AN1201 UCOSII 实例讲解

扩展实验 14：UCOSII 多任务运行 LED+KEY+LCD+触摸画笔

1. 实验目的：测试 UCOSII 多任务的创建和运行。
2. 实现现象：LED0，LED1 循环闪烁，LCD 部分区域循环变色，触摸屏下半部分具有触摸画板功能，按下 KEY0 按键可以进入触摸校准界面。
3. 用到的 UCOSII 函数简析：

   UCOSII 初始化函数：void OSInit (void) //UCOSII 初始化

   任务创建函数：

   INT8U OSTaskCreate (void (*task)(void *pd),void*pdata,OS_STK *ptos,INT8U prio)；

   启动任务函数：void OSStart (void);//启动任务

   延时函数 OSTimeDlyHMSM(0,0,0,200);

   这里我们着重介绍一下 OSTaskCreate()函数和 OSTimeDlyHMSM()函数：

   这个函数的入口参数是任务指针，任务传递参数，任务堆栈栈顶指针以及任务优先级。

   在这里我们介绍一下这几个入口参数的作用：

   1) 任务优先级：多任务执行的时候，操作系统必须选择一个任务来执行，因为一个 CPU 同一时间只能执行一个任务，这里 UCOSII 是按优先级抢占式规则来选择任务的，所以对于每个任务，都会定义一个优先级，而且优先级是唯一的。
   2) 任务堆栈：存储器中按先进后(LIFO)出原则组织的连续存储空间，作用是满足任务切换和相应中断时保存 CPU 寄存器的内容和任务调用其他函数的需要。这里学过单片机的人都知道中断的时候有一个现场保护的概念，现场保护就是用到堆栈。

      定义堆栈的方式：OS_STK TASK_START_STK[START_STK_SIZE];

      其中 START_STK_SIZE 是我们宏定义的任务堆栈的大小。
   3) 任务指针：就是指向任务执行入口地址的指针了。C 语言里面函数名字就可以看做函数的入口地址了。
   4) 任务传递参数：这个就不用讲解，顾名思义就是传递给任务的参数了。

   OSTimeDlyHMSM()函数是非常重要的函数，该函数表面看是进行延时，实际上是使任务运行延时（暂停）一段时间并进行一次任务调度，释放 CPU 使用权。所以简而言之，当任务运行到延时函数的时候，他将释放 CPU 使用权，等待延时结束之后重新进入就绪状态。

   任务初始化函数 OSInit (void)在系统启动之后必须先调用此函数初始化 UCOSII 之后才能调用创建函数 OSTaskStart 以及启动任务函数 OSStart();

   对于启动任务函数 OSStart()，UCOSII 要求是在调用此函数之前系统必须创建至少一个任务，这里我们就创建了 TaskStart 任务，在这个任务中，我们完成其他任务的创建。
4. 实验描述

   OSTaskCreate()函数创建 TaskStart 任务，,在 TaskStart()任务中 5 次调用 ucos 任务创

建函数 OSTaskCreate（）创建 5 个任务：TaskLed，TaskLed1，TaskLCD，TaskKey，TaskTouch。然后调用 OSTaskSuspend()函数将 TaskStart 任务挂起，因为在 5 个任务创建后，TaskStart 任务该做的事情已经完毕，挂起任务。

TaskLed： LED0 每隔 500ms 状态反转。

TaskLed1： LED1 每隔 200ms 状态反转。

TaskLCD: LCD 上半部分一定区域颜色循环更换。

TaskKey： 每隔 20ms 扫描按键值，当 KEY0 按下时，进入触摸屏校准界面。

TaskTouch： 每隔 2ms 扫描触摸屏下半部分的触摸点，并显示在 LCD 上。也就是我们的触摸画板程序。NOTE:如果触摸屏不准，请按下 KEY0 进入触摸屏校准程序。

扩展实验 15：UCOSII 任务的挂起，恢复和删除测试

1. 实验目的：测试 UCOSII 的任务的挂起，恢复和删除。
2. 实现现象：开机之后，LED0,LED1 闪烁，LCD 上半部分部分区域循环变色，触摸屏下半部分具有触摸画板功能。

按下 KEY0 按键，LED0 停止闪烁，LCD 停止循环变色，

按下 KEY1 按键，LED0 恢复闪烁，LCD 恢复循环变色，

按下 KEY2(WK_UP)按键，LED0 停止闪烁，LCD 停止循环变色。这个时候再按下 KEY1 按键将无法恢复，因为任务已经进入睡眠状态，不被调度。

3. 用到的 UCOSII 函数简析：

这里对于之前实验讲解过的函数我们不重复讲解了，只讲解新用到的函数。

任务挂起函数： INT8U OSTaskSuspend (INT8U prio)；将优先级别为 prio 的任务挂起，挂起任务就是停止任务的运行，并触发一次调度。

任务恢复函数： INT8U OSTaskResume (INT8U prio)；将优先级为 prio 的任务恢复，恢复任务就是让挂起的任务进入就绪状态，并触发一次调度。

任务请求删除函数： INT8U OSTaskDelReq (INT8U prio)；请求删除优先级别 prio 的任务。

任务删除函数： INT8U OSTaskDel (INT8U prio)；删除优先级为 prio 的任务，删除任务之后，任务身份吊销了，没法再运行了。

这里我们可以看到这里关于任务操作的函数的入口参数都只有一个优先级,因为优先级在 UCOSII 里面对于每个任务是唯一的，所以可以用来识别任务。

4. 实验描述

OSTaskCreate()函数创建 TaskStart 任务，在 TaskStart 任务 5 次调用 ucos 任务创建函数 OSTaskCreate 创建 5 个任务：TaskLed，TaskLed1，TaskLCD，TaskKey，TaskTouch。然后调用 OSTaskSuspend()函数将 TaskStart 任务挂起，因为在 5 个任务创建后，TaskStart 任务该做的事情已经完毕，挂起任务。然后 5 个任务在循环执行。

TaskLed： LED0 每隔 500ms 状态反转

TaskLed1： LED1 每隔 200ms 状态反转

TaskLCD: LCD 上半部分一定区域颜色循环更换

TaskKey： 每隔 20ms 扫描按键值

TaskTouch： 每隔 2ms 扫描触摸屏下半部分的触摸点，并显示在 LCD 上。也就是我们的触摸画板程序。

在任务 TaskKey 中，我们循环扫描键值，如果 KEY0 按下，那么调用

```
OSTaskSuspend(LED_TASK_Prio);
OSTaskSuspend(LCD_TASK_Prio);
```

将任务 TaskLed 和任务 TaskLCD 挂起，这个时候，任务将不在执行（LED0 停止闪烁，LCD 停止循环显示），处于等待状态直到在其他任务中调用解挂函数 OSTaskResume()将任务解挂。

如果 KEY1 按键，将调用函数：

```
OSTaskResume(LED_TASK_Prio);
OSTaskResume(LCD_TASK_Prio);
```

将任务 TaskLed 和任务 TaskLCD 恢复，这个时候，任务将重新开始进入就绪状态，并引发一次任务调度。我们便可以看到 LED0 恢复闪烁，LCD 恢复循环显示。

如果 KEY2 按键按下，那么将调用函数：

```
OSTaskDelReq(LED_TASK_Prio);
OSTaskDelReq(LCD_TASK_Prio);
```

请求将任务 TaskLed 和 TaskLCD 删除，记住，这里只是请求而不是删除。那么在任务 TaskLed 和 TaskLCD 执行的时候，将同时调用这个方法判断是否有任务删除请求，如果有那么将执行删除操作：

```
if(OSTaskDelReq(OS_PRIO_SELF)==OS_TASK_DEL_REQ)
  OSTaskDel(OS_PRIO_SELF);
```

也就是说，删除任务是分两步来执行，第一步为请求删除任务，第二步才是删除任务。这样做的好处是在系统设计的时候避免直接删除导致任务有些资源没有释放而导致系统运行不正常。

删除任务之后，任务 TaskLed 和 TaskLCD 将处于睡眠状态，将不会被系统调度。这个时候可以看到 LED0 不再闪烁，LCD 也不会循环显示。

扩展实验 16：UCOSII 信号量的使用测试

1. 实验目的：信号量创建请求发送使用测试。
2. 实验现象：

按下 KEY0 按键 LED0 将闪烁 5 次（10 次反转），
按下 KEY1 按键 LED0 将反转一次，
按下 KEY2 按键 LED0 将循环闪烁，同时再按下 KEY0,KEY1 之后 LED0 状态不会改变.。

3. 用到的 UCOSII 函数

这里我们不再重复讲解之前实验讲解过的函数。
信号量创建函数

OS_EVENT *OSSemCreate (INT16U cnt)//创建初始值为 cnt 的信号量。

信号量发送函数：

INT8U OSSemPost (OS_EVENT *pevent)；//调用一次，信号量计数器加 1。

信号量请求函数：

void OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)//请求一次，信号量计数器减 1。

信号量删除函数：

OS_EVENT *OSSemDel (OS_EVENT *pevent, INT8U opt, INT8U *err)//删除信号量，信号量相关函数将无效。

4. 实验描述

关于信号量的基本概念之类的我们不详细讲解，我们光盘提供的 UCOSII 相关的资料有详细讲解这些概念，同时，推荐看任哲的书籍《嵌入式实时操作系统 UCOSII 原理与应用》。
OSTaskCreate()函数创建 TaskStart 任务，在 TaskStart 任务 5 次调用 ucos 任务创建函数 OSTaskCreate（）创建 5 个任务：TaskLed，TaskLed1，TaskLCD，TaskKey，TaskTouch，同时调用信号量创建函数 Sem_Event=OSSemCreate(4) 创建信号量 Sem_Event。然后调用 OSTaskSuspend()函数将 TaskStart()任务挂起，因为在 5 个任务创建后，TaskStart 任务该做的事情已经完毕，挂起任务。然后 5 个任务在循环执行。

TaskLed： 若请求得到信号量有效，LED0 每隔 600ms 状态反转
TaskLed1： LED1 每隔 600ms 状态反转
TaskLCD: LCD 上半部分一定区域颜色循环更换
TaskKey： 每隔 20ms 扫描按键值 。
TaskTouch： 每隔 2ms 扫描触摸屏下半部分的触摸点，并显示在 LCD 上。也就是我们的触摸画板程序。

在任务 TaskKey 中，我们循环扫描键值，如果 KEY0 按下，那么将调用信号量发送函数发送 10 次信号量，信号量计数器将增 10

```
for(i=0;i<10;i++)
OSSemPost(Sem_Event); //连续发送信号量 10 次
```

这个时候任务 TaskLed 正在请求信号量等待状态，此时任务将进入就绪状态， 将可以看到 LED0 闪烁 5 次。

如果按下 KEY1，那么将调用信号量发送函数 OSSemPost(Sem_Event);发送 1 次信号量，信号量计数器将增加 1，这个时候可以看到 LED0 状态反转一次。

如果 KEY2(WK_UP)按键按下，那么将调用信号量删除函数删除信号量，信号量将无效。这个时候 LED0 恢复闪烁。

```
OSSemDel(Sem_Event,OS_DEL_ALWAYS,&err);
```

其中参数 OS_DEL_ALWAYS 表明立即删除信号量。

扩展实验 17: UCOSII 消息邮箱使用测试

1. 实验目的：消息邮箱创建请求发送测试。
2. 实验现象： KEY0 按键按下，LED0,LED1 没有变化，
   KEY1 按键按下，LED1 状态反转，
   KEY2(WK_UP)按下，LED0,LED1 状态反转。

用到的 UCOSII 函数
消息邮箱创建函数：

OS_EVENT *OSMboxCreate (void *msg)//创建消息邮箱

请求消息邮箱函数:

void *OSMboxPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)//请求消息邮箱

向邮箱发送消息函数:

INT8U OSMboxPost (OS_EVENT *pevent, void *msg)//向等待任务表中高优先级的任务发送消息

广播邮箱消息函数：

INT8U OSMboxPostOpt (OS_EVENT *pevent, void *msg, INT8U opt)//向等待任务表中所有任务发送消息

OSTaskCreate()函数创建 TaskStart 任务，在 TaskStart 任务 5 次调用 ucos 任务创建函数 OSTaskCreate()创建 5 个任务：TaskLed，TaskLed1，TaskLCD，TaskKey，TaskTouch。同时创建消息邮箱 Str_Box = OSMboxCreate ((void*)0),然后调用 OSTaskSuspend()函数将 TaskStart()任务挂起，因为在 5 个任务创建后，TaskStart 任务该做的事情已经完毕，挂起任务。然后 5 个任务在开始执行执行。

TaskLed： 如果收到消息 1 或 3，那么 LED0 反转
TaskLed1： 如果收到消息 2 或 3，那么 LED1 反转
TaskLCD: LCD 上半部分一定区域颜色循环更换
TaskKey： 每隔 20ms 扫描按键值 。
TaskTouch： 每隔 2ms 扫描触摸屏下半部分的触摸点，并显示在 LCD 上。也就是我们的触摸画板程序。

按键扫描任务中，如果 KEY0 被按下，那么将向消息邮箱 Str_Box 发送消息 1，

```
i=1;
OSMboxPost(Str_Box,&i); //发送消息 1
```

如果 KEY1 被按下，那么将发送消息 2，

```
i=2;
OSMboxPost(Str_Box,&i); //发送消息 2
```

如果 KEY2(WK_UP)被按下，将向所有等待任务表中所有任务发送消息 3，

```
i=3;
OSMboxPostOpt(Str_Box,&i,OS_POST_OPT_BROADCAST); 向所有任务广播消息 3
```

我们可以看到，如果我们按下 KEY0,因为 TaskLed1 的优先级别高于 TaskLed，所以当两个任务都在等待的时候，只有 TaskLed1 可以收到消息 1，所以两个 LED 都不反转。

如果按下 KEY1,那么 TaskLed1 收到消息 2，状态反转。
如果按下 KEY2(WK_UP)，那么 TaskLed1 和 TaskLed 都会收到消息，这个时候两个 LED 状态都会反转。

OSMboxPostOpt()和 OSMboxPost()的区别在于前者是广播消息，所有等待任务都可以收到，后者只会高优先级的任务收到。

扩展实验 18：UCOSII 消息队列的使用测试

1. 实验目的：消息队列创建发送请求测试.
2. 实验现象：KEY0 按键按下，显示 Received 到的消息数据顺序为 4,3,2,1,0
   KEY1 按键按下，显示 Received 到的消息数据顺序为 0,1,2,3,4
   当然，这只是测试程序，你的按键不要连续按。

3.用到的 UCOSII 函数

这里我们不重复讲解之前讲解过的函数。

消息队列创建函数：

OS_EVENT *OSQCreate (void **start, INT16U size)

LIFO 方式发送消息函数：

INT8U OSQPostFront (OS_EVENT *pevent, void *msg)//后进先出

FIFO 方式发送消息函数：

INT8U OSQPost (OS_EVENT *pevent, void *msg)//先进先出

OSTaskCreate()函数创建 TaskStart 任务，在 TaskStart 任务 4 次调用 ucos 任务创建函数 OSTaskCreate()创建 4 个任务：TaskLed，TaskLed1，TaskLCD，TaskKey 。同时创建消息队列 Str_Q = OSQCreate(&MsgGrp[0],N_MESSAGES);。然后调用 OSTaskSuspend()函数将 TaskStart 任务挂起，因为在 4 个任务创建后，TaskStart 任务该做的事情已经完毕，挂起任务。然后 4 个任务在开始执行执行。

TaskLed： LED0 循环闪烁，反转间隔为 200ms。
TaskLed1： LED1 循环闪烁,反转间隔为 200ms。
TaskLCD: 每隔 50ms 请求消息队列，并显示得到的消息。
TaskKey： 每隔 20ms 扫描按键值 。

TaskKey 进行按键扫描，这里在任务 TaskLCD 和 TaskKey 之间有一个任务挂起 OSTaskSuspend 和恢复 OSTaskResume 操作，这个操作的目的是为了让按键扫描之后发送消息到消息队列完成之后，TaskLCD 任务才开始请求消息邮箱，也就是消息发送完成了才开始请求，这样方便查看队列顺序。

如果为 KEY0 按下，那么以 LIFO(后进先出)方式向消息队列发送消息 0-4，发送之后，可以看到液晶显示收到的消息顺序为 4,3,2,1,0，为什么？因为是后进先出嘛，自然最后发送的 4 最先收到了。
如果 KEY1 按下，那么以 FIFO(先进先出)方式向消息队列发送消息 0-4，发送之后，可以看到液晶显示收到的消息顺序为 0,1,2,3,4，因为是先进先出嘛，自然最先发送的 0 最先收到。

ALIENTEK//广州星翼电子科技有限公司
开发板购买店铺：http://eboard.taobao.com
技术支持论坛: www.openedv.com
2012 年 7 月 22 日