BSD, Linux и компонентность

Роман Химов, Четверг, 15 Сентябрь 2005, 13:07

BSD против GNU/Linux

Процесс неспешного заполнения КаталогОСа историей развития славной ветви BSD UNIX наводит на странный вопрос - почему вокруг есть столько GNU/Linux, но так мало BSD?

Один из первых ответов, напрашивающийся сам по себе - разные лицензии. Да, конечно, GPL гарантирует пожизненную свободу кода, что по мнению многих разработчиков справедливее, чем BSD, когда написанный тобой код может взять и продать как часть своего продукта любая компания (доказано Microsoft) - это обидно. Но за этим все-таки есть еще один момент, если сами разработчики свободных систем в большинстве все-таки предпочитают лицензию GPL, а не BSD, то компании, которые хотят работать со свободным кодом уж точно больше заинтересованы именно в BSD (она дает им больше возможностей). Сложно сказать, насколько эти силы уравновешивают друг друга, но на мой взгляд в достаточной степени, чтобы убрать из рассмотрения вопрос лицензии.

Хорошо, давайте еще немного вспомним расхожие мнения о том, что BSD системы сами по себе корректнее реализуют различные стандарты (пример с TCP/IP стеком, пожалуй, наиболее убедителен - эту часть BSD еще не примерял на свою ОС или продукт только ленивый), имеют значительно большую историю развития и, в целом, более стабильны, нежели GNU/Linux. Не будем обсуждать корректность этих мнений, но нельзя не признать, что это есть.

И все-таки вокруг я вижу сотни дистрибутивов GNU/Linux (из которых активно занимаются разработками десятки) и лишь десятки *BSD (из которых реально активны единицы). В чем причина?

Давайте посмотрим на аргументацию типичных BSD-шников о том, почему их системы лучше, чем GNU/Linux. Далеко не на последнем месте вы услышите то, что BSD - это цельная система, которая собирается и разрабатывается в едином дереве исходников. Это применимо ко всем наиболее популярным сегодня ветвям BSD - FreeBSD, OpenBSD и NetBSD.

Любители OpenBSD сразу же пояснят, что именно это позволяет сохранять высочайший общий уровень безопасности этой системы - найденные типы ошибок разработчики OpenBSD исправляют действительно во всем дереве исходников. Аналогично происходит с изменениями любых библиотек и API - эти системы практически никак не связаны по рукам совместимостью и могут менять все что угодно в пределах системы, были бы изменения действительно лучше старых подходов.

В то же время, само название GNU/Linux сразу говорит о компонентности и разделении обязанностей. Действительно, Линус Торвальдс и его команда делают отличное ядро, разработчики GNU в разных проектах работают над цельными компонентами системы - GCC, GLibc, Coreutils и многие другие. Причем, это развитие действительно раздельное, хотя, конечно, существуют некие точки синхронизации работы, но, например, последняя стабильная вресия GLibc умышленно создавалась для компиляторов GCC из ветки 3.x и не

поддерживает компиляторы ветки 4.x.

Вроде бы непорядок - как так, разработчики Glibc не поспевают за развитием GCC?! Однако нет, на самом деле, они ровно и спокойно работают над своим компонентом системы. Аналогично работают и множество других проектов, составляющих цельную GNU/Linux систему. А дистрибьюторы упаковывают это все согласно своим соображениям в различные дистрибутивы, среди которых мы потом ломаем голову проблемой выбора.

То есть, изначально, GNU/Linux - система компонентная, а BSD - монолитная. Несмотря даже на то, что последней никто не мешает быть компонентной (все-таки, она не так уж плохо делится на части), никто этого даже не пытается делать, это такая система.

Linux против компонентности

В то же время, несмотря на осознание действенности давным-давно уже существовавших идей компонентности, даже сегодня все в том же стане Linux не все так гладко. В стандартной поставке ядра Linux есть занимательный документ, находящийся в "<tt>Documentation/stable_api_nonsense.txt</tt>". В этом документе линуксоиды последовательнопоясняют, почему в ядре нет и не может быть стабильного API. Аргументация, в общем-то, проста и удивительно схожа с тем, что мы слышим из стана BSD относительно всей системы - это позволяет делать все правильно, причем, во всей системе.

Там же Грег Кроа-Хартман приводит пример с интерфейсами USB, которые в ядре переписывались трижды. Опять-таки, примеры понятные, ведь действительно изменения позволяли избавиться от многих проблем и были необходимы, а имея под контролем все дерево исходников Linux можно было вносить эти изменения единовременно для всех поддерживаемых драйверов - вуаля!

С одной стороны все это нельзя не понять, все-таки ядро можно считать законченным компонентом системы и на том успокоиться. Однако, есть масса сторонних модулей ядра, которые от версии к версии просто вынуждены обрастать конструкциями "#ifdef ... #endif" в исходниках, которые обходят различные особенности различных версий ядра. Плюс к этому, разработчикам приходится постоянно выпускать все новые и новые версии модулей, посмотрите хотя бы на драйвера карт nVidia (закроем сейчас глаза на их проприетарную сущность).

Сами разработчики Linux на этот счет обычно говорят очень кратко и просто - не хотите иметь проблем с изменениями API ядра, значит, интегрируйте ваш модуль в официальное ядро. Это работает, но требует очень значительных усилий - требования к коду предъявляются очень высокие. Понятно, что от этого модуль становится только лучше, но это останавливает многих на пути к той самой интеграции, некоторых останавливает на пути рассмотрения GNU/Linux как целевой платформы для какого-либо железа, а это уже потери пользователей.

Сегодня Linux разрабатывается распределенно и покомпонентно, но все-таки затем собирается воедино под личным руководством Линуса Торвальдса и Эндрю Мортона - не очень-то легкая задача для такого объема кода, надо заметить. При этом, несмотря ни на что, качество последних релизов отнюдь не так высоко, как хотелось бы.

Мне кажется, сегодня стоит признать тот факт, что Linux стал слишком объемен и сложен, чтобы быть цельным и единым компонентом GNU/Linux. Нет, я не призываю к микроядерности, все что я хочу сказать, это то, что Linux пора прийти к реальной независимой компонентости, поскольку иначе все больше и больше сил разработчиков будет тратиться на обеспечение цельности Linux, а не на его развитие.

Кстати говоря, один из наследников BSD, MacOS X, именно к этому и пришла в последних версиях своих ядер - они внедрили механизмы, позволяющие новым ядрам работать со старыми модулями, а все изменения API теперь проводятся постепенно. Такой подход дает сторонним разработчикам время на "" изменений в API и заметно облегчает им жизнь.

Более того, еще один компонент GNU/Linux - сервер X.org также в седьмой версии переходит на полностью модульное существование. По иронии, объем кода X.org и Linux очень схож, да и суть кода во многом пересекается - некий стержень и масса модулей-драйверов вокруг него, грубое приближение, но работает. Так вот, X переходит на полную модульность и я уверен, что после того, как утрясутся все связанные с этим проблемы, мы увидим такие иксы, какими мы их никогда не видели - сторонние разработчики должны подхватить идею.

Приживется ли такой подход в Linux мне неизвестно, более того, с учетом того, что упомянутый мной документ появился в документации не так давно, в ближайшем будущем на это надеяться не приходится. Но я все-таки верю, что именно в этом будущее Linux как ядра, правильная компонентизация, кажется, еще никому не вредила.