嵌入式操作系统 μC/OS－II 在 MSP430F168 单片机上的移植

uC/OS－II Port for MSP430F168

杨书凯* 刘 慧 杨 立

YANG Shu－kai LIU Hui YANG Li

摘 要 本文给出了在 RAM 空间较少的单片机上移植嵌入式操作系统 μC/OS－II 的一种方法，并根据该方法成功将 μC/OS－II 移植到了只有 2K RAM 空间的 MSP430F168 单片机上。

关键词 μC/OS－II MSP430F168 移植

Abstract The μC/OS－II has been ported to MSP430F168 with 2K byte RAM. The method that seperating interrupt stack and task stack is useful to portμC/OS－II to MCU with few RAM.

Keywords μC/OS－II MSP430F168 Port

μC/OS－II 作为一种开源的实时嵌入式操作系统已经得到了广泛的应用，被移植到了多种处理器和微控制器上[1]。但在 MSP430 这类 RAM 空间较小的微控制器上移植 μC/OS－II 还有一些困难，本文重点介绍在 MSP430F168 上移植 μC/OS－II 的一些处理方法，这些处理方法对于在 RAM 空间较小的微处理器或微控制器上移植 μC/OS－II 都有一定的借鉴意义。

1 在 MSP430F168 上的移植 μCOS－II 需要做的工作

μCOS－II 其 90% 的代码是用 C 语言写的，可以直接移植到有 C 语言编译器的绝大部分处理器和微控制器上。移植工作主要都集中在多任务切换的实现上，因为这部分代码用来保存和恢复 CPU 现场（即写/读相关寄存器），不能用 C 语言，只能使用汇编语言完成。μC/OS－II 的全部源代码量大约是 6000－7000 行，15 个文件。将 μC/OS－II 移植到 MSP430F168 微控制器上只需要修改三个与单片机体系结构相关的文件 OS_CPU. H、OS_CPU_C. C 和 OS_CPU_A. S。具体的工作有以下几项：

（1） 用#define 设置一个常量的值（OS_CPU. H）；

（2） 声明 10 个数据类型（OS_CPU. H）；

（3） 用#define 声明三个宏（OS_CPU. H）；

（4） 用 C 语言编写六个简单的函数（OS_CPU_C. C）；

（5） 编写四个汇编语言函数（OS_CPU_A. ASM）。

由于 MSP430F168 微控制器 RAM 资源太少，μCOS－II 发生中断直接把现场保存在所运行的任务的堆栈上，所以每个任务堆栈必须留下足够的空间保存中断堆栈，因此造成 RAM 空间的浪费，这些对于 RAM 空间大的处理器可以，对于 RAM 空间只有 2KB 的 MSP430F168 来说，可能没有足够的 RAM 运行所有必需任务。所以要实现 μC/OS－II 中断栈与任务栈分离。实现 μC/OS－II 把中断栈与任务栈分离节省 RAM 空间，代价是增加任务调度时间。实现中断栈与任务栈分离要在移植中增加必要的变量和操作。

* 山东省海洋仪器仪表研究所 山东青岛 266001

2 OS_CPU. H 文件的移植

2.1 数据类型定义

数据类型的修改与所用的编译器有关，不同的编译器使用不同的字节长度表示某种数据类型。μC/OS－II 的作者为了增加该操作系统的可移植性，在头文件 OS_CPU. H 中重新定义了 μC/OS－II 自己使用的更明确的数据类型。这使得 μCOS－II 在不同平台上移植时只需要在该头文件中将编译器使用的数据类型与操作系统使用的数据类型做一个简单的映射。本系统应用的开发环境是 IAR Embedded Workbench for MSP430 2. 10A[2]，相对应于 MSP430 的编译器笔者对数据类型的映射做了如下修改。

```
typedef unsigned char BOOLEAN;
typedef unsigned char INT8U;
typedef signed char INT8S;
typedef unsigned int INT16U;
typedef signed int INT16S;
typedef unsigned long INT32U;
typedef signed long INT32S;
typedef float FP32;
typedef double FP64;
```

2.2 堆栈单位

在任务切换时，CPU 现场的寄存器将保存在当前运行任务的堆栈中，所以 OS_STK 数据类型应该与 CPU 的寄存器长度一致。

```
typedef unsigned int OS_STK;
```

2.3 状态寄存器（SR）

由于 MSP430 微控制器关于状态寄存器（SR）操作很多，所以设置变量 typedef unsigned int OS_CPU_SR;/* 定义 CPU 的状态寄存器为一个字 */

2.4 堆栈增长方向

μCOS－II 中的宏 OS_STK_GROWTH 定义了堆栈的生长方向，可以根据不同的 CPU 类型做相应的修改，MSP430 微控制器堆栈由高地址向低地址增长[3]，所以将其定义为 1。

```
#define OS_STK_GROWTH 1  /* MSP430 的堆栈生长是从高到低 */
```

2.5 宏定义

包括开关中断的宏定义,以及进行任务切换的宏定义。

```
#define OS_CRITICAL_METHOD 1
#if OS_CRITICAL_METHOD =1
#define OS_ENTER_CRITICAL( ) _DINT( )      /* 关中断 */
#define OS_EXIT_CRITICAL( ) _EINT( )     /* 开中断 */
#endif
```

任务切换函数

```
#define OS_TASK_SW( ) OSCtxSw( )    /* 任务级上下文切换函数 */
```

2.6 系统中断栈指针(实现中断栈与任务栈分离)

```
OS_CPU_EXT OS_STK * OSISRStkPtr;  /* 指向中断服务程序的堆栈的指针 */
```

2.7 对状态寄存器(SR)操作函数

```
OS_CPU_SR OSCPUSaveSR(void);
void OSCPURestoreSR(OS_CPU_SR cpu_sr)
```

3 OS_CPU_C.C 文件的移植

3.1 建立任务时需要对任务栈进行初始化,函数如下:

```
OS_STK * OSTaskStkInit (void ( * task)( void * pd),void * pdata,OS_STK * ptos, INT16U opt)
{
    INT16U * top;
    opt = opt;
    top = (INT16U * )ptos;
    top - - ;
    * top = (INT16U)task;      /* 任务的起始地址 */
    top - - ;
    * top = (INT16U)task;      /* 中断返回地址 */
    top - - ;
    * top = (INT16U)0x0008;  /* 状态寄存器 */
    top - - ;
    * top = (INT16U)pdata;     /* 通过 R14 传递参数 pdata */
    top - - ;
    * top = (INT16U)0x1515;
    return ((OS_ STK * )top);
}
```

3.2 系统 HOOK 函数

在该文件中需要实现几个操作系统规定的 HOOK 函数,如下:

```
OSInitHookBegin(void),      OSInitHookEnd (void)
OSTaskCreateHook(OS_TCB * ptcb),      OSTaskDelHook (OS_TCB * ptcb),
OSTaskStatHook(void),     OSTaskSwHook (void),
OSTCBInitHook (OS_TCB * ptcb),      OSTimeTickHook (void)
```

如果用户不使用,钩子函数一般都为空。但是为降低系统的功耗,使系统执行 Idle 前进入低功耗模式,所以可以把 OSTaskIdleHook(void)改写为:

```
void OSTaskIdleHook(void)
{
    LPM0;      /* 进入低功耗模式 LPM0. */
}
```

4 OS_CPU_A.S 文件的移植

4.1 入栈和出栈寄存器修改

MSP430 单片机在发生中断时只进行两条操作,先将 SR(状态寄存器)压入堆栈中保存,然后将中断发生时要执行的下一条指令的 PC 值压入堆栈保存。IAR Embedded Workbench 的 C 语言编译器在编译 C 语言写的中断程序时,除执行上两条操作外,还自动保存了 R12 ~ R15 四个寄存器[4]。由于在 μCOS - II 中必须保存所有的寄存器,所以添加两个宏保存和恢复剩下未保存的寄存器。

```
PUSHALL     MACRO
            push    R4
            push    R5
            push    R6
            push    R7
            push    R8
            push    R9
            push    R10
            push    R11
            ENDM
POPALL      MACRO
            pop     R11
            pop     R10
            pop     R9
            pop     R8
            pop     R7
            pop     R5
            pop     R4
ENDM
```

4.2 OSStartHighRdy()修改

该函数是在多任务启动后,为堆栈分离需要,先将 SP 保存到 OSISRStkPtr 中。然后从最高优先级任务的 TCB 控制块中获得该任务的堆栈指针 SP,通过 SP 依次将 CPU 现场恢复,这时系统就将控制权交给用户创建的该任务进程,直到该任务被阻塞或者被其他更高优先级的任务抢占 CPU。该函数仅在多任务启动时被执行一次,即执行最高优先级任务,之后多任务的调度和切换由其他函数实现。代码如下:

```
    RSEG    CODE
OSStartHighRdy
    call    #OSTaskSwHook
    mov.b   #1, &OSRunning
```

```
    mov. w    SP, &OSISRStkPtr
    mov. w    &OSTCBHighRdy, R13
    mov. w    @R13, SP
    POPALL
    reti
```

4.3 OSTaskSw()修改

任务级的上下文切换,当任务因为被阻塞而主动请求CPU调度时被执行。它的工作是先将当前任务的CPU现场保存到该任务堆栈中,然后获得最高优先级任务的堆栈指针,从该堆栈中恢复此任务的CPU现场,使之继续执行。这样就完成了一次任务切换。代码如下:

```
    OSCtxSW
    push    SR
    PUSHALL
    mov. w
    mov. w    &OSTCBCur, R13
    mov. w    SP, 0(R13)
    call      #OSTaskSwHook
    mov. b    &OSPrioHighRdy, R13
    mov. b    R13, &OSPrioCur
    mov. w    &OSTCBHighRdy, R13
    mov. w    R13, &OSTCBCur
    mov. w    @R13,SP
    POPALL
    reti
```

4.4 OSIntCtxSw()修改

中断级的任务切换,在时钟中断ISR(中断服务例程)中发现有高优先级任务等待的时钟信号到来则在中断退出后并不返回被中断任务,而是直接调度就绪的高优先级任务执行,从而能够尽快地让高优先级的任务得到响应,保证系统的实时性能。其原理基本上与任务级的切换相同,但是由于进入中断时已经保存了被中断任务的CPU现场,因此不用再进行类似的操作,只需恢复现场。代码如下:

```
    OSIntCtxSw
    Call      #OSTaskSwHook
    mov. b    &OSPrioHighRdy,R13
    mov. b    R13,&OSPrioCur
    mov. w    &OSTCBHighRdy,R13
    mov. w    R13,&OSTCBCur
    mov. w    @R13,SP
    POPALL
    ret
```

4.5 OSTickISR()修改

时钟中断处理函数主要任务是负责处理时钟中断,调用OSTimeTick函数。如果有等待时钟信号的高优先级任务,则需要在中断级别上调度其执行。本系统用看门狗定时器WDT作为时钟源。代码如下:

```
WDT_ISR
    PUSHALL
    bic. b    #0x01,IE1
    cmp. b    #0,&OSIntNesting
    jne       WDT_ISR_1
    mov. w    &OSTCBCur,R13
    mov. w    SP,0(R13)
    mov. w    &OSISRStkPtr,SP
WDT_ISR_1
    Inc. b    @OSIntNesting
    bis. b    #0x01,IE1
    call      #OSTimeTick
    call      #OSIntExit
    DINT
    cmp. b    #0,&OSIntNesting
    jne       WDT_ISR_2
    mov. w    &OSTCBHighRdy,R13
    mov. w    @R13,SP
    WDT_ISR_2
    POPALL
    reti
```

4.6 OSCPUSaveSR()和OSCPURestoreSR()函数

实现OS_CPU.H文件中对状态寄存器(SR)的操作函数

```
    OSCPUSaveSR
        mov. w    SR,R12
        ret
    OSCPURestoreSR
        mov. w    R12,SR
        ret
```

5 应用

本文的上述移植应用在了SXZ2-1型海洋自动化观测系统上。根据系统需要处理事务的优先级,通过任务分割,将系统程序分成数据接收、数据处理、数据显示、数据调取、参数设置等几个任务,简化了系统程序的设计,提高了系统程序的可靠性和实时性。

6 结语

本文通过把μC/OS-II中的中断栈和任务栈分离,减少了μC/OS-II需要的RAM空间容量,从而实现了在RAM空间较小的MSP430F168单片机上的移植。该方法对于μC/OS-II在其他RAM容量较小的单片机上移植也有一定的借鉴意义。

参考文献:

[1] Labrosse Jean J. 嵌入式实时操作系统μC/OS-II(第2版)[M]. 北京:北京航空航天出版社,2003:5.

[2] MSP430F16X Data Sheet[OL]. TI,Inc. ,2002:12.

[3] 胡大可编著. MSP430系列Flash型超低功耗16位单片机[M]. 北京:北京航空航天大学出版社. 2002:45.

[4] 魏小龙编著. MSP430系列单片机接口技术及系统设计实例[M]. 北京:北京航空航天大学出版社. 2003:67,68 uC/OS-II Port for MSP430F168.

(收稿日期:2007-06-07)