Переосознавая ОС

Скотт Эдвардс (J. Scott Edwards), Понедельник, 24 Январь 2005, 05:22

Это третья моя попытка все это записать. Первая была слишком подробной. Вторая по прежнему была слишком длинна. На этот раз я решил просто выложить свои идеи и постараться пояснить их просто и кратко. Более всего это похоже на обзор. Просто существует слишком много подробностей, чтобы пытаться выжать все их здесь. Многие из этих идей на самом деле не новы, о некоторых уже говорили до этого. Я просто пытаюсь понять, как их собрать вместе в единую систему.

Я работаю с компьютерами уже 30 лет и они, определенно, не стали ничуть проще. Фактически, я даже уверен в том, что они стали сложнее. Конечно, сегодня я могу делать такие вещи, о которых никогда и подумать бы не мог, но, похоже, что в то время как компьютеры стали в тысячи раз мощнее, они, в то же время, стали в тысячи раз сложнее.

Я опробовал практически все основные существующие операционные системы и я всегда попадал в какие-то сложные ситуации. Поэтому я не переставал думать, - "А что, если мы начнем с нуля и переосознаем то, как операционная система работает в целом?". Одной из первых операционных систем, с которыми мне довелось поработать, была Exec 8 на Univac 1108. В ней было реализовано большинство из используемых всеми основными современными операционными системами концепций. Сейчас вы, возможно, думаете, - "у нее не было графического интерфейса!". И это правда. Но я говорю о концепциях, лежащих внутри: процессах, файлах, о том как запускаются и взаимодействуют программы - вот об этих вещах.

Некоторое время я занимался мозговым штурмом и теперь я хочу выложить некоторые сложившиеся идеи. Сейчас это всего лишь идеи, некоторые или все из которых, возможно, вообще не будут работать. Возможно, что я не догадался до какой-то грани идеи, которая делает ее реализацию невозможной. Но я чувствую необходимость опубликовать их по нескольким причинам. Во-первых, я надеюсь, что когда они будут где-нибудь опубликованы, они перестанут кружить в моей голове, и я смогу поспать. Во-вторых, я не хочу ситуации вроде той, что была с Microsoft и SenderID.

И последнее, что я хотел бы сказать прежде чем начать. В качестве примеров я собираюсь упомянуть некоторое используемое мной ПО, такое как Linux, OpenBSD, gcc, K3B и прочее. Пожалуйста, не надо воспринимать это как унижение мной этого ПО. Просто я использую то, с чем лучше всего знаком. Я по прежнему считаю, что это отличное ПО, которое я использую каждый день и буду продолжать использовать в ближайшем обозримом будущем.

А теперь, без лишней суеты, я полагаю, что пора оттолкнуться обоими ногами и запрыгивать.

Простота

Я читал книгу "Better, Faster, Lighter Java" и, хотя автор говорит именно о Java, он мог бы так же говорить о вычислительной технике в общем.

Глава вторая называется "Сохраняйте " ("Keep It Simple") и поясняет ценность простоты. Она

даже доходит до утверждений о том, что "простота может быть ключевой ценностью". Я надеюсь, что я смогу передать или поставить ссылку на первую часть второй главы, поскольку я думаю, что это применимо ко всему в вычислительной технике.

Так что, главная моя цель - сделать всю систему настолько простой и элегантной, насколько это возможно. Если сложностей не избежать, то их следует сохранить в одном месте и не позволять им инфицировать оставшуюся систему.

Плоские файлы и реляционные базы данных

Самое серьезное из того, о чем я могу подумать, и что добавляет сложности в компьютерные программы - это хранение данных. Мне кажется, что в каждой нетривиальной программе есть серьезные куски кода просто для того, чтобы танцевать вокруг файлов. Существуют конфигурационны файлы и файлы данных, большинство из которых не используются напрямую программой, а интерпретируются и конвертируются во внутренний формат приложения.

Альтернатива хранению данных в плоских файлах (flat files) - хранить их в базе данных, чаще всего в реляционной базе данных. Хотя такой подход имеет некоторые преимущества перед плоскими файлами, используя реляционную базу данных, программе придется делать запросы к ней, а затем, как правило, преобразовыватьрезультат во внутренние структуры данных программы. Это особенно неудобно при использовании объектно-ориентированног программировани. Конечно, существуют и объектно-ориентированныебазы данных, но я не слышал, чтобы хоть одна из них нашла широкое применение, особенно из открытых, вроде MySQL или PostgreSQL.

Моя идея состоит в том, что необходимо сохранять данные на диске в родном объектном формате программы. Таким образом, вместо постоянных преобразований и интерпретаций данных, приложение может просто обращаться к такому объекту напрямую. А когда эти данные нужны в формате плоского файла, у вас будет приложение-конвертер (объект), которое сможет обращаться к внутренним данным и конвертировать их в файл или формат сплошного типа.

Например, предположим, что у вас на компьютере есть какие-либо сжатые (с помощью Ogg Vorbis или чего-то подобного) аудио объекты. И вы хотите зажечь аудио CD, который можно проигрывать в обычном плеере аудио CD. Вы создадите объект "список проигрывания" (playlist) и соедините его вывод (об этом позже) со входом объекта-конвертера Ogg Vorbis, а затем с объектом прожига аудио CD.

Где это хранится?

Еще одна проблема, которую я бы хотел решить, это "где находятся эти данные?" или "что я сделал с тем файлом?". Буквально вчера я искал электронную таблицу, в которой была некоторая необходимая мне информация. Одно время я держал ее на своей настольной машине. Потом, в течение прошлого года, я несколько раз менял эту машину и еще чаще переустанавлива различные операционные системы. Так что, в конце концов, я положил ее на свой файловый сервер. Но когда я начал искать ее, я не видел ее нигде. Хуже того, я не мог точно вспомнить как она была названа. Мне пришлось провести много поисков, прежде чем я

окончательно понял дурацкую вещь. Я потратил, наверное, минут 20 на ее поиски и пока шло время я уже начал волноваться, что я потерял ее и думал когда же (если вообще?) я делал ее архивную копию. Да мне просто не нужен такой стресс!

Моя мысль по этому поводу состоит в удалении разделения между различными разделами, дисками, файловыми серверами, носителями CD, DVD и прочими. Больше никаких разделов C:, D: и E:. Никаких host7:/shared... Никаких монтирований CD носителей. Все это всего лишь одно большое объектное пространство хранения данных. Если оно находится вовне, машина может его найти. Если это публичная информация, то ее можно получить через Интернет. Если она хранилась на кассете, CD или на архивном DVD и этот носитель не находится в приводе, ОС скажет вам на каком носителе хранился этот объект, и вы сможете найти его и вставить, а приложение сможет продолжать работу даже не зная что произошло. Если в это время у вас нет доступа к носителю, у вас должна быть возможность остановить приложение прямо там, где оно находится и перезапустить его позже в точности в том же состоянии, когда у вас появится носитель (подробнее об этом позже).

**Похоже, что Google, Apple и другие придвигаются к тому, чтобы сделать это реальностью в существующих сегодня операционных системах, так что это хорошая новость.

Ссылаемся друг на друга

Еще одной проблемой плоских файлов является отсутствие информации о данных, хранящихся в файле. В BeOS была база данных, встроенная в файловую систему, что было приятной добавкой, и, как я понимаю, в Longhorn от Microsoft тоже должна была появиться какая-то разновидность встроенной базы данных до того, как ее отменили. Также я недавно обнаружил, что файловая система ext3 для Linux позволяет добавлять к файлам дополнительные "".

Но я хочу пойти дальше всего этого. Моя идея в том, что необходимо брать объект так, как он определен в сегодняшних языках, вроде C++ и Eiffel, и добавлять некоторую дополнительную информацию к нему. Когда был создан объект, кем был создан объект, его контрольная сумма MD5, биты защиты, обычные мета-данные, которые имеет файл сегодня в ОС - информацию такого рода.

В дополнение к этому, самым важным добавлением является таблица ссылок. Это таблица, которая поддерживается ОС, и хранит список всех объектов, связанных с этим объектом. Это дает нам несколько преимуществ. Во-первых, она позволяет легко находить любой объект, который связан с некоторым объектом. Во-вторых, она позволяет определить, какие из объектов сейчас используются в системе. Если ни один активный объект в системе не ссылает на данный объект (отличный от папки), то его можно вернуть в хранилище и удалить из активной системы.

Вернемся к музыкальному примеру, все музыкальные объекты будут ссылаться на исполнителя, который их создал. Все треки Beatles будут ссылаться на объект Artist Beatles. Объект Artist Beatles не будет специально ссылаться на песни, так как песни будут единственным типом объектов, которые смогут ссылаться на Beatles. Мы ведь не хотим раздувать объект Artist лишними ссылками на каждый возможный тип объектов, который может ссылаться на него. Мы даже не знаем, что из себя представляет каждый тип объектов,

который может ссылаться. И даже если бы мы это сделали, нам бы пришлось менять класс Artist каждый раз при создании нового класса, который ссылается на класс Artist. Вместо всех объектов мы просто будем иметь общий список, который сможет указывать на любой тип объектов (класс "ANY" в Eiffel). Этот список будет автоматически поддерживаться ОС и обновляться во время работы программы. А когда вы захотите найти все имеющиеся у вас песни, созданные The Beatles, вы просто возьмете объект Beatles и поищете в списке его ссылок объекты класса "Track".
[newpage]

Версии

Я также предполагаю сохранение версий каждого объекта при его изменении. Так что, вместо реального изменения объекта будет производиться его клонирование и дальнейшее изменение клона. Таким образом, если вы сделали ошибку и вам необходимо откатить изменения, или если вы хотите увидеть, какие произошли изменения, вы можете увидеть различия. То есть у вас будет встроенный CVS.

Продолжая эту тему, должен быть механизм очистки системы от старых объектов, которые более не нужны. Например, если вы работали над каким-либо исходным кодом и версия 7 у вас работала, а потом вы сделали изменения в версиях 8, 9, 10, 11 и 12. А в версиях 9, 10 и 11 содержались ошибки, которые вы не хотите сохранять. Когда вы завершите, у вас должна быть возможность удалить эти ненужные версии.

Сегодня дисковое пространство чрезвычайно дешево. Я только что проверил, вполне можно купить жесткий диск на 200 гигабайт за чуть более 100. А диски Blu-ray от Sony будут способны хранить 23 гигабайта данных в одном слое. Имея такие объемы устройств хранения, я думаю, что сохранение нескольких копий вашей работы имеет смысл.

Безопасность

Безопасность - один из аспектов, который я еще не окончательно продумал. Но вот то, над чем я размышлял:

Различные части объекта могут иметь различные уровни безопасности. Например, если у вас есть объект Person, некоторая его информация может быть публична, например, имя человека и адрес его электронной почты. Но другие части могут быть доступны только меньшей группе, такой как ваши коллеги по работе, например, номер вашего сотового телефона. А какая-то информация будет доступна только избранным, такая как номер вашего социального страхования.
Все непубличные данные, когда они не находятся в оперативной памяти, будут шифроваться. Когда эти данные хранятся на любом носителе или передаются по сети, они будут зашифрованы.
Должен быть какой-то механизм создания групп, который позволит сгруппировать людей, схоже с механизмом групп в Unix/Linux.
Также должны существовать различные уровни безопасности. Например, когда вы авторизованы как вы сами, у вас будет доступ к некоторым данным просто исходя из вашей аутентификации по логину. Затем уровень безопасности выше, когда даже при том, что вы авторизованы своим логином, вам необходимо ввести еще один пароль для доступа к специфичным данным. Возможен еще один уровень безопасности, еще выше, когда вам

придется ввести еще один пароль или использовать какую-либо смарт-карту, ключ, или что-то подобное. В Mac OS X используется схожий механизм, разделяющий обычный доступ и "администраторск", который используется для установки ПО или внесения серьезных изменений в конфигурацию. Я бы хотел иметь эти несколько уровней из одного аккаунта пользователя. Например, вы можете иметь нормальный уровень безопасности, когда вы пишите отчет о глобальном потеплении. Но вам, наверное, захочется иметь повышенный уровень безопасности для объекта, который хранит номер вашего социального страхования. Я хочу также интегрировать что-то вроде SE Linux, который устанавливает "" того кто что может делать. Например, не должно быть одного могущественногопользователя, который может читать любой объект системы, даже тот, который содержит ваш номер социального страхования.

Кэширование в ОЗУ

Я также хочу изменить способ использования ОЗУ. Я хочу использовать основную оперативную память просто как быстрый кэш для объектов, хранящихся на жестком диске (или, если бы у нас не было жесткого диска, объектов, загруженных из сети). Таким образом, насколько это только возможно, цельный образ запущенного приложения хранится в актуальном состоянии на диске. Когда объект выполняется в ОЗУ, он должен копироваться на диск настолько скоро, насколько это возможно.

Такая схема дает множество преимуществ. Одно из них в том, что если у нас произойдет отключение питания, то потери будут менее важны. Что привело меня к одной из других идей...

Больше последовательноти

Еще одна вещь, которую я хочу опробовать - это сделать приложения более последовательныи. Похоже, что сегодняшние приложения до сих пор работают также, как они это делали годами ранее в системах пакетной обработки. Они стартуют полностью заново каждый раз, читают свои конфигурационны файлы, входные данные, производят свою обработку, пишут выходные данные и останавливаются.

Мне кажется, что вместо запуска из ничего каждый раз и затем завершения приложений так, как их никогда и не было, было бы лучше если (хотя бы некоторые) приложения сохраняли свое состояние нетронутым. Это будет более похоже на их включение и выключение.

Например, я не понимаю, почему офисный текстовый редактор не может быть более похож на печатную машинку? Положим, у вас есть листок бумаги в печатной машинке и вы находитесь в процессе напечатывания чего-нибудь. Вам звонят, отрывают от дела, вам надо лететь в Стамбул, вы выключаете ее (полагаем, что это электронная печатная машинка) и оставляете. Через шесть месяцев вы возвращаетесь, включаете ее, и она в том же состоянии, как она была когда вы ее оставили.

Хотя и правда то, что в большинстве текстовых редакторов есть функция "недавние ", с помощью которой вы можете переоткрывать документы, обычно это не совсем то же. А что если произойдет потеря питания, сможете ли вы легко возвратиться к тому состоянию, что было до нее?

Я полагаю, что должна быть возможность иметь приложения, которые можно просто включать и выключать, а они будут оставаться в том состоянии, в котором их выключили.

Спускаемся ниже

Большая часть из того, что я обсуждал до этого - это высокий уровень, в реальности лишь верхушка операционной системы. На самом же деле мой первоначальный план по доказательству концепции - построить упомянутое выше объектное хранилище поверх Linux и/или одной из BSD.

Но затем мне захотелось расширить объектную парадигму дальше. Я хочу, чтобы все в операционной системе было объектно-ориентированно, кроме самого ядра. Я полагаю, что в ядре получится что-то **вроде** виртуальной машины Java. Когда компьютер включается, он загружает виртуальную машину, скорее всего с чем-нибудь вроде Just In Time компилятора для ускорения. После этого все объектно-ориентированно.

Что в языке

Я знаю, что некоторые люди не соглашаются с тем, что объектно-ориентированноепрограммировани дало нам то, чего мы от него ожидали. Но я обнаружил, что если использовать правильный ОО язык, то он дает значительные преимущества в продуктивности.

Хорошо, сейчас я нарушу один из своих принципов и немного пройдусь по C++. Я знаю, что можно писать хороший код на C++. Можно писать хороший код на языке ассемблера. Вся штука в количестве вовлеченных сложностей. Я работал во многих проектах на C++, но я не работал ни над одним, где я бы чувствовал его продуктивность. Даже те проекты, что я делал полностью сам на C++, были ужасом. Я обнаружил, что обычно это приводит к одной из двух вещей: разработчики приклеиваются к небольшому подклассу языка, так что это на самом деле просто C с несколькими объектами, вброшенными тут и там; или он становится настолько сложен, что у большинства людей появляются трудности с его пониманием, а добавление или поддержка функций становится кошмаром. Я помню как в одном проекте разработчик жаловался, что для добавления одной простой возможности ему пришлось изменять 42 различных файла. Когда код становится настолько сложен, это плохо. В своей книге "Thinking in C++" (в русском переводе - "Философия C++" - примечание переводчика) Брюс Эккель говорит о том, что C++ даже запутаннее, чем Ada. В Интернете можнество сайтов о проблемах с C++, например: Why C and C++ Are Bad. По моему мнению, язык должен делать написание хорошо спроектированноо и легкого в поддержке кода настолько простым, насколько это возможно. И мне кажется, что C++ в области здорово проигрывает.

Я знаю, что сейчас 99% программистов всего мира просто прекратят чтение. Но мой опыт показывает, что Eiffel и Design by Contract (контрактное проектирование) - абсолютное благо для программировани.

Давайте я приведу пример где, как я думаю, контрактное проектирование может быть серьезным улучшением. Одной из самых серьезных причин дыр в безопасности ПО является переполнение буфера. И я думаю, что эта ошибка лежит в языке C, он просто ничего не

проверяет. А программисты не идеальны, они делают ошибки и С позволяет им делать ошибки. Я уверен, что если бы использовался язык вроде Eiffel, который проверяет состояния ошибок, ПО было бы более безопасно. (Единственная оговорка здесь в том, что вы не можете отключить эти проверки, они всегда должны быть на месте. Я бы с удовольствием пожертвовал немного производительноти ради безопасности.)

Теперь, когда я высказал все это, я хочу сказать, что Eiffel - это не совсем то, на чем я хотел бы программировать Это лучший из всех языков, что я использовал. Он включает в себя простоту, точность и легкость программировани. Программы проще отлаживать и они меньше нуждаются в отладке. Но я бы хотел изменить также и парадигму программировани.

Одна из вещей, которую я хочу сделать - это интеграция документации в язык программировани. (Уверен, сейчас и все оставшиеся программисты прекратят чтение) Я думаю, что должен существовать какой-либо способ сделать так, чтобы документация и справка программировалиь прямо в языке программировани. Она не должна быть отдельной сущностью. Я ненавижу документировани также как и любой программист, вы только что проделали этот сумасшедший объем работы по написанию отличной программы, а теперь вам приходится возвращаться назад и документироватьее. Что может означать столько же или больше работы, чем написание программы. Да это отстой! Так что, я думаю, что будет лучше, если создание документации и справки будет встроенно прямо в программу, в одном месте. Тогда это можно будет делать в то же время, когда и программу.

И в этих же строчках, я думаю, что это должно быть реализовано так, чтобы вы могли встраивать другие типы сущностей, кроме текста, в саму программу. Полагаю, что должна быть возможность просто вставить рисунок, диаграмму или даже картинку посреди исходного кода. Я думаю, что исходные коды должны быть всего лишь еще одним объектом, который может содержать объекты любых других типов.

Исходя из хранения кода как объектов, вместо текста, мне пришла в голову еще одна мысль - вы сможете смотреть/редактировать код в вашем любимом формате. Если вам нравится картина фигурных скобок, которую можно видеть в языках семейства С, вы можете просматривать программу вот так:

```
if (a == b) { c++; }
```

Можно будет даже убрать все эти аргументы выше фигурных скобок:

```
if (a == b) { c++; }
```

А если вы предпочитаете более легкий для чтения синтаксис:

```
if a = b then c := c + 1; end
```

[newpage]

Объектный редактор

Так как все хранится как объект, должен существовать простой редактор. Что-то очень схожее с текстовым редактором, что позволит вам просматривать и манипулировать

объектами. Он должен работать со всеми объектами (полагая, что у вас есть права на просмотр и редактирование этого объекта). Конечно, он не будет способен показывать или изменять те части объекта, что зашифрованы, если, правда, у вас нет ключа.

Например: во многих программах есть конфигурационны файл, теперь у них будет объект конфигурации и вы сможете имзенять конфигурацию таким способом. Похоже на сегодняшнее редактирование конфигурационноо файла текстовым редактором.

И ОС не бесконечна тоже

Полагаю, что то, что мы можем делать с приложениями, должно быть возможно и со всей операционной системой. Она должна находиться в более сохраненном состоянии при старте. Она должна просто восстанавливатья к тому состоянию, в котором она находилась, когда вы выключали ее.

У меня настолько плохо с памятью, что мне приходится часто оставлять мои машины постоянно включенными. И это стало одной из хороших вещей, что мне нравятся в Linux, то, что могу оставлять его включенным месяцами. Потом, когда я возвращаюсь, все в том же состоянии, что и когда я уходил. Что плохо в этом - так это отключения питания. Обычно мне приходится потратить час на восстановление подобия того, что я делал. И в конце концов это никогда в точности не повторяет все, что я делал. Я просто не могу понять, почему невозможно восстановить машину в практически то же состояние после отключения питания. Конечно, если оно застало вас посреди напечатывания чего-либо, что-то должно потеряться. Но почему нам приходится начинать из ничего?

Модули и ПО по требованию

Одной из проблем, с которой я столкнулся на нескольких вариациях Linux, это установка ПО. Иногда мне хотелось установить небольшую программу для выполнения одной задачи. Но когда я направлялся устанавливать ее, скажем, через RPM, у нее был список зависимостей длиной с милю. Конечно, "apt" и "yum" в большинстве случаев способны справиться с проблемой зависимостей, но меня по прежнему раздражает трата места и необходимость установки на мой жесткий диск всего этого дополнительногобарахла, которое я никогда не буду использовать.

Например, однажды я делал минимальную установку SuSE на машине (поскольку у нее было очень мало места на винчестере). Я хотел использовать K3B просто для создания CD и DVD с данными. Однако, когда я попробовал установить его, он потребовал библиотеки MP3, Ogg Vorbis, FLAC, список был достаточно длинным. Мне не было нужно ничего из этого всего на той машине, оно было совершенно излишним.

Я думаю, что ПО должно устанавливаться как меньшие модули, которые работают независимо. Я также думаю, что только минимальное количество программ должно инсталлироватьс в систему, когда она только устанавливается. Затем, когда вам понадобится выполнить определенную задачу, соответствующее ПО будет установлено автоматически (после вашей авторизации, конечно же) и будет установлено только то ПО, которое вам действительно нужно.

Я уже обращался к этому ранее, но я хочу упомянуть это еще раз. Я хочу разработать механизм общения независимых модулей друг с другом. Схоже с тем, как в Unix/Linux вы можете связать стандартный ввод и стандартный вывод вместе в конвейер. Но имея другие типы интерфейсов (аудио/видео/что угодно) и соединяя их во время исполнения для выполнения какой-либо другой функции. Например, если у вас есть, скажем, плагин фильтра, то вместо того, чтобы просто иметь возможность использовать этот плагин внутри аудио-редактора, вы сможете использовать его на уровне ОС и включать его между выводом какого-нибудь устройства и микшером. Или, возможно, вы встроите его в микшер.

Невозможно изменить все

Хотя, может быть, очень заманчиво попробовать изменить все в компьютерах, чтобы увидеть, можно ли сделать все лучше, я, все же, решил, что есть пределы того, что действительно важно.

Что же, аппаратура по прежнему остается железной. И было бы дорого менять аппаратуру, да и это сделало бы невозможным для людей без специальной аппаратуры использовать наше ПО, а также дало бы повод для споров о ПО с открытыми исходниками. Так что нам придется пользоваться существующей аппаратурой.

Мне кажется, что то же самое можно сказать и о сетях. Не имеет смысла использовать что-то отличное от TCP/IP. TCP/IP, похоже, достаточно гибок для того, чтобы поддерживать все, что я могу только представить. Таким образом мы сможем общаться со всем оставшимся миром, а это может быть только хорошо.

Независимость от порядка байтов

Одна из проблем которую я еще не решил - это независимость различных машин и данных от порядка байтов. Так как у машины есть доступ к определению класса для каждого объекта, она сможет автоматически преобразовыватьданные для обработки на текущем процессоре.

Диллема в том, в каком же виде следует их хранить? Или стоит всегда хранить на диске в формате того процессора, к которому он подключен? Заманчиво выглядит хранение информации всегда в одном формате, предположим, с обратным порядком байтов, чтобы удовлетворять сетевому порядку байтов. Но так выходит, что самые распространенны машины (x86) используют прямой порядок байтов и им придется всегда держать удар.

Альтернативой может быть то, что делается сейчас - хранить информацию на жестком диске в родном для процессора формате. Это дает преимущества в скорости, необходимость в преобразовании порядка байтов возникает только при передаче по сети.
[newpage]
Пользовательски интерфейс
Я еще не думал особо над деталями графического интерфейса, но я определенно хочу пойти дальше 2D интерфейса. Проект Sun Looking Glass выглядит здесь заманчиво и будет интересно посмотреть, что еще произойдет в этой области.

Я уже представлял что-то, расширяющее эту идею, когда у вас на самом деле уже нет окон, но есть нечто вроде различных комнат, музыкальная комната, библиотека, комната TV и

прочие. Но, как я уже сказал, я не прилагал к этим мыслям таких усилий, как к низлежащим технологиям.

Одной из возможностей, которая должна вытекать из модульности ПО, это то, что эти модули можно будет соединять таким образом, чтобы у приложения были разные интерфейсы без изменения самого приложения.

Например, у приложения будет управляющее соединение к нему. У него не будет интерфейса, закодированногов само приложение. Так вы сможете присоединять к нему интерфейсы, хоть GUI, хоть командную строку, все зависит от ваших предпочтений.

С чего начать?

Вот это большой вопрос. Принять подход сверху вниз или снизу вверх? Однажды я читал, что "Настоящие программисты начинают с середины в обе стороны!". Или, возможно, стоит начать с обоих концов, чтобы посмотреть, сойдутся ли они где-нибудь посередине.

На самом деле, мой текущий план - построить Объектное хранилище поверх существующей ОС. Посмотреть, собирается ли оно работать вообще и как хорошо оно работает. На самом деле, я не вижу ни одной причины, по которой было бы невозможно построить практически всю ОС поверх другой ОС, схоже с User Mode Linux.

Еще одна вещь, которую я хочу исследовать, это насколько хорошо объектная парадигма работает на низких уровнях операционной системы. Похоже, что она там придется кстати, но я могу ошибаться. Одна из странностей в Linux для меня, хотя похоже, что работает она весьма успешно - это симулирование устройств SCSI для приводов и дисков IDE и SATA. Это похоже на какую-то лишнюю работу.

Мне кажется, что это пришло из какой-то программы, вроде 'cdrecord', которая хотела общаться со SCSI устройствами. Хотя, я уверен, что для этого есть и другие серьезные причины. Но в объектно-ориентированноммире, приложение (cdrecord) можно написать для общения с абстрактным объектом (CD-Drive), который мог бы инициализироватся или как SCSI, или как IDE объект. Так что я хочу исследовать этот момент, чтобы посмотреть, возможно ли это, или я витаю в облаках.

Также я планирую использовать столько существующего ПО с открытыми исходниками, сколько возможно. Не имеет смысла писать все драйвера устройств с нуля (и на это потребуются годы). Так что я хочу посмотреть, какие драйвера можно будет использовать из Linux или любой из *BSD и, возможно, обернуть их в объектные интерфейсы. Похоже, что это и есть прелесть открытых исходников, то, что вам не обязательно надо начинать с нуля. По крайней мере, вы, как минимум, можете поглядеть на то, как кто-то другой делал что-то и попытать улучшить это. Если хотите, вы будете стоять на плечах гигантов.

Закругляюсь

Что ж, получилось уже больше, чем я хотел изначально. И есть еще множество подробностей, которые я бы действительно хотел включить сюда. Я открыл проект на SourceForge, где вы сможете получить дальнейшую информацию. Он будет находиться на

http://nwos.sourceforge.net/, когда я настрою его. Я планирую разместить копии моих рукописных записей, некоторые рисунки и подробности в следующие несколько дней.

Как я уже говорил во вступлении, может быть ничего из этого и не будет работать. Или, может быть, все получится настолько отвратительно медленным, что будет невозможно использовать. С другой стороны, с учетом того, что сегодня процессоры работают на частотах выше 3ГГц, я не думаю, что это будет проблемой. На самом деле, когда я провожу баланс чековой книжки, процессор, пожалуй, не слишком загружен. Я знаю, что иногда (например, при вращении фотографий) вы хотите иметь всю доступную скорость. Но, в общем случае, я с удовольствием отдам некоторое количество лошадок под капотом за простоту в использовании и надежность.

В конце концов, все это на самом деле тот же зуд, который не дает покоя. Так как я работаю с компьютерами целый день, практически каждый день, я постоянно вопрошаю себя "неужели это должно быть так сложно?" и "а нет ли лучшего способа?". Возможно его и нет, возможно, компьютеры, трудности и сложность идут рука об руку. Но мне просто надо убедиться самому. Я на самом деле никак не могу проститься с идеей того, что должен быть более простой путь и это постоянно меня преследует. Поэтому, мне необходимо доказать самому себе, что это либо возможно, либо невозможно.