标签： SmallRTOS RTOS

SmallRTOS 用户函数参考手册

根据陈明计 SmallRTOS 使用手册整理，便于在编程时查找

OS_INT_ENTER()

名称：OS_INT_ENTER()
所属文件：Os_cpu.h
原型：宏
功能描述：OS_INT_ENTER()通知 SMALLRTOS 一个中断服务函数正在运行，这样 RTOS 可以跟踪中断嵌套情况，通常与 OSIntExit()联合使用。
编译开关：EN_OS_INT_ENTER
调用者：中断

注意点：用户任务不能调用该函数。其实现与目标系统相关，移植时由用户实现。

OSClearSignal()

名称：OSClearSignal()
所属文件：Os_core.h
原型：void OSClearSignal(uint8 TaskID)
功能描述：OSClearSignal()无条件使任务休眠。如果它使当前任务休眠，则还需要调用 OSSched(),才会切换到其他任务。
编译开关：无
调用者：内核函数
参数： TaskID
注意点：用户任务不能调用该函数.

OSIntExit()

名称：OSIntExit()
所属文件：Os_core.h
原型：void OSIntExit(void)
功能描述：OSIntExit()通知 RTOS 一个中断服务函数已经执行完毕，这样 RTOS 可以跟踪中断嵌套情况。它通常与 OS_INT_ENTER()联合使用。当最后一层嵌套的中断服务函数执行完毕后，如果有更高优先级的任务就绪，则 RTOS 就会调用任务调度函数。在这种情况下，中断返回到更高优先级的任务，而不是返回中断了的任务

编译开关：无
调用者：中断
调用模块：OSIntCtxSW
注意点：用户任务不能调用该函数,只要是受 RTOS 管理的中断服务函数，在其退出之前必须调用此函数

OSIntSendSignal()

名称：OSIntSendSignal()
所属文件：Os_q.h
原型：void OSintSendSignal(uint9 TaskId)
功能描述：OSIntSendSignal()无条件指使指定任务就绪。如果指定任务比当前任务优先级高，则在全部中断退出后或调用 OSSched()后，开始执行高优先级任务。
编译开关：无
调用者：中断、内核函数
参数：TaskId:任务 ID
注意点：用户任务一般不需要调用该函数。

例：

```
void comm(void) interrupt 4
{
 OS_INT_ENTER();
 if(RI==1)
 {
     RI = 0;
     OSIntSendSignal(RECEIVE_TASK_ID);
  }
 if(TI==1)
  {
     TI=0;
     OSIntSendSignal(1);
  }
 OSIntExit();
}
```

OSQAccept()

名称：OSQAccept()
所属文件：Os_q.h

原型：uint8 OSQAccept(uint8 data * Ret, uint8 OS_Q_MEM_SEL *Buf)

功能描述：OSQAccept()函数检查消息队列中是否已经有需要的消息，他不同于 OSQPend(), 如果没有需要的消息，则 OSQAccept()并不使任务休眠。如果任务已经到达，则该消息被传递到用户任务。通常中断调用该函数，因为中断不允许等待。
编译开关：EN_OS_Q_ACCEPT
调用者：用户任务
参数：Ret: 返回消息；
　　Buf:指向队列的指针
返回值：NOT_OK:参数错误
　　OS_Q_OK：收到消息
　　OS_Q_NOT_OK：无消息
注意点：必须先建立消息队列，然后再使用。

示例：

```
void Task(void)
{
　　uint8 data *Ret;

　　while(1)
　　{
　　　　if(OSQAccept(CommQ) == OS_Q_OK)  //检查消息队列
　　　　{
　　　　　　.....//处理接收的消息
　　　　}
　　　　else {　}//，没有消息
　　}
}
```

OSQCreate()

名称：OSQCreate()
所属文件：Os_q.c
原型：uint8 OSQCreate(uint8 OS_Q_MEM_SEL *Buf, uint8 SizeOfBuf)

功能描述：OSQCreate()函数建立一个消息队列。任务或中断可以通过消息队列向其它一个或

多个任务发送消息,消息是一个 uint8 整型数 ，在不同应用中有不同含义。
编译开关：无
调用者：用户任务
参数：Buf:为队列分配的存储空间地址
SizeOfBuf:为队列分配的存储空间大小
注意点：OSQCreate()不分配内存，内存由用户自己分配。在 KEIL 中 OS_Q_MEM_SEL 一般定义为 xdata 或 idata

示例

```
uint8 OS_Q_MEM_SEL serialData[14];
void main(void)
{
...

OSQCreate(SerialData,14);
...

OSStart();

}
```

例 2：

```
uint8 OS_Q_MEM_SEL CommandData[16];        //给命令消息队列分配的队列空间
....

/* 建立所需要的消息队列 */
OSQCreate(CommandData,16);
.........
uint8 data temp;
while (1)
{
    OSQPend(&temp,CommandData,0);  /* 等待 CommandData 消息队列中的消息，存入 temp */
    /*收到消息才执行*/
  .........
}

.........
while(1)
{
    ......
      OSQPost(CommandData,temp);        /* 发送消息 */
```

```
    ......
}
```

OSQFlush()

名称：OSQFlush()
所属文件：Os_q.c
原型：void OSQFlush(uint8 OS_Q_MEM_SEL *Buf)

功能描述：OSQFlush()函数清空消息队列并且忽略发送到队列的所有消息，消息是一个 uint8 整型数，在不同应用中有不同的含义。
编译开关：EN_OS_Q_FLUSH
调用者：任务、中断
参数：Buf:指向队列的指针
注意点：必须先建立消息，然后在使用

示例

```
void Task()
{

    while(1)
    {
        if(OSQPost(SerialData,GetSerial())==OS_Q_FULL)
        {
            OSQFlush(SerialData);
        }
    }
}
```

OSQIntPost()

名称：OSQIntPost()
所属文件：Os_q.c
原型：uint8 OSQIntPost(uint8 OS_Q_MEM_SEL *Buf, uint8 Data)

功能描述：OSQIntPost()在中断或内核函数中使用，OSQIntPost()通过消息队列向任务发送消

息。
消息是一个 uint8 整型数，在不同应用中有不同的含义。如果消息队列已经存满，则返回错误码。
如果有任何等待消息的任务在等待消息，则这些任务中的最高级任务将得到消息。如果得到消息的任务比当前的任务优先级高，则在所有中断退出后或调用 OSSched()后，高优先级的任务将得到消息而恢复执行。消息队列是先入先出机制的，先入队列消息先被传递给任务。

编译开关：EN_OS_Q_INT_POST, EN_OS_Q_POST
调用者：中断、内核函数
参数：Buf:指向队列指针
　　Data:消息数据
返回值：OS_Q_FULL:队列满
　　OS_Q_OK:发送成功
调用模块：OSIntSendSignal,OS_ENTER_CRITICAL,OS_EXIT_CTRITICAL
注意：必须先建立消息队列，然后再使用。用户任务一般使用 OSQPost()

示例:

```
void comm(void) interrupt4
{
OS_INT_ENTER();
if(RI==1)
{
      RI = 0;
      if(OSQIntPost(SerialData,SBUF) == OS_Q_OK)
      {
            /* 将消息放入队列中*/
    }
      else
      {
            /*消息队列已满*/
      }
      OSIntExit();
}
```

OSQIntPostFront()

名称：OSQIntPostFront()
所属文件：Os_q.c
原型：uint8 OSQIntPostFront(uint8 OS_Q_MEM_SEL *Buf, uint8 Data)

功能描述：OSQIntPostFront()函数在中断或内核函数中使用。OSQIntPostFront()通过消息队

列向任务发送消息。OSQIntPostFront()和 OSQIntPost()非常相似，不同之处是 OSQIntPostFront()报将要发送的消息放在消息队列的最前面，就是按照后入先出方式工作，而不是按照先入先出的方式工作。

编译开关: EN_OS_INT_POST_FRONT, EN_OS_Q_POST_FRONT
调用者：中断、内核函数
参数：Buf:指向队列指针
　　Data:消息数据
调用模块：OSIntSendSignal, OS_ENTER)CRITICAL, OS_EXIT_CRITICAL
注意点：用户任务一般使用 OSQFront()

示例

```
void comm(void) interrupt 4
{
	OS_INT_ENTER();
	if(RI == 1)
	{
		RI = 0;
		if(OSQIntPostFront(SerialData,SBUF) == OS_Q_OK)
		{
			/*将消息放入队列中*/
		}
		else{
			/*消息队列已满*/
		}
	}
	OSIntExit();
}
```

OSQNMsgs()

名称：OSQNMsgs()
所属文件：Os_q.c
原型：uint8 OSQNMsgs(uint8 OS_Q_MEM_SEL *Buf)

功能描述：OSQNMsgs()用于取得制定消息队列中的消息数。
编译开关：EN_OS_Q_NMsgs
调用者：用户任务
参数：Buf:指向队列指针
返回值：消息数
调用模块：OS_ENTER)CRITICAL, OS_EXIT_CRITICAL

示例

```
void Task(void)
{
      OSQCreate(SerialData,14);
      while(1)
      {
            if(OSQNMsgs(SerialData)<NBYTE)
            {
                  /*处理消息*/
            }
            else
            {
            }
      }
}
```

OSQPend()

名称：OSQPend()
所属文件：Os_q.c
原型：uint8 OSQPend(uint8 *Ret, uint8 OS_Q_MEM_SEL *Buf,uint8 Tick)

功能描述：OSQPend()用于任务等待消息。消息通过中断或另外的任务向发送给需要的任务。如果调用 OSQPend()函数时，队列中已经存在消息，则消息被返回给 OSQPend()函数的调用者，队列中清除该消息。如果调用 OSQPend()函数时队列中没有需要的消息，则 OSQPend()将该任务加入等待队列中，并使当前任务休眠，直到这个任务得到需要的消息或超出定义的超时时间为止。如果多个任务等待同一个消息，则 rtos 默认他们中间优先级最高的任务取得消息。

编译开关：EN_OS_Q_PENT
调用者：用户任务
参数：Ret:返回的消息
      Buf:指向队列的指针
      Tick:等待时间
返回值：NOT_OK:参数错误
       OS_Q_OK:收到消息
       OS_Q_TMO:超时
      OS_Q_NOT_OK:无消息
调用模块：
OSRunningTaskID,OSClearSignal,OSSched,OS_ENTER_CRITICAL,OS_EXIT_CRITICAL
注意：先建立消息队列再使用。不允许中断调用该函数。

示例

```
void Task(void)
{
	while(1)
	{
		..
		if(OSQPend(&temp1,SerialData,20)==OS_Q_TMO)
		{
			/*。超时处理 */
		}
		else
		{
			/*取得消息*/
		}
		..
	err:
		..
	}
}
```

例 2：

```
uint8 OS_Q_MEM_SEL CommandData[16];		//给命令消息队列分配的队列空间
....

/* 建立所需要的消息队列 */
OSQCreate(CommandData,16);
.........
uint8 data temp;
while (1)
{
	OSQPend(&temp,CommandData,0);  /* 等待 CommandData 消息队列中的消息，存入
temp */
	/*收到消息才执行*/
   .........
}

.........
while(1)
{
```

```
        OSQPost(CommandData,temp);        /* 发送消息 */
    ......
}
```

OSQPost()

名称：OSQPost()
所属文件：Os_q.c
原型：uint8 OSQPost(uint8 OS_Q_MEM_SEL *Buf,uint8 Data)

功能描述：OSQPost()通过消息队列向任务发送消息。

编译开关：EN_OS_Q_POST
调用者：用户任务
参数：Buf:指向队列的指针
    Data:消息数据
返回值：OS_Q_FULL:队列已满
     OS_Q_OK:发送成功

调用模块：OSQIntPost,OSSched

注意：先建立消息队列再使用。

示例

```
void Task(void)
{
    while(1)
    {
        if(OSQPost(SerialData,GetSerial())==OS_Q_OK)
        {
            /*将消息放入消息队列中*/
        }
        else
        {
            /*消息队列已满*/
        }
    }
}
```

例 2.

```
uint8 OS_Q_MEM_SEL CommandData[16];          //给命令消息队列分配的队列空间
....

/* 建立所需要的消息队列 */
OSQCreate(CommandData,16);
.........
uint8 data temp;
while (1)
{
      OSQPend(&temp,CommandData,0);  /* 等待 CommandData 消息队列中的消息，存入
temp */
      /*收到消息才执行*/
   .........
}

..........
while(1)
{
      ......
         OSQPost(CommandData,temp);        /* 发送消息 */
      ......
}
```

OSQPostFront()

名称：OSQPostFront()
所属文件：Os_q.c
原型：uint8 OSQPostFront(uint8 OS_Q_MEM_SEL *Buf,uint8 Data)

功能描述：OSQPostFront()通过消息队列向任务发送消息。与 OSQPost()不同的是，OSQPostFront()把将要发送的消息放在消息队列的最前端，也就是按照后入后出的工作方式，而不是先入先出的工作方式。

编译开关：EN_OS_Q_POST_FRONT
调用者：用户任务
参数：Buf:指向队列的指针
      Data:消息数据
返回值：OS_Q_FULL:队列已满
       OS_Q_OK:发送成功

调用模块：OSQIntPostFront,OSSched

注意：先建立消息队列再使用。

示例

```
void Task(void)
{
      while(1)
      {
            if(OSQPostFront(SerialData,GetSerial())==OS_Q_OK)
            {
                  /*将消息放入消息队列中*/
            }
            else
            {
                  /*消息队列已满*/
            }
      }
}
```

OSQSize()

名称：OSQSize()
所属文件：Os_q.c
原型：uint8 OSQSize(uint8 OS_Q_MEM_SEL *Buf)
功能描述：OSQSize()用于取得指定消息队列消息总容量。
编译开关：EN_OS_Q_SIZE
调用者：用户任务
参数：Buf:指向队列的指针
返回值：消息队列总容量
调用模块：OS_ENTER_CRITICAL OS_EXIT_CRITICAL

OSRunningTaskID()

名称：OSRunningTaskID()
所属文件：os.h
原型：宏
功能描述：OSRunningTaskID()返回当前正在运行的任务的 ID.任务的 ID 与任务优先级一一对应，ID 为 0 的任务优先级最高。任务的 ID 在运行中不能改变。

编译开关：无
调用者：用户任务、内核函数
参数：无
返回值：当前正在运行的任务的 ID
示例：

```
void TaskB(void)
{
      while(1)
      {
            ShowChar = OSRunningTaskID()+'0';
            OSSendSignal(0);
            OSWait(K_TMO,1);
      }
}
```

OSSemAccept()

名称：OSSemAccept()
所属文件：Os_sem.c
原型：uint8 OSSemAccept(uint8 index)
功能描述：OSSemAccept()函数查看设备是否就绪或事件是否发生。与 OSSemPend()不同，如果设备没有就绪，则 OSSemAccept()函数并不使当前任务睡眠。一般在中断中用该函数查询信号量。

编译开关：EN_OS_SEM_ACCEPT.
调用者：用户任务
参数：index:信号量索引
返回值：NOT_OK:参数错误
        OS_SEM_OK:得到信号量
        OS_SEM_NOT_OK:没有得到信号量

调用模块：OSClearSignal,OSSched,OS_ENTER_CRITICAL,OS_EXIT_CRITICAL

注意：必须先建立信号量，然后再使用

```
void Task(void)
{
      while(1)
      {
            if(OSSemAccept()==OS_SEM_OK)
                  /* 查看设备是否就绪或事件是否发生*/
```

```
        {
            /*就绪就执行代码*/
        }
    }
}
```

OSSemCreate()

名称：OSSemCreate()
所属文件：Os_sem.c
原型：uint8 OSSemCreate(uint8 index,uint8 Data)
功能描述：OSSemCreate()建立并初始化一个信号量。

信号量的作用是：
1. 允许一个任务或其他任务或中断同步
2. 取得设备的使用权
3. 标志事件发生

编译开关：EN_OS_Q_SIZE
调用者：用户任务
参数：index:信号量索引
Data:信号量初始值

返回值：NOT_OK:没有这个信号量
OS_SEM_OK:成功
注意：必须先建立信号量，然后再使用

示例：

```
void Task(void)
{
    ..
    OSSemCreate();
    ...
    while（1）
    {
        }
    ..
}
```

例 2.

```
/*  分配信号量索引    */
#define IICSem          0
#define PCF8563Sem      1
.....

/*  初始化使用到的信号量  */
   OSSemCreate(IICSem, 1);              //初始值均为  1
   OSSemCreate(PCF8563Sem, 1);
.....

while (1)
{
   ......

      OSSemPend(PCF8563Sem,0);   /*无限等待信号量 PCF85863Sem   */

    /*得到信号量才能执行  */
         ..........

      OSSemPost(PCF8563Sem); /*  置起信号量,其它任务可用  */
}
```

OSSemIntPost()

名称：OSSemIntPost()
所属文件：Os_sem.c
原型：uint8 OSSemIntPost(uint8 index)
功能描述：OSSemIntPost()函数置指定的信号量。如果指定的信号量是 0 或大于 0,则 OSSemIntPost()函数递增该信号量并返回。如果有任何在等待信号量，则最高优先级的任务将得到信号量并进入就绪状态。与 OSSemPost()不同，任务调度函数并不被执行，OSSemIntPost()返回当前任务。

编译开关：EN_OS_SEM_INT_POST, EN_OS_SEM_POST
调用者：用户任务、内核函数
参数：index:信号量索引
返回值：NOT_OK:参数错误
      OS_SEM_OK:发送成功
调用模块：OSIntSendSignal, OS_ENTER_CRITICAL,OS_EXIT_CRITICAL
注意：一般在中断函数中调用该函数

示例：

```
#if EN_OS_SEM_POST>0
      uint8 OSSemPost(uint8 index)
{
      if(OSSemIntPost(index) == 0S_SEM_OK)
      {
            OSSched();
            return OS_SEM_OK;
      }
      else
            {
             return NOT_OK;
      }
}
#endif
```

OSSemPend()

名称：OSSemPend()
所属文件：Os_sem.c
原型：uint8 OSSemPend(uint8 index,uint8 Tick)

功能描述：OSSemPend()用于任务试图取得设备的使用权，需要和其它任务或中断同步，任务需要等待特定事件发生的场合。如果任务需要调用 OSSemPend()函数时，信号量的值大于0,则OSSemPend()函数递减该值并返回OS_SEM_OK.如果该值等于0,则OSSemPend()函数将当前任务加入该信号量的等待队列，并使当前任务睡眠，直到其它任务或中断置起信号量或超出预定时间。如果多个任务等待同一个信号量，则当信号量被置起时，RTOS 是其中优先等级最高的任务取得信号量并恢复执行。

编译开关：EN_OS_SEM_PENT
调用者：用户任务
参数：index：信号量索引
      Tick:等待时间
返回值：NOT_OK:参数错误
       OS_SEM_OK:收到信号量
       OS_SEM_TMO:超时
      OS_SEM_NOT_OK:没有得到信号量
调用模块：
OSRunningTaskID,OSClearSignal,OSSched,OS_ENTER_CRITICAL,OS_EXIT_CRITICAL
注意：先建立信号量再使用。不允许中断调用该函数。

示例

```
void TaskA(void)
{
      while(1)
      {
       OSSemPend(0,0);
       ... /*只有得到信号量置起，该任务才能执行*/
       ...
      }

}
```

例 2：

```
/*  分配信号量索引    */
#define IICSem          0
#define PCF8563Sem      1
.....

/*  初始化使用到的信号量  */
   OSSemCreate(IICSem, 1);              //初始值均为  1
   OSSemCreate(PCF8563Sem, 1);
.....

while (1)
{
   ......

      OSSemPend(PCF8563Sem,0);   /*无限等待信号量 PCF85863Sem   */

    /*得到信号量才能执行  */
         ..........

      OSSemPost(PCF8563Sem); /*  置起信号量,其它任务可用  */
}
```

OSSemPost()

名称：OSSemPost()

所属文件：Os_sem.c
原型：uint8 OSSemPost(uint8 index)

功能描述：OSSemPost()置起指定的信号量。如果指定的信号量是 0 或大于 0,则 OSSemPost()函数递增该信号量并返回。如果有任何在等待信号量，则最高优先级的任务将得到信号量并进入就绪状态。任务调度函数将进行任务调度，决定当前运行的任务是否依然为最高优先级就绪状态的任务

编译开关：EN_OS_SEM_POST
调用者：用户任务
参数：index:信号量索引
返回值：NOT_OK:参数错误
　　OS_SEM_OK:发送成功

调用模块：OSSemIntPost,OSSched

注意：先建立信号量再使用，中断中不允许调用该函数。

示例：

```
void TaskA(void)
{
      OSSemCreate(0,1);
      while(1)
      {
            OSSemPend(0,0);
            /*使用设备*/
            OSSemPost(0);
            ..
      }
}
```

例 2：

```
/*  分配信号量索引    */
#define IICSem          0
#define PCF8563Sem      1
.....

/*  初始化使用到的信号量  */
   OSSemCreate(IICSem, 1);              //初始值均为  1
   OSSemCreate(PCF8563Sem, 1);
```

```
while (1)
{
   ......

      OSSemPend(PCF8563Sem,0);   /*无限等待信号量 PCF85863Sem  */

    /*得到信号量才能执行  */
         ..........

      OSSemPost(PCF8563Sem); /*  置起信号量,其它任务可用  */
}
```

OSSemQuery()

名称：OSSemQuery()
所属文件：Os_sem.c
原型：uint8 OSSemQuery(uint8 index)

功能描述：OSSemQuery()函数获取某个信号量的值

编译开关：EN_OS_SEM_QUERY
调用者：用户任务
参数：index:信号量索引
返回值：信号量的值
调用模块：OS_ENTER_CRITICAL, OS_EXIT_CRITICAL

注意：先建立信号量再使用。

```
void Task(void)
{
      while（1）
      {
            if(OSSemQuery(0)>0)
            {
             /*有信号量时处理信号量*/
            }
            else
            {
             /*没有信号量时进行其它处理*/
            }
```

```
    }
}
```

OSSendSignal()

名称：OSSendSignal()
所属文件：Os_core.c
原型：void OSSendSignal(unit8 TaskId)
功能描述：OSSendSignal()无条件使指定任务就绪。如果指定任务的优先级比当前任务的优先级高，则高优先级的任务得以执行。
编译开关：无
调用者：用户任务
参数：TaskId：任务 ID
返回值：无
调用模块：OSSched

示例：

```
void Task(void)
{
while(1)
{
      ShowChar = OSRunningTaskID()+'0';
      OSSendSignal(0);
      OSWait(K_TMO,1);
}
}
```

OSStart()

名称：OSStart()
所属文件：Os_cpu.c
原型：void OSStart(void)

功能描述：OSStart()启动 small RTOS 的多任务环境

编译开关：无
调用者：只能是初始代码
参数：无
返回值：无
调用模块：LoadCtx

注意：在用户程序中只能调用一次，第二次调用则有可能使系统崩溃。在调用 OSStart()之前，不能使能全局中断标志，在调用后，自动使能全局中断标志

OSTimeTick()

名称：OSTimeTick()
所属文件：Os_cpu.c
原型：void OSTimeTick(void)
功能描述：每次系统节拍，small RTOS 都执行 OSTimeTick()函数。OSTimeTick()检查处于延时状态的任务是否到达延时时间，或者是正在等待的任务是否 9 超时。

编译开关：无
调用者：时钟中断或用户任务
参数：无
返回值：无
调用模块：无
注意：在任务或中断都可以调用 OSTimeTick(),若在任务中调用，则任务的优先级应该很高（任务 ID 值很小），这是因为 OSTimeTick()负责所有任务的延时操作。

示例：

```
#if EN_OS_INT_ENTER >0
#pragma disable /*除非最高优先级中断，否则必须加上这一句*/
#endif

void OSTickISR(void) interrupt OS_TIME_ISR
{
#if TICK_TIMER_SHARING>1
      static uint8 TickSum="0";

      TickSum = (TickSum +1)% TICK_TIMER_SHARING
      if(TickSum!=0) return;
#endif

#if EN_OS_INT_ENTER>0
      OS_INT_ENTER(); //中断开始处理
#endif

#if USER_TICK_TIMER_EN == 1
      UserTickTimer();  //用户函数
#endif
```

```
#if EN_TIMER_SHARING >0
        OSTimeTick(); //调用系统时钟处理函数
#else
        OSIntSendSignal(TIME_ISR_TASK_ID);
#endif
        OSIntExit();//中断处理结束
}
```

OSTaskDel()

名称：OSTaskDel()
所属文件：Os_Core.c
原型：uint8 OSTaskDel(uint8 TaskID) small
功能描述：删除任务

例：

```
void TaskB(void)
{
   uint8 i;
   for (i = 0; i < 10; i++)
   {
      PC_DispChar(20, 10, (i % 10) + '0', DISP_FGND_LIGHT_GRAY);
      OSWait(K_TMO, 15);
   }
   OSTaskDel(OSRunningTaskID());
}



void TaskD(void)
{
   uint8 i;

   for (i = 0; i < 30; i++)
   {
      PC_DispChar(20, 5, (i % 10) + '0', DISP_FGND_LIGHT_GRAY);
      OSWait(K_TMO, 20);
   }
   OSTaskDel(0);
}
```

OSTaskSpend()

函数名称: OS_TaskSuspend
功能描述: 使指定任务休眠，但不进行任务切换
输　入: TaskID： 任务 ID

原型：void OS_TaskSuspend(uint8 TaskID)　small;

例：
OS_TaskSuspend(OSRunningTaskID()); /* 任务进入等待状态 */

OSVersion()

名称：OSVersion()
所属文件：Os.h
原型：宏
功能描述：取得当前 RTOS 的版本号
编译开关：无
调用者：用户任务、内核函数
参数：无
返回值：SMALL RTOS 的版本号为*100,如果当前版本号为 1.21,则返回 121
调用模块：无

OSWait()

名称：OSWait()
所属文件：Os.c
原型：uint8 OSWait(uint8 typ,uint8 ticks)

功能描述：OSWait()函数使当前任务睡眠，它可以由以下方式恢复执行：指定 typ 为等待超时，在一个时间间隔后被唤醒；指定 typ 为等待信号，另一个任务或中断可以把它唤醒（通过 OSSendSignal()或 OSIntSendSignal 或是它们的组合），若其中任意一个事件出现，则任务恢复执行。

编译开关：无
调用者：用户任务、内核函数
参数：typ:等待事件类型。
　　K_SIG:等待信号；
　　K_TMO:等待超时.

或者是其中任意值的按位“或”
返回值：NOT_OK:参数错误
TMO_EVENT:超时到
SIG_EVENT:有信号
调用模块：OSIntSendSignal, OSSched.

示例：

```
void TaskB(void)
{
while(1)
{
	ShowChar = OSRunningTaskID+'0';
	OSSendSignal(0);
	OSWait(K_TMO,1);
}
}
```

OS_ENTER_CRITICAL()

#define OS_ENTER_CRITICAL()　　EA=0,Os_Enter_Sum++

关闭受 smallrtos 管理的中断，然后将变量 Os_Enter_Sum 加一。关中断是为了保护临界区代码。

OS_EXIT_CRITICAL()

#define OS_EXIT_CRITICAL()　　if(--Os_Enter_Sum==0）EA=1

打开受 smallrtos 管理的中断，然后将变量 Os_Enter_Sum 减一。关中断是为了保护临界区代码。