小型嵌入式操作系统的实现

写一个自己的操作系统是我一直以来的愿望，一来，学习，二来吗，装装X了。。哈哈

进过一段时间的学习，今天我写的代码终于实现了任务切换，也就是多任务环境了，虽然任务本身只是很简单那的闪烁你的LED小灯，但是仍然遮挡不住背后操作系统光辉的本质。

今天我决定，把我学习的经历分享出来，希望能起到抛砖引玉的作用。

一提到操作系统，大部分人的感觉是高端大气上档次，其实不然，仔细读完我的文章，你就能一步一步，从无到有，实现一个小型嵌入式操作系统，她可以实现基本的任务切换，也就是多任务了，以后的任务通信什么的，就很简单了。

我的硬件平台是STM32F103系列的单片机，这个很重要，因为嵌入式操作系统的核心就是任务切换，这段代码必须用汇编实现，大家都知道，汇编语言是平台相关语言，你用汇编语言在这个平台中点亮了一个LED灯，在另外一个平台中不能用时很平常的事情。

嵌入式操作系统最核心的地方就是任务切换，所以我会先从这方面开始讲起，

写这部分的时候，会用到很多底层和硬件平台相关的知识，用到什么内容我会在下面说明白。

1. 程序本质的剖析

写操作系统这个高端大气上档次的工作肯定要有一些铺垫了，最必须的就是对你写的程序的了解，也许你会说，我写的程序，我还能不理解吗，但是这次咱么要从寄存器角度分析。

咱们首先从类比学习开始，咱们先来理解中断，对于中断，学习单片机的小朋友们肯定很理解，咱么来一起回顾下，单片机是怎么用硬件实现中断的（更为具体的说明在Cortex-M3权威指南-carpter9中断的具体行为）其实中断就是多任务的环境了，只不过这个多任务环境只能有两个任务（在只有一个中断的前提下），那么只要咱么能模拟出来中断，那实现自己的操作系统也是很简单的呢。

CM3中断的一个完整过程由一下几个部分组成

1. 入栈
2. 取向量
3. 更新寄存器
4. 异常返回
5. 入堆栈

响应异常的第一个行动，就是自动保存现场的必要部分：依次把xPSR, PC, LR, R12以及R3‐R0由硬件自动压入适当的堆栈中，就是保存要切换出去的任务，因为下面就要开始执行中断这个任务了，如果不保存就无法实现这个任务的完全复原了。

1. 取向量

其实就是找到中断任务的入口地址，这样才能开始执行中断函数

1. 更新寄存器
2. 异常返回

当异常服务例程执行完毕后，需要很正式地做一个“异常返回”动作序列，从而恢复先前的系统状态，才能使被中断的程序得以继续执行

操作系统的任务切换也是大同小异

1. 屏蔽所有中断
2. 保存正在执行的任务的寄存器信息到任务独立的堆栈中
3. 从要投入运行的任务的堆栈中取出数据到寄存器中
4. 取消中断屏蔽

经过这四步，上一个任务已经被保存起来，等待下一次的运行，要运行的任务已经开始了运行

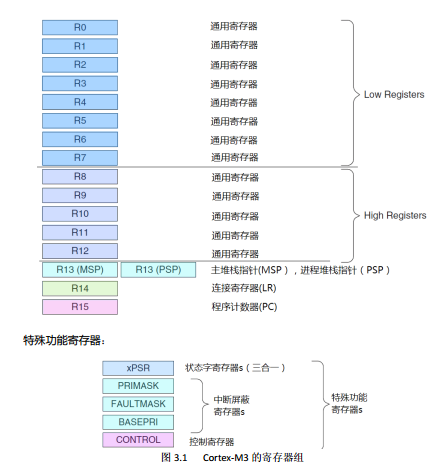
上面这四步只是一个大体的概述

在对CM3内核的实现描述前有一些准备知识

1. CM3寄存器基础（在Cortex-M3权威指南一书）
2. BASEPRI寄存器，用于中断屏蔽（在Cortex-M3权威指南一书）
3. 线程模式和handler模式，在保存上下文时用（在Cortex-M3权威指南一书）
4. 特权级和用户级，明白在Systick中断时的情况（在Cortex-M3权威指南一书）
5. PendSV异常，在这个异常中进行任务切换（在Cortex-M3权威指南一书）
6. SVC异常，启动OS，开始执行第一个任务就是通过呼叫SVC异常（在Cortex-M3权威指南一书）
7. MSP和PSP双堆栈指针的使用，保存寄存器时用（在Cortex-M3权威指南一书）
8. 中断控制及状态寄存器ICSR，知道如何触发PendSV异常（在Cortex-M3权威指南一书）
9. 向量表偏移量寄存器VTOR，第一次切入任务的（在Cortex-M3权威指南一书）
10. 向量表结构，得到MSP的初始值（在Cortex-M3权威指南一书）
11. 系统异常优先级寄存器，用于设置PendSV异常和Systick异常的优先级（在Cortex-M3权威指南一书）

下面详细说明上述几点

1. CM3寄存器基础



1. BASEPRI寄存器，用于中断屏蔽（在Cortex-M3权威指南一书）

在更精巧的设计中，需要对中断掩蔽进行更细腻的控制——只掩蔽优先级低于某一阈值的中断——它们的优先级在数字上大于等于某个数。那么这个数存储在哪里？就存储在BASEPRI中。不过，如1果往BASEPRI中写0，则另当别论——BASEPRI将停止掩蔽任何中断。例如，如果你需要掩蔽所有优先级不高于0x60的中断，则可以如下编程：

MOV R0, #0x60  
MSR BASEPRI, R0

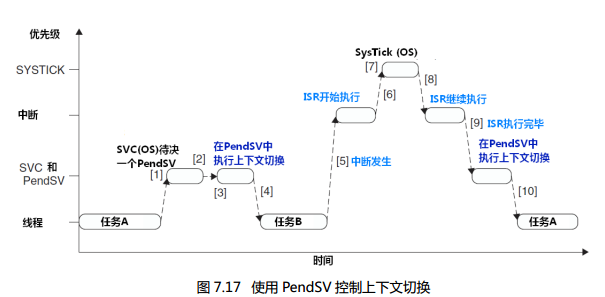
如果需要取消 BASEPRI 对中断的掩蔽，则示例代码如下：

MOV R0, #0  
MSR BASEPRI, R0

1. 线程模式和handler模式
2. 特权级和用户级



1. PendSV异常

试想一个这个过程一个ISR正在执行SysTick 异常会抢占其 ISR，在这时OS 不得执行上下文切换，否则将使中断请求被延迟，而且在真实系统中延迟时间还往往不可预知——任何有一丁点实时要求的系统都决不能容忍这种事。因此，在 CM3 中也是严禁没商量——如果 OS 在某中断活跃时尝试切入线程模式，将触犯用法 fault 异常。现在好了，PendSV 来完美解决这个问题了。PendSV 异常会自动延迟上下文切换的请求，  
直到其它的 ISR 都完成了处理后才放行。为实现这个机制，需要把 PendSV 编程为最低优先级的异常。如果 OS 检测到某 IRQ 正在活动并且被 SysTick 抢占，它将悬起一个 PendSV 异常，  
以便缓期执行上下文切换。

1. SVC异常

SVC 用于产生系统函数的调用请求。例如，操作系统不让用户  
程序直接访问硬件，而是通过提供一些系统服务函数，用户程序使用 SVC 发出对系统服务函  
数的呼叫请求，以这种方法调用它们来间接访问硬件。

1. MSP和PSP双堆栈指针的使用

一般情况下

在线程模式下使用PSP，在handler模式下使用MSP

所以在进行任务切换的时候，只需要把通用寄存器数据压入任务的私有堆栈。

在异常的时候，只能使用MSP堆栈指针，任务切换又是在PendSV异常中进行的，

所以进入PnedSV异常的时候，

1. 先把通用寄存器的内容保存到要切换出去的任务的私有堆栈（这是保存上文），
2. 保存通用寄存器到主堆栈，
3. 屏蔽所有中断，进入临界区
4. 调用C语言函数进行切换当前任务的TCB指针，
5. 返回到异常汇编函数中
6. 解除中断屏蔽
7. 从主堆栈中恢复数据到通用寄存器，
8. 从要切入任务的私有堆栈中恢复数据到通用寄存器
9. 退出异常
10. 中断控制及状态寄存器ICSR

ICSR的第28位是读写类型，向这个位写1就可以实现悬起PendSV异常

1. 向量表偏移量寄存器VTOR

把这个作为地址从中取出的就是向量表的第一块内容

1. 向量表结构

向量表的第一块内容是MSP 的初始值

1. 系统异常优先级寄存器

PendSV异常和Systick异常在操作系统中，应该设成最低，

通过这两个寄存器改变这两个异常的优先级

应该修改成0xf0

下面对于CM3这个内核说一下详细的实现步骤

咱们先从简单的来，加入现在你写了两个函数

并且有一个任务切换函数

void TaskSwitch(void);

void Task0(void)

{

while(1)

{

//do something task

//实现任务的主动切换，就是把当前任务切换出去把另一个任务切换进去

TaskSwitch();

}

}

void Task1(void)

{

while(1)

{

//do something task

//实现任务的主动切换，就是把当前任务切换出去把另一个任务切换进去

TaskSwitch();

}

}

在main函数中调用Task0函数，实现手动启动Task0,这就进入了任务切换的循环了，那么TaskSwitch怎么实现了，下面开始进入重点，开始一步一步说明，如何实现这个函数。

这里有一个前提

/\* 当前任务控制块的指针 \*/

OS\_TCB \* p\_OS\_TCB\_Current;

/\* 高优先级任务控制块指针 \*/

OS\_TCB \* p\_OS\_HighPriTCB\_Current;

首先先说一下TaskSwitch函数中实现了什么

1. 屏蔽中断，进入临界区
2. 根据相应的算法计算下一个应该切入的任务是那个，咱么这里很简单，

如果正在执行任务0，那么切换到任务1，

如果正在执行任务1，那么切换到任务0，

这就实现了最简单的任务切换。

1. 把p\_OS\_HighPriTCB\_Current指向要切入的函数
2. 触发PendSV异常
3. 解除中断屏蔽，退出临界区

这块的任务C语言就可以实现，但是用汇编写效率可能会更高

下面开始演示

1. 屏蔽中断，进入临界区

这里就要利用上面说的准备的知识了---BASEPRI寄存器，因为用的是一个八位寄存器的高四位作为优先级，这里只要把一个0x10的数写入BASEPRI寄存器，就可以实现屏蔽所有的中断。

1. 根据相应的算法计算下一个应该切入的任务是那个，咱么这里很简单，

如果正在执行任务0，那么切换到任务1，

如果正在执行任务1，那么切换到任务0，

这就实现了最简单的任务切换。

1. 把p\_OS\_HighPriTCB\_Current指向要切入的函数

用C语言即可实现

1. 触发PendSV异常

ICSR的第28位是读写类型，向这个位写1就可以实现悬起PendSV异常

1. 解除中断屏蔽，退出临界区

如果往BASEPRI中写0，则——BASEPRI将停止掩蔽任何中断

完成这几步就完成了任务切换，最基本的多任务环境就实现了。

具体的代码见---------- SHIOSV1能主动进行任务切换---文件