

Анатомия Linux-архитектур реального времени

От "мягких" к "жестким" системам реального времени

М. Тим Джонс (mtj@mtjones.com)

инженер-консультант

Emulex

30.10.2008

Дело не в том, что Linux® не является достаточно быстрым или эффективным, однако в некоторых случаях одной скорости оказывается недостаточно. Вместо этого необходимо обеспечить возможность детерминированно соблюдать требования планировщика по времени в пределах заданной ошибки. В этой статье мы рассмотрим различные варианты обеспечения требований реального времени в Linux и способы, с помощью которых обеспечивается выполнение этих требований — начиная с ранних архитектур, которые основывались на решениях по виртуализации, и заканчивая теми возможностями, которые сегодня имеются в стандартном ядре 2.6.

В данной статье исследуются некоторые из архитектур Linux, которые поддерживают характеристики реального времени, а также обсуждается, что на самом деле означает *архитектура реального времени*. Существует несколько решений, направленных на добавление в Linux поддержки реального времени. Здесь рассматриваются подход на основе тонкого ядра (или микро-ядра), подход на основе нано-ядра и подход на основе ядра ресурсов. В заключение описываются те возможности реального времени, которые имеются в стандартном ядре 2.6, а также описываются способы их реализации и использования.

Определение режимов реального времени и требования к ним

Основой для обсуждения архитектур реального времени на основе Linux будет служить приводимое ниже определение режима *реального времени*. Данное определение предложено Дональдом Джиллесом (Donald Gillies) и приводится в Realtime Computing FAQ (ссылка приводится в разделе [Resources](#)):

Системой реального времени является такая система, в которой успех вычислений зависит не только от их логической правильности, но также и от времени, когда готовы результаты вычислений. Если не удастся обеспечить соответствие временным ограничениям, то считается, что произошла системная ошибка.

Другие статьи из серии *Анатомия...*, автором которых является Тим, на сайте developerWorks

- [Anatomy of the Linux SCSI subsystem](#)
- [Anatomy of the Linux file system](#)
- [Anatomy of the Linux networking stack](#)
- [Anatomy of the Linux kernel](#)
- [Anatomy of the Linux slab allocator](#)
- [Anatomy of Linux synchronization methods](#)
- [Все статьи Тима из серии *Anatomy of...* articles](#)
- [Все статьи Тима на сайте developerWorks](#)

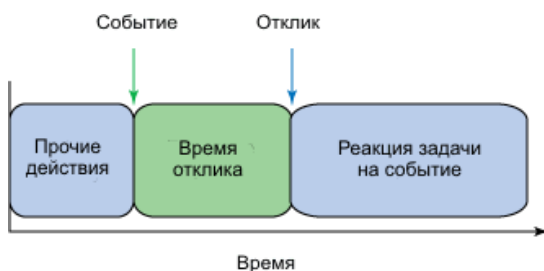
Другими словами, чтобы гарантировать временное поведение, система должна вести себя детерминированным образом в широком диапазоне нагрузок (от минимальной до самой неблагоприятной, максимальной нагрузки). Заметим, что в определении ничего не говорится о производительности, так как режим реального времени не имеет к этому никакого отношения: здесь речь идет только о предсказуемости. Например, при использовании быстрого современного процессора Linux способен обеспечить типичное время отклика на прерывание 20 мкс, но случайным образом это время может значительно увеличиваться. В этом заключается основная проблема: дело не в том, что Linux не является достаточно быстрым или эффективным, проблема заключается в отсутствии предсказуемого поведения.

Приводимые далее примеры показывают сущность данной проблемы. На Рисунке 1 показаны результаты измерения времени отклика на прерывание. При поступлении прерывания (*события*) работа центрального процессора прерывается и он переходит к обработке прерывания. Некоторое время затрачивается на то, чтобы определить, какое именно событие произошло, затем, проделав некоторую работу, планировщик переключает необходимую задачу на работу с данным событием (*переключение контекста* - *context switch*). Время, которое проходит от поступления сигнала прерывания до создания необходимой задачи (предполагается, что эта задача будет иметь наиболее высокий приоритет) определяется как *время отклика*. Для систем реального времени это время отклика должно быть детерминированным и соответствовать заранее известному наиболее продолжительному времени ожидания.

Переключение контекста

Процесс создания планировщиком новой задачи на основе прерывания подразумевает *переключение контекста*. Это означает сохранение текущего (на момент поступления прерывания) состояния центрального процессора и затем восстановление состояния данной задачи. Непосредственно переключение контекста является задачей операционной системы, но оно зависит от архитектуры используемого процессора.

Рисунок 1. Время задержки и время отклика на прерывание



Полезным примером такого процесса является срабатывание стандартной подушки безопасности в современных автомобилях. В какой-то момент датчик сообщает о столкновении и отправляет прерывание центральному процессору, и тогда операционная система должна быстро выделить задачу, которая управляет работой подушки безопасности, причем ее выполнению не должны мешать другие задачи, которые не связаны с требованиями реального времени. Если подушка безопасности сработает на секунду позднее, чем это требуется, то лучше совсем отказаться от ее использования.

Помимо детерминированности при обработке прерываний, для обработки в режиме реального времени важно обеспечить планирование задач с поддержкой периодических интервалов времени. Рассмотрим рисунок 2, где показано планирование для периодически вызываемых задач. Такое периодическое измерение и обработка необходимы для многих систем управления. Для обеспечения стабильной работы системы необходимо, чтобы определенная задача выполнялась через некоторый период времени (p). Например, рассмотрим работу антиблокировочной системы (АБС) в автомобиле. Эта система опрашивает скорость каждого колеса автомобиля и контролирует давление в тормозном устройстве каждого колеса (для предотвращения его блокировки) с частотой до 20 раз в секунду. Для обеспечения нормальной работы системы опрос датчика и управление должно выполняться через периодические интервалы времени. Это означает, что через определенный период времени выполнение других задач прерывается и выполняется задача, связанная с работой АБС.

Рисунок 2. Планирование периодических задач



Жесткий режим реального времени и мягкий режим реального времени

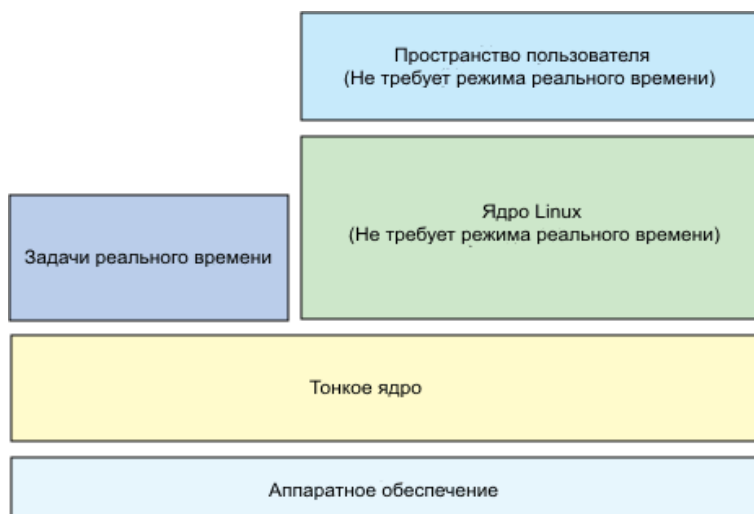
Операционная система, которая способна поддерживать необходимые временные требования к задачам реального времени (даже при наиболее неблагоприятных нагрузках на процессор) называется системой с поддержкой *жесткого режима реального времени*. Однако жесткая поддержка реального времени не всегда является необходимой. Если операционная система способна обеспечивать временные требования в среднем, то она называется системой с поддержкой *мягкого режима реального времени*. Жесткие системы реального времени необходимы там, где неспособность выполнить временные требования может приводить к катастрофическим результатам (например, задержка в включении подушки безопасности или ошибка в управлении тормозным давлением, в результате чего колесо скользит непозволительно долго). Мягкие системы реального времени могут не удовлетворять каким-то временным ограничениям, и это не приводит к системной ошибке (например, пропуск кадра при воспроизведении видео).

Теперь, когда мы имеем представление о требованиях к режиму реального времени, обратимся к архитектурам Linux реального времени и рассмотрим, какой уровень поддержки реального времени они способны обеспечить и каким образом.

Подход на основе тонкого ядра

Подход на основе тонкого ядра (или микроядра) использует второе ядро для абстрагирования интерфейса между аппаратными устройствами и ядром Linux (см. рисунок 3). То ядро Linux, которое не работает в режиме реального времени, выполняется в фоновом режиме как задача с низким уровнем приоритета для тонкого ядра и на нем выполняются все задачи, не относящиеся к реальному времени. Задачи реального времени выполняются непосредственно в тонком ядре.

Рисунок 3. Подход на основе тонкого ядра к жесткому режиму реального времени



Основным назначением тонкого ядра (кроме выполнения задач реального времени) является управление прерываниями. Тонкое ядро перехватывает прерывания, благодаря этому гарантируется, что работа тонкого ядра не будет прервана другим ядром, которое не выполняет задачи реального времени. Это позволяет тонкому ядру обеспечить жесткую поддержку режима реального времени.

Хотя подход на основе тонкого ядра имеет свои преимущества (жесткая поддержка реального времени вместе со стандартным ядром Linux), он имеет и недостатки. Обычные задачи и задачи реального времени не зависят друг от друга, это может значительно усложнить отладку. Кроме этого, обычные задачи не обеспечивают полную поддержку платформы Linux (именно по этой причине выполнение ядра носит название *тонкого*).

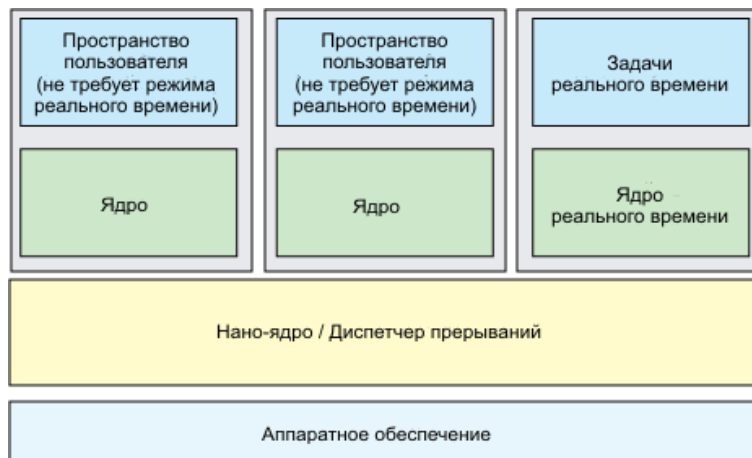
Примерами реализации такого подхода являются RTLinux (теперь закрытая система, принадлежит Wind River Systems), Real-Time Application Interface (RTAI) и Xenomai.

Подход на основе наноядра

В то время как тонкое ядро использует минимальное ядро, которое включает управление задачами, подход на основе наноядра движется далее в этом направлении, еще более

минимизируя размеры ядра. Такое ядро уже является в меньшей степени ядром, а в большей степени уровнем абстракции аппаратного обеспечения (HAL). Нано-ядро обеспечивает возможность совместного использования аппаратных ресурсов для нескольких операционных систем, которые выполняются на более высоком уровне (см. рисунок 4). Так как нано-ядро позволяет абстрагировать аппаратное обеспечение, то оно способно управлять приоритетами операционных систем, исполняемых на более высоком уровне, и благодаря этому, обеспечивать жесткую поддержку режима реального времени.

Рисунок 4. Нано-ядро для поддержки абстрагирования аппаратного обеспечения



Отметим схожесть между данным подходом и виртуализацией, которая обеспечивает возможность выполнения нескольких операционных систем. В нашем случае нано-ядро позволяет абстрагировать аппаратные устройства от обычного ядра и ядра реального времени. Это аналогично тому, как гипервизоры абстрагируют голое аппаратное обеспечение от выполняемых операционных систем. Более подробную информацию можно найти в разделе [Ресурсы](#).

Примером подхода на основе нано-ядра является Adaptive Domain Environment for Operating Systems (ADEOS). ADEOS обеспечивает возможности одновременного выполнения операционных систем. При возникновении определенного аппаратного события, ADEOS по очереди обращается ко всем операционным системам и определяет, кто из них будет осуществлять обработку события.

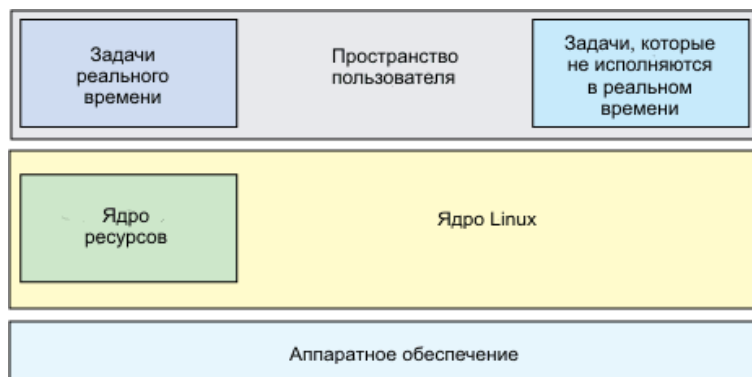
Подход на основе ядра ресурсов

Еще один вариант архитектуры реального времени - подход на основе ядра ресурсов. В этом подходе к ядру добавляются модули, которые обеспечивают резервирование ресурсов различного типа. Такое резервирование обеспечивает доступ к системным ресурсам с временным разделением (центральный процессор, сеть или доступ к диску). Эти ресурсы имеют несколько параметров резервирования, таких как период повторного доступа, необходимое время обработки (то есть, период времени, необходимый для выполнения обработки), а также временные ограничения.

Ядро ресурсов предоставляет набор прикладных интерфейсов программирования (API) которые позволяют задачам запрашивать подобное резервирование (см. рисунок 5). Затем

ядро ресурсов объединяет эти запросы с целью, определяя график, который обеспечивает гарантированный доступ с учетом временных ограничений, действующих для конкретных задач (или же возвращается ошибка, если невозможно гарантировать такой доступ). Используя алгоритм планирования, такой как Earliest-Deadline-First (EDF), ядро затем может производить динамическое планирование нагрузки.

Рисунок 5. Подход на основе ядра ресурсов обеспечивает резервирование ресурсов

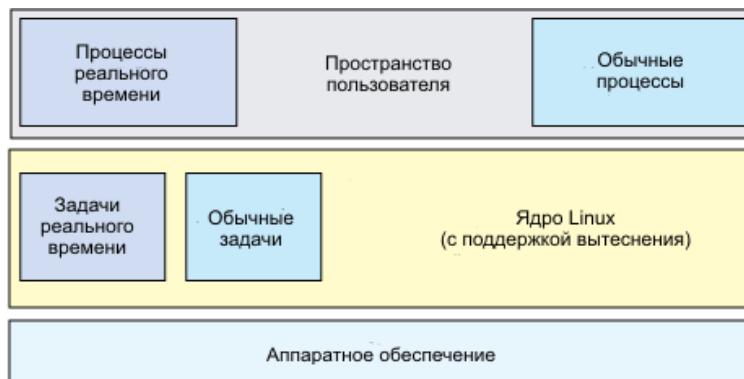


Одним из примеров реализации подхода на основе ядра ресурсов является Linux/RK от компании CMU, который интегрирует переносимое ядро ресурсов в Linux в качестве загружаемого модуля. Сегодня данный подход воплощен в коммерческой системе TimeSys Linux/RT.

Поддержка реального времени в стандартном ядре 2.6

Хотя описанные выше решения и представляю интерес с точки зрения архитектуры, все они работают "вокруг" ядра. Возникает вопрос - а возможно ли внести какие-то изменения в стандартное ядро Linux, чтобы обеспечить поддержку режима реального времени?

Сегодня от ядра 2.6 можно получить поддержку мягкого режима реального времени на основе простой конфигурации, которая обеспечивает для ядра полную вытесняемость (см. рисунок 6). В стандартном ядре 2.6 Linux, когда пользовательский процесс выполняет обращение к ядру (при помощи системного вызова), этот процесс не может быть вытеснен. Это означает, что если процесс с низким приоритетом делает системный вызов, то процесс с высоким приоритетом должен ждать пока системный вызов не будет завершен - только после этого он сможет получить доступ к центральному процессору. Новая опция конфигурирования `CONFIG_PREEMPT` изменяет поведение ядра и позволяет вытеснять процесс, если имеется задание с более высоким приоритетом (даже если этот процесс ожидает завершения системного вызова).

Рисунок 6. Стандартное ядро 2.6 Linux с возможностью вытеснения

Однако использование данной опции связано с определенными компромиссами. Хотя эта опция обеспечивает поддержку мягкого режима реального времени и обеспечивает более гладкое выполнение операционной системы, все это имеет определенную цену. Добавленные при помощи `CONFIG_PREEMPT` функции приводят к небольшому снижению пропускной способности и небольшому уменьшению производительности ядра. Данная опция полезна для персональных компьютеров и встраиваемых систем, однако может не являться правильным выбором для всех случаев (например, для серверов).

Новый планировщик O(1)

Новый планировщик O(1), который входит в состав ядра 2.6, обеспечивает значительные преимущества по производительности, даже при наличии большого количества задач. Этот новый планировщик способен выполняться в ограниченное время независимо от количества имеющихся задач. Дополнительную информацию по новому планировщику и принципах его работы можно получить в разделе [Ресурсы](#).

Другой полезной опцией в ядре 2.6 являются таймеры высокого разрешения. Эта новая опция позволяет таймерам работать с разрешением до 1 мкс (если это позволяет используемое аппаратное устройство), а также для повышения производительности реализует управление таймерами на основе красно-черных деревьев. Использование красно-черных деревьев позволяет активировать большое количество таймеров без ущерба для производительности работы подсистемы таймеров ($O(\log n)$).

Проделав небольшую дополнительную работу, при помощи патча `PREEMPT_RT` можно добавить поддержку жесткого режима реального времени. Чтобы добиться этого, патч `PREEMPT_RT` вносит несколько важных изменений. К ним относятся изменение некоторых используемых в ядре примитивов блокировки с целью сделать их полностью вытесняемыми, реализация наследования приоритета для мьютексов ядра и преобразование обработчиков прерываний в потоки ядра (также с целью обеспечить их полную вытесняемость).

Заключение

Linux не только является идеальной платформой для экспериментов и оценки алгоритмов реального времени - сегодня поддержка реального времени реализована в стандартном ядре 2.6. Вы можете также обеспечить поддержку мягкого режима реального времени в

стандартном ядре или же, приложив немного дополнительных усилий (патч для ядра), вы сможете создавать приложения с поддержкой жесткого режима реального времени.

В этой статье приводится краткий обзор методов реализации вычислений в реальном времени для ядра операционной системы Linux. В многочисленных ранних попытках для выделения из стандартного ядра задач реального времени использовался подход на основе тонкого ядра. Затем стали использоваться подходы на основе нано-ядра, которые действовали во многом аналогично гипервизорам, использующимся сегодня в решениях по виртуализации. Наконец, современное ядро Linux обеспечивает собственные средства для поддержки как мягкого, так и жесткого режима реального времени.

В данной статье приведен лишь самый общий обзор методов поддержки реального времени в Linux. В разделе [Ресурсы](#) содержится дополнительная информация о том, где можно получить материалы и методы по организации режимов реального времени.

Ресурсы

Научиться

- Оригинал статьи: "[Anatomy of real-time Linux architectures](#)" (EN).
- В статье Тима "[Virtual Linux](#)" (EN) (developerWorks, декабрь 2006 г.) описывается, как подходы на основе тонкого ядра или наноядра опираются на архитектуры виртуализации, которые сегодня завоевали большую популярность и широко используются.
- В статье Тима "[Inside the Linux scheduler](#)" (developerWorks, июнь 2006 г.) можно получить дополнительную информацию о планировщике O(1), входящем в состав ядра 2.6.
- Прочтите [все статьи Тима из серии Anatomy of...](#) на сайте developerWorks.(EN)
- Прочтите [all все статьи Тима по операционной системе Linux](#) на сайте developerWorks.(EN)
- Отличным местом для того, чтобы начать изучение вычислений в реальном времени, является [Realtime Computing FAQ](#). Здесь описываются основы Portable Operating System Interface (POSIX), операционных систем реального времени и информация по анализу в реальном масштабе времени. (EN)
- В интересной обзорной статье А. Гамбье "[Real-time Control Systems](#)" (EN) (PDF) рассматривается проектирование систем управления реального времени и обсуждаются различные алгоритмы планирования.(EN)
- Материалы 6-го семинара [6th Real-time Linux Workshop](#) (проходил в ноябре 2007 г.) содержат статьи и ресурсы, которые будут полезны при изучении самых современных способов поддержки реального времени в операционной системе Linux. (EN)
- В ранних версиях [RTLinux](#), [RTAI](#) и [Xenomai](#) использовался подход на основе тонкого ядра. Хотя каждое из этих решений слегка отличается от других, общая ценность такого подхода может считаться доказанной.(EN)
- [ADEOS](#) является уровнем абстрагирования аппаратного обеспечения (HAL), или наноядром, которое обеспечивает возможности реального времени для ядра Linux. Также это решение используется для кластеризации и отладки ядра в SMP-системах (симметричные мультипроцессорные системы).(EN)
- В статье "[Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior](#)" (EN) описывается компонент, который обеспечивает гарантированный доступ к ресурсам с временным разделением (таким как центральный процессор, сеть или жесткий диск) при помощи набора функций (API) резервирования.
- [Красно-черное дерево](#) - это дерево двоичного поиска с автоматическим балансированием. Хотя алгоритм является достаточно сложным, на практике он очень эффективен и выполняется за время $O(\log n)$.(EN)
- Патч [PREEMPT_RT](#) обеспечивает поддержку жесткого режима реального времени для стандартного ядра 2.6 Linux. На этих Web-страницах содержится подробная информация по установке и использованию конфигурации RT_PREEMPT, а также полезный раздел [вопросов и ответов](#).(EN)
- В сценарии [инверсии приоритетов](#) задача с низким приоритетом удерживает ресурс, который необходим задаче с более высоким приоритетом. (Такая проблема возникла

в марсианском аппарате Pathfinder, работавшем под управлением ОС VxWorks от Wind River.) Решением этой проблемы является [наследование приоритета \(priority inheritance\)](#), в результате чего повышается приоритет задачи с низким приоритетом, что разрешает ей выполниться и освободить ресурс. (EN)

- Компания Novell недавно объявила о выходе версии 1.0 [SUSE Linux Enterprise Real-Time \(SLERT\)](#). Этот пакет содержит обновления для ядра и поддержку в режиме реального времени POSIX для приложений, требующих вычислений в реальном времени. К двум наиболее важным особенностям относятся защита центрального процессора (CPU shielding) и присваивание приоритетов в процессе выполнения. (EN)
- [Xenomai/SOLO](#) объявили об отходе от подхода на основе дополнительного ядра и намерении развиваться в направлении более тесной интеграции с уровнем ядра Linux. Такой подход обеспечивает поддержку традиционного программирования POSIX (для пользовательского пространства). (EN)
- В разделе [Linux](#) сайта developerWorks содержатся дополнительные материалы для разработчиков Linux. Также просмотрите наши [наиболее популярные статьи и учебные пособия](#). (EN)
- Обратитесь [к советам по Linux](#) и [учебным пособиям для Linux](#), которые находятся на сайте developerWorks.
- Оставайтесь в курсе новостей, посещая [раздел технических новостей и Web-трансляций developerWorks](#). (EN)

Получить продукты и технологии

- Используйте [ознакомительные версии программного обеспечения IBM](#), которые можно загрузить непосредственно с сайта developerWorks, в своем следующем проекте для Linux. (EN)

Обсудить

- [Участвуйте в обсуждении данного материала на форуме.](#)
- Примите участие в деятельности сообщества [developerWorks](#) и используйте блоги, форумы, трансляции и обсуждения важных вопросов [в новых разделах developerWorks](#). (EN)

Об авторе

М. Тим Джонс

М. Тим Джонс (M. Tim Jones) является архитектором встраиваемого программного обеспечения и автором работ: *Программирование Приложений под GNU/Linux*, *Программирование AI-приложений* и *Использование BSD-сокетов в различных языках программирования*. Он имеет опыт разработки процессоров для геостационарных космических летательных аппаратов, а также разработки архитектуры встраиваемых систем и сетевых протоколов. Сейчас Тим работает инженером-консультантом в корпорации Эмулекс (Emulex Corp.) в г.Лонгмонт, Колорадо.

© Copyright IBM Corporation 2008

(www.ibm.com/legal/copytrade.shtml)

[Торговые марки](#)

(www.ibm.com/developerworks/ru/ibm/trademarks/)