```
void  OSInit (OS_ERR  *p_err)
{
    CPU_STK      *p_stk;
    CPU_STK_SIZE  size;



#ifdef OS_SAFETY_CRITICAL

/*这个定义我没找到，可能是留给以后的，或是留给我们自己写*/
    if (p_err == (OS_ERR *)0) {
        OS_SAFETY_CRITICAL_EXCEPTION();
        return;
    }
#endif

    OSInitHook();
    /* 这个函数是留给用户写的，编程的人可根据自己情况写，当然空的也行
*/

    OSIntNestingCtr                 = (OS_NESTING_CTR)0;
    /* 中断计数器，每来一个硬件中断这个就加 1，                          */

    OSRunning                       =  OS_STATE_OS_STOPPED;
    /* OSRunning 是说明是否要启动多任务切换功能，很明显初始化时是不启
动这个功能*/

    OSSchedLockNestingCtr           = (OS_NESTING_CTR)0;
    /* 这个是为了给任务上锁，开始为 0，如果在某个任务某个段不想让别的
    任务打断就用上锁函数，这个参数会被加上，这样，别的任务就不能被切换，
因为负责任务调度的函数
    会是否不为 0，不为 0 就不会调度*/

#if OS_CFG_SCHED_LOCK_TIME_MEAS_EN > 0u
    OSSchedLockTimeBegin            = (CPU_TS)0;
    /*上锁时，硬件定时器的时间，直接读定时器的寄存器*/
    OSSchedLockTimeMax              = (CPU_TS)0;
    /*上锁的最长时间是多少，这个时间会影响实时性（好像是这样，因为我没
用过这个系统做过什么太大的项目，通过说系统代码来理解）*/
    OSSchedLockTimeMaxCur           = (CPU_TS)0;
    /*看程序，个人认为功能同上*/
#endif

#ifdef OS_SAFETY_CRITICAL_IEC61508
```

```
    OSSafetyCriticalStartFlag       =  DEF_FALSE;
    /*这个为 1 的话，就不能建新的任务了*/
#endif

#if OS_CFG_SCHED_ROUND_ROBIN_EN > 0u
/*这个 ucos3 一个优先级可以带任意多个任务，每次只运行该优先级的第一个任务，用这个可以在每个固定时间将最前面的任务调到最后，总觉得这样会影响实时性，下面细说*/
    OSSchedRoundRobinEn             = DEF_FALSE;
    OSSchedRoundRobinDfltTimeQuanta = OSCfg_TickRate_Hz / 10u;
#endif

    if (OSCfg_ISRStkSize > (CPU_STK_SIZE)0) {
        p_stk = OSCfg_ISRStkBasePtr;
        /* Clear exception stack for stack checking.                */
        if (p_stk != (CPU_STK *)0) {

        /*关于 OSCfg_ISRStkBasePtr 我也不是很清楚，不用也行*/
            size  = OSCfg_ISRStkSize;

            /*在 os_cfg_app.c 中有定义 CPU_STK*
const  OSCfg_ISRStkBasePtr= (CPU_STK*)&OSCfg_ISRStk[0];*/
            while (size > (CPU_STK_SIZE)0) {

            /*CPU_STK OSCfg_ISRSt[OS_CFG_ISR_STK_SIZE]这也是在 os_cfg_app.c 定义的*/
                size--;

                /*#define  OS_CFG_ISR_STK_SIZE 128u 在 os_cfg_app.h 中定义*/
                *p_stk = (CPU_STK)0;
                p_stk++;
            }
        }
    }

#if OS_CFG_APP_HOOKS_EN > 0u
    OS_AppTaskCreateHookPtr = (OS_APP_HOOK_TCB )0;
    /* 这是勾子函数指针，我们希望在任务调度时做一些我们希望它做的事， 就写个函数，让这个指针指向那个函数，然后让相应的勾子函数去运行它，下面会细说。以下同样*/
    OS_AppTaskDelHookPtr    = (OS_APP_HOOK_TCB )0;
    OS_AppTaskReturnHookPtr = (OS_APP_HOOK_TCB )0;
```

```
    OS_AppIdleTaskHookPtr   = (OS_APP_HOOK_VOID)0;
    OS_AppStatTaskHookPtr   = (OS_APP_HOOK_VOID)0;
    OS_AppTaskSwHookPtr     = (OS_APP_HOOK_VOID)0;
    OS_AppTimeTickHookPtr   = (OS_APP_HOOK_VOID)0;
#endif

    OS_PrioInit();
    /* 初始化优先级                    */

    OS_RdyListInit();
    /* 初始化就绪任务列表                                */

    OS_TaskInit(p_err);
     /* 初始化任务管理                      */
    if (*p_err != OS_ERR_NONE) {
        return;
    }

#if OS_CFG_ISR_POST_DEFERRED_EN > 0u
    OS_IntQTaskInit(p_err);
     /* 初始化中断队列函数          */
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif

    OS_IdleTaskInit(p_err);
     /* 初始化空任务，一般没有别的任务工作的话，就由它来占 CPU，空任务
的优先级要最低*/
    if (*p_err != OS_ERR_NONE) {
        return;
    }

    OS_TickTaskInit(p_err);
     /* 初始化时钟节拍函数                                */
    if (*p_err != OS_ERR_NONE) {
        return;
    }


#if OS_CFG_STAT_TASK_EN > 0u
/* 初始化统计任务                          */
    OS_StatTaskInit(p_err);
    if (*p_err != OS_ERR_NONE) {
```

```
        return;
    }
#endif


#if OS_CFG_FLAG_EN > 0u
/* ucos 的资源有 FLAG 控制，sem 控制,Mem 管理，Msg 消息管理，Mutx 互斥信
号管理，队列管理，软定时器管理*/
    OS_FlagInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_MEM_EN > 0u
/* Initialize the Memory Manager module                     */
    OS_MemInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if (OS_MSG_EN) > 0u
 /* Initialize the free list of OS_MSGs                      */
    OS_MsgPoolInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_MUTEX_EN > 0u
/* Initialize the Mutex Manager module                      */
    OS_MutexInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_Q_EN > 0u
    OS_QInit(p_err);
```

```
     /* Initialize the Message Queue Manager module            */
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_SEM_EN > 0u
/* Initialize the Semaphore Manager module                */
    OS_SemInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_TMR_EN > 0u
/* Initialize the Timer Manager module                    */
    OS_TmrInit(p_err);
    if (*p_err != OS_ERR_NONE) {
        return;
    }
#endif


#if OS_CFG_DBG_EN > 0u
    OS_Dbg_Init();
#endif

    OSCfg_Init();
}
```

使用这个操作系统首先要初始化这个系统的资源，OSInit 来初始化这些资源的。OSIntNestingCtr 这个变量，在来中断时，这个就加 1，说明这时还在处理外部中断
事情，这时候可能有任务进入就绪状态，但也不会被调用。OSSched 这个调度函数会去看这个变量是否为 0，为 0
就可以调度，不然不能。这个是在自己编写移植函数时调用处理中断函数前调用下 OSIntEnter，然后在调用
处理中断函数后调用 OSIntExit 就好了。

```
/************************************************************************
*************************************************
void  OSIntEnter (void)
{
```

```
    if (OSRunning != OS_STATE_OS_RUNNING) {                    /* Is OS
running?                                                 */
        return;                                                /*
No                                                       */
    }

    if (OSIntNestingCtr >= (OS_NESTING_CTR)250u) {            /* Have we
nested past 250 levels?                                  */
        return;                                                /*
Yes                                                      */
    }

    OSIntNestingCtr++;                                         /* 这里就
把 OSIntNestingCtr+1 了                                    */
}
*********************************************************************************
*********************************************************/
/*********************************************************************************
*********************************************************
void  OSIntExit (void)
{
    CPU_SR_ALLOC();



    if (OSRunning != OS_STATE_OS_RUNNING) {
    /* Has the OS started?                                     */
        return;
    /* No                                                      */
    }

    CPU_INT_DIS();
    if (OSIntNestingCtr == (OS_NESTING_CTR)0) {
    /* Prevent OSIntNestingCtr from wrapping                   */
        CPU_INT_EN();
        return;
    }
    OSIntNestingCtr--;

    /*看这，每退出一个中断处理就减 1，中断处理是会嵌套的*/
    if (OSIntNestingCtr > (OS_NESTING_CTR)0) {
    /* ISRs still nested?                                      */
        CPU_INT_EN();
    /* Yes                                                     */
```

```
        return;
    }

    if (OSSchedLockNestingCtr > (OS_NESTING_CTR)0) {
    /* Scheduler still locked?                                  */
        CPU_INT_EN();
    /* Yes                                                      */
        return;
    }

    OSPrioHighRdy   = OS_PrioGetHighest();
    /* Find highest priority                                    */
    OSTCBHighRdyPtr = OSRdyList[OSPrioHighRdy].HeadPtr;
    /* Get highest priority task ready-to-run                   */
    if (OSTCBHighRdyPtr == OSTCBCurPtr) {
    /* Current task still the highest priority?                 */
        CPU_INT_EN();
    /* Yes                                                      */
        return;
    }

#if OS_CFG_TASK_PROFILE_EN > 0u
    OSTCBHighRdyPtr->CtxSwCtr++;
    /* Inc. # of context switches for this new task             */
#endif
    OSTaskCtxSwCtr++;
    /* Keep track of the total number of ctx switches           */

    OSIntCtxSw();
    /* Perform interrupt level ctx switch                       */
    CPU_INT_EN();
}
*********************************************************************
*********************************************************/
```

我们也看到了，在这两个函数中都查看了 OSRunning，因为只有这个变量为 1 时，这个系统才真正能启动多任务处理能力，我们这样看一下

```
int maint()/*这只是为了举例*/
{
    OS_ERR p_err;
    。。。。。。
    OSInit (&p_err);
    /*OSTaskCreate 这个函数以后会细说的，这里的每个参数这里就不说了*/
    OSTaskCreate (OS_TCB        task1,
```

```
                    CPU_CHAR      "fisrt_tsk",
                    OS_TASK_PTR    task_func1,
                    void          *p_arg,
                    OS_PRIO        prio1,
                    CPU_STK       *p_stk_base,
                    CPU_STK_SIZE   stk_limit,
                    CPU_STK_SIZE   stk_size,
                    OS_MSG_QTY     q_size,
                    OS_TICK        time_quanta,
                    void          *p_ext,
                    OS_OPT         opt,
                    OS_ERR        *p_err);

       OSTaskCreate (OS_TCB        task2,
                    CPU_CHAR      "second_tsk",
                    OS_TASK_PTR    task_func2,
                    void          *p_arg,
                    OS_PRIO        prio1,
                    CPU_STK       *p_stk_base,
                    CPU_STK_SIZE   stk_limit,
                    CPU_STK_SIZE   stk_size,
                    OS_MSG_QTY     q_size,
                    OS_TICK        time_quanta,
                    void          *p_ext,
                    OS_OPT         opt,
                    OS_ERR        *p_err);
   OSStart (&p_err);
}
```

这里建了两个任务，而且其实 OSTaskCreate 这里是有调用 OSSched，但在这之前会先看 OSRunning，因这在这里
这个两个任务建立时很明显，OSRunning 还为 0，这时是在 OSTaskCreate 运行不到 OSSched 就会退出，所以不会有
任务动作，直到 OSStart (&p_err)这时 OSRunning 为 1，会在这个两任务中找一个优先级最大的运行，然后以后再
建什么任务，就可以在 OSTaskCreate 调用到 OSSched，如果你这个新建的任务优先级最高，那么就会直接调度到
这个新建的任务。如果没调用 OSStart，那么这个系系统也就不能真正启动。

OSSchedLockTimeBegin 这个是给任务上锁时读取，定时寄存器的值，

OSSchedLockTimeMax　当解锁时会计算下这次上锁的时间多长，如果比这个值大，就把这次的上锁时间给这个变量

OSSchedLockTimeMaxCur 看代码，功能好像同上

```
/************************************************************************
************************************
/*
下面说的是上锁函数，OSSchedLockNestingCtr 这变量会在 OSSched 中被检查，
只要这个变量不为 0 就不会进行调度了
OSSchedLock 会使 OSSchedLockNestingCtr 加 1，OSSchedUnlock 会使
OSSchedLockNestingCtr 减 1。至于 OSSched 先不说以后会说的。
*/
void  OSSchedLock (OS_ERR  *p_err)
{
    CPU_SR_ALLOC();/*CPU_CRITICAL_ENTER(),CPU_CRITICAL_EXIT()和这两个
函数一起用的，看到了我说完上锁就说它们吧*/



#ifdef OS_SAFETY_CRITICAL
    if (p_err == (OS_ERR *)0) {
        OS_SAFETY_CRITICAL_EXCEPTION();
        return;
    }
#endif

#if OS_CFG_CALLED_FROM_ISR_CHK_EN > 0u
    if (OSIntNestingCtr > (OS_NESTING_CTR)0) {
    /* Not allowed to call from an ISR                          */
        *p_err = OS_ERR_SCHED_LOCK_ISR;
        return;
    }
#endif

    if (OSRunning != OS_STATE_OS_RUNNING) {
    /* Make sure multitasking is running                        */
       *p_err = OS_ERR_OS_NOT_RUNNING;
        return;
    }

    if (OSSchedLockNestingCtr >= (OS_NESTING_CTR)250u) {
    /*带个问题，这是能上多少层锁啊？这有点不懂因为只要这个由不变 0，就
不能调度了，这是要给一个任务上几层锁有毕要吗？*/
       *p_err = OS_ERR_LOCK_NESTING_OVF;
        return;
    }
/************************************************************************
************************************************
```

```
*


       *
*                                以上是检查功能，真正的功能从下面开始

                                                    *
*


       *
*********************************************************************
*************************************************/
    CPU_CRITICAL_ENTER();           /*禁中断*/
    OSSchedLockNestingCtr++;
    /* 把这个标志加 1，就表示上锁了                                    */
#if OS_CFG_SCHED_LOCK_TIME_MEAS_EN > 0u
    OS_SchedLockTimeMeasStart();

    /*读取当前上锁时间，下面就说它，这个很简单的*/
#endif
    CPU_CRITICAL_EXIT();    /*开中断*/
   *p_err = OS_ERR_NONE;
}
*********************************************************************
*************************************************/
/*********************************************************************
*****************************************************
/*解锁函数*/
void  OS_SchedLockTimeMeasStart (void)
{
    if (OSSchedLockNestingCtr == 1u) {
        OSSchedLockTimeBegin = CPU_TS_TmrRd();
        /*读取当前时间，也就是定时寄存器的值，CPU_TS_TmrRd()是留给我们
写*/
    }
}
*********************************************************************
*************************************************/
/*********************************************************************
*****************************************************
void  OSSchedUnlock (OS_ERR  *p_err)
{
    CPU_SR_ALLOC();
```

```
#ifdef OS_SAFETY_CRITICAL
    if (p_err == (OS_ERR *)0) {
        OS_SAFETY_CRITICAL_EXCEPTION();
        return;
    }
#endif

#if OS_CFG_CALLED_FROM_ISR_CHK_EN > 0u
    if (OSIntNestingCtr > (OS_NESTING_CTR)0) {
    /* Not allowed to call from an ISR                          */
        *p_err = OS_ERR_SCHED_UNLOCK_ISR;
        return;
    }
#endif

    if (OSRunning != OS_STATE_OS_RUNNING) {
    /* Make sure multitasking is running                        */
       *p_err = OS_ERR_OS_NOT_RUNNING;
        return;
    }

    if (OSSchedLockNestingCtr == (OS_NESTING_CTR)0) {
    /* See if the scheduler is locked                           */
       *p_err = OS_ERR_SCHED_NOT_LOCKED;
        return;
    }

    CPU_CRITICAL_ENTER();
    OSSchedLockNestingCtr--;
    /* 这里就解了一层锁，是不是上锁就看这个变量是不是>0        */
    if (OSSchedLockNestingCtr > (OS_NESTING_CTR)0) {
        CPU_CRITICAL_EXIT();
        /* 我想要是能进这里说明你在一个任务里多次调用 OSSchedLock，会
返回一个错误   */
       *p_err = OS_ERR_SCHED_LOCKED;
        return;
    }

#if OS_CFG_SCHED_LOCK_TIME_MEAS_EN > 0u
    OS_SchedLockTimeMeasStop();/*这里就计算咱给任务上锁时间，并看是不
是比以前上锁最长时间还长。*/
```

```
#endif

    CPU_CRITICAL_EXIT();
    /* Scheduler should be re-enabled                          */
    OSSched();
    /* 解锁正常之后就看是不是在上锁的时候有更高优先级任务就绪，有的话
就切换任务*/
    *p_err = OS_ERR_NONE;
}
************************************************************************
*****************************************************/
/************************************************************************
*****************************************************
void  OS_SchedLockTimeMeasStop (void)
{
    CPU_TS_TMR  delta;


    if (OSSchedLockNestingCtr == (OS_NESTING_CTR)0) {
    /* Make sure we fully un-nested scheduler lock             */
        delta = CPU_TS_TmrRd()
        /* 上锁结束就再读一次定时器，但这个上锁时间应该不能长，不然定
时器寄存器会溢出我想最好在一次           */
              - OSSchedLockTimeBegin;
        if (delta > OSSchedLockTimeMax) {
        /* 看这次上锁时间是不是比以前最长的上锁时间还长，要长就给
OSSchedLockTimeMax 赋值       */
            OSSchedLockTimeMax    = delta;
        }
        if (delta > OSSchedLockTimeMaxCur) {
        /* 我总觉得这个同上，也许我对这个系统还了解的还不
够                */
            OSSchedLockTimeMaxCur = delta;
        }
    }
}
************************************************************************
*****************************************************/
```

下面就说这三个，这三个函数都是留给我们写的

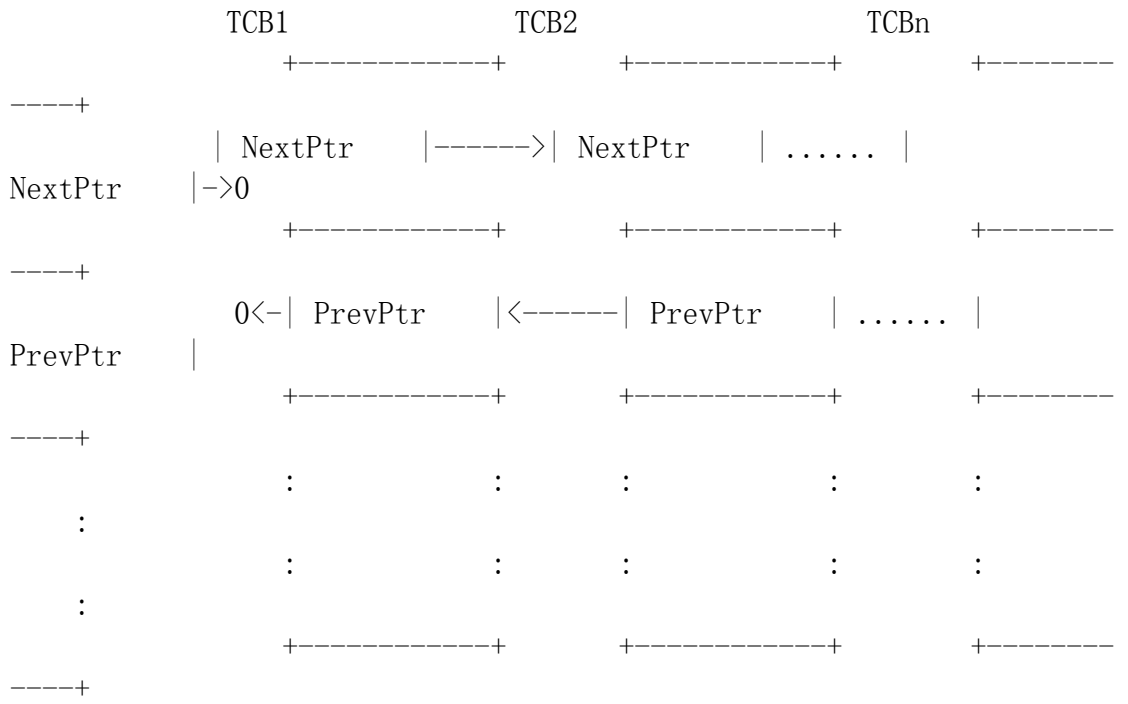
```
    CPU_SR_ALLOC()
    CPU_CRITICAL_ENTER()
    CPU_CRITICAL_EXIT()
```

先说 CPU_SR_ALLOC()的写法，就是定义一个 32 位的 cpu_sr 变量，开始为 0，这就是用来保存硬件状态寄存器的

```
#define  CPU_SR_ALLOC()              CPU_SR  cpu_sr = (CPU_SR)0
```
再说 CPU_CRITICAL_ENTER(),这个用汇编写，两件事，先把当前的状态寄存器的值保存到 cpu_sr 中，然后在状态
寄存器中设置禁中断。（不同的单片机写移植函数会有不同，这里先不说，要是什么是说移植代码再细说）
CPU_CRITICAL_EXIT() 这个就是把 cpu_sr 保存的值赋给状态寄存器。

```
/*********************************************************************
*********************************
#ifdef OS_SAFETY_CRITICAL_IEC61508
    OSSafetyCriticalStartFlag       =  DEF_FALSE;           /*这个为 1
```
的话，就不能建新的任务了*/
```
#endif
```
再说下这个，OSSafetyCriticalStartFlag 这个变量要是为 1 的话，就不能建新的任务，
如果调用下面这个函数 OSSafetyCriticalStartFlag 就会被置位，
```
void  OSSafetyCriticalStart (void)
{
    OSSafetyCriticalStartFlag = DEF_TRUE;
}
```
然后我们看一下这段代码，这是 OSTaskCreate 里的代码，我们先就看这一部分，由这个可以看出要是 OSSafetyCriticalStartFlag
为真，那么就会直接退出 OSTaskCreate，不会建任务了。
```
 *****************************************************
    if (OSSafetyCriticalStartFlag == DEF_TRUE) {
       *p_err = OS_ERR_ILLEGAL_CREATE_RUN_TIME;
        return;
    }
 *****************************************************
*********************************************************************
*********************************/

/*********************************************************************
*********************************
```
好吧，再说下面这两个变量。
```
#if OS_CFG_SCHED_ROUND_ROBIN_EN > 0u
    OSSchedRoundRobinEn             = DEF_FALSE;/*初始化时是不用这个
```
功能 的*/
```
    OSSchedRoundRobinDfltTimeQuanta = OSCfg_TickRate_Hz / 10u;/*这个是
```
默认时间间隔，你可以不用的*/
```
#endif
```
ucos-iii 可以带无限个任务，首先 ucos-iii 提供的优先级不是无限的，能带无限个任务的原因是每个优先级可以带
无限个任务。这个原理就留到创建任务时讲。



（说明：这个链上的任务是就绪的，如果不是就绪态的话就会从这个链上移除，还有一点这个链我没用完整的）
同一优先级上的就绪任务就是如图连在一起的，假设这些任务的优先级是 40，如果 40 之前的优先级任务都休息了当前就绪
的任务优先级最高的就是 40 的话，那个调度时就会把这里的 TCB1 管理的任务拿出来执行，这个优先级上其他的任务
虽然是就绪，但 CPU 就是不让它们干活。
每次调度的时候先做 TCB1 管理的任务。如果 OSSchedRoundRobinEn 置位的话
那么每过 OSSchedRoundRobinDfltTimeQuanta（当然这是默认时间间隔，是你不自己设置的话才会用它，当说到任务创建时
就明白了）时间就会把第一个调到最后，这样这里的 TCB2 就变成第一个了，这样，CPU 再找这个优先级上的就绪任务时就会
找到 TCB2 管理的任务。TCB1 得等，直到它前面的任务，自动放弃（也就是说，这个任务自己将自己移到这个链最后）或是
从就绪态变成别的状态（从这个链上移除）。
然后说下相关的三个函数。
/***
第一个
这个是我们自己调用的，只调用一次就好了，所以别放到任务的 while(1)里
```
void  OSSchedRoundRobinCfg (CPU_BOOLEAN  en,/*是否启用同优先级任务轮转功能，也就是上面解释的*/
                            OS_TICK      dflt_time_quanta,/*时间间隔，OSSInit 中已设，这里我们可以改*/
                            OS_ERR      *p_err)
{
```

```
    CPU_SR_ALLOC();



#ifdef OS_SAFETY_CRITICAL
    if (p_err == (OS_ERR *)0) {
        OS_SAFETY_CRITICAL_EXCEPTION();
        return;
    }
#endif

    CPU_CRITICAL_ENTER();
    if (en != DEF_ENABLED) {
        OSSchedRoundRobinEn = DEF_DISABLED;
    } else {
        OSSchedRoundRobinEn = DEF_ENABLED;           /*这是设置使能*/
    }

    if (dflt_time_quanta > (OS_TICK)0) {
        OSSchedRoundRobinDfltTimeQuanta = dflt_time_quanta; /*这是设
置时间间隔*/
    } else {
        OSSchedRoundRobinDfltTimeQuanta = (OS_TICK)(OSCfg_TickRate_Hz
/ (OS_RATE_HZ)10);
        /*如果我们传的 dflt_time_quanta=0 的话*/
    }
    CPU_CRITICAL_EXIT();
   *p_err = OS_ERR_NONE;
}
**********************************************************************
**********************************/
```

下面的是在任务中调用的，用于任务自己将自己从本优先级的最前面调到最后面。要想看懂这个代码得先知道任务在
系统的组织。这先放一下，等说完任务建立时再说它吧（在代码中我用了中文注示，看一下吧）

```
/**********************************************************************
**********************************
void  OSSchedRoundRobinYield (OS_ERR  *p_err)
{
    OS_RDY_LIST  *p_rdy_list;
    OS_TCB       *p_tcb;
    CPU_SR_ALLOC();
```

```
#ifdef OS_SAFETY_CRITICAL
    if (p_err == (OS_ERR *)0) {
        OS_SAFETY_CRITICAL_EXCEPTION();
        return;
    }
#endif

#if OS_CFG_CALLED_FROM_ISR_CHK_EN > 0u
    if (OSIntNestingCtr > (OS_NESTING_CTR)0) {              /* Can't
call this function from an ISR                    */
       *p_err = OS_ERR_YIELD_ISR;
        return;
    }
#endif

    if (OSSchedLockNestingCtr > (OS_NESTING_CTR)0) {
    /* Can't yield if the scheduler is locked                 */
       *p_err = OS_ERR_SCHED_LOCKED;
        return;
    }

    if (OSSchedRoundRobinEn != DEF_TRUE) {
    /* Make sure round-robin has been enabled                 */
       *p_err = OS_ERR_ROUND_ROBIN_DISABLED;
        return;
    }

    CPU_CRITICAL_ENTER();
    p_rdy_list = &OSRdyList[OSPrioCur];
    /* Can't yield if it's the only task at that priority     */
    if (p_rdy_list->NbrEntries < (OS_OBJ_QTY)2) {
        CPU_CRITICAL_EXIT();
       *p_err = OS_ERR_ROUND_ROBIN_1;
        return;
    }

    OS_RdyListMoveHeadToTail(p_rdy_list);
    /* 将当前优先级的最前面任务移到这个优先级链的最后             */
    p_tcb = p_rdy_list->HeadPtr;
    /* 取新的当前优先级最前面的任务                  */
    if (p_tcb->TimeQuanta == (OS_TICK)0) {
    /* 给它一个时间片,到时间它会被移到这个优先级链的最后           */
        p_tcb->TimeQuantaCtr = OSSchedRoundRobinDfltTimeQuanta;
```

```
    } else {

    /*记住，这里我一直强调的是当前优先级链。也不知道前面我有说明白没*/
        p_tcb->TimeQuantaCtr = p_tcb->TimeQuanta;
        /* Load time slice counter with new time                    */
    }

    CPU_CRITICAL_EXIT();

    OSSched();
    /* Run new task                                                 */
   *p_err = OS_ERR_NONE;
}

*********************************************************************************
*********************************/
```

下面的函数是在 OSTimeTick（时钟节拍）中调用的，传入的是当前有任务活动的优先级的任务链，它会把时间戳减 1
减到 0 就把最前面的任务移到这个优先级任务链的最后。p_tcb->TimeQuanta 建任务时给的，当前面的任务被移到最后时
这个任务就变成最前面了，同时 TimeQuanta 给 p_tcb->TimeQuantaCtr 赋值。如果 TimeQuanta=0，那就用 OSSchedRoundRobinDfltTimeQuanta

```
/*********************************************************************************
*********************************
void  OS_SchedRoundRobin (OS_RDY_LIST  *p_rdy_list)
{
    OS_TCB   *p_tcb;
    CPU_SR_ALLOC();



    if (OSSchedRoundRobinEn != DEF_TRUE) {
    /* Make sure round-robin has been enabled                       */
        return;
    }

    CPU_CRITICAL_ENTER();
    p_tcb = p_rdy_list->HeadPtr;
    /* Decrement time quanta counter                                */

    if (p_tcb == (OS_TCB *)0) {
        CPU_CRITICAL_EXIT();
        return;
    }
```

```
    if (p_tcb == &OSIdleTaskTCB) {
        CPU_CRITICAL_EXIT();
        return;
    }

    if (p_tcb->TimeQuantaCtr > (OS_TICK)0) {
        p_tcb->TimeQuantaCtr--;
    }

    if (p_tcb->TimeQuantaCtr > (OS_TICK)0) {
    /* Task not done with its time quanta                     */
        CPU_CRITICAL_EXIT();
        return;
    }

    if (p_rdy_list->NbrEntries < (OS_OBJ_QTY)2) {
    /* See if it's time to time slice current task            */
        CPU_CRITICAL_EXIT();
        /* ... only if multiple tasks at same priority            */
        return;
    }

    if (OSSchedLockNestingCtr > (OS_NESTING_CTR)0) {
    /* Can't round-robin if the scheduler is locked           */
        CPU_CRITICAL_EXIT();
        return;
    }

    OS_RdyListMoveHeadToTail(p_rdy_list);
    /* Move current OS_TCB to the end of the list             */
    p_tcb = p_rdy_list->HeadPtr;
    /* Point to new OS_TCB at head of the list                */
    if (p_tcb->TimeQuanta == (OS_TICK)0) {
    /* See if we need to use the default time slice           */
        p_tcb->TimeQuantaCtr = OSSchedRoundRobinDfltTimeQuanta;
    } else {
        p_tcb->TimeQuantaCtr = p_tcb->TimeQuanta;
        /* Load time slice counter with new time                  */
    }
    CPU_CRITICAL_EXIT();
}
*********************************************************************
*********************************/
```

```
/*********************************************************************************************************
*********************************
    OS_AppTaskCreateHookPtr = (OS_APP_HOOK_TCB )0;


    OS_AppTaskDelHookPtr    = (OS_APP_HOOK_TCB )0;
    OS_AppTaskReturnHookPtr = (OS_APP_HOOK_TCB )0;

    OS_AppIdleTaskHookPtr   = (OS_APP_HOOK_VOID)0;
    OS_AppStatTaskHookPtr   = (OS_APP_HOOK_VOID)0;
    OS_AppTaskSwHookPtr     = (OS_APP_HOOK_VOID)0;
    OS_AppTimeTickHookPtr   = (OS_APP_HOOK_VOID)0;
*********************************************************************************************************
*********************************/
```

下面说下这些勾子函数，这些是在 os.h 中定义的举一例

```
OS_EXT            OS_APP_HOOK_TCB          OS_AppTaskCreateHookPtr
typedef  void                           (*OS_APP_HOOK_TCB)(OS_TCB *p_tcb);
```

这样用 OS_APP_HOOK_TCB 声明的是函数指针，这个指针指向一个返回值为空，并有一个输入参数的函数

在 os_app_hooks.c 中有 App_OS_SetAllHooks(void)函数。在这个函数里

。。。。。。

```
OS_AppTaskCreateHookPtr = App_OS_TaskCreateHook
```

。。。。。。

而在 os_app_hooks.c 中

```
void  App_OS_TaskCreateHook (OS_TCB  *p_tcb)
{
    (void)&p_tcb;
}
```

最后在大家写移植函数时就这样写

```
void  OSTaskCreateHook (OS_TCB  *p_tcb)
{
#if OS_CFG_APP_HOOKS_EN > 0u
    if (OS_AppTaskCreateHookPtr != (OS_APP_HOOK_TCB)0) {
        (*OS_AppTaskCreateHookPtr)(p_tcb);
    }
#else
    (void)p_tcb;
#endif
}
```

然后再说 OS_AppTaskCreateHookPtr 这些在 App_OS_SetAllHooks(void)设置

```
void  App_OS_ClrAllHooks (void)清理
```

这个我们可以自己调用在一个任务中调用，我想一般是在我们写个初始化任务中调用这些。用过 ucos 的话就知道，一

必要的资源在 OSInit()里初始化了，但有些是开发人员认为客观必要的，那么就会在第一个任务中去初始化，如果定时
中断，和这些勾子函数。
**********/
下面是优先级链的初始化，在调用 OSInit 运行到这时，还没有任务建立，所以优先级链为空的。还是看下代码：
/**********
```
void  OS_PrioInit (void)
{
    CPU_DATA  i;
/*在 os.h 中定义了 OS_PRIO_TBL_SIZE 大小。
#define  OS_PRIO_TBL_SIZE          (((OS_CFG_PRIO_MAX - 1u) /
DEF_INT_CPU_NBR_BITS) + 1u)
```
OS_CFG_PRIO_MAX 是优先级值的最大数，是在 os_cfg.h 中定义的，os_cfg.h 是配置文件，我们可以根据我们实际所需，
在这个文件中定制内核。目前在 os_cfg.h 中定义为 64：
```
#define OS_CFG_PRIO_MAX                  64u
```
至于 DEF_INT_CPU_NBR_BITS 我们可以定义（在 lib_def.h 中定义），这个值可以是 8，16，32。说到这，可以有人会迷茫，好了这时就不得不
说下在内核中优先级的管理方式，这样大家就能看懂这段代码了，这段代码很简单。*/
```
    for (i = 0u; i < OS_PRIO_TBL_SIZE; i++) {
         OSPrioTbl[i] = (CPU_DATA)0;
    }
}
```

这里说下优先级的管理方式，
我们定义 OS_CFG_PRIO_MAX=64
DEF_INT_CPU_NBR_BITS=8（OSPrioTbl[]这个数组里的数据也是定义为 8 位长）
现在我们建了第一个任务，优先级是 20，这时我们看一下优先级插入代码
。。。。。。
```
void  OS_PrioInsert (OS_PRIO  prio)
{
    CPU_DATA  bit;
    CPU_DATA  bit_nbr;
    OS_PRIO   ix;


    ix             = prio / DEF_INT_CPU_NBR_BITS;
    bit_nbr        = (CPU_DATA)prio & (DEF_INT_CPU_NBR_BITS - 1u);
```

```
    bit             = 1u;
    bit          <<= (DEF_INT_CPU_NBR_BITS - 1u) - bit_nbr;
    OSPrioTbl[ix] |= bit;
}
```

。。。。。。

因为我们假设本次优先级为 20，也就是说 prio=20,DEF_INT_CPU_NBR_BITS=8;

ix=2,

bit_nbr = 20 & 7=4(二进制计算是 00010100&00000111=00000100)其实这就是相当于 20 除以 8 求余数。这是为了得到一个 8 以下的数。

bit_nbr   = (CPU_DATA)prio & (DEF_INT_CPU_NBR_BITS - 1u);这句其实是以前的方法

看过 ucosiir 就知道

{ix=(prio>>3)

bit_nbr=(prio&7)}

bit=1

bit<<= (DEF_INT_CPU_NBR_BITS - 1u) - bit_nbr;左移 3 位

OSPrioTbl[ix] |= bit

这样就是 20 分成两部分，一部分是 8 的部数 2，一部分是 8 的余数 4，

然后就会在 OSPrioTbl[2]=|8，

OSPrioTbl[2]看成二进制，

```
|+|+|+|+|+|+|+|+|
|0|0|0|0|1|0|0|0|
|+|+|+|+|+|+|+|+|
```

我们可以这么看，一个数 num 如果 16<=num<24,那么这个优先级就一定会在 OSPrioTbl[2]的某一位设个 1，如果 num 越大

就会在 OSPrioTbl[2]更低的位置设 1，如果我们又建一个任务用的优先级是 21 的话那么就会在现在这个 1 后面再设一个 1。

优先级 prio 的数值越小，优先级越高。当任务是就绪的，也就是可以工作的时候才会去设 OSProTbl[]中的位。好了，

先不说了，等下次说完任务的建立和调度时再细说。

OS_RdyListInit();这是初始化就绪任务链，一个任务就绪时，不只在 OSPrioTbl 中设置一下，这里也要设置，这个也

留到建任务时说。

剩下的这回也不说了，等说到相应资源时再说，下面说任务建立与调度。

**********/