



№67, 26 августа 2010

Обзор среды разработки Anjuta

Система распределённых вычислений BOINC

Qt 4 в Haiku OS: почувствуй себя комфортно

**Схемотехника в Linux с помощью gEDA:
создание паттерна**



Колонка главного редактора



Близится осень, а это означает, что совсем скоро выйдет из своей привычной спячки корпоративный мир. Маркетологи начнут радовать (или огорчать) общественность пресс-релизами о новых инициативах и продуктах компаний. Толпы интересующихся и просто зевак соберутся на выставках, конференциях и мастер-классах. Интернет наполнится пользователями и... результатами их деятельности. Так происходило раньше, и мало что изменилось в этот год. Единственное – возникает ощущение, что со временем данный эффект (глобального «выпадения» лета из корпоративной жизни) сглаживается. Возможно, ощущение обманчивое, потому что лагерь Open Source здесь всегда немного отличался от остальной «тусовки» ИТ. Ведь даже в то время, когда практически исчезают новости от крупных компаний, в сообществе находятся энтузиасты, поддерживающие ленту событий в подвижном состоянии.

И приятно видеть, что в мире FLOSS еще очень много (больше, чем в других областях) людей, которые трудятся в свое удовольствие. Пусть для них эта работа может и не являться основной, именно такие энтузиасты зачастую совершают маленькие революции в своей любимой области. А весь мир узнает о них уже из тех самых пресс-релизов, презентаций на выставках и публикаций в интернете...

Главный редактор
Дмитрий Шурупов
(osa@samag.ru)

«Open Source»
электронное приложение к журналу
«Системный администратор»
№67, 26 августа 2010 г.

РЕДАКЦИЯ

Исполнительный директор

Владимир Положевец

Главный редактор

Дмитрий Шурупов

Верстка и оформление

Владимир Лукин

Иллюстрация на обложке

Анна Герцова

Сайт электронного приложения:

<http://osa.samag.ru>

За содержание статей ответственность несет автор.
Все права на опубликованные материалы защищены.

Новости мира Open Source

В Vim 7.3 включили исправления и улучшения за 2 года

В ночь на 16 августа был представлен новый релиз популярного консольного текстового редактора Vim («Vi Improved») – 7.3.

В новую версию Vim – 7.3 – вошли изменения, работа над которыми велась в течение двух последних лет (с момента выпуска Vim 7.2). Поэтому, несмотря на то, что релиз формально является «минорным», авторы утверждают, что «он не такой уж и минорный». С другой стороны, до крупного (major) он тоже не дотягивает. Так или иначе, к основным изменениям в Vim 7.3 разработчики относят следующие:

- ☑ «постоянная» функция отмены действий (операции для «undo» записываются в отдельный файл, так что могут быть выполнены даже после прекращения работы с документом и возобновления его редактирования);
- ☑ поддержка шифрования по алгоритму Blowfish;
- ☑ поддержка скрытия части текста;
- ☑ интерфейсы для языков Lua и Python 3.

Кроме того, можно отметить заметные подвижки в работе Vim с Perl: представлен ряд исправлений для Perl 5, а также появилась поддержка Perl 6.

Open Source помогает Virgin America

По словам ИТ-директора Virgin America, американской авиакомпании, специализирующейся на бюджетных перелетах самолетами, в ИТ-инфраструктуре преимущественно используется Open Source, и это оправдывает себя не только с экономической точки зрения.

Рави Симхамбхатла (Ravi Simhambhatla), CIO в Virgin America, отметил, что переход компании с проприетарного программного обеспечения на Open Source позволил сэкономить миллионы долларов. Но, что интересно, не это стало главной причиной: «Мы перешли на Open Source не из-за бесплатности, а потому, что это ПО просто работает». Такая позиция («Open Source, в отличие от проприетарных продуктов, просто работает») пока не очень распространена среди крупных корпоративных пользователей, однако она позволяет зафиксировать, безусловно, приятную для всего FLOSS-сообщества тенденцию.

Из используемых программных продуктов в Virgin America отмечаются операционная система из линейки Red Hat Enterprise Linux, СУБД MySQL, сервер приложений Apache Tomcat, свободная VPN-реализация OpenVPN, а также Apache SpamAssassin для фильтрации спама в корпоративной почте.

Ubuntu Linux 10.10 лишилась SPARC и IA64, получает жесты и multitouch

В следующем релизе Ubuntu Linux – 10.10 «Maverick Meerkat» – не будет поддержки архитектур SPARC и IA64. Их поддержка была под вопросом уже долгое время. В начале июня Скотт Джеймс Ремнант (Scott James Remnant) предупреждал, что если не найдутся энтузиасты, готовые заниматься портами Ubuntu Linux для этих архитектур, их поддержка будет убрана к моменту заморозки релиза 10.10 «Maverick Meerkat». И теперь, в августе, «опасения» подтвердились, и техническим советом проекта было принято соответствующее решение.

Тем временем, Марк Шаттлворт (Mark Shuttleworth) сообщил в своем блоге, что в тестовых сборках Ubuntu 10.10 появились первые версии специального фреймворка UTouch, разработанного в Canonical для поддержки сенсорных дисплеев. В качестве тестовой площадки для проверки корректности функционирования жестов сейчас используется компактный ноутбук/планшет Dell Latitude XT2, 12,1-дюймовый экран которого поддерживает multitouch. Команда разработчиков Ubuntu решила развить идею жестов до целых «предложений», т.е. возможности задавать команды из целых наборов последовательно выполняемых жестов. Весь программный код опубликован на Launchpad под свободными лицензиями GPL v3 и LGPL v3.

Ожидается, что в Ubuntu Linux 10.10 лишь немногие GTK-программы будут поддерживать скроллинг с помощью жестов. В Evince появятся некоторые дополнительные возможности, которые могут заинтересовать разработчиков (и они захотят включить подобные функции в свои приложения).

Управление окнами с помощью жестов появится в Unity, который войдет в состав Ubuntu Linux 10.10 Netbook Edition.

Red Hat предложила еще 3 года поддержки для RHEL

Американская Linux-компания Red Hat объявила о запуске новой услуги: ELS (Extended Life Cycle Support) позволяет продлить техническую поддержку по RHEL (Red Hat Enterprise Linux) на три дополнительных года.

Таким образом, теперь вместе с RHEL клиенты могут получить гарантированную поддержку аж на 10 лет. Данная инициатива Red Hat придется очень кстати для пользователей, до сих пор работающих с выпущенным в 2003 году RHEL 3. Их поддержка завершилась бы уже в октябре этого года, но вместе с ELS она может быть продлена.

Поддержка ELS направлена на устранение критических ошибок и уязвимостей, а не привнесение каких-либо новшеств в дистрибутивы. Более того, она распространяется не на все компоненты дистрибутива, а только на ключевые в ИТ-инфраструктуре пакеты. Например, в списке пакетов, не поддерживаемых по ELS для RHEL 3, можно увидеть рабочие среды GNOME и KDE, офисный пакет OpenOffice.org, веб-браузер Firefox, почтовый клиент Evolution и другие десктоп-приложения.

Ilummos уходит от OpenSolaris, становясь самостоятельным проектом

По последней информации от Гаррета Д'Амора (Garret D'Amore), лидера инициативы Ilummos, проект уходит от OpenSolaris и становится самостоятельным форком, который будет развиваться без оглядки на своего «родителя».

В связи с недавним известием о том, что Oracle фактически «похоронила» проект OpenSolaris, Гаррет высказал предположение, что последний коммит в публичный репозиторий OpenSolaris был уже сделан (18 августа). Теперь инициативный Open Source-сообществом проект Ilummos должен начинать свой собственный путь становления и развития.

Автор высказывает радость по этому поводу, поскольку, если бы проекту пришлось сохранять соответствие кода Ilummos дереву исходников OpenSolaris, то в проекте оставалось бы не так много места для инноваций. Кроме того, Д'Амор сообщил, что недавние изменения Ilummos уже позволили производить сборку всей системы на самой системе (т.е. собирать Ilummos на компьютере с Ilummos), что является важным достижением.

Можно также отметить недавний релиз основанной на OpenSolaris платформы Nexenta Core Platform 3.0. Ожидается, что уже начиная со следующей версии она будет основана на кодовой базе Ilummos.

Вместе с тем, как и предполагалось, совет сообщества по развитию OpenSolaris (OpenSolaris Governing Board, OGB) снял с себя все полномочия и прекратил существование. Окончательное решение было принято 23 августа, во время последней встречи OGB.

GNU/Linux установили во всех школах Дзержинска

По инициативе городского управления образования на персональные компьютеры всех школ и образовательных учреждений города Дзержинск (Нижегородская область) была установлена операционная система GNU/Linux.

Об этом стало известно от Сахаровой Валентины Владимировны, заместителя начальника управления образования администрации Дзержинска. Необходимость в переводе школ на GNU/Linux и пакет свободного программного обеспечения была вызвана тем, что 1 января 2011 года заканчивается срок действия лицензии на базовый пакет проприетарного программного обеспечения, установленного в школах в рамках государственной программы «Первая Помощь».

Почти на 500 ПК в 50 образовательных учреждениях (39 школ и 11 специализированных учебных заведений) был установлен дистрибутив российской компании ALT Linux – «Альт Линукс Школьный Юниор». Установку производили специалисты нижегородской компании ООО «Элсис» с привлечением участников Нижегородской группы пользователей Linux, сотрудников PingWin Software и нижегородского представительства «ГНУ/Линуксцентр».

Управление образования администрации Дзержинска планирует за оставшееся до нового года время провести обучение учителей, чтобы полностью подготовить школы к переходу на свободное программное обеспечение.

Дмитрий Шурупов,
по материалам www.nix.ru
(osa@samag.ru)

Обзор среды разработки Anjuta

Обзор среды разработки Anjuta я начну с Qt Creator. В свое время перешел на него с KDevelop по двум причинам. Новые версии Qt Creator выходили очень часто и «весили» совсем немного. Qt Creator был быстр и надежен. А потом Qt, будучи удобной библиотекой для программистов на C++ старой закалки, получила уклон в «мобильные системы» и начала превращаться в эдакую платформу а-ля Java с набором классных виджетов. Появился даже язык Qt Quick и соответствующие призывы, что теперь вам не нужно знать C++: «Qt Quick и визуальные средства разработки! Создай программу для мобильного телефона за три щелчка мыши!» Конечно, я немного утрирую... Но дело в том, что к соответствию текущим веяниям в развитии Qt устремился и Qt Creator. Он становился больше, менее удобным для меня и менее предсказуемым в работе. Так, например, на каком-то этапе развития этой среды мои проекты перестали компилироваться. Когда я поборол эту проблему, оказалось, что файлы локализации в случае запуска моих программ в Qt Creator перестали загружаться – подставлялась локаль «C» –

так почему-то действовал вызов `QLocale::system().name()`. Также иногда не загружались картинки из ресурсов. Прошло довольно много времени, пока я понял, что Qt Creator ведет себя так лишь в случае, если после установки согласиться с запуском IDE – там, где галочка вроде «Запустить Qt Creator сейчас». Вот когда Qt Creator запускается таким образом, он и ведет себя «непредсказуемо». Я об этом не знал и принимал такое поведение за постоянное.

В то время великого огорчения решил перейти на другую среду разработки или хотя бы подыскать ее – чтобы было к чему обратиться в случае, когда пользоваться Qt Creator по каким-то причинам не смогу или не захочу. На ум сразу пришли две среды разработки: KDevelop и Anjuta. Anjuta я пробовал очень давно, чуть ли не на заре её создания, а к KDevelop испытываю смешанные чувства. С одной стороны, я очень долго пользовался KDevelop3 (несколько лет). Тогда я писал программы под GTK+, а KDevelop3 был универсальной IDE без явного уклона в C++/KDE. Новый KDevelop4 мне почему-то представлялся чудовищ-

но большим и громоздким – впрочем, как я выяснил позже, это были напрасные опасения. Но душа моя метнулась в сторону Anjuta (<http://projects.gnome.org/anjuta/>).

Первый взгляд

Anjuta, хотя формально и относится к нише программного обеспечения для GNOME, остается средством универсальным, не заточенным сугубо под C или C++ вместе с GTK+/GNOME. Для удобства при создании нового проекта дается выбор из множества вариантов: это и GTK-проекты, и программы под SDL и wxWidgets, и Java, и Python. Хотя логотипом Anjuta служит лошадка, программа названа в честь любимой девушки разработчика-создателя Набы Кумара.

Дистрибутив Anjuta невелик и без труда устанавливается из исходников – эти два фактора всегда внушают мне уважение к программе. Так я знаю, что не завишу от сборщиков пакетов. «Без труда устанавливается из исходников» следует трактовать так: в любом современном дистрибутиве Linux найдутся все нужные для сборки средства. И при этом не надо будет отпрашиваться к шаманам для науки бить в бубен. Кстати, Anjuta разделена на две части: основную и пакет дополнений. Вообще, Anjuta основывается на плагинах – даже движок текстового редактора можно выбрать: GtkSourceView (по умолчанию) или Scintilla. У последней свои настройки и возможности, в частности свёртывание блоков кода. Плагины включаются и выключаются в «Правка → Настройка → Общие → Установленные модули». Например, по умолчанию отключен модуль отладчика, а когда вы его включаете, появляются соответствующие пункты меню (как основного, так и контекстных), а также отладочная панель и окно точек останова.

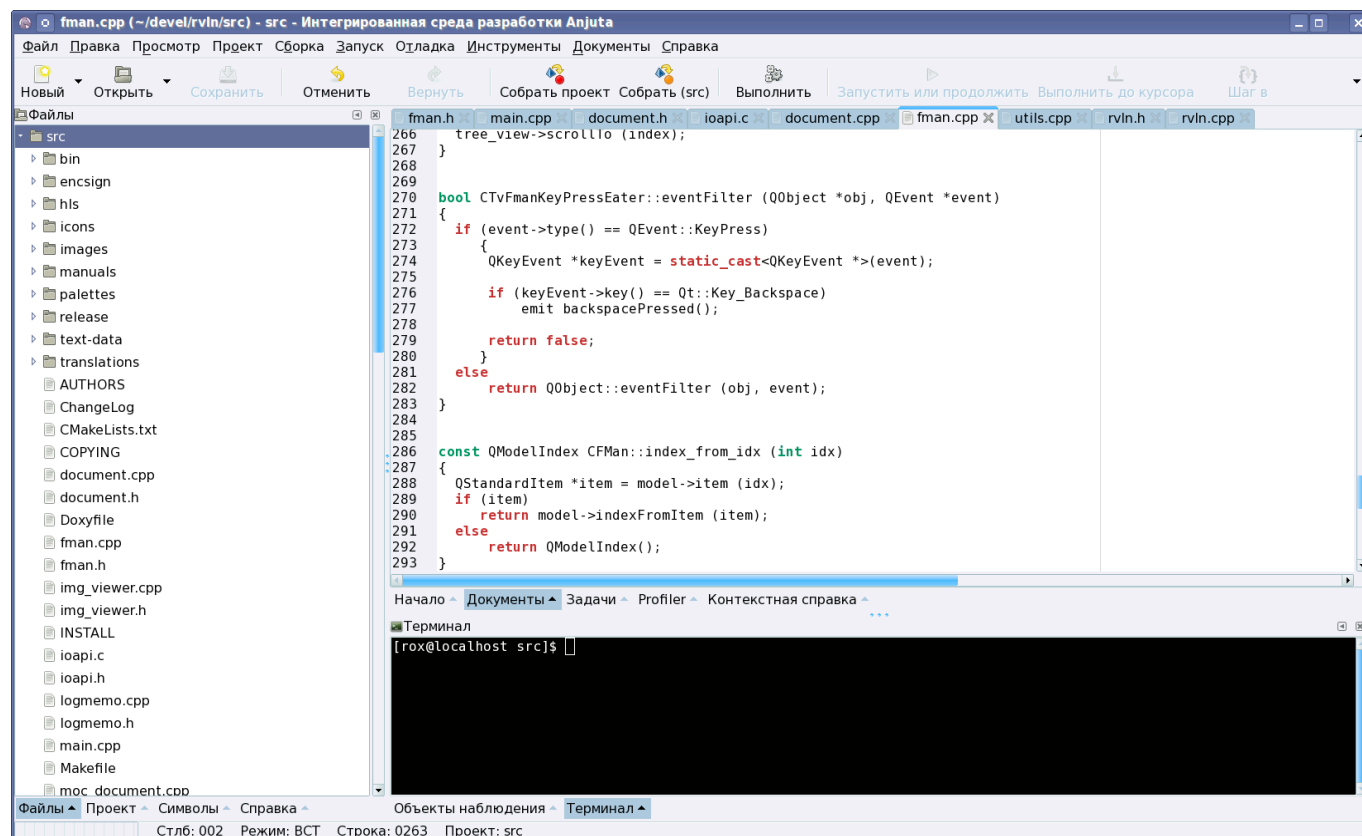
Первым делом после установки и запуска Anjuta я открыл в среде свой проект Qt-программы. Конечно, не сам файл проекта от Qt Creator и не Qt-файл проекта, а просто создал новый проект на основе существующих исходников. Управление собственно настройками сборки осталось на ручной правке про-

файла, а Anjuta взяла на себя сборку по готовому makefile, запуск такого-то исполняемого файла (с возможностью правки переменных окружения для сеанса запуска) и правку кода с различными возможностями навигации по нему. Это именно те задачи, которые мне нужны от любой IDE. Ну, и редкий запуск отладчика. Вместо отладки в gdb я предпочитаю самостоятельный вывод вспомогательных данных на консоль.

Интерфейс, возможности, впечатления

Интерфейс у Anjuta меня не озадачил – вполне традиционный для IDE: слева панель с файлами, посередине вкладки редактора, снизу лог и консоль. Меню и сообщения переведены на русский. Там, где файловая панель, есть три вкладки: «Файлы», «Проект» и «Символы». «Файлы» – наиболее удобный способ открывать файл из текущего каталога, места расположения исходника. Вкладка «Проект» служит для отображения иных данных – структуры проекта согласно проекту automake. То бишь, допустим, объектные файлы и имена связанных с ними исходников, цели сборки, и так далее. Наконец, на вкладке «Символы» представлены все функции и константы. При клике по имени осуществляется переход к телу (определению) функции.

Из контекстного меню (например, на имени функции) тоже можно перейти к её телу. А вот к объявлению не получается, хотя есть и соответствующий пункт меню – «Объявление метки». Однако он работает так же, как и «Реализация метки», то есть переносит вас к первой строке определения функции. Во всяком случае, для исходника на C++ это получается именно так. В пакете дополнений есть средство, показывающее в виде своеобразной карты всю структуру наследования классов в проекте. На снимке экрана, помещенном на сайте Anjuta, это смотрится здорово: диаграмма, разные стрелочки, списки функций-членов и переменных. Однако для меня остается загадкой, как это работает на практике. На панели этого «графика наследования» (Inheritance Graph) в контекстном меню есть пункт «Обновить», но при нажатии у меня ничего не происхо-



Общий вид Anjuta

дит. Какой еще негативный опыт я получил в Anjuta? Был такой нюанс: запускаю из среды свою программу, потом закрываю её. Встроенный терминал предлагает нажать Enter, иначе в следующий раз при запуске будет задавать лишние вопросы. Я искал, где можно отключить это, но не нашел. Как результат – лишние щелчки мышью и нажатия на клавиатуру. Однако лучше расскажу, что мне понравилось.

Начну с поиска. Благо, разработчики всё большего количества программ отказываются от окон поиска, заменяя их полями ввода в том же окне, где и происходит редактирование. По крайней мере, три замечательные IDE: Qt Creator, KDevelop и Anjuta – в этом солидарны. Правда, в Anjuta клавиша <F3>, которая традиционно закреплена за функцией «Найти дальше», по умолчанию служит запуску программы. Но настройки можно переопределить, хотя и тут есть небольшая трудность: пунктов меню много, а никакого поиска по ним в настройках сочетаний клавиш нет (в Qt Creator, например, есть). Эта мелочь – в копилку уже описанного лишнего нажатия Enter во встроенном терминале: они тормозят работу, они излишни, без них можно обойтись.

В пакете дополнений к Anjuta есть два полезных средства для отладки: профилировщик (показывает скорость выполнения той или иной функции и кто её вызывает) и плагин работы с valgrind, чтобы обнаруживать утечки памяти. Я уже не говорю о взаимодействии с gdb – все функции отладки и отладочные сведения доступны из самой Anjuta. Это сделано, на мой взгляд, очень удобно. Мелочи вроде функций «Создать пакет» (выполняет то же, что и make dist), «Заплатка» (выбираете файл с патчем, и он применяется к исходнику) – это тоже монетки в копилку приятных впечатлений. Другая удобная вещь – «Задачи» – TODO для проекта, да и не только для него, эдакий ежедневник. Можно привязывать задачи к датам, а можно и просто заносить что-то на память. Кроме того, есть встроенный справочник по API – по сути, тот же DevHelp (<http://live.gnome.org/devhelp>). Наличие той или иной документации зависит от установленных из вашего Linux-дистрибутива пакетов. Традиционно в нужном формате доступна документация к GNOME, GTK+, GtkSourceView2, Poppler и еще некоторым библиотекам. Пользователям Qt не повезло, зато у них есть подобная возможность в Qt Creator. Кстати, в Anjuta работает и контекстная справка – конечно, только по тем API, документация к которым установлена. Ставите курсор на имя функции, нажимаете <Shift> + <F1> – вуаля!

В редакторе есть автоматическое дополнение на основе просканированного кода. Когда имя функции уже введено, дается подсказка по её параметрам. Да и вообще редактор и его подсветка с настройками по умолчанию – очень приятные. Еще порадовала функция «Выделить блок кода». Нечто подобное есть в Qt Creator, но там можно выделять блок либо от курсора до начала блока, либо до конца. Оба движка редактора: GtkSourceView и Scintilla – оснащены внушительным набором модулей подсветки, покрывающим все популярные и не совсем популярные языки программирования. В составе Anjuta идет большой предустановленный набор кусочков кода – сниппетов, названных в программе, однако, макросами. Что до сниппетов, которых сами разработчики прямо именуют «сниппетами», то в недрах зреет навороченный плагин редактора сниппетов, о котором подробно рассказывает в своем блоге один из разработчиков Dragos Dena (<http://dragos-dena.blogspot.com/2010/07/anjuta-snippets-plugin-weekly-report-7.html>).

Как и любая полновесная среда разработки, Anjuta поддерживает системы контроля версий: Git, Subversion и CVS. Всё это работает через отдельные модули, которые можно включать в окне настроек, на вкладке «Общие → Установленные модули». Кстати, там по умолчанию много чего выключено – например, модуль поддержки Valgrind.

Итоги

Подведу итоги. Если бы я писал программы, завязанные на GTK+ и autotools, то выбрал бы в качестве IDE именно Anjuta. Но поскольку текущие мои предпочтения «платформы» лежат в области Qt, я разрываюсь между KDevelop и Qt Creator – в зависимости от того, какая из сред разработки вдохновляет меня в такой-то момент больше.

Работа над Anjuta не прекращается вот уже 11 лет, начиная с 1999 года. Любители истории могут посетить сохраненный на archive.org сайт проекта за начало нынешнего века – <http://web.archive.org/web/20010924065759/anjuta.sourceforge.net>. Весь современный процесс разработки полностью открыт – можно даже прямо на сайте наблюдать, какие патчи находятся на рассмотрении, или просматривать код через Git. В общем, Anjuta – отличный пример добротного свободного ПО.

Петр Семилетов
(tea@list.ru)

Система распределённых вычислений BOINC



Гуртом і батька легше бити

Украинская народная мудрость

Эта статья посвящена BOINC (Berkeley Open Infrastructure for Network Computing, <http://boinc.berkeley.edu>) – открытой программной платформе университета Беркли для распределённых вычислений. BOINC представляет собой программный комплекс с открытым исходным кодом (и распространяется под лицензией LGPL), который состоит из серверной и клиентской частей. Изначально разрабатывался специально для крупнейшего (и наиболее известного) проекта распределённых вычислений – SETI@home, занимающегося поиском радиосигналов

внеземных цивилизаций. Впоследствии разработчики сделали платформу доступной для сторонних проектов. Сегодня BOINC является универсальной платформой для вычислительных проектов в различных областях науки: математики, физики, биологии, астрономии, климатологии, криптографии.

С чего всё начиналось?

Началом поиска внеземных цивилизаций при помощи обработки сигналов, поступающий с радиотелескопов, можно считать 1959 год, когда в журнале Nature была опубликована статья, в которой предлагалось искать в радиосигналах повторяющиеся области, свидетельствующие об их искусственном происхождении. В 1960 году начались первые поиски по програм-

ме SETI (Search for Extraterrestrial Intelligence – поиск внеземного разума). С одной стороны данный проект требовал колоссальных технических и финансовых затрат, а с другой – пользовался крайне низкой поддержкой. За долгие годы наблюдений на складах накопились километры плёнки с записанной, но не обработанной информацией. Дело в том, что для обнаружения периодически повторяющихся элементов в радиосигнале требуется выполнять преобразование Фурье (http://ru.wikipedia.org/wiki/Преобразование_Фурье), а это очень трудоёмкая задача (особенно учитывая объёмы информации, требующей обработки).

Поначалу обработка сигнала осуществлялась на суперкомпьютерах, установленных непосредственно на телескопах, но в 1995 году Дэвид Геди предложил использовать виртуальный суперкомпьютер, состоящий из большого числа подключённых к интернету обыкновенных настольных ПК. Так появился проект SETI@home (SETI at home – «SETI дома»). Первоначальный клиент не был похож на современный: он мог работать только с одним проектом SETI@home и имел проблемы с безопасностью. Для устранения этих недостатков в феврале 2002 года была запущена работа над системой BOINC. Первый проект, работающий на новой системе Predictor@home, стартовал в середине 2004 года. С 2005 года и SETI@home был переведен на использование системы BOINC.

К 26 сентября 2001 года проект SETI@home выполнил 1021 операций с плавающей точкой, попав в Книгу Рекордов Гиннеса как крупнейший вычислительный процесс в истории человечества. По состоянию на ноябрь 2009 года BOINC объединяет около 300 000 активных компьютеров (всего – 2,4 миллиона зарегистрированных компьютеров) и имеет производительность 769 Тфлопс («тера» – 10^{12}), что приблизительно соответствует мощности шестого суперкомпьютера из таблицы Топ 500 (<http://top500.org>) за июнь этого года. Крупнейшим же проектом, использующим систему BOINC, является Folding@home. Он посвящён симуляции свёртывания белков, что должно помочь в лечении ряда болезней (<http://ru.wikipedia.org/wiki/Folding@home>). По состоянию на 1 февраля 2010 года производительность данного проекта оценивалась около 8,542 Тфлопс («пета» – 10^{15}), что сопоставимо с первыми строчками рейтинга суперкомпьютеров.

Научный вклад, достигнутый с помощью BOINC, можно оценить, посетив страницу (<http://boinc.berkeley.edu/wiki/Publications>

[by BOINC projects](#)), где собраны все научные публикации, сделанные в рамках проектов. Интересно, что один проект – Enigma@Home – ставит своей целью расшифровать три оригинальных сообщения, закодированные легендарной электро-механической криптографической машиной Энигма и перехваченные в северной Атлантике в 1942 году. На данный момент два сообщения расшифрованы, а к третьему было применено двойное шифрование, и перехвачено оно с потерями символов (http://www.3dnews.ru/software-news/enigma_home_proekt_vzloma_shifrov_vtoroi_mirovoi_voini), что усложняет расшифровку.

Как это работает?

Итак, как работает система BOINC и что необходимо сделать, чтобы присоединиться к этому проекту? Как уже упоминалось, BOINC состоит из серверной и клиентской частей.

В основе серверной – набор PHP-скриптов, необходимых для управления проектами: регистрации участников, выдачи заданий, получения результатов, управления базами данных проектов.

Клиентская часть, так называемый BOINC-клиент или BOINC-менеджер, представляет собой универсальный клиент для работы с различными проектами, управляемыми сервером на базе BOINC. Он позволяет участвовать одновременно в нескольких распределённых проектах.

Менеджер написан с использованием кросс-платформенной библиотеки WxWidgets и работает на большинстве современных операционных систем: GNU/Linux x86 и x64 (пакеты есть для популярных дистрибутивов), Mac OS X 10.4.0+, Windows XP/Vista/7 (в том числе и 64-битные версии). Энтузиасты расширяют список доступных платформ благодаря подготовке сторонних сборок (<http://boinc.berkeley.edu/trac/wiki/DownloadOther>). Кроме этого, имеется возможность использования GPU – процессоров графической карты (в случае, если ваша видеокарта поддерживает такую работу). Подробнее об установке можно прочитать здесь (<http://boinc.berkeley.edu/download.php>).

Первый вопрос, который следует решить: к какому проекту присоединиться? Вопрос на самом деле не простой, учитывая количество доступных проектов. Когда этот пункт выполнен, следует установить менеджер проектов и зарегистрироваться в системе BOINC. Проще всего сделать это через сам менеджер

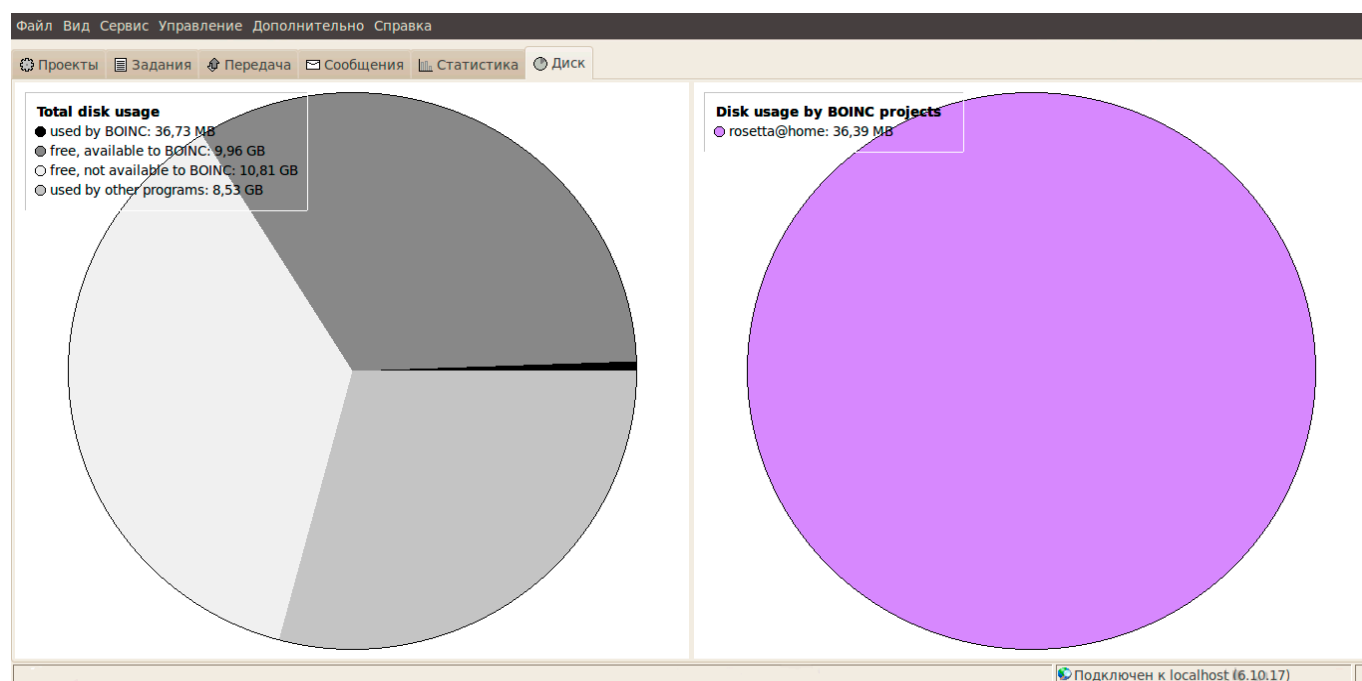


Рисунок 1. BOINC-менеджер, расширенный вид

(в нём можно найти все доступные проекты, зарегистрироваться в любом из них и сразу начать работу).

Другой способ – выбрать на официальном сайте BOINC (<http://boinc.berkeley.edu/projects.php>) понравившийся проект, перейти на страницу проекта и зарегистрироваться там. В случае регистрации через сайт, всё равно придётся подключаться к проекту, указав свои данные, через менеджер. Работу клиента можно настроить: указать сколько процессоров (в случае многопроцессорной системы) может использовать BOINC, при какой максимальной загрузке процессора менеджеру следует перейти в режим ожидания, настройки подключения к интернету. После старта работы над проектом в менеджере можно увидеть текущий прогресс по заданиям, проектам, статистику по работе за определённый период. Следует отметить, что потребление интернет-трафика крайне мало и вряд ли будет обузой для пользователей.

BOINC менеджер имеет два режима отображения: стандартный и расширенный. На мой взгляд, расширенный гораздо удобнее: в нём всё выложено на отдельные вкладки, структурировано. На **рис. 1**, например, показана вкладка со статистикой использования дискового пространства.

На странице «Задания» также можно найти кнопку «Показать графику», которая запускает новое окно с графическим отображением процесса вычислений. Например, на **рис. 2** показано такое окно для проекта SETI@home: в нём можно найти всю основную информацию о расчёте, но основное предназначение, как мне кажется, заключается во внесении разнообразия в процесс расчёта и хоть какая-то возможность «увидеть, что же делается». После запуска расчёта менеджер может работать без отображения окна

программы – в качестве системного процесса (демона); для открытия окна программы нужно воспользоваться главным меню – менеджер BOINC находится в разделе «Системные утилиты».

За выполненные расчёты пользователю начисляются баллы, которые оценивают объем проведенной работы. Для каждого участника проекта ведётся подробная статистика получения баллов. Также пользователь может присоединиться к команде (например, команда, представляющая страну/город/университет) или создать свою. На странице проекта можно найти рейтинг лучших пользователей/команд.

Естественно возникает вопрос достоверности получаемых результатов – ведь участник может фальсифицировать результаты или вообще прекратить участие в проекте. Эти вопросы решаются следующим образом: во-первых, для каждого выданного задания назначается крайняя дата предоставления результатов. Если к этому времени задание не будет выполнено, оно передаётся другому участнику. Вопрос проверки достоверности решается за счёт избыточных вычислений: одно и то же задание выдаётся пяти (по умолчанию, но может быть изменено руководителем проекта) участникам. Если результат ваших вычислений не совпадёт с результатом определённого числа пользователей с этим же заданием (по умолчанию – троих), то результат не будет учтён, а баллы – не начислены. Теоретически можно запустить и свой проект в рамках системы BOINC, но это достаточно непростой вопрос. Со структурой проекта можно ознакомиться здесь (<http://www.ibm.com/developerworks/ru/library/l-boinc/index.html#ibm-pcon>), а в официальной wiki описан процесс создания собственного проекта (<http://boinc.berkeley.edu/trac/wiki/ProjectMain#CreatingABOINCproject>).

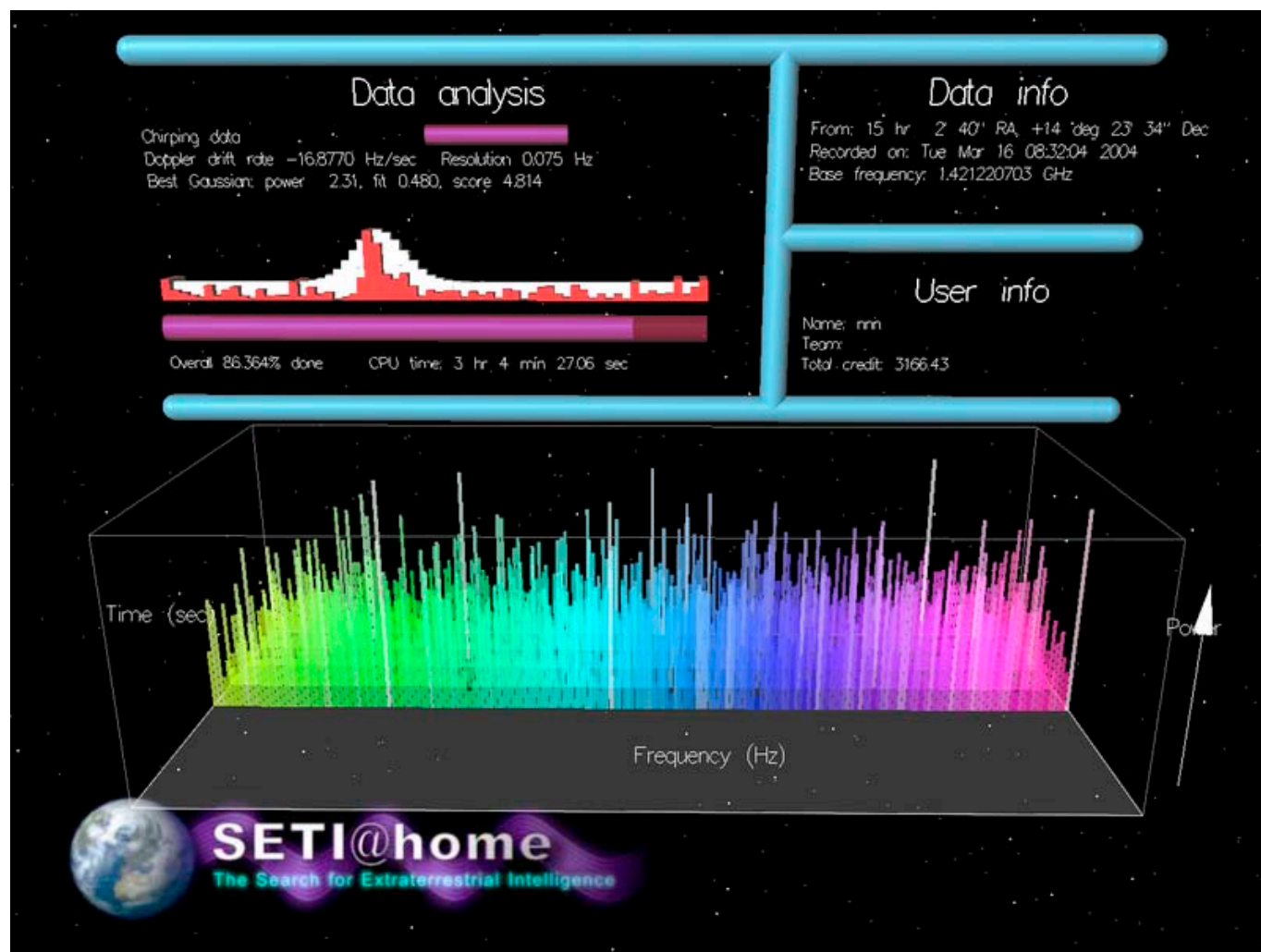


Рисунок 2. Графическое представление результатов

Если по какой-то причине предлагаемый клиент вас не устроит – можно воспользоваться альтернативным – KboincSpy (<http://kboincspy.sourceforge.net>), однако данный проект развивается достаточно вяло по сравнению с официальным клиентом.

Вместо заключения

BOINC – интересный проект, который на практике доказал возможность создания виртуального суперкомпьютера, состоящего

из множества обыкновенных домашних десктопов. С ним каждый может поделиться частичкой своих вычислительных возможностей и, может быть, помочь сделать значительное открытие в различных областях знаний.

Антон Деревянко
(anton.derevyanko@gmail.com)

Qt 4 в Haiku OS: почувствуй себя комфортно

Несмотря на то, что Haiku ещё находится на стадии альфа-тестирования, прикладных программ для этой операционной системы создано уже довольно много. Наследство, доставшееся от BeOS, можно найти на BeBits (<http://www.bebits.com>). Немало и приложений, созданных исключительно для Haiku, – они представлены на сайте Haikuware (<http://haikuware.com>). Там только в разделе Games на момент написания статьи насчитывалось 420 файлов, в том числе и такая классика жанра, как Heroes of Might and Magic II, портированная усилиями Michael de Oliveira. Тем не менее, пользователь Haiku всё ещё ощущает дефицит программ: нет родного офисного пакета, браузер WebPositive не слишком удобен, поскольку работа над ним далеко не завершена, не хватает удобных «читалок» и т.п.

Разработчику приложений для Haiku OS мешает незавершённость API операционной системы – в частности, продолжающаяся работа над классом панели инструментов (BToolBox) и отсутствие документации к некоторым классам (например, к менеджерам компоновки BCardLayout, BGridLayout и др., описания которых не найти в Be Book).

В этих условиях перенос на Haiku программного обеспечения, созданного с помощью такой популярной библиотеки классов, как Qt, в значительной степени устранил бы имеющийся дефицит полезных приложений ежедневного использования и способствовал бы популяризации молодой операционной системы.

С другой стороны, программист, получив в свои руки такой привычный инструмент разработки, как Qt, смог бы в значительной мере ускорить создание программ для Haiku. Собственно, это и является основной целью проекта Qt-Haiku (<http://www.qt-haiku.ru>), занимающегося портированием Qt в Haiku OS.

Где взять

Имеющиеся версии портов Qt можно найти по следующей ссылке: http://www.qt-haiku.ru/index.php?option=com_rokdownloads&view=folder&Itemid=60. На момент написания статьи были доступны для скачивания следующие версии Qt: 4.5.1 и 4.7.0. Каждая из них была представлена в двух вариантах: минимальном (библиотеки для запуска портированных Qt-приложений) и полном – собственно инструментом разработчика.

Поскольку Qt 4 в качестве компилятора требует g++ 4, для работы с библиотекой необходимо использовать gcc4 или гибридную сборку Haiku. Я рекомендую скачать версию Qt 4.7.0, поскольку 4.5.1

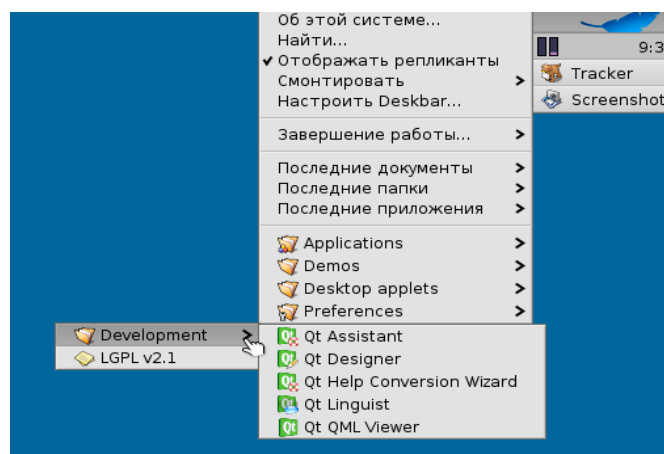


Рисунок 1. Программная среда Qt в меню DeskBar

в последнем релизе Haiku (R1/Alpha 2) некорректно показывает изображения.

Как установить

Установка Qt ничем не отличается от установки других приложений Haiku: необходимо распаковать архив ZIP в каталог /boot. После этого в меню DeskBar появятся пункты Qt – Development (см. рис. 1).

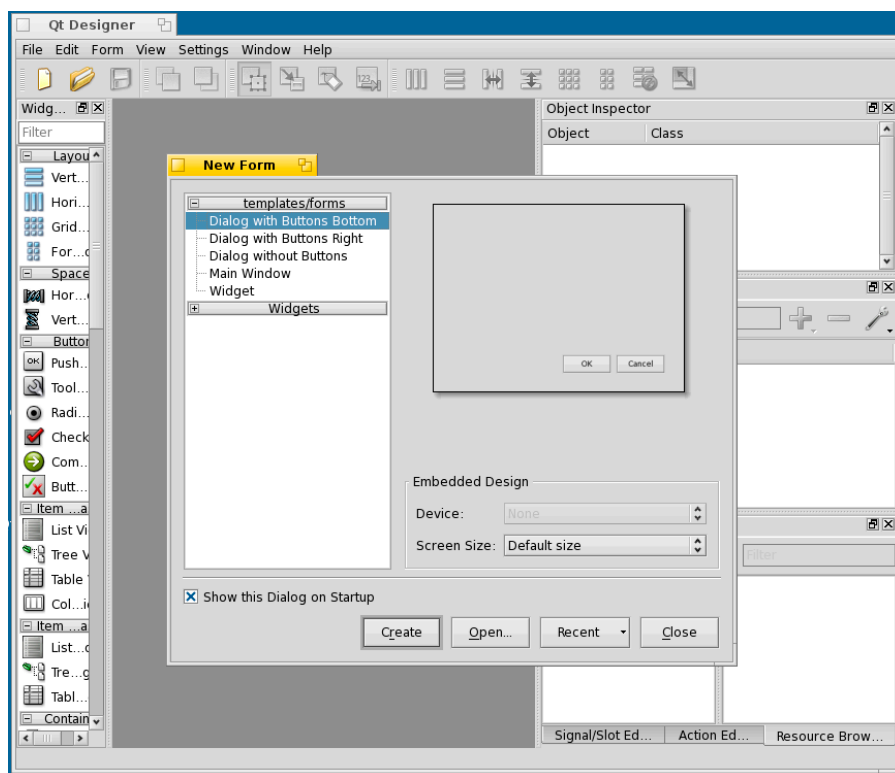


Рисунок 2. Выбираем программу по умолчанию для редактирования файлов ресурсов

Хотя в версии Qt для Haiku вы и не найдёте интегрированной среды разработки Qt Creator, все другие инструменты выглядят привычно (см. **рис. 2**).

Здесь же хочу выделить одну примечательную особенность: для того, чтобы установить Qt в Haiku, последнюю вовсе не обязательно устанавливать на жёсткий диск компьютера. Установить и работать с Qt в Haiku можно и в режиме LiveCD с последующим сохранением результатов компиляции на флеш-брелок. При этом Qt Designer, Assistant и Linguist будут запускаться не просто быстро, а очень быстро, не говоря уж про консольные утилиты вроде qmake. Разумеется, в случае LiveCD после перезагрузки компьютера библиотеку придётся устанавливать заново.

Как подготовить систему для работы с Qt?

Если вы собираетесь просто запускать программы, использующие Qt, то систему никак «готовить» не надо – после установки библиотеки всё и так готово. Вы можете загрузить понравившиеся Qt-программы по следующей ссылке: http://www.qt-haiku.ru/index.php?option=com_rokdownloads&view=folder&Itemid=61&id=7:qt-apps-for-haiku-os. Особенно рекомендую браузер

Arora (см. **рис. 3**) и просмотрщик электронных книг CoolReader (см. **рис. 4**).

Если же вы собрались программировать с использованием Qt и у вас так называемая гибридная сборка Haiku (а именно таковой является официально рекомендуемая для загрузки альфа-версия), то необходимо запустить терминал и ввести следующую команду:

```
setgcc gcc4
```

Иначе компиляция Qt-программ (с помощью gcc 2-й версии) приведёт к ошибке на этапе сборки (linking).

Как компилировать?

Компиляция осуществляется привычным для любого Qt-программиста способом:

```
qmake -project
qmake MyProgramme.pro
make
```

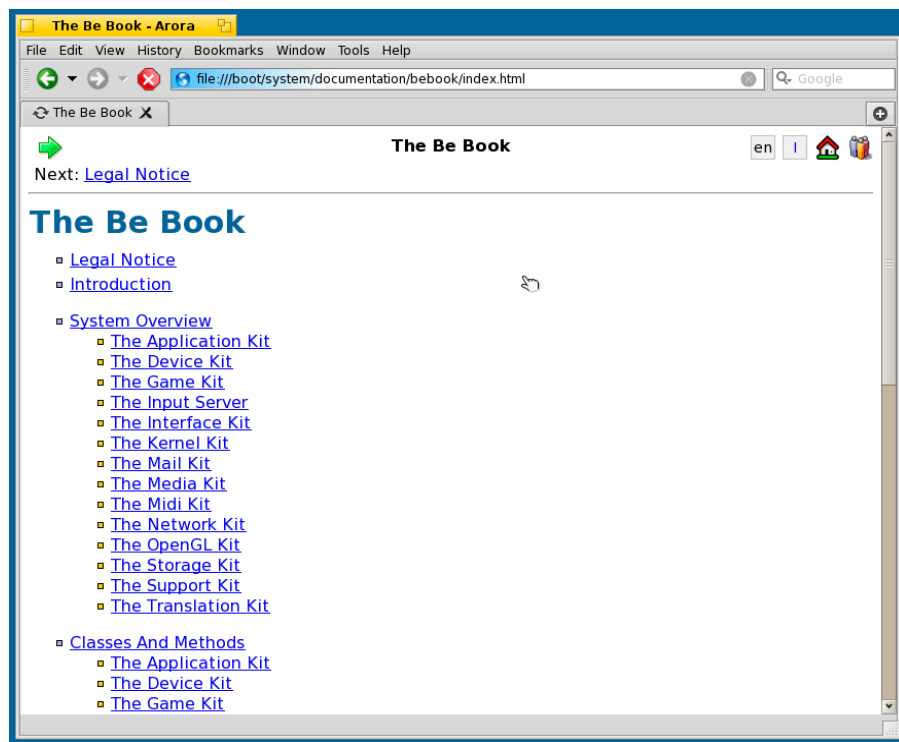


Рисунок 3. Просмотр Be Book в браузере Arora

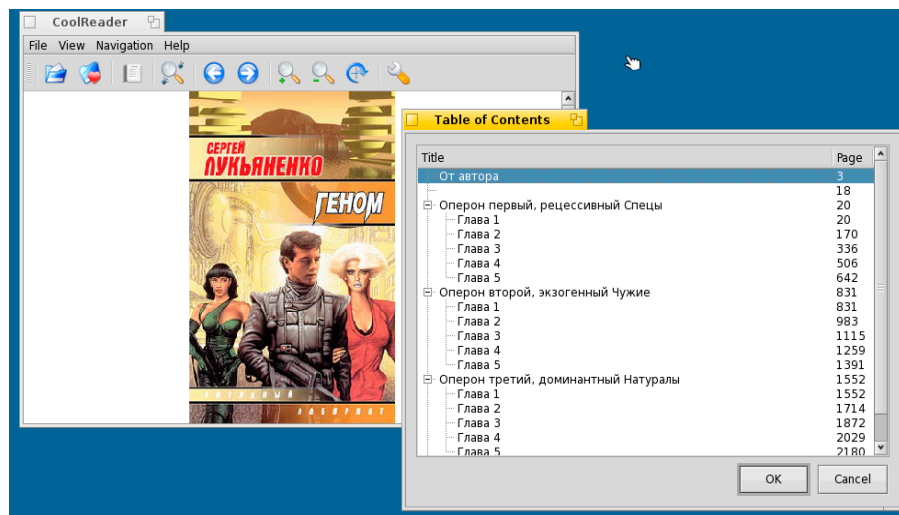


Рисунок 4. Чтение книги FB2 в CoolReader

Как правило, сборка программы происходит без проблем, однако в ряде случаев может потребоваться изменение исходного текста. Рассмотрим эти случаи.

Во-первых, использование директивы условной компиляции:

```
#if defined(Q_WS_X11)
```

приведёт к ошибке, поскольку, как мы помним, для построения графического интерфейса пользователя Haiku не использует X-сервер.

Во-вторых, следующий фрагмент кода (см. **листинг 1**) хоть и компилируется без ошибок, при запуске программы будет работать неправильно (см. **рис. 5**).

Листинг 1

```
QLocale locale;
QString sLocale = locale.system().name();
```

Поэтому в программах, использующих системную локаль для автоматического определения языка интерфейса, придётся устанавливать его вручную.

Поставка приложений Haiku, использующих библиотеку Qt

После того, как вы откомпилировали программу, необходимо создать инсталляционный пакет, который в Haiku представляет собой простой архив ZIP. Создайте рабочую директорию для вашей программы (например, build). В ней – дерево каталогов:

- ☒ build → apps → your_programme (где your_programme – каталог с названием, соответствующим имени вашей программы);
- ☒ build → home → config → be → Qt → Office (или Games для игр, Internet – для программ, работающих с сетью, и т.д.).

В каталог `your_programme` помещаем вспомогательные файлы и каталоги, необходимые для работы программы, а также её исполняемый файл.

Откроем терминал в нужном каталоге (Office, Games и т.п.) и наберём команду:

```
ln -s /boot/apps/your_programme/bin/your_programme
```

Так мы создадим символическую ссылку на программу в этом каталоге. Упакуем директории `apps` и `home` с помощью `ZipOMatic` и переименуем архив в `your_programme.zip`.

Теперь для установки программы на компьютер пользователя достаточно распаковать полученный архив в каталог `/boot`. Программа станет доступной в меню `DeskBar` в разделе `Qt` (см. рис. 6).

Несмотря на то, что набор программ на сайте проекта Qt-Haiku пока невелик, ряд портированных приложений KDE, включая `KOffice`, можно найти на сайте `TiltOS` (<http://tiltos.com/drupal>). Проект `TiltOS` был основан Grzegorz Dabrowski и позиционируется как операционная система на базе Haiku OS. По сути же он предоставляет репозиторий пакетов, т.е. скорее является дистрибутивом. Программы `TiltOS` распространяются не в виде ZIP-архивов, а в виде пакетов (*.pkg), созданных с помощью системы `Vox`. Последняя была разработана для Linux-проекта `Pingwinek` и позднее портирована на Haiku. Пакеты, созданные в `Vox`, бинарно совместимы с Haiku. Кстати, среда разработ-

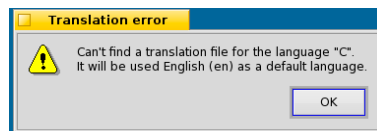


Рисунок 5. Ошибочное определение языка системы в Haiku OS

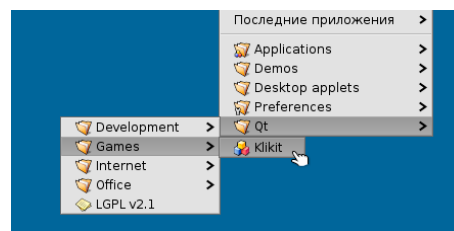


Рисунок 6. Ссылки на Qt-программы в меню DeskBar

ки `Paladin`, работа с которой была рассмотрена в предыдущих выпусках «Open Source» (№№064-66), устанавливается именно из пакета *.pkg.

Для установки репозитория `TiltOS` достаточно скачать соответствующий пакет `TiltOS for Haiku` и запустить установщик двойным щелчком по пакету. Предоставляемый `TiltOS` набор приложений вполне достаточен для комфортной работы с Haiku OS, которая, надеюсь, займёт достойное место на ваших компьютерах.

Андрей Кузнецов
(linmedsoft@gmail.com)

Схемотехника в Linux с помощью gEDA.

Часть 4: создание паттерна

Начало этой статьи читайте в выпуске «Open Source» 066 (от 18.08.2010), а начало цикла – в выпусках №№057-058. – Прим. ред.

Создание паттерна для PCB

Библиотеки компонентов для PCB делятся на два типа: `Oldlib` (или `M4 Lib`) и `Newlib`.

Первая использует для формирования описания компонента препроцессор `m4`, а у второй – для каждого компонента «фут-принт» хранится в отдельном файле с расширением `.fr`.

Для создания первого компонента нужны текстовый редактор и листок бумаги (лучше миллиметровки) с карандашом. Для создания второго – сам PCB и все (хотя можно и так же, т.е. при помощи миллиметровки и текстового редактора).

Рассматривать `Oldlib` я не буду, т.к. для новых компонентов рекомендуется использовать стиль `Newlib`. Желющие могут ознакомиться с этим методом здесь: <http://pcb.gpleda.org/pcb-20081128/pcb.html#Library-Creation> (на английском).

`Oldlib` удобен для создания типовых компонентов (или семейств) вроде `PLS`-разъемов с разным количеством выводов или элементов с разным расстоянием между выводами.

С другой стороны, при интерактивном просмотре через PCB мы не увидим все возможные варианты компонентов, созданные таким образом, и, соответственно, не сможем поместить их на холст (при генерации из `gschem` такой компонент все равно отобразится, но для ручного создания или корректирования это не очень удобно). В любом случае создание компонента – не совсем интуитивно-понятное действие.

Рассмотрим его поподробнее.

Классический способ создания паттерна

Начнем с классического способа создания паттерна. Для этого запустим PCB и сделаем активным слой `Component`. На этом слое надо поместить выводы: для выводных элементов используем инструмент `VIA`, для формирования выводов для `SMD`-элементов (для поверхностного монтажа) – `LINE`. К другим инструментам пока обращаться нельзя и не имеет смысла. Если нужна толстая площадка – рисуется линия нужной длины и толщины. После того, как создана линия, её концы будут скруглены – пока не обращаем на это внимание, но берем на заметку: максимальный размер вывода будет соответствовать прямоугольнику, в который полностью вписывается наша линия с учетом скругления концов. Вот так будет выглядеть заготовка корпуса для транзистора `IRF2804S-7P`, предназначенного для поверхностного монтажа и выполненного в корпусе `D2PAK-7` (см. рис. 1).

Длина ножек выводов – 3,2 мм, расстояние между ними – 1,27 мм, средняя пропускается, толщина ножки – 0,9 мм, толщина линии для стока – 10,8 мм, длина – 8,15 мм. Вообще размеры берутся из справочного листка или путем самостоятельных замеров. Теперь нужно каждому выводу назначить номер. Для этого наводим курсор мыши на вывод (важно не выделять его, а если выделение где-то есть – убрать его) и нажимаем <п>, вводим номер и закрываем окно (у нас номера 1, 2, 3, 5, 6, 7 и подложка – 8).

После этого, если необходимо нарисовать контуры объекта (например, для визуального позиционирования, определения габаритов и взаимного размещения компонентов на плате), выбирается слой `Silk`. На нем при помощи дуг и линий ри-

суются контуры объекта. Для нашего транзистора ничего особенного не получилось (точные размеры берутся из справочного листка, а я нарисовал примерно, т.к. габариты транзистора меньше, чем рекомендованные размеры для расположения выводов) (см. **рис. 2**).

Далее выделяем все элементы, ставим курсор по центру (позиция курсора на этом этапе определит точку привязки, у которой будет выводиться имя элемента на схеме, например: Q1, R1), нажимаем <Shift> и правую кнопку мыши, в контекстном меню выбираем Operation on selection → Convert selection to element. Выводы приобретут сероватый оттенок (если посмотрите внимательно на переключатель слоёв, найдете слой, которому соответствуют выводы).

Теперь нужно убрать скругление углов для контактных площадок. Для этого выделяем их, поочередно щелкая левой кнопкой мыши с зажатой клавишей <Shift>, и выбираем пункт меню Select → Change square-flag of selected objects → Pins. Наш компонент примет такой вид (см. **рис. 3**).

Остался последний штрих – сохранить компонент как файл .fp. Для этого он выделяется, вызывается контекстное меню (<Shift> + клик правой кнопкой мыши), выбирается пункт Operation on selection → Copy selection to buffer, нажимается <Esc> для выхода из режима вставки и сразу выбирается пункт меню Buffer → Save buffer elements to file. Вводим имя файла, указываем расширение .fp. Всё, на этом компонент закончен. Можно поместить его в каталог с библиотекой компонентов (системный: /usr/share/pcb/newlib) и пользоваться в PCB или указывать в качестве footprint в gschem.

Для редактирования существующего компонента шаги несколько отличаются (точнее, требуется предварительная подготовка). После запуска PCB редактируемый компонент добавляется через менеджер библиотеки компонентов (меню Windows → Library) на холст, далее компонент выделяется и через контекстное меню выполняется команда Operation on selection → Cut selection to buffer, после чего вызывается команда Buffer → Break buffer elements to pieces и затем Buffer → Paste buffer to layout.

После этих операций на холст будет помещен разбитый на части элемент – мы можем выполнить необходимые действия по редактированию, сделать преобразование в элемент и сохранить, как это описано выше.

Пример готового файла irf2804s-7p.fp можно найти в архиве http://osa.samag.ru/pub/pcb_geda.zip.

Создание корпуса из PDF

Данный способ полезен, когда в справочном листке есть описание корпуса и указаны размеры. В случае «подопытного» транзистора IRF2804S-7P такая информация есть на странице 9 (см. файл irf2804s-7p.pdf в архиве http://osa.samag.ru/pub/pcb_geda.zip).

Потребуется следующие инструменты:

- ☒ утилита pstoeedit с патчем <http://ftp.penguin.cz/pub/users/utx/pstoeedit/pstoeedit-3.45-pcb-enhanced.patch> (у меня в системе установлена версия pstoeedit 3.50, и я не стал ставить патч, при этом конвертация работала без проблем);
- ☒ редактор векторной графики, который понимает SVG и может создавать PostScript, а также умение работать с ним; я воспользовался Inkscape (ещё я попробовал XFig – результат тот же, только для него нужно будет поставить transfig для экспорта в различные форматы, в том числе и PostScript).

Все вышеперечисленные программы есть в стандартных репозиториях Arch Linux, так что для установки достаточно одной команды:

```
# pacman -S pstoeedit inkscape
```

или:

```
# pacman -S pstoeedit xfig transfig
```

Последовательность действий тоже оказывается достаточно простой. Для начала открывается справочный листок и смотрит-

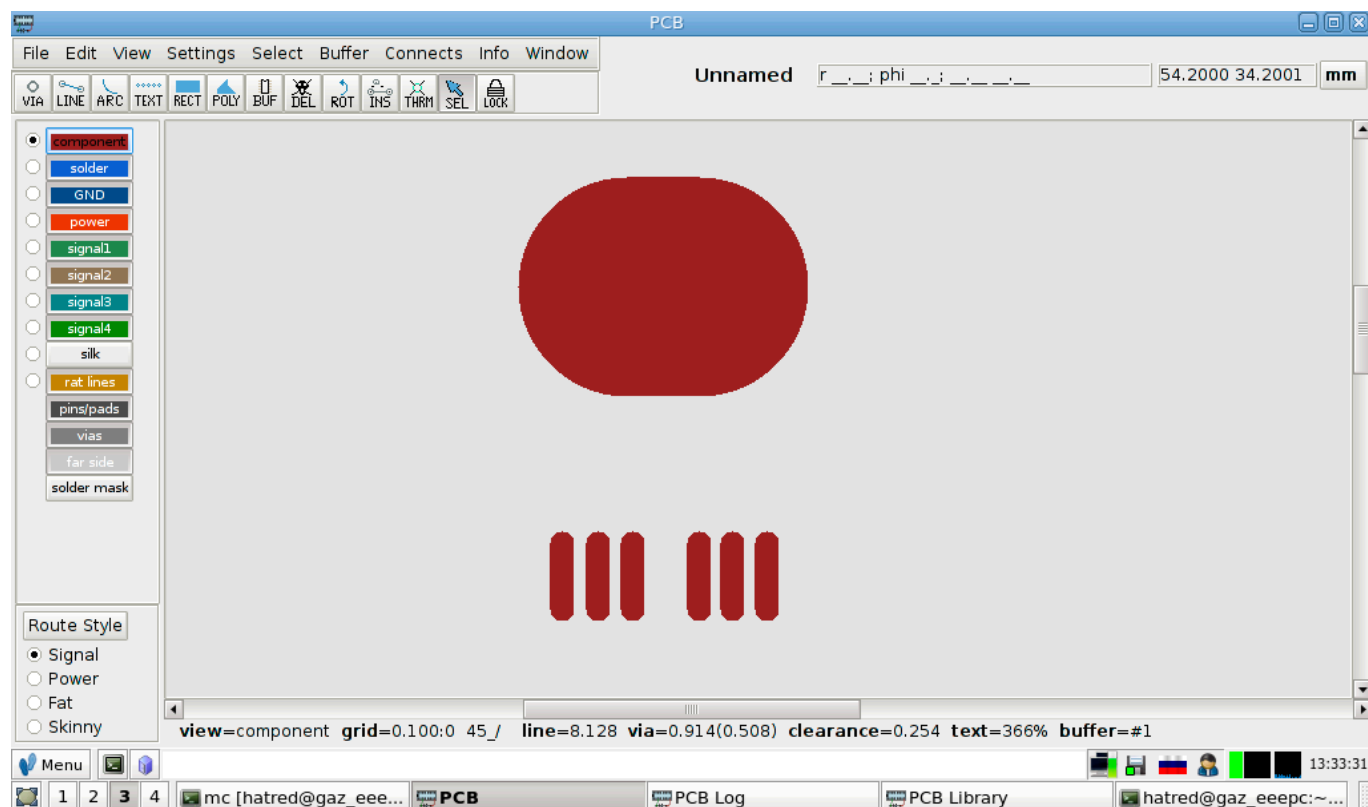


Рисунок 1. Заготовка корпуса для транзистора IRF2804S-7P, предназначенного для поверхностного монтажа и выполненного в корпусе D2PAK-7

ся, на какой странице располагается чертеж корпуса и выводов, после чего в терминале дается такая команда (все на примере транзистора IRF2804S-7P, страница 9):

```
$ pstoeedit -page 9 irf2804s-7p.pdf irf2804s-7p-new.svg
```

При использовании XFig:

```
$ pstoeedit -page 9 -f "xfig: -metric" irf2804s-7p.pdf \
  irf2804s-7p-new.fig
```

После этого открываем получившийся файл в векторном редакторе и удаляем все лишнее со страницы. Промежуточный вариант наших действий показан на **рис. 4**.

Удаляем внутреннюю штриховку и делаем заливку – я не сообразил, как её тут сделать (точнее, там достаточно хитро нарисованы контуры, или это издержки конвертации...), поэтому просто нарисовал залитые прямоугольники и в результате получил следующее (см. **рис. 5**).

Сохраняемся, затем делаем «Сохранить как...» и сохраняем в формате PostScript.

В результате получаем файл irf2804s-7p-new.ps, для которого выполняем следующую команду:

```
$ pstoeedit -xscale 0.555632727 -yscale 0.555632727 -f "pcb: \
  -stdnames -grid 0.1 -mm -snapdist 0.4" \
  irf2804s-7p.ps irf2804s-7p.pcb
```

По (пока?) непонятным мне причинам, при убранных опциях -xscale и -yscale размеры получаются в два раза больше, поэтому они задают масштаб 1:2. Флаг -stdnames – это аргумент конвертера, который задает стандартные имена для слоёв, -grid задает шаг сетки, -mm – использовать миллиметры вместо mils, -snapdist задает отображаемый шаг сетки.

Полученный файл загружаем в PCB, в моём случае он открылся так (см. **рис. 6**).

Теперь опять удаляем все лишнее и помним, что для подготовки элемента используются два слоя: Component для выво-

дов и Silk для комментариев, дополнительных графических построений. Если какой-то вывод находится не на слое Component (как у нас 1-7 на слое signal1 и 8 на слое solder), выделяем их, делаем целевой слой активным и выполняем команду Edit → Move selected to current layer. То же самое для габаритов и подписей, но помещая их на слой Silk.

Замечание: номера выводов подставляются автоматически, однако на всякий случай проверьте их вручную.

В результате всех этих махинаций получаем примерно такую картину (см. **рис. 7**).

Что делать дальше, мы уже знаем – это я описывал, когда рассказывал про создание компонента с нуля. Корпус, полученный по такому рецепту, доступен в архиве http://osa.samag.ru/pub/pcb_geda.zip (под названием irf2804s-7p-new.fp), промежуточные файлы – там же.

А если растр?

PDF – векторный формат, но даже в нем часть изображений может быть внедренными JPEG-файлами. Как в таких случаях быть: кусать локти, любясь на готовый footprint при невозможности с ним что-либо сделать? Отнюдь: здесь нам поможет векторизация растровых изображений. Я знаю как минимум два способа её сделать:

- ☒ Воспользоваться утилитой potrace (есть в Arch Linux: pacman -S potrace).
- ☒ Воспользоваться данным инструментом в Inkscape: «Контур → Векторизовать растр», предварительно вставив его в документ: «Файл → Импортировать». Кстати, инструмент в Inkscape основан на коде potrace, так что его можно считать графическим интерфейсом к ней.

В обоих случаях потребуется информация о DPI (разрешении) изображения, чтобы можно было точки перевести в реальные размеры. В случае с Inkscape вставленную картинку нужно растянуть/сжать с сохранением пропорций до её физических размеров (помогут направляющие, линейки), либо при преобра-

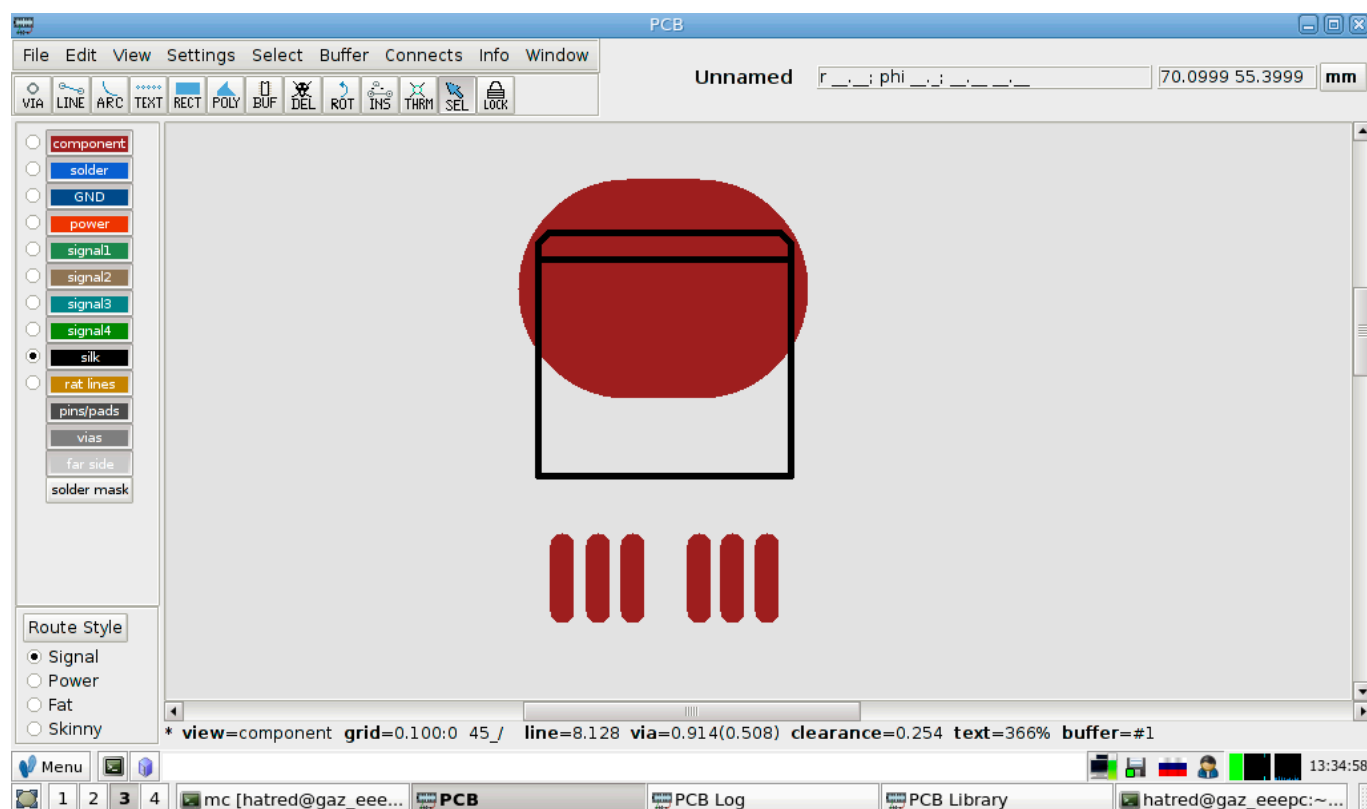


Рисунок 2. Рисуем контуры объекта (слой Silk)

зовании в PCB указать корректные -xscale, -yscale. Думаю, идея дальше ясна и подробно я останавливаться на этом не буду: попробуйте сами и напишите мне.

Дополнительные сведения

Файл описания паттерна (.fp) имеет простой текстовый формат. Теоретически его достаточно несложно сгенерировать при помощи скриптов. Подробности с его описанием можно найти в документе на английском языке: http://www.brorson.com/gEDA/land_patterns_20070818.pdf. Эту информацию можно использовать для тонкой ручной доводки нарисованного элемента (например, для вывода номер 1 в последнем поле директивы Pin[] написать 0x101 вместо 0x01 или строки, которая там есть).

Другие способы создания/получения символов и паттернов

Однажды мне попался сайт одного производителя, где были футпринты для его компонентов для PCB, но это скорее редкость. Более актуальным вариантом может оказаться конвертирование компонентов из других EDA. Для Eagle поможет следующая ссылка: <http://blog.mithis.net/archives/pcb/23-eagle2geda-symbol-converter> (прочитайте комментарий: в скрипте есть ошибка – нужно исправить одну букву, и все будет работать).

Существуют и личные коллекции компонентов. Например, <http://www.herveq.fr/linux/gedasym.php> или <http://www.luciani.org/geda/pcb/pcb-footprint-list.html>.

Если у вас есть сведения о каких-то других источниках условных обозначений и корпусов – буду рад сообщениям в почту или в форум: <http://osa.samag.ru/forum>.

Организация библиотеки компонентов

Итак, нужные символы и паттерны найдены или созданы. Теперь стоит вопрос организации библиотеки: ведь мы не будем каждый раз кидать их в системные директории, поскольку можем и не иметь прав администратора на системе (на работе, к примеру). Как же быть? Сложно ли сделать свою пользовательскую директорию с компонентами, и как её организовать? Нет, несложно – начнем с организации. Для начала определимся, в какой директории будут находиться наши обозначения, можно выбрать что-то вроде ~/gaf/ со следующими подкаталогами:

но – начнем с организации. Для начала определимся, в какой директории будут находиться наши обозначения, можно выбрать что-то вроде ~/gaf/ со следующими подкаталогами:

- ☒ **symbols** – для символов,
- ☒ **packages** – для паттернов (футпринтов).

В обеих директориях нужно сделать подкаталоги для сортировки по типам (транзисторы отдельно, микросхемы отдельно, контроллеры отдельно и так далее). Такую же структуру каталогов предлагаю использовать и локально внутри каждого проекта (дополнительно расположите там файл README, который будет описывать настройку окружения, чтобы эти компоненты подключились): это полезно, когда вы будете кому-то отправлять проект вместе с исходниками или получать от кого-то, находясь внутри команды.

Теперь о подключении библиотеки. Для этого нужно настроить три приложения:

- ☒ **из комплекта gEDA**: все через один файл ~/gEDA/gafrc;
- ☒ **gsch2pcb (хотя он тоже в комплекте gEDA)**: через файл конфигурации ~/gEDA/gsch2pcb;
- ☒ **PCB**: ожидал, что и у него придется править конфигурационный файл, но оказалось, что это единственный, у кого настройка библиотек делается через графический интерфейс.

Компоненты gEDA

Настраиваются через ~/gEDA/gafrc, в который нужно добавить код следующего содержания (на примере, как это сделано у меня):

```
; local project libraries
(component-library ".symbols" "Local project symbols")
//Добавляет в список путей поиска библиотек локальные для проекта
//символы (работает только в случае, если текущая рабочая директория
//и расположение файла с принципиальной схемой совпадают)

; define symbols prefix
(define user-sym-path "/home/hatred/gaf/symbols")
//Определяет префикс пути поиска символов (я же предлагал создавать
//подкаталоги для сортировки компонентов)
```

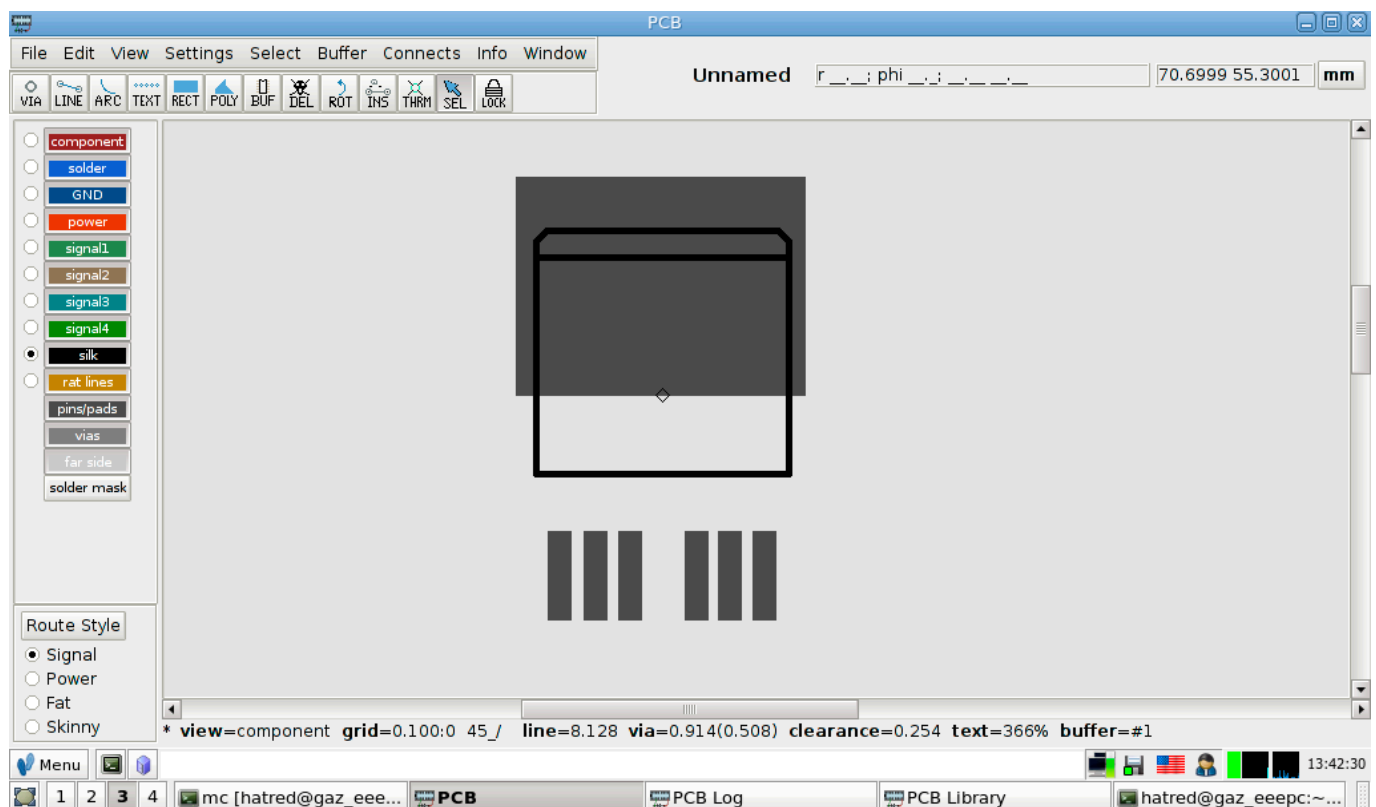


Рисунок 3. Убираем скругление углов для контактных площадок

```
(for-each
  (lambda (dir)
    (if (list? dir)
        (component-library (build-path user-sym-path (car dir)) 1
                           (cadr dir))
        (component-library (build-path user-sym-path dir))))
  )
(reverse '(
; Symbols subdirs
//Сначала поддиректория в /home/hatred/gaf/symbols, потом
//комментарий, который будет отображаться в менеджере библиотеки
("hatred-MOSFET" "MOSFET transistors with footprint")
("hatred-microschems" "User's Microschems symbols")
("connectors" "User's Connectors")
("sensors" "User's Sensors")
("crystal" "User's Crystals")
)))
```

```
(component-library "./symbols" "Local project symbols")
(component-library "/home/hatred/gaf/symbols/hatred-MOSFET" 1
                  "MOSFET transistors with footprint")
(component-library "/home/hatred/gaf/symbols/hatred-microschems" 1
                  "User's Microschems symbols")
(component-library "/home/hatred/gaf/symbols/connectors" 1
                  "User's Connectors")
(component-library "/home/hatred/gaf/symbols/sensors" 1
                  "User's Sensors")
(component-library "/home/hatred/gaf/symbols/crystal" 1
                  "User's Crystals")
```

Добавляем строки после комментария Symbols subdirs для каждой новой директории в /home/hatred/gaf/symbols. Данный конфигурационный файл можно представить и в более простом виде:

```
elements-dir ~/gaf/packages
```

Немного пояснений: синтаксис конфига – Lisp-подобный (кто настраивал Emacs, будут довольны).

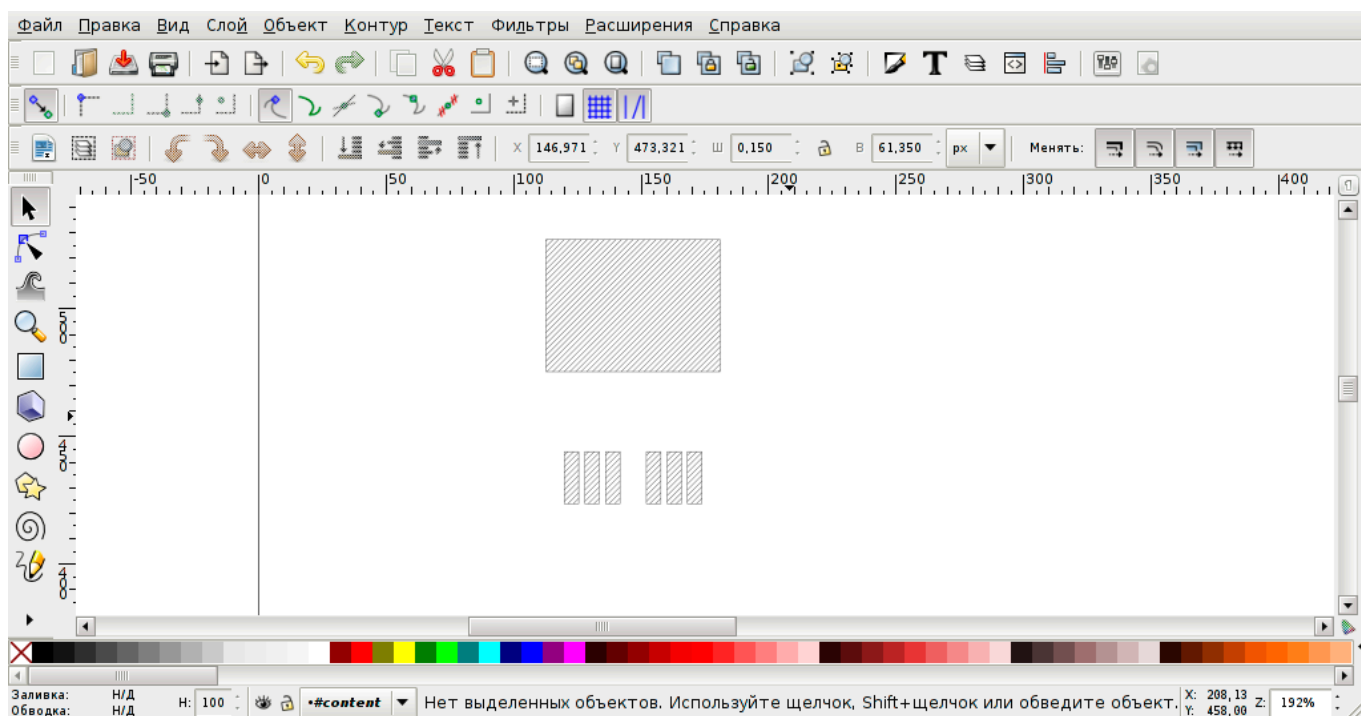


Рисунок 4. Промежуточный вариант после удаления всего лишнего со страницы

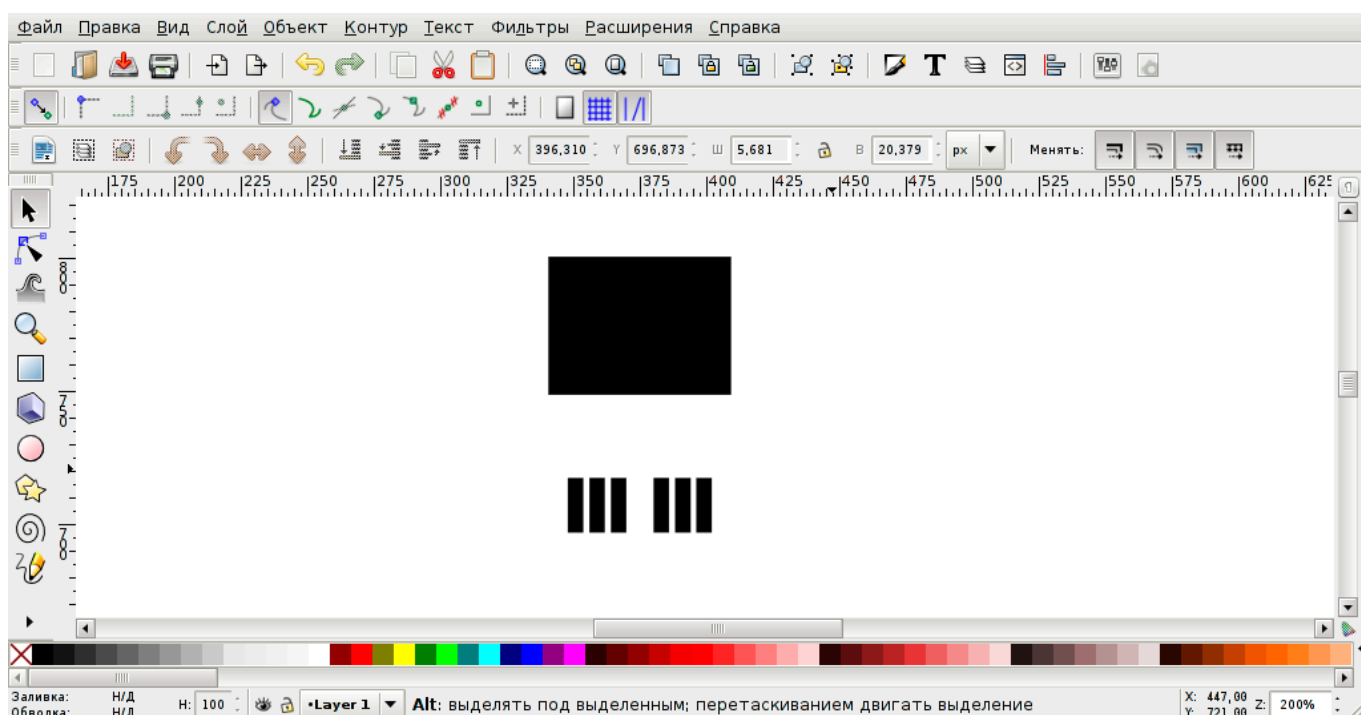


Рисунок 5. Удаляем внутреннюю штриховку и делаем заливку

Кому как удобно, так и используйте. Теоретически первый способ можно доработать для автоматической генерации списка библиотек.

gsch2pcb

Настраивается через файл `~/gEDA/gsch2pcb`, в который нужно (в нашем случае) добавить всего одну строчку:

```
elements-dir ~/gaf/packages
```

Далее он автоматически просмотрит все подкаталоги в поиске компонентов, что, согласитесь, очень удобно. Таких строчек может быть несколько. Директорию `./packages` в список вносить не нужно – она и так автоматически просматривается (если существует).

PCB

С PCB все тоже достаточно просто – закрываются все запущенные копии PCB, запускаем новую, следуем в `File → Preferences → Library`, а далее в строке ввода (через «:») вводим пути к библиотекам, которые тоже просматриваются рекурсивно. В нашем случае надо записать туда следующее значение:

После произведенных действий все символы отображаются в `gschem` и `PCB`, их видит `gsch2pcb` (и как следствие, работает `xgschm2pcb`). Добавление нового компонента требует перезапуска приложений, чтобы они его увидели. Перед отправкой скопируйте все символы и паттерны в каталоги `symbols` и `packages` соответственно внутри директории проекта, запакуйте и отправляйте. Кроме того, в `gschem` есть возможность импортировать символ прямо в документ. Для этого, когда добавляете компонент, в выпадающем списке надо выбрать «Внедрить компонент».

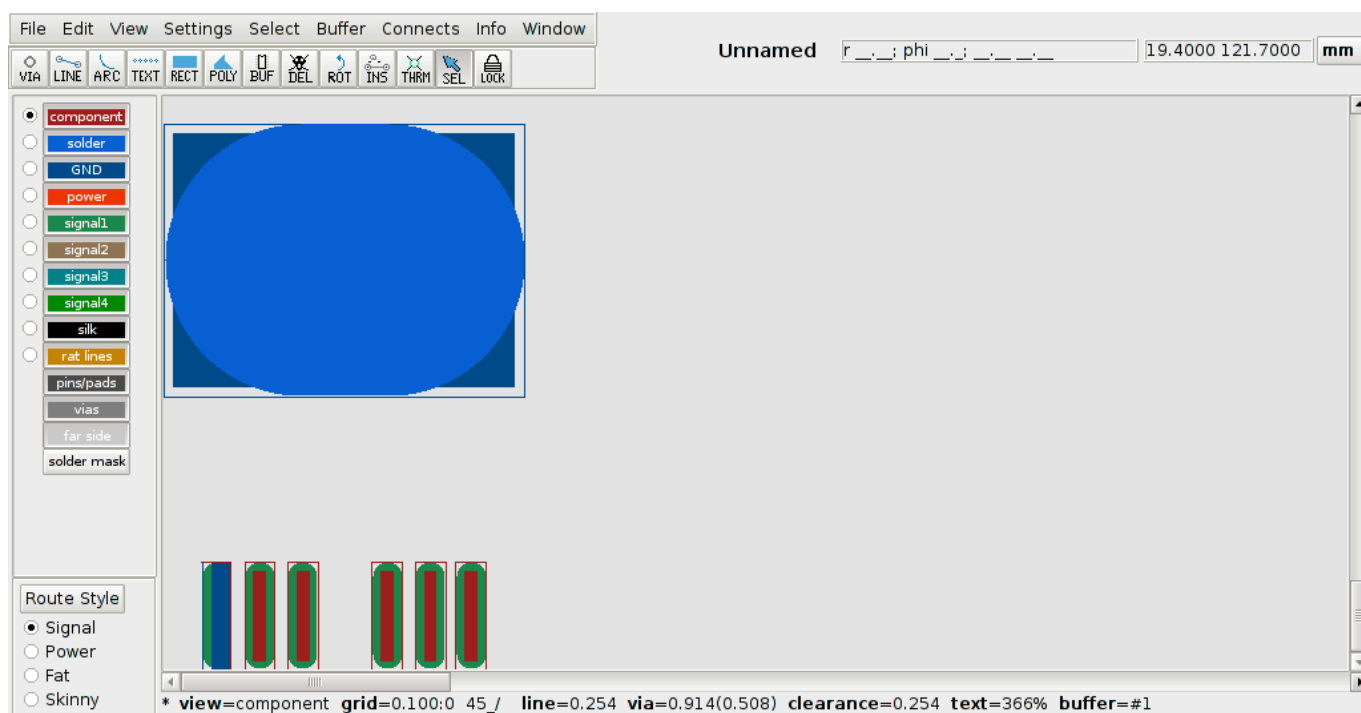


Рисунок 6. Файл, загруженный в PCB

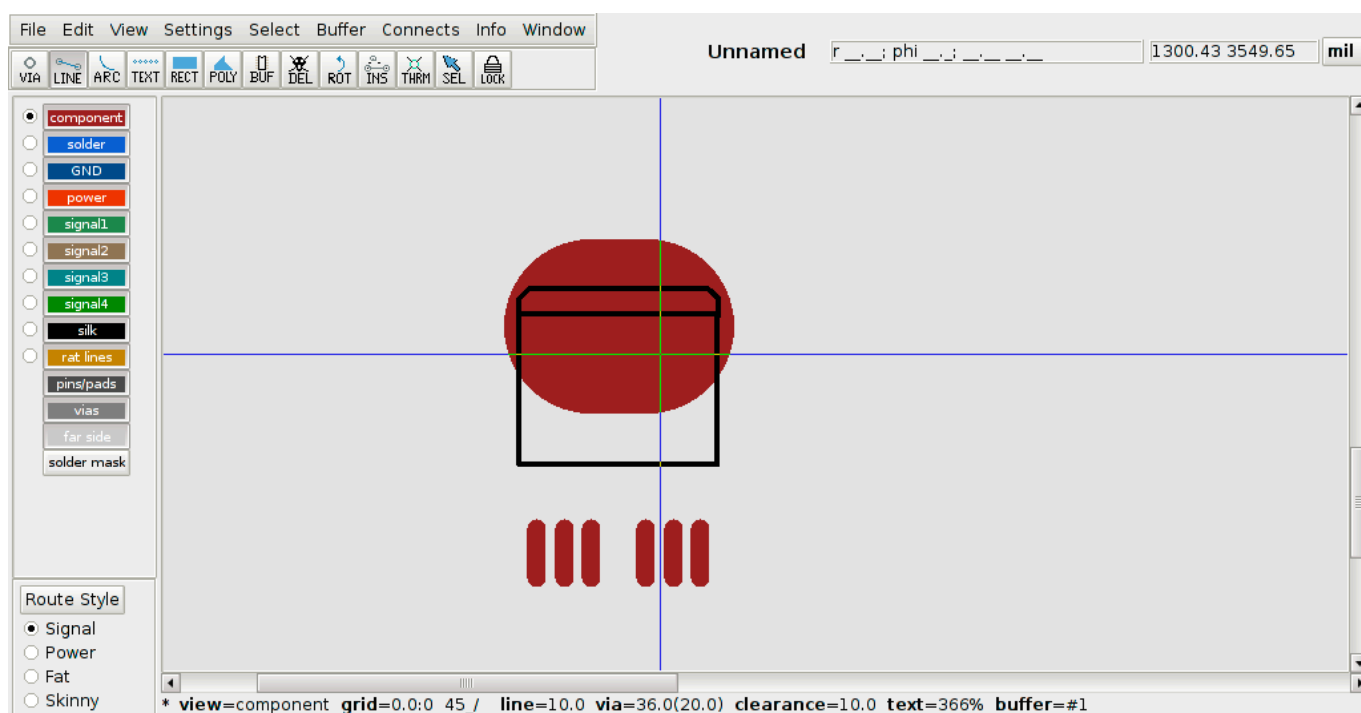


Рисунок 7. Вот что мы получили

в схему» – тогда можно обойтись без локального для проекта подкаталога symbols.

P.S. При подготовке материала были сделаны несколько ошибок. Некоторые специально, а некоторые были обнаружены позже, но я решил не исправлять их. Если найдете – пишите!

1. gEDA/gaf Symbol Creation Document – http://geda.seul.org/wiki/geda:gschem_symbol_creation.
2. gEDA/gaf Master Attribute Document – http://geda.seul.org/wiki/geda:master_attributes_list.
3. net= attribute mini-HOWTO – http://geda.seul.org/wiki/geda:na_howto.
4. Creating gschem symbols quickly and easily using tragesym and a spreadsheet programm – http://geda.seul.org/wiki/geda:tragesym_tutorial.
5. The official manual of pcb – <http://pcb.gpleda.org/manual.html>.
6. Manual on footprint creation – http://www.brorson.com/gEDA/land_patterns_20070818.pdf.
7. Getting Started With PCB – <http://www.delorie.com/pcb/docs/gs/gs.html>.
8. Man gsymcheck.
9. Gschem Symbol and PCB Element Transistor Guide – http://geda.seul.org/wiki/geda:transistor_guide.
10. PDF → PCB → footprint conversion – <http://www.penguin.cz/~utx/pstoedit-pcb>.
11. Eagle2geda Symbol Converter – <http://blog.mithis.net/archives/pcb/23-eagle2geda-symbol-converter>.

Александр Дроздов
(hatred@inbox.ru)

«Open Source» приглашает к сотрудничеству!

Электронное приложение «Open Source» всегда открыто для сотрудничества с новыми авторами, с читателями и их конструктивными предложениями по улучшению издания, обоснованной критикой и любыми отзывами, с компаниями, занимающимися разработкой и продвижением программного обеспечения с открытым кодом. Приветствуются все энтузиасты, желающие опубликовать у нас свои статьи. Тематика нужных материалов очевидна из предназначения приложения, то есть FOSS (Free and

Open Source Software): теория и практическое применение; исторические сведения, анализ сегодняшнего положения, прогнозы на будущее и другие аспекты, связанные с открытым ПО.

Среди наиболее интересных на данный момент общих тем можно выделить:

- ☑ общие обзоры новых и/или интересных проектов Open Source и конкретных приложений, свежих версий дистрибутивов Linux, *BSD и других систем;
- ☑ советы и рекомендации новичкам в GNU;

- ☑ истории успеха применения/распространения ПО с открытым кодом;
- ☑ философия и идеология Free Software;
- ☑ разработка приложений с применением средств Open Source.

Желательный объем статей: 6000 или 12000 символов (с пробелами). Примеры актуальных сейчас тем для статей публикуются на <http://osa.samag.ru/todo>. Но не стоит строго ограничиваться приведенными выше рамками!

Публичное обсуждение «Open Source» проводится на форуме сайта журнала «Системный администратор» по адресу: <http://osa.samag.ru/forum>. Связаться с редакцией можно по электронной почте osa@samag.ru.

P.S. За статьи мы платим.

Подписные индексы:

20780*

+ диск с архивом статей

81655**

без диска

по каталогу агентства
«Роспечать»**88099***

+ диск с архивом статей

по каталогу агентства
«Пресса России»

* Годовой
 ** Полугодовой
 *** Диск вкладывается
 в февральский
 номер журнала,
 распространяется только
 на территории России

Подписка на журнал «Системный администратор»

Российская Федерация

- ✓ Подписной индекс: годовой – **20780**,
полугодовой – **81655**
Каталог агентства «Роспечать»
- ✓ Подписной индекс: годовой – **88099**,
полугодовой – **87836**
Объединенный каталог «Пресса
России»
Адресный каталог «Подписка за ра-
бочим столом»
Адресный каталог «Библиотечный
каталог»
- ✓ Альтернативные подписные агентства:
агентство «Интер-Почта»
(495) 500-00-60, курьерская доставка
по Москве
агентство «Вся Пресса»
(495) 787-34-47
агентство «Курьер-Пресссервис»
агентство «ООО Урал-Пресс»
(343) 375-62-74
- ✓ Подписка On-line
<http://www.arzi.ru>
<http://www.gazety.ru>
<http://www.presscafe.ru>

СНГ

В странах СНГ подписка принимается
в почтовых отделениях по националь-
ным каталогам или по списку номенкла-
туры «АРЗИ»:

- ✓ **Азербайджан** – по объединенному
каталогу российских изданий через
предприятие по распространению пе-
чати «Гасид» (370102, г. Баку, ул. Джа-
вадхана, 21)

- ✓ **Казахстан** – по каталогу «Россий-
ская пресса» через ОАО «Казпочта»
и ЗАО «Евразия пресс»
- ✓ **Беларусь** – по каталогу изданий стран
СНГ через РГО «Белпочта» (220050,
г. Минск, пр-т Ф. Скорины, 10)
- ✓ **Узбекистан** – по каталогу «Davriy
nashrlar», российские издания через
агентство по распространению печати
«Davriy nashrlar» (7000029, г. Ташкент,
пл. Мустакиллик, 5/3, офис 33)
- ✓ **Армения** – по списку номенклатуры
«АРЗИ» через ГЗАО «Армпечать»
(375005, г. Ереван, пл. Сасунци Давида,
д. 2) и ЗАО «Контакт-Мамул» (375002,
г. Ереван, ул. Сарьяна, 22)
- ✓ **Грузия** – по списку номенклату-
ры «АРЗИ» через АО «Сакпресса»
(380019, г. Тбилиси, ул. Хошараульская,
29) и АО «Мацне» (380060, г. Тбилиси,
пр-т Гамсахурдия, 42)
- ✓ **Молдавия** – по каталогу через ГП «По-
шта Молдовей» (МД-2012, г. Кишинев,
бул. Штефан чел Маре, 134)
по списку через ГУП «Почта Придне-
стровья» (МД-3300, г. Тирасполь, ул.
Ленина, 17)
по прайс-листу через ООО агентство
«Editil Periodice» (МД-2012, г. Киши-
нев, бул. Штефан чел Маре, 134)
- ✓ Подписка для **Украины**:
Киевский главпочтамт
Подписное агентство «KSS»
Телефон/факс (044)464-0220