# μC/OS-II 在 STM32 上移植学习

主要学习 micrium 应用笔记 AN-1018
需要移植的文件：
OS_CPU.H
OS_CPU_C.C
OS_CPU_A.ASM
//OS_DBG.C

1.OS_CPU.H
访问临界代码方法 OS_CRITICAL_MENTHOD#3
笔记中的移植用了 OS_CRITICAL_MENTHOD#3 来访问临界代码。
******************************************************************************

```
#define    OS_CRITICAL_METHOD     3

#if OS_CRITICAL_METHOD == 3
#define    OS_ENTER_CRITICAL()    {cpu_sr = OS_CPU_SR_Save();}
#define    OS_EXIT_CRITICAL()     {OS_CPU_SR_Restore(cpu_sr);}
#endif
```

******************************************************************************
以上是相关的程序片段。如果应用程序中用了这两个宏，那么要定义一个局部变量并初始化为 0，如 OS_CPU_SR cpu_sr = 0;
那 OS_CPU_SR_Save()和 OS_CPU_SR_Restore()具体做了什么呢？
******************************************************************************

```
OS_CPU_SR_Save
    MRS       R0, PRIMASK                                        ; Set prio int mask
to mask all (except faults)
    CPSID     I
    BX        LR

OS_CPU_SR_Restore
    MSR       PRIMASK, R0
    BX        LR
```

******************************************************************************
以上是 OS_CPU_SR_Save()和 OS_CPU_SR_Restore()程序片段。
Cortex-M3 中断屏蔽寄存器组：PRIMASK, FAULTMASK, BSAEPRI
PRIMASK 中的 bit0 置位后将屏蔽所有可配置优先级的中断。
MRS       R0, PRIMASK ;保存了 PRIMASK 的值
CPSID     I           ;关中断(可配置优先级的中断)。
OS_CPU.H 中其它的都比较简单。

2.OS_CPU_C.C
OSTaskStkInit ()
Cortex-M3 中的 μC/OS-II 的任务栈结构：

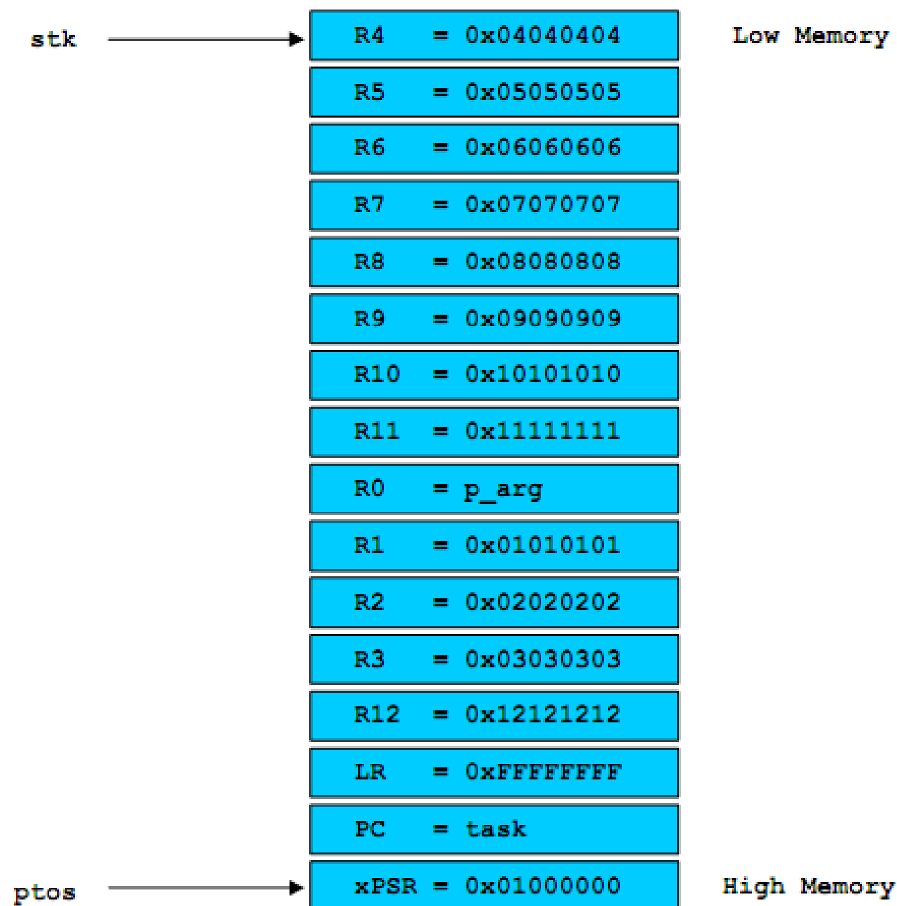



**Figure 3-2, The Stack Frame for each Task for ARM Cortex-M3 port.**

在 OS_CPU.H 中定义任务栈压栈方向是递减的：

```
#define   OS_STK_GROWTH   1      /* Stack grows from HIGH to LOW memory on ARM */
```

异常发生的时候，硬件会依次压栈 xPSR,PC,LR,r12,r3,r4,r1,r0 等 8 个寄存器。所以堆栈的高处是 xPSR,PC,LR,R12,R3,R2,R1,R0。R4-R11 由程序来压栈、弹栈。OSTaskStkInit 只要注意堆栈的顺序就可以了。

```
OS_STK *OSTaskStkInit (void (*task)(void *p_arg), void *p_arg, OS_STK *ptos, INT16U opt)
{
    OS_STK *stk;
    (void)opt;                          /*'opt' is not used, prevent warning                 */
    stk = ptos;                         /*Load stack pointer                                 */

                                        /* Registers stacked as if auto-saved on exception*/
    *(stk)=(INT32U)0x01000000L;         /*xPSR                                               */
    *(--stk)=(INT32U)task;              /* Entry Point                                       */
    *(--stk) = (INT32U)0xFFFFFFFEL;     /*R14 (LR) (init value will cause fault if ever used)*/
    *(--stk)=(INT32U)0x12121212L;       /*R12                                                */
    *(--stk)=(INT32U)0x03030303L;       /*R3                                                 */
    *(--stk)=(INT32U)0x02020202L;       /*R2                                                 */
    *(--stk)=(INT32U)0x01010101L;       /*R1                                                 */
```

```
    *(--stk)=(INT32U)p_arg;              /* R0 : argumen                                  */

                                         /* Remaining registers saved on process stack   */
    *(--stk)=(INT32U)0x11111111L;        /*R11                                          */
    *(--stk)=(INT32U)0x10101010L;        /*R10                                          */
    *(--stk)=(INT32U)0x09090909L;        /*R9                                           */
    *(--stk)=(INT32U)0x08080808L;        /*R8                                           */
    *(--stk)=(INT32U)0x07070707L;        /*R7                                           */
    *(--stk)=(INT32U)0x06060606L;        /*R6                                           */
    *(--stk)=(INT32U)0x05050505L;        /*R5                                           */
    *(--stk)=(INT32U)0x04040404L;        /*R4                                           */

    return (stk);
}
```

OS_CPU_C.中另外两个比较重要的函数就是 OS_CPU_SysTickInit()和 OS_CPU_SysTickHandler()。OS_CPU_SysTickInit()初始化了 SysTick，OS_CPU_SysTickHandler()则是 SysTick 的中断服务函数。OS_CPU_C.C 中的其它函数都是些 HOOK。

3.OS_CPU_A.ASM

①OS_CPU_SR_Save()和 OS_CPU_SR_Restore()。

②OSStartHighRdy()

```
OSStartHighRdy
    LDR       R0, =NVIC_SYSPRI14        ;(1) Set the PendSV exception priority
    LDR       R1, =NVIC_PENDSV_PRI
    STRB      R1, [R0]

    MOVS      R0, #0                    ; (2)Set the PSP to 0 for initial context switch call
    MSR       PSP, R0

    LDR       R0, =OSRunning            ; (3)OSRunning = TRUE
    MOVS      R1, #1
    STRB      R1, [R0]

    LDR       R0, =NVIC_INT_CTRL        ; (4)Trigger the PendSV exception (causes context switch)
    LDR       R1, =NVIC_PENDSVSET
    STR       R1, [R0]

    CPSIE     I                         ;(5) Enable interrupts at processor level
```

在执行到 OSStartHighRdy()时，多任务执行的环境已经有了。OSStartHighRdy()就是触发 PendSV 中断，以使当前优先级最高的就绪状态的任务开始执行。

(1)设置 PendSV 的优先级，其中

```
    NVIC_SYSPRI14          EQU        0xE000ED22
```

NVIC_PENDSV_PRI EQU 0xFF

Cortex-M3 中有一组 System Handler Priority Registers 用来给 memory manage, bus fault, usage fault, debugmonitor,SVC, SysTick,PendSV 设置优先级。PendSV 的中断优先级寄存器位于 0xE000ED22，共 8 位。

(2)设置堆栈指针 PSP 为空,因为此时还没有任务在执行，所以没有被压栈的任务，故堆栈指针为空。

(3)置位 OSRunning。

(4)触发 PendSV 中断，其中

NVIC_INT_CTRL EQU 0xE000ED04

NVIC_PENDSVSET EQU 0x10000000

NVIC_INT_CTRL 是 NVIC 中的 Interrupt Control State Register,置位这个寄存器的第 28 位将触发 PendSV 中断。

(5)打开中断。CPSIE 是复位 PRIMASK 的 Bit0。

③OSCtxSw 和 OSIntCtxSw

OSCtxSw 和 OSIntCtxSw 都只是触发下 PendSV。

④OS_CPU_PendSVHandler

```
OS_CPU_PendSVHandler
    CPSID   I                           ; (1)Prevent interruption during context switch
    MRS     R0, PSP                     ; PSP is process stack pointer
    ; Skip register save the first time
    CBZ     R0, OS_CPU_PendSVHandler_nosave
    SUBS    R0, R0, #0x20               ; (2)Save remaining regs r4-11 on process stack
    STM     R0, {R4-R11}

    LDR     R1, =OSTCBCur               ;(3)OSTCBCur->OSTCBStkPtr = SP;
    LDR     R1, [R1]
    STR     R0, [R1]                    ; R0 is SP of process being switched out
    ; At this point, entire context of process has been saved

OS_CPU_PendSVHandler_nosave
    PUSH    {R14}                       ; (4)Save LR exc_return value
    LDR     R0, =OSTaskSwHook           ; OSTaskSwHook();
    BLX     R0
    POP     {R14}

    LDR     R0, =OSPrioCur              ; (5)OSPrioCur = OSPrioHighRdy;
    LDR     R1, =OSPrioHighRdy
    LDRB    R2, [R1]
    STRB    R2, [R0]
```

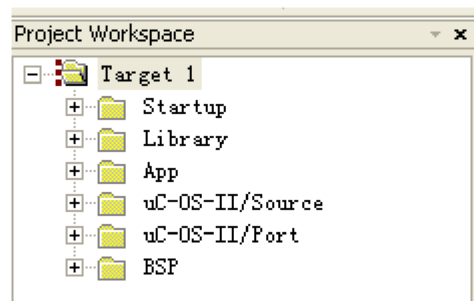
```
LDR     R0, =OSTCBCur          ; (6)OSTCBCur  = OSTCBHighRdy;
LDR     R1, =OSTCBHighRdy
LDR     R2, [R1]
STR     R2, [R0]

LDR     R0, [R2]               ; (7)R0 is new process SP; SP = OSTCBHighRdy->OSTCBStkPtr;
LDM     R0, {R4-R11}           ; (8)Restore r4-11 from new process stack
ADDS    R0, R0, #0x20
MSR     PSP, R0                ; Load PSP with new process SP
ORR     LR, LR, #0x04          ; Ensure exception return uses process stack
CPSIE   I
BX      LR                     ;(9) Exception return will restore remaining context
```

(1)先检查 PSP 是否为空。在 OSStartHighRdy 设置 PSP 为空。
(2)保存 R4-R11
(3)保存 SP 到 OSTCBCur 中
(4)调用 OSTaskSwHook
(5)OSPrioCur = OSPrioHighRdy
(6)OSTCBCur = OSTCBHighRdy
(8)恢复 R4-R11

4. OS_DBG.C
不知道 OS_DBG.C 是干啥的，先不管

修改 micrium 为 STM32 做的 μC/OS-II 移植
为啥不用 micrium 官方的移植？
因为 micrium 官方的移植有些东西《嵌入式实时操作系统 μC/OS-II》没有，这些看不懂，所以想在 micrium 官方的移植做个简化。
KEIL3.8 工程结构：




1.使用 st V3.4.0 库中的 startup_stm32f10x_hd.s 作为启动代码
需要修改的地方：
①用 OS_CPU_PendSVHandler 替换 startup_stm32f10x_hd.s 中所有的 PendSV_Handler
②用 OS_CPU_SysTickHandler 替换 startup_stm32f10x_hd.s 中所有的 SysTick_Handler

2.APP.C
因为打算把工程中 uC-CPU Group 给删了，而 BSP_IntDisAll()调用了其中的函数，故在 APP.C

增加#define IntDisAll()　__set_PRIMASK(0x01)，其中把 BSP_IntDisAll()改成了 IntDisAll()。
只保留 LED 相关的任务函数，其它的都删掉。

3.修改 app_cfg.h 和 OS_CFG.H

把 PROBE 相关的东西禁掉。其它需要修改的，详见工程。

4.修改 includes.h，bsp.h

因为用的 ST 库是 V3.4.0，把#include　<stm32f10x_lib.h>改成#include　<stm32f10x.h>
在 bsp.h 中也做类似的修改。其它需要修改的，详见工程。

5.BSP 部分

BSP 部分只保留 BSP.C 和 BSP.H，需要修改的部分，详见工程。