# 小嵌操作系统 SmallRTOS 常用接口函数介绍

——SmallRTOS 开源网站提供 http://www.smallrtos.org

“小嵌”操作系统 SmallRTOS 是一个源代码开放的、易于移植的、面向深度嵌入式应用的微内核实时操作系统，主要应用领域为工业控制，智能传感器开发，智能终端，物联网，机器人等。任何人在遵循 SmallRTOS 许可协议的前提下均可免费使用该嵌入式实时操作系统。SmallRTOS 系统最新版本源代码及示例工程的发布网站为：http://www.smallrtos.org

下载到 SmallRTOS 的源代码压缩包后，进行解压，可以看到 SmallRTOS 的目录结构如下：

Kernel： 存放 SmallRTOS 的内核文件 OS 及和 CPU 相关的移植文件；
Demo： 存放 SmallRTOS 提供的示例文件；
Doc： 存放 SmallRTOS 相关说明/教程文档；
License： 存放 SmallRTOS 使用许可；

小嵌操作系统 SmallRTOS 在设计时，其文件命名、函数名及变量命名由专用的前缀进行区分：前缀为 OS，表示为 SmallRTOS 的内核，这些是与平台无关的内核部分，在进行跨平台移植时，无需更改；前缀为 Fit，表示为硬件（芯片类型等）相关的部分，在进行移植时，这一部分的文件、函数及变量需要根据硬件平台（芯片类型等）进行适当的调整；

为了突显嵌入式操作系统配置及裁剪的灵活性，在 SmallRTOS 系统设计之初就进行了全面的考量，系统中相关参数及功能模块采用了宏定义的方式进行配置，常用的配置参数均保存在文件“OSType.h”中。系统中常用的配置如下所示：

**OSTICK_RATE_HZ**：该参数配置了系统运行的“速率”，其决定了系统内核调度的最小时间粒度。默认配置为 1000Hz，最小时间粒度为 1 毫秒。在 Demo 示例工程中，采用 configTICK_RATE_HZ 配置；
**OSTICKS_PER_MS**：该参数用于表示每毫秒时间内系统运行的 Ticks 数量，Ticks 具体含义稍后会介绍，该参数与 OSTICK_RATE_HZ 关联，无需用户专门配置；
**OSNAME_MAX_LEN**：系统中名称的最大长度，包括任务名称，软件定时器名称等，默认配置为 10 字节。在 Demo 示例工程中，采用 configMAX_NAME_LEN 配置；
**OSLOWEAST_PRIORITY**：系统最低优先级数值，配置为 0；
**OSHIGHEAST_PRIORITY**：系统最高优先级数值。原则上讲，系统不会限制最高优先级数值，不过优先级越高，占用的资源越多，推荐最高优先级不超过 31，系统默认最高优先级为 8。在 Demo 示例工程中，采用 configMAX_PRIORITIES 配置；
**OSTOTAL_HEAP_SIZE**：为小嵌操作系统分配的栈空间大小。用户采用系统函数创建的任务、信号量、互斥信号量、消息队列、软件定时器等均使用该栈空间。在 Demo 示例工程中，采用 configTOTAL_HEAP_SIZE 配置。
**OSMINIMAL_STACK_SIZE**：为任务分配的最小栈空间大小，默认配置 32 个字长。任务使用的栈空间量由具体的任务决定，若任务中局部变量较多，使用空间较大，则需配置更大的栈空间。在 Demo 示例工程中，采用 configMINIMAL_STACK_SIZE 配置；
**OSPEND_FOREVER_VALUE**：定义永久挂起的数值，用于信号量、互斥信号量、消息队列等永久等待定义数值，在 32 位宽的芯片中，推荐配置为 0xFFFFFFFF。在 Demo 示例工程中，采用 configPEND_FOREVER_VALUE 配置；

**OS_SEMAPHORE_ON**：是否启用系统信号量的标识。数值为 1 则启用信号量功能，数值为 0，则不启用信号量。在 Demo 示例工程中，采用 configUSE_SEMAPHORE 配置；
**OS_MSGQ_ON**：是否启用消息队列的标识。数值为 1 则启用消息队列，数值为 0，则不启用消息队列。在 Demo 示例工程中，采用 configUSE_MSGQ 配置；
**OSMSGQ_MAX_MSGNUM**：消息队列中保存的消息数量，默认配置为 5。在向该消息队列发送消息时，若消息数达到该数值时，则消息队列已满，需挂起等待，或者把旧数据冲掉。在 Demo 示例工程中，采用 configMSGQ_MAX_MSGNUM 配置；
**OS_MUTEX_ON**：是否启用互斥信号量的标识。数值为 1 则启用互斥信号量，数值为 0，则不启用互斥信号量。在 Demo 示例工程中，采用 configUSE_MUTEX 配置；
**OS_TIMER_ON**：是否启用软件定时器的标识。数值为 1 则启用软件定时器，数值为 0，则不启用软件定时器。在 Demo 示例工程中，采用 configUSE_TIMER 配置；
**OSCALLBACK_TASK_PRIO**：用于配置软件定时器响应任务的优先级，推荐采用 OSHIGHEAST_PRIORITY-1。在 Demo 示例工程中，采用 configCALLBACK_TASK_PRIORITY 配置；

小嵌操作系统 SmallRTOS 是多任务抢占式操作系统，高优先级任务可以抢占执行，体现了操作系统的实时性。在 SmallRTOS 系统中，优先级 0 为最低优先级，该优先级为空闲任务 OSIdleTask 的专用优先级，用户创建的任务无法使用。除此之外，用户可以根据任务的重要程度自行配置。

操作系统运行频率（速率）常用 Ticks 表示，Ticks 亦被称为时钟滴答。操作系统“滴答”由硬件以规律性的定时中断产生。在小嵌操作系统 SmallRTOS 中，时钟“滴答”决定了系统的最小时间粒度，这个参数可以根据硬件平台进行配置。在 SmallRTOS 系统中，该参数采用宏定义 configTICK_RATE_HZ 进行配置。在 SmallRTOS 提供的大部分示例工程中，均配置的为 1000Hz，即每个时钟滴答间隔是 1 毫秒。该参数会影响 SmallRTOS 系统对任务的控制精度。

操作系统启动后，会按照任务的优先级选择性的执行，最先执行处于等待状态的最高优先级任务，直至该任务让出执行权或者被其它更高优先级的任务抢断。在系统运行的过程中，如果没有符合条件的任务需要执行，则运行系统中预留的 OSIdleTask（空闲任务）。

下面是小嵌操作系统 SmallRTOS 中经常用到的接口函数，供大家使用时参考；

## 一、系统中任务相关的 API 函数

```
OSTaskHandle_t OSTaskCreate(OSTaskFunction_t  pxTaskFunction,
                            void*             pvParameter,
                            const uOS16_t     usStackDepth,
                            uOSBase_t         uxPriority,
                            sOS8_t*           pcTaskName)
```

OSTaskCreate 为小嵌操作系统 SmallRTOS 的任务创建函数，其中参数 OSTaskFunction_t pxTaskFunction 为任务函数，该任务函数定义为 void TaskFunction( void *pParameters )；任务函数中的参数 void *pParameters 亦即 OSTaskCreate 的第二个参数；第三个参数为任务的栈空间 usStackDepth，栈空间需要根据任务占用的空间多少进行调整；第四个参数为任务的优先级，除最低优先级 0 为系统保留外，均可使用。第五个参数为任务名字，任务名字也就是任务的标签，主要方便在调试时区分不同的任务。

函数 OSTaskCreate 的返回值为 OSTaskHandle_t 类型的任务句柄，该句柄可以被其它系统函数调用，以设置或控制任务的状态；

**void OSTaskSleep( const uOSTick_t uxTicksToDelay )**

OSTaskSleep 为小嵌操作系统 SmallRTOS 系统中任务延迟执行设置函数，通过此函数，可以把当前任务休眠若干毫秒的时间。参数 uOSTick_t uxTicksToDelay 代表休眠的时间长短，单位为 Ticks，用户可以通过 OSTICKS_PER_MS 把毫秒转换为 Ticks 计数；

**OSTaskYield()**

OSTaskYield 函数为 SmallRTOS 中的任务控制类函数，在任务函数中调用，用于让出当前任务的执行权，并切换到下一个处于等待执行状态的任务；函数 OSTaskYield 会把当前任务移到等待执行队列的末尾，若只有当前任务处于等待执行状态，则仍然执行该任务；

**void OSTaskSuspend( OSTaskHandle_t TaskHandle )**

OSTaskSuspend 函数为系统任务挂起函数，任务挂起后，将不再参与系统的调度，但任务仍然保留在系统中，任务占用的资源不会释放。参数 TaskHandle 用于表示待挂起的任务句柄，该数值若为 NULL，则挂起当前正在运行的任务；

**void OSTaskResume( OSTaskHandle_t TaskHandle )**

OSTaskResume 函数用于把挂起的任务恢复（激活），使该任务重新参与系统的调度，参数 TaskHandle 表示待恢复的任务句柄，该参数不允许为 NULL。此函数不是中断安全函数，不允许在中断函数中调用该函数；

**sOSBase_t OSTaskResumeFromISR( OSTaskHandle_t TaskHandle )**

OSTaskResumeFromISR 函数用于把挂起的任务恢复（激活），使该任务重新参与系统的调度，参数 TaskHandle 表示待恢复的任务句柄，该参数不允许为 NULL；此函数为中断安全函数，只能在中断服务函数中调用；

**OSStart()**

函数 OSStart 为 OSStartScheduler()函数的宏定义，OSStartScheduler()函数是 SmallRTOS 中的任务调度启动函数；在该函数中，系统会设置空闲任务 OSIdleTask 及时钟中断（时钟滴答）；OSIdleTask 任务为系统空闲任务，若系统中无其它需要执行的任务，则会调用该空闲任务，空闲任务可以用于统计当前系统的利用率，及释放处于待删除状态任务的资源；时钟中断则为系统的 ticks 配置，整个系统的运行即依赖此 ticks 驱动运行；

## 二、任务同步信号量相关的 API 函数

**OSSemHandle_t OSSemCreate( void )**

函数 OSSemCreate 为信号量（Semaphore）创建函数，用于创建任务间及中断与任务间同步的信号量。信号量创建后，默认有效信号量计数值为 0，表示该信号量无有效信号，对应的 OSSemPend 函数处于阻塞状态，等待有效信号；函数返回值为 OSSemHandle_t 类型的句柄，方便用于对该信号量的操控；

**sOSBase_t OSSemPend( OSSemHandle_t SemHandle, uOSTick_t xTicksToWait)**

函数 OSSemPend 为信号量等待函数，在任务执行函数中调用，用于等待相关同步的信号量；参数 OSSemHandle_t SemHandle 为信号量句柄，参数 uOSTick_t xTicksToWait 为任务阻塞时间，单位为 Tick 数，若设置为 OSPEND_FORVER_VALUE，则会永远阻塞，直至指定信号量 SemHandle 获取到有效信号；

**sOSBase_t OSSemPost( OSSemHandle_t SemHandle)**

函数 OSSemPost 用于向指定信号量发送有效信号，使处于等待该信号量的任务获取同步信号，以便恢复执行。注：此函数不能在中断服务函数中调用。

```
sOSBase_t OSSemPostFromISR( OSSemHandle_t SemHandle)
```

函数 OSSemPostFromISR 用于向指定信号量发送有效信号，使处于等待该信号量的任务获取同步信号，以便恢复执行。注：此函数只能在中断服务函数中调用。

### 三、任务同步消息相关的 API 函数

```
OSMsgQHandle_t OSMsgQCreate( const uOSBase_t uxQueueLength, const uOSBase_t uxItemSize)
```

函数 OSMsgQCreate 为消息队列的创建函数，用于创建任务间同步操作的消息队列，参数 uxQueueLength 为消息队列中的消息数的容量（消息数目超过此容量，则发送任务挂起，直到消息队列有空闲位置），第二个参数 uxItemSize 为单个消息的长度。其返回值为 OSMsgQHandle_t 类型的消息队列句柄，方便对消息队列的操控；

```
sOSBase_t OSMsgQReceive( OSMsgQHandle_t MsgQHandle, void * const pvBuffer, uOSTick_t xTicksToWait)
```

函数 OSMsgQReceive 用于在任务中接收指定消息队列的消息，在任务函数中调用。该函数为任务阻塞函数。参数 MsgQHandle 为消息队列句柄，参数 pvBuffer 表示消息的指针，参数 xTicksToWait 为消息队列等待接收时间，单位为 Tick，若设置为 OSPEND_FORVER_VALUE，则会永远等待，直至指定消息队列 MsgQHandle 获取到有效消息为止。函数返回值代表消息接收状态，若为 FALSE 则未接收到有效消息，若为 TRUE 则接收到有效消息；

```
sOSBase_t OSMsgQReceiveFromISR( OSMsgQHandle_t MsgQHandle, void * const pvBuffer)
```

函数 OSMsgQReceiveFromISR 用于在中断函数中接收指定消息队列的消息；参数 MsgQHandle 为消息队列句柄，参数 pvBuffer 表示消息的指针，该函数不会阻塞。函数返回值代表消息接收状态，若为 FALSE 则未接收到有效消息，若为 TRUE 则接收到有效消息；注：此函数只能在中断服务函数中调用。

```
sOSBase_t OSMsgQSend( OSMsgQHandle_t MsgQHandle, const void * const pvItemToQueue, uOSTick_t xTicksToWait)
```

函数 OSMsgQSend 用于向指定的消息队列发送消息，使处于等待该消息的任务获取同步消息，并恢复执行，其中参数 MsgQHandle 表示消息队列，参数 pvItemToQueue 表示消息地址（指针），参数 xTicksToWait 为消息发送等待接收时间，单位为 Tick，若设置为 OSPEND_FORVER_VALUE，则会永远等待，直至指定消息队列 MsgQHandle 有空闲位置。函数返回值代表消息发送状态，若为 FALSE 则消息发送失败，若为 TRUE 则消息发送成功；注：此函数不能在中断服务函数中调用。

```
sOSBase_t OSMsgQSendFromISR( OSMsgQHandle_t MsgQHandle, const void * const pvItemToQueue)
```

函数 OSMsgQSendFromISR 用于向指定的消息队列发送消息，使处于等待该消息的任务获取同步消息，并恢复执行，其中参数 MsgQHandle 表示消息队列，参数 pvItemToQueue 表示消息地址（指针），如果消息队列已满，则函数不会阻塞，直接返回发送失败信息。注：此函数只能在中断服务函数中调用。

### 四、定时器相关的 API 函数

```
OSTimerHandle_t OSTimerCreate(OSTimerFunction_t Function, void* pvParameter,
uOSBase_t uxPeriodicTimeMS, sOS8_t* pcName)
```

接口函数 OSTimerCreate 用于创建定时器。其中参数 Function 为定时器的服务函数，用于响应定时器，函数 OSTimerFunction_t 的定义类型为 void TimerFunction(void *pParameters )。pvParameter 为定时器服务函数的参数，不用时可以设置为 NULL。参数 uxPeriodicTimeMS 为定时器的周期，单位为毫秒。参数 pcName 为定时器的名称，方便区分不同的定时器；注意：定时器服务函数中禁止添加信号量等待、消息队列等待等阻塞函数，为不影响系统的性能，定时器服务函数耗时越少越好。

```
uOSBase_t OSTimerStart(OSTimerHandle_t const TimerHandle)
```

定时器创建完毕后并不会自动启动，需要用户调用启动函数 OSTimerStart()，之后定时器才会生效。参数 TimerHandle 为定时器句柄，为定时器创建函数 OSTimerCreate()的返回值；

```
uOSBase_t OSTimerStop(OSTimerHandle_t const TimerHandle)
```

定时启动后，用户可以通过接口函数 OSTimerStop()停止定时。参数 TimerHandle 为定时器句柄，为定时器创建函数 OSTimerCreate()的返回值；

**简单示例程序**

下面是采用实时嵌入式操作系统 SmallRTOS 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、任务延时等功能。代码如下：

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

int main( void )
{
    // 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();
    // 创建任务
    TaskCtrlHandle = OSTaskCreate(TaskCtrl, NULL, OSMINIMAL_STACK_SIZE,
OSLOWEAST_PRIORITY+1, "Ctrl");
    TaskTest1Handle = OSTaskCreate(TaskTest1, NULL, OSMINIMAL_STACK_SIZE,
```

```
OSLOWEAST_PRIORITY+1, "Test1");
    TaskTest2Handle = OSTaskCreate(TaskTest2, 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
    }
}
```