Создавая новое поколение - часть 2
ОС
Николас Блэхфорд (Nicholas Blachford), Пятница, 22 Октябрь 2004, 00:36

ОС нового поколения
Итак, у нас есть какое-то новейшее оборудование, производительноть которого оставит даже лучшие настольные системы далеко позади. Но. Нам нужна ОС.

Единственное, она не должна быть похожа на Unix. Именно так, это должна быть **НЕ** Unix. Сколько нам нужно версий Unix? Если вы хотите настольную, берите OS X, хотите серверную - скорее всего подойдет любая версия Unix. Если вы хотите "" - берите Linux или BSD, в зависимости от того, какая Свобода вам больше нравится, если вам нужна Безопасность, держите OpenBSD. Существуют вариации Unix для практически любых задач, и это даже до того, как вы начнете просматривать дистрибутивы Linux. Неужели нам действительно нужен еще один Unix?

Я бы гораздо больше предпочел что-нибудь новое, что-то, что сможет использовать преимущества новых технологий и использовать лучшие возможности разных ОС. Проект Ruby OS (ROS) даже ведет онлайн-страницы, где они собирают лучшие идеи [RubyList].

Писать ОС с нуля сегодня - очень серьезное дело, это предприятие на много лет и для выполнения оно потребует множества квалифицированнх людей. И это всего лишь ОС, потом вам понадобятся драйверы, компилятор, документация для разработчиков и приложения. Альтернатива, которую почти все сегодня и используют, это создание ОС или на основе какой-нибудь существующей, или создание клона существующей ОС. В данном случае я хочу сделать и то, и другое. Я бы взял Haiku [Haiku] (ранее известную как OpenBeOS) и ответвил свою ОС от нее.

Существуют много вариантов операционных систем на выбор, но я бы остановился на Haiku, поскольку она исходит от BeOS, а значит будет отлично работать со множеством процессоров и множеством потоков, а также у нее нет множества наследственных проблем, а значит не надо будет делать обходных путей и не надо ничего крушить (в данном случае она всего лишь служит базой для новой ОС, а, значит, проблемы обратной совместимости могут быть полностью проигнорированы. Современный API и особое внимание к мультимедийным возможностям также будут плюсом, как и дружественная к коммерческим инициативам ультра-бесплатная лицензия MIT. Для любого коммерческого проекта, а этот проект больше похож именно на коммерческий по своей сути, вопросы лицензирования гораздо сложнее, чем могут себе предположить адвокаты и их рекомендации стоит выслушать очень внимательно [Licenses].

Что еще важно, так это тот факт, что Haiku, конечно же, будет по своей структуре схожа с BeOS, а из этого следует то, что она будет "микроядерного ". Однако, я это изменю, вынося из ядра все что можно, так чтобы оно стало настоящим микроядром [Micro]. С технической точки зрения, ядро фактически превратится в экзоядро [Kernel], однако система будет работать как микроядерная ОС.

Микроядерные системы используются в системах жесткого реального времени и системах

высокого риска (то есть там, где системная ошибка вполне может кого-нибудь убить).
Жесткое реальное время и пуленепробиваемя стабильность - хорошие характеристики и хорошая цель, однако настоящие микроядра практически никогда не используются в коммерческих настольных системах, где позже они часто модифицируются и становятся "микроядерного ", когда некоторые части переносятся в ядра (например, графика в Windows NT, сетевые возможности в BeOS-Dano).

В микроядерной системе части ядра ОС перемещаются в пространство пользователя, где работают как отдельные задачи. Такой подход имеет преимущества с точки зрения стабильности, безопасности и простоты, но дробление ядра на задачи означает, что процессору придется часто менять контексты задач, а это выливается в серьезные потери производительноти. Именно поэтому, чтобы уменьшить количество переключений, некоторые части обычно возвращают в ядро и "настоя" микроядра не используются в настрольных системах. Экзоядра удаляют из ядра еще больше, так что все его задачи сводятся к работе в качестве мультиплексора аппаратных средств компьютера. Я не слышал, чтобы какая-либо ОС общего назначения использовала такой подход сегодня.

Почему не Linux?

Итак, я выбираю новую ОС, которая на сегодня еще не закончена, потом я хочу выбросить ядро и использовать технологию, которая известна как снижающая производительноть... Но есть еще один способ для моего безумия, Linux опробован и оттестирован со многими драйверами, хорошей поддержкой приложений и он быстр. Но будем прагматичны: помимо того что он другой, есть ли еще какие-либо достойные причины не начинать с Linux?

Да, макроядерные (монолитные) ОС, каковой и является Linux, быстрее чем микроядерные ОС, потому что они чаще всего проектируются для и используются на однопроцессорны системах. И полагаю, что большинство экспериментов с микроядрами также проводится на однопроцессорны машинах.

Это важно, поскольку, как я пояснил в первой части, наша новая система не будет однопроцессорно, а будет основана на той идее, что аппаратно у нас будет как минимум два процессора общего назначения, и это число будет возрастать в будущем. А значит ОС должна сполна использовать этот факт.

Разбивая компоненты ядра на задачи пользовательскоо уровня так, как это делает микроядро, мы можем запускать их параллельно на нескольких ядрах ЦП, отдельные компоненты могут работать над своими задачами и им не нужно специально поддерживать многопроцессорнсть.

Запуск макроядерных ОС на нескольких ЦП добавляет сложности, многие и без того сложные части приходится заставлять работать одновременно. При этом макроядро использует единое адресное пространство и не пользуется преимуществами защищенной памяти: игрушечное приложение вроде Xeyes имеет в едином адресном пространстве работающие вместе защиту памяти, сетевой стек, файловую систему и все наиболее критические части.

Разделение функциональност между отдельными частями делает компоненты проще, а это значит, что в них будет меньше ошибок и их будет проще исправить, если они найдутся.

Такая структура также уменьшает вероятность ошибок в одной части, которые приводят к обрушению другой, так как все разделено на части в своих защищенных областях памяти. Надежность заложена в архитектуре, а не просто в коде.

Это не к тому, чтобы сказать, что Linux ненадежен - мой собственный опыт показывает, что Linux очень высоконадежная система. Но это к тому, что Linux может пострадать от того же самого синдрома, что и Windows, когда плохо написанный драйвер приводит к нестабильности системы. В системе, которую предлагаю я, плохо написанный драйвер не сможет обрушить другие компоненты системы (в реальности это может быть все-таки вызовет проблемы, но для этого надо чтобы он был написан совсем плохо).
[newpage]

Не приведет ли это к потерям производительноти?

Без сомнения, на однопроцессорны системах мы увидим отрицательное воздействие на производительноть. Однако, как я пояснил в первой части, аппаратура для этой системы основывается на многоядерном процессоре, а это все меняет.

Для того, чтобы понять что произойдет, нам сперва следует разобраться в том, что конкретно ударяет по производительноти микроядра в первом случае.

Переключение контекста происходит тогда, когда одна задача должна остановиться и дать другой возможность выполняться. Переключение контекста влечет за собой сохранение "состояния" процессора в оперативной памяти (содержимое регистров данных и состояния), а эта операция может занимать десятки тысяч тактов процессора. С микроядром такое происходит чаще, так как функциональност ядра разбита на различные задачи и, чтобы работать, необходимо переключаться между ними.

Это является проблемой, так как выполнение переключения требует времени, в течение которого процессор не может делать никакой полезной работы. Очевидно, что если их тысячи за секунду, то производительноть ухудшается. Еще более важно то, что переключение контекста может вызвать частичное вытеснение кэша, а это имеет сильный негативный эффект на дальнейшую производительноть, больший чем само переключение контекста.

Макроядерный подход не страдает от таких проблем с производительнотью, так как все в ядре и нет необходимости в переключении контекста, когда поток команд переходит с одной внутренней части ядра на другую.

Уже говорилось, что хорошо спроектированно микроядро не обязательно должно быть медленным, в какой-то мере можно избежать потерь производительноти на переключение контекстов, собирая сообщения вместе и передавая их вместе (асинхронно), это уменьшает потребность в переключении контекстов. Однако, большинство исследований микроядер производилось в Unix, и, исходя из синхронной природы API Unix [async], асинхронный обмен сообщениями не использовался, что приводило к бОльшим переключениям контекста и, из-за этого, понижению производительноти. Поэтому репутация микроядер, как медленных, как минимум частично незаслужена.

И не забывайте, что система основана на экзоядре. Традиционное микроядро пропускает сообщения через ядро к их получателям. Экзоядро не связывается с самим сообщением, оно просто говорит получателю, что есть сообщение и оставляет его разбираться с ним. Это

уменьшает накладные расходы на передачу сообщений.

BeOS использовала технику асинхронных сообщенией и это, безусловно, очень быстрая ОС. Однако, сетевой стэк, будучи вне ядра, зарекомендовал себя как тяжкая потеря производительноти и позже был перемещен внутрь, что однозначно повысило производительноть при работе с сетью. Ве никогда не выпускала таких коммерческих релизов, но это вошло в Zeta [Zeta] (единственный легальный способ получить полноценную BeOS сегодня).

Использование нескольких ядер

Разница с ситуацией, когда у нас несколько ядер процессоров, в том, что разделенные компоненты ядра смогут работать одновременно на разных ядрах, поэтому не придется так часто переключать контексты. По прежнему будет необходима пересылка сообщений между ядрами, но это не даст таких накладных расходов, как переключение контекстов и не будет плохо влиять на эффективность кэша.

Использование множественных ядер совместно с асинхронной передачей сообщений приведет к тому, что наша экзо/микроядерная ОС сможет превзойти синхронную макроядерную ОС. Каждая передача сообщения или вызов функции будет требовать времени, и это время будет фиксировано. Асинхронная передача сообщений позволит передавать бОльшие объемы данных за один раз, а это уменьшит количество проходящих сообщений по сравнению с системой, использующей синхронные вызовы API.

Та самая техника, которая ухудшает производительноть на одном ядре процессора для микроядер и повышает ее для макроядер, может иметь совершенно противоположныйэффект на многоядерном процессоре, представляя микроядра как более высокопроизводиельную архитектуру ОС. Это не случится мгновенно, но эффект будет тем яснее, чем большее количество ядер будет использоваться и разные части ОС смогут исполняться на своих ядрах.

Возможно вы возразите, что при таком распылении, приложение, работающее на нескольких ядрах заставит сменяться компоненты ОС и тем самым уменьшать производительноть. Конечно, это риск, но все тяжелые вычислительные задачи скорее всего будут работать на процессорах Cell, которые не будут работать над ОС. Помните, что это настольная машина, так что ядра процессоров скорее всего будут просто сидеть ничего не делая большую часть времени. Многие любят обсуждать относительные показатели производительноти ОС или аппаратуры, но очень немногие действительно используют эту производительноть.

Следует заметить, что ядро Linux 2.6 включает в себя асинхронный ввод/вывод. Асинхронный обмен сообщениями был включен и в ОС, основанную на FreeBSD, проектом DragonFly BSD [Dragon].

И все же не Linux?

Конечно, Linux (или *BSD) имеет свои преимущества, но похоже, что микроядерный подход может обеспечить не только все непременные аттрибуты микроядерной архитектуры, но и будет иметь в запасе лучшую производительноть. Такой подход также лежит в русле общего принципа простоты, который я обозначил в первой части, и дает нам шанс исследовать

проектирование ОС с другой стороны и увидеть каковы результаты.

В то время как система будет большей частью работать как микроядерная ОС, тот факт, что в реальности она использует экзоядро даст приложениям возможность практически полностью игнорировать ОС и работать с аппаратурой напрямую безопасно и разделенно. Прямой доступ к аппаратуре - не очень одобряемый подход, но он может быть полезен для приложений, использующих FPGA, и имеет потенциал для создания значительных ускорений в работе приложений [Exo]. Он также позволяет еще кое-что, что будет еще более полезно...
[newpage]

Совместимость со старыми системами

Только то, что вы создаете новую систему не означает, что вы должны обходиться без полезных программ. В наши дни есть несколько интересных способов заставить хорошие приложения работать даже на совершенно новой системе.

Главная проблема любой новой платформы - недостаток приложений. Конечно, существуют эмуляторы, но они сложны и, как правило, работают не так хорошо, как родная среда, а зачастую и гораздо хуже.

В наши дни вы можете взять практически любые необходимые приложения для Linux или других Unix-подобных операционных систем. "Открытые " означает, что один из способов достижения совместимости - включение всей ОС, все что для этого необходимо, это удостовериться, что она работает на вашей аппаратуре.

В то же время, двойная загрузка это неприятность и, в конце концов, вы захотите запускать приложения из единой среды наряду с приложениями для вашей новой платформы.

Вы можете сделать что-нибудь вроде слоя виртуализации, например, MacOnLinux позволяет вам запускать OS X поверх вариаций Linux для PowerPC, но виртуализация всегда ударяет по производительноти и приложения по прежнему работают в своих средах.

Но есть и другой путь. Комбинация специализированого высокопроизводиельного виртуализационнго слоя и существующей технологией X Windows.

Виртуализационнй слой называется Xen [Xen], он позволяет нескольким операционным системам работать на одной аппаратуре с практически 100% производительнотю [XenPerf], разделяя аппаратуру между ОС. Использование X Windows позволит использовать дисплей вне основной ОС.

Итак, включая Unix или похожую ОС и запуская ее параллельно с основной, мы получаем полный набор современных приложений без необходимости когда-либо покидать основную ОС. Возможно это будет кошмар в плане удобства использования, но это можно деликатно поправить.

Основная ОС также сможет воспользоваться подсистемами вторичной ОС. Например, если основная ОС не поддерживает всех USB устройств (многие USB устройства не поддерживают стандарты и поэтому нуждаются в специальной поддержке), то она может пользоваться поддержкой USB во вторичной ОС. Это хак, но полезный и законный для новой ОС, которая

поначалу будет ограничена в возможностях.

Включая Xen в качестве технологии для нашей новой ОС мы можем получить всю производительноть в основной системе и около 100% во вторичной, предложенная мной микроядерная / экзоядерная архитектура для новой ОС это позволяет, сам Xen это экзоядро и уже есть драйвера, которые позволяют запускать на нем Linux.

Так что, в конце концов, получается, что мы будем использовать Linux (или BSD)...

Что насчет драйверов?

Преимущество интегрированныхсистем (когда и аппаратура, и программная часть производятся одной компанией) в том, что основные драйвера сравнительно легко получить от самих компаний, так как вы покупаете их продукты. Но как вам скажет любой, кто занимался альтернативнымиОС, проблема возникает, когда вы захотите поддерживать продукты, которые вы не производите. Если у вас есть документация это хорошо, но если вы не можете заплатить за эти устройства, создание драйверов для любой новой платформы будет непростой задачей.

Начинать с чистого листа всегда дорого, но это также дает множество преимуществ. Используя микроядерный подход с драйверами можно будет работать по-разному, давая ОС возможность инсталляции драйверов по принципу drag&drop в работающей системе (а-ля BeOS). Также станет возможным перезапусскать даже самые критические части на живой системе, весьма полезно, если вы работаете над чем-либо, что еще не успели сохранить.

Вывод

Итак, мы имеем экзоядрифицированную (exokernelified, а вот попробуйте скормить это проверяльщику грамматики) версию Haiku в качестве базовой ОС, но система также может нормально работать с существующими Unix приложениями в одной среде. Сервисы ОС будут обеспечиваться высокоуровневым "kits" из Haiku, так что у нас будут те же возможности, что и в BeOS, но это не все. Если мы говорим о новой системе, мы можем пойти дальше и сделать их лучше.

В третьей части я опишу, что я хотел бы сделать с безопасностью, файловой системой и управлением файлами.

Ссылки / Подробная информация

[RubyList] Список лучших предложений для Ruby OS.

[Haiku] Haiku (ранее OpenBeOS).

[Licenses] Дискуссия об использовании ПО с открытыми исходниками в коммерческих проектах.

[Micro] Страничка о микроядрах.

[Kernel] Введение в различные виды ядер.

[async] Полезная информация, Unix использует синхронный обмен сообщениями, что ставит микроядра в невыгодное положение.

[Zeta] Zeta - продолжатель дела BeOS от Yellowtab.

[Dragon] Есть попытки добавить асинхронный обмен сообщениями в Unix, одна из целей проекта DragonflyBSD как раз и является в том, чтобы сделать это для ядра FreeBSD. DragonFlyBSD.

[Exo] Подробная информация об экзоядрах: документы, слайдшоу.

[Xen] Xen разделяет аппаратуру между несколькими ОС. Проект Xen.

[XenPerf] Производительноть работы с Xen очень близка к нормальной производительноти систем.

Об авторе:
Николас Блэхфорд (Nicholas Blachford) - 33-х летний эмигрант из Британии, ныне живущий в Париже, но не говорящий по-французски (еще). Интересуется различными причудливыми темами (Аппаратное обеспечение, программное обеспечение, фотография) и всякими другими вещами, особенно теми, в которых используются новые технологии. На сегодняшний день безработный.