# 目录

[目录 1](#_Toc525233346)

[第19章 Cortex-M4-UCOS-II 2](#_Toc525233347)

[19.1 操作系统介绍 2](#_Toc525233348)

[19.1.1 操作系统简介 2](#_Toc525233349)

[19.1.2 常见的操作系统 2](#_Toc525233350)

[19.2 UCOS操作系统介绍 2](#_Toc525233351)

[19.2.1 操作系统的调度原则 2](#_Toc525233352)

[19.2.2 操作系统的程序结构 2](#_Toc525233353)

[19.2.3 操作系统的任务调度 3](#_Toc525233354)

[19.2.4 操作系统的任务状态 3](#_Toc525233355)

[19.2.5 操作系统的任务中断 3](#_Toc525233356)

[19.3 UCOS操作系统项目工程 4](#_Toc525233357)

[19.3.1 UCOS操作系统源码获取 4](#_Toc525233358)

[19.3.2 UCOS操作系统文件介绍 4](#_Toc525233359)

[19.3.3 UCOS操作系统建立工程 4](#_Toc525233360)

[19.3.4 UCOS操作系统建立截图 5](#_Toc525233361)

[19.3.5 UCOS操作系统必要代码 6](#_Toc525233362)

[19.3.6 UCOS操作系统创建任务 8](#_Toc525233363)

[19.4 UCOS操作系统任务管理 8](#_Toc525233364)

[19.4.1 ucos任务组成 8](#_Toc525233365)

[19.4.2 任务创建 9](#_Toc525233366)

[19.4.3 任务删除 9](#_Toc525233367)

[19.4.4 任务挂起 10](#_Toc525233368)

[19.4.5 解除挂起 10](#_Toc525233369)

[19.4.6 改变任务优先级 10](#_Toc525233370)

[19.4.7 任务栈检查 10](#_Toc525233371)

[19.5 UCOS操作系统任务间通信 11](#_Toc525233372)

[19.5.1 信号量 11](#_Toc525233373)

[19.5.2 消息邮箱 13](#_Toc525233374)

[19.5.3 消息队列 16](#_Toc525233375)

[19.5.4 互斥信号量 20](#_Toc525233376)

[19.6 UCOS操作系统事件标志组 22](#_Toc525233377)

[19.7 UCOS操作系统软件定时器 23](#_Toc525233378)

[19.7.1 延时函数 23](#_Toc525233379)

[19.7.2 软件定时器 23](#_Toc525233380)

[19.8 UCOS操作系统其它功能 23](#_Toc525233381)

# Cortex-M4-UCOS-II

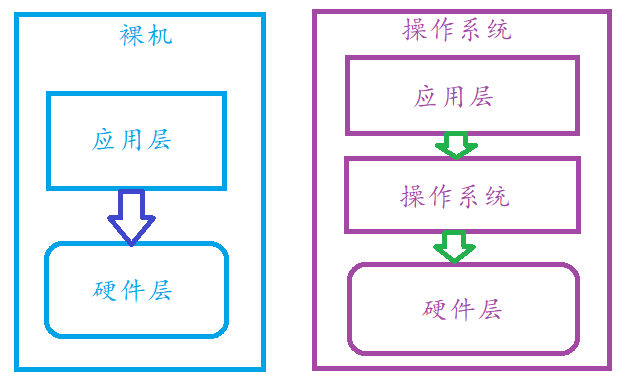
## 操作系统介绍

### 操作系统简介

1. 什么是操作系统。

操作系统是管理和控制计算机硬件与软件资源的计算机程序，是直接运行在“裸机”上的最基本的系统软件，任何其他软件都必须在操作系统的支持下才能运行。介于APP和硬件之间。

1. 操作系统在程序中的结构（对比）



### 常见的操作系统

常见的操作系统：

IOS（苹果）、安卓（）、Windows、Linux、塞班、UCOS、FreeRTOS、RT-T(国产)、Vxworks、eCos等等。

1. 实时操作系统(RTOS)

UCOS（学习免费，商用需要注册）、FreeRTOS、RT-T(国产)、Vxworks、eCos

会给每个任务一个基础的时间长度，根据任务的需求适当调整任务时长。任务与任务直接切换都有相应的规则。

1. 分时操作系统

Windows----2000以前、Linux----2.6以前

给每一个任务分配一个固定的时长，如果这个任务在固定的时长内已经空闲，分时操作系统还是会给该任务规定好的时长。

1. 半实时操作系统

Windows----7/8/10、Linux----2.6以后

## UCOS操作系统介绍

### 操作系统的调度原则

UCOS操作系统调度原则是：以任务优先级为调度原则。任务优先级编号越小的，任务优先级越高。

注意点：每个任务只有一个优先级，不同任务优先级编号不能一致。

### 操作系统的程序结构

1. 没有操作系统时

整个裸机项目工程中有且仅有一个死循环，这个死循环存在main函数中。其它地方可以有死循环但是必须要有退出条件。

1. 有操作系统时

整个操作系统项目工程中可以有多个死循环，每个任务中有一个死循环，main函数中无死循环。

注意点：操作系统在main中必须完成两点：①设定好操作系统心跳节拍；②创建一个任务(该任务必须创建成功)。

### 操作系统的任务调度

任务调度：指的是什么时候会发生任务调度。

两种情况下会发生任务调度。

1. 操作系统的心跳节拍到达。心跳节拍称为一个Tick。UCOS中有一个专用的操作系统心跳节拍函数：OSTimeTick();
2. 操作系统中的程序代码调用OS\_Sched();该函数时会产生任务调度。OS\_Sched()：称为任务调度函数。

注意：任务调度函数我们不能够直接调用，只能是操作系统调用。程序员调用该函数会使得UCOS操作系统崩溃。

注意点①：操作系统发生任务调度不一定切换任务；任务切换一定需要产生任务调度。

注意点②：操作系统中高优先级任务必须要有释放CPU控制权函数调用。

### 操作系统的任务状态

UCOS操作系统中，任务状态有五种：停止/休眠、准备就绪、运行、挂起/等待、中断。只有当任务进入到运行状态才可以执行任务中的程序代码。

停止：任务被删除。

休眠：当前没有任何一个任务准备就绪，运行状态中没有任务执行。

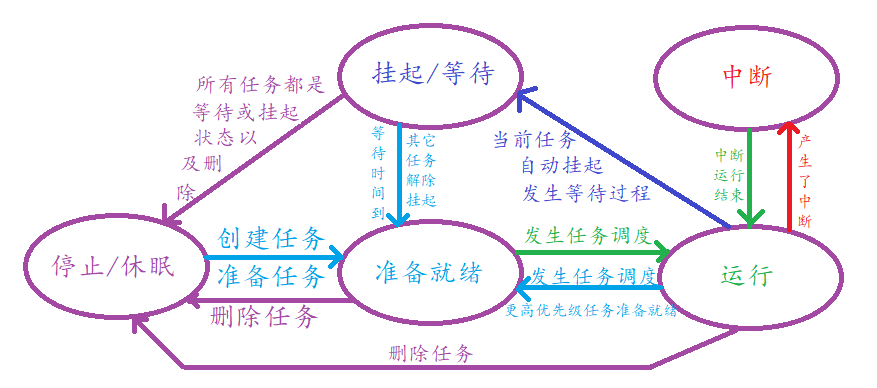
准备就绪：任务运行需要满足的条件都准备完成。

运行：任务正在运行。

挂起：任务需要等待某一个条件发生，其他任务可以将这个任务挂起。

等待：延时相关的。

中断：程序发生了中断事件，CPU要去处理中断服务函数，中断操作系统。



### 操作系统的任务中断

中断状态中需要注意以下几点：

1. 从运行状态进入中断状态必须要调用函数OSIntEnter()；该函数的功能是用来告诉操作系统当前程序进入中断。
2. 从中断状态退出后进入运行状态之前必须调用函数OSIntExit()；该函数的功能是用来告诉操作系统当前程序退出中断。
3. 程序从中断中退出，操作系统返回到任务中去，这时不一定返回到原来的任务中。

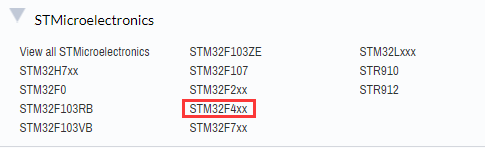
## UCOS操作系统项目工程

### UCOS操作系统源码获取

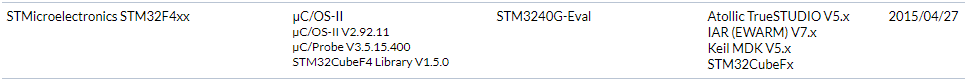
1. 找到官网。

官网链接：<https://www.micrium.com/>

进入下载界面：<https://www.micrium.com/downloadcenter/>



1. 从官网下载对应的操作系统文件。



使用的是UCOS-II；使用的工具为Keil MDK 5.x平台。

下载下来的压缩包：Micrium\_STM3240G-EVAL\_OS2.zip

注意：必须登录才能下载，下载过程是免费的。

### UCOS操作系统文件介绍



查看示例工程：

1. 初始化好系统滴答定时器，在系统滴答中断中配置好心跳节拍。
2. 初始化操作系统
3. 创建一个任务
4. 启动操作系统
5. 如果启动失败进入while(1)

### UCOS操作系统建立工程

1. 新建工程文件夹：Ucos\_Project\_Demo
2. 在Ucos\_Project\_Demo文件夹下建立Cmsis文件夹、User文件夹、Ucos文件夹、Ucos\_Task文件夹、Librares文件夹(使用标准库)
3. 拷贝启动文件到Cmsis文件夹下
4. 拷贝Ucos中的源码到Ucos文件夹下
5. 在User中建立inc和src文件夹
6. 拷贝标准库文件到Librares文件夹下
7. 打开软件、新建工程、保存工程、选择芯片型号
8. 配置虚拟文件夹并且添加真实文件(配置三色品字形图标)
9. 配置工程(配置魔术棒、宏定义、头文件路径)
10. 编写最简单的操作系统代码
11. 编译并且下载

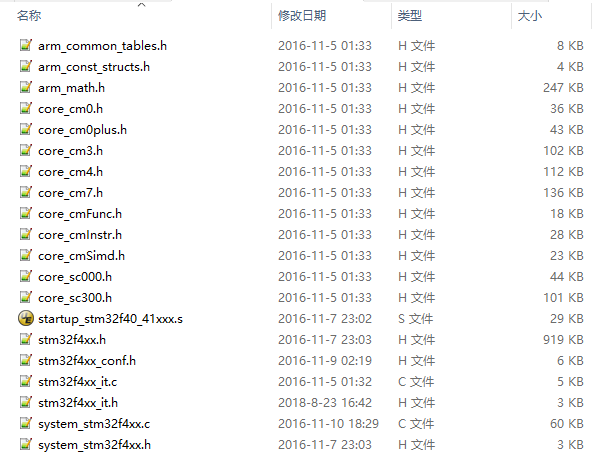
### UCOS操作系统建立截图

1. 创建文件夹



1. 拷贝相应的文件

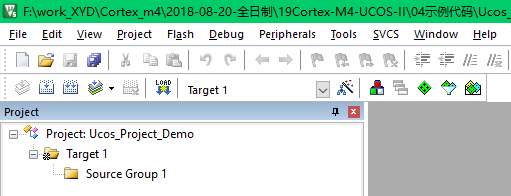
cmsis文件：



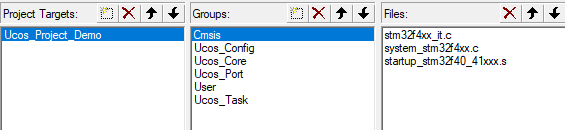
Ucos文件：

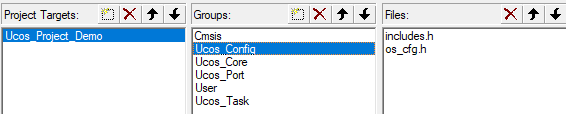


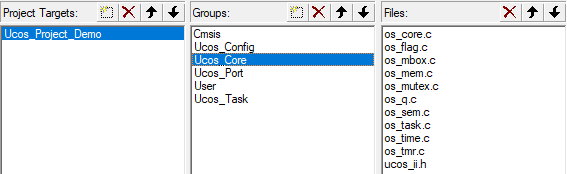
1. 建立工程

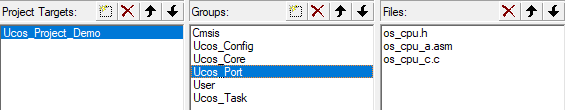


1. 配置虚拟文件夹

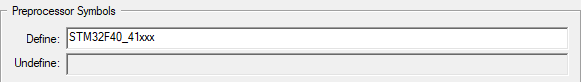


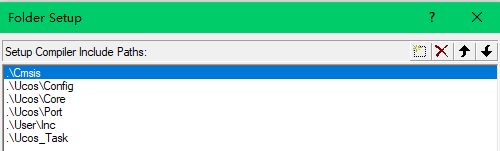






1. 配置工程





### UCOS操作系统必要代码

|  |
| --- |
| /\*  函数功能：初始化系统滴答定时器  函数形参：任意毫秒  函数返回值：无  备注：控制操作系统的心跳节拍----中断方式  作者：李工  时间：2018-09-19  \*/  void SysTick\_Init(u16 n\_ms)  {  SysTick->CTRL = 0;//清空配置  //选择时钟源为21MHz    SysTick->VAL = 0;    SysTick->LOAD = n\_ms \* 21000;    SysTick->CTRL |= (0X1 << 1);//开启模块级中断使能    SysTick->CTRL |= (0X1 << 0);//开启递减计数器--使能  }  /\*  函数功能：系统滴答定时器的中断服务函数  函数形参：无  函数返回值：无  备注：告诉操作系统进中断、出中断、心跳节拍函数  作者：李工  时间：2018-09-19  \*/  void SysTick\_Handler(void)  {  OSIntEnter();//进中断    if(SysTick->CTRL & (0X1 << 16))  {  OSTimeTick();//心跳节拍函数  }    OSIntExit();//出中断  }  /\*  函数功能：主函数  函数形参：无  函数返回值：整型  备注：使用UCOS操作系统  作者：李工  时间：2018-09-19  \*/  int main(void)  {  //初始化系统滴答定时器  SysTick\_Init(1000 / OS\_TICKS\_PER\_SEC);  //初始化操作系统  OSInit();  //创建一个任务  OSTaskCreate(os\_start\_task,NULL,&OS\_STK\_START\_TASK\_SIZE[OS\_STK\_START\_TASK\_LENGTH - 1],OS\_START\_TASK\_PRIO);  //启动操作系统  OSStart();  //如果任务创建失败则执行这个无限循环  while(1)  {  ;  }  } |

### UCOS操作系统创建任务

|  |
| --- |
| INT8U OSTaskCreate (void (\*task)(void \*p\_arg),  void \*p\_arg,  OS\_STK \*ptos,  INT8U prio)  函数功能：创建一个任务  函数形参：void (\*task)(void \*p\_arg)：函数指针，用来存放任务函数  void \*p\_arg：任务函数的形参传递，如果无形参传递使用NULL即可  OS\_STK \*ptos：任务的任务栈的栈顶  INT8U prio：任务的优先级编号  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  使用方法：  OSTaskCreate(os\_start\_task,NULL,&OS\_STK\_START\_TASK\_SIZE[OS\_STK\_START\_TASK\_LENGTH - 1],OS\_START\_TASK\_PRIO); |

任务的具体实现：

|  |
| --- |
| #define OS\_START\_TASK\_PRIO 10  #define OS\_STK\_START\_TASK\_LENGTH 256  OS\_STK OS\_STK\_START\_TASK\_SIZE[OS\_STK\_START\_TASK\_LENGTH];  /\*  函数功能：启动任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_start\_task(void \*p\_arg)  {    while(1)  {    }  } |

## UCOS操作系统任务管理

### ucos任务组成

任务控制块（底层，不需要掌握）、任务优先级，任务栈，任务函数。

任务控制块：当创建一个任务的时候，ucos会分配一段内存空间给这个任务，这段内存空间就称为任务控制块。任务控制块记录了任务的相关信息：任务函数地址、任务的优先级、任务状态、任务栈地址。

任务优先级：每个任务都有一个唯一的优先级，优先级也只我们这个任务在ucos中的唯一标识，任务的优先级是ucos做任务调度的依据。

任务栈：当任务切换的时候保存该任务的状态（切换之前执行到哪里了，局部变量）。

任务函数：当任务被调度的时候，要完成的工作。

### 任务创建

|  |
| --- |
| INT8U OSTaskCreate (void (\*task)(void \*p\_arg),  void \*p\_arg,  OS\_STK \*ptos,  INT8U prio)  函数功能：创建一个任务，这种方式创建任务不可以检查任务栈的使用情况。  函数形参：void (\*task)(void \*p\_arg)：函数指针，用来存放任务函数  void \*p\_arg：任务函数的形参传递，如果无形参传递使用NULL即可  OS\_STK \*ptos：任务的任务栈的栈顶  INT8U prio：任务的优先级编号  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  INT8U OSTaskCreateExt (void (\*task)(void \*p\_arg),  void \*p\_arg,  OS\_STK \*ptos,  INT8U prio,  INT16U id,  OS\_STK \*pbos,  INT32U stk\_size,  void \*pext,  INT16U opt)  函数功能：创建一个任务，这种方式创建任务可以检查任务栈的使用情况。  函数形参：void (\*task)(void \*p\_arg)：函数指针，用来存放任务函数  void \*p\_arg：任务函数的形参传递，如果无形参传递使用NULL即可  OS\_STK \*ptos：任务的任务栈的栈顶  INT8U prio：任务的优先级编号  INT16U id：任务的ID号，无实际意义，任务与任务之间可以重复ID号  OS\_STK \*pbos：任务栈的栈底  INT32U stk\_size：任务栈的大小  void \*pext：是指向用户提供的内存位置的指针，该位置用作TCB扩展。可以使用NULL。  INT16U opt：OS\_TASK\_OPT\_STK\_CHK使用最多的一个宏定义，目的检查任务栈使用。  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败    注意点：  1. 不允许创建已经存在的优先级任务  2. 不允许创建任务的优先级值比最低优先级任务还大  3. 不允许在中断中创建任务 |

### 任务删除

|  |
| --- |
| INT8U OSTaskDel (INT8U prio)  函数功能：删除一个任务  函数形参：INT8U prio：任务的优先级编号，该优先级形参还可以填写：OS\_PRIO\_SELF；当前任务的优先级  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  INT8U OSTaskDelReq (INT8U prio)  函数功能：请求删除一个任务  函数形参：INT8U prio：任务的优先级编号，该优先级形参还可以填写：OS\_PRIO\_SELF；当前任务的优先级  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  示例：任务A要删除任务B；  任务A中编写：OSTaskDelReq(任务B的优先级);  任务B中编写：if (OSTaskDelReq(OS\_PRIO\_SELF) == OS\_ERR\_TASK\_DEL\_REQ)  {OSTaskDel(OS\_PRIO\_SELF);}在while(1)调用  注意点：想要删除别的任务必须调用请求删除任务；想要删除自身任务，直接调用删除任务即可。 |

### 任务挂起

|  |
| --- |
| INT8U OSTaskSuspend (INT8U prio)  函数功能：挂起一个任务  函数形参：INT8U prio：任务的优先级编号，该优先级形参还可以填写：OS\_PRIO\_SELF；当前任务的优先级  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败 |

### 解除挂起

|  |
| --- |
| INT8U OSTaskResume (INT8U prio)  函数功能：解除挂起任务  函数形参：INT8U prio：任务的优先级编号  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败 |

### 改变任务优先级

|  |
| --- |
| INT8U OSTaskChangePrio (INT8U oldprio,  INT8U newprio)  函数功能：改变一个任务的优先级  函数形参：INT8U oldprio：旧的优先级  INT8U newprio：新的优先级  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  注意：程序中不要去使用这个函数，因为容易将优先级改的混乱 |

### 任务栈检查

|  |
| --- |
| INT8U OSTaskStkChk (INT8U prio,  OS\_STK\_DATA \*p\_stk\_data)  函数功能：检查任务的任务栈使用率  函数形参：INT8U prio：任务的优先级编号，该优先级形参还可以填写：OS\_PRIO\_SELF；当前任务的优先级  OS\_STK\_DATA \*p\_stk\_data：保存任务栈使用率的指针  typedef struct os\_stk\_data {  INT32U OSFree; /\* Number of free bytes on the stack \*/  INT32U OSUsed; /\* Number of bytes used on the stack \*/  } OS\_STK\_DATA;  示例：OS\_STK\_DATA p\_stk\_data;  printf("%d%%",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  函数返回值：OS\_ERR\_NONE：表示创建成功；否则表示失败  注意点：检查任务栈函数的使用必须要以拓展方式创建任务；  INT8U OSTaskCreateExt();  INT16U opt：OS\_TASK\_OPT\_STK\_CHK使用最多的一个宏定义，目的检查任务栈使用。 |

## UCOS操作系统任务间通信

### 信号量

（保护那段代码不被别的进程抢走）

作用：保护共享资源。

信号量：在UCOS操作系统里面就相当于一个全局变量，释放信号量让这个信号量的值加1，使用信号量让这个信号量的值减1。最小值为0。

案例：打印店，一台打印机，多台PC。打印机就属于共享资源。PC释放信号量，打印机使用信号量。

|  |
| --- |
| INT16U OSSemAccept (OS\_EVENT \*pevent)  函数功能：使用一个信号量；获取信号量  函数形参：OS\_EVENT \*pevent：信号量指针  函数返回值：返回信号量值的大小  注意：该函数在中断中使用，不需要等待就去获取  OS\_EVENT \*OSSemCreate (INT16U cnt)  函数功能：创建一个信号量  函数形参：INT16U cnt：初始的时候给信号量一个值  函数返回值：信号量事件控制块指针  事件控制块：是一个结构体  OS\_EVENT \*OSSemDel (OS\_EVENT \*pevent,  INT8U opt,  INT8U \*perr)  函数功能：删除一个信号量  函数形参：OS\_EVENT \*pevent：信号量（事件控制块）指针  INT8U opt：删除信号量的方式；OS\_DEL\_NO\_PEND或者OS\_DEL\_ALWAYS  INT8U \*perr：存储错误类型；如果为OS\_ERR\_NONE表示成功，否则表示失败  函数返回值：信号量事件控制块指针  void OSSemPend (OS\_EVENT \*pevent,  INT32U timeout,  INT8U \*perr)  函数功能：等待一个信号量  函数形参：OS\_EVENT \*pevent：信号量指针  INT32U timeout：等待的时间，时间跟tick相关；？当timeout = 0时候，表示无限等待  INT8U \*perr：存储错误类型；如果为OS\_ERR\_NONE表示成功，否则表示失败  INT8U OSSemPendAbort (OS\_EVENT \*pevent,  INT8U opt,  INT8U \*perr)  函数功能：中止正在等待信号量的任务  函数形参：OS\_EVENT \*pevent：信号量指针  INT8U opt：中止等待的类型；OS\_PEND\_OPT\_NONE或者OS\_PEND\_OPT\_BROADCAST  INT8U \*perr：存储错误类型；如果为OS\_ERR\_NONE表示成功，否则表示失败  函数返回值：信号量值的大小  INT8U OSSemPost (OS\_EVENT \*pevent)  函数功能：发送一个信号量  函数形参：OS\_EVENT \*pevent：信号量指针  函数返回值：存储错误类型；如果为OS\_ERR\_NONE表示成功，否则表示失败  void OSSemSet (OS\_EVENT \*pevent,  INT16U cnt,  INT8U \*perr)  函数功能：设置信号量的值  函数形参：OS\_EVENT \*pevent：信号量指针  INT16U cnt：信号量的值  INT8U \*perr：存储错误类型；如果为OS\_ERR\_NONE表示成功，否则表示失败  注意点：该函数的使用必须建立在没有任务在等待信号量时  使用该函数之前必须使用OSSemPendAbort(xxxx,OS\_PEND\_OPT\_BROADCAST,xxxx); |

信号量使用：

|  |
| --- |
| OS\_EVENT\_KEY\_SEM = \*OSSemCreate (0);  /\*  函数功能：KEY任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_key\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;  u8 error = 0;    p\_arg = p\_arg;    while(1)  {  if(key\_scan() != KEY\_ERR)  {  error = OSSemPost(&OS\_EVENT\_KEY\_SEM);  if(error != OS\_ERR\_NONE)  {  printf("file:%s\tline:%d\r\n",\_\_FILE\_\_,\_\_LINE\_\_);  }  }  OSTimeDly(1);//释放CPU控制权    OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  printf("os\_key\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  }  /\*  函数功能：LED任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_led\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;  u8 error = 0;    p\_arg = p\_arg;    while(1)  {  OSSemPend(&OS\_EVENT\_KEY\_SEM,0,&error);  if(error == OS\_ERR\_NONE)  {  NEGATE\_LED1 = !NEGATE\_LED1;  }    OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  printf("os\_led\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  } |

### 消息邮箱

作用：可以传送具体的内容到其他任务中去。

|  |
| --- |
| OS\_EVENT \*OSMboxCreate (void \*pmsg)  函数功能：创建消息邮箱  函数形参：初始的消息内容  函数返回值：消息邮箱的事件控制块指针  void \*OSMboxPend (OS\_EVENT \*pevent,  INT32U timeout,  INT8U \*perr)  函数功能：获取一封邮件  函数形参：OS\_EVENT \*pevent：消息邮箱的事件控制块指针  INT32U timeout：等待时间长度；0表示：无限等待  INT8U \*perr：保存错误类型；OS\_ERR\_NONE：表示获取成功；其他表示失败  函数返回值：返回接收到的邮件的首地址  INT8U OSMboxPost (OS\_EVENT \*pevent,  void \*pmsg)  函数功能：发送一份邮件  函数形参：OS\_EVENT \*pevent：消息邮箱的事件控制块指针  void \*pmsg：发送邮件内容的首地址  函数返回值：错误类型；OS\_ERR\_NONE：表示邮件发送成功；其他表示失败  注意点：消息邮箱只能存放一封邮件，如果不及时读取邮件，会产生邮件覆盖现象，邮件丢失现象 |

消息邮箱的使用：

|  |
| --- |
| OS\_EVENT OS\_EVENT\_KEY\_MBOX;  OS\_EVENT\_KEY\_MBOX = \*OSMboxCreate (NULL);  typedef struct {  u8 key\_value;//保存按键的值  u8 message\_buff[20];//保存内容  }KEY\_MESSAGE;  KEY\_MESSAGE key\_task\_message;  /\*  函数功能：KEY任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_key\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;  u8 error = 0;  u8 button\_value = 0;    p\_arg = p\_arg;    while(1)  {  button\_value = key\_scan();  if(button\_value != KEY\_ERR)  {  error = OSSemPost(&OS\_EVENT\_KEY\_SEM);  if(error != OS\_ERR\_NONE)  {  printf("file:%s\tline:%d\r\n",\_\_FILE\_\_,\_\_LINE\_\_);  }  switch(button\_value)  {  case KEY\_ONE:  key\_task\_message.key\_value = 1;  strcpy((char \*)key\_task\_message.message\_buff,"key\_task\_message\_1");  break;  case KEY\_TWO:  key\_task\_message.key\_value = 2;  strcpy((char \*)key\_task\_message.message\_buff,"key\_task\_message\_2");  break;  case KEY\_THR:  key\_task\_message.key\_value = 3;  strcpy((char \*)key\_task\_message.message\_buff,"key\_task\_message\_3");  break;  case KEY\_FOU:  key\_task\_message.key\_value = 4;  strcpy((char \*)key\_task\_message.message\_buff,"key\_task\_message\_4");  break;  }  error = OSMboxPost(&OS\_EVENT\_KEY\_MBOX,&key\_task\_message);  if(error != OS\_ERR\_NONE)  {  printf("file:%s\tline:%d\r\n",\_\_FILE\_\_,\_\_LINE\_\_);  }  }  OSTimeDly(1);//释放CPU控制权    // OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  // printf("os\_key\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  }  KEY\_MESSAGE beep\_task\_message;  /\*  函数功能：BEEP任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_beep\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;    u8 error = 0;    p\_arg = p\_arg;    while(1)  {  beep\_task\_message = \*(KEY\_MESSAGE \*)OSMboxPend(&OS\_EVENT\_KEY\_MBOX,0,&error);  if(error == OS\_ERR\_NONE)  {  switch(beep\_task\_message.key\_value)  {  case 1:  printf("%s\r\n",beep\_task\_message.message\_buff);  break;  case 2:  printf("%s\r\n",beep\_task\_message.message\_buff);  break;  case 3:  printf("%s\r\n",beep\_task\_message.message\_buff);  break;  case 4:  printf("%s\r\n",beep\_task\_message.message\_buff);  break;  }  }    OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  printf("os\_beep\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  } |

消息邮箱只能发送一则消息。获取消息的地方如果处理比较慢就会丢失消息。

消息邮箱使用的时候必须是发送消息的任务发送比较慢，接收消息的任务处理消息能力比较强。

### 消息队列

作用：传送具体的内容到其他任务。

消息队列中能够放的邮件的数量要比消息邮箱多，消息队列能够存放多少封邮件由程序员决定，但是不能超过极限值并且注意占用内存情况。

消息队列：存储一队消息。消息容量增加了。很好的避免接收方处理消息能力弱的时候而丢失消息。

消息队列使用的时候可以是发送消息的任务发送比较快，接收消息的任务处理消息能力比较弱。

使用的是队列形式存储；满足的是先进先出原则。

|  |
| --- |
| OS\_EVENT \*OSQCreate (void \*\*start,  INT16U size)  函数功能：创建消息队列  函数形参：void \*\*start：消息队列的首地址  INT16U size：消息队列的长度(能够存放多少封邮件)  函数返回值：消息队列的事件控制块指针  INT8U OSQFlush (OS\_EVENT \*pevent)  函数功能：清空消息队列中的消息  函数形参：OS\_EVENT \*pevent：消息队列的事件控制块指针  函数返回值：返回OS\_ERR\_NONE：清空消息队列成功  void \*OSQPend (OS\_EVENT \*pevent,  INT32U timeout,  INT8U \*perr)  函数功能：从消息队列中获取一封邮件  函数形参：OS\_EVENT \*pevent：消息队列的事件控制块指针  INT32U timeout：等待时间长度  INT8U \*perr：存储错误类型；OS\_ERR\_NONE：表示成功  函数返回值：返回从消息队列中获取到的邮件信息  INT8U OSQPost (OS\_EVENT \*pevent,  void \*pmsg)  函数功能：往消息队列中写入一封邮件  函数形参：OS\_EVENT \*pevent：消息队列的事件控制块指针  void \*pmsg：消息队列中的消息内容  函数返回值：错误类型；返回OS\_ERR\_NONE：写入邮件到消息队列成功 |

消息队列的使用：

|  |
| --- |
| /\*  函数功能：KEY任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_key\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;  u8 error = 0;  u8 button\_value = 0;  static u8 value = 0;    p\_arg = p\_arg;    while(1)  {  button\_value = key\_scan();  if(button\_value != KEY\_ERR)  {  error = OSSemPost(&OS\_EVENT\_KEY\_SEM);  if(error != OS\_ERR\_NONE)  {  printf("file:%s\tline:%d\r\n",\_\_FILE\_\_,\_\_LINE\_\_);  }  switch(button\_value)  {  case KEY\_ONE:  key\_task\_message[value].key\_value = 1;  strcpy((char \*)key\_task\_message[value].message\_buff,"key\_task\_message\_1");  break;  case KEY\_TWO:  key\_task\_message[value].key\_value = 2;  strcpy((char \*)key\_task\_message[value].message\_buff,"key\_task\_message\_2");  break;  case KEY\_THR:  key\_task\_message[value].key\_value = 3;  strcpy((char \*)key\_task\_message[value].message\_buff,"key\_task\_message\_3");  break;  case KEY\_FOU:  key\_task\_message[value].key\_value = 4;  strcpy((char \*)key\_task\_message[value].message\_buff,"key\_task\_message\_4");  break;  }  error = OSQPost(&OS\_EVENT\_KEY\_QUEUE,&key\_task\_message[value]);  if(error != OS\_ERR\_NONE)  {  printf("file:%s\tline:%d\r\n",\_\_FILE\_\_,\_\_LINE\_\_);  }  value++;  if(value == START\_TASK\_QUEUE\_LENGTH)  {  value = 0;  }  }  OSTimeDly(1);//释放CPU控制权    // OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  // printf("os\_key\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  }  /\*  函数功能：BEEP任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_beep\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;    u8 error = 0;  static u8 value = 0;    p\_arg = p\_arg;    while(1)  {  beep\_task\_message[value] = \*(KEY\_MESSAGE \*)OSQPend(&OS\_EVENT\_KEY\_QUEUE,0,&error);  if(error == OS\_ERR\_NONE)  {  OSTimeDly(200);  switch(beep\_task\_message[value].key\_value)  {  case 1:  printf("%s\r\n",beep\_task\_message[value].message\_buff);  break;  case 2:  printf("%s\r\n",beep\_task\_message[value].message\_buff);  break;  case 3:  printf("%s\r\n",beep\_task\_message[value].message\_buff);  break;  case 4:  printf("%s\r\n",beep\_task\_message[value].message\_buff);  break;  }  value++;  if(value == START\_TASK\_QUEUE\_LENGTH)  {  value = 0;  }  }    // OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  // printf("os\_beep\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  }  } |

消息队列中：如果发生消息队列的任务太快，处理消息的任务太慢，还是会丢失消息。

传过去的是传的地址，不是地址中的内容。如果需要保存不同的消息，需要给不同的地址。

### 互斥信号量

与信号量的原则一致。用来保护共享资源。但是这个资源的数目只能是一个。二值性问题。

两个任务同时操作一个硬件，这个时候需要加互斥信号量保护。

互斥信号量中需要有一个空闲的优先级作为优先级反转用；该优先级必须为能够获取到该互斥信号量的所有任务中优先级最高的还要高。

哪个任务使用了共享资源，就必须在该任务中释放共享资源。

|  |
| --- |
| OS\_EVENT \*OSMutexCreate (INT8U prio,  INT8U \*perr)  函数功能：创建一个互斥信号量  函数形参：INT8U prio：优先级，用作互斥信号量中优先级翻转用；比能够获得共享资源的所有任务优先级都要高  INT8U \*perr：错误类型：OS\_ERR\_NONE：创建成功，其他表示失败  函数返回值：返回互斥信号量事件控制块指针  void OSMutexPend (OS\_EVENT \*pevent,  INT32U timeout,  INT8U \*perr)  函数功能：获取互斥信号量  函数形参：OS\_EVENT \*pevent：互斥信号量事件控制块指针  INT32U timeout：等待时间长度  INT8U \*perr：错误类型：OS\_ERR\_NONE：获取成功，其他表示失败    INT8U OSMutexPost (OS\_EVENT \*pevent)  函数功能：释放互斥信号量  函数形参：OS\_EVENT \*pevent：互斥信号量事件控制块指针  函数返回值：错误类型：OS\_ERR\_NONE：释放成功，其他表示失败 |

互斥信号量的使用：

|  |
| --- |
| /\*  函数功能：LED任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_led\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;  u8 error = 0;    p\_arg = p\_arg;    while(1)  {  OSMutexPend(&OS\_EVENT\_MUTEX,0,&error);  if(error == OS\_ERR\_NONE)  {  NEGATE\_LED1 = !NEGATE\_LED1;  }    OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  printf("os\_led\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));    OSTimeDly(100);    OSMutexPost(&OS\_EVENT\_MUTEX);  }  }  /\*  函数功能：BEEP任务  函数形参：万能指针  函数返回值：无  备注：无  作者：李工  时间：2018-09-19  \*/  void os\_beep\_task(void \*p\_arg)  {  OS\_STK\_DATA p\_stk\_data;    u8 error = 0;    p\_arg = p\_arg;    while(1)  {  OSMutexPend(&OS\_EVENT\_MUTEX,0,&error);  printf("os\_beep\_task\r\n");  OSTimeDly(1000);//释放CPU控制权    OSTaskStkChk(OS\_PRIO\_SELF,&p\_stk\_data);  printf("os\_beep\_task = %d%%\r\n",p\_stk\_data.OSFree \* 100 / (p\_stk\_data.OSFree + p\_stk\_data.OSUsed));  OSMutexPost(&OS\_EVENT\_MUTEX);  }  } |

## UCOS操作系统事件标志组

事件标志组：发生某些事件，产生某些标志，对标志进行组合。相当于利用全局变量做为FLAG。

|  |
| --- |
| OS\_FLAG\_GRP \*OSFlagCreate (OS\_FLAGS flags,  INT8U \*perr)  函数功能：创建事件标志组  函数形参：OS\_FLAGS flags：初始化给个标志状态  INT8U \*perr：错误类型  函数返回值：事件标志组控制块指针。  OS\_FLAGS OSFlagPend (OS\_FLAG\_GRP \*pgrp,  OS\_FLAGS flags,  INT8U wait\_type,  INT32U timeout,  INT8U \*perr)  函数功能：获取一个标志  函数形参：OS\_FLAG\_GRP \*pgrp：事件标志组控制块指针。  OS\_FLAGS flags：程序员写的一个值。  INT8U wait\_type：等待的类型  OS\_FLAG\_WAIT\_CLR\_ALL：等待程序员写的值中1的位置全部为0.  OS\_FLAG\_WAIT\_SET\_ALL：等待程序员写的值中1的位置全部为1.  OS\_FLAG\_WAIT\_CLR\_ANY：等待程序员写的值中1的位置任意一个为0.  OS\_FLAG\_WAIT\_SET\_ANY：等待程序员写的值中1的位置任意一个为1.  加上OS\_FLAG\_CONSUME可以达到自动清标志。  INT32U timeout：等待的时长  INT8U \*perr：错误类型  函数返回值：如果获取正确，返回的是当前这个标志的值。  OS\_FLAGS OSFlagPost (OS\_FLAG\_GRP \*pgrp,  OS\_FLAGS flags,  INT8U opt,  INT8U \*perr)  函数功能：设置一个标志  函数形参：OS\_FLAG\_GRP \*pgrp：事件标志组控制块指针。  OS\_FLAGS flags：程序员写的一个值。  INT8U opt：类型；标志类型：  OS\_FLAG\_SET：将写入值中1的位置置为1.  OS\_FLAG\_CLR：将写入值中1的位置清为0.  INT8U \*perr：错误类型  函数返回值：返回设置的标志值。 |

## UCOS操作系统软件定时器

### 延时函数

|  |
| --- |
| void OSTimeDly (INT32U ticks)  函数功能：延时多少个心跳节拍  函数形参：ticks：心跳节拍数量  INT8U OSTimeDlyHMSM (INT8U hours,  INT8U minutes,  INT8U seconds,  INT16U ms)  函数功能：延时多少小时、分钟、秒钟、毫秒  函数形参：hours：小时范围：0~255  minutes：0~59  seconds：0~59  ms：0~999 |

### 软件定时器

|  |
| --- |
| OS\_TMR \*OSTmrCreate (INT32U dly,  INT32U period,  INT8U opt,  OS\_TMR\_CALLBACK callback,  void \*callback\_arg,  INT8U \*pname,  INT8U \*perr)  函数功能：创建一个软件定时器  软件定时器：定时准确度不高，想要精准的定时，还是采用硬件定时器。  利用硬件定时器产生中断的方式去操作。 |

## UCOS操作系统其它功能

|  |
| --- |
| void OSSchedLock (void)  函数功能：给调度器上锁：不让调度器调度任务。  注意：在有挂起的任务里面不能随便使用该函数。  如果想要使用必须：在进入挂起之前给调度器解锁。  写UCOS操作系统的代码的时候会有一个起始任务，这个任务一般用作开机界面；  一般情况下起始任务的优先级很低；  为了防止创建其它任务的时候跳转到其它任务中去执行程序，都会加上调度器上锁。  创建所有的任务结束以后会进行解锁。  void OSSchedUnlock (void)  函数功能：给调度器解锁。  INT16U OSVersion (void)  函数功能：用来获取版本号的；UCOS版本号。  OS\_ENTER\_CRITICAL()  宏定义：功能：进入临界区；关闭中断功能。就算发生中断，也不会进入中断服务函数。  OS\_EXIT\_CRITICAL()  宏定义：功能：退出临界区；开启中断功能。发生中断，就会进入中断服务函数 |