Алексей Федорчук, "Почему DragonFly?"

Содержание

Хардверная ретроспектива

Несколько лет назад я писал, что мы живем в интересное время: впервые со времен появления персональных компьютеров софверная индустрия не смогла выполнить свое сакральное предназначение - вытрясать пользовательские кошельки на перманентный апгрейд своих машин, что обеспечивало бы финансовые источники для дальнейшего развития аппаратных средств (последняя версия). И ныне мы видим плоды этого: развитие главного компьютерного "железа" - процессоров и чипсетов, то есть того, что определяет лицо платформы - фактически прекратилось.

Конечно, время от времени производители бодро рапортуют об очередном повышении тактовой частоты "камней", увеличении кэша level n до совершенно невообразимых, некогда вполне достаточных для основной памяти, объемов, а то и добавлении очередного level'a, дополнительных наборов инструкций по замысловатыми названиями, встраивания в чипсеты поддержки всего, чего можно (и даже того, чего, недавно, казалось - нельзя, типа полноценной 3D-графики). Однако накал страстей вокруг всего этого не тот, несколько напоминая бег на стометровку - мало кого (кроме фанатов этого вида спорта) волнует, какую десятую секунды отыграл один суперспринтер у другого.

А главное, что, подобно рекордам в стометровке, достижения "камнестроителей" все меньше волнуют широкие пользовательские массы. Ведь от медведя не убежит и олимпийский чемпион, а успеть за пивом в магазин перед его закрытием способен любой мало-мальски тренированный человек. Так и с компьютерами: настольно-пользовательские задачи в большинстве случаев решаются средствами любого подручного "железа", купленного за пределами лавки древностей. А "тяжелые" задачи как требовали часов (а то и суток) пахоты персоналок, так и требуют (и потому решаются разумными людьми обычно не на PC'шках).

Фактически за последние годы производители предложили нам два архитектурных решения, претендующих на революционность. Первым (по времени) была архитектура Pentium 4. Не будем распространяться здесь о ее достоинствах или недостатках как таковой. Для нас сейчас важно только то, что она обеспечила возможность роста тактовой частоты процессоров, поражающего воображение (пользователя, тянущегося к бумажнику). И к тому же казавшейся безграничной...

Правда, на счет безграничности жизнь довольно скоро внесла свои коррективы. И разговоры о том, что эта технология позволит играючи достичь тактовой частоты в 10 гигагерц, как-то стихли сами собой - нынче даже о четырех гигагерцах почему-то не любят говорить вслух.

Впрочем, по сему поводу "минутной печали не стоит, друзья, предаваться - Ведь быстрым машинам нет смысла у нас появляться". Ибо чисто случайно оказалось, что гигагерцев хватало на всех (кроме тех, которым, как уже говорилось, их не будет хватать всегда). И разница между 2-х и 4-гигагерцной машиной в быстродействии была бы, думаю, меньше, чем между P-133 и P-166...

Вторая, казалось бы, революционная, новинка - 64-разрядные вычисления. Вспомним, каким прорывом в светлое будущее были 32-битные процессоры для PC, те самые первые "трешки", которые сделали возможным портирование на эту архитектуру Unix и его порождений - свободных POSIX-совместимых операционок. Повторится ли история ныне?

Ответ можно дать уже сейчас - и ответ, увы, отрицательный. Потому что в те далекие уже годы аппаратура PC едва поспевала за софтом - 32-битные ОС разменивали уже второй десяток лет своего существования, и приложений, использующих 32 разряда на полную катушку, было уже вдоволь. Нынче же - да вы и сами знаете, как нынче обстоит дело с переходом на 64 бита "внутре" ОС, а уж когда дело дойдет до пользовательских программ - и пророк Заратустра предсказать не возьмется.

Потому что на самом деле 64-разрядность в пользовательском сегменте не востребована. Ведь что она дает по большому счету? Увеличение объема максимально адресуемой памяти? Да, прекрасно, но есть ли у пользователя 64 Гбайт ОЗУ и терабайты дискового пространства? И главное, нужны ли они ему в принципе? И понадобятся ли в обозримом будущем. Так что единственная ниша пользовательских приложений, где 64 бита хоть как-то заиграют - параноидальная криптография. Конечно, в свете недавних событий нас попробуют убедить, что без последней в современном мире не выжить каждой домохозяйке, однако и здравый смысл пока еще не совсем отменили...

Так что усилия "камнестроителей" пропали бы втуне. Если бы еще не одно новшество, о котором я сознательно не упоминал ранее - Hyper Threading, то есть виртуальная мультипроцессорность. Каковая в некоторых (правда, весьма редких) задачах давала вполне даже реальный прирост производительности. Опять же, мало значащий для пользователей самой распространенной операционки, работающих преимущественно интерактивно. Но весьма способствующий производительности труда брата-POSIX'ивиста (вследствие врожденной лености отдающего предпочтение всякого рода скриптам, пакетным заданиям и прочим средствам автоматизации).

Однако Hyper Threading - не более, чем суррогат истинной мультипроцессорности (своего рода мультипроцессорность для бедных, но гордых). И, сказавши **А**, нужно неизбежно открыть рот для произнесения **Б**. То есть переходить к собственно двухпроцессорным конфигурациям в пользовательском сегменте.

Разговоры о двухпроцессорных пользовательских десктопах на нашем веку возникают не впервые. Помните, как дешевенькие Celeron'ы первого призыва можно было вставлять в относительно не очень дорогие двухпроцессорные "мамы", получая таким образом нечто вроде "народного суперкомпьютера"? А еще масла в это пламя подлили Opteron'ы, также в начальном своем уровне вполне доступные для широких масс. Правда, первая линия "народной мультипроцессорности" была очень быстро пресечена производителем. А во втором случае дешевизна базовой платформы с лихвой компенсировалась дороговизной требуемой для нее регистровой памяти. Да и в любом случае цена двухпроцессорного решения будет более чем вдвое выше однопроцессорного - лишний сокет тоже денег стоит:-).

Однако идея мультипроцессорности для народа витала в воздухе - никаким иным способом создать впечатление прогресса уже не удается. И первый шаг в этом направлении сделала, насколько мне известно, IBM со своими Power4 - вероятно, абсолютными рекордсменами по "чистому" (=тестовому) быстродействию. В том числе и благодаря тому, что имели варианты с двумя и более процессорными ядрами в едином корпусе.

Сами по себе Power4 (как и идущие им на смену Power5) ориентировались на индустриальный сектор. Однако на базе их были созданы процессоры G5 - сердце современных Mac'ов, имеющие, в том числе, и двухъядерный вариант. У которого есть все шансы со временем широко распространиться по столам пользователей платформы от Apple.

Правда, пользователям PC'шек от этого было бы ни холодно, ни жарко. Однако и их "камнестроители" не заставили себя ждать: по слухам, инженерные семплы двухъядерных процессоров AMD64 уже имеют место быть в природе. Аналогичных "камней" от Intel, вроде бы, еще никто не видел. Однако недавно "случайно" просочилась информация о их грядущей стоимости. Вполне, нужно сказать, привлекательной - менее трехсот ихних денег за более чем 3-гигагерцную модель "в одном стакане". Что, согласитесь, уже вполне вписывается в рамки "компьютера для народа" (хотя и не для всякого его представителя).

Так что вполне возможно, что уже при следующем апгрейде перед нами встанет выбор - приобретение машины с традиционным одноядерным процессором или, за чуть большое деньги - машины с процессором двухъядерным. Готовы ли мы к этому? То есть - способны ли получить от последней хоть какой-то выигрыш в производительности, компенсирующий затраты?

Софтверная перспектива

Исходя из самых общих соображений очевидно, что ожидать двукратного увеличения быстродействия от самого факта удвоения числа процессоров (вне зависимости, каждый ли в своем сокете, или "в одном стакане") не приходится. Во-первых, на "железном" уровне два и более процессоров будут совместно использовать какие-то общие ресурсы компьютера - память, кэши, шины (в зависимости от конкретной аппаратной реализации).

Во-вторых, неизбежны потери быстродействия за счет "накладных расходов" - синхронизации и согласования операций, выполняемых на разных процессорах.

В третьих, системные и прикладные задачи, выполняемые на многопроцессорной машине, должны допускать их распараллеливание, иначе любое увеличение "числодробителей" доставит мало радости пользователю.

И, наконец, в четвертых, эффективность многопроцессорных конфигураций в значительной мере определяется стилем выполнения пользовательских задач. Очевидно, что преимущественно интерактивные методы работы от удвоения "камней" выиграют весьма мало - в любом случае тут узким местом окажется пресловутый человеческий фактор.

Решение проблем многозадачности на "железном" уровне - это задача производителей аппаратуры, и влиять на нее мы не в силах (а сможем только учитывать). А вот минимизация "накладных расходов" и распараллеливание задач - это сугубо вахта разработчиков софта, в первую очередь - системного (хотя в последнем случае ее разделяют и создатели программ прикладных). Ну а эффективное использование достижений тех и других - это уже сфера деятельности нас, пользователей.

И нужно сказать, что пользователи Unix-подобных (или, точнее, POSIX-совместимых) операционок, в силу самой специфики их работы и укоренившихся привычек, подготовлены к многозадачности лучше других. И способны получить от нее б*о*льший выигрыш, даже без кардинального update собственной личности.

Действительно, что происходит на типичной пользовательской Unix-машине (в очередной раз оговорюсь, что речь я веду только о пользовательских десктопах и, где-то, о рабочих станциях, но не о серверах)? На ней постоянно что-то компилируется, архивируется/разархивируется, кодируется/декодируется, бэкапится, - и все параллельно, и, в большей или меньшей степени, без интерактивного участия пользователя. Озаботившегося, разумеется, заблаговременно, скриптами для запуска своих задач, выводом полученных данных в логи и прочие файлы, и так далее - за интерактивным режимом остается только просмотр результатов (и, конечно же, их осмысление:-).

Так что вырисовывается заманчивая картина - все это изобилие параллельно работающих задач выполняется действительно параллельно, будучи раскиданным по разным процессорам. Дело остается за малым - реализовать ее в кодах.

Изначально создатели Unix (и ранних его клонов) ни о какой многопроцессорности не помышляли. И один из краеугольных камней его идеологии - концепция монолитных процессор, выполняемых на одном процессоре квазипараллельно, за счет квантования времени, - казалось бы, препятствует реализации распараллеливания задач по разным "камням".

Тем не менее, когда многопроцессорные серверы и рабочие станции стали реальностью в индустриальном секторе, в дополнение концепции процесса была создана и концепция т.н. нитей, или потоков (threads - тот же общегерманский корень, что и в названии литературного жанра - *Пряди об исландцах*). Это - части процесса, выполняемые параллельно и почти независимо друг от друга (в том числе и на отдельных процессорах), разделяющие, тем не менее, ресурсы составленного из них процесса. То есть собственного контекста, в том числе и отдельного пространства памяти, они не имеют, почему носят еще и имя легковесных процессов (light weight process) - обычные Unix-процессы в этом случае можно называть "тяжелыми".

В разных системах концепция нитей была реализована по разному. Выделяют нити ядра, прикладные нити и

промежуточные между ними варианты - прикладные нити, поддерживаемые нитями ядра, с деталями можно ознакомиться в монументальной книге Юреша Вахалии (см. список ссылок). Для нас сейчас важно, что концепция нитей создала предпосылки для реализации мультипроцессорных архитектур в Unix.

Нужно заметить, что сама по себе понятие возникло задолго до Unix - чуть ли не со времен Очакова и ламповой электроники. И уже тогда были выявлены существенные недостатки этой концепции. Однако - увы - за истекшие годы ничего лучшего для поддержки мультипроцессорности придумано не было.

Как уже говорилось (и это очевидно из общих соображений), проблема мультипроцессорности встала в первую очередь в индустриальном секторе. Где по ряду причин (в том числе и исторических) традиционно преобладали проприетарные представители Unix-семейства. И разработчики последних доблестно эту проблему разрешили. Можно спорить, где она была решена лучше, где - не так хорошо, однако общепризнанно: масштабируемость - главная отличительная черта (и главный козырь) и AIX от IBM, и Solaris от Sun, и прочих их братьев-конкурентов.

Свободные POSIX-совместимые ОС, в отличие от проприетарных, разрабатывались преимущественно или в университетско-академической среде, или просто энтузиастами-любителями, как правило, на подручном оборудовании. Среди которого многопроцессорные суперкомпьютеры встречались не так уж и часто (солнце народной мультипроцессорности еще не показало из-за горизонта своих первых лучей). И потому долгое время поддержка многопроцессорности была слабой стороной и Linux, и BSD-систем (по крайней мере, для платформы Intel и совместимых).

Движение свободных операционок в корпоративный сектор, в первую очередь в качестве серверов разного рода, поставило перед ними задачи многопроцессорной поддержки и масштабируемости. И задачи эти постепенно решались: в том или ином виде многопроцессорные конфигурации давно поддерживаются ядром Linux и FreeBSD, недавно такая поддержка появилась в NetBSD и OpenBSD (для платформы i386, в иных архитектурах она имелась, вроде бы, и раньше). Тем не менее, согласно всем встречавшимся мне мнениям, ни одна из этих ОС не дотягивает еще до масштабируемости проприетарных Unix'ов.

Время от времени на форумах возникают споры, где лучше реализована многозадачность - в Linux или во FreeBSD. В обсуждение чего мы вдаваться не будем - уже сам по себе факт периодичности таких споров свидетельствует, что и там, и там многозадачность реализована не идеальным образом. Что косвенно подтверждается недавним тестированием всех свободных операционок (интересно, что участвовавшая в нем Solaris для Intel-архитектуры отнюдь не показала затмевающих результатов - видимо, сведения о ее замечательной масштабируемости относятся исключительно к родной платформе).

Правда, в мире свободного софта испокон веков развивалось другое течение, косвенно связанное с многопроцессорностью - так называемые микроядерные ОС. Идея их - в том, чтобы максимально возможную часть обязанностей ядра (взаимодействие с устройствами, файловыми системами и т.д.) вынести в пользовательское пространство памяти, оставив за ядром только коммуникационные функции (за подробностями - опять же к Вахалии). Что, теоретически рассуждая, должно было бы обеспечить упрощение устройства операционки, легкость ее портирования на новые архитектуры (в том числе и многопроцессорные), а также возможности масштабирования.

Из микроядерных операционок наибольшее признание получило ядро Mach, разработанное в Университете Карнеги-Меллона, а затем разрабатывавшееся в Университете Юты (нужно сказать, что на обоих сайтах давно не видно признаков активности).

Ядро Mach легло в основу ряда проектов разработки свободных ОС - наиболее известен из них Hurd - долгострой проекта GNU, разработка которого недавно переведена на другое микроядерное ядро L4. Однако были другие попытки создания свободных микроядерных ОС - мне известны проекты xMach (правда, сайт его не откликается уже больше года) и Yamit (совсем недавно и его постигла та же участь - возможно, временно).

Можно видеть, что судьба свободных микроядерных проектов была по преимуществу печальной. Что, помимо их невостребованности, объясняется еще и технологическими причинами: судя по всему, разработчики не смогли обеспечить приемлемую производительность своих систем: ведь упрощение устройства ядра влечет за собой усложнение межпроцессных коммуникаций.

Как ни странно, удачные реализации микроядерной архитектуры имели место быть в проприетарном секторе: на ранних версиях Mach основывалась знаменитая система NEXTStep, видевшаяся лет 15 назад платформой фантастического будущего. А предпоследняя версия (Mach 3) легла, вместе с системными службами FreeBSD, в фундамент современной MacOS X. От которой вторично отпочковался уже собственно свободный проект - Darwin (или, точнее, Gnu Darwin).

Таков был исторический фон, на котором развернулись описанные далее события.

Хроника текущих событий

Итак, где-то в середине июня 2003 г. Мэтт Диллон (Matt Dillon), известный, вместе с группой товарищей объявил о начале работы над новой ОС BSD-семейства - DragonFlyBSD. Возник сайт проекта - http://www.dragonflybsd.org и репозиторий ее исходников (созданный 16 июня 2003 г. - этот день можно считать именинами системы) на кодовой базе FreeBSD 4-й ветки, имевшей статус стабильной (хотя в то время уже вовсю развивалась ветка 5-я, вбирающая в себя все инновации BSD-мира).

Новая система получила и собственный тотем - стрекозу, что должно символизировать легкость и быстроту ее. Однако заглавная буква **F** в названии ОС (как подчеркивает Мэтт Диллон, вопреки правилам английской орфографии) говорит, что и ассоциация с мощью дракона и его мудростью также не возбраняется. И кстати, разработчики не гнушаются сокращенных названий своей системы - DragonFly и даже DFBSD; вслед за ними будем и мы время от времени прибегать к ним.

Может возникнуть (и многократно возникает) вопрос: для чего нужна еще одна BSD-система? Разве не вдоволь насмотрелись мы на изобилие Linux-дистрибутивов, чтобы и FreeBSD желать той же участи?

Вопрос этот, конечно, носит сугубо риторический характер. Ведь если новые операционки создаются - значит, это кому-то нужно. И каждая такая система (если она, конечно, действительно нова и оригинальна) привносит в наш мир что-то свое, увеличивая, тем самым, сложность его и разнообразие.

Маленькое отступление: *Бытует мнение, что если бы 4.4BSD (еще не разделившаяся на Net- и FreeBSD) не погрязла бы в тяжбе с ATT и получила бы свободу в конце 80-х - начале 90-х годов, то в разработке Linux не было бы никакой необходимости. Несмотря на свою горячую любовь к BSD-системам во всех их проявлениях, не могу с этим согласиться: если бы Linux'а не было - его следовало бы изобрести. Потому что без него (и привнесенного Линусом метода разработки софта) жить было бы намного менее интересно...*

А в оригинальности DragonFlyBSD отказать невозможно. Ибо, при практически полном внешнем сходстве с прототипом (FreeBSD 4.X), "внутре" у нее все другое: управление памятью и процессами, представление о драйверах устройств и виртуальной файловой системе, вплоть до нового типа файлов - вариантных символических ссылок (varsimlinks).

Впрочем, на теоретической стороне вопроса я останавливаться не буду. Со всеми инновациями DragonFly можно ознакомиться на сайте проекта и в документации к нему (см. ссылки). Вкратце остановлюсь только на наиболее важных моментах.

В основу DragonFly положена модель легковесных нитей ядра (LWKT - Light Weight Kernel Threads), что само по себе и не ново. Новым стал механизм планирования нитей - вместо единого планировщика (sheduler) их было введено несколько, по числу процессоров. Нити привязаны к своим процессорам изначально, однако допускается передача выполнения нити с одного процессора на другой при некоторых особых условиях. Данные отдельных нитей могут быть кэшированы независимо для каждого процессора.

Подобно системам с микроядерной архитектурой, в DragonFly максимум функций ядра вынесен из его пространства памяти в пользовательское пространство (userland). В первую голову это относится к драйверам устройств - таким образом достигается как рост производительности, так и надежность системы в целом, ибо вероятность "падения" системы от воздействия неправильно работающего драйвера стремится к нулю.

Это повлекло за собой отказ от традиционного для Unix механизма системных вызовов (каковой лишь

эмулируется в целях совместимости). Его место занял механизм сообщений (messages) и их очередей, т.н. портов (ports), подобный применяющемуся в микроядре Mach, упоминавшемся в предыдущем разделе.

При этом DragonFly не является микроядерной ОС - базовые функции, такие, как поддержка устройства, несущего корневую файловую систему, как и ее самой, все равно возлагаются на ядро (и размещаются в его пространстве памяти). Однако почти все прочее может быть безболезненно собрано в качестве модулей юзерланда (и в одной из последующих заметок цикла мы в этом убедимся).

Таким образом, можно видеть, что основные инновации DragonFly ориентированы на работу в многопроцессорных системах. А вопрос о том, есть ли что делать простому пользователю на многопроцессорном, с позволения сказать, компьютере, мы рассмотрели уже в ходе обзора хардверной ретроспективы.

Далее, важно, что если матушка DragonFly, FreeBSD, изначально предназначенная только для архитектуры i386, все более эволюционирует в сторону кроссплатформенности (в 5-й ветке к поддержке древней Alpha добавились PowerPC и Sparc), то наша "стрекоза" возвращается на исходные рубежи. И единственной поддерживаемой архитектурой в ней является Intel-совместимая. Правда, в разработке находится и поддержка ее 64-битного расширения от AMD - того самого, которое, похоже, принял на вооружение и Intel (имеется ввиду поддержка собственно 64-битного режима - в режиме 32-разрядном DragonFly прекрасно работает на Opteron и Athlon64 уже сейчас.

Такое ограничение в плане поддерживаемого "железа" может показаться отступлением от истинного Unix Way. Однако... Много лет назад в одном толстом компьютерном ежемесячнике (PC Magazine/RE) была опубликована статья (автора, к сожалению, не помню, а журнал давно потерян) с жутковато-апокалиптическим названием: **Через десять лет все платформы, кроме IBM PC, уйдут в небытие**. Во времена, когда вполне справно, кроме Mac и PC, на столах пользователей трудились и Amiga, и Next, когда начинались, пусть робкие, но попытки выхода на десктопы HP-PA и Alpha, это казалось невероятным.

Однако пророчество исполнилось - и опережающими темпами. И хотя усилия, скажем разработчиков NetBSD вдохнуть вторую жизнь во все древние архитектуры вызывают уважение (хотя и напоминают несколько пассы некроманта), принудительная сила реальности такова: на ближайшие годы i386 (плюс AMD64 в разных ипостасях) останется практически единственной настольной платформой. И разработчики DragonFly с этой реальностью считаются (хотя, вероятно, и вынужденно - вследствие малолюдности команды): в их планах переноса на другие архитектуры не найти (правда, теоретически возможность этого и не отрицается). Что компенсируется возможностью оптимизации под платформу, единственно значимую практически. И, забегая далеко вперед, замечу: такая оптимизация дает свои плоды: по визуальному быстродействию в настольных условиях DragonFly существенно опережает FreeBSD как 5-й, так и даже 4-й ветки. Правда, при выполнении некоторых условий, о которых речь пойдет в третьей заметке цикла.

Наконец, в DragonFly на уровне ядра поддерживается механизм, напоминающий prelinking (предварительное связывание с разделяемыми библиотеками) - насколько мне известно, особенность почти уникальная. И обещающая значительный прирост скорости загрузки (а возможно, и быстроты исполнения) сложных программ, линкуемых со множеством библиотек. Во всяком случае, пример Linux'a, где prelinking уже довольно давно реализован как пользовательский процесс, дает все основания для оптимизма.

Это все технологические обоснования для того, чтобы отнестись к DragonFly не просто как к еще одному BSD-клону. Но есть ведь и субъективный фактор. И это - личность организатора проекта.

К моменту начала работы над DragonFly Мэтт Диллон был широко известен (в узких кругах) благодаря трем разработкам: Си-компилятору для платформы Amiga (именно из этой ОС пришла в DragonFly идея "ядерного прелинкинга"), утилите `dcron` и, главное, системе управления виртуальной памятью во FreeBSD. Не то чтобы он был единственным автором последней, однако вклад его в эту тему был одним из определяющих современный облик FreeBSD. Да и к аналогичной подсистеме ядра Linux он приложил руку.

Что немаловажно, в специальной статье (присутствующей в официальной документации FreeBSD) Мэтт сумел описать архитектуру виртуальной памяти языком, понятным для широких масс трудящихся. Очень рекомендую к прочтению - во введении к ней высказано немало интересных мыслей общего характера. Тем

более, что русский перевод ее доступен, например, здесь.

Так вот, позволю себе процитировать фрагмент из означенного мемуара:

Самой большой ошибкой, которую может допустить программист, является игнорирование истории, и это именно та ошибка, которую сделали многие другие современные операционные системы... Я плохо переношу тех, кого не учит история.

И эта фраза (значение которой не ограничивается программированием) вселяет надежду, что в DragonFly ошибки прошлого будут учтены. И первое же знакомство с этой системой эту надежду подтверждает.

Однако вернемся к истории DragonFly. Некоторое время проект развивался как бы закулисно. Конечно, все желающие ознакомиться с прототипом системы могли свободно получить ее исходники с сайта проекта через CVS и развлекаться с ними в свое удовольствие (нужно ли говорить, что DragonFly распространялась и распространяется на условиях лицензии BSD?). Однако в виде, пригодном для установки простыми смертными, она не существовала. По крайней мере, не распространялась.

Как вдруг, начиная с мая 2004 г., один за другим начинают появляться iso-образы компактов различных бета-версий DragonFly. Они не имели еще инсталлятора: следовало, руководствуясь документацией (вполне, впрочем, ясной), вручную разметить диск, создать файловые системы, перенести на них с дистрибутивного CD необходимые каталоги и произвести еще кое-какие манипуляции (типа создания файлов устройств и настройки стартовых сервисов). Задача была не то чтобы сверхъестественно сложной - но и не вполне тривиальной.

А затем... Затем, в июне 2004 г., появился пре-релиз DragonFly, точнее, DragonFlyBSD 1.0RC1. От своих бета-предшественников он отличался тем, что уже имел инсталлятор - BSD Installer, разработанный в рамках самостоятельного проекта как универсальный установщик для любых BSD-систем. Но впервые, насколько я знаю, опробованный именно на DragonFly.

Отличительная особенность BSD Installer в том, что теоретически к нему могут подключаться фронт-энды, созданные на базе самых разных библиотек. В дистрибутиве DragonFly был использован текстовый (псевдографический) фронт-энд на основе библиотеки `ncurces` (подробнее о нем будет говориться в следующей заметке).

Надо заметить, что уже в те далекие времена установка проходила без малейших осложнений (в том числе и на "железо", на которое FreeBSD устанавливалась с трудом). Однако к использованию система была пригодной с трудом. Ибо, кроме базиса (соответствующего FreeBSD Distributions), не содержала почти ничего - ни Иксов, ни прекомпилированных пакетов (за исключением нескольких консольных утилит типа `cvsup` и `cdrtools`), ни чего-либо вроде системы портов (хотя в будущем таковая была обещана).

Нужно сказать, что пре-релизная стадия для DragonFly оказалась очень короткой: уже в начале июля (11-го, если не ошибаюсь, числа), практически в годовщину проекта, было объявлено о выходе релиза - DragonFlyBSD 1.0-RELEASE. Правда, и он просуществовал недолго: как это нередко бывает в него вкралось несколько мелких, но весьма неприятных ошибок. Хотя выявлены они были мгновенно и столь же быстро исправлены в баго-фиксном релизе 1.0A. Каковой и по сей день доступен на официальном сайте проекта и является официальной (точнее, супер-официальной) версией системы.

Начиная с этого момента, DragonFly можно было считать более-менее пригодной к использованию. Сам по себе дистрибутив по прежнему не включал ни пакетов, выходящих за рамки базовой системы, ни системы портов. Однако с самого момента его выхода прекомпилированные пакеты для DragonFly можно было найти сразу на двух сайтах - LiveBSD и GoBSD. Причем если на первом имелся лишь самый минимум-миниморум дополнительных приложений, то второй содержал практически все (за небольшими исключениями), что имелось в коллекции портов FreeBSD (с системой портов коей DragonFly сохраняет, при некоторых оговорках, совместимость и по сей день).

Однако жизнь не стояла на месте, и дальнейшая работа над системой не прекращалась: с интервалом в 3-5 дней на ftp-сервере проекта и его зеркалах появляются текущие снапшоты - как в виде дерева исходников в CVS-репозитории, доступном через `cvsup` (который, напомню, имеется в составе базовой системы), так и в виде iso-образов дистрибутивных CD. Образы снапшотов выходят в свет в двух вариантах: собранные

компилятором `gcc` версии 2.95 и 3.4.X. Первые позиционируются как рабочие, вторые - как преимущественно экспериментальные. Что, впрочем, не мешает и им функционировать вполне справно.

Вообще говоря, модель разработки DragonFly отличается от таковой FreeBSD своей линейностью: она не разветвляется на стабильную и разрабатываемую ветки (по крайней мере, пока). Просто некоторые из промежуточных снапшотов имеют в имени пометку **stable**.

Надо заметить, что команда разработчиков DragonFly невелика: на соответствующей странице сайта перечислено, кроме Мэтта, 21 имя. Но эффективность их работы не может не вызвать восхищения.

Параллельно разработке самой DragonFly развивается упомянутый выше сайт GoBSD, созданный специально для дистрибуции новой системы. Распространяемый на нем вариант DragonFly основывается на одном из промежуточных стабильных снапшотов, но несколько отличается от оригинала по комплектации.

Двигалось дело и с портированием сторонних приложений. Во-первых, как я уже сказал, в DragonFly по сию пору не возбраняется использование системы портов матушки FreeBSD. Правда, ввиду того, что это все же разные ОС с разными ядрами, не все порты родительской коллекции под DragonFly собираются успешно. Однако для коррекции этого выступает подсистема `dfports` - в нее включены "неудачные", с точки зрения DragonFly, порты. И такой метод работает. Доказательством тому длинный список бинарников на GoBSD: они собраны именно через обычные порты FreeBSD с учетом изменений `dfports`.

Позднее на GoBSD пошли другим путем: прикрутили в DragonFly `pkgsrc` - портообразную систему, заимствованную из проекта NetBSD. И таким образом в распоряжении пользователя новой ОС сразу оказалось все изобилие открытого и бесплатного софта, портированного на эту платформу - а надо отметить, что на NetBSD он портирован если не весь, то Bouno Parte. Именно наличием тарбалла древа `pkgsrc`, адаптированного для DragonFly, и отличается в первую очередь диск, распространяемый с GoBSD, от снапшотов проекта и его зеркал.

Однако оба варианта - не более, чем временные паллиативы. Потому что в рамках проекта DragonFly разрабатывается и собственная система управления пакетами, появление которой ожидается во второй официальной версии.

Такова на сегодняшний день короткая, но насыщенная история операционной системы DragonFlyBSD. Надеюсь, она не окончена, и будет дописываться по мере развертывания событий. А пока зададим несколько практических вопросов.

Пригодна ли DragonFly в современном его виде для практического использования? Не скажу за серверный сегмент, но в настольном - безусловно, пригодна. За тот промежуток времени, что она стоит на моей машине, я не столкнулся ни с какими нестабильностями или необъяснимыми глюками. А все неясности в ее работе были связаны исключительно с собственным недопониманием и легко разрешались после некоторых раздумий и чтения документации.

Каковая, к слову сказать, достаточно обширна. Кроме монументального DragonFlyBSD Handbook (как будто этого мало - стопы распечатки толщиной 12 см?), имеется еще ряд документов, перечисленных в конце заметки. Нужно только помнить, что пока система развивается быстрее, чем актуализируется документация...

Вопрос второй: можно ли сей же момент перейти на DragonFly как основную рабочую систему? И здесь я ответил бы - да, хотя и с некоторыми оговорками. Потому что, кроме надежного и полнофункционального базиса, для нее есть и достаточно прекомпилированных пакетов, и аж две, проверенные временем и адаптированные системы пакетного менеджмента, и поддержка всего обычного "железа". В последнем отношении отличия с FreeBSD если и есть - то только в положительную сторону.

Внешне DragonFlyBSD очень похожа на FreeBSD, и любой пользователь последней освоится с ней очень быстро. Да и приверженец таких Linux-дистрибутивов, как Slackware, CRUX, Archlinux, Gentoo, не почувствует себя уж совсем в незнакомой остановке. А в неясных случаях ответы можно найти в документации.

И, наконец, вопрос третий и последний: а нужно ли немедленно все бросить и идти собирать свой

велосипед, то есть переходить на DragonFly? И тут ответ будет, скорее всего, отрицательным: менять привычную и отлаженную Linux- или BSD-систему на DragonFly прямо сейчас резона нет. И она еще не раскрыла своего потенциала, звездный час этой операционки, я уверен, в будущем (и не очень далеком).

Однако уже сейчас ничто не мешает к этому будущему готовиться. Установив новую ОС параллельно наличной BSD- или Linux-системе и ковыряясь с ней в свободное время. Хотя, если есть твердое желание перейти на какую-либо BSD-систему - почему бы и не DragonFly? Повторяю, никаких причин, тому препятствующих, ныне не осталось.

Ссылки

Ю. Вахалия. Unix изнутри. СПб: Питер, 2003, 844 с.
Фундаментальное руководство по устройству Unix'ов всякого рода.

DragonFlyBSD
Официальная документация проекта (скорее, ее программа). Доступна также в русском переводе, пока еще неполном и несовершенном.

DragonFly BSD Handbook.
Подробное руководство, аналог знаменитого FreeBSD Handbook, и которого в значительной мере заимствовано содержание.

Ресурсы по теме
Тематическая подборка всех доступных на данный момент материалов по DragonFly BSD.

Алексей Федорчук, "Почему DragonFly?". Источник: Библиотека ЛинуксЦентра