第10章从 µC/OS 升级到 µC/OS-II

本章描述如何从µC/OS 升级到 µC/OS-II。如果已经将µC/OS移植到了某类微处理器上，移植µC/OS-II所要做的工作应当非常有限。在多数情况下，用户能够在1个小时之内完成这项工作。如果用户熟悉µC/OS的移植，可隔过本章前一部分直接参阅10.05节。

10.0 目录和文件

用户首先会注意到的是目录的结构，主目录不再叫 \SOFTWARE\uCOS。而是叫\SOFTWARE\uCOS-II。所有的µC/OS-II文件都应放在用户硬盘的\SOFTWARE\uCOS-II 目录下。面向不同的微处理器或微处理器的源代码一定是在以下两个或三个文件中： OS_CPU.H, OS_CPU_C.C，或许还有OS_CPU_A.ASM.。汇编语言文件是可有可无的，因为有些C编译程序允许使用在线汇编代码，用户可以将这些汇编代码直接写在 OS_CPU_C.C.中。

与微处理器有关的特殊代码，即与移植有关的代码，在 µC/OS 中是放在用微处理器名字命名的文件中的，例如，Intel 80x86的实模式（Real Mode），在大模式下编译（Large Modle）时，文件名为Ix86L.H， Ix86L_C.C， 和Ix86L_A.ASM.。

表 L10.1在µC/OS-II中重新命名的文件.

\SOFTWARE\uCOS\Ix86L	\SOFTWARE\uCOS-II\Ix86L
Ix86L.H	OS_CPU.H
Ix86L_A.ASM	OS_CPU_A.ASM
Ix86L_C.C	OS_CPU_C.C

升级可以从这里开始：首先将µC/OS目录下的旧文件复制到µC/OS-II 的相应目录下，并改用新的文件名，这比重新建立一些新文件要容易许多。表10.2给出来几个与移植有关的新旧文件名命名法的例子。

表 L10.2对不同微处理器从µC/OS到µC/OS-II，要重新命名的文件.

\SOFTWARE\uCOS\I80251	\SOFTWARE\uCOS-II\I80251
I80251.H	OS_CPU.H
I80251.C	OS_CPU_C.C
\SOFTWARE\uCOS\M680x0	\SOFTWARE\uCOS-II\M680x0
M680x0.H	OS_CPU.H
M680x0.C	OS_CPU_C.C
\SOFTWARE\uCOS\M68HC11	\SOFTWARE\uCOS-II\M68HC11
M68HC11.H	OS_CPU.H
M68HC11.C	OS_CPU_C.C

\SOFTWARE\uCOS\Z80	\SOFTWARE\uCOS-II\Z80
Z80.H	OS_CPU.H
Z80_A.ASM	OS_CPU_A.ASM
Z80_C.C	OS_CPU_C.C

10.1 INCLUDES.H

用户应用程序中的INCLUDES.H 文件要修改。以80x86 实模式，在大模式下编译为例，用户要做如下修改：

- 变目录名 μC/OS 为 μC/OS-II
- 变文件名 IX86L.H 为 OS_CPU.H
- 变文件名UCOS.H 为 uCOS_II.H

新旧文件如程序清单 L10.1和 L10.2所示

10.2 OS_CPU.H

OS_CPU.H 文件中有与微处理器类型及相应硬件有关的常数定义、宏定义和类型定义。

10.2.1 与编译有关的数据类型s

为了实现 μC/OS-II，用户应定义6个新的数据类型：INT8U、INT8S、INT16U、NT16S、INT32U、和INT32S。这些数据类型有分别表示有符号和无符号8位、16位、32位整数。在μC/OS中相应的数据类型分别定义为：UBYTE、BYTE、UWORD、WORD、ULONG和LONG。用户所要做的仅仅是复制μC/OS中数类型并修改原来的UBYTE为INT8U，将BYTE为INT8S，将UWORD修改为INT16U等等，如程序清单 L10.3所示。

程序清单 L10.1 μC/OS 中的 INCLUDES.H

```
/*
*********************************************************************
*                    INCLUDES.H
*********************************************************************
*/

#include   <STDIO.H>
#include   <STRING.H>
#include   <CTYPE.H>
#include   <STDLIB.H>
#include   <CONIO.H>
#include   <DOS.H>

#include   "\SOFTWARE\UCOS\IX86L\IX86L.H"
#include   "OS_CFG.H"
#include   "\SOFTWARE\UCOS\SOURCE\UCOS.H"
```

程序清单 L10.2 μC/OS-II 中的 INCLUDES.H

```
/*
*********************************************************************
*                    INCLUDES.H
*********************************************************************
*/

#include   <STDIO.H>
#include   <STRING.H>
#include   <CTYPE.H>
#include   <STDLIB.H>
#include   <CONIO.H>
#include   <DOS.H>

#include   "\SOFTWARE\uCOS-II\IX86L\OS_CPU.H"
#include   "OS_CFG.H"
#include   "\SOFTWARE\uCOS-II\SOURCE\uCOS_II.H"
```

程序清单 L10.3µC/OS到µC/OS-II 数据类型的修改.

```
/* uC/OS data types:                                                  */
typedef unsigned char  UBYTE;    /* Unsigned  8 bit quantity          */
typedef signed   char  BYTE;     /* Signed    8 bit quantity          */
typedef unsigned int   UWORD;    /* Unsigned 16 bit quantity          */
typedef signed   int   WORD;     /* Signed   16 bit quantity          */
typedef unsigned long  ULONG;    /* Unsigned 32 bit quantity          */
typedef signed   long  LONG;     /* Signed   32 bit quantity          */


/* uC/OS-II data types                                                */
typedef unsigned char  INT8U;    /* Unsigned  8 bit quantity          */
typedef signed   char  INT8S;    /* Signed    8 bit quantity          */
typedef unsigned int   INT16U;   /* Unsigned 16 bit quantity          */
typedef signed   int   INT16S;   /* Signed   16 bit quantity          */
typedef unsigned long  INT32U;   /* Unsigned 32 bit quantity          */
typedef signed   long  INT32S;   /* Signed   32 bit quantity          */
```

在µC/OS中，任务栈定义为类型OS_STK_TYPE，而在µC/OS-II中任务栈要定义类型OS_STK.，为了免于修改所有应用程序的文件，可以在OS_CPU.H中建立两个数据类型，以Intel 80x86 为例，如程序清单 L10.4所示。

程序清单 L10.4 µC/OS 和 µC/OS-II任务栈的数据类型

```
#define OS_STK_TYPE  UWORD       /* 在 uC/OS 中                  */
#define OS_STK      INT16U      /*  在 uC/OS-II 中              */
```

10.2.2OS_ENTER_CRITICAL()和OS_EXIT_CRITICAL()

µC/OS-II和µC/OS一样，分别定义两个宏来开中断和关中断：OS_ENTER_CRITICAL()和OS_EXIT_CRITICAL()。在µC/OS向µC/OS-II升级的时候，用户不必动这两个宏。.

10.2.3OS_STK_GROWTH

大多数微处理器和微处理器的栈都是由存储器高地址向低地址操作的，然而有些微处理器的工作方式正好相反。µC/OS-II设计成通过定义一个常数OS_STK_GROWTH来处理不同微处理器栈操作的取向：

对栈操作由低地址向高地址增长，设OS_STK_GROWTH 为 0
对栈操作由高地址向低地址递减，设OS_STK_GROWTH 为 1

有些新的常数定义（#define constants ）在μC/OS中是没有的，故要加到OS_CPU.H中去。

10.2.4OS_TASK_SW()

OS_TASK_SW()是一个宏，从μC/OS升级到μC/OS-II时，这个宏不需要改动。当μC/OS-II从低优先级的任务向高优先级的任务切换时要用到这个宏，OS_TASK_SW()的调用总是出现在任务级代码中。

10.2.5OS_FAR

因为Intel 80x86的结构特点，在μC/OS中使用过OS_FAR 。这个定义语句（#define ）在μC/OS-II中去掉了，因为这条定义使移植变得不方便。结果是对于Intel 80x86，如果用户定义在大模式下编译时，所有存储器属性都将为远程(FAR).

在μC/OS-II中，任务返回值类型定义如程序清单L10.5所示。用户可以重新编辑所有OS_FAR的文件，或者在μC/OS-II中将OS_FAR定义为空，去掉OS_FAR，以实现向μC/OS-II的升级。

程序清单 L10.5 在 μC/OS 中任务函数的定义

```
void OS_FAR task (void *pdata)
{
    pdata = pdata;
    while (1) {
        .
        .
    }
}
```

10.3 OS_CPU_A.ASM

移植μC/OS 和μC/OS-II 需要用户用汇编语言写4个相当简单的函数。

```
OSStartHighRdy()
OSCtxSw()
OSIntCtxSw()
OSTickISR()
```

10.3.1OSStartHighRdy()

在μC/OS-II中，OSStartHighRdy ()要调用OSTaskSwHook()。OSTaskSwHook()这个函数在μC/OS中没有。用户将最高优先级任务的栈指针装入CPU之前要先调用OSTaskSwHook()。还有， OSStartHighRdy要在调用OSTaskSwHook()之后立即将OSRunning设为1。程序清单L10.6 给出OSStartHighRdy()的示意代码。.μC/OS只有其中最后三步。

程序清单 L10.6 OSStartHighRdy()的示意代码

```
OSStartHighRdy:
    Call OSTaskSwHook();                                          调用OSTaskSwHook();
    Set OSRunning to 1;                                            置 OSRunning 为 1;
    Load the processor stack pointer with OSTCBHighRdy->OSTCBStkPtr;
                                将 OSTCBHighRdy->OSTCBStkPtr 装入处理器的栈指针;
    POP all the processor registers from the stack;  从栈中弹出所有寄存器的值;
    Execute a Return from Interrupt instruction;              执行中断返回指令;
```

10.3.2OSCtxSw()

在μC/OS-II中，任务切换要增作两件事，首先，将当前任务栈指针保存到当前任务控制块TCB后要立即调用OSTaskSwHook()。其次，在装载新任务的栈指针之前必须将OSPrioCur设为OSPrioHighRdy 。OSCtxSw()的示意代码如程序清单L10.7所示。μC/OS-II加上了步骤L10.7(1)和(2)。

程序清单 L10.7 OSCtxSw()的示意代码

```
OSCtxSw:
    PUSH processor registers onto the current task's stack;
                                              所有处理器寄存器的值推入当前任务栈;
    Save the stack pointer at OSTCBCur->OSTCBStkPtr;
    Call OSTaskSwHook();                                1)
    OSTCBCur  = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;                                                  (2)
    Load the processor stack pointer with OSTCBHighRdy->OSTCBStkPtr;
                                将 OSTCBHighRdy->OSTCBStkPtr 装入处理器的栈指针;
    POP all the processor registers from the stack;  从栈中弹出所有寄存器的值;
    Execute a Return from Interrupt instruction;
```

10.3.3OSIntCtxSw()

如同上述函数一样，在μC/OS-II.中，OSCtxSw()也增加了两件事。首先，将当前任务的栈指针保存到当前任务的控制块TCB后要立即调用OSTaskSwHook()。其次，在装载新任务的栈指针之前必须将OSPrioCur 设为OSPrioHighRdy。程序清单L10.8给出OSIntCtxSw()的示意代码。μC/OS-II.中增加了L10.8（1）和（2)。

程序清单 L10.8 OSIntCtxSw()的示意代码

```
OSIntCtxSw():
    Adjust the stack pointer to remove call to OSIntExit(), locals in OSIntExit()
      and the call to OSIntCtxSw();
                                        调整由于调用上述子程序引起的栈指针值的变化;
    Save the stack pointer at OSTCBCur->OSTCBStkPtr;
                                              保存栈指针到OSTCBCur->OSTCBStkPtr;
    Call OSTaskSwHook();                                  调用OSTaskSwHook();(1)
    OSTCBCur  = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;                                                (2)
    Load the processor stack pointer with OSTCBHighRdy->OSTCBStkPtr;
                                 将 OSTCBHighRdy->OSTCBStkPtr 装入处理器的栈指针;
    POP all the processor registers from the stack; 从栈中弹出所有寄存器的值;
    Execute a Return from Interrupt instruction;            执行中断返回指令;
```

10.3.4OSTickISR()

在µC/OS-II和µC/OS 中，这个函数的代码是一样，无须改变。

10.4 OS_CPU_C.C

移植 µC/OS-II 需要用C语言写6个非常简单的函数：

```
OSTaskStkInit()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskSwHook()
OSTaskStatHook()
OSTimeTickHook()
```

其中只有一个函数OSTaskStkInit()是必不可少的。其它5个只需定义，而不包括任何代码。

10.4.1OSTaskStkInit()

在µC/OS中，OSTaskCreate()被认为是与使用的微处理器类型有关的函数。实际上这个函数中只有一部分内容是依赖于微处理器类型的。在µC/OS-II中，与使用的微处理器类型有关的那一部分已经从函数OSTaskCreate() 中抽出来了，放在一个叫作OSTaskStkInit()的函数中。

OSTaskStkInit()只负责设定任务的栈，使之看起来好像中断刚刚发生过，所有的CPU寄存器都被推入堆栈。作为提供给用户的例子，程序清单L10.9给出Intel 80x86实模式，在大模式下编译的 µC/OS的OSTaskCreate()函数的代码。程序清单L10.10是同类微微处理器的µC/OS-II的OSTaskStkInit()函数的代码。比较这两段代码，可以看出：从 [L10.9(1)]

OS_EXIT_CRIITICAL() 到 [L10.9(2)] 调用 OSTaskStkInit() 都抽出来并移到了 OSTaskStkInit()中。

程序清单L10.9 μC/OS 中的 OSTaskCreate()

```
UBYTE OSTaskCreate(void (*task)(void *pd), void *pdata, void *pstk, UBYTE p)
{
    UWORD OS_FAR *stk;
    UBYTE        err;



    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[p] == (OS_TCB *)0) {
        OSTCBPrioTbl[p] = (OS_TCB *)1;
        OS_EXIT_CRITICAL();                                          (1)
        stk    = (UWORD OS_FAR *)pstk;
        *--stk = (UWORD)FP_OFF(pdata);
        *--stk = (UWORD)FP_SEG(task);
        *--stk = (UWORD)FP_OFF(task);
        *--stk = (UWORD)0x0202;
        *--stk = (UWORD)FP_SEG(task);
        *--stk = (UWORD)FP_OFF(task);
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = (UWORD)0x0000;
        *--stk = _DS;
        err    = OSTCBInit(p, (void far *)stk);                      (2)
        if (err == OS_NO_ERR) {
            if (OSRunning) {
                OSSched();
            }
        } else {
            OSTCBPrioTbl[p] = (OS_TCB *)0;
```

```
        }
        return (err);
    } else {
        OS_EXIT_CRITICAL();
        return (OS_PRIO_EXIST);
    }
}
```

程序清单 L10.10 μC/OS-II中的OSTaskStkInit()

```
void *OSTaskStkInit (void (*task)(void *pd), void *pdata, void *ptos, INT16U
opt)
{
    INT16U *stk;



    opt    = opt;
    stk    = (INT16U *)ptos;
    *stk-- = (INT16U)FP_SEG(pdata);
    *stk-- = (INT16U)FP_OFF(pdata);
    *stk-- = (INT16U)FP_SEG(task);
    *stk-- = (INT16U)FP_OFF(task);
    *stk-- = (INT16U)0x0202;
    *stk-- = (INT16U)FP_SEG(task);
    *stk-- = (INT16U)FP_OFF(task);
    *stk-- = (INT16U)0xAAAA;
    *stk-- = (INT16U)0xCCCC;
    *stk-- = (INT16U)0xDDDD;
    *stk-- = (INT16U)0xBBBB;
    *stk-- = (INT16U)0x0000;
    *stk-- = (INT16U)0x1111;
    *stk-- = (INT16U)0x2222;
    *stk-- = (INT16U)0x3333;
    *stk-- = (INT16U)0x4444;
    *stk   = _DS;
    return ((void *)stk);
}
```

10.4.2OSTaskCreateHook()

OSTaskCreateHook()在μC/OS中没有，如程序清单L10.11所示，在由.μC/OS 向μC/OS-II升级时，定义一个空函数就可以了。注意其中的赋值语句，如果不把Ptcb赋给Ptcb，有些编译器会产生一个警告错误，说定义的Ptcb变量没有用到。

程序清单10.11 μC/OS-II 中的OSTaskCreateHook()

```
#if OS_CPU_HOOKS_EN
OSTaskCreateHook(OS_TCB *ptcb)
```

```
{
    ptcb = ptcb;
}
#endif
```

用户还应该使用条件编译管理指令来处理这个函数。只有在OS_CFG.H 文件中将OS_CPU_HOOKS _EN设为1时，OSTaskCreateHook()的代码才会生成。这样做的好处是允许用户移植时可在不同文件中定义钩子函数。

10.4.3OSTaskDelHook()

OSTaskDelHook() 这个函数在μC/OS中没有，如程序清单10.12所示，从μC/OS 到μC/OS-II，只要简单地定义一个空函数就可以了。注意，如果不用赋值语句将ptcb赋值为ptcb，有些编译程序可能会产生一些警告信息，指出定义的ptcb变量没有用到。

程序清单 L10.12 μC/OS-II中的OSTaskDelHook().

```
#if OS_CPU_HOOKS_EN
OSTaskDelHook(OS_TCB *ptcb)
{
    ptcb = ptcb;
}
#endif
```

也还是要用条件编译管理指令来处理这个函数。只有把OS_CFG.H. 文件中的OS_CPU_HOOKS_EN 设为1，OSTaskDelHook()的代码才能生成。这样做的好处是允许用户移植时在不同的文件中定义钩子函数。

10.4.4OSTaskSwHook()

OSTaskSwHook() 在μC/OS 中也不存在。从μC/OS向μC/OS-II升级时，只要简单地定义一个空函数就可以了，如程序清单L10.13所示。

程序清单 L10.13 μC/OS-II中的OSTaskSwHook()函数

```
#if OS_CPU_HOOKS_EN
OSTaskSwHook(void)
{
}
#endif
```

也还是要用编译管理指令来处理这个函数。只有把OS_CFG.H 文件中的OS_CPU_HOOKS_EN设为1，OSTaskSwHook() 的代码才能生成。.

10.4.5OSTaskStatHook()

OSTaskStatHook()在μC/OS中不存在，从μC/OS向μC/OS-II升级时，只要简单地定义一个空函数就可以了，如程序清单L10.14所示。

也还是要用编译管理指令来处理这个函数。只有把OS_CFG.H 文件中的OS_CPU_HOOKS_EN设为1，OSTaskSwHook() 的代码才能生成。

程序清单 L10.14 μC/OS-II中的OSTaskStatHook()函数

```
#if OS_CPU_HOOKS_EN
OSTaskStatHook(void)
{
}
#endif
```

10.4.6OSTimeTickHook()

OSTimeTickHook()在μC/OS中不存在，从μC/OS向μC/OS-II升级时，只要简单地定义一个空函数就可以了，如程序清单L10.15所示。

也还是要用编译管理指令来处理这个函数。只有把OS_CFG.H 文件中的OS_CPU_HOOKS_EN设为1，OSTimeTickHook()的代码才能生成。

.

程序清单 L10.15 μC/OS-II中的OSTimeTickHook()

```
#if OS_CPU_HOOKS_EN
OSTimeTickHook(void)
{
}
#endif
```

10.5总结

表T10.3总结了从μC/OS向μC/OS-II.升级需要改变得地方。其中*processor_name.?*是μC/OS中移植范例程序的文件名。

表 T10.3 升级 μC/OS到 μC/OS-I要修改的地方

μC/OS	*μC/OS-II*

Processor_name.H	OS_CPU.H
数据类型： UBYTE BYTE UWORD WORD ULONG LONG	数据类型： INT8U INT8S INT16U INT16S INT32U INT32S
OS_STK_TYPE	OS_STK
OS_ENTER_CRITICAL()	不变
OS_EXIT_CRITICAL()	不变
—	增加了 OS_STK_GROWTH
OS_TASK_SW()	不变
OS_FAR	定义OS_FAR 为空，或删除所有的 OS_FAR
Processor_name.ASM	OS_CPU_A.ASM
OSStartHighRdy()	增加了调用 OSTaskSwHook(); 置 OSRunning 为 1 (8 bits)
OSCtxSw()	增加了调用 OSTaskSwHook(); 拷贝OSPrioHighRdy 到 OSPrioCur (8 bits)
OSIntCtxSw()	增加了调用OSTaskSwHook(); 拷贝 OSPrioHighRdy 到 OSPrioCur (8 bits)
OSTickISR()	不变
Processor_name.C	OS_CPU_C.C
OSTaskCreate()	抽出栈初始部分，放在函数 OSTaskStkInit() 中
—	增加了空函数 OSTaskCreateHook()
—	增加了空函数 OSTaskDelHook()
—	增加了空函数 OSTaskSwHook()
—	增加了空函数 OSTaskStatHook()
—	增加了空函数 OSTimeTickHook()