Василий @qdx
Пользователь

29 сентября 2011 в 17:00

FreeRTOS: мьютексы и критические секции

Программирование микроконтроллеров*

Здравствуйте. Это заключительная статья о многопоточном окружении FreeRTOS в которой я расскажу про мьютексы и критические секции.
Ссылки на предыдущие части:

- FreeRTOS: введение.
- FreeRTOS: межпроцессное взаимодействие.

Мьютексы.

Рассмотрим ситуацию, при которой необходимо иметь совместный доступ к ресурсу (порт, область памяти, какая-либо переменная) да ещё т чтобы в единицу времени только один таск мог обращаться к данному ресурсу, в то время как остальные задачи, желающие получить доступ ресурсу должны ждать своей очереди.
На помощь приходит специальный тип семафора — мьютекс(mutex). Данная абстракция представляет собой своеобразный жетон, имея кото таск может иметь доступ к ресурсу, и который необходимо вернуть по окончанию работы с ресурсом, причем вернуть «жетон» может только таск, который его взял.
Данную логику иллюстрируют следующие изображения:

Рисунок 1. Исходная ситуация — 3 таска, которые хотят получить доступ к ресурсу.

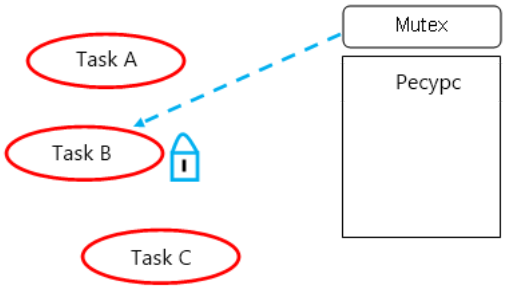

Рисунок 2. Таск B — взял «жетон» (мьютекс).

Рисунок 3. Таск B получил доступ к ресурсу и выполняет нужные ему действия.

Рисунок 4. После окончания работы таск B возвращает жетон и в результате другие таски могут получить доступ к ресурсу.
Для создания мьютекса используется специальная API функция:

```
xSemaphoreHandle xSemaphoreCreateMutex( void );
```

Которая возвращает созданный мьютекс, или NULL, если недостаточно памяти.
Следующий код показывает механизм работы с мьютексом:

```
 if( xSemaphoreTake( xMutex, portMAX_DELAY ) == pdTRUE )
{
        // Здесь происходит защищенный доступ к ресурсу.

       // Отдаем "жетон".
       xSemaphoreGive( xMutex );
}
```

Критические секции.

Что если по ходу работы логики Вашего кода, необходим короткий участок кода, выполнение которого не должно быть прервано переключе на другой таск (примером такого участка может послужить запись какого-либо значения в порт). Для этого в FreeRTOS используются специа макросы:

```
taskENTER_CRITICAL();
{
```

```
    // Здесь тот самый код, выполнение которого не должно быть прервано.
}
taskEXIT_CRITICAL();
```

Во время выполнения этого участка кода не может быть произведено переключение контекста (т.е. переключение на другую задачу), но мог приходить прерывания на версиях FreeRTOS с поддержкой вложенных прерываний, но не все, а только те у которых приоритет выше конста **configMAX_SYSCALL_INTERRUPT_PRIORITY**.

Другой подход построения критических секций — приостановка работы планировщика. Разница лишь в том, что критическая секция, постро на макросах, защищает выполняемый участок кода от совместного доступа других тасков, и прерываний, а критическая секция, построенна приостановке планировщика — только от других тасков.
Для этого используют 2 API функции:

```
void vTaskSuspendAll();
{
    // Код критической секции.
}
portBASE_TYPE xTaskResumeAll();
```

При этом значение, которое возвращает функция **xTaskResumeAll();** указывает на необходимость выполнения принудительного переключе контекста с помощью **taskYIELD();**

Инверсия приоритетов и тупики.

В прошлой статье я вскользь упомянул о gatekeeper task, как о решении проблемы инверсии приоритетов и попадания тасков в тупик. Здесь более подробно хотел описать, что это такое.

Инверсия приоритетов — это ситуация при которой, **Task A** имеющий более высокий приоритет, чем **Task B** ожидает завершения его рабо он первым захватил „жетон» (мьютекс).

Тупик(deadlock) — это ситуация при которой 2 таска ожидают друг друга т.к. каждый должен отдать "жетон» нужный другому. Для пример

1. Task A выполняется и захватывает «жетон» X.
2. Выполнение работы Task A прерывается Task B.
3. Task B захватывает «жетон» Y, перед этим пытаясь захватить «жетон» X, который сейчас используется Task A. В результате Task B впа ожидание Task A.
4. Task A продолжает выполняться, и хочет захватить «жетон» Y, который используется Task B и в результате тоже впадает в ожидание.

Для того чтобы избежать «тупиков» необходимо чтобы все таски получали «жетон» в определенном порядке, например, сначала таск A, пот т.д.

Пожалуй, на этом все, интересно было бы услышать поправки по всему циклу, дабы скорректировать статьи, и в дальнейшем ссылаться на н Для дальнейшего изучения я бы, прежде всего, посоветовал прочитать оригинальное руководство «Using the FreeRTOS real time kernel», информация из которого использовалась при написании статей, а также документацию на официальном сайте.

freertos, микроконтроллеры