基于 μC/OS_II 的优先级调度算法的改进

王娜娜,郭 兵
(四川大学 计算机学院,四川 成都 610065)

摘 要:μC/OS_Ⅱ内核最多可以管理64个任务,当工程的复杂度增加时,必须改换其他的开发平台,导致了前期工作变为徒劳。通过简易可行的方法来增加任务管理数目很有必要。μC/OS_Ⅱ内核原来的优先级调度算法的优先级变量总共8位,只用了其中的低6位,高2位未被使用。在尽量不改变内核的数据结构的情况下,为了增加内核可以管理任务的数目,把第7位使用起来,这样存放优先级就绪表的行信息将会增加1位,可以使任务数增加到128个。任务的就绪状态由一个二维数组存储,这样可以增加内核管理的任务数目,而较小程度地改变内核本身原来的数据结构,通过改进使它在嵌入式开发中的应用更广泛。

关键词:μC/OS_Ⅱ;就绪表;优先级;任务调度
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2011)11-0011-04

Improvement of Priority Scheduling Algorithm Based on μC/OS_II

WANG Na-na,GUO Bing
(College of Computer,Sichuan University,Chengdu 610065,China)

Abstract:The μC/OS_Ⅱ kernel can manage up to 64 tasks. When the complexity of the project increases,if the number of tasks is more than 64,you must switch to other development platforms,which may cause a lot of pre-development work void. To solve this problem, according to μC/OS_Ⅱ itself the task of scalability,a feasible method to increase the number of management tasks is proposed based on the original priority scheduling algorithm. Through the analysis of the μC/OS_Ⅱ kernel,find that the variables priority of original priority scheduling algorithm is a total of eight bits. And only the lower 6 bits of those 8 bits are used. The improved algorithm expands the number of tasks from 64 to 128. This improvement will make it more widely used in the actual embedded development.

Key words:μC/OS_Ⅱ;ready list;priority;task scheduling

0 引 言

μC/OS_Ⅱ操作系统是源代码开放的实时嵌入式操作系统。在μC/OS_Ⅱ中,其中有几个比较重要的概念,实时内核和任务切换与调度,以及任务优先级如何分配等[1]。其中,实时内核负责管理各个任务,确保所有时间关键的事件尽可能高效地得到处理,比如为每个任务分配CPU时间,并完成任务间的通信。μC/OS_Ⅱ内核最多可以管理64个任务,而且不同的任务的优先级必须与其他的任务不相同,所以可管理的任务数和优先级是一样多的,所以优先级为0~63,其中0为最高优先级,63为最低优先级,由于系统保留了4个最高优先级和4个最低优先级,所以用户可以使用的优先级为56个[2],而且μC/OS_Ⅱ系统的任务调度的依据是通过查找就绪优先级表,所以每个任务的优先级必须不同,所以用户定义的任务数最多为56个。因此,当工程的复杂性增大,如果总任务数目超过64个,就必须改换可以管理更多任务的开发平台,这样会浪费很大的人力物力[3]。

考虑到这种情况,在开发前就必须考虑到任务数的限制,根据μC/OS_Ⅱ本身的任务可扩展性,文中提出了一种可行的增加管理任务数的算法,鉴于大多数的任务项目是以128任务居多,可以把任务数由原来的64个扩展到128个。

1 μC/OS_Ⅱ的任务管理与调度算法

原来的就绪表中变量OSRdyGrp,共1byte,每位代表8个任务,若8个任务中任何一个任务进入就绪态则对应位置1。OSRdyTbl[8],是一维数组,共64bit,代表每一个优先级,每位优先级与一个任务一一对应,

收稿日期:2011-04-14;**修回日期**:2011-07-21
基金项目:四川省杰出青年科技基金(2010JQ0011)
作者简介:王娜娜(1985-),女,山东东营人,硕士研究生,主要研究领域为嵌入式实时系统;郭 兵,副教授,主要研究方向为嵌入式实时系统、SoC和中间件。

如果该任务就绪,就将相对应位置1。OSMapTbl[ ]用于在就绪表中将某任务对应位置1或置0操作。OSUnMapTbl[ ]是长度为256字节数组,用于查找当前就绪表中优先级最高的任务,是用于降低调度算法的时间复杂度的常量。系统运行时,任务状态的每次变化都要在就绪表OSRdyGrp中记录,所以内核函数可以根据就绪表判断任务的状态以及优先级。而且根据就绪表查找当前系统中就绪且优先级最高的任务,完成任务调度[4]。其中任务优先级用变量priority表示,OSMapTbl[ ]映射表的定义为INT8UconstOSMapTbl[ ] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80},作用是把3位的二进制数转换为相应的字节掩码,是以位图形式定义的。在就绪表中主要通过以下算法记录任务状态的变化。

1.1 原内核使任务进入就绪态的算法

首先,要使priority右移3位,在OSMapTbl中实现二进制数转换为相应的字节掩码,在OSRdyGrp标识是哪一组的进入了就绪态,实现如下:

```
OSRdyGrp | = OSMapTbl[priority>>3];
```

其次,就是确认是这一组的哪个任务,所以对priority与0x07相与,在OSMapTbl中实现二进制数转换为相应的字节掩码,在OSRdyTbl[priority>>3]中标识,实现如下:

```
OSRdyTbl[ priority>>3] | =OSMapTbl[priority& 0x07 ];
```

1.2 原内核使任务脱离就绪态的算法

其思路与进入就绪态的思路恰好相反,算法要实现把相应的任务对应的就绪表中的状态置0,首先就是把OSRdyTbl[priority> > 3]中对应的任务置0,代码如下:

```
if ( (OSRdyTbl[priority>> 3 ] & = ~OS2MapTbl
[priority& 0x07 ]) = = 0x00);
```

其次,要查看本组是否还有处于就绪态的任务,如果没有,则对应的OSRdyGrp也要置0,反之,则不需要置0,具体的代码如下:

```
OSRdyGrp & = ~OSMapTbl[ priority>>3 ] ;
```

1.3 原内核在就绪表中查找处于就绪状态且优先级最高的任务的算法

查找最高的就绪的任务就要遍历就绪表,实时系统的特点就是要保证时间的实时性,所以为了降低时间的复杂度,更快速的查找处于最高优先级的任务,内核引入了OSUnMapTbl[ ]映射表的概念,映射表可以实现由一个字节到三位二进制的快速映射。首先,要查找组,即处于优先级中最高的组。代码如下:

```
y =OSUnMapTbl[OSRdyGrp];
```

其次,要查找最高任务所在的组的哪一位,用一变量存储,实现代码如下:

```
x =OSUnMapTbl[OSRdyTbl[ y ]];
```

最后,要得出优先级数priority,就是把变量y左移3位与x相加即可,代码如下:

```
priority= y <<3+ x;
```

此内核支持64个优先级,优先级用变量priority的低6位表示,将其分为2组(高3位和低3位),每组值范围为0~7,算法中用y表示高位值,x表示低位值。

2 改进后的优先级调度算法

通过对μC/OS_Ⅱ内核的分析发现原来的优先级调度算法的priority变量总共8位,而原内核只用了其中的低6位,高2位未使用[5],为在尽量不改变内核的数据结构的情况下增加内核可以管理任务的数目,考虑到可以把priority的第7位使用起来,这样存放优先级就绪表的行信息将会增加1位,可以使任务数增加到128个。把原先INT8U OSRdyGrp,设置为INT8U OSRdyGrp[2],使OSRdyGrp可以对应OSRdyTbl[ ]中增加的行数。这样可以改变内核管理的任务数目,而较小程度地改变内核本身原来的数据结构。

2.1 改进后使任务放入就绪表

改进后的内核,就绪表变量OSRdyTbl[ ]增加为16个数组,用OSRdyGrp[2]来替换原来的OSRdyGrp,即用一个数组来替换原来的变量。OSRdyGrp[2]中,任务仍按优先级分组,8个任务为一组,共16组。OSRdyTbl[ ]数组与优先级在就绪表中的任务是否就绪是一一对应的,现在可以存储128个任务的信息,每一位对应的任务如果处于就绪态,则将该位置为1,每一位对应的任务如果脱离就绪态,则将该位置为0。OSRdyTbl[ ]前0~7行表示优先级为0~63的就绪状态,8~15行表示优先级为64~127的任务的就绪状态[6]。OSRdyGrp[ ]的每一位与16组任务中是否有进入就绪态的任务一一对应,即如果每一组中存在任何一个进入就绪态的任务,则相应位置为1,如果有任务脱离就绪态,则会检查本组是否还有处于就绪态的任务,如果没有任何任务处于就绪态,则相对应的位置0。OSRdyGrp[0]表示与OSRdyTbl[ ]的0~7行有没有进入就绪状态的对应,OSRdyGrp[1]表示与OSRdyTbl[ ]的64~127行有没有进入就绪状态的对应。由于最大任务数的改变,OSRdyTbl[ ]数组的下标则可以最大为15,如图1所示。其中,OSMapTbl[ ]是在ROM中用于限制OSRdyTbl[ ]数组的元素下标范围为0~7,所以算法要有所改进才能使OSRdyTbl[ ]数组的最大下标可以达到15。同时priority也要修改,将原来的未使用的第7位使用,用优先级priority的低

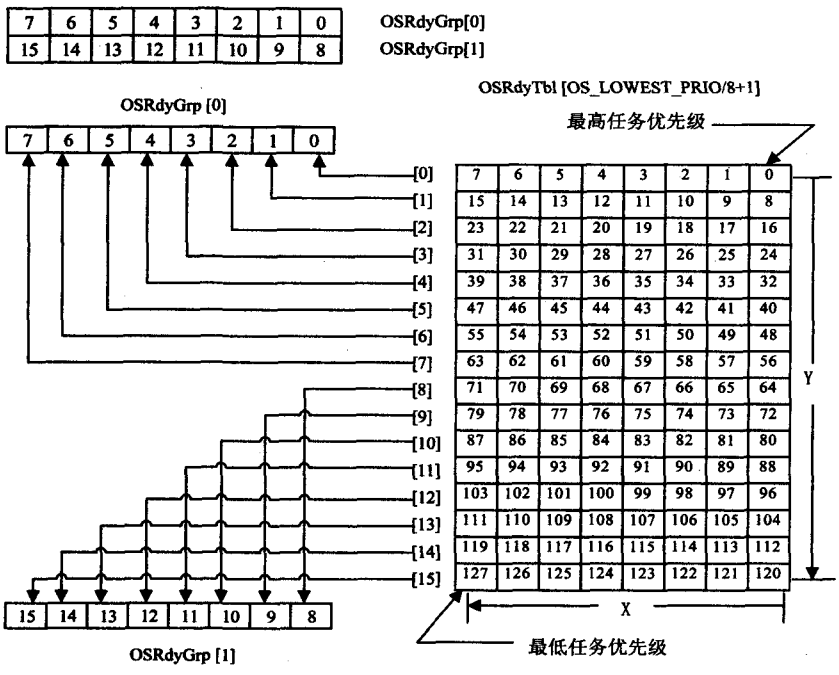

图 1　改进的 μC/OS_II 的任务就绪表

7 位表示,因为组数增加了一倍,由原来的 8 组成为了 16 组,所以由 4 位表示,将 priority 分为 2 组(高 4 位和低 3 位),算法中仍用 y 表示高位值,与 OSRdyGrp 中的组一一对应,x 表示低位值,与 OSRdyTbl[ ]组内的值一一对应,如图 2 所示。

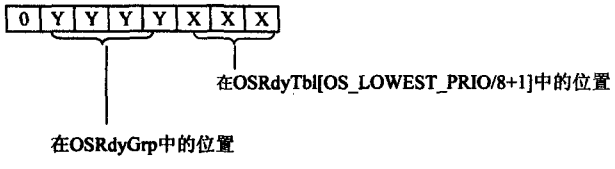

图 2　改进后 μC/OS_II 的优先级定位字节 priority

Bit0 in OSRdyGrp is 1 when any bit in OSRdyTbl [0] is 1.

Bit1 in OSRdyGrp is 1 when any bit in OSRdyTbl [1] is 1.

Bit2 in OSRdyGrp is 1 when any bit in OSRdyTbl [2] is 1.

Bit3 in OSRdyGrp is 1 when any bit in OSRdyTbl [3] is 1.

Bit4 in OSRdyGrp is 1 when any bit in OSRdyTbl [4] is 1.

Bit5 in OSRdyGrp is 1 when any bit in OSRdyTbl [5] is 1.

Bit6 in OSRdyGrp is 1 when any bit in OSRdyTbl [6] is 1.

Bit7 in OSRdyGrp is 1 when any bit in OSRdyTbl [7] is 1.

Bit8 in OSRdyGrp is 1 when any bit in OSRdyTbl [8] is 1.

Bit9 in OSRdyGrp is 1 when any bit in OSRdyTbl[9] is 1.

Bit10 in OSRdyGrp is 1 when any bit in OSRdyTbl[10] is 1.

Bit11 in OSRdyGrp is 1 when any bit in OSRdyTbl[11] is 1.

Bit12 in OSRdyGrp is 1 when any bit in OSRdyTbl[12] is 1.

Bit13 in OSRdyGrp is 1 when any bit in OSRdyTbl[13] is 1.

Bit14 in OSRdyGrp is 1 when any bit in OSRdyTbl[14] is 1.

Bit15 in OSRdyGrp is 1 when any bit in OSRdyTbl[15] is 1.

改进后的内核使任务进入就绪态的算法,首先,要使 priority 右移 3 位,在 OSMapTbl 中实现二进制数转换为相应的字节掩码,在 OSRdyGrp 标识是哪一组的进入了就绪态。其次,就是确认是这一组的哪个任务,所以对 priority 与 0x07 相与,在 OSMapTbl 中实现二进制数转换为相应的字节掩码,在 OSRdyTbl [priority>>3]中标识。优先级低于 64 的任务仍然按原来的方式进入就绪表,实现如下:

```
OSRdyGrp [0]      |=OSMapTbl [priority>>3];
OSRdyTbl [ priority >>3 ] |= OSMapTbl [ priority &0x07] ;
```

优先级高于 63 的任务,是对 OSRdyGrp [1]操作置 1,代码如下:

```
OSRdyGrp [1]|=OSMapTbl [(priority &0x3F) >> 3];
OSRdyTbl [ priority >>3 ] |= OSMapTbl [ priority &0x07] ;
```

2.2　改进后从就绪表中删除任务

其思路与进入就绪态的思路恰好相反,算法要实现把相应的任务对应的就绪表中的状态置 0,首先就是把 OSRdyTbl [priority> > 3]中对应的任务置 0,优先级小于 64 的任务脱离就绪态的代码如下:

```
If ((OSRdyTbl [ priority >>3] &= ~ OSMapTbl [priority &0x07]) == =0)
OSRdyGrp[0]  &= ~OSMapTbl[priority>>3];
```

优先级小于 63 的任务脱离就绪态的代码如下:

```
If ((OSRdyTbl [ priority >>3] &= ~ OSMapTbl [priority &0x07]) == =0)
OSRdyGrp [1] &= ~OSMapTbl [(priority&0x3F) >>3] ;
```

以上算法将任务表数组 OSRdyTbl[ ]中相应元素

的相应位清0,对不处于就绪态任务的进行标识;而对于OSRdyGrp[ ],只有当脱离就绪态的任务所在任务组中,没有任何一个任务处于就绪态时,才将相应的位清0,因为它是与一组任务的状态对应的。也就是说,OSRdyTbl[priority>>3]所有的位都是0时,OSRdyGrp[ ]相应的位才清0[7]。

2.3 改进后查找进入就绪态的优先级最高的任务

原内核中找到进入就绪态的优先级最高的任务,为了降低时间复杂度,只需查找另外一张存在内存的表,即优先级判定表OSUnMapTbl[256],改进后的算法仍然使用这张表。利用OSRdyGrp[0]为下标来查OSUnMapTbl表,返回的字节就是所有组中处于就绪态的任务中优先级最高的组所在的位置[8]。这样设计是为了节约时间,这也是实时系统必须要考虑的。修改后的查找进入就绪态的优先级最高的任务代码如下。

首先,如果OSRdyGrp[0]!=0则前8组中存在处于就绪态的任务,其算法思路与原内核一致,如下:

```
y = OSUnMapTbl[OSRdyGrp[0]];
x = OSUnMapTbl[OSRdyTbl[y]];
priority = (y << 3) + x;
```

其次,如果OSRdyGrp[0]=0则前8组中没有处于就绪态的任务,则在后8组中查找处于就绪态且优先级最高的任务,如下:

```
y = OSUnMapTbl[OSRdyGrp[1]] +8;
x = OSUnMapTbl[OSRdyTbl[y]];
priority = (y << 3) + x;
```

3 优先级算法的验证及性能分析

修改内核中相应的函数,然后用以测试函数测试可行性。修改任务调度涉及的主要函数,主要包括内核结构中的任务管理、时间管理以及时间控制块[9]。

因新内核支持的优先级为0~127,在Windows XP操作系统下安装Borland C/C++ V4.5x编译器,用一个演示μC/OS_Ⅱ的多任务处理能力的程序来验证可以处理的最大任务数[10]。在内核自带的OS_TASK.C文件中OSTaskCreate函数中每建立一个任务就会使变量OSTaskCtr加1,可以用OSTaskCtr统计系统的任务数,用sprintf(s, "%5d", OSTaskCtr)来显示任务数。通过测试函数验证,发现通过优先级算法的改进来扩展任务的做法是可行的。

下面从几个方面讨论这种改进的调度算法的优劣。改进前后的时间复杂度均为O(1)。而执行算法所耗费的存储空间不会增加,这里主要考虑辅助存储空间[11]。这种改进只修改了OSRdyGrp的结构,由原来的8位的OSRdyGrp改为8位的数组OSRdyGrp[2],内核的其他结构都不用改变,较有效地避免了内存的浪费,而且可以较容易地在原内核的基础上修改[12]。

因为增加了判定是不是高优先级,在原先的算法的基础上将会进行一次比较运算,执行算法所耗费的时间会增加,所以语句执行时间会增加,这是这个算法的不足和应该进一步优化的。

4 结束语

μC/OS_Ⅱ是国内的8位单片机中使用的比较多的实时操作系统。μC/OS_Ⅱ实时操作系统的任务调度采用位图方式,优先级事先设定,实时性很强。这个算法在尽量维持原先内核数据结构的情况下,以较少的内存空间开销达到了增加内核管理的任务数的目的,较好地改善了μC/OS_Ⅱ的任务数少的缺点,所以这个改进在实际的嵌入式应用中还是比较有意义的。

参考文献:

[1] 石世光,陈云洽,叶齐明.嵌入式操作系统μC/OS的运行机制[J].计算机技术与发展,2006,16(8):85-87.

[2] 王忠凯,赵 磊.UCOS-Ⅱ任务调度研究[J].山东理工大学学报,2009,23(2):30-35.

[3] 王 玲,杨红雨,张昭瑜.μC/OS2Ⅱ中优先级调度算法的改进及实现[J].四川大学学报:自然科学版,2005(4):700-705.

[4] Labrosse J J.嵌入式实时操作系统μC/OS_Ⅱ[M].邵贝贝,译.北京:北京航空航天大学出版社,2003.

[5] 罗 蕾.嵌入式实时操作系统及应用开发[M].北京:北京航空航天大学出版社,2005.

[6] 陈文星,张辉宜,周秀丽.嵌入式Linux的实时性改进技术[J].计算机技术与发展,2006,16(10):114-117.

[7] 姚 伟.嵌入式系统低功耗软件技术研究[J].计算机技术与发展,2011,21(1):112-115.

[8] Gibbons P B,Muchnic S S. Efficient Instruction Scheduling for a Pipelined Architecture[J]. SIGPLAN Notices,2004,39(4):169-174.

[9] HeurschA C,Cramhow D,Horstkotte A,et al. Steps Towards A Fully Preemptable Linux Kernel[J]. Real-time Programming,2003,3(1):17-20.

[10] 汪建新,潘雪珍.一个嵌入式实时操作系统的设计[J].江南大学学报:自然科学版,2005,4(4):375-377.

[11] Ghattas R,Dean A C. Preemption Threshold Scheduling:Stack Optimality,Enhancements and Analysis[C]//Real-Time and Embedded Technology and Applications Symposium.[s.l.]:[s.n],2007:147-157.

[12] 徐文清,杨红雨.一种基于动态优先级的实时混合任务调度算法[J].四川大学学报:自然科学版,2006(6):544-548.