文章编号:1007-1423(2013)15-0037-05 DOI:10.3969/j.issn.1007-1423.2013.15.009

一种 μC/OS-II 中任务调度机制的改进方法

赵国富 , 赵士伟

(山东理工大学网络信息中心, 淄博 255049)

摘 要: μC/OS-II 是一种源代码开放的实时嵌入式操作系统,支持抢占式固定优先级的任务调度方法。扩展 μC/OS-II 的任务调度机制,使之支持设定同优先级任务,并实现同优先级任务时间片轮转,高优先级任务可抢占的任务调度方式。μC/OS-II 中任务调度机制的改进方法既保留原调度机制的高度实时性,也增加任务调度方式的灵活性,并且实现简单、有效。

关键词: μC/OS-II; 时间片轮转; 任务调度

0 引 言

μC/OS-II 内核采取了占先式的固定优先级的任务调度策略,2.52 版本可支持 64 个任务, 每个任务都具有唯一的优先级,但是不支持同优先级任务的调度。占先式的固定优先级的任务调度策略需要在建立任务时就为每个任务设置唯一的固定的优先级,然而,在实际应用中,往往有些任务不易确定优先级的高低,或者逻辑上处于相同的优先级。如多点的温度或气压数据采集,若理解为不同的优先级任务去调度,不是一个好的逻辑设计。因而,研究基于 μC/OS-II 内核的,相同优先级的任务设置和调度方法具有重要的现实意义。本文中深入分析了 μC/OS-II 的任务调度机制,对任务调度函数进行简单的改进,仅增加几行代码,使之能够支持相同优先级的简单的时间片轮转调度方法, 完善了 μC/OS-II 的内核调度机制。

1 μC/OS-II 内核的任务调度算法

μC/OS-II 内核采取的是占先式的固定优先级的任务调度策略,每个任务的优先级都是事先唯一确定的。为了提高任务切换效率加强系统的实时性, 该内核在任务调度中采取了特殊的位图组织形式, 内核将系统中处于就绪态的任务在就绪表(Ready List)进行标注, 每次调度时利用查表法查找当前就绪的最高优先级的任务,然后切换该任务运行。μC/OS-II 的 2.52 版本可支持 64 个任务, 任务的调度与变量 OSRdyGrp 和 OSRdyTbl[]相关,任务就绪表位图结构如图 1 所示。

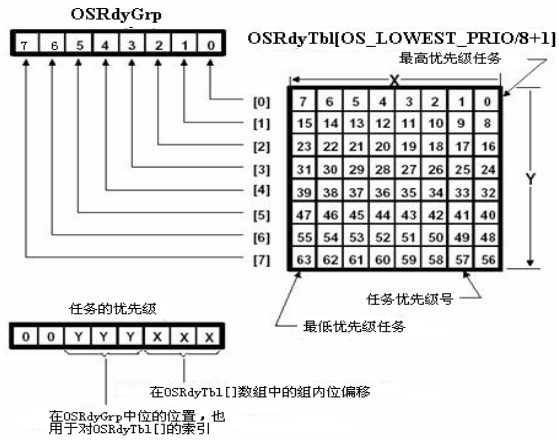

图 1 μC/OS-II 的任务就绪表

具体的任务调度过程如下:

(1)查找任务就绪表,找到处于就绪态的优先级最高的任务;

(2)根据查找到的具有最高优先级的就绪任务的优先级,查找任务控制块优先级表,得到拟调度任务的

收稿日期:2013-04-25 修稿日期:2013-05-10

作者简介:赵国富(1971-),男,山东潍坊人,高级工程师,硕士,研究方向为空间数据处理、计算机应用

任务控制块地址；

(3)切换程序运行环境至拟调度任务的运行环境，完成任务调度。

μC/OS-II 内核采取的是固定优先级的调度方式，每个任务建立时为之设置相应的任务控制块结构，而任务控制块优先级表则以优先级为索引，保存了指向各优先级的任务的任务控制块的指针，任务调度的第 2 步需要查找该表找到任务控制块地址，而完成调度，任务控制块优先级表的结构如图 2 所示。

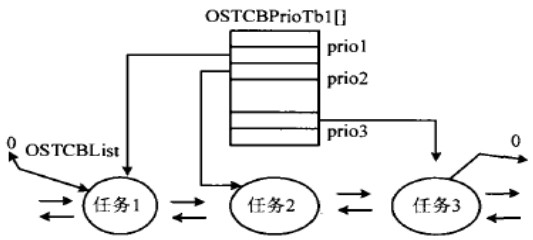

图 2 μC/OS-II 中 OSTCBList 结构

为了能够使 μC/OS-II 内核支持同优先级的任务调度，有的研究者扩展了任务控制块的数据结构[4]，并改造了任务控制块优先级表，如图 3 所示。图中任务 1、任务 2 和任务 3 为不同优先级的任务，任务 1、任务 4 和任务 5 为同优先级的任务，当没有比任务 1 更高的优先级任务时，每时间片任务 1、任务 4 和任务 5 就依次在运行态和就绪态切换。

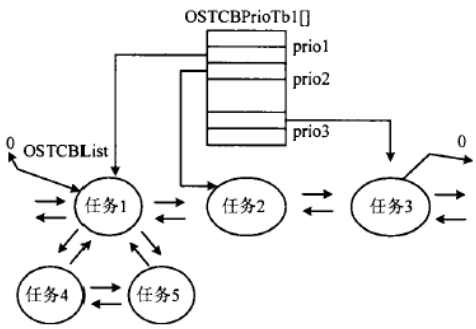

图 3 改进后的 OSTCBList 数据结构

可以看出这种方法工作量比较大，而且效率也不高。为了使同优先级的任务形成一个循环链表，需要在 OS_TCB 中扩展两个指针域，使就绪任务和等待任务不同优先级的控制块形成线性链表，而同优先级的控制块形成循环链表。这为就绪任务进入等待任务或等待任务进入就绪任务时链表上的任务控制块的插入和删除增添了不便。另外，这种做法涉及到的需要修改的内核函数很多，稍有不慎就会使系统崩溃。

2 较简单的内核任务调度改进方法

2.1 对 μC/OS-II 内核中任务优先级的重新解释

考虑到比较小的嵌入式系统应用程序任务优先级级别比较少，以及 μC/OS-II 的就绪表的特点及 μC/OS-II 的每个任务都具有唯一的优先级，任务的优先级也是任务的唯一编号(ID)，可以作为任务的唯一标识这几方面，对 μC/OS-II 的任务的优先级做了如下规定：

(1)内核仍最多支持 64 个任务，但是现存的 0~63 级这 64 级不同的优先级作为任务的编号。

(2)重新规定任务的优先级为 0~7 级，共 8 个不同的优先级级别，第 0 级优先级最高，第 7 级优先级最低(此处同 μC/OS-II 原内核任务优先级大小规定相同，优先级号小表示优先级高)。

(3)每个不同的优先级最多可以有 8 个任务，与不同的任务编号对应。任务编号与优先级的对应规则如表 1。

表 1 任务编号与优先级的对应规则表

任务编号	优先级
0~7	0
8~15	1
16~23	2
24~31	3
32~39	4
40~47	5
48~55	6
56~63	7

任务创建时把所需要设定的优先级作为需要设定的任务编号，也即任务创建时必须指定任务的编号，任务的编号和优先级的关系如表 1。

2.2 改进后的 μC/OS-II 内核调度方法

该改进后的内核，调度方法为：在就绪表中查找是否有更高优先级的任务就绪，如果有则调度更高优先级的就绪任务；如果没有更高优先级任务就绪则查看本优先级内是否有就绪的任务，若有并且当前任务的时间片已运行完则进行调度。下面详细介绍具体实现方法。

观察 μC/OS-II 的就绪表如图 1 所示，修改后的就绪表如图 4 所示。

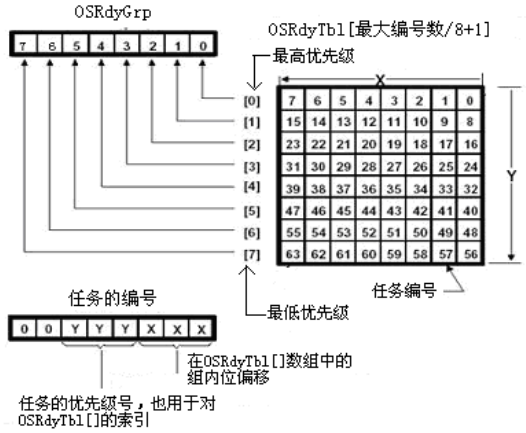

图 4 修改后的就绪表

经修改后，就绪表的变量数组OSRdyTbl[]的大小由最大任务编号数决定而不是最低优先级，内核中可以不必修改而直接把 OS_LOWEST_PRIO 当成最大编号数。任务的编号分成 2 部分：3~5 位为任务的优先级 y，0~2 位为任务在 OSRdyTbl[]数组内的偏移位 x，任务的编号也即 y×8+x，这与未修改前优先级的数值是一样的。同理，OSRdyGrp 的各位对应相应的优先级是否有就绪任务，规则如下：

●当 OSRdyTbl[0]中任何一位是 1 时，即优先级为 0，OSRdyGrp 的第 0 位置 1；

●当 OSRdyTbl[1]中任何一位是 1 时，即优先级为 1，OSRdyGrp 的第 1 位置 1；

●当 OSRdyTbl[2]中任何一位是 1 时，即优先级为 2，OSRdyGrp 的第 2 位置 1；

●当 OSRdyTbl[3]中任何一位是 1 时，即优先级为 3，OSRdyGrp 的第 3 位置 1；

●当 OSRdyTbl[4]中任何一位是 1 时，即优先级为 4，OSRdyGrp 的第 4 位置 1；

●当 OSRdyTbl[5]中任何一位是 1 时，即优先级为 5，OSRdyGrp 的第 5 位置 1；

●当 OSRdyTbl[6]中任何一位是 1 时，即优先级为 6，OSRdyGrp 的第 6 位置 1；

●当 OSRdyTbl[7]中任何一位是 1 时，即优先级为 7，OSRdyGrp 的第 7 位置 1；

算法实现的关键是同优先级任务如何调度。修改前，内核的调度代码如下：

```
void OS_Sched (void)
{
    #if OS_CRITICAL_METHOD == 3
   OS_CPU_SR  cpu_sr;
    #endif
        INT8U  y;
    OS_ENTER_CRITICAL();
  If((OSIntNesting==0)&&(OSLockNesting==0)){
   y=OSUnMapTbl[OSRdyGrp];                ①
OSPrioHighRdy=(INT8U)((y<<3)+
                OSUnMapTbl[OSRdyTbl[y]]);      ②
    if (OSPrioHighRdy ! = OSPrioCur) {
    OSTCBHighRdy=OSTCBPrioTbl[OSPrioHighRdy];
   OSCtxSwCtr++;
   OS_TASK_SW();
    }
     }
  OS_EXIT_CRITICAL();
  }
```

标号①②的代码用于找出就绪态且优先级最高的任务的优先级，其中标号①的代码根据 OSRdyGrp 的内容在 OSUnMapTbl[]表中查找出就绪的优先级最高的任务所在的组，代码②组合就绪的优先级最高的任务所在的组号和组内地址，得到就绪的优先级最高的任务的优先级，接着进行调度。

做了前面提到的规定之后发现标号①的代码可以找到进入就绪态的任务的最高优先级，下一步就应该判断是同优先级任务调度还是不同优先级的任务调度，对于不同的情况再给予不同的处理，修改后代码为：

```
void OS_Sched(void)
{
/* 部分代码与原代码相同省略 */
  y=OSUnMapTbl[OSRdyGrp];              ①
  /* 判断是否与当前运行态任务的优先级相同 */
  /* 若相同则查找下时间片需要运行的任务号 */
  /* 与运行态优先级不同则判断需要运行的任务号 */
  /* 任务切换 */
/* 部分代码与原代码相同省略 */
}
```

同一优先级的任务调度采用轮转法，在通常的轮

转法中，系统将所有的进程按先来先服务的原则，排成一个队列，每次调度时把 CPU 分配给队首进程，并令其执行一个时间片[5]。如果按这个方法进行修改，代码修改量势必会很大。考虑到在修改中做了同一优先级最多有 8 个任务且每个任务有唯一编号的规定，可以按任务编号进行轮转调度，即调度时，查找同优先级中编号大于当前任务编号且编号最小的任务，如果没有则再在同优先级的编号最小的任务开始轮转。

该方法的实现也可以借助 OSUnMapTbl[]表，只不过在查找该表时如果需要把同优先级的编号小于等于当前任务编号的任务屏蔽掉。为了实现方便可以在内核中创建一屏蔽字 OSMapSchedTbl[],如表 2 所示，查表时，先把 OSRdyTbl[y]与 OSMapSchedTbl[当前任务控制块指针.OSTCBX]相与，屏蔽掉同优先级中编号小于等于当前任务的编号的任务。若相与的值不为 0，即表示同优先级中有任务编号大于当前任务编号的任务就绪，则根据相与后的值查表 OSUnMapTbl[]得到值 z,否则直接根据 OSRdyTbl[y]查表 OSUnMapTbl[]得到值 z，再与优先级号 y 组合，便为下一轮转需要调度的任务编号。

表 2　OSMapSchedTbl[]的值

下标	位掩码（二进制）
0	1111 1110
1	1111 1100
2	1111 1000
3	1111 0000
4	1110 0000
5	1100 0000
6	1000 0000
7	0000 0000

例如，当前任务的编号为 26，同时处于就绪态的还有编号为 24、27、29 的任务，即 OSRdyGrp 的值为 0x08，OSRdyTbl[3]的值为 0x2d。

在进行调度时首先通过语句：

y=OSUnMapTbl[OSRdyGrp]

可以得到当前就绪的任务的最高优先级为 3，经比较与当前任务的优先级相同。由于当前任务的同优先级内编号偏移量为 2，接着把 OSRdyTbl [3] 的值与 OSMapSchedTbl[2]相与得结果为 0x28，再查表 OSUnMapTbl[0x28]得 3，故下时间片需要运行的任务编号是(3<<3)+3=27，这样就可以进行任务调度了。而如果此时 OSRdyTbl[3]的值为 0x05，即仅有编号为 24、26 的任务就绪，OSRdyTbl[3]的值与 OSMapSchedTbl[2]相与的结果是 0，这时需根据 OSRdyTbl[3]值查 OSUnMapTbl[]表，得 OSUnMapTbl[0x05]=0，故下一时间片需要运行的任务编号是(3<<3)+0=24。

通过分析，可以知道同优先级的任务可以轮转调度，由于采用了 μC/OS-II 的原有查表形式，仅是将原优先级作为任务编号看待，这样实现的基于优先级的时间片轮转调度算法在源程序基础上改动就很少，实现源码如下：

```
void OS_Sched(void)
{
  #if OS_CRITICAL_METHOD==3
       OS_CPU_SR cpu_sr;
  #endif
  INT8U y;
  INT8U z;
   OS_ENTER_CRITICAL();
  if(OSIntNesting == 0)&&(OSLockNesting == 0)){
      y = OSUnMapTbl[OSRdyGrp];
if(y==OSTCBCur->OSTCBY) {/* 就绪的最高任务优先级改变 */
  z=OSRdyTbl[y]&&SMapSchedTbl[OSTCBCur->OSTCBX];
  if(z! =0) /* 同优先级有更高编号任务就绪 */
   OSPrioHighRdy=(INT8U)((y << 3) + OSUnMapTbl[z]);
  Else
OSPrioHighRdy=(INT8U)((y<<3)+
              OSUnMapTbl[OSRdyTbl[y]]));
  }
  else
OSPrioHighRdy=(INT8U)((y<<3)+ UnMapTbl[OSRdyTbl[y]]);
  if (OSPrioHighRdy! =OSPrioCur){
    OSTCBHighRdy=OSTCBPrioTbl[OSPrioHighRdy];
     OSCtxSwCtr++;
     OS_TASK_SW();
   }
  }
    OS_EXIT_CRITICAL();
  }
```

对中断级调度由函数 OSIntExt()的修改方法与之相同。另外，需要在 OS_CORE.C 文件中建立屏蔽字 OSMapSchedTbl[]，代码如下：

INT8U const OSMapSchedTbl[]={0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00};

可以看出,仅在原任务调度程序的基础上,增加了2个判断条件就实现了简单的基于优先级的时间片轮转调度方式,当然对任务数有了限制,优先级级别有所减少。

如果应用程序中需要的优先级数多于8个,可以再规定某组任务具有不同的优先级,然后在调度程序中再做一次判断,例如,如表3规定:

表3 允许15个优先级的任务编码与优先级对应表

任务编号	优先级
0～7	0～7
8～15	8
16～23	9
24～31	10
32～39	11
40～47	12
48～55	13
56～63	14

在调度程序中,仅需要判断取得的y值是否为0,若为0则直接查找最小编号即最大优先级的任务,不为0做法与上雷同,具体程序不再赘述。

3 结 语

本文通过对原内核的优先级和任务编号重新解释,对任务调度程序作了很小的改动,实现了高优先级可抢占,同优先级简单的轮转调度的算法,但是对具有相同优先级的任务数量做了限制(不能超过8个任务),并且任务的优先级级别数量虽可改变但比原内核的优先级级别数量少。这对于需要具有同优先级的任务,且同优先级任务数量不超过8个的应用还是很有参考价值的。

参考文献

[1]Jean J.Labrosse. 嵌入式实时操作系统μC/OS-II[M]. 邵贝贝等译. 北京航空航天大学出版社,2003

[2]Zhi-hui Xiong,Mao-jun Zhang, Si-kun Li, etc. Virtual Embedded Operating System for Hardware/Software Co-design [J]. ASIC, 2005. ASICON 2005. 6th International Conference On, 2005(10):24~27

[3]Young-Ran Lee, Sang-Young Cho, Jeong-Bae Lee. The Design a Virtual Prototyping Based on ARMulator[J]. Computer and Information Science, 2005. Fourth Annual ACIS International Conference on, 2005(?):387~390

[4]陈开发. 在μC/OS-II中实现同优先级调度的方法[J]. 单片机与嵌入式系统应用,2004(11):79~80

[5]汤子瀛,哲风屏,汤子丹. 计算机操作系统[M]. 西安电子科技大学出版社,2001

An Improved Method of μC/OS-II's Task Scheduler Mechanism

ZHAO Guo-fu , ZHAO Shi-wei

(Network Information Center, Shandong University of Technology, Zibo 255049)

Abstract: The μC/OS-II is a real-time embedded operating system which source code is open and it sustains preemptive machine-made PRI task scheduler method. Through extending μC/OS-II's task scheduler mechanism, makes it sustain setting same PRI task, realizes same PRI task time slice cycle, high priority preemptive task scheduler method. The improved μC/OS-II task scheduler mechanism not only holds its original high real time nature, but also enhances its scheduler method agility, and also makes it simple and efficiency.

Keywords: μC/OS-II; Time Slice Cycle; Task Scheduler