* 报错1：

.\Objects\MP.sct(7): error: L6236E: No section matches selector - no section

原因：

没有找到合适的启动文件，增加启动文件后没有报错；

* 报错2：

FCARM - Output Name not specified, please check 'Options for Target - Utilities'

原因：

工程中存在文件的属性有误，在工程中添加了启动文件.s文件，但是文件属性为image文件，因此编译报错；

* 报错3：

warning: #223-D: function “assert\_param” declared implicitly

原因：

缺少宏定义：STM32F10X\_HD, USE\_STDPERIPH\_DRIVER，增加宏定义即可

* 问题4：

搭建完操作系统，建立起任务，操作系统不能正常进行任务工作，经调试，发现一直进入

B OSStartHang 的位置。

**原因：（参考：http://www.xiaovdiy.cn/?post=10）**

因为STM32本身的异常中断PendSV\_Handler替代了UCOS的异常中断OSPendSV，使其不能正常执行。

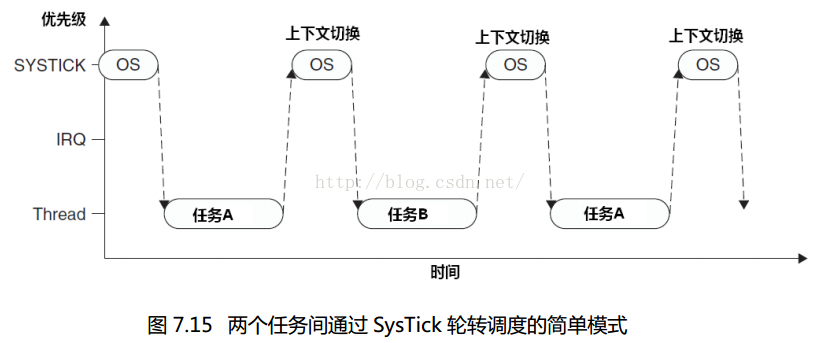
在stm32启动文件中，指定了两个硬件中断入口：



这两个函数有什么用呢？（参考：<http://blog.csdn.net/u012351051/article/details/50991007>）

在uCOS的移植中有两个函数是移植工作的核心，相当于将**裸机程序**与**操作系统**的“神经”搭在一起，使得操作系统得以运行，这两个系统中断分别为：

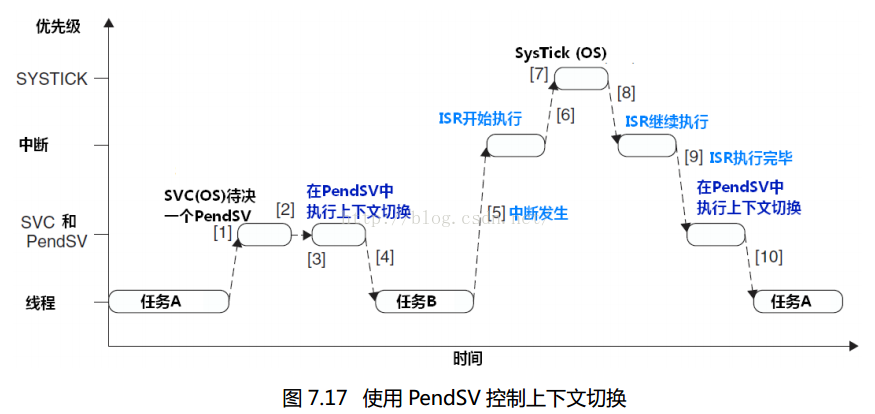
1. SysTick\_Handler：滴答定时器中断，这个中断相当于操作系统的心脏，给操作系统提供心跳，滴答定时器会一直不停的不断触发中断，在它的中断服务函数中，提供进程/任务的上下文切换，和任务调度的工作，简单描述如下图：



1. PendSV\_Handler：这是最简单的调度模式，如果在滴答定时器中断的过程中，发生了其

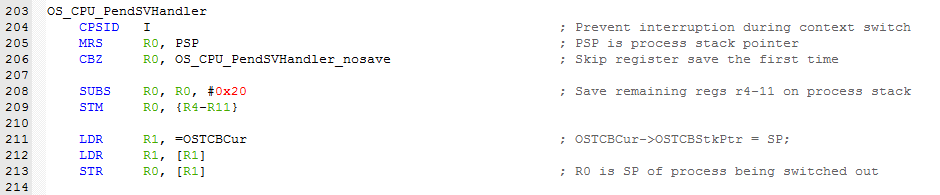
他的中断，该怎么办？很显然，对于一个实时操作系统，中断的优先级必然是高的，不能为了任务切换而延迟响应其他的中断，所以早期的操作系统大多检测当前是否有中断在活跃，只有在没有任何中断需要响应时，才执行上下文切换，显然这样是不太好的，毕竟中断是有突发性的。这样就引入了PendSV，叫可挂起系统中断。

PendSV的作用是会自动延迟上下文切换的请求，直到其他的ISR都完成后才会放行，其实它相当于上下文切换的缓冲作用，为了实现这个机制，需要将PendSV编程为最低优先级。当OS检测到某IRQ正在活动，并且被SysTick抢占，它将触发一个PendSV异常，以便缓期执行上下文切换，过程如下图所示：



**解决方式：**

系统任务切换延迟函数，定义在os\_cpu\_a.asm文件中；在启动文件中，将PendSVHandler中断函数入口，修改为OS\_CPU\_PendSVHandler, 同时修改相应的属性，即可解决。



修改方法：



