Время реакции на события в ОСРВ

С операционными системами реального времени связано много мифов. Эта ситуация дополнительно усугубляется маркетингом, так как если рассуждать формально, то под определение ОСРВ можно подвести почти все что угодно, исходя из того, что ресурсы любого компьютера ограничены, а следовательно объем входных данных для любого алгоритма также ограничен, следовательно и время выполнения этого алгоритма также ограничено сверху некоторой константой. Поэтому для любого конкретного приложения можно задать такие временные характеристики, что абсолютно любая ОС будет способна их выполнить. Поэтому в настоящее время ярлык "ОСРВ" вешается на все продукты, где этот ярлык может помочь продажам.
Кроме того, возникает путаница между понятиями "система реального времени" и ОСРВ. Система реального времени - это конечный продукт (устройство), которое имеет заданные временные характеристики, имеющий аппаратное обеспечение, операционную систему и прикладное ПО, где каждый из этих компонентов способен работать в реальном времени.
Стандарт POSIX 1003.1 определяет ОСРВ как ОС, которая обеспечивает требуемый уровень сервиса в определенный промежуток времени. Это определение достаточно расплывчато, так как ничего не говорится о том, насколько жёсткими должны быть требования к этому уровню сервиса. В качестве примера можно привести аудио- или видеовоспроизводящую систему. Ее разработчика может вполне устраивать, что раз в день или в неделю ПО, входящее в состав устройства, может спровоцировать даже видимые пользователю, но кратковременные задержки. Иначе обстоит дело с промышленными или транспортными системами, где выход за определенные заранее задержки может спровоцировать катастрофические последствия. Различия между этими требованиями привели к разделению ОСРВ на ОС мягкого и жёсткого реального времени. Первые обеспечивают работу системы "в целом", но не обязаны гарантированно укладываться в заранее определенные временные рамки; вторые, напротив, должны гарантировать, что временные характеристики будут соблюдены даже при наихудшем стечении обстоятельств.
Поскольку путем смягчения требований к "уровню сервиса" любую

ОС можно рассматривать как ОСРВ, следует признать что удовлетворительное определение ОСРВ, которые позволило бы чётко разделять классы ОС между собой до сих пор отсутствует. Поэтому попытаемся вывести его из теоретических изысканий в области планирования процессов, а также из практических соображений.

Одной из задач ОС является обеспечение многозадачности, то есть возможности выполнять несколько псевдопараллельных потоков на процессоре, который в каждый момент времени выполняет только один поток инструкций. Потоки имеют приоритеты.
ОСРВ можно определить как ОС, которая всегда обеспечивает выполнение наиболее высокоприоритетного потока, при этом задержка перехода этого потока из состояния готовности в состояние выполнения должна быть известна заранее и не должна зависеть от параметров времени выполнения. Иными словами, наиболее высокоприоритетный поток в системе должен начать выполнение спустя фиксированное время после того, как он стал готов к выполнению, причем на это время не должны влиять ни количество объектов ОС, ни количество других (низкоприоритетных) потоков, ни их действия. Это свойство обеспечивает такую характеристику ОСРВ как предсказуемость - возможность предсказать время выполнения любого сервиса ОС, а также время реакции на внешнее событие. Если известен верхний предел задержки перехода потока из состояния готовности в состояние выполнения, а также известно количество потоков на каждом уровне приоритета, то становится возможным предсказать максимальную возможную задержку для любого потока. Например, если известно, что в системе 2 потока с высоким приоритетом, каждый из которых работает максимум 50 мкс, 1 поток среднего приоритета, работающий 100 мкс, и N потоков низкого приоритета, то время реакции на событие обрабатываемое среднеприоритетным потоком должно быть не более чем X + 2*50мкс (максимальное время работы высокоприоритетных потоков) + 100мкс (время обработки события среднеприоритетным потоком) = X + 200 мкс, где X - внутренние фиксированные задержки внутри операционной системы. Для каждого высокоприоритетного потока это время

уже X + 50 (время работы второго высокоприоритетного потока) + 50 (время обработки события высокоприоритетным потоком) = 100 мкс. Если в системе только один высокоприоритетный поток, то время реакции на обрабатываемое им событие равно задержке вносимой планировщиком ОС + время работы самого потока: X + 50 мкс. И так далее. Если время реакции высокоприоритетных потоков начинает зависеть от количества и действий низкоприоритетных потоков, то поведение системы становится непредсказуемым и она перестает удовлетворять критериям ОС жесткого реального времени.
Одним из способов достижения предсказуемости является использование детерминистичных алгоритмов. Детерминизм - свойство алгоритма, которое гарантирует, что при заданных входных данных, всегда будет выполнена одинаковая последовательность шагов. В приложении к рассуждениям выше, детерминизм ядра ОСРВ заключается в том, что при активизации высокоприоритетных потоков, ОС всегда проходит через фиксированную последовательность шагов, имеющих фиксированную задержку, которая может быть измерена. Следует отдельно отметить, что ОСРВ это совершенно необязательно "быстрая" ОС. При ее разработке во главу угла ставится именно обеспечение характеристик реального времени, и для их достижения зачастую приходится жертвовать производительностью.

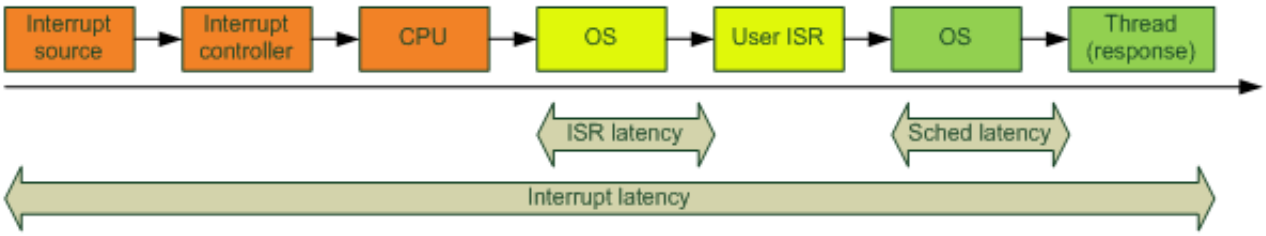
Внешние воздействия и реакция на них

Следует также определить понятие времени реакции на внешее событие, о которых говорилось выше. Внешний мир обладает довольно ограниченными возможностями влияния на процессы, происходящие внутри вычислительной системы. Обычным способом влияния на процессор являются прерывания. Природа внешних событий может быть довольно разнообразной: от прихода сетевого пакета или нажатия клавиш на клавиатуре, до получения управляющим компьютером турбореактивного двигателя команд от органов управления в кабине пилотов. Все эти события приводят к возникновению прерывания, после которого должна начаться его обработка, которая и будет являться реакцией на это событие. Временем реакции, таким

образом, является задержка между возникновением события, и завершением выполнения кода, реализующего обработку события.
Как уже упоминалось выше, встраиваемая система может реагировать на множество событий, каждое из которых может иметь выделенный вектор прерывания, а также поток, имеющий определенный приоритет. Рассмотрим типичный процесс обработки прерывания.

Процесс обработки прерывания

Процесс обработки прерываний состоит из нескольких стадий, они показаны ниже:



Если событие произошло в точке 0 на временной шкале, то процесс его обработки выглядит следующим образом: вначале сигнал о событии проходит через внутреннюю логику источника прерываний, это то время, которое проходит от момента фиксации события датчиком, до установки в активное положение запроса прерывания. Затем запрос обрабатывается контроллером прерываний. Контроллер прерываний выполняет функции приоритезации источников прерываний, маскировки отдельных линий и так далее, даже в случае если источник размаскирован, серия проверок происходящая внутри контроллера также вносит некоторую задержку, которая изображена на рисунке как "interrupt controller". Далее запрос прерывания подается на соответствующий вход процессора. Последний также анализирует состояние флагов прерываний, находит соответсвующий вектору низкоуровневый обработчик (который обычно устанавливается ОС) и передает ему управление. Задержка от возникновения события до передачи управления самому низкоуровневому обработчику целиком вносится аппаратным обеспечением, она

показана оранжевым цветом на рисунке.
Далее управление получает ОС, выполняет действия по сохранению контекста выполнявшегося потока, переключение стеков, поиск установленного пользователем обработчика и передачу ему управления. Эта задержка вносится ОСРВ и называется задержкой обработчика прерывания (ISR latency). Пользовательский обработчик должен выполнить действия специфичные для устройства-источника прерываний, размаскировать вектор, после чего активировать поток-обработчик данного события. Следует отметить, что в большинстве процессорных архитектур все более низкоприоритетные прерывания остаются замаскированными все время работы пользовательского ISR, причем не всегда эти приоритеты могут быть настраиваемы программно. Кроме того, один и тот же вектор может соответствовать разным событиям, которые имеют разный приоритет с точки зрения обработки, ISR не может блокироваться, и поэтому не может иметь разделяемых данных с потоками и т.д. Поэтому обрабатывать прерывания прямо в ISR не рекомендуется, это ухудшает возможности портирования кода на другие архитектуры, а также снижает гибкость в назначении приоритетов событиям и усложняет программирование.
После завершения пользовательского ISR управление снова получает ОС, которая (если состояние потоков изменилось) запускает планировщик, выбирающий для выполнения наиболее приоритетный поток. Эта задержка называется задержкой планирования (scheduling latency). Наконец, управление получает поток, соответствующий произошедшему событию (если из прерывания был активирован не самый высокоприоритетный поток, между ISR и потоком могут также выполняться другие, более приоритетные потоки), который производит обработку произошедшего прерывания, что и составляет реакцию на событие. После завершения кода обработки прерывания последнее считается обработанным.
Как видно из этого описания, от ОС зависит лишь часть времени реакции, поэтому применение ОСРВ еще не делает систему способной работать в реальном времени: все остальные компоненты также должны соответствовать этим критериям, включая аппаратное обеспечение и пользовательское ПО (код

обработки события как в потоке так и в ISR).

Схемы синхронизации

Хотя активация потока из прерывания выглядит простой и четко определенной вещью, на самом деле возможны раличные реализации этого механизма, по-разному влияющие на латентность. Переход потока в активное состояние обычно заключается в том, что изменяется состояние планировщика (состояние его внутренних структур данных, содержащих активные потоки, потоки готовые к выполнению и т.д.).
Так как прерывания - это асинхронные события, которые могут прерывать в том числе выполнение и самого планировщика, необходима синхронизация доступа к его структурам данных. Наиболее простой является т.н. унифицированная схема синхронизации: в этом случае планировщик запрещает прерывания во время своей работы, аналогично поступают и обработчики прерываний, для того чтобы исключить возможное прерывание самого обработчика другими прерываниями во время работы со структурами данных планировщика. Альтернативной является т.н. сегментированная схема: в ней обработчик прерывания разделяется на две части или два сегмента, один называется собственно обработчиком, а второй - т.н. отложенной процедурой. Предполагается, что обработчики должны только ставить в очередь "отложенную процедуру", в которой и происходят действия по дальнейшей обработке прерывания. При этом, отложенная процедура не является потоком, и на нее распространяется большинство ограничений, характерных и для обработчиков, в частности, невозможность блокироваться и ждать чего-либо.
Унифицированная схема, которая предполагает манипуляции со структурами данных планировщика внутри обработчиков прерываний, очевидно, обладает более высокой латентностью: другие прерывания, в том числе высокоприоритетные, должны дождаться завершения выполнения сервисов ОС во всех других обработчиках. Максимальное время реакции в таком случае определяется как время работы всех вложенных ISR + работа планировщика + время наиболее длинной критической секции внутри ОС + время работы обработчика прерывания в потоке.

Сегментированная схема обеспечивает лучшую латентность обработчиков за счет очень малого количества работы выполняемой внутри ISR, однако время реакции зачастую оказывается хуже, чем в унифицированной схеме. В сегментированной схеме сервисам ОС (включая планировщик) нет необходимости полностью запрещать прерывания, достаточно запретить выполнение "отложенных процедур". Если любая работа со структурами данных планировщика может производиться только из них, но не из обработчиков, то нет нужды запрещать прерывания во время выполнения сервисов ОС. Отдельного упоминания заслуживают ОС, которые "никогда не запрещают прерывания". Действительно, сегментированная схема синхронизации позволяет при определенных усилиях реализовать ОС которая технически пользоваться инструкциями запрета прерываний не будет. Однако необходимость синхронизации планировщика и прерываний никуда не девается, просто роль запрета прерываний играет в этом случае запрет "отложенных процедур", но поскольку внешние события доставляются с помощью их же, время реакции ОС вовсе не становится равным нулю. Как и в случае унифицированной схемы, оно состоит из максимальной критической секции внутри ядра + время работы всех вложенных ISR и отложенных процедур + работа планировщика + время работы обработчика прерывания в потоке.

Профайлинг и трассировка

Так как ОС может содержать множество критических секций, влияющих на латентность, единственным достоверным способом узнать точное время блокировки прерываний ядром можно только с помощью инструментации и трассировки. Ядро должно предоставлять возможности записывать время входа и выхода из критических секций с высоким разрешением, тогда это позволит разработчику системы реального времени провести тестовый запуск приложения с модифицированным ядром, узнать размер всех критических секций, которые выполняются при работе данного приложения, и затем отталкиваться от полученных значений при проектировании системы. При этом критически важно, чтобы полученные числа оставались константами, так как

если при изменении приложения они будут изменяться, то легко выйти за пределы гарантированного времени реакции, что может повлечь некорректную работу системы.

Многопроцессорные системы

Другим важным вопросом является разработка многопроцессорных ОСРВ. Это тема отдельной большой статьи, поэтому остановимся вкратце только на том, может ли поддержка многопроцессорности влиять на латентность.
Если ОСРВ обеспечивает автоматическое распределение потоков по процессорам, это может быть реализовано с помощью некоторого общего для всех процессоров хранилища потоков, готовых к выполнению. Если это так - доступ к такому хранилищу должен быть синхронизирован для каждого из процессоров, что приводит к необходимости использования спин-блокировок. В этом случае максимальная латентность увеличивается за счет того, что планировщик каждого из процессоров теоретически может оказаться вынужден ожидать завершения работы планировщика на всех других процессорах. То есть компонент латентности "работа планировщика" умножается на количество процессоров в системе.
Кроме того, на многопроцессорных системах сложнее получить врмя максимальной критической секции с помощью профилирования, так как текущий процессор может ожидать других процессоров, действия которых не учитываются при трассировке и это может вносить неучтенные задержки.

Тестовый стенд

В качестве тестовой платы использовалась Freescale i.MX6Q Sabre Lite board (Cortex A9 MPCore), аппаратный таймер EPIT был настроен на генерацию прерываний примерно каждые 6 мс. В качестве обработчика использовался единственный высокоприоритетный поток. Для имитации нагрузки использовалось 2000 низкоприоритетных потоков (все с одинаковым значением приоритета), которые пробуждались одновременно каждые 10 с. Ситуация с одновременным пробуждением большого числа потоков в пределах одного тика

системного таймера может возникнуть и в реальной системе, когда большое число потоков ожидают различных событий с таймаутом, возможна такая последовательность событий, даже в случае различных значений таймаутов, что все таймауты сработают одновременно и все ожидавшие потоки одновременно перейдут в состояние готовности к выполнению. ОС жесткого реального времени должна корректно обрабатывать эту ситуацию, резко возросшая нагрузка не должна приводить к существенному росту времени реакции.
Относительно большое число тестовых потоков используется для большей выраженности эффекта роста латентности от числа потоков, в случае наличия зависимости (нефиксированной латентности), это будет проявляться и в системе с 20-ю потоками, просто не столь явно. ОСРВ непригодна для использования в системах жесткого реального времени в случае факта наличия такой зависимости, т.к. это означает отсутствие детерминизма и как следствие - отсутствие предсказуемости, так как латентность перестает зависеть только от приоритета, и начинает зависеть от ряда других факторов которые довольно трудно измерить и учесть при проектировании.
В качестве объекта измерения была выбрана задержка между входом в обработчик и активацией соответствующего высокоприоритетного потока. Эта величина характеризует комбинированную задержку связанную как с работой, производимой ОС по активации потока из обработчика, так и с задержкой планирования, проходящей между выходом из ISR до активации потока. Задержка обработчика не измерялась. Измерение проводились с помощью осциллографа и GPIO.
В качестве сравниваемых ОС использовалась одна из распространенных на рынке бесплатных ОСРВ и FX-RTOS 2.3. Обе ОС используют идентичные настройки компиляции, оптимизации по-умолчанию. Выполнение кода происходит из встроенной флеш-памяти. Также использовался идентичный период системного таймера (1 мс), отключены оптимизации планировщика с помощью битовых и прочих операций. Кроме системного таймера и источника тестовых прерываний других источников прерываний в системе нет, тестовое прерывание имеет более высокий приоритет по сравнению в системным таймером (может прерывать последний). Проверки ошибок отключены. Обе ОС

используют одинаковую схему синхронизации (унифицированную).
Так как процессор ARM Cortex A9 имеет кэш данных индексируемый виртуальными адресами, кэш не работает до момента включения модуля трансляции адресов (MMU). С целью достижения максимальной производительности, перед запуском теста, подготавливаются таблицы страниц и включается страничная адресация отображающая физическое и виртуальное адресные пространства как 1 к 1. Тест запускается на одном из ядер (остальные ядра не включаются).

Результаты теста

Популярная и широко используемая ОСРВ, с которой производилось сравнение, имеет следующие характеристики: Минимальная задержка составляет 1.5 мкс. Каждые 10 секунд (период одновременной активации группы низкоприоритетных потоков) наблюдается увеличение задержки на 2 порядка, до 120-130 мкс, в отдельных случаях до более чем 500 мкс. Таким образом, данную ОС нельзя использовать в системах жёсткого реального времени. Даже если указанная задержка имеет ограничение сверху, сами значения в сотни микросекунд при работе на процессоре с частотой 400МГц слишком велики.
С FX-RTOS ситуация обстоит следующим образом: Минимальная задержка составляет 2 мкс. При этом нужно отметить, что в отличие от предыдущего случая, FX-RTOS использует выделенный стек прерываний, который хотя и привносит небольшую задержку, снижает требования к размеру стека потоков, а также предотвращает переполнение стека при превышении определенного порога частоты прерываний. Во время активации группы низкоприоритетных потоков тестовое прерывание чаще происходит во время критических секций и средняя латентность возрастает до 3.5 мкс, в отдельных случаях (вероятно, при прерывании тестовым обработчиком обработчика системного таймера) до 6 мкс. За тестовый период (30 минут, 180 активаций группы низкоприоритетных потоков) не было зафиксировано превышения задержки сверх 6 мкс.

Заключение

Не всякая ОС, имеющая в своем названии "реального времени" или даже "жёсткого реального времени" на самом деле является таковой. Расплывчатость формулировок и определений позволяет использовать их в маркетинговых целях, вводя в заблуждение потенциальных покупателей. При построении системы жесткого реального времени не следует полагаться только на рекламные брошюры, а обязательно проводить тестирование разрабатываемой системы с учетом самого худшего сценария, который может возникнуть в процессе ее работы. Невыполнение этих правил может привести к тому, что даже широко используемые в индустрии решения могут оказаться неработоспособными при определенном стечении обстоятельств, которое не было учтено при тестировании.