**Глава 3.**

**Управление прерываниями.**

**3.1 Введение и обзор.**

**События.**

Встраиваемые системы реального времени должны выполнять действия в ответ на события, которые возникают из окружения. Например, пакет, прибывающий по Ethernet (событие) может требовать передачи его в TCP/IP стек для обработки (действие). Не тривиальные системы должны обслуживать события, которые возникают от нескольких источников, каждый из которых будет иметь различные вычислительные затраты и требования отклика. В каждом случае, принятие решения должно быть выполнено на основе лучшей стратегии реализации обработки событий:

1. Как должно быть обнаружено событие? Обычно используются прерывания, но и входы также могут быть опрошены.
2. Когда используются прерывания, как много обработки должно быть выполнено внутри обработчика прерывания (ISR) и сколько еще надо выполнить снаружи? Обычно желательно сохранять каждый ISR минимально коротким.
3. Как могут события взаимодействовать с основным (не ISR) кодом, и как этот код может быть структурирован для наилучших условий обработки потенциальных асинхронных случаев.

FreeRTOS не обязывает применять какую-либо определенную стратегию обработки разработчика приложения, но предоставляет возможности позволяющие выбрать стратегию, реализуемую простым и удобным для сопровождения способом. Заметьте, что только API функции и макросы, завершающиеся суффиксом «FromISR» или «FROM\_ISR» должны быть использованы внутри обработчика прерывания.

**Обзор.**

Назначение этой главы – дать читателям хорошее понимание:

* Какие FreeRTOS функции могут быть использованы внутри обработчиков прерываний.
* Какая схема отсрочки прерываний может быть реализована.
* Как создать и использовать двоичные семафоры и счётные семафоры.
* Разница между двоичным и счётным семафором.
* Как использовать очередь для передачи данных в и из ISR.
* Модель вложенных прерываний порта FreeRTOS для Cortex-M3.

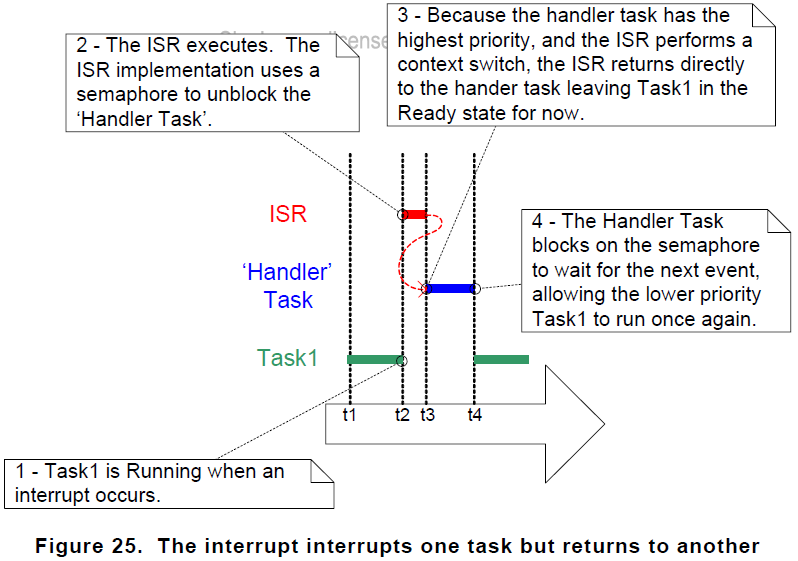
**3.2 Обработка отложенных прерываний.**

**Двоичные семафоры, используемые для синхронизации.**

Двоичные семафоры могут быть использованы для разблокировки задачи всякий раз, когда происходят отдельные прерывания, таким образом, эффективно синхронизируя задачу с прерыванием. Это позволяет реализовывать основную часть обработки прерывания внутри синхронизируемой задачи, оставляя лишь небольшую, очень быстро обрабатываемую часть непосредственно в ISR. Обработка прерывания, как говорят, была отложена до задачи обработчика.

Если обработка прерывания особенно критична ко времени, то приоритет задачи обработчика может быть установлен так, чтобы гарантировать, что задача обработчика всегда будет вытеснять другую задачу в системе. ISR тогда может быть реализован так, что будет включать переключение контекста, чтобы гарантировать, что ISR вернёт управление прямо задаче обработчика, когда сам ISR завершит выполнение. Это создаёт эффект непрерывной обработки всего события, как будто, если бы всё было реализовано внутри ISR непосредственно. Эта схема демонстрируется на рисунке 25.

1. Задача 1 запускается, когда происходит прерывание.
2. ISR запускается. Реализация ISR использует семафор для разблокировки «задачи обработчика».
3. Так как задача обработчика имеет наивысший приоритет, и ISR выполняет переключение контекста, возвращается прямо в задачу обработчика, помещая задачу 1 в состояние «Готов».
4. Задача обработчика блокируется на семафоре для ожидания следующего события, позволяя более низкоприоритетной задаче 1 запуститься снова.



Задача обработчика использует блокирующий «взять» вызов семафора, что означает вхождение в состояние «Блокирован», для ожидания происшествия события. Когда событие происходит, ISR использует операцию «дать» на том же семафоре, для разблокировки задачи так, что требуемое обработки событие может быть обработано.

«Взятие семафора» и «отдача семафора» - это концепции, которые имеют различный смысл, в зависимости от сценария их использования. В классической терминологии семафора, «взять семафор» - это эквивалент операции P(), и «отдать семафор» - это эквивалент операции V().

В этом сценарии синхронизации прерывания, двоичный семафор может считаться концептуально очередью с единичной длиной. Очередь может содержать максимум один элемент в любой момент времени, так что она всегда либо пустая, либо полная (hence, binary). Вызовом xSemaphoreTake(), задача обработчика фактически пытается прочитать из очереди с временем блокировки, заставляя задачу войти в состояние «Блокирован», если очередь пуста. Когда событие происходит, ISR использует xSemaphoreGiveFromISR() функцию, чтобы поместить метку (семафор) в очередь, тем самым заполняя её. Это приводит к выходу задачи обработчика из состояния «Блокирован» и удалению метки, тем самым опустошая очередь. Когда задача обработчика завершит свою обработку, она вновь попытается прочитать очередь, и, обнаружив её пустой, снова входит в состояние «Блокирован», в ожидании очередного события. Эта последовательность демонстрируется на рисунке 26.

Рисунок 26 показывает прерывание «отдающее» семафор, несмотря на то, что оно его не брало, и задача «берёт» семафор, но никогда не отдаёт его назад. Именно по этому, данный сценарий был описан в терминах очереди. Это часто приводит к конфузу, так как в других сценариях использования семафора, в которых задача, которая берёт семафор, должна всегда отдавать его обратно (см. главу 4), работают другие правила.

**Написание обработчиков прерывания FreeRTOS.**

Архитектура Cortex-M3 и порт FreeRTOS позволяют писать ISRы полностью на C, даже когда ISR хочет сам переключить контекст. Следующие примеры демонстрируют реализацию ISR.

**API Функция vSemaphoreCreateBinary().**

Дескрипторы всех различных типов семафоров FreeRTOS сохраняются в переменной типа xSemaphoreHandle. Перед тем, как использовать семафор, он должен быть создан. Чтобы создать двоичный семафор, используйте API функцию vSemaphoreCreateBinary().

Прототип функции приведён в листинге 42.

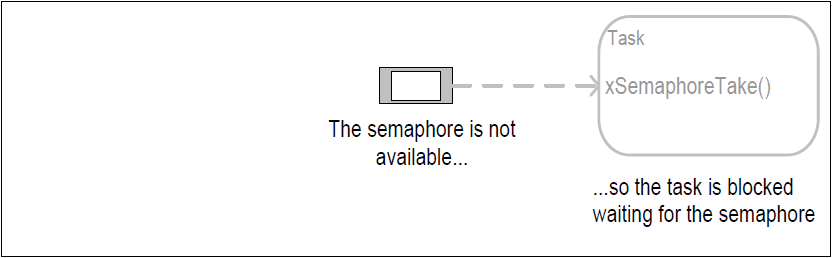
Листинг 42. Прототип функции vSemaphoreCreateBinary().

void vSemaphoreCreateBinary(xSemaphoreHandle xSemaphore);

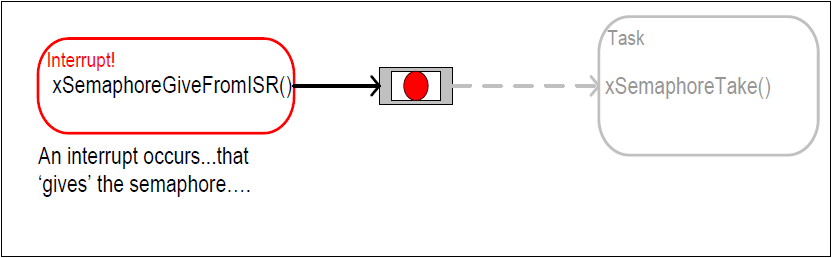
Параметры:

xSemaphore – созданный семафор. Заметьте, что функция vSemaphoreCreateBinary() на самом деле реализована как макрос, так что дескриптор семафора передаётся в неё напрямую, вместо его адреса. Примеры в этой главе включают вызовы vSemaphoreCreateBinary(), которые могут быть использованы как ссылки и скопированы.

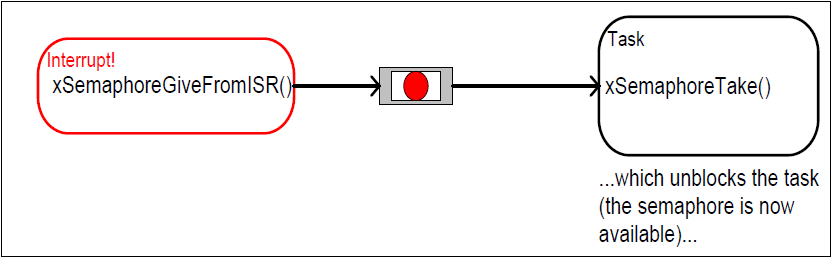
Семафор не доступен… Так что задача блокируется, в ожидании семафора.



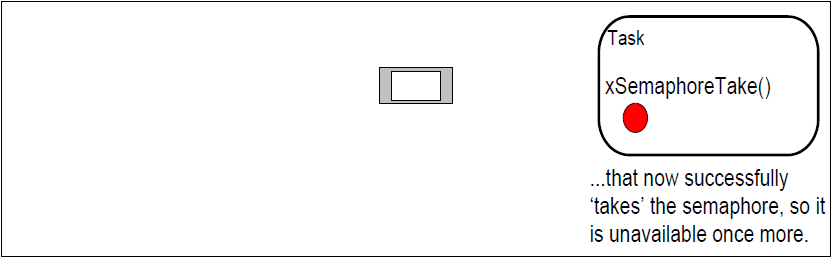
Прерывание! Происходит прерывание, которое «выдаёт» семафор.



…Которое разблокирует задачу (так как семафор теперь доступен).



…Которая, в свою очередь, теперь успешно «берёт» семафор, так что он теперь больше не доступен.



Теперь задача может выполнить действие, а когда оно его завершит, она снова попытается «взять» семафор, что приведёт к её блокировке.

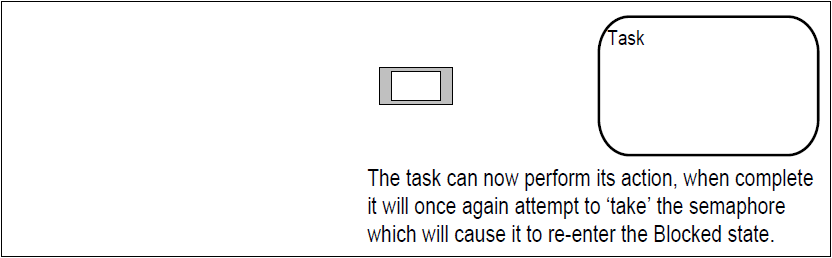


Рисунок 26. Использование бинарного семафора для синхронизации задачи с прерыванием.

**API функция xSemaphoreTake().**

«Взятие» семафора означает «получить» или «принять» семафор. Семафор может быть взят только если он доступен. В классической терминологии семафоров, xSemaphoreTake() – это эквивалент операции P().

Все различные типы семафоров FreeRTOS, за исключением рекурсивных семафоров, могут быть «взяты», используя функцию xSemaphoreTake().

Функция xSemaphoreTake() не должна использоваться в обработчиках прерываний.

Прототип функции приведён в листинге 43.

Листинг 43. Прототип API функции xSemaphoreTake().

portBASE\_TYPE xSemaphoreTake(xSemaphoreHandle xSemaphore,   
 portTickType xTicksToWait);

Параметры:

xSemaphore – получаемый семафор. Семафор передаётся в функцию переменной типа xSemaphoreHandle, он должен быть явно создан, перед тем как использовать.

xTicksToWait – максимальное количество времени, которое задача должна оставаться в состоянии «блокирован», в ожидании семафора, если он пока не доступен. Если xTicksToWait равен 0, то функция вернёт управление немедленно, если семафор не доступен. Время блокировки указывается в тик-периодах, так что абсолютное время, зависит от частоты тиков. Константа portTICK\_RATE\_MS может быть использована для конвертации времени, указанного в миллисекундах во время, указанное в тиках. Установка этого параметра в значение portMAX\_DELAY приведёт к тому, что задача будет ждать семафор бесконечно (без таймаута), если INCLUDE\_vTaskSuspend установлен в «1» в файле FreeRTOSConfig.h.

Возвращаемые значения:

Возможны два значения:

1. pdPASS. Возвращается, только если вызов xSemaphoreTake() был успешен при получении семафора. Если было определено время блокировки (xTicksToWait не был равен 0), то вызывающая задача будет помещена в состояние «блокирован» для ожидания семафора, если он не был мгновенно получен, но семафор станет доступным, прежде чем истечёт время блокировки.
2. pdFALSE. Семафор не доступен. Если было указано время блокировки (xTicksToWait не равен 0), то вызывающая задача была помещена в состояние «блокирован» для ожидания семафора, но время блокировки истекло прежде, чем семафор стал доступен.

**API функция xSemaphoreGiveFromISR().**

Все различные типы семафоров FreeRTOS, за исключением рекурсивных семафоров, могут быть «отданы», с помощью функции xSemaphoreGiveFromISR(). Это специальная форма xSemaphoreGive(), которая предназначена для использования в обработчиках прерывания.

Прототип функции приведён в листинге 44.

Листинг 44. Прототип API функции xSemaphoreGiveFromISR().

portBASE\_TYPE xSemaphoreGiveFromISR(xSemaphoreHandle xSemaphore,   
 portBASE\_TYPE \*pxHigherPriorityTaskWoken);

Параметры:

xSemaphore – «отдаваемый» семафор. Семафор передаётся в переменной типа xSemaphoreHandle и должен быть создан явно перед использованием.

pxHigherPriorityTaskWoken – это возможно, что единственный семафор будет иметь одну или несколько заблокированных задач, которые ожидают, пока семафор не станет доступен. Вызов xSemaphoreGiveFromISR() может сделать семафор доступным, и это приведёт к тому, что задача покинет состояние блокирован. Если вызов xSemaphoreGiveFromISR() приведёт к тому, что задача покинет состояние «заблокирован», и разблокированная задача имеет приоритет выше или равный, чем приоритет задачи, выполняемой в настоящее время (задача, которая была прервана), то функция xSemaphoreGiveFromISR() установит параметр \*pxHigherPriorityTaskWoken в значение pdTRUE. Если xSemaphoreGiveFromISR() установит это значение в pdTRUE, то переключение контекста должно выполняться перед тем, как произойдёт выход из прерывания. Это гарантирует, что прерывание вернётся прямо в более приоритетную «готовую» задачу.

Возвращаемые значения:

Возможны два возвращаемых значения:

1. pdPASS. Будет возвращено, только если вызов xSemaphoreGiveFromISR() успешен.
2. pdFAIL. Если семафор уже доступен, он не может быть отдан. И тогда xSemaphoreGiveFromISR() вернёт pdFAIL.

**Пример 12. Использование двоичного семафора для синхронизации задачи с прерыванием.**

Этот пример использует двоичный семафор для разблокировки задачи из под прерывания – фактически синхронизируя задачу с прерыванием.

Простая периодическая задача используется для генерации прерывания каждые 500 мс. В данном случае используется програмно генерируемое прерывание, так как это позволяет управлять временем генерации прерывания, что в свою очередь позволяет исследовать последовательность исполнения более легким способом. Листинг 45 показывает реализацию периодической задачи. mainTRIGGER\_INTERRUPT() просто устанавливает бит в регистре набора ожидающих прерываний контроллера прерываний.

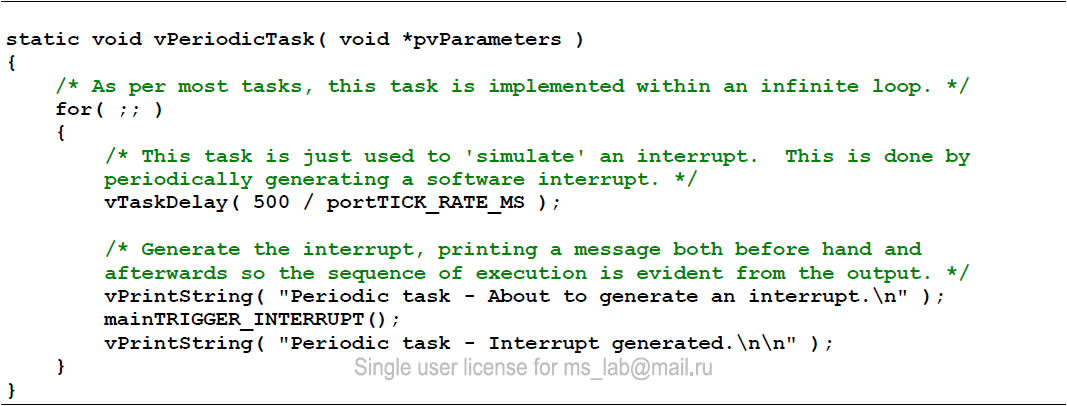
Листинг 45. Реализация задачи, которая периодически генерирует программные прерывания в примере 12.

Комментарии:

Как и большинство задач, эта задача реализована как бесконечный цикл.

Эта задача просто используется для симуляции прерывания. Это достигается периодической генерацией программного прерывания.

Печать сообщения выполняется как до, так и после генерации прерывания, так что последовательность исполнения будет хорошо просматриваться на выводе.



Листинг 46 показывает реализацию задачи обработчика – задачи, которая синхронизирована с программным прерыванием посредством использования двоичного семафора. Сообщение выводится на печать на каждой итерации задачи, так что последовательность, в которой выполняется задача и прерывание очевидна из вывода, производимого в результате выполнения примера.

Листинг 46. Реализация задачи обработчика (задача, которая синхронизирована с прерыванием) в примере 12.

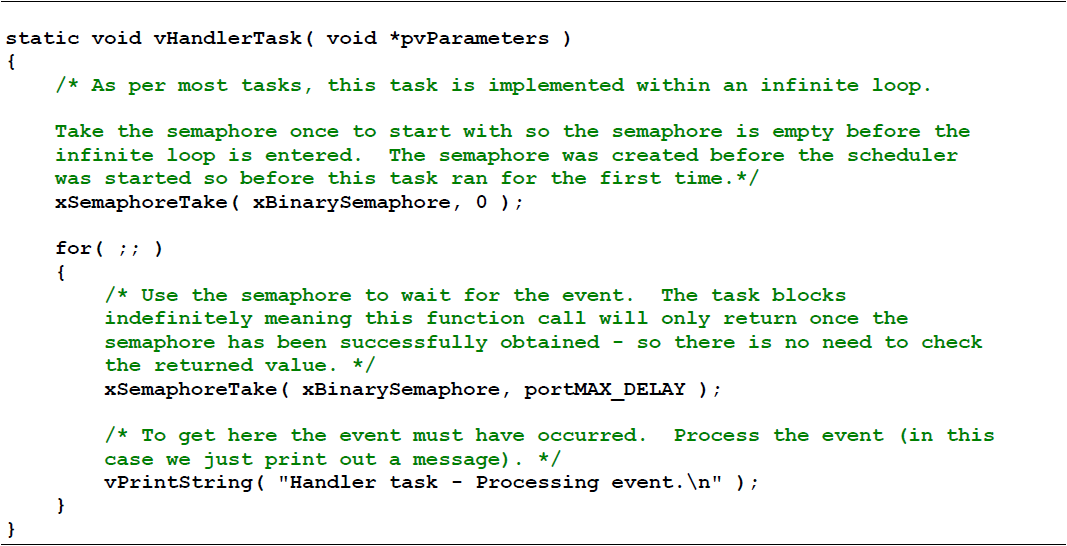
Комментарии:

Как и большинство задач, эта задача реализована как бесконечный цикл.

Берём семафор сразу после старта, чтобы семафор стал пустым прежде, чем зайдем в бесконечный цикл. Семафор был создан перед запуском планировщика, и следовательно, до того, как в первый раз запустилась эта задача.

Используем семафор для ожидания события. Эта задача блокируется на неопределённое время, что означает, что вызов этой функции вернёт управление только при успешном получении – поэтому нет необходимости проверять возвращаемое значение.

До печати добираемся после того, как произойдет событие. Обрабатываем событие (в этом случае мы просто печатаем сообщение).



Листинг 47 показывает подпрограмму обработки прерывания, которая представляет собой простую C-функцию. Она очень маленькая и содержит только очистку флага прерывания и «отдачу» семафора для разблокировки задачи обработчика.

Макрос portEND\_SWITCHING\_ISR() является частью порта FreeRTOS Cortex-M3 и безопасным эквивалентом taskYIELD() для ISR. Это приводит к принудительному переключению контекста только, если его параметр не ноль (не равен pdFALSE).

Обратите внимание на то, как используется xHigherPriorityTaskWoken. Он инициализируется значением pdFALSE перед тем, как быть переданным по ссылке в xSemaphoreGiveFromISR(), где он получит значение pdTRUE, только если xSemaphoreGiveFromISR() приведёт к тому, что состояние «готов» получит задача равного или более высокого приоритета, чем выполняемая на данный момент. portEND\_SWITCHING\_ISR() затем выполняет переключение контекста только если xHigherPriorityTaskWoken будет равен pdTRUE. Во всех остальных случаях в переключении контекста нет необходимости, поскольку задача, которая выполнялась перед прерыванием, все ещё имеет самый высокий приоритет среди задач, способных выполняться.

Листинг 47. Обработчик программного прерывания, используемый в примере 12.

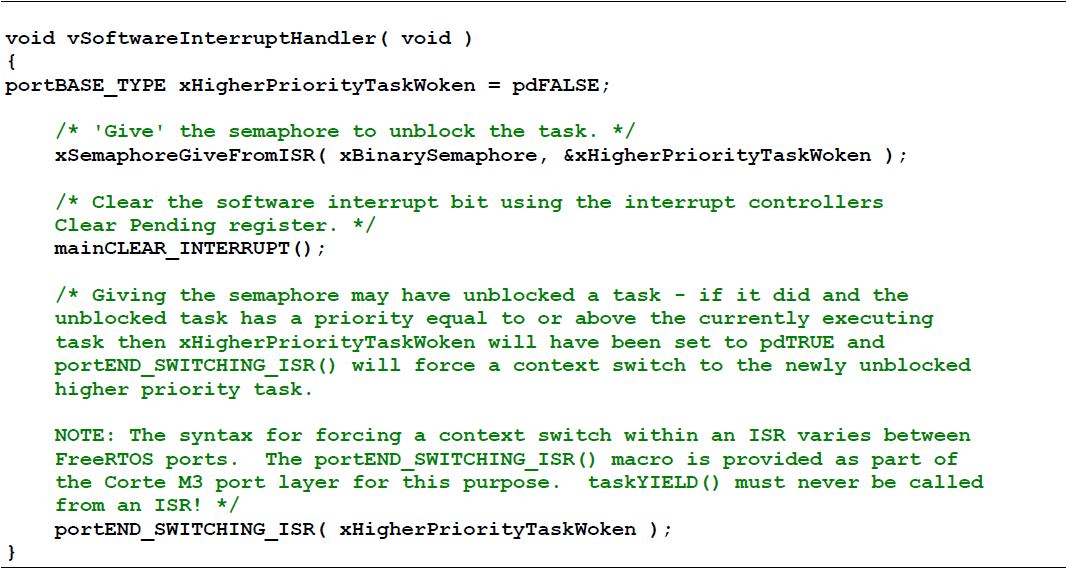
Комментарии:

Отдаём семафор для разблокировки задачи.

Очищаем флаг прерывания, используя регистр очистки ожидающих прерываний контроллера.

Отдаваемый семафор может иметь разблокированную задачу. В этом случае, если разблокируемая задача имеет приоритет равный, или выше чем текущая выполняемая задача, то xHigherPriorityTaskWoken будет установлен в pdTRUE и portEND\_SWITCHING\_ISR() принудительно переключит контекст в новую разблокированную задачу с более высоким приоритетом.

ЗАМЕЧАНИЕ: синтаксис для принудительного переключения контекста внутри ISR различается на разных портах FreeRTOS. Макрос portEND\_SWITCHING\_ISR() предоставляется как часть уровня порта Cortex M3 для этих целей. taskYIELD() никогда не должен вызываться из ISR.



Функция main() создаёт двоичный семафор и задачи, конфигурирует программное прерывание и запускает планировщик. Реализация показана в листинге 48.

Листинг 48. Реализация main() в примере 12.

Комментарии:

Конфигурируем аппаратное обеспечение и средства отладки.

Прежде чем использовать семафор – он должен быть явно создан. В этом примере создан двоичный семафор.

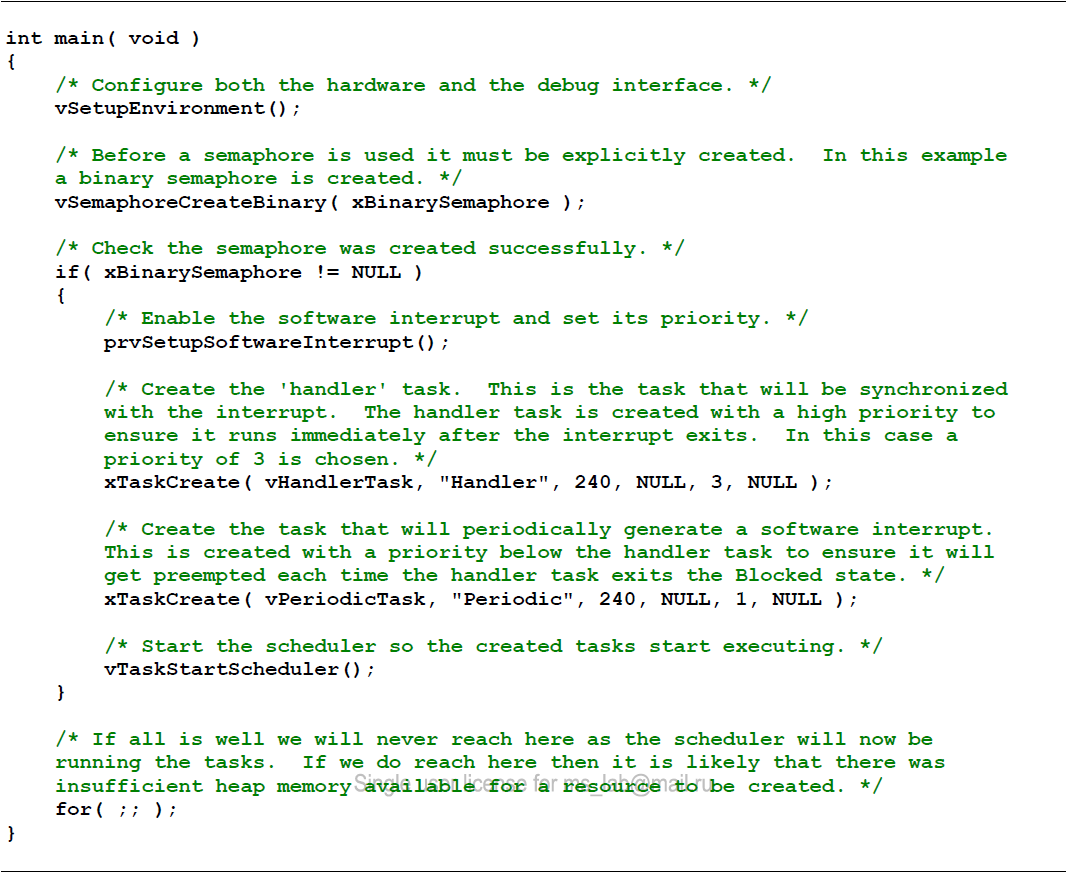
Проверяем, что семафор был успешно создан.

Разрешаем программные прерывания и устанавливаем их приоритет.

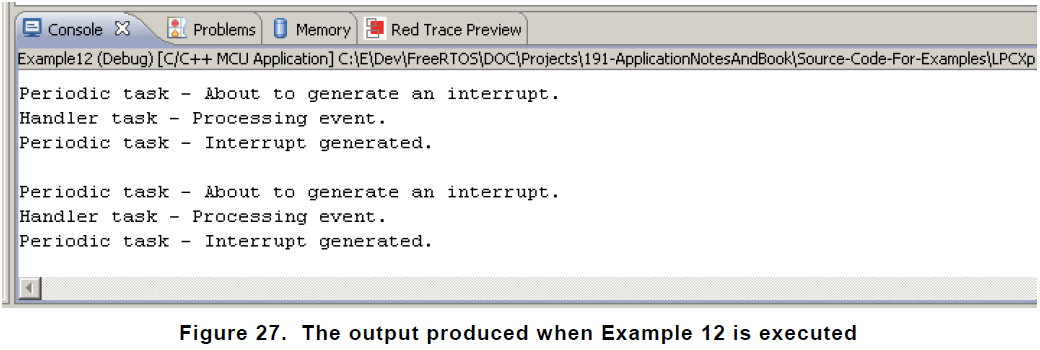
Создаем задачу обработчика. Это задача, которая будет синхронизирована с прерыванием. Задача обработчика создаётся с более высоким приоритетом, чтобы гарантировать её запуск немедленно после выхода из прерывания. Для неё выбран приоритет 3.

Создаем задачу, которая периодически генерирует программное прерывание. Она создаётся с приоритетом ниже, чем задача обработчика, чтобы гарантировать её вытеснение всякий раз, когда задача обработчика покидает состояние «блокирован».

Запускаем планировщик, и теперь все созданные задачи начинаются выполняться.



Пример 12 производит вывод, показанный на рисунке 27. Как ожидалось, задача обработчика исполняется вскоре после того, как генерируется прерывание, так что вывод из задачи обработчика делит вывод, производимый периодической задачей. Дальнейшее объяснение предоставлено на рисунке 28.

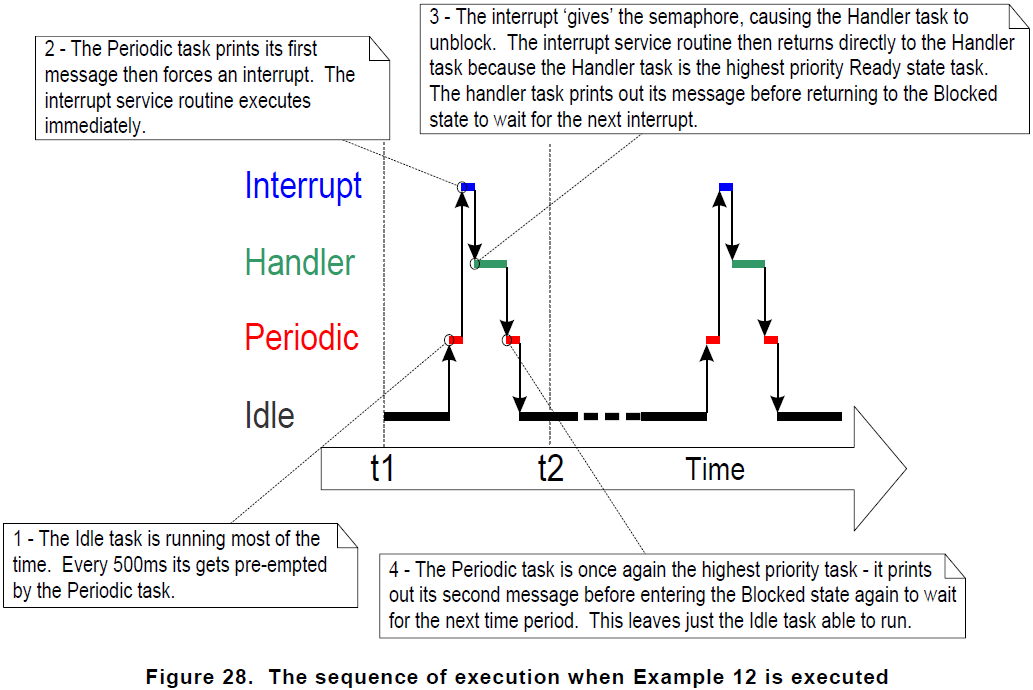


1 – большую часть времени выполняется холостая задача. Каждые 500 мс она вытесняется периодической задачей.

2 – периодическая задача печатает её первое сообщение, затем возбуждает прерывание. Обработчик прерывания стартует немедленно.

3 – прерывание «отдаёт» семафор, что приводит к разблокировке задачи обработки. Затем обработчик прерывания передаёт управление прямо в задачу обработки, потому что задача обработки имеет наивысший приоритет среди задач, находящихся в состоянии «готов». Задача обработки печатает её сообщение перед тем как снова вернуться в состояние «заблокирован», для ожидания следующего прерывания.

4 – периодическая задача снова становится задачей с наивысшим приоритетом – она печатает своё второе сообщение, прежде чем заблокироваться, чтобы ожидать своего периода исполнения. Её блокировка просто делает холостую задачу способной к исполнению.



**3.3 Счётные семафоры.**