ОСОБЕННОСТИ РАЗЛИЧНЫХ АРХИТЕКТУР ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX В ЗАДАЧАХ ИНТЕРАКТИВНОГО МОДЕЛИРОВАНИЯ УСТРОЙСТВ В РЕАЛЬНОМ ВРЕМЕНИ.

Рубцов М.О.
ГОУВПО «Мордовский государственный университет им. Н. П. Огарева», г. Саранск
E-mail: rumx@mail.ru

Аннотация. Проведен обзор различных вариантов архитектуры операционной системы реального времени на основе Linux. Показаны принципиальные требования к такой архитектуре. Рассмотрены микроядерный, наноядерный подход к архитектуре системы, архитектура на основе резервирования ядра, а так же механизмы организации реального времени в стандартном 2.6 ядре Linux. Показано удобство использования последнего варианта архитектуры для задачи интерактивного моделирования устройств в реальном времени.
Ключевые слова: система реального времени, Linux, архитектура, интерактивное моделирование.

Особенности ядра Linux при работе с задачами реального времени

Кроме зарекомендовавшей себя надежности и производительности системы на основе Linux привлекают возможностью получить стабильную систему реального времени с минимальными затратами, в том числе для задачи интерактивного моделирования устройств в реальном времени. В связи с этим требуется рассмотреть некоторые из специальных системных архитектур, которые поддерживают характеристики реального времени, поскольку архитектуры Linux общего назначения для задач реального времени не подходят. Хотя Linux характеризуется как быстрая или эффективная система, в некоторых случаях недостаточно одной только «скорости» системного кода. В задачах реального времени требуется возможность гарантированно выполнять сроки запусков задач с заранее определенными допусками по времени.

В качестве основы для рассмотрения примем следующее определение системы реального времени: «Операционная система, в которой успешность работы любой программы зависит не только от её логической правильности, но и от времени, за которое она получила этот результат. Если система не может удовлетворить временным ограничениям, должен быть зафиксирован сбой в её работе» [1].

Основываясь на этом определении, можно вывести, что система должна быть предсказуема, чтобы гарантировать ее поведение во времени в условиях переменной нагрузки (от минимально возможной до критической). Это определение ничего не говорит о скорости, но оно касается прогнозируемости. Ядро Linux в дистрибутивах общего назначения на современном процессоре может обеспечивать среднее время отклика на прерывание равным 20 мкс, но в одном случае это время может стать значительно дольше. Эта фундаментальная проблема ядра Linux общего назначения не означает, что ядро не быстрое или малоэффективное, а просто означает, что оно плохо предсказуемо. В ряде приложений это недопустимо.

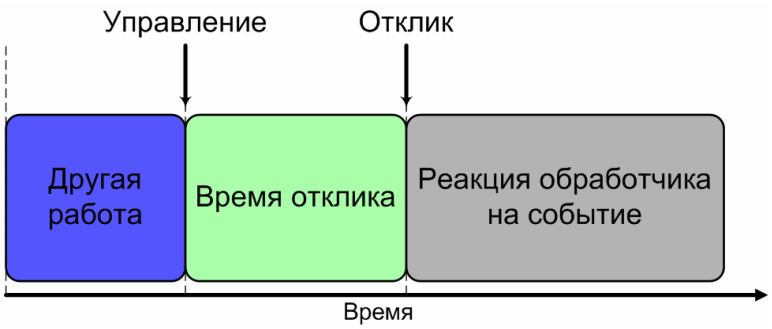

Рис. 1. Время отклика системы.

Для задачи моделирования устройств это означает плохо предсказуемый характер управления. Рис. 1 показывает измерение времени отклика такого виртуального устройства

на внешнее событие управления (например, прерывание). Процессору требуется выполнить некоторые действия, чтобы определить, какого типа событие произошло, и какая задача на это событие должна активироваться. После небольшой паузы требуемая задача активируется и заставляет моделируемое устройство изменить свое поведение. Время между наступлением события и активацией требуемой задачи (предполагается, что это наиболее приоритетная задача) называется временем отклика. Время отклика, таким образом, должно иметь заранее известные временные рамки.

В большинстве операционных систем, и в Linux в том числе, процесс переключения на новое задание, основанное на прерывании – это переключение контекста задачи. Таким образом, скорость переключения задачи зависит, в общем случае, и от операционной системы, и от нижележащей архитектуры процессора.

Кроме предсказуемости в процессе обработки прерываний также необходим системный планировщик задач, который поддерживает периодические интервалы. Моделирование интерактивных устройств требует периодической обработки управляющих сигналов, что возможно путем организации дополнительной задачи, которая запускается системой с заданным интервалом.

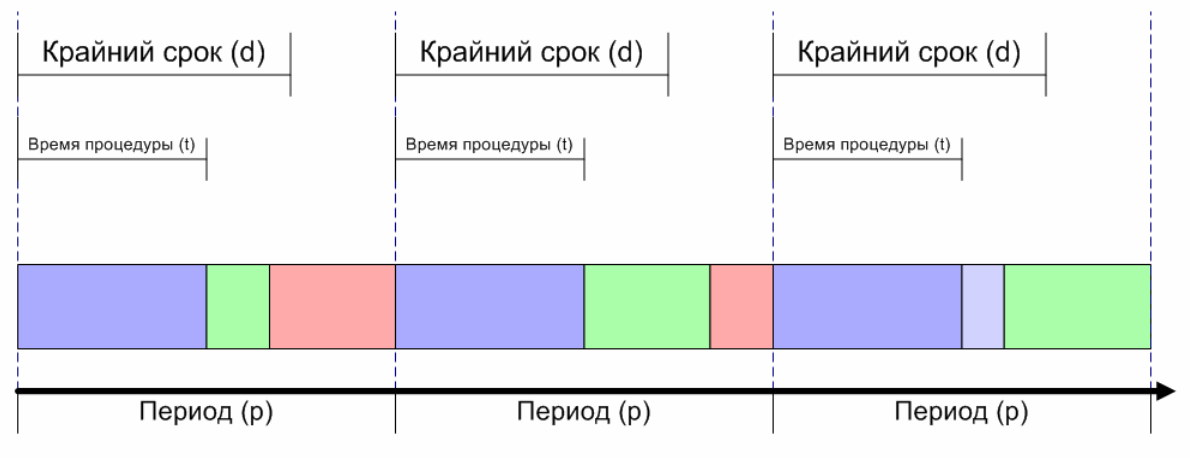

Рис. 2. Периодический запуск задачи реального времени

Рис. 2 условно изображает такой периодический запуск. Задача, выделенная синим цветом, изображает периодическую обработку и должна выполняться с периодом p. Время этой обработки фиксировано и равно t. Если t больше либо равно крайнему сроку d, по определению будет зафиксирован сбой задачи реального времени. Обычно задача укладывается в крайний срок, и система продолжает выполнение других задач (показаны другими цветами).

Операционная система, которая может поддерживать желаемые сроки задач реального времени (даже при экстремальных нагрузках) называется системой жесткого реального времени. Если операционная система может поддерживать нужные интервалы в среднем, то она называется системой мягкого реального времени.

Является ли необходимым использование операционной системы жесткого реального времени для моделирования управляемого устройства? Если речь идет о тестирующей системе, аварийной системе, и других системах, дающих результат, с определенной гарантией, то использование системы жесткого реального времени позволит эту гарантию дать. Поскольку для задачи интерактивного моделирования устройства интерес представляет как раз гарантированное моделирование, далее речь пойдет о системах жесткого реального времени. Рассмотрим некоторые известные архитектуры Linux реального времени [2], чтобы выбрать наиболее удобный вариант для задачи моделирования интерактивных устройств.

Архитектура «тонкое ядро»

Концепция «тонкое ядро» (или микроядро) использует второе ядро как интерфейс абстрагирования аппаратной части от ядра Linux (Рис. 3). Ядро Linux без реального времени запускается в фоновом режиме как наименее приоритетная задача микроядра, и содержит все задачи, выполняемые без требования реального времени. Задачи реального времени выполняются непосредственно на микроядре.

Первичная задача микроядра (кроме запуска задач реального времени) – это управление прерываниями. Тонкое ядро перехватывает прерывания для того, чтобы гарантировать, что задачи без требования реального времени не могут вытеснять задачи тонкого ядра.

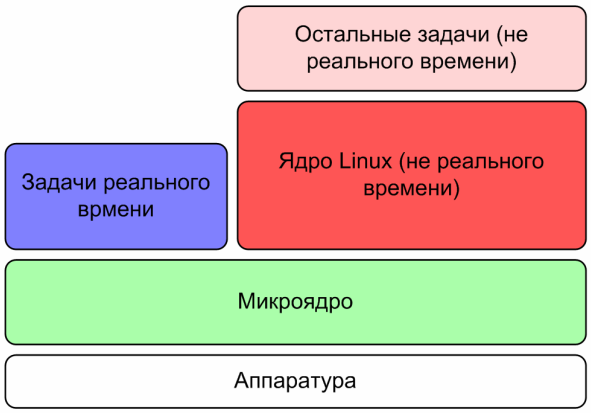

Рис. 3. Архитектура на основе принципа микроядра

Вслед за преимуществами микроядерного подхода (поддержку жесткого реального времени в стандартном ядре Linux) отметим его недостатки. Задачи реального времени и обычные задачи должны быть полностью независимые, что может сделать систему гораздо более сложной, особенно если требуется синхронизация между ними. В случае моделирования интерактивного устройства чаще всего требуется наличие интерфейса пользователя, визуально изображающее состояние устройства, а так же меняющего режимы его работы. Интерфейс пользователя реализуется в виде отдельной обычной задачи, и ее синхронизация с задачей реального времени в микроядерной архитектуре является нетривиальной.

Задачи без поддержки реального времени уже не имеют полной поддержки платформы Linux (поскольку их ядро не взаимодействует с аппаратной частью напрямую). Если моделирование интерактивного устройства осуществляет в реальном времени, то вывод результата, например, в виде переменных состояния, то драйвер устройства вывода тоже должен являться задачей микроядра, что большую сложность в организацию всей системы, особенно если исходный код драйвера недоступен.

Примеры концепции микроядра включают RTLinux (сейчас собственность Wind River Systems) [3], RTAI [4], Xenomai [5].

Архитектура «наноядро»

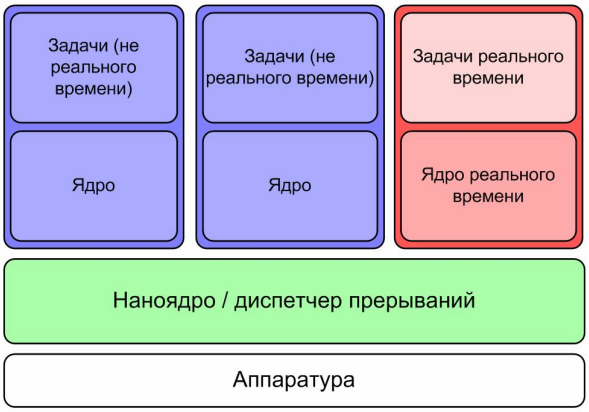

Рис 4. Архитектура системы с наноядром

Концепция микроядра основана на минимальном ядре, которое включает управление задачами, но концепция наноядра делает еще один шаг в сторону его уменьшения. Это даже не столько ядро, сколько уровень абстрагирования аппаратуры (Hardware Abstraction Level, HAL). Наноядро нужно для разделения аппаратных ресурсов между многими операционными системами на более высоком уровне (Рис. 4). Поскольку наноядро абстрагирует аппаратную часть, оно может осуществлять приоритезацию вышележащих операционных систем и таким образом, поддерживать реальное время (на примере проекта ADEOS) [6]. Можно заметить сходство между этим подходом и виртуализацией для запуска многих операционных систем. В этом случае наноядро абстрагирует железо от ядер реального времени и нереального времени. Так же действует гипервизор, который абстрагирует аппаратную часть от операционных систем - клиентов.

Однако, все названные сложности моделирования интерактивных устройств, возникающие в архитектуре на основе микроядра, в наноядерной концепции сохраняются.

Архитектура по принципу ядра-ресурса

Другая архитектура реального времени – концепция ядра-ресурса. Этот подход добавляет модуль к ядру, чтобы сделать резервирование ресурсов различного типа. Резервирование гарантирует доступ к ресурсам, разделяемым во времени (ЦП, сеть, или даже диск). Ресурсы имеют такие же параметры резервирования, как ранее рассмотренные параметры резервирования одного процессорного времени (период, требуемое время обработки, и крайний срок).

Ядро может объединить объединяет запросы на резервирование, чтобы определить план, обеспечивающий гарантированный доступ к ресурсам, а так же позадачные ограничения (или вернет задаче ошибку, если требуемое не может быть гарантированно). Используя планировщик, чаще всего EDF [7], ядро может управлять динамически меняющейся нагрузкой.

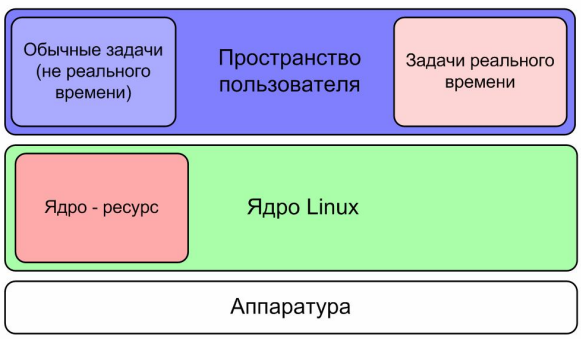

Рисунок 5. Архитектура на основе принципа резервирования

Один пример реализации ядра-ресурса – это CMU Linux/RK, который внедряет переносимое ядро-ресурс в Linux в качестве загружаемого модуля. Эта реализация включена в коммерческое предложение TimeSys Linux/RT.

В концепции ядра-ресурса облегчается задача синхронизации между процессами реального времени и остальными процессами. Как следствие, необходимость в переписывании драйвера устройства ввода-вывода отпадает, если этот драйвер сам по себе оперирует в ограниченном времени. Так же к достоинствам можно отнести снятие ограничений ядра Linux в доступе к аппаратуре системы.

Реальное время в стандартном ядре 2.6

Рассмотренные выше подходы по историческим причинам оперируют на периферии ядра. В ядре 2.6 появилась возможность получить производительность мягкого реального времени путем простого конфигурирования ядра, делающего ядро полностью вытесняемым. (Рис. 6). Когда пользовательская задача делает системный вызов в стандартном 2.6. ядре Linux, она не может быть вытеснена. Таким образом, если низкоприоритетная задача делает системный вызов, другая высокоприоритетная задача обязана ждать, пока этот вызов завершится. Новая опция в конфигурации CONFIG_PREEMPT меняет такое поведение ядра, позволяя вытеснять процессы, если более приоритетная работа стала доступна к выполнению (даже если вытесняемый процесс еще не завершил системный вызов).

Эта опция отключена в стандартной конфигурации, поскольку имеет «побочные эффекты». Хотя она включает производительность мягко реального времени и даже при большой загрузке позволяет настольным системам работать визуально более «плавно», за это приходится платить небольшим уменьшение производительности самого ядра. Причиной является добавление опцией CONFIG_PREEMPT дополнительных действий. Таким образом, эта опция полезна для настольных и внедряемых систем, но не рекомендуется для использования, например, в серверах.

Увеличить предсказуемость системы на основе ядра 2.6 помогает новый планировщик O(1). Он также очень выгоден по критерию производительности, особенно если много задач выполняется одновременно. Его основное достоинство - он может оперировать в ограниченном времени независимо от числа выполняемых задач [8].

Другая полезная опция в конфигурации ядра 2.6 существует для таймеров высокого разрешения. Эта новая опция позволяет таймерам оперировать с разрешением до 1 мкс (если это поддерживается оборудованием) и реализует эффективное управление временем с помощью черно-красного дерева [9]. Используя это дерево, можно заводить большое число одновременно активных таймеров без влияния на производительность подсистемы таймеров (Поскольку время поиска в черно-красном дереве всегда пропорционально log(n), где n – общее количество таймеров).

Дополнительная работа надо ядром 2.6 позволяет получить поддержку жесткого реального времени (патч PREEMPT_RT). Этот патч делает несколько модификаций чтобы заявить о поддержке жесткого реального времени. Изменения включают переписывание некоторых примитивов синхронизации ядра, чтобы они стали полностью вытесняемыми (рис. 6), реализацию наследование приоритетов для внутриядерных мьютексов и конвертирование обработчиков прерывания в потоки ядра (чтобы они тоже стали полностью вытесняемыми).

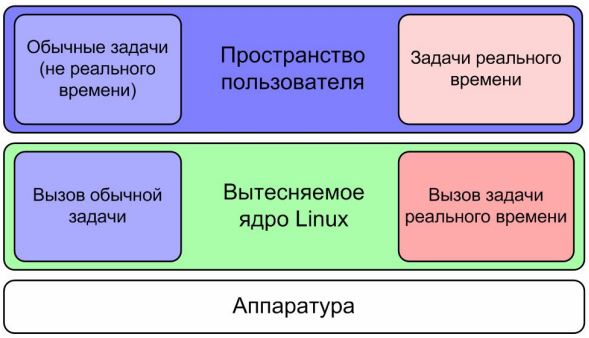

Рис. 6. Механизмы реального времени в стандартном ядре Linux

Все названные механизмы снимают ограничения, названные для концепций микроядра / наноядра. Драйвер устройства ввода-вывода оперирует в том же ядре, что и задачи реального времени, и, следовательно, могут использоваться (если сами удовлетворяют ограничениям по времени для выполняемого моделирования). Для синхронизации задач могут использоваться стандартные примитивы синхронизации API Linux. Таким образом, новые механизмы ядра Linux делают его пригодным для использования в задаче моделирования устройств реального времени.

Заключение

Ядро 2.6 Linux не только совершенная платформа для экспериментов и характеризации алгоритмов. В нем можно также найти поддержку для процессов реального времени. Поддержку мягкого реального времени можно найти в стандартном 2.6 ядре, и с небольшими модификациями (патчи ядра) можно построить приложения жесткого реального времени.

Литература

1. Википедия − Операционная система реального времени. http://ru.wikipedia.org/wiki/Операционная_система_реального_времени
2. M. Tim Jones - Anatomy of real-time Linux architectures. http://www.ibm.com/developerworks/linux/library/l-real-time-linux/
3. RTLinuxFree – What is RTLinux? http://www.rtlinuxfree.com/
4. RTAI – Beginners guide. https://www.rtai.org/index.php?module=documents&JAS_DocumentManager_op=viewDocument&JAS_Document_id=3
5. Philippe Gerum − The XENOMAI project http://archive.opengroup.org/public/member/q202/documentation/forums/rtf/Copy%20of%20xenomai%20white%20paper%208%20Apr%2002.pdf
6. Karim Yaghmour − Building a Real-Time Operating system on top of the Adaptive Domain Environment for Operating Systems http://www.opersys.com/ftp/pub/Adeos/rtosoveradeos.pdf
7. Wikipedia − Earliest deadline first scheduling http://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling
8. M. Tim Jones − Inside the Linux scheduler http://www.ibm.com/developerworks/linux/library/l-scheduler/
9. Wikipedia − Red-black tree http://en.wikipedia.org/wiki/Red-black_tree