```
uOSBool_t OSTaskSignalClear( OSTaskHandle_t const TaskHandle )
```

函数 OSTaskSignalClear()用于对指定任务的信号进行清除，参数 TaskHandle 为等待接收信号的任务句柄，禁止传入空指针；函数返回值代表信号清除状态，若为 OS_FALSE 则未成功清除信号，若为 OS_TRUE 则成功清除对应任务中的信号数值，信号数值进行复位。

六、轻量级同步消息的 API 函数

```
uOSBool_t OSTaskSignalWaitMsg( sOSBase_t xSigValue, uOSTick_t const uxTicksToWait)
```

函数 OSTaskSignalWaitMsg()为轻量级同步消息等待函数，用于等待系统中别的任务函数或者中断响应函数发出的同步消息。任务函数在调用该接口函数后，即转入挂起等待状态。参数 xSigValue 为输出参数，在成功接收到消息后，该参数会获取具体的消息数值信息。参数 uxTicksToWait 为等待时间，单位为 Tick，若设置为 OSPEND_FORVER_VALUE，则会永远等待，直至接收到同步消息。函数返回值代表消息接收状态，若为 OS_FALSE 则未接收到有效消息，若为 OS_TRUE 则接收到有效消息；注意，该函数不允许在中断响应函数中使用。

```
uOSBool_t OSTaskSignalEmitMsg( OSTaskHandle_t const TaskHandle, sOSBase_t const xSigValue, uOSBool_t bOverWrite )
```

函数 OSTaskSignalEmitMsg()为轻量级消息发射函数，用于向指定的目标任务发送同步消息。参数 TaskHandle 为等待接收消息的任务句柄，即目标任务句柄，禁止向该参数传入空指针；参数 xSigValue 为待发送出的消息数值；参数 bOverWrite 表示既有的消息数值是否在目标任务未响应的情况下可以覆盖重写，若为 OS_TRUE，则代表可以覆盖重新赋值，若为 OS_FALSE，则该信号数值不允许覆盖，该函数直接返回 OS_FALSE，代表向目标任务发送消息失败。函数返回值代表消息发送状态，若为 OS_FALSE 则未发送出有效消息，若为 OS_TRUE 则发送出有效消息；注意，该函数不允许在中断响应函数中使用。

```
uOSBool_t OSTaskSignalEmitMsgFromISR( OSTaskHandle_t const TaskHandle, sOSBase_t const xSigValue, uOSBool_t bOverWrite )
```

函数 OSTaskSignalEmitMsgFromISR()为轻量级消息发射函数，用于向指定的目标任务发送同步消息。参数 TaskHandle 为等待接收消息的任务句柄，即目标任务句柄，禁止向该参数传入空指针；参数 xSigValue 为待发送出的消息数值；参数 bOverWrite 表示既有的消息数值是否在目标任务未响应的情况下可以覆盖重写，若为 OS_TRUE，则代表可以覆盖重新赋值，若为 OS_FALSE，则该信号数值不允许覆盖，该函数直接返回 OS_FALSE，代表向目标任务发送消息失败。函数返回值代表消息发送状态，若为 OS_FALSE 则未发送出有效消息，若为 OS_TRUE 则发送出有效消息；注意，该函数只允许在中断响应函数中使用。

```
uOSBool_t OSTaskSignalClear( OSTaskHandle_t const TaskHandle )
```

函数 OSTaskSignalClear()用于对指定任务的消息进行清除，参数 TaskHandle 为等待接收消息的任务句柄，禁止传入空指针；函数返回值代表消息清除状态，若为 OS_FALSE 则未成功清除消息，若为 OS_TRUE 则成功清除目标任务中的消息数值，消息数值进行复位。

简单示例程序

下面是采用蹄牛操作系统 TINIUX 实现的多任务处理演示示例，主要包括任务创建、信号量创建、消息队列创建、任务延时等功能。代码如下：

```
//任务句柄
OSTaskHandle_t      TaskCtrlHandle = OS_NULL;
OSTaskHandle_t      TaskTest1Handle = OS_NULL;
OSTaskHandle_t      TaskTest2Handle = OS_NULL;
//信号量句柄
OSSemHandle_t       SemHandle = OS_NULL;
//消息队列句柄
OSMsgQHandle_t      MsgQHandle = OS_NULL;
//任务函数
static void TaskCtrl( void *pvParameters );
static void TaskTest1( void *pvParameters );
static void TaskTest2( void *pvParameters );

int main( void )
{
    //Initialize the parameter in TINIUX
    OSInit();

    // 创建消息队列，消息容量5个，消息长度为int类长度.
    MsgQHandle = OSMsgQCreate( 5, sizeof( uint32_t ) );
    // 创建信号量
    SemHandle = OSSemCreate(0);
    // 创建任务
    TaskCtrlHandle = OSTaskCreate(TaskCtrl, OS_NULL, OSMINIMAL_STACK_SIZE,
OSLOWEAST_PRIORITY+1, "Ctrl");
    TaskTest1Handle = OSTaskCreate(TaskTest1, OS_NULL, OSMINIMAL_STACK_SIZE,
OSLOWEAST_PRIORITY+1, "Test1");
    TaskTest2Handle = OSTaskCreate(TaskTest2, OS_NULL, OSMINIMAL_STACK_SIZE,
OSLOWEAST_PRIORITY+1, "Test2");

    // 启动系统
    OSStart();
    for( ;; );
}

static void TaskCtrl( void *pvParameters )
{
    unsigned int uiValueToSend = 0;
    for( ;; )
    {
        //发送信号量
        OSSemPost(SemHandle);
        //发送消息队列
        OSMsgQSend( MsgQHandle, &uiValueToSend, OSPEND_FOREVER_VALUE );
```

```
        uiValueToSend += 1;
        //延时等待
        OSTaskSleep(200*OSTICKS_PER_MS);
    }
}

static void TaskTest1( void *pvParameters )
{
    unsigned int uiReceivedValue;
    for( ;; )
    {
        //等待接收消息
        OSMsgQReceive( MsgQHandle, &uiReceivedValue, OSPEND_FOREVER_VALUE );
        //do something here
    }
}

static void TaskTest2( void *pvParameters )
{
    for( ;; )
    {
        //等待信号量
        OSSemPend(SemHandle, OSPEND_FOREVER_VALUE);
        //do something here
    }
}
```