1）实验平台：【正点原子】 NANO STM32F103 开发板
2）摘自《正点原子STM32 F1 开发指南(NANO 板-HAL 库版)》关注官方微信号公众号，获取更多资料：正点原子

图 34.1.1 两个任务使用事件进行通信的示意图

在图 34.1.1 中任务 1 是发信方，任务 2 是收信方。任务 1 负责把信息发送到事件上，这项

操作叫做发送事件。任务 2 通过读取事件操作对事件进行查询：如果有信息则读取，否则等待。

读事件操作叫做请求事件。

为了把描述事件的数据结构统一起来，UCOSII 使用叫做事件控制块(ECB)的数据结构来描

述诸如信号量、邮箱（消息邮箱）和消息队列这些事件。事件控制块中包含包括等待任务表在

内的所有有关事件的数据，事件控制块结构体定义如下：

```
typedef struct

{

 INT8U OSEventType;

//事件的类型

 INT16U OSEventCnt;

//信号量计数器
```

void *OSEventPtr;

//消息或消息队列的指针

INT8U OSEventGrp;

//等待事件的任务组

INT8U OSEventTbl[OS_EVENT_TBL_SIZE];//任务等待表

#if OS_EVENT_NAME_EN > 0u

INT8U *OSEventName;

//事件名

#endif

} OS_EVENT;

信号量

信号量是一类事件。使用信号量的最初目的，是为了给共享资源设立一个标志，该标志表

示该共享资源的占用情况。这样，当一个任务在访问共享资源之前，就可以先对这个标志进行

查询，从而在了解资源被占用的情况之后，再来决定自己的行为。

信号量可以分为两种：一种是二值型信号量，另外一种是 N 值信号量。

二值型信号量好比家里的座机，任何时候，只能有一个人占用。而 N
值信号量，则好比公

共电话亭，可以同时有多个人（N 个）使用。

UCOSII 将二值型信号量称之为也叫互斥型信号量，将 N
值信号量称之为计数型信号量，

也就是普通的信号量。本章，我们介绍的是普通信号量，互斥型信号量的介绍，请参考《嵌入

式实时操作系统 UCOSII 原理及应用》5.4 节。

接下来我们看看在 UCOSII 中，与信号量相关的几个函数（未全部列出，下同）。

1） 创建信号量函数

在使用信号量之前，我们必须用函数 OSSemCreate
来创建一个信号量，该函数的原型

为：

OS_EVENT *OSSemCreate (INT16U cnt);

该函数返回值为已创建的信号量的指针，而参数 cnt
则是信号量计数器（OSEventCnt）

的初始值。

2） 请求信号量函数

任务通过调用函数 OSSemPend 请求信号量，该函数原型如下：

void OSSemPend（OS_EVENT *pevent, INT16U timeout, INT8U *err);

其中，参数 pevent 是被请求信号量的指针，timeout 为等待时限，err
为错误信息。

为防止任务因得不到信号量而处于长期的等待状态，函数 OSSemPend
允许用参数

timeout 设置一个等待时间的限制，当任务等待的时间超过 timeout
时可以结束等待状态而

进入就绪状态。如果参数 timeout 被设置为 0，则表明任务的等待时间为无限长。

3） 发送信号量函数

任务获得信号量，并在访问共享资源结束以后，必须要释放信号量，释放信号量也叫

做发送信号量，发送信号通过 OSSemPost 函数实现 。OSSemPost
函数在对信号量的计数

器操作之前，首先要检查是否还有等待该信号量的任务。如果没有，就把信号量计数器

OSEventCnt 加一；如果有，则调用调度器 OS_Sched(
)去运行等待任务中优先级别最高的

任务。函数 OSSemPost 的原型为：

INT8U OSSemPost(OS_EVENT *pevent);

其中，pevent 为信号量指针，该函数在调用成功后，返回值为
OS_ON_ERR，否则会

根据具体错误返回 OS_ERR_EVENT_TYPE、OS_SEM_OVF。

4 ） 删除信号量函数

应用程序如果不需要某个信号量了，那么可以调用函数 OSSemDel
来删除该信号量，

该函数的原型为：

OS_EVENT *OSSemDel (OS_EVENT *pevent,INT8U opt, INT8U *err);

其中，pevent 为要删除的信号量指针，opt 为删除条件选项，err 为错误信息。

邮箱

在多任务操作系统中，常常需要在任务与任务之间通过传递一个数据（这种数据叫做“消

息” ）的方式来进行通信。为了达到这个目的，可以在内存中创建一个存储空间作为该数据的

缓冲区。如果把这个缓冲区称之为消息缓冲区，这样在任务间传递数据（消息）的最简单办法

就是传递消息缓冲区的指针。我们把用来传递消息缓冲区指针的数据结构叫做邮箱（消息邮箱）。

在 UCOSII 中，我们通过事件控制块的 OSEventPrt
来传递消息缓冲区指针，同时使事件控

制块的成员 OSEventType 为常数
OS_EVENT_TYPE_MBOX，则该事件控制块就叫做消息邮箱。

接下来我们看看在 UCOSII 中，与消息邮箱相关的几个函数。

1） 创建邮箱函数

创建邮箱通过函数 OSMboxCreate 实现，该函数原型为：

OS_EVENT *OSMboxCreate (void *msg);

函数中的参数 msg 为消息的指针，函数的返回值为消息邮箱的指针。

调用函数 OSMboxCreate 需先定义 msg
的初始值。在一般的情况下，这个初始值为

NULL；但也可以事先定义一个邮箱，然后把这个邮箱的指针作为参数传递到函数

OSMboxCreate 中，使之一开始就指向一个邮箱。

2） 向邮箱发送消息函数

任务可以通过调用函数 OSMboxPost 向消息邮箱发送消息，这个函数的原型为：

INT8U OSMboxPost (OS_EVENT *pevent,void *msg);

其中 pevent 为消息邮箱的指针，msg 为消息指针。

3） 请求邮箱函数

当一个任务请求邮箱时需要调用函数
OSMboxPend，这个函数的主要作用就是查看邮

箱指针 OSEventPtr 是否为 NULL，如果不是 NULL
就把邮箱中的消息指针返回给调用函数

的任务，同时用 OS_NO_ERR 通过函数的参数 err
通知任务获取消息成功；如果邮箱指针

OSEventPtr 是 NULL，则使任务进入等待状态，并引发一次任务调度。

函数 OSMboxPend 的原型为：

void *OSMboxPend (OS_EVENT *pevent, INT16U timeout, INT8U *err);

其中 pevent 为请求邮箱指针，timeout 为等待时限，err 为错误信息。

4） 查询邮箱状态函数

任务可以通过调用函数 OSMboxQuery 查询邮箱的当前状态。该函数原型为：

INT8U OSMboxQuery(OS_EVENT *pevent,OS_MBOX_DATA *pdata);

其中 pevent 为消息邮箱指针，pdata 为存放邮箱信息的结构。

5） 删除邮箱函数

在邮箱不再使用的时候，我们可以通过调用函数 OSMboxDel
来删除一个邮箱，该函

数原型为：

OS_EVENT *OSMboxDel(OS_EVENT *pevent,INT8U opt,INT8U *err);

其中 pevent 为消息邮箱指针，opt 为删除选项，err 为错误信息。

关于 UCOSII
信号量和邮箱的介绍，就到这里。更详细的介绍，请参考《嵌入式实时操作

系统 UCOSII 原理及应用》第五章。

34.2 硬件设计

本节实验功能简介：本章我们在 UCOSII 里面创建 6 个任务：开始任务、LED0 任务、LED1

任务、数码管显示任务、按键扫描任务和主任务，开始任务用于创建信号量、创建邮箱、初始

化统计任务以及其他任务的创建，之后挂起；LED0 任务用于 DS0 控制，提示程序运行状况；

LED1 任务用于测试信号量，是请求信号量函数，每得一个信号量，DS1 就会闪一下；数

码管显示任务用于测试数码管显示；按键扫描任务用于按键扫描，优先级最高，将得到的键值

通过消息邮箱发送出去；主任务则通过查询消息邮箱获得键值，并根据键值执行信号量发送

（DS1 控制）、数码管显示变化的控制。

所要用到的硬件资源如下：

1）指示灯 DS0 、DS1

2）2 个按键（KEY0/KEY1）

3）数码管

这些，我们在前面的学习中都已经介绍过了。

34.3 软件设计

本章，我们在第十七章实验（实验 12 ）的基础上修改，具体方法同上一章一模一样，本

章我们就不再详细介绍了。

在加入 UCOSII 代码后，我们只需要修改 main.c 函数了，打开 main.c，输入如下代码：

```
//////////////////////////UCOSII 任务设置////////////////////////////////////

//START 任务

//设置任务优先级

#define START_TASK_PRIO

10 //开始任务的优先级设置为最低

//设置任务堆栈大小

#define START_STK_SIZE

64

//任务堆栈

OS_STK START_TASK_STK[START_STK_SIZE];

//任务函数

void start_task(void *pdata);

//LED0 任务

//设置任务优先级

#define LED0_TASK_PRIO

7

//设置任务堆栈大小
```

#define LED0_STK_SIZE

64

//任务堆栈

OS_STK LED0_TASK_STK[LED0_STK_SIZE];

//任务函数

void led0_task(void *pdata);

//数码管显示任务

//设置任务优先级

#define SMG_TASK_PRIO

6

//设置任务堆栈大小

#define SMG_STK_SIZE

64

//任务堆栈

OS_STK SMG_TASK_STK[SMG_STK_SIZE];

//任务函数

void smg_task(void *pdata);

//LED1 任务

//设置任务优先级

#define LED1_TASK_PRIO

5

//设置任务堆栈大小

#define LED1_STK_SIZE

64

//任务堆栈

OS_STK LED1_TASK_STK[LED1_STK_SIZE];

//任务函数

void led1_task(void *pdata);

//主任务

//设置任务优先级

#define MAIN_TASK_PRIO

4

//设置任务堆栈大小

#define MAIN_STK_SIZE

128

//任务堆栈

OS_STK MAIN_TASK_STK[MAIN_STK_SIZE];

//任务函数

void main_task(void *pdata);

//按键扫描任务

//设置任务优先级

#define KEY_TASK_PRIO

3

//设置任务堆栈大小

#define KEY_STK_SIZE

64

//任务堆栈

OS_STK KEY_TASK_STK[KEY_STK_SIZE];

//任务函数

void key_task(void *pdata);

//

OS_EVENT * msg_key;

//按键邮箱事件块指针

OS_EVENT * sem_led1;

//LED1 信号量指针

//共阴数字数组

//0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, .,全灭

u8

smg_num[]={0xfc,0x60,0xda,0xf2,0x66,0xb6,0xbe,0xe0,0xfe,0xf6,0xee,0x3e,0x9c,0x7a,0x9e,0x8e,0

```
x01,0x00};
u8 smg_duan=0;//数码管段选
int main(void)
{
 HAL_Init(); //初始化 HAL 库
 Stm32_Clock_Init(RCC_PLL_MUL9); //设置时钟,72M
 delay_init(72); //初始化延时函数
LED_Init();
 //初始化与 LED 连接的硬件接口
KEY_Init();
 //按键初始化
LED_SMG_Init(); //数码管初始化
 OSInit();
OSTaskCreate(start_task,(void *)0,(OS_STK
*)&START_TASK_STK[START_STK_SIZE-1],START_TASK_PRIO );//创建起始任务
OSStart();
}
//开始任务
void start_task(void *pdata)
{
```

```
 OS_CPU_SR cpu_sr=0;

pdata = pdata;

msg_key=OSMboxCreate((void*)0); //创建消息邮箱

sem_led1=OSSemCreate(0);

 //创建信号量

OSStatInit();

 //初始化统计任务.这里会延时 1 秒钟左右

OS_ENTER_CRITICAL();

 //进入临界区(无法被中断打断)

OSTaskCreate(led0_task,(void *)0,

(OS_STK*)&LED0_TASK_STK[LED0_STK_SIZE-1],LED0_TASK_PRIO);

OSTaskCreate(smg_task,(void *)0,

(OS_STK*)&SMG_TASK_STK[SMG_STK_SIZE-1],SMG_TASK_PRIO);

OSTaskCreate(led1_task,(void *)0,

(OS_STK*)&LED1_TASK_STK[LED1_STK_SIZE-1],LED1_TASK_PRIO);

OSTaskCreate(main_task,(void *)0,

(OS_STK*)&MAIN_TASK_STK[MAIN_STK_SIZE-1],MAIN_TASK_PRIO);

OSTaskCreate(key_task,(void *)0,

(OS_STK*)&KEY_TASK_STK[KEY_STK_SIZE-1],KEY_TASK_PRIO);

OSTaskSuspend(START_TASK_PRIO); //挂起起始任务.
```

```
OS_EXIT_CRITICAL();

//退出临界区(可以被中断打断)

}

//LED0 任务

void led0_task(void *pdata)

{

u8 t;

while(1)

{

t++;

delay_ms(10);

if(t==8)LED0=1; //LED0 灭

if(t==100)

//LED0 亮

{

t=0;

LED0=0;

}

}

}
```

```
//LED1 任务
void led1_task(void *pdata)
{
u8 err;
while(1)
{
OSSemPend(sem_led1,0,&err);
LED1=0;
delay_ms(200);
LED1=1;
delay_ms(800);
}
}
//数码管显示任务
void smg_task(void *pdata)
{
while(1)
{
LED_Write_Data(smg_num[smg_duan],7);//数码管显示
 LED_Refresh();//刷新显示
```

```
 delay_ms(10);
}
}
//主任务
void main_task(void *pdata)
{
u32 key=0;
u8 err;
while(1)
{
key=(u32)OSMboxPend(msg_key,10,&err);
switch(key)
{
case KEY0_PRES://发送信号量
OSSemPost(sem_led1);
break;
case KEY1_PRES://数码管显示加 1
 smg_duan++;
 if(smg_duan==17) smg_duan=0;
break;
```

```
}

delay_ms(10);

}

}

//按键扫描任务

void key_task(void *pdata)

{

u8 key;

while(1)

{

key=KEY_Scan(0);

if(key)OSMboxPost(msg_key,(void*)key);//发送消息

delay_ms(10);

}

}
```

该部分代码我们创建了 6
个任务：start_task、led0_task、smg_task、led1_task、main_task

和 key_task，优先级分别是 10 和 7~3，堆栈大小除了 main_task 是
128，其他都是 64。

该程序的运行流程就比上一章复杂了一些，我们创建了消息邮箱
msg_key，用于按键任务

和主任务之间的数据传输（传递键值），另外创建了信号量 sem_led1，用于 LED1 任务和主任

务之间的通信。

软件设计部分就为大家介绍到这里。

34.4 下载验证

在代码编译成功之后，我们通过下载代码到 NANO STM32 V1 上，可以看到数码管显示如

图 34.4.1 所示：