单片机 RTOS 随想曲

本文标为随想曲,是说明这不是一篇结构严谨的文章,而是想到哪写到哪,希望读者能喜欢这种风格。以下为本文正文：

对于搞单片机的特别用 8051 系列工程师来说，谈到单片机的 RTOS，很多时候会问一句：“为什么要用 RTOS？单片机就这一点资源，使用 RTOS 能保证效率吗？”

对于这个问题，我会反问：“你用单片机的目的是什么？是为了用单片机的 C 编程，单片机的汇编编程甚至于用单片机的二进制指令编程？”上个世纪 80 年代，工程师用二进制指令给 Z80 编程，现在还有谁在用？现在还有人死抱着汇编不放，但越来越多的人工程师使用 C 编程（我起初也是使用汇编的），为什么？因为我们的目的是在有限的时间甚至是不充足的时间内把项目保质保量的完成！使用是么工具和方法是次要的（如果你的项目以成本放在第一位，则另当别论，这时，也是要考虑开发时间的）。时间就是金钱啊，一个产品在单片机上增加些许成本是可以接受的。况且，使用 8051 系列单片机时，单片机资源也常有富余，CPU 一般情况也只是空转，这就为它使用 RTOS 创造了条件。

那么，使用 RTOS 的好处呢？我举一个例子吧。假设我们编一个串行通讯程序，通讯协议如下：

数据包长度为 NBYTE，起始字节为 STARTBYTE1,STARTBYTE2，最后一个字节为检验和，中间字节不可能出现连续出现 STARTBYTE1，STARTBYTE2。

第一种方法，在中断中处理协议：

```
unsigned char Buf[NBYTE-2];
bit GetRight=0;
        void comm(void) interrupt 4
//"串行口中断"
{
    static unsigned char Sum,Flag=0,i;
    unsigned char temp;

    if(RI==1)
    {
        RI=0;
        temp=SBUF;
        switch(Flag)
        {
            case 0:
                if(temp==STARTBYTE1)
                {
                    Flag=1;
                }
                break;
            case 1:
```

```
            if(temp==STARTBYTE2)
            {
                Sum=STARTBYTE1+STARTBYTE2;
                i=0;
                Flag=2;
                break;
            }
            if(temp==STARTBYTE1) break;
            Flag=0;
            break;
        case 2:
            if(temp==STARTBYTE1)
            {
                Flag=3;
                break;
            }
            Sum+=temp;
            if((i>=(NBYTE-3))&&Sum==0)
            {
                GetRight=1;
                Flag=0;
                break;
            }
            Buf[i++]=temp;
            break;
        case 3:
            if(temp==STARTBYTE2)
            {
                Sum=STARTBYTE1+STARTBYTE2;
                Flag=2;
                i=0;
                break;
            }
            Sum+=STARTBYTE1;
            if((i>=(NBYTE-3))&&Sum==0)
            {
                GetRight=1;
                Flag=0;
                break;
            }
            Buf[i++]=STARTBYTE1;
            if(temp==STARTBYTE1)
            {
                break;
```

```
                    }
                    Sum+=temp;
                    if((i>=(NBYTE-3))&&Sum==0)
                    {
                        GetRight=1;
                        Flag=0;
                        break;
                    }
                    Buf[i++]=temp;
                    Flag=2;
                    break;
                }
            }
}
```

第二种方法，使用队列

中断函数：

```
        void comm(void) interrupt 4
//"串行口中断"
{
    if(RI==1)
    {
        RI=0;
        SBUF 入队;
    }
}
```

主程序不断调用的函数:

```
unsigned char Buf[NBYTE-2];

        unsigned char ReadSerial(unsigned char *cp)
{
    unsigned char i;
    unsigned char temp,Sum;

    temp=队列中数据个数;
    if(temp<(NBYTE)) return 0;
    出队 temp;
    if(temp!=STARTBYTE1) return 0;
    temp=队列首字节;
```

```
    if(temp!=STARTBYTE2) return 0;
    出队 temp;
    sum=STARTBYTE1+STARTBYTE2;
    for(i=0;i<NBYTE-3;i++)
    {
        temp=队列首字节;
        if(temp==STARTBYTE1)
        {
            temp=队列次首字节;
            if(temp==STARTBYTE2) return 0;
        }
        出队 temp;
        *cp++=temp;
        Sum+=temp;
    }
    temp=队列首字节;
    Sum+=temp;
    if(Sum!=0) return 0;
    出队 temp;
    return 1;
}
```

第三种方法，使用 RTOS

中断函数：

```
        void comm(void) interrupt 4
//"串行口中断"
{
    OS_INT_ENTER();
    if(RI==1)
    {
        RI=0;
        OSIntSendSignal(RECIVE_TASK_ID);
    }
    OSIntExit();
}
```

ID 为 RECIVE_TASK_ID 的任务

```
    void Recuve(void)
{
    unsigned char temp,temp1,Sum,i;
```

```
    OSWait(K_SIG,0);
    temp=SBUF;
    while(1)
    {
        while(1)
        {
            OSWait(K_SIG,0);
            temp1=SBUF;
            if((temp==STARTBYTE1)&&(temp1==STARTBYTE2)) break;
            temp=temp1;
        }
        Sum=STARTBYTE1+STARTBYTE2;

        OSWait(K_SIG,0);
        temp=SBUF;
        for(i=0;i<NBYTE-3;i++)
        {
            OSWait(K_SIG,0);
            temp1=SBUF;
            if((temp==STARTBYTE1)&&(temp1==STARTBYTE2))
            {
                OSWait(K_SIG,0);
                temp=SBUF;
                i=-1;
                Sum=STARTBYTE1+STARTBYTE2;
                continue;
            }
            Buf[i]=temp;
            Sum+=temp;
            temp=temp1;
        }
        Sum+=temp1;
        if(Sum==0) OSSendSignal(命令解释任务 ID);
    }
}
```

这几种方法的比较:

可读性和编程容易性方面，第三钟方法最好（如果允许使用 goto 语句，程序更加简单易读），第二种次之（因为要编队列程序），第一种最差。如果协议更加复杂，这方面更加明显。程序简单易读，自然出错机会小了。

RAM 占用方面，第三种方法较少，第二种最多（因为队列占用大量空间），第一种最少。

中断执行时间方面，第三种方法最长，第二种最短，第一种较长。

从功能方面，第三种方法最强，它还可以进行超时处理（虽然例子程序没有），其它方法均不行。

如果数据来的太快，命令处理程序来不及处理，三种方法处理方式不太一样，第一种和第三种方法类似：丢弃以前数据，第二种则是丢弃后到的数据。而且，第二种方法必须等命令处理程序完成后才处理下一个数据包，而第一种和第三种方只需命令处理程序将数据收取后就可处理下一个数据包。也就是说，第一种和第三种与命令处理程序并行处理，第二种方法为串行处理。

现在，一般情况下，开发的效率第一，执行的效率（包括执行时间和资源占用）第二。在这种情况下，降低些许效率换取开发的效率的较大提高，何乐而不为？何况，单个模块的执行的效率高不等于整个程序执行效率高。例如，如果程序需要等待一段时间，一般用程序延时或定时器延时。无论何种方法，CPU 不再处理其它工作，效率很低。而用 RTOS，等待的时候 CPU 可以处理其它工作，效率得到提高。

以下摘自《uC/OS-II--源码公开的实时嵌入式操作系统》

“实时内核也称为实时操作系统或 RTOS。使用它使得实时应用程序的设计和扩展变得容易。不需要大的改动就可以增加新的功能。通过应用程序分割为若干独立的任务，RTOS 使得应用程序的设计过程大为简化。使用可剥夺性的内核时，所有时间要求苛刻的事件都得到了尽可能快捷、有效的处理。通过有效的服务；如信号量、邮箱、队列、延时、超时等；RTOS 使得资源得到更好的利用。

“如果应用项目对额外的需求可以承受，应该考虑使用实时内核。这些额外的需求是：内核的价格，额外 ROM/RAM 开销，2 至 4 百分点的 CPU 额外负担。

“还有没提到的一个因素是使用实时内核增加的价格成本。在一些应用中，价格就是一切，以至于对使用 RTOS 连想都不敢想。”

总而言之，适用的就是最好的，不要拒绝 RTOS，在它适用的情况下，它工作得很好。

陈明计于 2002 年 3 月 1 日

（本文例子均经过验证，并附源程序下载。欢迎大家讨论，本人 email：chenmingji@cmmail.com）