, развивающиеся по модели ПО ЗК (программное обеспечение с закрытым кодом), легко решали задачу обновления установщика с каждым выпуском дистрибутива, Linux-системы и подобные им, основанные на ПО ОК (программное обеспечение с открытым кодом), столкнулись с трудностями: работа по постоянному воссозданию конфигуратора силами одной рабочей группы очень трудоёмка. Такая работа требует усилий сразу многих специалистов: системного аналитика, системного администратора, программиста (в том числе, знакомого с программированием пользовательского интерфейса), дизайнера и т. д. Сложность конфигуратора пропорциональна сложности дистрибутива.

Следовательно, конфигуратор Linux-системы должен создаваться с активным привлечением всего сообщества, причём самостоятельное расширение функциональности должно быть доступно в первую очередь членам сообщества, системным администраторам на местах. Коротко говоря, конфигуратор будет эффективен только тогда, когда, доверя некоторую обязанность пользователю (обычному оператору), обычный системный администратор предпочтёт воспользоваться не

shell/Perl, а средствами имеющегося конфигуратора, так как с их помощью задачу решить будет легче и быстрее.

Какие требования предъявляются к идеальному конфигуратору ПО ОК?
Возможность создания пользовательской объектной модели. Пользователь не обязан знать тонкости *реализации* системы, но само *решение* его задачи должно быть описано в *пользовательских* терминах. Это самое сложное требование. Несоблюдение его ведёт либо к просачиванию в пользовательский интерфейс узкопрофессиональных терминов (при этом пользователь может отказаться решать задачу вообще), либо к потере гибкости (одна кнопка - один результат). Кроме того, внутри конфигуратора необходимо постоянно отслеживать соответствие состояния системы в терминах объектной модели действительному положению дел в системе, производить синхронизацию, избегать "тупиков" и невосстановимых потерь данных.
Основные типовые задачи должны быть либо *уже* решены с помощью конфигуратора, либо решаемы в два-три приёма. Задачи более сложные или нестандартные также должны быть решаемы, причём сложность решения не должна нарастать лавинообразно при линейном росте сложности задачи. Подступая с таким требованем к неидеальному конфигуратору, следует прежде всего определить, какой класс задач с его помощью не решается или решается слишком большими усилиями.
Основное требование к конфигуратору ПО ОК - возможность привлечения сообщества к его разработке. Оно распадается на три:
Простота доработки. Следует учитывать, что

основные потребители конфигуратора как инструмента разработки - системные администраторы. Их рабочее время ограничено, и, вдобавок, от них нельзя требовать умения помногу и безошибочно программировать на различных языках. Так что, во-первых, для добавления функциональности в конфигуратор хотелось бы не писать слишком много программного кода, особенно если он не относится непосредственно к решению задачи. Во-вторых, для этого должно быть достаточно знаний системного администратора плюс вменяемого по размерам руководства.
Конфигуратор должен быть продуманно и внятно документирован. Это общее для любого ПО требование имеет здесь особую важность: необходимо доходчиво изложить "идеологию" и архитектуру конфигуратора, так как от этого зависит, насколько модуль, написанный сторонним человеком из сообщества, окажется пригоден к общему использованию. Важно также создать ряд методичек и примеров написания таких модулей.
Конфигуратор должен чётко делиться на модули, как отвечающие разным функциональностям системы (горизонтальное деление), так и реализующие различные по уровню объектные модели (системную, пользовательскую и интерфейсную). Первое требование позволит свободно добавлять новые функциональности, а второе - разделять (возможно, между различными членами сообщества), работу по написанию пользовательского интерфейса, логики решения пользовательской задачи и по конкретной настройке системы.

Нами было рассмотрено три с половиной дистрибутива Linux и их конфигураторы: SuSE Linux 9.1 и YaST2, Debian Sarge и DebConf, Red Hat

Enterprise Linux 4 AS и system-config, а также находящийся ко времени исследования на стадии разработки ALT Linux 3.0 Compact и ALTerator. Вкратце результаты исследования по приведённым выше критериям выглядят так.

YaST2 Наиболее "далеко ушедший" в плане возможностей конфигуратор. Многие из "готовых решений" и в самом деле уже готовы. В SuSE разработан специальный язык программирования (YCL), позволяющий быстро описать и интерфейс и логику работы отдельного модуля. Системные ресурсы, с точки зрения YCL, унифицированы, так что никакими операциями по настройке, кроме *read, write* и *execute*, программисту пользоваться не надо. Системный уровень не только отделён от пользовательского, но и соединён с ним специальным "трубопроводом" (SCR), что позволяет запускать YaST2 на одной машине, а настраивать любую другую! YaST2 и YCL прекрасно документированы. Однако написание модуля для YaST2 остаётся делом довольно трудоёмким, так как требует одновременно всех упомянутых нами в начале знаний. Все авторы модулей YaST2 - сотрудники SuSE.

DebConf Наиболее "многообещающий" конфигуратор, возможности которого до конца не осознаны. Произошёл от естественного стремления не задавать пользователю одни и те же вопросы повторно, раз уж его уже спрашивали (например, при установке системы). DebConf предоставляет кеш таких вопросов/ответов и другие возможности манипуляции ими, несколько видов автоматически оформляемого интерфейса и **очень** простой текстовый протокол обмена данными. В результате DebConf-ом пользуются **все** сопровождающие пакеты

в Debian (наличие конфигурационного сценария, работающего через DebConf - часть дисциплины Debian). На самом деле это означает и горизонтальное (вопро-ответ - пользовательская модель, результат работы сценария - системная),и тем более - вертикальное модульное деление. Очень остроумно решается проблема масштабирования и автоматической настройки: используется "никакой" интерфейс, то есть настройка системы идёт только на основании данных из кеша ответов. Документация и примеры по DebConf весьма толковы. Разработка логики на верхнем, пользовательском, уровне в DebConf пока находится в зачаточном состоянии (Debian - сообщество профессионалов), но единственным "родовым" недостатком DebConf можно назвать только невозможность сочинять сложные интерфейсные модели. В этом направлении сообщество просто ещё не думало...

system-config Конфигуратор в "старом стиле" - не рассчитанный на сообщество, с множеством недостатков на нестандартных задачах и в нестандартных условиях, совершенно не документированный технически. Единственное достоинство: основные "операторские" задачи в самом деле решены, так что дыра залатана.

ALTerator На время исследования находился в состоянии разработки. Использует изначально модульную технологию и разделение интерфейсной, пользовательской и системной моделей. Связь между модулями может осуществляться с помощью простого текстового протокола, а сами модули можно писать на каком угодно языке программирования (в YaST2 это верно только для модулей нижнего уровня). Активно используется язык программирования Scheme, что, с одной стороны может усложнить жизнь системным

администраторам, но, с другой стороны, упрощает написание интерфейсных модулей, что, учитывая горизонтальное деление, системным администраторам делать (может быть) и не придётся. "Тёмная лошадка" среди конфигураторов: при надлежащем развитии из него может выйти почти идеальный (правда, не такой простой, как DebConf), при ненадлежащем - что угодно.

14.00 – 14.45: Обеденный перерыв

Дневное заседание (14.45 - 17.00)

14.45 – 15.10

Алексей Федосеев
ООО "Айя"

Использование и модификация проекта User Mode Linux для организации хостинга на основе виртуальных выделенных серверов

Основанием для данной работы явилась необходимость организации набора виртуальных серверов на одной машине. Такая задача возникает, например, в случае организации виртуального выделенного хостинга — пользователю выделяется собственный полноценный сервер, в рамках которого он обладает правами суперпользователя; для операционной системы же этот сервер является отдельным процессом или набором процессов.

К системе организации виртуальных серверов в этом случае предъявляются следующие требования:

• изолированность виртуальных серверов друг от друга;

• встроенный механизм ограничения использования ресурсов одним сервером;

• поддержка виртуальных сетевых интерфейсов;

• наличие средств администрирования.

Одним из решений, удовлетворяющих данным требованиям, является проект User Mode Linux (UML). Целью проекта, по сути представляющего собой модификацию ядра Linux, является запуск ядра операционной системы в качестве отдельного пользовательского процесса.

Запуск виртуальной машины с привилегиями простого пользователя позволяет настроить доступ виртуальной машины только к необходимым ресурсам системы. Виртуальные диски представлены в виде файлов, возможно создание Copy-on-Write образов дисков и проецирование директорий главного сервера в структуру каталогов виртуального. Виртуальные сетевые интерфейсы можно организовать несколькими способами, в том числе через TUN/TAP. Виртуальные консоли запускаемой системы могут быть связаны с файлами, файлами устройств терминалов и даже на TCP-портами.

Данная работа не претендует на роль сравнительного обзора существующих средств организации виртуальных выделенных серверов[1] . Однако, было бы интересно сравнить возможности UML с некоторыми широко распространенными аналогами, среди которых были выделены два: FreeBSD Jail и vserver linux.

1А таких систем, начиная от простых разработок и заканчивая серьезными коммерческими, например Virtuozzo, достаточно много.

Критерий оценки	UML	vserver	jail
Операционная система	Linux	Linux	FreeBSD
Виртуальные сетевые интерфейсы	+	+	+
Уровень прав суперпользователя	+	+	+
Возможность перезагрузки	+	-	+
Возможность смены дистрибутива	+	-	-
Специальное ядро хост-системы	нет	да	нет
Доступ к файловой системе хоста	+	–bind	unionfs
Права суперпользователя для запуска	нет	да	нет

В таблице представлены результаты сравнения функциональности рассматриваемых решений. Видно, что проект UML не уступает аналогичным существующим решениям.

Для администрирования виртуальным выделенным сервером на базе UML, а также для расширения его функциональности применяется набор дополнительных патчей:

• консоль управления — специализированная программа, взаимодействующая с виртуальным сервером и способная управлять им посредством набора команд (таких как start, stop, pause, proc — посмотреть содержимое файла в директории /proc, sysrq — отправить прерывание и т. п.);

• токены ввода-вывода — дополнительное средство ограничения виртуального сервера — подсчитывается число запросов к дискам, которое не должно привысить определенной в секунду величины.

• вывод ошибок ядра в поток stderr.

В рамках представляемого проекта эти и другие средства управления виртуальным сервером были структурированы и объединены в общий интерфейс администрирования. Этот интерфейс представляет собой комплекс утилит для управления и получения статистической информации о работе набора виртуальных серверов. В его состав входят программы, реализующие следующие функции:

• добавление новых и удаление существующих виртуальных серверов;

• запуск, останов и временное прерывание работы виртуального сервера;

• мониторинг состояния исполнения и уровня загрузки системы;

• простейшая система балансировки нагрузки (nice scheduler);

• комплексный файервол (организованный на базе iptables), задающий параметры доступа к виртуальным серверам по виртуальным сетевым интерфейсам;

• система централизованного бэкапа;

• программа для входа в виртуальную систему через одну из консолей.

Комплекс скриптов написан на языке Ruby.

Разработанное решение на основе проекта User Mode Linux позволяет построить достаточно гибкую систему виртуального выделенного хостинга и успешно применяется в работе нашей компании. Исходные тексты программ и комплект документации для администратора в настоящий момент готовятся к публикации под лицензией GPL в сентябре этого года.

15.10 – 15.35

Андрей Столяров
МГУ им. Ломоносова, ВМК

Библиотека InteLib - инструмент мультипарадигмального программирования

Аннотация

Доклад посвящен практическому решению проблемы интеграции выразительных механизмов языков программирования принципиально различной природы в рамках одного проекта. Дается краткая классификация существующих подходов и указываются их недостатки. В качестве решения проблемы предлагается подход, получивший название метода непосредственной интеграции, и описывается библиотека InteLib, позволяющая, не выходя за рамки языка C++, записывать выражения, семантически эквивалентные и синтаксически близкие конструкциям языка Lisp (а в перспективе – и других языков, таких как Рефал, Prolog, Datalog и др.)

Все многообразие существующих в настоящее время языков программирования можно при желании классифицировать по признаку господствующих

традиций осмысления программы, принятых в сообществе программистов, пишущих на данном языке. Так, традиционные императивные языки стимулируют осмысление программы как набора приказов по изменению тех или иных данных; объектно-ориентированные языки позволяют представлять программу как набор абстрактных объектов, обменивающихся сообщениями; при работе на языках логического программирования программа представляется как система фактов и аксиом, а исполнение программы состоит в попытке опровержения заданного утверждения; наконец, функциональные языки программирования описывают программу как набор функций, вычисляющих значение по заданным аргументам, а выполнение программы представляется вычислением значения некоторого выражения.

Менее фундаментальные особенности языков программирования также накладывают определенный отпечаток на мышление программиста; прекрасный пример этого приводит Тимоти Бадд в книге [1].

Решение вопроса о "<наиболее правильном"> или "<наиболее удобном"> образе мышления существенно зависит от решаемой задачи. Так, на языке Lisp удобно обрабатывать формульные и другие слабо структурированные данные, но до крайности неудобно строить диалоговые системы на основе событийно-ориентированных оконных интерфейсов, а для языка Java дела обстоят ровно противоположным образом.

Типичной является ситуация, когда в силу тех или иных причин для реализации крупного проекта

выбирается тот или иной язык программирования (чаще всего выбор падает на императивный или императивно-объектный язык), при этом в проекте возникают небольшие подзадачи, которые было бы гораздо удобнее решать на совершенно другом языке.

Традиционные средства интеграции разнородных парадигм программирования можно условно разделить на несколько основных подходов:

1. создание нового языка программирования (возможно, как расширения одного из существующих)

2. встраиваемые интерпретаторы

3. расширяемые интерпретаторы

4. трансляция из одного языка в другой

5. пакеты взаимосвязанных программ

6. сборка объектных модулей, полученных из разных систем программирования

7. реализация частных возможностей

Эти подходы подробно анализируются в работе [2]. К сожалению, каждый из них имеет определенные недостатки, существенно ограничивающие области их применения.

Являющаяся основным предметом рассмотрения данного доклада библиотека InteLib [3] реализует качественно иной подход к решению проблемы

интеграции разнородных языковых выразительных средств.

Благодаря поддержке в языке C++ концепции абстрактных типов данных и возможности переопределения символов стандартных операций для объектов пользовательских типов язык C++ может рассматриваться как язык *моделирования алгебр*. С другой стороны, семантика языка Lisp может рассматриваться как *алгебра S-выражений*, одной из операций которой является *вычисление* S-выражения.

Чтобы представить конструкции языка Lisp средствами C++, достаточно описать класс, способный хранить (инкапсулировать) S-выражение любого поддерживаемого типа и дать набор операций для построения точечных пар и списков из атомарных S-выражений. Используя свойство полиморфности объектов, можно построить иерархию классов с общим предком для хранения различных типов S-выражений, что позволит вводить новые типы S-выражений.

В результате получаем возможность записи в языке C++ выражений, семантически эквивалентных и синтаксически близких конструкциям языка Lisp (или другого моделируемого языка).

```
(defun isomorphic (tree1 tree2)
 (cond ((atom tree1) (atom tree2))
       ((atom tree2) NIL)
       (t (and (isomorphic (car tree1)
                           (car tree2))
               (isomorphic (cdr tree1)
                           (cdr tree2))))))
```

Рисунок 1: Код функции ISOMORPHIC на языке Лисп

```
LSymbol ISOMORPHIC("ISOMORPHIC");
void LispInit_isomorphic() {
  static LSymbol TREE1("TREE1");
  static LSymbol TREE2("TREE2");
  (L|DEFUN, ISOMORPHIC, (L|TREE1, TREE2),
    (L|COND,
      (L|(L|ATOM, TREE1), (L|ATOM, TREE2)),
        (L|(L|ATOM, TREE2), NIL),
        (L|T, (L|AND,
```

```
    (L|ISOMORPHIC, (L|CAR, TREE1),
                   (L|CAR, TREE2)),
    (L|ISOMORPHIC, (L|CDR, TREE1),
                   (L|CDR, TREE2)))))).Evaluate();
}
```

Рисунок 2: Код функции ISOMORPHIC на языке Си++

Сказанное можно проиллюстрировать на примере функции, исходный код которой в синтаксисе языка Lisp приведен на рис.1.

Следует особо подчеркнуть, что код, приведенный на рис. 2, является корректным кодом на языке C++. Трансляция этого кода производится обычным компилятором C++; никакого предварительного преобразования кода не требуется.

Библиотека состоит из двух слоёв. На первом из них вводятся S-выражения как структуры данных, на втором на базе введенных S-выражений строятся вычислители, моделирующие семантику соответствующих языков программирования (в настоящее время - языков Lisp и Refal). Первый слой позволяет, помимо прочего, использовать концепцию S-выражений для построения универсальных гетерогенных контейнеров [3].

Ссылки

[1] T. A. Budd. *Multy-Paradigm Programming in LEDA*. Addison-Wesley, Reading, Massachusets, 1995.

[2] А. Столяров. Интеграция разнородных языковых механизмов в рамках одного языка программирования. *Диссертация на соискание степени канд. физ.-мат. наук*, Москва, 2002.

[3] Stolyarov, A. V. A framework of heterogenous dynamic data structures for object-oriented environment: the S-expression model In *Knowledge-Based Software Engineering. Proceedings of the 6th JCKBSE*, vol. 108 of *Frontiers in Artificial Intelligence and Applications*, pages 75–82, Protvino, Russia, August 2004. IOS Press.

[4] Официальный сайт проекта InteLib. http://www.intelib.org

15.35 – 16.00

Дмитрий Левин
Москва, ALT Linux

Проект: Sisyphus

Hasher: технология безопасного выполнения приложений

Аннотация

Рассматривается одно из относительно новых применений *hasher*'а как инструмента быстрого развёртывания среды для изолированного выполнения приложений. Приводятся примеры ситуаций, в которых

такое применение оказывается востребованным.

Изолированное выполнение приложений.
Разнообразные популярные средства виртуализации, в том числе виртуальные машины (такие как VMware и QEMU), модифицированные ядра Linux (такие как Linux Virtual Server и Virtuozzo) и User Mode Linux, можно рассматривать как различные формы изолированного выполнения приложений. Эти средства отличаются друг от друга предоставляемыми возможностями, потребляемыми ресурсами, сложностью реализации и удобством в использовании для решения тех или иных задач. Оказывается, что *hasher*, который разрабатывался первоначально для сборки пакетов, тоже можно рассматривать как средство виртуализации, удобное для решения некоторых классов задач, связанных с изолированным выполнением приложений.

Основы архитектуры *hasher*'а.

hasher базируется на принципе создания новой среды выполнения для каждой задачи.

В основе архитектуры *hasher*'а лежит трёхпользовательская модель: вызывающий непривилегированный пользователь (*C*) и два непривилегированных вспомогательных псевдопользователя; первый (*R*) играет роль root в порождаемой среде выполнения (далее псевдоroot), второй (*U*) - обычного пользователя, выполняющего программы (далее псевдосборщик).

Архитектура *hasher*'а призвана исключить атаки вида *UR*, *UC*, *RC*, а также все виды атак на root.

Переключение между вызывающим и вспомогательными пользователями осуществляется с помощью привилегированной программы, написанной специально для этих целей. Кроме того, с помощью этой программы удаляются процессы, запущенные вспомогательными псевдопользователями и не завершившиеся в срок, а также создаются устройства и монтируются каталоги. Наконец, эта программа предоставляет возможность контролировать ресурсы, выделяемые процессам вспомогательных пользователей, для защиты от DoS-атак.

Более подробное описание архитектуры *hasher*'а можно найти в [1].

Предоставляемые возможности

Будучи первоначально средством сборки пакетов, *hasher* вычисляет минимальное замкнутое подмножество пакетов репозитория, необходимое для работы указанного базисного набора пакетов, и порождает из этих пакетов среду выполнения. Механизм кэширования существенно ускоряет процесс формирования этой среды, и при неизменности множества пакетов, используемых для создания среды выполнения, доводит скорость формирования до скорости распаковки архива.

Наряду с созданием среды выполнения, *hasher* обеспечивает и удаление этой среды по окончании работы.

С помощью *hasher*'а можно в любой момент обеспечить завершение всех процессов, работающих в среде выполнения.

Процессу, запускаемому в среде выполнения, могут быть установлены ограничения на используемые ресурсы, поддерживаемые ядром Linux, а также ограничения на объём выходных данных, время простоя и общее время работы. В сочетании с дисковыми квотами и правилами iptables, которые можно установить для вспомогательных пользователей, эти ограничения позволяют полностью контролировать программы, запускаемые в среде выполнения.

Перед запуском приложения средствами *hasher*'а в среду выполнения могут быть смонтированы файловые системы, которые будут автоматически размонтированы по окончании работы этого приложения.

Средствами *hasher*'а для запускаемого процесса может быть специально создан и предоставлен управляющий терминал вне зависимости от того, смонтирован ли /dev/pts в среду выполнения или нет.

Кроме того, средствами *hasher*'а для запускаемого процесса может быть специально создан и предоставлен канал для X11 forwarding (как trusted, так и untrusted) вне зависимости от того, какие ограничения для вспомогательных пользователей установлены правилами iptables.

В каждый момент времени может быть создано и использоваться произвольное число сред выполнения, каждая со своей парой предварительно созданных вспомогательных пользователей.

Требуемые ресурсы
Для создания среды выполнения *hasher*'у требуется

репозиторий пакетов, совместимый с Sisyphus, на базе которого и будет порождаться среда выполнения.

За исключением времени, затрачиваемого на формирование среды выполнения, и пространства файловой системы, используемого для кэширования этой среды, у *hasher*'а практически нет накладных расходов на выполнение приложений.

Из других средств виртуализации такими показателями обладают только модифицированные ядра Linux.

Работоспособность *hasher*'а проверена на ядрах Linux 2.2.x, 2.4.x и 2.6.x. Для монтирования файловых систем средствами *hasher*'а требуется ядро Linux не старее 2.4.x.

Сложность реализации
hasher реализован с помощью shell-скриптов общим объёмом менее 75Кб, и вспомогательной программы, написанной на C, объёмом исходного кода менее 75Кб, из которых объём кода, выполняемого с правами root, составляет менее 50Кб.

Таким образом, *hasher* является самым легким (и, как следствие, потенциально самым безопасным) средством виртуализации с перечисленными выше возможностями.

Примеры использования
Вот наиболее очевидные примеры использования *hasher*'а в качестве средства изолированного выполнения приложений:

Тестирование программ в эталонной среде.
Свойство *hasher*'а создавать среду выполнения удобно использовать для того, чтобы тестировать программы в предсказуемом и легко воспроизводимом окружении.

Отладка программ.
Непривилегированные программы удобно отлаживать в изолированной среде, создаваемой *hasher*'ом, особенно когда отладка в хост-системе невозможна по тем или иным причинам.

Запуск приложений, требующих изолированной среды выполнения по соображениям безопасности.
Если политика безопасности не допускает установки программы (со всем, что ей необходимо для работы) в хост-систему, то запуск такой программы в среде, создаваемой *hasher*'ом, может быть приемлемым решением.

Обработка данных, требующая изолированной среды выполнения по соображениям безопасности.
В случае, когда источник данных, с которыми предстоит работать, не вызывает доверия, а на обрабатывающее эти данные ПО нельзя положиться в полной мере, обработка таких данных в среде, создаваемой *hasher*'ом, может быть приемлемым решением.

Ссылки

[1] Dmitry V. Levin, hasher: технология безопасной сборки дистрибутива, тезисы докладов 1-ой международной конференции

разработчиков свободных программ на Протве, 2004.

16.00 – 16.25

Алфекс Кейнокенн, Иван Тарасов
Franklin Electronic Publishers

Разработка opensource ПО для распознавания языков, частых ошибок на базе AI-like алгоритмов

- Алгоритмы определения принадлежности слов к языку без словаря В области встраиваемых устройств на базе opensource решений периодически возникает проблема создания эффективных безсловарных алгоритмов определения принадлежности набранных слов к тому или иному языку. Помимо этого к сожалению многие люди не обладают методом "слепой" печати, в связи с вышеописанными проблемами требуется эффективные решения их. Часть доклада будет посвящена теории создания подобных алгоритмов, которые не требуют наличие огромных баз слов и работают на основе прототипов языков.

- Сбор весовых характеристик к сочетаниям символов Для работы так называемых алгоритмов основанных на основе прототипов языков требуется эффективные и наиболее оптимальные алгоритмы для автоматизированного анализа языка и словарей для выведения наиболее точных весовых характеристик для сочетания символов того или иного языка.

- Прототипы натуральных языков и их анализ По мере накопления прототипов языков требуется их анализ по мере эксплуатации, здесь будут рассмотрены способы самоанализа и коррекции различными методами.

- Определение частых фонетических и прочих ошибок на основе прототипа языка Эта часть доклада посвящена расширению прототипов для определения не только принадлежности слов к языку а также к определению фонетических и прочих ошибок и анализу наиболее подходящих вариантов для замены.

- Совместная работа безсловарных алгоритмов с словарями Совмещение безсловарных алгоритмов и словарных связана с потребностью наиболее верной проверки орфографии языка и его семантики. Данная часть доклада посвящена рассмотрению наиболее оптимальных алгоритмов для совмещения с словарями.

- Типизация и группирование натуральных языков
Типизация и группирование языков это еще один аспект подобного анализа, посколько в дальнейшем будут проблемы связанные с схожими семантиками и прототипами языков, требуется более четкая и ясная обработка групп и типов языка.

16.25 – 16.50

Денис Овсиенко
Москва, ALTLinux

Конфигурация современных сетей в среде Linux

ALT Linux Sisyphus и современные сети

1 Факты

Начиная с ноября 2004 года проводится ряд модификаций репозитория пакетов ALTLinux Sisyphus. Этот процесс направлен на повышение эффективности использования Sisyphus в сети. Вносимые модификации можно разделить на две группы.

1.1 Новая роль: network-config-subsystem

До этого пакет net-scripts был единственным штатным средством конфигурирования сети в ALTLinux. Вместо него была введена абстракция network-config-subsystem, которая представляет собой нечто, обеспечивающее конфигурацию сетевых ресурсов Linux-хоста. Сейчас network-config-subsystem предоставлется двумя пакетами: net-scripts и etcnet. net-scripts -- одна из бывших составных частей пакета initscripts, разрабатываемого в RedHat и используемого во многих других дистрибутивах.

/etc/net --- система конфигурации сети, которая разрабатывается с 2004 года и позиционируется как замена для той части, initscripts, которая

обеспечивает конфигурацию сети. Эти два пакета взаимоисключают друг друга, но могут использоваться по очереди. В процессе внедрения / etc/net был адаптирован ряд пакетов, принимающих участие в процессе конфигурации, в результате чего система может работать корректно независимо от того, какая именно из двух реализаций установлена.

1.2 Постепенный отказ от net-tools

Пакет net-tools в числе прочих содержит программы

/sbin/ifconfig, /sbin/route и /sbin/arp. Эти программы когда-то были единственными утилитами для управления сетевыми интерфейсами, маршрутами и ARP-таблицей. С тех пор Linux три раза сменил версию стабильного ядра и два раза --- сетевой экран. Сейчас для адекватного конфигурирования сети в Linux используются пакеты iproute2 и iptables. Несмотря на это, до сих пор значительная часть документации ссылается на устаревшие команды.

Для улучшения ситуации было принято решение не устанавливать по умолчанию пакет net-tools. Так как некоторые другие пакеты его требуют, используя ifconfig или route в своих сценариях, их мантейнерами были внесены соответствующие правки для ликвидации этой зависимости.

2 Мотивы

2.1 Недостаточная гибкость

Проект /etc/net разрабатывался, исходя из опыта использования initscripts в различных дистрибутивах и сопровождения net-scripts в рамках ALTLinux.

Практика показала, что дизайн net-scripts не приспособлен для расширения его функций и поддержки новых типов интерфейсов.

В то же время современные сети медленно, но верно меняют набор используемых протоколов; Linux-системы поддерживают всё больше аппаратного обеспечения, в том числе с горячим подключением; растёт популярность нетривиальных конфигураций Linux-маршрутизаторов.

2.2 Громоздкая конфигурация

В системе net-scripts для каждого IP-адреса на интерфейсе необходимо создавать отдельный файл. В результате для одного логического интерфейса будет создано несколько файлов, отличающихся только двумя переменными.

2.3 Чрезмерное использование переменных

В net-scripts каждый параметр интерфейса должен быть определён в переменной. Во-первых, это приводит к большому количеству переменных, а во-вторых, ограничивает набор параметров, которые можно передавать программам конфигурации, набором доступных переменных.

Например, при определении статических маршрутов net-scripts полагается на единожды определённый формат файла, а /etc/net просто передаёт строки утилите ip.

2.4 Дублирование информации

Из-за того, что данные конфигурации, относящиеся к одному интерфейсу, в net-scripts находятся в разных местах, они связаны по имени интерфейса. При его изменении необходимо изменить это имя во всех файлах, в которых оно упоминается. Кроме того, имеет место и дублирование кода. Например, каждый из ifup-сценариев net-scripts кроме кода, относящегося к своему виду интерфейса, содержит вызов ifup-post, так как сценарий ifup не вызывает вспомогательные сценарии, а передаёт на них управление.

2.5 Решение

В этом контексте был проведён долговременный анализ роли подсистемы конфигурации сети в работе Linux-системы и выделен ряд требований, выполнить которые можно было только в рамках новой разработки:

- модульная поддержка протоколов
- модульная поддержка различных видов аппаратного обеспечения
- обработка зависимостей
- поддержка профилей конфигурации
- устойчивость к аппаратным переконфигурациям

3 Подробности

3.1 Модульность

Скелет системы образуют несколько сценариев, выполняющих самые основные функции в общем виде. В зависимости от контекста какие-то стадии конфигурации окажутся востребованы, а какие-то нет. Поддержка каждого протокола и вида интерфейса выделена в отдельные файлы. Это значительно уменьшает дублирование кода и облегчает его сопровождение. В настоящее время поддерживаются следующие протоколы: IPv4, IPv6, IPX.

3.2 Строгая типизация интерфейсов

Одно из требований к конфигурационной базе --- каждый обслуживаемый интерфейс должен относиться к какому-либо из 5 предусмотренных типов. Тип интерфейса играет роль в порядке запуска и останова интерфейсов. Каждый из обрабатываемых системой видов интерфейсов принадлежит к какому-либо типу. Распределение видов интерфейсов по типам следующее:

- локальная петля и dummy: виртуальные
- Ethernet, Pent@NET, USB, PLIP, WiFi: полноценные физические
- VLAN, мосты Ethernet, bonding, TEQL: производные физические
- туннели IPv4 и IPv6, туннели IPSec: независимые логические
- PPP: зависимые логические

3.3 Ориентировка на шаблоны

Основу конфигурационной базы составляет набор системных опций и интерфейс default. Другие интерфейсы наследуют его конфигурацию и дополняют её своими опциями. Таким образом, нет необходимости для каждого интерфейса определять весь набор поддерживаемых опций, появляется возможность изменять конфигурацию нескольких интерфейсов за раз. Наследование опций в /etc/net двухуровневое: сначала общие для всех интерфейсов опции шаблона default, затем опции, специфические для текущего типа интерфейса, затем опции конкретного взятого интерфейса.

3.4 Простота и масштабируемость

За счёт модульности и наследования простые конфигурации, востребованные на большинстве Linux-систем, реализуются несложно и быстро. В то же время при усложнении конфигурации используется тот же принцип формирования конфигурации. Конфигурационная база представляет собой иерархическую структуру в виде дерева каталогов, в котором каталог каждого интерфейса содержит все файлы и каталоги, к нему относящиеся. Это позволяет оперировать опциями сразу на уровне интерфейсов.

3.5 Наименование файлов

В системе tc net имя файла трактуется особым способом: в нем может присутствовать ссылка как на имя профиля так на имя хоста, к которому относится содержимое данного файла. Причем это соглашение справедливо для практически всех файлов, в том

числе и для настроек по умолчанию. В результате база конфигурации представляет собой не просто иерархическую структуру, а многомерное дерево. Практически это выражается в очень широком наборе доступных конфигураций, которые можно получить простыми приемами.

3.6 Встроенная поддержка QoS

/etc/net реализует встроенную поддержку QoS путём предоставления интерфейса к утилите tc. Интерфейс представляет собой дерево каталогов и файлов специального вида, располагающееся в конфигурационном каталоге интерфейса. Структура спланирована таким образом, чтобы максимально исключить дублирование параметров.

3.7 Дополнительные удобства:

- автодополнение переменных sysctl
- быстрая конфигурация VLAN через vlantab
- конфигурация нескольких систем из одной конфигурационной базы
- поддержка QoS
- поддержка сетевого экрана (в разработке)
- поддержка ifplugd
- автоматическая верификация конфигурационной базы

4 Планы

- завершение внедрения в ALT Linux 3.0
- внедрение в другие дистрибутивы
- поддержка сетевого экрана
- разработка собственного GUI-конфигуратора

16.50 – 17.10: Кофе

Вечернее заседание.

17.10 – 19.30

Круглый стол "Сообщество и корпорации". Ведущий Александр Боковой (IBM LTC, Samba project).

27 июля

Утреннее заседание (9.30 - 13.40)

9.30 – 9.55

Вадим Машков
EMT, Киев

Проект migration.osdn.org.ua

Рекомендации по миграции на свободное ПО

Вспомните те времена, когда DOS доминировала на большинстве персональных компьютерах. Затем появилась графическая надстройка – Windows 3.1, которая потребовала смены подходов к решению проблем офисной автоматизации.

Подобные эволюционные процессы являются естественными для любого типа программного обеспечения.

В настоящий момент при принятии решении о миграции на новую программную платформу у пользователей появился выбор. Одним из них является использование СПО.

Останавливаться подробно на вопросе “почему именно свободное ПО” не будем, об этом написано много материала, например www.emt.com.ua/boss.html.

Отметим только, что основными причинами перехода на свободное ПО являются: уменьшения стоимости владения ПО (ТСО), вопросы связанные с легализацией ПО, потребность в открытых стандартах.

Пример последнего - ситуация не такого далекого прошлого, когда к Вам, использующим Office 95, попал документ в формате Office 97. Сколько проблем возникло из-за несовместимости форматов. Это лишь один печальный пример, показывающий несостоятельность проприетарного ПО при смене форматов.

Первое, что необходимо сделать, это описать существующую систему. Необходимо провести инвентаризацию существующего аппаратного и программное обеспечение, а также описать модель действующей информационной системы.

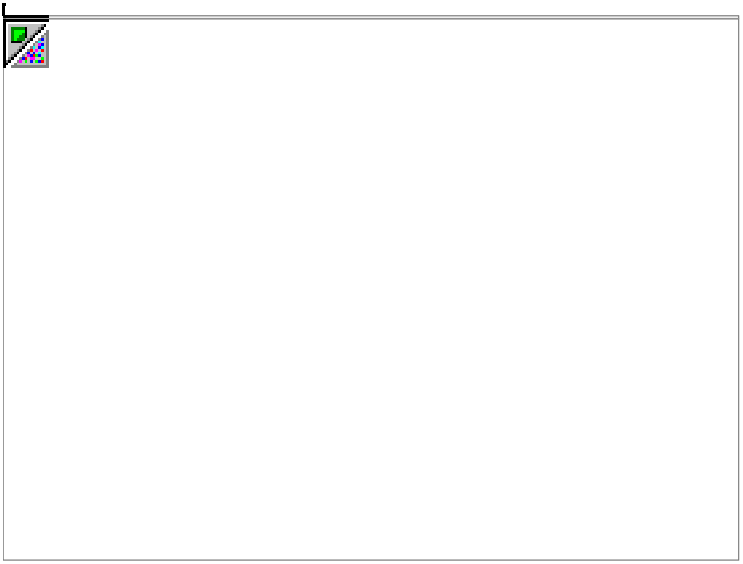
Следующим этапом является описание функциональных возможностей проектируемой системы, чтобы избежать переплаты за избыточную функциональность и повышенное требование к аппаратному обеспечению. На этом этапе нам пригодится построенная раннее модель действующей информационной системы, из которой мы можем перенести требования к её функциональным возможностям и учесть их при проектировании новой системы.

Зачастую на этом этапе производится расчет и сравнение совокупной стоимости владения как действующей так и проектируемой информационной системы.

Для примера возьмём типичное подразделение в организации или небольшую фирму.

Как правило это один выделенный сервер и несколько рабочих станций. Сервер обеспечивает коммуникационные сервисы, файловое хранилище, авторизацию, управление печатью.

Рабочие станции используются для выполнения стандартных офисных задач.



Исторически сложилось так, что в Украине зачастую используется проприетарное ПО. Причин множество, основная заключалась в отсутствии законодательной базы, регулирующей авторские права. Но так уж сложилось и нам с этим приходится жить.

Итак, возникла необходимость в смене используемых

программных решений. Выбор остановился на свободном ПО.

Для принятия решения о возможности перехода на использование свободного ПО, необходимо сформировать рабочую группу. В состав такой группы мы рекомендуем включить:

- Технических специалистов, которые решают технические вопросы;
- Руководителя для решения вопросов не-технического характера. Это избавит технических специалистов от решения вопросов не входящих в их компетенцию;
- Разумно включить в состав группы независимого Эксперта. Это специалист, который хорошо знаком с аспектами использования свободного ПО. Если такого специалиста нет в компании, найдите его среди компаний предоставляющие поддержку свободных решений.

(http://migration.osdn.org.ua/index.php?id=158)

Миграция состоит из нескольких логически

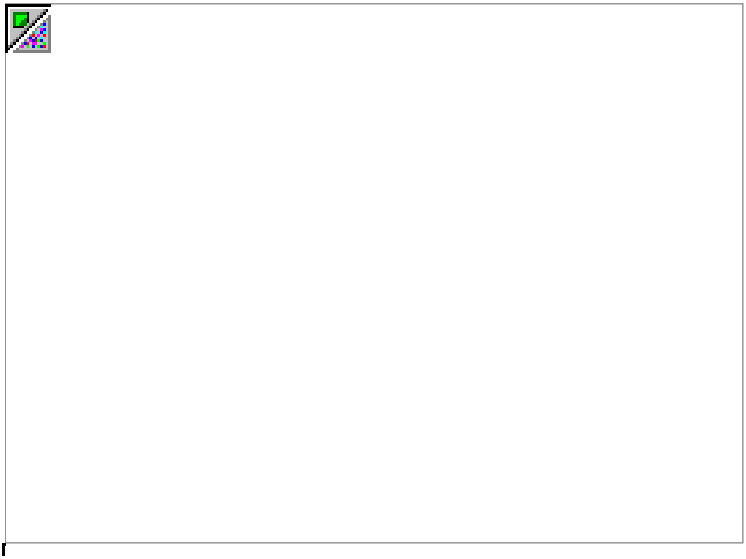
целостных этапов.

Все эти этапы необходимы. Некоторые из них требуют много ресурсов, иные достаточно просто реализовать. Последовательность не является догмой -- некоторые из этих пунктов могут выполняться параллельно.

Первое, что необходимо сделать, это описать существующую систему. Необходимо провести инвентаризацию существующего аппаратного и программное обеспечение, а также описать модель действующей информационной системы.

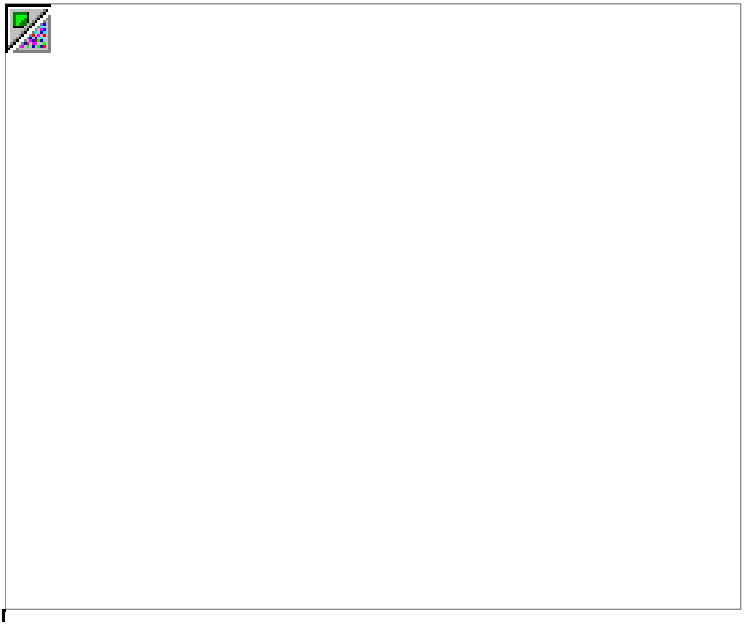
Следующим этапом является описание функциональных возможностей проектируемой системы, чтобы избежать переплаты за избыточную функциональность и повышенное требование к аппаратному обеспечению. На этом этапе нам пригодится построенная раннее модель действующей информационной системы, из которой мы можем перенести требования к её функциональным возможностям и учесть их при проектировании новой системы.

Зачастую на этом этапе производится расчет и сравнение совокупной стоимости владения как действующей так и проектируемой информационной системы.



В общем, можно отметить, что зачастую стоимость владения свободного ПО ниже проприетарного, за счёт отсутствия обязательных лицензионных платежей и низком риске вирусных атак. Основная составляющая стоимости в свободных решениях является поддержка.

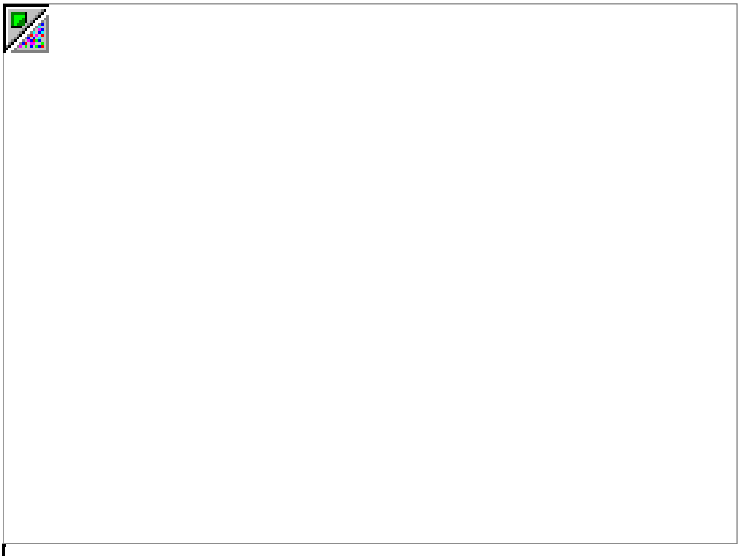
После того, как уточнены и описаны функциональные требования к проектируемой информационной системе, необходимо произвести поиск свободных программных решений реализующие их.



Вернёмся к модели информационной системы и рассмотрим как закрывается её функциональность при помощи свободных решений.

ПО на сервере заменяется свободными программными продуктами. При этом данный процесс протекает незаметно для пользователя.

Стоит отметить, что всё представленное ПО, входит в состав любого дистрибутива Linux.



В настоящее время большую часть функциональности клиентов закрывается свободными аналогами. Если нет свободного аналога, то одним из возможных решений решений может быть: использование эмуляторов окружения, терминальных решений либо оставить данный проприетарный продукт работать под управлением проприетарной ОС.

Для принятия решения о внедрении новой информационной системы на базе свободного ПО, мы рекомендуем провести Пилотный проект.

При проведении Пилотного проекта необходимо определить: объект испытаний, цель проведения работ, какие результаты должны получить как произвести их оценку.

При выборе объекта испытаний необходимо найти

золотою середину. Во-первых, важность выбранного объекта для возможности оценки использования свободного ПО. Во-вторых, проведение пилотного проекта не должно оказывать критическое влияние на ведение бизнеса.

В любом случае проведение пилотного проекта является выгодным шагом. Его результаты можно использовать и на других участках и при сравнительном анализе с пилотными проектами, выполненными на базе проприетарного ПО.

В качестве примера рассмотрим Пилотный проект, целью которого является использование свободного офисного пакета OpenOffice.org. Объектом выполняемых работ являются макросы MS Excel, которые используются для вычислений. В результате выполненных работ получаем макросы, работающие в OpenOffice.org и реализующие требуемую функциональность. В процессе замены офисного пакета, необходимо проверить правильность работы макросов и выяснить как пользователи оценивают работу офисных пакетов “до” и “после”.

Тщательное планирование – залог успеха миграции. При планировании учитывайте особые случаи. Это может быть: сложное-специализированное ПО; недостаточная подготовка персонала, что может потребовать его обучения; особые требования со стороны руководства; и т.п.

Составьте подробный план, учитывая необходимость создания резервных копий существующих данных, их переносу в новую систему; развёртывания системы; сроки обучения пользователей.

На этапе обучения пользователей уделите особое внимание тем, кто является приверженцем старой системы.

На основании дополнительных данных после проведения пилотного проекта уточните окончательную стоимость проекта миграции.

Определите ответственных и составьте подробный календарный план, установив в нём промежуточные контрольные точки.

Пусть ничто вас не останавливает!

Действуйте согласно ранее утверждённому плану!

Результатом миграции является информационная система, которая полностью удовлетворяет требованиям по функциональности и способная к масштабированию без особых финансовых и технических затрат. Возможно будет целесообразно совместное использование свободных и проприетарных продуктов. Кстати , часть коммерческих проприетарных продуктов можно заменить их бесплатными аналогами:

http://www.linuxrsp.ru/win-lin-soft/table-rus.html

http://tech.stolica.ru/article.php?id=2004101801

После перехода на использование решений на базе СПО мы:

- повышаем безопасность системы в целом;
- улучшаем управляемость рабочими станциями;
- снижаем зависимость от разработчика ПО;

а также снижаем стоимость внедрения и эксплуатации системы.

Данный ресурс предназначен, для обмена опытом внедрения свободного ПО. Это позволит снизить риск принятия ошибочных решений. Здесь вы сможете найти поддерживаемые свободные решения компаний СНГ.

9.55 – 10.20

Виталий Липатов
Etersoft

<Wine>

10.20 – 10.45

Петр Новодворский
Москва, IBM LTC

Компонентная сборочная система Samba 4.0

Аннотация
Доклад посвящен новой системе сборки, которая будет использоваться в новой версии Samba. Новая система сборки может быть адаптирована для любого сложного проекта, который состоит из значительного количества библиотек, модулей и программ и должен собираться на большом количестве платформ и операционных систем.

При проектировании Samba 4 были представлены следующие требования к новой версии:

- полнота реализации протокола;
- возможность автоматического тестирования;
- полностью асинхронная модель работы сервера;
- возможность как многонитевой работы серверов так и без использования нитей;

- реализация различных не POSIX "backend", т. е. возможность работы сервера на самых разных платформах.

В прошлых версиях Samba протокол CIFS/SMB был реализован не полностью, а возможности добавлялись по надобности. Новая версия Samba ориентируется на полную реализацию CIFS/SMB.

В связи с этим, сама Samba из набора серверов, обеспечивающих те или иные сервисы, превратилась в целую многоуровневую инфраструктуру. Каждый сервис вставляется в стек протоколов и реализует некоторый заранее объявлений интерфейс. Более того, есть модули, в зависимости от выбора которых, сервис будет работать или в многонитевом режиме так и без нитей. Кроме этого, новая Samba – это платформа для построения сервисов, а это значит, что надо отслеживать изменения в API каждой из библиотек, иметь возможность компилировать их статические и динамические версии.

В следствие всех этих изменений, потребовалась новая система сборки, которая поддерживала бы все эти функции.

Существующие системы сборки неполностью обеспечивают нужды проекта Samba. Новые системы не работают на требуемом количестве платформ, а существующая программа automake не обеспечивает возможности ввести требуемый уровень абстракции.

Samba4 состоит из большого количества компонентов: библиотек, модулей и исполняемых программ, более 50% кода в Samba генерируется автоматически. Разработчик, при добавлении новой

возможности в программный код должен внести соответствующие обновления и в файлы сборки, но из-за общей сложности и разветвленности системы, без нужного уровня абстракции это станет сложной задачей.

Система сборки Samba определяет 4 обычных компонента: MODULE, EXT_LIB, BINARY и LIBRARY, а так же специальные BUILDVAR и SUBSYSTEM. Весь исходный код Samba4 разбит на подсистемы, в каждом компоненте SUBSYSTEM указывается какие объектные файлы входят в каждую подсистему. После того, как файлы распределены по подсистемам, можно узнать, с помощью определения недостающих символов в объектных файлах, как подсистемы зависят друг от друга.

Далее описываются компоненты LIBRARY, BINARY и MODULE, которые содержат информацию о версии (в случае LIBRARY), а также о том, какие подсистемы входят в тот или иной обычный компонент. На этом этапе руководствуясь информацией о том, как зависят друг от друга подсистемы, можно сказать, как зависят друг от друга и эти компоненты и выявить циклические зависимости. Так же описываются требуемые внешние библиотеки, их описание генерируется автоматически с помощью сценариев autoconf.

Самым высоким уровнем среди компонентов является BUILDVAR. В этих компонентах описывается то, каким образом будет собран тот или иной компонент: модуль, программа или библиотека. Так же, возможно указать, как он будет собираться относительно других компонентов. Таким образом, у разработчика есть возможность указать отношение

между сборочными компонентами, а не вид сборки того или иного компонента.

Процесс сборки с помощью сборочной системы Samba4 разделен на три этапа. На первом этапе используется система configure для выяснения параметров системы, на которой будет осуществляться сборка. Далее в работу вступают собственные сценарии, написанные на perl, которые из файлов описания компонент создают обычные файлы сборки на языке GNU Make. После этого GNU Make обрабатывает эти файлы и компилирует программу. Таким образом, в сборочной системе задействовано три средства: GNU Make, Autoconf и Perl, а они присутствуют почти на всех операционных системах и платформах существующих сегодня.

Итак, новая система сборки, является гибкой заменой automake, с высоким уровнем абстракции, позволяющая разработчику отслеживать изменения компонентов, связи между ними и быстро вносить дополнения к общей инфраструктуре программного кода. Система сборки позволяет разработчику пакетов быстро добиться того варианта сборки, который ему требуется. Система работает на может работать на большинстве платформ, которые существуют сегодня и может быть адаптирована почти к любому сложному проекту.

10.45 – 11.10

Георгий Курячий
МГУ ВМК, ALT Linux

Компьютерная грамотность в школе: научение или дрессура?

Преданья старины

Преподавание компьютерной грамотности в нашей стране имеет чуть ли не самую долгую, и уж точно - самую драматическую историю. Когда-то в области ЭВМ Советский Союз был "впереди планеты всей", и в немалой степени - именно за счёт привлечения к этому делу молодёжи.

Кудесники в белых халатах

программист - служитель культа

"Классический" подход к использованию ЭВМ, в том числе и в школах, основывался на том, что между собственно ЭВМ и пользовательской задачей стоит ещё один человек - программист, обученный командовать машиной и решающий составляющий решение задачи (программу). Обучение было нацелено на воспитание именно программистов, делалось это - не без блеска - выборочно и в редких школах. Программы основывались на трёх слонах: теория алгоритмов, элементная база и дискретная математика вместе с алгеброй логики. Естественно, освоение разумного объёма этих наук требовало от школьника незаурядных математических и логических способностей. Стоит вспомнить, что ЭВМ тогда были только в вычислительных центрах и

программисты-профессионалы исчислялись десятками, их было даже меньше, чем физиков-ядерщиков.

Не отказались от схемы "теория алгоритмов - элементная база - дискретная математика" и при создании первых общешкольных программ по информатике. Объём знаний в них был ещё больше урезан (что само по себе приводило к пустому перечислению терминов), и к тому же в дисциплину "Информатика" планировалось внедрить основы кибернетики и собственно информатики (по иронии судьбы в именно это и не было сделано). Программирование в отсутствие самих ЭВМ, а впоследствии - в отсутствие единой методической и инструментальной базы (никакого стандарта на школьные классы, насколько нам известно, нет до сих пор) породило устойчивое мнение, что школьная программа по информатике из рук вон плоха.

Алгол я бы выучил только за то, что им разговаривал Дейкстра

По сути дела, главным достоинством такой программы было то, что она всё-таки помогала научиться *программировать*, хотя эта способность, как показало время, и не востребована массово. В специализированных школах и классах разрабатывались удобные для преподавания программирования языки (иные имели романтические названия "Шпага-2", "Рапира" и т. п.), проводились занятия по методам программирования и т. п. Ученики приобретали три главных компоненты профессии: понимание, знания и навыки. Если бы не элитарность этого направления, лучшего и желать нельзя.

В то же время "обычные" школы перебивались чем Билл послал: долгое время основным языком преподавания (заметим, всё ещё программирования) был, уж извините, Бейсик. Проблема с языками, ориентированными на преподавание, очень проста: официально всё ПО относилось к ПО ЗК, и не могло быть куплено государством, а стало быть, введено в программу.

Новь

Таким образом школа оказалась совершенно неготова к резко изменившейся ситуации, которую в 80-е годы именовали "компьютер входит в каждый дом". Выяснилось, что область применения ЭВМ растёт столь стремительно, что старый, "доскональный" подход к компьютерной грамотности требует немыслимого времени на обучение, и даёт в результате отличного специалиста узкого профиля.

Кофеварка-пылесос

Компьютер - это многофункциональный бытовой прибор. И ну перечислять.

В то время как становилось всё более очевидно, что (персональный) компьютер - это такой же бытовой прибор, как телевизор или холодильник, только значительно более многофункциональный. В школьных программах (не тех, что одобрены министерством, а тех, что _действительно_ читались в школах) возобладал "перечислительный" подход к компьютеру: подобно тому, как учат забивать гвозди, стали учить набору текста в текстовом редакторе, рисованию картинок в графическом и т. п.

windows и без того бестолковое, поможет ли оно преодолеть компьютеробоязнь?

Положение осложнялось двумя нюансами. Во-

первых, каждая школа использовала такое ПО, какое удавалось "достать", так что говорить о единой - пускай даже неофициальной - программе говорить не приходилось. Во-вторых, таким ПО оказывалось как правило ПО ЗК, ориентированное на заучивание, а не понимание, в котором всё обучение сводится к тому, что бы помнить, какая кнопочка какую функцию выполняет.

Хуже того: сами производители этого ПО ЗК активно поддерживают такое понимание "обучения", предлагают курсы такого же содержания и даже называют это "наукой"! Вдобавок они обычно смотрят сквозь пальцы на бесплатную передачу и копирование этого ПО, которые при других обстоятельствах они без стеснения называют "кражей". Выходит, что взрослым красть нельзя, а детям и учителям - можно? Но всё это заслонил лозунг "первая доза - бесплатно!".

Занимаюсь на труде синхрофазотроном

Это для уроков труда. И сначала пускай войдёт в каждый дом.

На самом деле такой - инструментальный - подход к компьютерной грамотности не лишён в основе своей смысла: есть некоторый набор задач, которые принято решать с помощью компьютера и некоторый набор инструментов в этом компьютере помогающий в решении. Естественно понимать под "грамотностью" способность этим инструментом воспользоваться. Главное здесь вот что: *цель* обучения не сам компьютер, а круг задач, с помощью него решаемый. Можно сказать, что отсылка электронных писем или набор и распечатка текста - занятие для урока *труда*, ибо преследует те же цели: инициировать бытовые и начальные

профессиональные навыки.

Тысячи функций нельзя запомнить. Даже если имитировать пылесос.

Однако при лобовом подходе, отягощённом непригодным к изучению ПО, напрочь отметаются вне других составляющих: знания и понимание. Про понимание было сказано выше: если вместо изучения задачи и инструмента нам предлагают заучить названия клавиш, вряд ли мы продвинемся в понимании самой задачи и принципов её решения. Что же касается знаний, то часть преподавательского сообщества вообще считает, что автоматизация убивает знания: можно ли школьнику пользоваться калькулятором, если он результате не будет знать таблицы умножения? можно ли пользоваться проверкой синтаксиса, если школьник пишет "карова пашла даитца", едва отключив проверку? Наконец, непонимание задачи и неумение самостоятельно построить её решение приводит к безынициативности и созданию ложных логических связок. Так, люди начинают пугаться при смене внешнего вида кнопочек, теряться при перестановке местами пунктов меню и предпочитают вообще лишний раз не наживать на кнопку, а дождаться подсказки со стороны компьютера.

Поиск Пути

15 лет в ВМШ

Автор этих строк пятнадцать (с первого курса университета) лет проработал преподавателем информатики в Вечерней Математической Школе при факультете Вычислительной Математики и Кибернетики Московского Государственного Университета им. М. В. Ломоносова (далее, для краткости, ВМШ. ВМиК и МГУ соответственно).

Ему не приходилось, как то бывает в обыкновенных школах, иметь дело с детьми, которым _неинтересно_ учиться. Уровень учеников ВМШ был самый разный, так как на информатику часто ходили по причине того,, что в школе её не было. Из пятнадцати последние десять лет преподавание велось на базе UNIX-подобных ОС (FreeBSD, Linux) с использованием различных языков программирования (в основном, Си, но были также и Паскаль, и Питон, и даже AWK, вызвавший, между прочим, наибольший интерес).

За годы преподавания автор укрепился во мнении, что *методически* "старый" подход к обучению компьютерной грамотности верен (всё-таки школьные программы составляют профессионалы), только материал отчасти негоден (почему-то эти профессионалы не имеют представления о ПО ОК и о том, что его свободно можно рекомендовать во *все* школы), а отчасти нацелен на воспитание узкого профессионала.

Машина мыслить не может, только ездить

Только исполнитель команд

Всё должно вернуться на круги своя. Новое время выдало правильную формулировку: ЭВМ - это всего лишь прибор, наподобие пылесоса. Он не обладает интеллектом, душой, волей, он - даже не покорный исполнитель команд пользователя, а сами эти команды, машинка для их материализации. Орудие в духе марксовой лопаты как усилителя ладони. Компьютер - усилитель мозга.

Алгоритм, без него никак

Для этого стоит чётко показать, что именно делает компьютер: выполняет команды. Для того, чтобы

задачу решить, человек слагает команды в *алгоритм*. Вот и первый слон из трёх вернулся на своё место. Стало быть, как ни странно это нынче звучит, умение программировать (пускай на самых простых задачах) - это часть современной компьютерной грамотности.

Как самому собрать аппарат?

понимание, почему так

Далее. Сама способность ЭВМ выполнять команды - не есть что-то мифическое. Аналогии с привычным телевизором недостаточно, так как его функциональность - фиксирована, а функциональность компьютера - динамическая (например, поведение персонажей компьютерных игр). Следует раскрыть уровень, на котором фиксируется функциональность ЭВМ, объяснить как работает логика простых ключей и как из неё собирается "анализатор и думатель". Вот и два других слона возвращаются из небытия на полагающиеся им места.

представление об архитектуре и принципах работы

Конечно, теперь этого мало. Необходимо показать, что вся динамика работы компьютера - это такие же точно программы, как и те, что может написать школьник, только посложнее и поспецифичнее. Возникает понятие операционной среды, которая предназначена для удобного пользование компьютером.

Машина при человеке, а не человек при машине

К пульту сразу, память пальцев

Наконец, нужно постоянно создавать ситуации, провоцирующие активное поведение школьника за компьютером. Человек должен командовать машиной, а не наоборот! Здесь особенно хороши

UNIX-подобные системы с интерфейсом *командной* строки. В первый же день занятий, ещё не умея программировать, школьники узнают, как зарегистрироваться в системе и выполнить самые простейшие команды, и в дальнейшем компьютер используется не только как инструмент программирования, но и как пособие по теоретическим курсам (этому изрядно помогает документированность и открытость операционных систем ПО ОК). Тем самым нарабатываются навыки именно взаимодействия с компьютером, а не только программирования.

Наехать на скрепку

Очень важно при этом научить самостоятельно ставить себе задачу, искать необходимую для решения информацию и доводить решение до конца. Это умение, опять-таки, относится не только и не столько к программированию, сколько к пользованию компьютером вообще. Именно отсутствие этого умения у большинства пользователей одного известного программного продукта и породило "венец творения" - скрепку, то и дело долбящую своей верхней частью стекло монитора.

Альтернативы, различные варианты решения одних и тех же задач

Принципиально важно воспитать у школьника подход к решению задачи не от инструмента (как бы этим микроскопом гвоздь забить), а от задачи (чем бы забить гвоздь), дать понять что у одной и той же задачи может быть множество непохожих решений, достигнутых с помощью самых разных инструментов. В этом тоже весьма помогает Linux, где всевозможных инструментов наблюдается полное

разнообразие.

Это - та самая черепаха, на которой и стоят три слона. Часто её называют "Преодоление Компьютеробоязни", но истинное её имя - Орудие Труда.

Приложение 1. Путь Гармонии

тем, для кого компьютер - один из многих инструментов

Цель этой программы - "озвучить" три упомянутые выше требования: построить изложение от алгоритма, дать представление о внутреннем устройства и способе функционирования ЭВМ и прибить в зародыше компьютеробоязнь вместе с "пассивностью за пультом". Программа рассчитана на любой уровень начальной подготовки и несколько раз (с использованием в качестве базового ЯП также Паскаля и Си) читалась для слушателей ВМШ при ВМиК МГУ.

Архитектура и программное обеспечение персонального компьютера (2 семестра)

- ЦПУ - ОЗУ - шина - устройства в/в
- Загрузка (ПЗУ) - диск - файл - файловая система - операционная система
- Файлы - текстовый редактор - операции с файлами, файловые менеджеры, архивы, типы файлов
- Внешние устройства и понятие "драйвер" - лазерный диск и Fliash, работа с ними - видеоподсистема и звук - работа с графикой и музыкой
- Передача данных - сеть - Интернет - понятие протокола - WWW, почта и др.
- "Офис"

- Прочее

Программирование на языке Python (2 семестра)

- Машинные коды - язык программирования - компилятор - интерпретатор
- Командная строка Python - арифметика - строки - списки
- Переменные и присваивание - функции - условия и циклы
- Оформление программы - докумантационная строка - комментарии
- Модули и методы - математика - графика - прочие - помощь по Python
- В/В - файлы - операция "%"
- Структуры данных (более полное описание) - сегменты - ассоциативные массивы

Приложение 2. Путь Мудрости

собственно ВМШ, для имено программистов (на самом деле - дисциплина мышления, а не инструмент)

Нижеследующая программа составлена из частей программы "Информатика" ВМШ при ВМиК МГУ разных лет, все курсы читались школьникам в том или ином виде (однако как единое целое - ещё ни разу). Как можно видеть, она ориентирована на будущего программиста, однако не привязана ни какому конкретно инструменту разработки и предназначена, в первую очередь, для воспитания *дисциплины мышления*. Мы прекрасно осознаём, что программа такого наполнения и глубины требует специального контингента школьников: тех кому это интересно, и кто имеет соответствующие способности. Стоит заметить при этом, что в ВМШ идут также и из школ, где информатика в загоне или её нет вообще; этот курс (с пропуском наиболее

сложных мест) оказался полезен и таким детям, причём отдача иногда была даже сильнее, и были случаи, когда школьник вообще не ходил на свою школьную информатику, а получал справку о прослушивании её в ВМШ.

Модельная Машина (проект ММ2) - это простейшая двухадресная ЭВМ классической архитектуры с крайне простой системой команд (их у неё 16), процедурами и константами в ПЗУ и т. п. На сегодняшний день существует эмулятор ММ2, пакет разработки программ (компилятор и компоновщик) и декомпилятор. Архитектура ММ2 настолько проста, что позволяет поначалу писать программы в машинных кодах, а затем уже переходить к языку ассемблера, попутно изучая процессы компиляции и компоновки.

За неимением места темы курсов представлены "крупноблочно", в основном для ого, чтобы показать *направление* преподавания. В действительности многие из тем должны даваться вперемежку, дабы свести к минимуму эффект "волшебных слов" (делайте это, а что это и зачем, мы расскажем потом). Кроме того, последний, четвёртый, семестр сделан факультативным, так как он обычно приходится на весну 11-го класса, когда у детей и родителей есть заботы поважнее информатики, которую на вступительных экзаменах не сдают.

Первый семестр

- Архитектура и программное обеспечение персонального компьютера (1-я часть)
- Математические основы ЭВМ
- Системы счисления и побитовые операции
- Классическая архитектура фон Неймана, одно-

двух- и трёхадресные реализации
- Элементы алгебры логики, моделирование арифметики
- Аппаратная реализация АЛ (различные варианты)
- Модельная Машина
- Адресация и порядок выполнения, архитектура и карта памяти
- Переход и условный переход
- Каноническая схема цикла (восстановление - условие - тело - изменение)
- Самомодификация, вызов подпрограмм и функций в/в
- Моделирование типов данных

Второй семестр
- Архитектура и программное обеспечение персонального компьютера (2-я часть)
- История вычислительной техники
- Программирование на языке Python (1-я часть)
- Машинные коды - язык программирования - компилятор - интерпретатор
- Командная строка Python - арифметика - строки - списки
- Переменные и присваивание - функции - условия и циклы
- Оформление программы - докумантационная строка - комментарии
- Модули и методы - математика - графика - прочие - помощь по Python

Третий семестр
- Алгоритмы и структуры данных
- Понятие сложности алгоритма
- Алгоритмы сортировки и поиска
- Пересборная схема, метод волны, метод Уоршалла и т. п.

- Программирование на языке Python (2-я часть)
- Структуры данных (более полное описание) - сегменты - ассоциативные массивы
- Рекурсия и элементы функциональное программирования
- Объекты как контейнеры методов
- Исключения, генераторы и пр.
- Полезные модули
- Основы Linux (1-я часть)
- Терминал и командная строка - файловая система - основные команды
- Права доступа и процессы
- Установка и настройка
- Пакеты

Четвёртый семестр
- Основы Linux (2-я часть)
- Обработка текстов, регулярные выражения
- Настройка окружения
- Понятие о TCP/IP и сокетах
- X11
- Программирование на языке Python (3-я часть)
- Использование регулярных выражений
- Элементы объектного моделирования - классы - наследование - перегрузка методов
- Программирование сетевых взаимодействий

11.10 – 11.35

Михаил Якшин
Москва, ALT Linux

Система автоматизированного тестирования и контроля качества оборудования "Inquisitor"

Система Inquisitor разрабатывалась по заказу MaxSelect как открытая система автоматизированного полного тестирования компьютеров в промышленных масштабах. Основная цель системы – тестирование надежности работы как отдельных комплектующих машины, так и всех их вместе как единого комплекса.

С помощью системы решаются следующие задачи:

1. Разнообразное и полное тестирование отдельно взятой машины;
2. Прошивка серийных номеров и выполнение иных технологических операций;
3. Обновление BIOSов;
4. Ведение базы данных по протестированным компьютерам, сбор и хранение детализированной информации об их компонентах;
5. Наблюдения состояния тестового стенда в реальном времени, учет работы операторов тестирования;
6. Установка программного обеспечения на протестированные машины (OEM-версия ALT Linux Compact);

Технологически система Inquisitor работает по

принципу, по которому работает большинство LiveCD/DVD систем: некий загрузчик загружает ядро и минимальный initrd, который подгружает необходимые модули и подключает уже полноценный образ Live-системы либо по полноценным сетевым протоколам (NFS), либо с диска. Затем система производит детект оборудования (hotplug/libhw) и переходит к выполнению основной программы тестирования.

Изначально система разрабатывалась как самодостаточная единичная система, состоящая из сервера, предоставляющего сервисы загрузки по PXE (серверы dhcp/tftp) и файловую систему Live-образа через NFS, но применение системы показало ее перспективность и были созданы несколько вариантов сборок:

1. Inquisitor Enterprise Notebook / Desktop- полная система, тестирующая ноутбуки / desktop'ы;

2. Inquisitor Pro - система для дилеров и продавцов, не включая в себя всех элементов производственного процесса (в частности, прошивки серийных номеров и полноценного доступа к центральной информационной базе);

3. Inquisitor Service - система, обладающая максимальными возможностями по управлению ее поведением (для сервис-центров);

4. Inquisitor LiveCD/DVD - версия, загружающаяся с CD/DVD, позволяющая провести автономное тестирование и выгрузить отчет о нем на внешний носитель (дискету, USB-накопитель и т.п.)

Все существующие Inquisitor'ы, тестирующие оборудование MaxSelect, объединяются в одну сеть с помощью информационной базы. Информационная база - центральное место пересечения всех информационных потоков подобного компьютерного производства и сопровождения: туда стекается информация о производстве (Inquisitor), торговая информации (из торговых баз), информация о поддержке (системы сервиса и службы поддержки), информация от конечных пользователей (с сайта www.maxselect.ru) и т.п. Отчеты, выгруженные Inquisitor LiveCD/DVD на внешний носитель, также могут быть загружены в информационную базу.

С технической точки зрения, Inquisitor представляет собой некий вариант инсталляции (дистрибутив) ALT Linux, использующий репозитарий Sisyphus в качестве базы, собирающийся с помощью separator как загрузчик (нулевая стадия), некий загрузочный образ (первая стадия) и Live-образ (вторая стадия). В зависимости от варианта сборки, может быть собран как NFS-образ, так и ISO-образ CD/DVD.

В Inquisitor входят следующие тесты, большинство из которых основаны на свободном программном обеспечении с некоторыми доработками:

- CPU: проверка процессора под нагрузкой (cpuburn, mencoder) и
- скейлинга (libhw/cpufreq);
- Память и видеопамять: mx/memtest - userspace тест памяти;
- HDD: SMART-проверка - smartctl;
- Нагрузочное тестирование: сборка пакетов в rpm/hasher;
- Оптический привод: запись и чтение диска

cdrecord/readcd;
- Видеокарта: демо-версии коммерческих игр (UT2004, Doom 3), а также свободные игры и 3D engines (Tenebrae, Vegastrike, Crystal Space);
- Матрица, батарея: собственные разработки.

Так как в процессе тестирования компьютер может зависнуть или перестать отвечать, на тестируемой машине загружается watchdog и сервер постоянно отслеживает его жизнеспособность: если watchdog долго не отвечает, то компьютер считается зависшим (например, от перегрева) и об этом будет подан сигнал оператору тестирования.

По мере развития стало ясно, что система позволяет не только проверять стабильность оборудования, но и проводить другие тесты. Например, данные системы могут быть использованы в качестве benchmark: FPS в 3D-тестах, производительность в тестах процессора и памяти, время жизни от батареи и т.п.

С помощью системы может быть реализована программа сертификационных испытаний конфигураций оборудования: устанавливаются дополнительные мониторинги (температуры, напряжения и т.п.) - если некая конфигурация проходит жесткие тесты в этих условиях, не выходит за установленные рамки показателей мониторинга и обеспечивает адекватное быстродействие по benchmark'овым показателям - она сертифицируется.

Еще одно перспективное применение системы - определение совместимости оборудования с системами на базе ALT Linux, составление с ее помощью hardware compatibility lists.

11.35 – 12.00: Кофе

12.00 – 12.25

Дмитрий Белявский, Виктор Вагнер,
Артем Чуприна
ООО "Криптоком"

Криптографическая русификация OpenSSL: решения, проблемы, перспективы

OpenSSL - развитый OpenSource инструмент, реализующий криптографические стандарты. OpenSSL реализует как протоколы SSL и TLS, так и работу с ASN1-структурами (X509, S/MIME, PKCS8, PKCS12).

Текущая версия 0.9.8 содержит в той или иной степени поддержку большинства имеющих практическое значение RFC, определяющих использование криптографии в информационном обороте.

Исторически в OpenSSL поддерживались, похоже, только два асимметричных алгоритма (RSA и DSA), затем был добавлен еще один (EC DSA), и авторам OpenSSL оказалось проще не создавать инфраструктуру работы с несколькими алгоритмами, а рассеять обработку конкретных алгоритмов по всему коду. Количество же алгоритмов хэширования и шифрования заставило их к современной версии

создать инфраструктуру, позволяющую добавлять достаточно произвольные новые алгоритмы. Для асимметричных алгоритмов такую инфраструктуру пришлось создать нам, когда мы попытались добавить два российских алгоритма подписи.

Фирма "Криптоком", изучив специфику предполагаемых национальных стандартов, разработала инфраструктуру для добавления асимметричной криптографии. Разработанная инфраструктура позволяет легко добавлять любой алгоритм, похожий в достаточно широком смысле на алгоритм Эль-Гамаля.

Разработанная инфраструктура предусматривает реализацию добавляемых алгоритмов в виде отдельных модулей. Это решение обладает рядом преимуществ перед непосредственным добавлением алгоритмов в ядро OpenSSL:

- Архитектура OpenSSL становится более внятной;

- Инфраструктура позволяет сравнительно легко добавлять поддержку

национальной криптографии для тех стран, которые не используют

стандартные алгоритмы (Россия, Япония, Китай).

Эта инфраструктура позволила нам добавить алгоритмы российской криптографии как с использованием проприетарной сертифицированной реализации длинной арифметики и аппаратных средств, так и с использованием реализации

OpenSSL.

Реализованы криптографические алгоритмы:

- Подпись по ГОСТ Р 34.10-94;
- Подпись по ГОСТ Р 34.10-2001;
- Дайджест по ГОСТ Р 34.11-94 года;
- Симметричное шифрование по ГОСТ 28147-89.
- Улучшена поддержка S/MIME:
- Добавлена возможность добавления множественных подписей;
- Исправлена работа с большими файлами.

Среди прочих результатов надо отметить сокращение количества жестко закодированных привязок к конкретным стандартам, которые теперь стали свойствами отдельных алгоритмов. Кроме того, разработана и передана авторам OpenSSL дополнительная система тестов, которая позволила обнаружить ряд ошибок в оригинальном коде OpenSSL в процессе подготовки версии 0.9.8 к релизу.

На данный момент реализация поддерживает в полном объеме работу с ASN1-структурами. Ведется работа по добавлению российской криптографии в SSL/TLS.

На основании достигнутых результатов можно

выделить несколько перспективных направлений развития.

1. Реализация TLS в соответствии с национальными стандартами.

2. Генерализация работы с асимметричной криптографией в библиотеке XMLSec, являющейся основой криптографии в OpenOffice 2.0.

3. Обобщение сделанного решения на более широкий класс алгоритмов.

Кроме того, существует ряд идей, направленных на улучшение собственно архитектуры OpenSSL, направленных на более четкое разделение на независимые модули.

Мы считаем, что включение нашего патча в сборки OpenSSL в российских дистрибутивах операционных систем с открытым кодом было бы полезным с точки зрения распространения свободного софта в сферы, где требуется применение российских криптографических алгоритмов.

Кроме того, сейчас - оптимальный момент для такого включения: версия 0.9.8 не входит в состав ни одного из современных дистрибутивов, и переход на новую версию с одновременным приложением нашего патча позволит уменьшить трудозатраты.

С другой стороны, можно будет получить сертифицированное решение на базе open source системы, если совместить open source дистрибутив, содержащий OpenSSL с этим патчем, с сертифицированной реализацией криптографических

алгоритмов. Тем самым для проприетарного программного обеспечения будет оставлена ограниченная область с четко определенной ответственностью за ее работоспособность.

12.25 – 12.50

Игорь Головин, Андрей Столяров

МГУ, ВМК

Мультипарадигмальный подход к преподаванию программирования и роль свободного ПО

Аннотация

Рассматриваемый подход заключается в одновременном преподавании различных парадигм программирования (логического, функционального программирования и т.п.) на возможно более ранних этапах обучения. Такой подход более эффективно и гибко развивает у обучаемых алгоритмическое мышление по сравнению с традиционным подходом, основанным исключительно на императивном программировании.

Преподавание альтернативных парадигм программирования выдвигает ряд требований к используемому в процессе обучения программному обеспечению, прежде всего – к системам программирования и операционным системам.

Утверждается, что этим требованиям наиболее полно отвечают свободно

распространяемые системы программирования и операционные системы семейства Unix, среди которых также можно выбрать свободно распространяемые ОС.

Традиционная схема обучения основам программирования выглядит следующим образом:

- понятие алгоритма с использованием алгоритмических нотаций, таких как машина Тьюринга, нормальные алгоритмы Маркова и др.
- основы базового императивного языка программирования, в качестве которого, как правило, выступают Pascal, Basic, Fortran, C.
- базовые навыки составления алгоритмов
- введение в низкоуровневое программирование на основе избранной архитектуры (чаще всего – 8086), сопровождаемое изучением соответствующего языка ассемблера
- возможно, введение в императивно-объектное программирование на основе C++, Java или Delphi

Изучение альтернативных языков программирования, таких как Lisp, Prolog и т.п., выносится на поздние стадии обучения, если вообще включается в программу.

Такой подход мы называем *монопарадигмальным*.

К достоинствам такого подхода можно отнести то, что студент получает навыки программирования на одном или нескольких популярных языках программирования. Также следует отметить, что соответствующие среды программирования широко доступны, равно как и учебные пособия.

В то же время такая схема обладает рядом

существенных недостатков. Главным из них является ограниченность чисто императивного подхода, включающая два аспекта. Во-первых, за рамками поля зрения студента остается ряд полезных методов программирования; так, многие студенты, прошедшие подготовку по традиционной схеме, не умеют пользоваться рекурсией, не умеют достаточно гибко использовать динамические структуры данных и т.п.

Во-вторых, не для всех студентов обучение на основе императивного программирования является наиболее подходящим. Ряд студентов испытывают трудности при освоении чисто императивного подхода. В рамках монопарадигмального обучения эти трудности распространяются на весь процесс программирования в целом, лишая студента перспектив освоения программирования на профессиональном уровне.

Существует альтернативная точка зрения, состоящая в том, что процесс начального обучения следует построить с одновременным использованием разнородных парадигм программирования (включая, кроме традиционной императивной, как минимум функциональную и логическую) на возможно более ранних стадиях. В этом случае обучение может включать одновременное изучение нескольких (разнородных) языков программирования. Назовем такой подход *мультипарадигмальным*.

При таком подходе подача материала строится на основе рассмотрения разнообразных приемов решения одной и той же задачи, в то время как при монопарадигмальном подходе решение (и, как следствие, мышление студента) ограничивается возможностями выбранного языка программирования.

Императивную парадигму, равно как и любую другую, нельзя считать универсальной, т.е. одинаково подходящей для любых возникающих задач. При мультипарадигмальном подходе студенты учатся оценивать достоинства и недостатки разнородных подходов, приемов и языков программирования.

Встречаются студенты, индивидуальные особенности которых располагают к изучению альтернативных парадигм программирования; в частности, некоторым студентам функциональное или логическое программирование дается легче, чем традиционное императивное. Нельзя не отметить, что переход от функционально-декларативных методов программирования к императивным существенно проще, чем в обратную сторону; студентов, ориентированных на императивное программирование, зачастую вообще невозможно переучить.

Одним из главных препятствий к внедрению мультипарадигмального подхода является отсутствие массового и общепринятого языка, сочетающего разнородные парадигмы. В частности, язык C++, часто называемый мультипарадигмальным, основан, тем не менее, на сочетании процедурного и объектно-ориентированного программирования и поддерживает лишь смежные с ними парадигмы (например, обобщенное программирование), не стимулируя при этом мышление, например, в терминах функционального программирования. Попытки создания такого языка неоднократно предпринимались, однако разработанные в результате языки широкого распространения не получили. Поэтому в процессе применения мультипарадигмального подхода приходится использовать несколько языков программирования

(например, Pascal, Lisp, Prolog и Refal).

В результате особое значение приобретает выбор соответствующих систем программирования и, разумеется, операционной системы.

Многие из рассматриваемых языков программирования выглядят неестественно в окружении, основанном на графических интерфейсах пользователя (GUI). Функциональное и логическое программирование в его чистом виде запрещает побочные эффекты функций, поэтому для взаимодействия с GUI приходится нарушать концептуальную целостность соответствующих языков.

Большинство интегрированных сред разработки основное внимание концентрируют на создании GUI, что является, безусловно, сложной, но чрезмерно специфичной задачей, которая выходит за рамки обучения основам программирования. Утверждается, что изучение основ алгоритмизации и программирования является само по себе достаточно сложной задачей, чтобы распылять внимание и силы учащихся на изучение многочисленных и несовместимых между собой сред разработки.

Поэтому мы считаем, что для обучения основам программирования в рамках мультипарадигмального подхода следует опираться на концепцию консольных приложений, которая достаточно универсальна для работы в любом языковом окружении. При этом внимание студента концентрируется не на особенностях системы программирования и средствах ввода-вывода, а на сути решаемой задачи и реализуемых алгоритмах.

Таким образом, одним из основных требований к используемому окружению является наличие развитой культуры консольных приложений.

Другим важным требованием является легкость в установке и сопровождении, а также доступность соответствующих программных средств. Так, некоторые коммерческие системы программирования следует исключить из рассмотрения по причине их дороговизны. Кроме того, задача поддержки различных проприетарных систем в условиях компьютерного класса на несколько десятков рабочих мест требует зачастую недопустимо высоких трудозатрат.

Ключевым вопросом в выборе программного обеспечения является выбор операционной системы. В отличие от языка программирования можно выбрать единую операционную систему, удовлетворяющую требованиям мультипарадигмального подхода. В настоящее время существует только две возможности: системы семейства Unix и MS Windows. Другие операционные системы (MacOS, VAX/VMS и др.) исключаются из-за слабой распространенности или дороговизны соответствующих аппаратных платформ. Популярная некогда система MS-DOS безнадежно устарела.

Большинство систем программирования под MS Windows не удовлетворяет всем перечисленным выше требованиям. Особенно следует отметить отсутствие в традициях MS Windows упоминавшейся выше культуры консольных приложений, в результате чего разработка программ на альтернативных языках программирования вызывает у студентов ложное ощущение ущербности используемых выразительных средств.

В то же время операционные системы семейства Unix предлагают широкий выбор свободных систем программирования, которые полностью удовлетворяют перечисленным требованиям

(развитая культура консольных приложений, единообразная дистрибуция и администрирование, доступность). В особенности следует обратить внимание на свободно распространяемые операционные системы (*BSD и Linux). В базовую конфигурацию дистрибутивов большинства таких систем входят средства разработки для альтернативных языков программирования.

На основе всего сказанного выше можно сделать однозначный выбор в пользу применения свободно распространяемого программного обеспечения при обучении студентов программистских специальностей в рамках мультипарадигмального подхода.

12.50 – 13.15

Григорий Панов
Марийский государственный технический университет

Интеграция автоматизированных систем учета системами управления взаимоотношений с клиентами

В последнее время большими темпами идет развитие систем взаимоотношений с клиентами, или, говоря короче, CRM систем. Это связано в первую очередь с осознанием компаниями преимуществ, которые предоставляют подобные системы. Обзор преимуществ. Существует большое количество коммерческих CRM систем, есть и несколько свободных продуктов. Перечисление свободных

CRM систем. Среди них выделяется система SugarCRM. Преимущества системы SugarCRM. То есть система обеспечивает большие возможности по организации взаимоотношений с клиентами. Но в ней не реализована возможность взаимодействия с другими системами, существующими в компании. В частности, с системами автоматизированного учета. Такое взаимодействие могло бы принести значительные выгоды, так как позволяет пользователям SugarCRM получать сведения о документообороте в режиме реального времени, что, в свою очередь, позволяет осуществить более эффективную организацию деятельности компании и получить в конечном итоге большую прибыль.

Но вследствии того, что используемые в настоящее время системы автоматизации учета являются в основном коммерческими, то осуществить интеграцию с такими системами проблематично из-за закрытости форматов данных и интерфейсов.

Выходом из данной ситуации является использование систем с открытым исходным кодом.

Приведем пример интеграции системы Ананас с системой SugarCRM.

Исходные данные (языки, архитектура). Архитектура системы Ананас спроектирована таким образом: есть клиентская часть, серверная часть и ядро, содержащее всю логику работы.

Было принято решение об использовании веб-сервисов для организации взаимодействия систем, поскольку архитектура системы Ананас позволяет легко реализовать вызовы методы веб-сервиса,

используя классы ядра. Протокол взаимодействия - SOAP. Используемая библиотека для реализации протокола SOAP на серверной части - gSOAP. Используемая библиотека для реализации протокола на клиентской части - встроенное расширение SOAP PHP5. Реализация довольно простая. Требования к клиентской части. (описание рисунка). И собственно реализация клиентской и серверной частей. Особенностью реализации серверной части является наличие модуля mod_gsoap. Система могла бы работать и без него, но при этом требовалось бы открыть для доступа какой нибудь порт. А при использовании mod_gsoap запросы передаются через 80-й порт. Трудности с использованием mod_gsoap под Apache2.

Какие возникли трудности:

1.пришлось дописывать код mod_gsoap.

2.пришлось разобраться со структурой системы SugarCRM.

3.проблемы с реализацией SOAP протокола в PHP5.

4.не решена проблема безопасности передачи данных.

5.не решена проблема инсталлятора для клиентской части.

В заключение хотелось бы отметить перспективность дальнейшего развития данного направления, т.е.

интеграции свободных систем между собой, т.к. оно осуществляется, как показала практика, довольно просто и позволяет получить очевидные преимущества.

13.15 – 13.40

Александр Сенько
МГТУ им. Н.Э. Баумана, БЕН РАН

Михаил Якшин
ALT Linux

Подходы к организации распределенных систем учета (ввод и каталогизация информации)

В настоящее время значительное внимание уделяется созданию систем учета различного рода документов. Подобные системы решают одни и те же задачи, строятся на основе сходных бизнес-процессов и могут быть заменены типовой системой ввода, учета и выдачи информации, настраиваемой под определенную предметную область и определенный круг задач.

Система строится из одной или нескольких однотипных компонентов-ячеек, обладающих определенными свойствами. В минимальном варианте система может состоять из одной ячейки - локального компонента (ЛК).

Основная задача ЛК – ввод, редактирование, хранение данных и представление их в форме различных отчетов и справок.

Компонент состоит из нескольких слабосвязанных

подсистем, выполняющих определенные функции и решающих определенные задачи. Каждая из таких задач может решаться на отдельном АРМ. Приблизительный состав подсистем и решаемых ими задач выглядит так:

• подсистема ввода данных (первичный ввод данных, проверка и подтверждение, редактирование ранее введенных данных);

• подсистема поиска (поиск по локальным данным, генерация отчетов по шаблонам);

• администрирование (настройка схемы данных, управление правами и привилегиями пользователей).

В качестве интерфейса предлагается использовать Web-интерфейс, широко применяемый для систем подобного рода, где присутствует одна база данных и множество различных АРМ для работы с ней. Такой подход также легко позволяет организовать доступ к системе как с локальных рабочих мест внутри организации, так и извне (для авторизованных пользователей).

Для создания более сложных систем имеет смысл применять распределенный подход. В этом случае система строится из нескольких ЛК, объединенных в единую сеть. При этом каждый из компонентов дополняется определенной функциональностью, позволяющей обмениваться данными с другими компонентами. В этом случае мы можем говорить уже о распределенном компоненте (РК).

К подсистеме ввода данных добавляется функция импортирования данных, а к подсистеме поиска – возможность выгрузки (экспортирования) данных в другой РК.

РК могут быть связаны с использованием одной из двух схем:

• интеграция в единое хранилище – импортирование данных из подчиненного компонента в

вышестоящий;
• виртуальная интеграция – данные физически хранятся только в одном РК, передача данных между БД не происходит; поиск и получение информации выполняется с помощью распределенного запроса;
Первую схему предполагается использовать в следующих случаях:
• затруднена передача данных между РК по постоянному каналу (физически нет быстрого доступа в Интернет, повышенные требования к безопасности и т.п.);
• данные в нижестоящей организации обновляются достаточно редко и предпочтительнее иметь их экспортированную копию в вышестоящем РК.
Преимуществами второй схемы являются:
• снижение требований к программно-аппаратным комплексам каждого РК в отдельности (вследствие существенного снижения объемов хранимой информации);
• максимальная актуальность получаемых при поиске результатов (т.к. нет промежуточных синхронизаций).
Для реализации второй схемы необходимо дополнить поисковую систему каждого РК модулем поддержки распределенных запросов (на основе технологии XMPP/Jabix). При этом также усложняется решение вопросов безопасности.
Стоит отметить, что при организации сложных сетей с использованием вышеизложенных принципов обе схемы могут комбинироваться для создания сетей смешанной интеграции.

При построении сложных сетей со структурой, отличной от древовидной, возникает проблема дублирования данных – не имеет значения, какая при этом используется схема интеграции. Одним из

решений этой проблемы является четкое соблюдение древовидной структуры (организационная мера). При виртуальной интеграции работа по выявлению дублей в результатах поискового запроса возлагается на специальный агент-интегратор сети Jabix – независимый модуль, не являющийся частью рассматриваемой системы.

Разработку системы целесообразно разделить на четыре этапа, в соответствии с изложенными выше структурными элементами:

1. Разработка полнофункционального ЛК;
2. Разработка РК: добавление к ЛК функций обмена данными между собой;
3. Добавление к РК поддержки распределенного поиска и получения информации на основе технологии XMPP/Jabix;
4. Коммутация РК с различными схемами данных.

Первый этап является основным и наиболее трудоемким. По его окончании мы имеем полнофункциональную локальную систему с on-line доступом.

По завершении второго этапа мы можем строить распределенные системы с интеграцией в единое хранилище.

Завершение третьего этапа дает нам возможность строить системы с виртуальной и смешанной интеграцией. При этом трудозатраты на написание конвертеров для системы Jabix минимальны, т.к. на этом этапе схема данных во всей системе остается единой, следовательно, и все конвертеры одинаковы.

Четвертый этап позволяет перейти к системам с различными схемами данных внутри себя (или – как вариант – к интеграции систем, созданных на предыдущем этапе). Этот этап можно разделить на два – в зависимости от степени различия схем данных:

1. различаются (и могут настраиваться динамически – в процессе работы системы) только наборы атрибутов отдельных сущностей, сами сущности и связи между ними одинаковы и фиксированы;
2. в дополнение к предыдущему могут различаться (а, возможно, и динамически изменяться) и наборы самих сущностей и их связи.

В первом случае задача ведения БД в каждом из РК не представляет особых сложностей; создание конвертеров для импортирования, экспортирования данных и поддержки Jabix можно автоматизировать.
Трудозатраты и возможные проблемы во втором случае пока оценить довольно трудно. Положительным является тот момент, что реальных задач, требующих создания подобных систем, пока нет.

13.40 – 14.30: Обеденный перерыв

Дневное заседание (14.30 - 17.00)

14.30 – 14.55

Юрий Седунов, Андрей Паскаль
Москва, ALT Linux

Кроссплатформеная модельная реализация учетной системы для нужд электронного государства

Доклад посвящен проекту разработки макета учетной

системы, выполняемому ALT Linux по заказу Министерства экономического развития и торговли РФ. Рассматриваются современные требования, предъявляемые к учетным системам, как средствам ведения электронного административного учета. Рассказывается об архитектуре разрабатываемой в рамках проекта учетной системы и основных подходах к ее реализации.

14.55 – 15.20

Андрей Стрельников
Марийский государственный технический университет

Применение бизнес правил в системах разграничения доступа

15.20 – 17.00

Круглый стол "Свободное программное обеспечение в электронном государстве" Ведущие Анатолий Якушин, Алексей Новодворский.

16.40 – 17.00: Кофе

Вечернее заседание.

17.00 – 18.30: Продолжение круглого стола
18.30 – 19.00: Закрытие конференции.

Дополнительное выступление

27 июля, 9:00

Лепихов К.А.

Новые технологии и проекты сообщества Mozilla.org

Аннотация

Доклад посвящен обзору новых проектов сообщества Mozilla. Также дано краткое описание языка XUL и технологий, которые могут быть использованы с его помощью. В конце доклада производится взгляд в будущее разработок сообщества Mozilla.

Новинки в CVS

Конец 2004 года и начало 2005 года принесли много нового для разработчиков и простых пользователей сообщества Mozilla - в дереве разработки произошло заметное оживление по части обновления кода старых проектов и "вливания" новых разработок. Также нельзя обойти без внимания событие осени 2004 – выход браузера Mozilla Firefox 1.0 и последующие за ним новые версии почтового клиента Thunderbird и календаря-планировщика Sunbird. Итак, посмотрим, что нового появилось в CVS.

Rhino

Долгое время в Mozilla была своя реализация JavaScript (или ECMAScript, стандартизованный в документе ECMA-262), написанная на языке C. Хорошо это или плохо, но это существенно сужало свободу выбора и переносимость js компилятора - т.к. он был архитектурно зависим и плохо поддавался расширению функциональности. Все эти проблемы решает проект Rhino - реализация

JavaScript на языке Java. Rhino позволяет:

- реализация всех возможностей JavaScript 1.5;

- поддержка прямого скриптинга из Java;

- JavaScript shell для запуска JavaScript скриптов;

- Компилятор JavaScript для преобразования текста на JavaScript в класс на Java.

В дополнении ко всему, в Rhino реализован интерфейс JavaAdapters, который позволяет реализовать в JavaScript любой Java интерфейс, или наоборот, расширить существующий Java класс js объектами. Rhino поддерживает возможность интернационализации (все сообщения JavaScript engine можно перевести). Rhino достаточно безопасен, т.к. позволяет полностью отследить цепочки выполнения кода для локализации проблемы.

XForms

Web формы в современном мире играют все более важную роль и их разработчикам пора обратить на

них внимание ;) XForms создан на базе известных W3C стандартов XML Schema, XPath, и XML Events, и он может решить те проблемы и ограничения, которые существуют на сегодняшний день для создания форм с помощью текущей HTML модели. Дополнительные возможности, которые часто включают в web формы – это поддержка проверки данных, введенных клиентом, обработка событий, и run-time зависимость элементов формы друг от друга. В HTML, все эти возможности приходится решать с помощью громоздких JavaScript библиотек, которые необходимо обновлять в случае изменения параметров в форме. XFroms позволяют избежать большинства этих проблем, т.к. реализуют весь описанный функционал внутренними средствами, при этом создатели форм могут использовать XML для их создания.

Цель проекта XForms для Mozilla - реализация XForms 1.0 для продуктов Mozilla. Код поддержки XForms включен в дерево разработки mozilla, существуют сборки браузеров под различные платформы с поддержкой XForms.

XUL и XULRunner

XUL или XML User Interface Language, это, как видно из его названия, язык, который позволяет создавать достаточно богатые по функциональности пользовательские интерфейсы, которые можно запускать ("отрисовывать") как стандартные приложения, так и загружать их из сети Internet. При этом приложения на XUL можно легко настраивать, менять в них текст или графические объекты, переводить на различные языки и тд. Для программирования на XUL не требуется специальных

навыков, любой web-разработчик, знакомый с Dynamic HTML (DHTML), сможет быстро выучить XUL и начать создавать свои приложения.

Отличительные особенности XUL:

- Мощный язык разметки с поддержкой пользовательских элементов (widgets). В отличие от DHTML, с помощью которого можно создавать web страницы, с помощью XUL можно создавать переносимые приложения, содержащие окна, кнопки и ссылки.

- Основан на существующих стандартах. XUL - это язык XML, основанный на стандарте W3C XML 1.0. Приложения, написанные на XUL, используют дополнительные стандарты W3C как HTML 4.0; Cascading Style Sheets (CSS) 1 and 2; Document Object Model (DOM) Levels 1 2; JavaScript 1.5, включающий ECMA-262 Edition 3 (ECMAscript); XML 1.0.

- Межплатформенная переносимость. XUL может быть использован на любой платформе, поддерживаемой Mozilla.

- Отделенная логика для отображения и формирования интерфейса.

- Легкость локализации, модификации или настройки.

XULRunner

С точки зрения mozilla, XUL приложения ничего не отличается от HTML – сначала скачивается объект, потом происходит его рендеринг и отображение – таким образом, можно запускать пользовательские приложения, расположенные в сети, а не на локальном диске клиента. Эти приложения распространяются в виде обычного .jar архива с xul контентом и библиотеками. Пример таких приложений - extensions для Mozilla Firefox или Mozilla Thunderbird. Но есть одна проблема - пользователь вынужден ставить себе браузер для запуска таких приложений, что затрудняет распространение ПО на базе XUL из-за большого размера сопутствующего ПО. XULRunner и XRE (XUL Runtime Evironment) - попытка избавиться от этой зависимости, позволяет запускать XUL приложения обычным образом: через программу-загрузчик (XULRunner), или посредством встраиваемых библиотек для рендеринга (XRE).

Будущее продуктов Mozilla

На сегодняшний день, в развитии проектов Mozilla можно отметить ряд ключевых моментов:

- миграция на XUL/XULRunner (Firefox 2.0/Mozilla 2.0)

- создание Mozilla как платформы для Web разработки (аналог NSS как платформа для PKI разработки).

- реализация самых последних W3C стандартов в продуктах Mozilla(XForms,MathML,SVG).

Тезисы присланные на конференцию

Бартунов О.С., Сигаев Ф.Г.

Написание расширений для PostgreSQL с использованием GiST

Приводится краткое описание обобщенного поискового дерева (GiST) в PostgreSQL, обзор популярных модулей и пример написания пользовательских расширений с использованием GiST.

Полностью текст статьи приводится на http://conf.altlinux.ru