昵称：sky1991
园龄：4年8个月
粉丝：28
关注：0
+加关注

< 2017年3月 >

日	一	二	三	四	五	六
26	27	28	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

最新随笔
1. 安卓使用WIFI调试程序
2. 一步步写STM32 OS【四】OS基本框架
3. 一步步写STM32 OS【三】PendSV与堆栈操作
4. 一步步写STM32 OS【二】环境搭建
5. 一步步写STM32 OS【一】 序言
6. [转]关于WM_NCHITTEST消息
7. [转]ARM QT实现多点触摸
8. W3C SVG标准Web颜色值列表
9. [转]SYSZUXpinyin中文输入法的移植（到QT）

一步步写STM32 OS【四】OS基本框架

一、上篇回顾

上一篇文章中，我们完成了两个任务使用PendSV实现了互相切换的功能，下面我们接着其思路往下做。这次我们完成OS基本框架，即实现一个非抢占式（已经调度的进程执行完成,然后根据优先级调度等待的进程）的任务调度系统，至于抢占式的，就留给大家思考了。上次代码中Task_Switch实现了两个任务的切换，代码如下：

```
void Task_Switch()
{
  if(g_OS_Tcb_CurP == &TCB_1)
    g_OS_Tcb_HighRdyP=&TCB_2;
  else
    g_OS_Tcb_HighRdyP=&TCB_1;
  OSCtxSw();
}
```

我们把要切换任务指针付给跟_OS_Tcb_HighRdyP，然后调用OSCtxSw触发PendSV异常，就实现了任务的切换。如果是多个任务，我们只需找出就绪任务中优先级最大的切换之即可。

二、添加任务调度功能

为了实现这一目标我们至少需要知道任务的状态和时间等数据。我们定义了一个任务状态枚举类型OS_TASK_STA，方便添加修改状态。在OS_TCB结构体中添加了两个成员TimeDly和State，TimeDly是为了实现OS_TimeDly，至于State与优先级一起是作为任务切换的依据。

```
typedef enum OS_TASK_STA
{
  TASK_READY,
  TASK_DELAY,
} OS_TASK_STA;

typedef struct OS_TCB
{
  OS_STK *StkAddr;
  OS_U32 TimeDly;
  OS_TASK_STA State;
}OS_TCB,*OS_TCBP;
```

说到任务切换，我们必须面对临界区的问题，在一些临界的代码两端不加临界区进去和退出代码，会出现许多意想不到的问题。以下地方需要特别注意，对关键的全局变量的写操作、对任务控制块的操作等。进入临界区和退出临界区需要关闭和开启中断，我们采用uCOS中的一部分代码：

```
PUBLIC OS_CPU_SR_Save
  PUBLIC OS_CPU_SR_Restore

OS_CPU_SR_Save
    MRS     R0, PRIMASK
    CPSID   I
```

10. [转]交叉编译tslib1.4

我的标签

stm32(5)

OS(4)

MSP430(3)

调试(2)

蜂鸣器(1)

汉字(1)

汇编(1)

驱动(1)

时钟(1)

位带(1)

更多

随笔分类(81)

Arm(11)

C#、QT(10)

FPGA(4)

Java(1)

Linux(3)

单片机(48)

一步步写STM32 OS(4)

随笔档案(84)

2014年3月 (1)

2013年11月 (3)

2013年10月 (1)

2013年8月 (1)

2013年6月 (1)

2013年5月 (9)

2013年4月 (5)

2013年2月 (3)

2012年11月 (5)

```
    BX      LR

OS_CPU_SR_Restore
    MSR     PRIMASK, R0
    BX      LR
```

```
#define  OS_USE_CRITICAL    OS_U32 cpu_sr;
#define  OS_ENTER_CRITICAL()  {cpu_sr = OS_CPU_SR_Save();}
#define  OS_EXIT_CRITICAL()   {OS_CPU_SR_Restore(cpu_sr);}
#define  OS_PendSV_Trigger() OSCtxSw()
```

一个OS至少要有任务表，我们可以用数组，当然也可以用链表。为了简单，我们使用数组，使用数组下表作为优先级。当然，必要的地方一定要做数组越界检查。

```
#define OS_TASK_MAX_NUM 32
OS_TCBP OS_TCB_TABLE[OS_TASK_MAX_NUM];
```

为了使OS更完整，我们定义几个全局变量，OS_TimeTick记录系统时间，g_Prio_Cur记录当前运行的任务优先级，g_Prio_HighRdy记录任务调度后就绪任务中的最高优先级。

```
OS_U32 OS_TimeTick;
OS_U8 g_Prio_Cur;
OS_U8 g_Prio_HighRdy;
```

下面三个函数与PendSV一起实现了任务的调度功能。

OS_Task_Switch函数功能：找出已就绪最高优先级的任务，并将其TCB指针赋值给g_OS_Tcb_HighRdyP，将其优先级赋值g_Prio_HighRdy。注意其中使用了临界区。

```
void OS_Task_Switch(void)
{
  OS_S32 i;
  OS_TCBP tcb_p;
  OS_USE_CRITICAL
  for(i=0;i<OS_TASK_MAX_NUM;i++)
  {
    tcb_p=OS_TCB_TABLE[i];
    if(tcb_p == NULL) continue;
    if(tcb_p->State==TASK_READY) break;
  }
  OS_ENTER_CRITICAL();
  g_OS_Tcb_HighRdyP=tcb_p;
  g_Prio_HighRdy=i;
  OS_EXIT_CRITICAL();
}
```

OS_TimeDly至当前任务为延时状态，并将延时时间赋值给当前TCB的TimeDly成员，并调用OS_Task_Switch函数，然后触发PendSV进行上下文切换。OS_Task_Switch找到就绪状态中优先级最高的，并将其赋值相关全局变量，作为上下文切换的依据。

```
void OS_TimeDly(OS_U32 ticks)
{
    OS_USE_CRITICAL

    OS_ENTER_CRITICAL();
    g_OS_Tcb_CurP->State=TASK_DELAY;
    g_OS_Tcb_CurP->TimeDly=ticks;
    OS_EXIT_CRITICAL();
    OS_Task_Switch();
    OS_PendSV_Trigger();
}
```

SysTick_Handler实现系统计时，并遍历任务表，任务若是延时状态，就令其延时值减一，若减完后为零，就将其置为就绪状态。

2012年10月 (5)
2012年9月 (4)
2012年8月 (27)
2012年7月 (13)
2012年6月 (6)

积分与排名

积分 - 81360
排名 - 3027

最新评论

1. Re:[MSP430] 蜂鸣器演唱音乐
厉害啊，music_tab是怎么写的？
--侯胜滔

2. Re:一步步写STM32 OS【三】PendSV与堆栈操作
我大四呢，这不是选了一个坑爹的毕业设计嘛，看到你的帖子了，很赞！
--李可jesscia

3. Re:一步步写STM32 OS【三】PendSV与堆栈操作
@李可jesscia哈哈，博客好久不更新了，没想到还有评论。这里说声抱歉，这个博客是我本科时写的，现在已经不做那些东西了。年久失修，多数都已忘却。所以这个还要靠你自己了，加油。。。...
--sky1991

4. Re:一步步写STM32 OS【三】PendSV与堆栈操作
博主，我是初学嵌入式，看了你的内容很有用，求解一个问题，我想"验证中断级任务切换"，但是不会编写"OSTickISR"汇编，求助
--李可jesscia

5. Re:一步步写STM32 OS【一】序言
感谢分享，受益匪浅
--逝者如斯_不舍昼夜

```
void SysTick_Handler(void)
{
  OS_TCBP tcb_p;
  OS_S32 i;
  OS_USE_CRITICAL

  OS_ENTER_CRITICAL();
  ++OS_TimeTick;
    for(i=0;i<OS_TASK_MAX_NUM;i++)
    {
      tcb_p=OS_TCB_TABLE[i];
      if(tcb_p == NULL) continue;
      if(tcb_p->State==TASK_DELAY)
      {
        --tcb_p->TimeDly;
        if(tcb_p->TimeDly == 0)
          tcb_p->State=TASK_READY;
      }
    }
  OS_EXIT_CRITICAL();
}
```

当所有任务都没就绪怎么办？这时就需要空闲任务了，我们把它设为优先级最低的任务。WFE指令为休眠指令，当来中断时，退出休眠，然后看看有没有已就绪的任务，有则调度之，否则继续休眠，这样可以减小功耗哦。

```
void OS_Task_Idle(void)
{
  while(1)
  {
    asm("WFE");
    OS_Task_Switch();
    OS_PendSV_Trigger();
  }
}
```

当一个任务只运行一次时（例如下面main.c的task1），结束时就会调用OS_Task_End函数，此函数会调用OS_Task_Delete函数从任务表中删除当前的任务，然后调度任务。

```
void OS_Task_Delete(OS_U8 prio)
{
  if(prio >= OS_TASK_MAX_NUM) return;
  OS_TCB_TABLE[prio]=0;
}

void OS_Task_End(void)
{
  printf("Task of Prio %d End\n",g_Prio_Cur);
  OS_Task_Delete(g_Prio_Cur);
  OS_Task_Switch();
  OS_PendSV_Trigger();
}
```

三、OS实战

下面是完整的main.c代码：

```
#include "stdio.h"
#include "stm32f4xx.h"

#define OS_EXCEPT_STK_SIZE 1024
```

```
#define TASK_1_STK_SIZE 128
#define TASK_2_STK_SIZE 128
#define TASK_3_STK_SIZE 128

#define TASK_IDLE_STK_SIZE 1024
#define OS_TASK_MAX_NUM 32
#define OS_TICKS_PER_SECOND 1000

#define  OS_USE_CRITICAL    OS_U32 cpu_sr;
#define  OS_ENTER_CRITICAL()  {cpu_sr = OS_CPU_SR_Save();}
#define  OS_EXIT_CRITICAL()   {OS_CPU_SR_Restore(cpu_sr);}
#define  OS_PendSV_Trigger() OSCtxSw()

typedef signed char OS_S8;
typedef signed short OS_S16;
typedef signed int OS_S32;
typedef unsigned char OS_U8;
typedef unsigned short OS_U16;
typedef unsigned int OS_U32;
typedef unsigned int OS_STK;

typedef void (*OS_TASK)(void);

typedef enum OS_TASK_STA
{
  TASK_READY,
  TASK_DELAY,
} OS_TASK_STA;

typedef struct OS_TCB
{
  OS_STK *StkAddr;
  OS_U32 TimeDly;
  OS_U8 State;
}OS_TCB,*OS_TCBP;

OS_TCBP OS_TCB_TABLE[OS_TASK_MAX_NUM];
OS_TCBP g_OS_Tcb_CurP;
OS_TCBP g_OS_Tcb_HighRdyP;
OS_U32 OS_TimeTick;
OS_U8 g_Prio_Cur;
OS_U8 g_Prio_HighRdy;

static OS_STK OS_CPU_ExceptStk[OS_EXCEPT_STK_SIZE];
OS_STK *g_OS_CPU_ExceptStkBase;

static OS_TCB TCB_1;
static OS_TCB TCB_2;
static OS_TCB TCB_3;
static OS_TCB TCB_IDLE;
static OS_STK TASK_1_STK[TASK_1_STK_SIZE];
static OS_STK TASK_2_STK[TASK_2_STK_SIZE];
static OS_STK TASK_3_STK[TASK_3_STK_SIZE];
static OS_STK TASK_IDLE_STK[TASK_IDLE_STK_SIZE];

extern OS_U32 SystemCoreClock;

extern void OSStart_Asm(void);
extern void OSCtxSw(void);
extern OS_U32 OS_CPU_SR_Save(void);
extern void OS_CPU_SR_Restore(OS_U32);

void task_1(void);
void task_2(void);
void task_3(void);

void OS_Task_Idle(void);
void OS_TimeDly(OS_U32);
void OS_Task_Switch(void);
void OS_Task_Create(OS_TCB *,OS_TASK,OS_STK *,OS_U8);
void OS_Task_Delete(OS_U8);
void OS_Task_End(void);
void OS_Init(void);
void OS_Start(void);

void task_1(void)
{
```

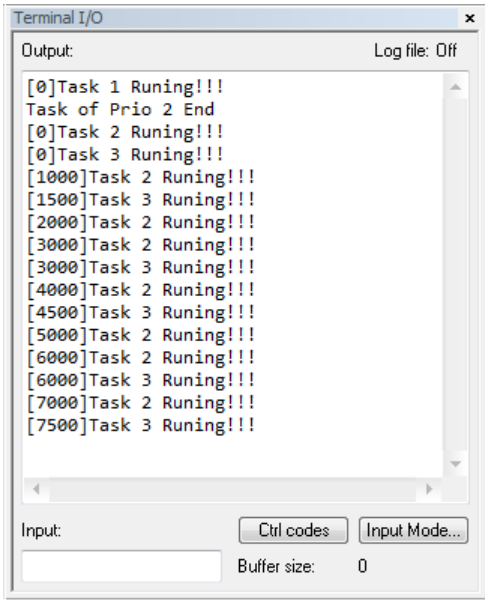
```
    printf("[%d]Task 1 Runing!!!\n",OS_TimeTick);

    OS_Task_Create(&TCB_2,task_2,&TASK_2_STK[TASK_2_STK_SIZE-1],5);
    OS_Task_Create(&TCB_3,task_3,&TASK_3_STK[TASK_3_STK_SIZE-1],7);
}

void task_2(void)
{
  while(1)
  {
    printf("[%d]Task 2 Runing!!!\n",OS_TimeTick);
    OS_TimeDly(1000);
  }
}

void task_3(void)
{
  while(1)
  {
    printf("[%d]Task 3 Runing!!!\n",OS_TimeTick);
    OS_TimeDly(1500);
  }
}

void OS_Task_Idle(void)
{
  while(1)
  {
    asm("WFE");
    OS_Task_Switch();
    OS_PendSV_Trigger();
  }
}

void OS_TimeDly(OS_U32 ticks)
{
    OS_USE_CRITICAL

    OS_ENTER_CRITICAL();
    g_OS_Tcb_CurP->State=TASK_DELAY;
    g_OS_Tcb_CurP->TimeDly=ticks;
    OS_EXIT_CRITICAL();
    OS_Task_Switch();
    OS_PendSV_Trigger();
}

void OS_Task_Switch(void)
{
  OS_S32 i;
  OS_TCBP tcb_p;
  OS_USE_CRITICAL
  for(i=0;i<OS_TASK_MAX_NUM;i++)
  {
    tcb_p=OS_TCB_TABLE[i];
    if(tcb_p == NULL) continue;
    if(tcb_p->State==TASK_READY) break;
  }
  OS_ENTER_CRITICAL();
  g_OS_Tcb_HighRdyP=tcb_p;
  g_Prio_HighRdy=i;
  OS_EXIT_CRITICAL();
}

void OS_Task_Delete(OS_U8 prio)
{
  if(prio >= OS_TASK_MAX_NUM) return;
  OS_TCB_TABLE[prio]=0;
}

void OS_Task_End(void)
{
  printf("Task of Prio %d End\n",g_Prio_Cur);
  OS_Task_Delete(g_Prio_Cur);
  OS_Task_Switch();
  OS_PendSV_Trigger();
}
```

```
void OS_Task_Create(OS_TCB *tcb,OS_TASK task,OS_STK *stk,OS_U8 prio)
{
    OS_USE_CRITICAL
    OS_STK  *p_stk;
    if(prio >= OS_TASK_MAX_NUM) return;

    OS_ENTER_CRITICAL();

    p_stk      = stk;
    p_stk      = (OS_STK *)((OS_STK)(p_stk) & 0xFFFFFFF8u);

    *(--p_stk) = (OS_STK)0x01000000uL;                          //xPSR
    *(--p_stk) = (OS_STK)task;                                  // Entry Point
    *(--p_stk) = (OS_STK)OS_Task_End;                  // R14 (LR)
    *(--p_stk) = (OS_STK)0x12121212uL;                          // R12
    *(--p_stk) = (OS_STK)0x03030303uL;                          // R3
    *(--p_stk) = (OS_STK)0x02020202uL;                          // R2
    *(--p_stk) = (OS_STK)0x01010101uL;                          // R1
    *(--p_stk) = (OS_STK)0x00000000u;                           // R0

    *(--p_stk) = (OS_STK)0x11111111uL;                          // R11
    *(--p_stk) = (OS_STK)0x10101010uL;                          // R10
    *(--p_stk) = (OS_STK)0x09090909uL;                          // R9
    *(--p_stk) = (OS_STK)0x08080808uL;                          // R8
    *(--p_stk) = (OS_STK)0x07070707uL;                          // R7
    *(--p_stk) = (OS_STK)0x06060606uL;                          // R6
    *(--p_stk) = (OS_STK)0x05050505uL;                          // R5
    *(--p_stk) = (OS_STK)0x04040404uL;                          // R4

    tcb->StkAddr=p_stk;
    tcb->TimeDly=0;
    tcb->State=TASK_READY;
    OS_TCB_TABLE[prio]=tcb;

    OS_EXIT_CRITICAL();
}

void SysTick_Handler(void)
{

  OS_TCBP tcb_p;
  OS_S32 i;
  OS_USE_CRITICAL

  OS_ENTER_CRITICAL();
  ++OS_TimeTick;
    for(i=0;i<OS_TASK_MAX_NUM;i++)
    {
      tcb_p=OS_TCB_TABLE[i];
      if(tcb_p == NULL) continue;
      if(tcb_p->State==TASK_DELAY)
      {
        --tcb_p->TimeDly;
        if(tcb_p->TimeDly == 0)
          tcb_p->State=TASK_READY;
      }
    }
  OS_EXIT_CRITICAL();
}


void OS_Init(void)
{
  int i;
  g_OS_CPU_ExceptStkBase = OS_CPU_ExceptStk + OS_EXCEPT_STK_SIZE - 1;
  asm("CPSID   I");
  for(i=0;i<OS_TASK_MAX_NUM;i++)
    OS_TCB_TABLE[i]=0;
  OS_TimeTick=0;
  OS_Task_Create(&TCB_IDLE,OS_Task_Idle,&TASK_IDLE_STK[TASK_IDLE_STK_SIZE-
1],OS_TASK_MAX_NUM-1);
}

void OS_Start(void)
{
  OS_Task_Switch();
  SystemCoreClockUpdate();
```

```
  SysTick_Config(SystemCoreClock/OS_TICKS_PER_SECOND);
  OSStart_Asm();
}

int main()
{

  OS_Init();
  OS_Task_Create(&TCB_1,task_1,&TASK_1_STK[TASK_1_STK_SIZE-1],2);
  OS_Start();

  return 0;
}
```

os_port.asm变化不大，具体内容可以下载文章末尾提供的工程参考。

老规矩，下载调试，全速运行，观察Terminal IO窗口：



从输出来看，我们已经完成了目标。但不保证稳定性，可能有不少Bugs。至此，可以说其实写一个OS并不难，难的是写一个稳定安全高效的OS。所以，现在只是走了一小步，想要完成一个成熟的OS，还需要不断测试，不断优化。例如，我们采用数组存储任务表，也可以采用链表，各有优缺点。我们只有一个任务表，也可以分成多个表，例如就续表，等待表等等。我们的任务调度部分运行时间不确定，对于实时OS，这是不可以的，怎么修改呢，例如像uCOS的查找表法那样。现在我们的系统只能创建并调度任务，还未加入其他功能，例如信号量、邮箱、队列、内存管理等。其实到了这里，大家完全可以发挥自己的创造力，参照本文开发自己的OS。如果以后有时间的话，还会再写几篇文章继续完善我们的OS。

四、工程下载

stepbystep stm32 os basic.rar

分类: 一步步写STM32 OS

标签: stm32, OS

sky1991
关注 - 0
粉丝 - 28
+加关注

1 0

« 上一篇：一步步写STM32 OS【三】PendSV与堆栈操作
» 下一篇：安卓使用WIFI调试程序

posted @ 2013-11-03 22:18 sky1991 阅读(4598) 评论(0) 编辑 收藏