

中断嵌套是指在执行一个中断服务程序时，如果发生了另一个更高优先级的中断，那么当前的中断服务程序会被暂停，转而去处理这个更高优先级的中断。处理完毕后，之前被暂停的中断服务程序会继续执行。

这个过程可以想象成一个大中断嵌套着多个小中断。当大中断发生时，所有的小中断都会被暂停，转去执行大中断处理程序。等大中断处理完毕后，之前被暂停的小中断会继续执行。

需要注意的是，在某些系统中，高优先级的中断可以打断低优先级的中断，但低优先级的中断不能打断高优先级的中断。这种机制可以保证系统在紧急情况下能够迅速响应，同时也能避免不同中断之间的冲突和干扰。

另外，在某些嵌入式系统中，中断嵌套可能会被禁用或者受到限制。这种情况下，如果发生了高优先级的中断，低优先级的中断可能会被忽略或者延迟处理。因此，在设计系统时，需要根据实际情况来确定是否需要支持中断嵌套功能。

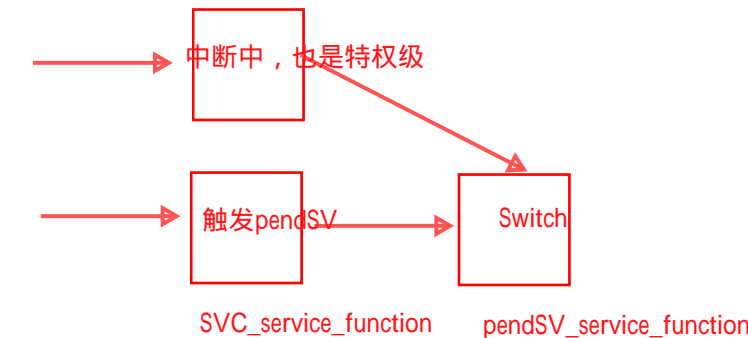
1. 执行任务调度的异常A优先级必须比其他异常、中断要低。（因为A抢断其他中断OtherIS，执行任务切换到线程态，会发生系统错误；）
2. 所以，A的优先级必须低。
3. 但是，另一个问题，A触发时候，若有其他中断在发生，任务切换不会响应（低优先级中断被放弃），会延时一个周期再次进行触发，若多个中断与之“共振”，系统切换实现不了；（另外：A被抢占的话也会有问题，所以，A执行时候会关闭中断。）
4. 为了避免3中的问题，可以在pendSVC中执行任务调度功能，则：遇到中断进行时候，悬起该异常，等otherIS执行完成后，再执行任务调度，保证每次都会响应任务调度；

？为什么用SVC？？？
这涉及到ARM的特权级及线程级对异常的访问功能限制，

AI应答：
是的，如果用户级线程直接访问 PendSV 寄存器，可能会导致 fault（故障）或异常。PendSV 寄存器是 ARM 架构中用于实现系统调用的特殊寄存器，它是特权级别的寄存器，只能在操作系统内核中访问。

如果用户级线程直接访问 PendSV 寄存器，这将被视为非法访问，因为用户级线程没有权限直接操作特权级别的寄存器。这可能会导致硬件故障或操作系统内核的干预，从而导致系统不稳定或崩溃。

因此，为了确保系统的稳定性和安全性，用户级线程应该通过标准的系统调用接口来请求操作系统内核的服务，而不是直接访问特权级别的寄存器



关于ARM的 特权级 和 线程级：

什么是PendSV

转载

sean-zhao

于 2015-03-12 21:49:27 发布


17540

收藏 106

版权

分类专栏:

ucosII

ucosII 专栏收录该内容

0 订阅 3 篇文章 订阅专栏

原文： http://www.cnblogs.com/sky1991/p/stepbystep_stm32_os_3.html

一、什么是PendSV

PendSV是可悬起异常，如果我们把它配置最低优先级，那么如果同时有多个异常被触发，它会在其他异常执行完后再执行，而且任何异常都可以中断它。更详细的内容在《Cortex-M3 权威指南》里有介绍，下面我摘抄了一段。

OS 可以利用它“缓期执行”一个异常——直到其它重要的任务完成后才执行动作。悬起 PendSV 的方法是：手工往 NVIC 的 PendSV悬起寄存器中写 1。悬起后，如果优先级不够 高，则将缓期待待执行。

- PendSV的典型使用场合是在上下文切换时（在不同任务之间切换）。例如，一个系统中有两个就绪的任务，上下文切换被触发的场合可以是：
- 1、执行一个系统调用
 - 2、系统滴答定时器（SYSTICK）中断，（轮转调度中需要）

让我们举个简单的例子来帮助理解。假设有这么一个系统，里面有两个就绪的任务，并且通过SysTick异常启动上下文切换。但若在产生 SysTick 异常时正在响应一个中断，则 SysTick异常会抢占其 ISR。在这种情况下，OS是不能执行上下文切换的，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。

为解决此问题，早期的 OS 大多会检测当前是否有中断在活跃中，只有在无任何中断需要响应 时，才执行上下文切换（切换期间无法响应中断）。然而，这种方法的弊端在于，它可以把任务切换动作拖延很久（因为如果抢占了 IRQ，则本次 SysTick在执行后不得作上下文切换，只能等待下一次SysTick异常），尤其是当某中断源的频率和SysTick异常的频率比较接近时，会发生“共振”，使上下文切换迟迟不能进行。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。

使用 PendSV 控制上下文切换个中事件的流水账记录如下：

1. 任务 A呼叫 SVC来请求任务切换（例如，等待某些工作完成）
2. OS接收到请求，做好上下文切换的准备，并且悬起一个 PendSV异常。
3. 当 CPU退出 SVC后，它立即进入 PendSV，从而执行上下文切换。
4. 当 PendSV执行完毕后，将返回到任务 B，同时进入线程模式。
5. 发生了一个中断，并且中断服务程序开始执行
6. 在 ISR执行过程中，发生 SysTick异常，并且抢占了该 ISR。
7. OS执行必要的操作，然后悬起 PendSV异常以作好上下文切换的准备。
8. 当 SysTick退出后，回到先前被抢占的 ISR中，ISR继续执行
9. ISR执行完毕并退出后，PendSV服务例程开始执行，并且在里面执行上下文切换
10. 当 PendSV执行完毕后，回到任务 A，同时系统再次进入线程模式。

我们在uCOS的PendSV的处理代码中可以看到：

```
1 OS_CPU_PendSVHandler
2     CPSID I ; 关中断
3     ;保存上文
4     ;.....
5     ;切换下文
6     CPSIE I ;开中断
7     BX LR ;异常返回
```

它在异常一开始就关闭了中端，结束时开启中断，中间的代码为临界区代码，即不可被中断的操作。PendSV异常是任务切换的堆栈部分的核心，由他来完成上下文切换。PendSV的操作也很简单，主要有设置优先级和触发异常两部分：

```
1 NVIC_INT_CTRL EQU 0xE000ED04 ; 中断控制寄存器
2 NVIC_SYSPRI14 EQU 0xE000ED22 ; 系统优先级寄存器(优先级14)。
3 NVIC_PENDSV_PRI EQU 0xFF ; PendSV优先级(最低)。
4 NVIC_PENDSVSET EQU 0x10000000 ; PendSV触发值
5
6 ; 设置PendSV的异常中断优先级
7
8 LDR R0, =NVIC_SY
9 LDR R1, =NVIC_PE
```

sean-zhao

关注

24

106

0


```
10 | STRB R1, [R0] ; 触发PendSV异常
11 | LDR R0, =NVIC_INT_CTRL
12 | LDR R1, =NVIC_PENDSVSET
13 | STR R1, [R0]
```

二、堆栈操作

Cortex M4有两个堆栈寄存器，主堆栈指针（MSP）与进程堆栈指针（PSP），而且任一时刻只能使用其中的一个。MSP为复位后缺省使用的堆栈指针，异常永远使用MSP，如果手动开启PSP，那么线程使用PSP，否则也使用MSP。怎么开启PSP？

```
1 | MSR PSP, R0 ; Load PSP with new process SP
2 | ORR LR, LR, #0x04 ; Ensure exception return uses proc
```

很容易就看出来了，置LR的位2为1，那么异常返回后，线程使用PSP。

写OS首先要将内存分配搞明白，单片机内存本来就很小，所以我们当然要斤斤计较一下。在OS运行之前，我们首先要初始化MSP和PSP，OS_CPU_ExceptStkBase是外部变量，假如我们给主堆栈分配1KB（256*4）的内存即OS_CPU_ExceptStk[256],则OS_CPU_ExceptStkBase=&OS_CPU_ExceptStk[256-1]。

```
1 | EXTERN OS_CPU_ExceptStkBase
2 | ;PSP清零，作为首次上下文切换的标志
3 | MOVS R0, #0
4 | MSR PSP, R0
5 | ;将MSP设为我们为其分配的内存地址
6 | LDR R0, =OS_CPU_ExceptStkBase
7 | LDR R1, [R0]
8 | MSR MSP, R1
```

然后就是PendSV上下文切换中的堆栈操作了，如果不使用FPU，则进入异常自动压栈xPSR, PC, LR, R12, R0-R3，我们还要把R4-R11入栈。如果开启了FPU，自动压栈的寄存器还有S0-S15，还需吧S16-S31压栈。

```
1 | MRS R0, PSP
2 | SUBS R0, R0, #0x20 ;压入R4-R11
3 | STM R0, {R4-R11}
4 |
5 | LDR R1, =Cur_TCB_Point ;当前任务的指针
6 | LDR R1, [R1]
7 | STR R0, [R1] ; 更新任务堆栈指针
```


出栈类似，但要注意顺序

```
1 | LDR R1, =TCB_Point ;要切换的任务指针
2 | LDR R2, [R1]
3 | LDR R0, [R2] ; R0为要切换的任务堆栈地址
4 |
5 | LDM R0, {R4-R11} ; 弹出R4-R11
6 | ADDS R0, R0, #0x20
7 |
8 | MSR PSP, R0 ;更新PSP
```

三、OS实战

新建os_port.asm文件，内容如下：

```
1 | NVIC_INT_CTRL EQU 0xE000ED04 ; Interrupt control state reg
2 | NVIC_SYSPRI14 EQU 0xE000ED22 ; System priority register (p
3 | NVIC_PENDSV_PRI EQU 0xFF ; PendSV priority value (lowe
4 | NVIC_PENDSVSET EQU 0x10000000 ; Value to trigger PendSV exc
5 |
6 | RSEG CODE:CODE:NOROOT(2)
7 | THUMB
8 |
9 |
10 | EXTERN g_OS_CPU_ExceptStkBase
11 |
12 | EXTERN g_OS_Tcb_CurP
13 | EXTERN g_OS_Tcb_HighRdyP
14 |
15 | PUBLIC OSStart_Asm
16 | PUBLIC PendSV_Handler
17 | PUBLIC OSCtxSw
18 |
19 | OSCtxSw
20 | LDR R0, =NVIC_INT_CTRL
21 | LDR R1, =NVIC_PENDSVSET
22 | STR R1, [R0]
23 | BX LR ; Enable interrupts at processo
24 |
25 | OSStart_Asm
26 | LDR R0,
```

 **sean-zhao** 关注

 24   106  0 

```
27      LDR      R1, =NVIC_PENDSV_PRI28      |      STRB      R1, [R0]
29
30      MOVS      R0, #0                      ; Set the PSP to 0 for initia
31      MSR      PSP, R0
32
33      LDR      R0, =g_OS_CPU_ExceptStkBase      ; Initialize the MSP to the
34      LDR      R1, [R0]
35      MSR      MSP, R1
36
37      LDR      R0, =NVIC_INT_CTRL      ; Trigger the PendSV exceptio
38      LDR      R1, =NVIC_PENDSVSET
39      STR      R1, [R0]
40
41      CPSIE      I                      ; Enable interrupts at proces
42
43 OSStartHang
44      B      OSStartHang      ; Should never get here
45
46
47
48 PendSV_Handler
49      CPSID      I                      ; Prevent interruption during
50      MRS      R0, PSP      ; PSP is process stack pointe
51      CBZ      R0, OS_CPU_PendSVHandler_nosave      ; Skip register save the first
52
53      SUBS      R0, R0, #0x20      ; Save remaining regs r4-11 o
54      STM      R0, {R4-R11}
55
56      LDR      R1, =g_OS_Tcb_CurP      ; OSTCBCur->OSTCBStkPtr
57      LDR      R1, [R1]
58      STR      R0, [R1]      ; R0 is SP of process being s
59
60      ; At this point, entire conte
61 OS_CPU_PendSVHandler_nosave
62      LDR      R0, =g_OS_Tcb_CurP      ; OSTCBCur = OSTCBHighR
63      LDR      R1, =g_OS_Tcb_HighRdyP
64      LDR      R2, [R1]
65      STR      R2, [R0]
66
67      LDR      R0, [R2]      ; R0 is new process SP; SP = OSTCB
68
69      LDM      R0, {R4-R11}      ; Restore r4-11 from new proc
70      ADDS      R0, R0, #0x20
71
72      MSR      PSP, R0      ; Load PSP with new process S
73      ORR      LR, LR, #0x04      ; Ensure exception return use
74
75      CPSIE      I
76      BX      LR      ; Exception return will resto
77
78      END
```

main.c内容如下：

```
1  #include "stdio.h"
2  #define OS_EXCEPT_STK_SIZE 1024
3  #define TASK_1_STK_SIZE 1024
4  #define TASK_2_STK_SIZE 1024
5
6  typedef unsigned int OS_STK;
7  typedef void (*OS_TASK)(void);
8
9  typedef struct OS_TCB
10 {
11     OS_STK *StkAddr;
12 }OS_TCB,*OS_TCBP;
13
14
15 OS_TCBP g_OS_Tcb_CurP;
16 OS_TCBP g_OS_Tcb_HighRdyP;
17
18 static OS_STK OS_CPU_ExceptStk[OS_EXCEPT_STK_SIZE];
19 OS_STK *g_OS_CPU_ExceptStkBase;
20
21 static OS_TCB TCB_1;
22 static OS_TCB TCB_2;
23 static OS_STK TASK_1_STK[TASK_1_STK_SIZE];
24 static OS_STK TASK_2_STK[TASK_2_STK_SIZE];
25
26 extern void OSStart_Asm(void);
27 extern void OSCtxSw(void);
28
29 void Task_Switch()
30 {
31     if(g_OS_Tcb_CurP == &TCB_1)
32         g_OS_Tcb_HighRdyP=&TCB_2;
33     else
34         g_OS_Tcb_HighRdyP=&TCB_1;
35
36     OSCtxSw();
37 }
38
39
40 void task_1()
41 {
42     printf("Task
```

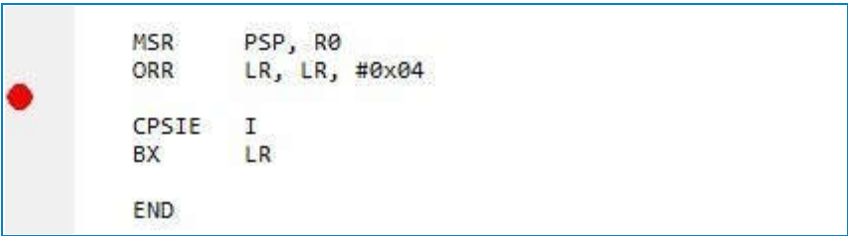
 **sean-zhao** 关注

 24   106  0 

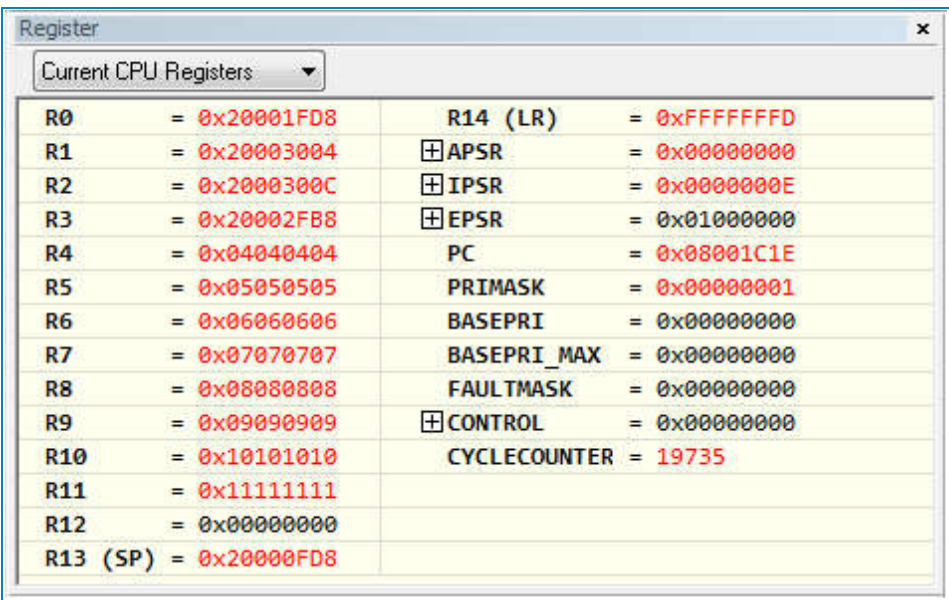
```
43 | Task_Switch(); 44 | printf("Task 1 Running!!!\n");
45 | Task_Switch();
46 | }
47 |
48 | void task_2()
49 | {
50 |
51 |     printf("Task 2 Running!!!\n");
52 |     Task_Switch();
53 |     printf("Task 2 Running!!!\n");
54 |     Task_Switch();
55 | }
56 |
57 | void Task_End(void)
58 | {
59 |     printf("Task End\n");
60 |     while(1)
61 |     {}
62 | }
63 |
64 | void Task_Create(OS_TCB *tcb,OS_TASK task,OS_STK *stk)
65 | {
66 |     OS_STK *p_stk;
67 |     p_stk = stk;
68 |     p_stk = (OS_STK *)((OS_STK)(p_stk) & 0xFFFFFFFF8u);
69 |
70 |     * (--p_stk) = (OS_STK)0x01000000uL; //xPSR
71 |     * (--p_stk) = (OS_STK)task; // Entry Point
72 |     * (--p_stk) = (OS_STK)Task_End; // R14 (LR)
73 |     * (--p_stk) = (OS_STK)0x12121212uL; // R12
74 |     * (--p_stk) = (OS_STK)0x03030303uL; // R3
75 |     * (--p_stk) = (OS_STK)0x02020202uL; // R2
76 |     * (--p_stk) = (OS_STK)0x01010101uL; // R1
77 |     * (--p_stk) = (OS_STK)0x00000000u; // R0
78 |
79 |     * (--p_stk) = (OS_STK)0x11111111uL; // R11
80 |     * (--p_stk) = (OS_STK)0x10101010uL; // R10
81 |     * (--p_stk) = (OS_STK)0x09090909uL; // R9
82 |     * (--p_stk) = (OS_STK)0x08080808uL; // R8
83 |     * (--p_stk) = (OS_STK)0x07070707uL; // R7
84 |     * (--p_stk) = (OS_STK)0x06060606uL; // R6
85 |     * (--p_stk) = (OS_STK)0x05050505uL; // R5
86 |     * (--p_stk) = (OS_STK)0x04040404uL; // R4
87 |
88 |     tcb->StkAddr=p_stk;
89 | }
90 |
91 |
92 | int main()
93 | {
94 |
95 |     g_OS_CPU_ExceptStkBase = OS_CPU_ExceptStk + OS_EXCEPT_STK_SIZE - 1;
96 |
97 |     Task_Create(&TCB_1,task_1,&TASK_1_STK[TASK_1_STK_SIZE-1]);
98 |     Task_Create(&TCB_2,task_2,&TASK_2_STK[TASK_1_STK_SIZE-1]);
99 |
100 |     g_OS_Tcb_HighRdyP=&TCB_1;
101 |
102 |     OSStart_Asm();
103 |
104 |     return 0;
105 | }
```

编译下载并调试：

在此处设置断点



此时寄存器的值，可以看到R4-R11正是我们给的值，单步运行几次，可以看到进入了我们的任务task_1或task_2，任务里打印信息，然后调用Task_Switch进行切换，OSCtxSw触发PendSV异常。




IO输出如下：



至此我们成功实现了使用PenSV进行两个任务的互相切换。之后，我们使用使用SysTick实现比较完整的多任务切换。

中断及pendSV	难波儿万的博客	539
通过任务及任务切换一节读者已经了解了任务切换的详细过程，其实要实现任务切换的功能前面讲的还远远不够。因为，P...		
ucos iii stm32移植MDK编译		11-11
根据官方stm32f107例程修改，去掉了bsp，改为MDK编译环境。运行正常		
cm0中断优先级_CM3任务切换详解 - coreyggj的个人空间 - OSCHINA...		9-9
PendSV叫做可悬起系统调用,与之相对的叫做系统服务调用(SVC)。 两者的区别是SVC异常必须在执行SVC指令后立即得到...		
嵌入式面试之实时系统技术沟通(三)_实时系统面试_嵌入式知行合一的博...		9-17
(1)PendSV中断:用于实现RTOS上下文切换的中断 (2)原因:如果直接在RTOS时基定时器中断进行上下文切换,当该时器中断...		
怎样去理解异常SVC和PendSV	lunei	3873
什么是SVC和PendSV SVC（系统服务调用）和 PendSV（可悬挂系统调用）。它们多用于在操作系统之上的软件开发中。...		
STM32中断基础知识笔记	weixin_42490599的博客	24
配置好中断后如果有触发，即会进入中断服务函数，那么中断服务函数也有固定的函数名，可以在 startup_stm32f10x_hd.s...		
十五、中断管理_冲向大厂搬砖的博客		9-16
PendSV中断服务程序的作用:保存当前任务的现场、挑出当前优先级最高的就绪任务(在前面说到的链表中找,实际直接调用V...		
FreeRTOS 任务切换_freertos任务切换原理_比特冬哥的博客		9-21
一、PendSV 异常 PendSV(可挂起的系统调用)异常对 OS 操作非常重要,其优先级可以通过编程设置。可以通过将中断控制...		
Nu_LB_NUC140_PendSV.rar		02-19
Nu_LB_NUC140_PendSV.rar ARM Cortex-M0权威指南(中文) 高清扫描版.pdf 移植的例子		
一步步写STM32 OS【三】PendSV与堆栈操作		08-03
假设有这么一个系统，里面有两个就绪的任务，并且通过SysTick异常启动上下文切换。但若在产生 SysTick 异常时正在响...		
一图理解M0不同优先级中断及Pendsv切换_大吉机器人的博客		9-15
一步步写STM32 OS【三】PendSV与堆栈操作 08-03 假设有这么一个系统,里面有两个就绪的任务,并且通过SysTick异常启...		
...处理器内部寄存器分析_stm32中svc和 pendsv中断		9-19
1、PendSV可以像普通的中断一样被悬起,而不会像SVC那样不被立即执行就会升级上访到硬fault, 操作系统可利用它能缓期...		
Cortex-M3 PendSV 中断 系统调用 说明	吾生也有涯，而知也无涯	7386
参考 Cortex-M3权威指南中文版 PendSV异常是和系统调用有些类似，cpu 需要手动将往NVIC 的PendSV 悬起寄存器中写1...		
有了Systick中断为什么还要PendSV中断？	wcc243588569的博客	5826
文章目录问题：原因：1.在SysTick中断里完成任务切换会降低操作系统的实时性：问题：看过Cortex-m3/m4操作系统RTO...		
cortex-M3 的SVC、PendSV异常，与操作系统(ucos实时系统) 热门推荐	@角色扮演#	2万+
SVC异常是？ PendSV异常是？ ucos 任务切换时机？ ucos 如何满足实时性(实现)？ ucos中，systick的优先级？ SVC和Pe...		
2.PendSV的触发	huhuandk的博客	1508
PendSV典型使用场合是在上下文切换时（在不同任务之间切换）。我们先简单的写几段代码实现PendSV的中断触发，当...		
【FreeRTOS】3. PendSV异常	Ethan-Code's blog	1720
PendSV异常，解析三个问题：怎么触发PendSV异常？何时使用MSP何时切换PSP？ PendSV如何实现上下文切换？ 1.触...		
移植LiteOS到STM32		12-15
华为LiteOS移植程序，LiteOS移植到STM32F103R8T6，已成功		
一步步写STM32 OS		03-15
基于STM32，讲解任务切换，详细给出汇编参考，并完成一个简单的操作系统，实现了任务调度，对于OS的入门非常有好...		
浅谈FreeRTOS任务启动与切换流程	飞临云的博客	2616
一个轻量级操作系统最核心的地方就在于任务的执行与切换，像FreeRTOS和ucOSIII在任务启动与切换方面都差不多，本...		
UCOSIII（1）——SVC与PenSV实现任务切换	C~Tian	780
本文基于STM32F407ZGT6————— SVC异常： SVC(系统服务调用，亦简称系统调用)用于产生系...		
RTOS系列文章（2）： PendSV功能，为什么需要PendSV	猪哥的专栏	8872
背景大多数嵌入式RTOS在Cortex-M3/M4上的移植都需要PendSV，比如uCOS、RT-Thread、FreeRTOS等，本文就对Pen...		
STM32的"异常"、“中断”和“事件”区别和理解	胡图图	6809
1 异常与中断（Cortex-M3）1.1 异常与中断原话： Cortex-M3 在内核水平上搭载了一个异常响应系统，支持为数众多的系...		
手写RTOS-PendSV中断	Mr_Define的博客	1919
今天这一篇，我们说一下操作系统都要用到的PendSV中断，整个操作系统中，要自己写的的汇编代码不超过20行，全部都...		
STM32F103 FreeRTOS任务启动与切换流程	billionguy的博客	1606
SVC_Handler是用于启动第一个任务的中断； PendSV_Handler是用于每次任务切换中断； SysTick_Handler是一个定时器...		
PendSV_Handler 最新发布		07-27
PendSV_Handler是ARM Cortex-M处理器中的一种中断处理函数。它是特殊的系统中断处理函数，用于处理与任务切换相...		

“相关推荐”对你有帮助么？

-  非常没帮助
-  没帮助
-  一般
-  有帮助
-  非常有帮助

关于我们 招贤纳士

 sean-zhao

关注

 24   106  0