文章编号:1671-7848(2012)S_0-0218-04

uC/OS-II中同优先级任务调度算法研究

李大鹏
(辽宁省科学技术情报研究所办公室,辽宁 沈阳 110180)

摘 要:源代码开放的嵌入式实时操作系统uC/OS-II具有移植简单、使用方便等优点得到了广泛的应用,但由于不支持任务的同优先级调度方式,在针对某些特定的应用(如处理TCP/IP协议)时不仅增加了编程的复杂度,在某些情况下甚至要通过优先级反转的方法规避调度死区,给系统的正常运行带来了隐患。在分析其任务调度机理基础之上,本文介绍了同优先级任务调度的原理,通过为任务控制块变量(OS_TCB)增加上下同优先级OS_ TCB指针的方法,实现了对处于就绪态的同优先级任务间的调度切换,并就关键的步骤进行了解释说明并给出了具体的测试方法。结果证明该方法不仅不破坏uC/OS-II原有的体系结构且代码的改动量较少,更丰富了uC/OS-II的功能,使之能更好的适应工程实际情况的需要。

关 键 词:操作系统;uC/OS-II;优先级;任务调度
中图分类号:TP 27 **文献标识码**:A

Research on Task Scheduler of Equal Priority in uC/OS-II

LI Da-peng
(Institute of Scientific & Technical Information of Liaoning Province, Shenyang 110180, China)

Abstract: uC/OS-II, an embedded real-time operation system with open source, is applied widely because of its convenience and being easy to transplant. If applied on some special situations, such as dealing with TCP/P protocol, the complexity of program will be increased since uC/OS-II doesn't support task scheduler of equal priority. In some case, the method of priority inversion even must be used to avoid schedule dead zone. So based on the analysis of its original mechanism, we add double pointers to the OS_ TCB variable which can link other OS_ TCB variable with same priority. The ready tasks can be switched through these two pointers. On the same time, the key steps and the test methods are discussed in detail. The test results prove that this way not only has the character which doesn't damage the system of uC/OS-II and has less modifiability, but also adds new functions to uC/OS-II so that it can meet the needs of practice more better.

Key words: operating system; uC/OS-II; priority; task scheduler

1 引 言

众所周知,在嵌入式系统开发的早期阶段,由于硬件设备很简单,软件的编程和调试工具也很原始,因此与硬件系统配套的软件都必须从头编写,且程序大都采用宏汇编语言。但近几年随着高性能的手持设备、移动设备和复杂的工业控制装置(例如数控机床和机器人等)的继续出现,人们越来越意识到在嵌入式开发中使用操作系统的必要性,具体体现在以下3个方面:

①操作系统能有效管理越来越复杂的系统资源。②操作系统能够把硬件虚拟化,使得开发人员从繁忙的驱动程序移植和维护中解脱出来。③操作系统能够提供库函数、驱动程序、工具集以及应用程序。

因此,一些商业性质的嵌入式操作系统,如Vxwork, Nucleus和一些免费的嵌入式操作系统,如uc/OS-II, Linux等应运而生并越来越为人所熟知。其中,由于uC/OS_ II具有内核精简,源代码开放等优点,得到了非常广泛的应用,但是,uC/OS_ II完全抢占式的任务调度机制也使得在某些应用场合中略显不便。

2 uC/OS_ II

1)简介 uC/OS最早由Jean J. Labrosse开发,自1992第1版问世以来,已有成千上万的开发者把它成功地应用于各种系统,安全性和稳定性已经得到认证,现已经通过美国FAA认证。目前得到广泛应用的版本是uC/OS_ II 2.86。

uC/OS_ II是一个可裁减、可固化、完全抢占

收稿日期:2012-03-01; **收修定稿日期**:2012-03-22
作者简介:李大鹏(1979-),男,辽宁沈阳人,助理研究员,主要从事科研管理、控制理论与控制工程等方面的工作。

式的内核，它运行最高优先级的就绪任务，每个任务使用自己独立的堆栈。提供了许多系统服务，如邮箱机制，队列机制，信号量机制，固定大小的内存分区以及时间相关的函数等等。并且它最大的特点就是它的源代码开放，这是其他商业实时内核无法比拟的[1]。

2）uC/OS-II 中的任务管理　由于 uC/OS-II 是完全可剥夺型的实时内核，且任务的优先级不允许重复，所以总是运行就绪条件下优先级最高的任务[2]。每个已建立任务的 OS_ TCB 地址均放在数组 OSTCBPrioTbl[ ]的相应位置中并以 OSTCBList 为头组织为双向链表，当进行任务调度时，只要找到处于就绪状态中任务的最高优先级，即可根据下标在 OSTCBPrioTbl[ ]中找到对应任务的 OS_ TCB，并将其赋值给 OSTCBHighRdy，实现任务切换[2]。

3）存在问题　虽然 uC/OS-II 这样的设计可以很好地适应大多数情行，但在实际的应用中，会经常碰到一些特殊情况，它们要求任务的优先级可以重复，举 3 个具有代表性的例子：

①作为模拟量输入设备，要采集多路的温度或湿度，每个任务对应一个传感器，负责对数据的采集、整理、运算和传输，如果为每个任务分配不同的优先级，则不是一个很好设计逻辑，也与真实的情况不完全相符。

②在 TCP/IP 协议栈的处理过程中，常常是为每个连接动态创建一个任务，在一些商业性质的 TCP/IP 协议栈(如 RTIP)中都默认任务的优先级是可以重复的，因此在移植这些协议栈到 uC/OS-II 中，必须人为的跟踪任务的创建和删除情况，这不仅破坏了协议栈原来的封装性，也为编程增加了麻烦。

③为了解决系统运行过程中的优先级反转问题，常常需要将占有资源的低优先级任务的优先级提升，如果任务的优先级不可以重复，则必须做出规定或进行轮询查找，这都会耗费 CPU 的处理时间。

有文献中提出为任务增加时间片的方法，虽然在一定程度上可以克服 uC/OS-II 中完全的抢占式任务调度带来的弊端，但并没有实现同优先级任务调度，正对上面提出的问题，依旧不能很好的解决。

3　同优先级调度实现

为了解决上面提到的问题，在本文中提出一种实现同优先级调度的解决方案，原则是尽可能的减少对系统的改动，尤其是系统与用户的接口部分，使在 uC/OS-II 原来版本中运行正常的程序做最少的改动就可以在新环境中运行。

1）总体思想　对原来 uC/OS-II 中任务调度方法进行分析不难发现，如果每个 OSTCBPrioTbl[ ]中对应的任务块 OS_ TCB 不是一个，而是若干个组成的循环链表；OSRdyGrp 和 OSRdyTbl[ ]每次的改变都对此优先级下所有的任务进行轮循；在任务的删除、创建和优先级更改过程中能正确的对相关变量进行操作，则很容易实现同优先级的任务调度。

为了尽可能地减少对系统的改动，做出如下规定：

如果某个优先级下所有的任务都处于等待状态，则 OSRdyGrp 和 OSRdyTbl[ ]相应的位为 0；如果一个以上(包括一个)的任务处于就绪态，在 OSRdyGrp 和 OSRdyTbl[ ]相应的位为 1，且在 OSTCBPrioTbl[ ]中相应的位置一定保存着指向处于就绪态 OS_ TCB 的地址。以上两点保证了不破坏 uC/OS-II 原有的任务调度机制。

程序执行的流程图，如图 1 所示。

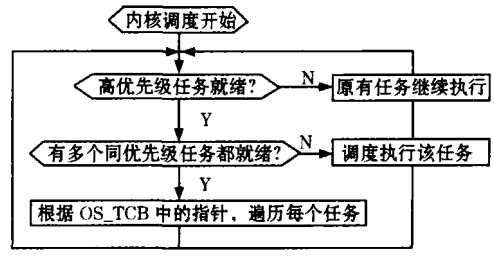

图 1　任务调度过程示意图

2）对 OS_ TCB 的修改　为实现同一优先级下所有任务控制块的双向循环链表结构，必须为 OS_ TCB 结构体定义增加 2 个指针域：

```
struct os_ tcb * OSTCBEquPriPrev;
struct os_ tcb * OSTCBEquPriNext;
```

分别指向同优先级下的其他 OS_ TCB，这样使之形成一个双向循环链表，加上原有的 2 个指针，则每个任务的 OS_ TCB 都在一个线性链表和一个循环链表中切换，如图 2 所示。

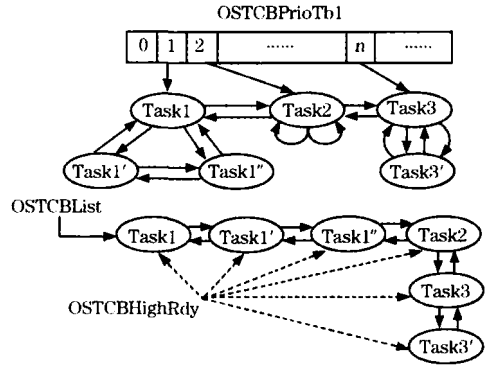

图 2　修改后系统变量间关系示意图

这 2 个指针是实现任务同优先级调度的基础，在任务的建立、删除、状态切换和调度的过程中，

都要涉及到对其的操作。

3）增加一种任务状态和一个变量 uC/OS-II运行时常见的任务状态可以分为两大类，为了方便起见，在等待态中增加一个因为调用OSTimeDly()和OSTimeDlyHMSM()而使任务挂起的标志 - OS_STAT_ TIMEDLY，其地位和OS_ STAT_ SEM等相同且将其加入到OS_ STAT_ PEND_ ANY中，这样处理的目的是当若干个具有相同优先级的任务在因为调用OSTimeDly()而挂起时方便对OSRdyGrp和OSRdyTbl[]的操作。

同时，为简化处理过程，增加一个数组变量ISamePriSwitchFlag[OS_LOWEST_PRIO]，其作用是跟踪记录每某个优先级下任务的个数，如果某个优先级下任务的个数不大于1，则处理的流程与原来相同。

4）增加一个状态处理函数 由于在一个优先级下可能存在多个OS_ TCB，而且经常会对此优先级下的任务控制块进行查询或设置操作，所以统一将其封装成一个函数，根据传入的参数不同完成4个功能：

①得到某优先级下第一个处于OS_ STAT_RDY状态的OS_ TCB地址；②查找对应于某OS_EVENT的OS_ TCB并返回其地址；③得到某优先级下的任务状态，任务状态包括：全部为就绪、部分为就绪和全部为等待；④根据任务的不同ID找到其对应的OS_ TCB并返回其地址。

函数功能虽然非常简单，但对于同优先级调度的实现确起着非常重要的作用，下面对其做进一步的介绍。

函数名称及参数和返回值说明如下：

OS_ TCB * GetCurrentTask(INT8U pri, HANDLE_ITEM item, void * argc)。其中，pri是要操作的任务优先级，函数会根据此参数从OSTCBPrioTbl[]中寻找相应的OS_ TCB链，item是一个枚举值，对应于上面提到的四中操作中的一种，argc是输入输出参数，根据item代表不同的含义。函数的返回值为正确的OS_ TCB地址或NULL。

代码分析：

以得到某优先级下第一个处于OS_ STAT_RDY状态的OS_ TCB地址为例。首先，根据pri变量在OSTCBPrioTbl[]得到相应OS_ TCB的入口地址：OS_ TCB * ptcb = OSTCBPrioTbl[pri]；其次，遍历循环链表，找到第一个状态为OS_ STAT_ RDY的OS_ TCB：

```
while (ptcb->OSTCBStat != OS_STAT_RDY)
ptcb = ptcb->OSTCBEquPriNext;
```

最后，将匹配的OS_ TCB返回：return ptcb。

其余三项功能与之类似，都是对链表进行顺序的查找工作，这里不再详述。

5）增加一个对STCBPrioTbl操作的函数 当某个优先级下存在多个任务时，如果其中某个任务的状态发生改变，必须保证对STCBPrioTbl[]的实时更新，这样才不会破坏原有的调度机制，因此增加了这样一个函数void SetOSTCBPrioTbl(OS_TCB * ptcb)，作用是将第一个处于就绪态的任务块地址放入STCBPrioTbl[]相应的位置。

6）修改函数OS_ Sched OS_ Sched()函数负责uC/OS-II中的任务调度，其原理很简单：当因为中断使某个任务处于就绪态或有任务主动放弃CPU的控制权时，先根据OSRdyGrp利用OSUnMapTbl[]和OSRdyTbl[]找到处于就绪态的任务的最高优先级，然后根据比较情况决定是调用相关函数进行任务切换还是继续执行原来的任务。改为支持同优先级后，在原来的调度原则上新加了同优先级任务间的切换，所以在原来的判断条件后在加一个“或”的条件：当前优先级下任务的个数是否大于1，如果此条件成立，则有可能是在同优先级的任务间进行切换，切换过程照旧。即：

```
if (OSPrioHighRdy != OSPrioCur || SamePriNum[OSPrioCur] > 1)
{ 原有的任务切换函数……}
```

7）修改事件相关函数 在利用信号量、信息队列等来改变任务的运行状态时，最终都会归结为对函数OS_ EventTaskRdy()和OS_ EventTaskWait()的调用。因为这2个函数的执行过程相似，所有这里以EventTaskWait()为例具体进行说明。

```
OS_ENTER_CRITICAL();                                    ①
pri = OSTCBCur->OSTCBPrio;                              ②
if (SamePriSwitchFlag[pri] > 1){
GetCurrentTask (pri, GET_LIST_STATE, (void *)
&argc);                                                 ③
if (argc == ALL_NOT_RDY)
OSRdyTbl[y] &= ~OSTCBCur->OSTCBBitX;                    ④
else
SetOSTCBPrioTbl(OSTCBCur);}                             ⑤
else
OSRdyTbl[y]&= ~OSTCBCur->OSTCBBitX;                     ⑥
OS_EXIT_CRITICAL();                                     ⑦
```

①和⑦的作用是开关中断，保证操作的原子性；

②的作用是取得当前任务块的优先级；

③作用是判断此优先级下所有任务整体状态；

④和⑥的作用是如果此优先级下所有任务都处于等待状态，则修改OSRdyTbl[]的状态值

⑤的作用是调用SetOSTCBPrioTbl函数，将此优先级下处于就绪态的任务块地址放入OSTCBPrioTbl[]中，等待被调度执行。

8）修改函数 OSTimeTick　在函数 OSTimeTick()中，会对所有的任务控制块进行扫描，如果某个 OS_ TCB 的 OSTCBDly 大于零，则对其减一，如果减一后正好等于零，则修改此 OS_ TCB 的状态，同时对 OSRdyGrp 和 OSRdyTbl[ ]进行相应的修改。由于所有任务的 OS_ TCB 均处于以 OSTCBList 为头的链表中，额外要做的工作，就是当 ptcb- > OSTCBDly = =0 时首先进行下述操作：ptcb- > OSTCBStat & = ~OS_ STAT_ TIMEDLY。

9）扩展任务操作函数　在 uC/OS-II 中，由于每个优先级下最多只有一个任务，因此优先级和任务块(OS_TCB)二者是惟一对应的，对任务的相关操作(比如删除和优先级修改等)只要指定相应的优先级即可。其实在 OS_ TCB 的定义中，已经为进一步的扩展做好了准备，这个成员变量就是 OSTCBId，在创建任务时，只要为 OSTCBId 赋予不同的值，即使优先级相同也可以加以区分，因此对 os_ task. c 文件中的函数都做统一的修改：函数参数增加一个 task_ id 变量，这使得原来的由优先级惟一确定某个任务块(OS_TCB)变为优先级 - 任务 ID 二者联合来惟一确定某个任务块(OS_TCB)。确定后就是对链表的具体操作(插入、删除)，这里不再重复。

4　实例测试

本案例的测试环境为：基于 uC/OS-II2.86 版本并支持同优先级任务调度的嵌入式操作系统，编译调试采用的是 Realview，硬件平台为 NXP 公司的 ARM7TDMI 内核处理器 LPC2132，具体分为以下几步：

Step1　建立任务

```
OSTaskCreate(Task1,(void * )0, &Stk1[128],35, 1);
OSTaskCreate(Task2,(void * )0,&Stk2 [128],40, 2);
OSTaskCreate(Task3,(void * )0,&Stk3 [128],40, 3);
OSTaskCreate(Task4,(void * )0,&Stk4 [128],40, 4);
```

Step2　建立信号量和全局变量

```
TaskSem2 = OSSemCreate(0);
TaskSem3 = OSSemCreate(0);
INT32U T1C =0,T2C =0,T3C =0,T4C =0;
```

Step3　任务 1 的处理函数描述　给 T1C 加 1 后，激活 TaskSem2 和 TaskSem3，然后调用 OSTimeDly 休眠 20 ms。

Step4　任务 2，3 的处理函数描述　当被激活时，分别给各自的记数变量(T2C,T3C)加 1。

Step5　任务 4 的处理函数描述　给 *T4C* 加 1 后，调用 OSTimeDly 休眠 60 ms，当 *T4C* 记数到 30 时，修改自身优先级为 40，当 *T4C* 记数到 40 时，删除自身任务。

测试的目的主要是检测任务间的切换是否正确，由任务中延时的时间可知，理论上 Task1，Task2 和 Task3 的调度周期比 Task4 快 3 倍，这通过 *T1C*，*T2C*，*T3C* 和 *T4C*4 个变量的计数值体现。程序经编译后下载到目标平台执行，在 Realview Watch 窗口可以得到当 *T4C* = 30 时 *T1C* = 90，*T2C* = 89，*T3C* = 89，*T1C*，*T2C* 和 *T3C* 与 *T4C* 的比例近似为 3 倍的关系，由于在进行同优先级任务调度时，是以链表的形式依次进行，可能同优先级的 3 个任务都处于就绪态时，Task4 最先得到调度执行，因此 *T4C* 的值会比 *T2C* 和 *T3C* 大 1。据此可说明任务切换正确，再继续运行一段时间，得到一组新值：*T1C* = 441，*T2C* = 441，*T3C* = 441，*T4C* = 40，说明当 Task4 中 T4C 等于 40 时，将自身删除，其他 3 个任务继续正常执行，且 *T1C*，*T2C* 和 *T3C* 相等。由此可以断定，任务的建立、优先级更改和删除正确。至此，可以说明修改后支持同优先级调度的 uC/OS-II 系统运行稳定，任务间切换正常，可以满足一般条件下的使用。

5　结　语

本文基于 uC/OS-II 给出了一种同优先级任务调度的实现方法，但并没有在效率、错误处理方面做更仔细的研究，有兴趣的读者可以在之基础上做进一步的探讨，也可以继续拓展，比如实现同优先级任务间基于时间片的调度等等。

参考文献：

[1] 李宏，马晓军，LwIP 协议在 uc/OS 操作系统中的实现[J]. 信息工程大学学报. 2005,6(2):81-83。

[2] 王海燕，uC/OS II 任务调度的改进与实现[J]，现代电子技术，2006,14(2):41-43.

(上接第 217 页)

[11] Bing Chen, Xiao-Ping Liu, Shao-Cheng Tong, and Chong Lin, observer-Based Stabilization of T-S Fuzzy Systems With Input Delay. *IEEE TRANSACTIONS ON FUZZY SYSTEMS*, 2008, 16(3):652-663.

[12] Fatima El Haoussi El Houssaine Tissir. Robust H_∞ Controller Design for Uncertain Neutral Systems via Dynamic Observer Based Output Feedback. International Journal of Automation and Computing, 2009.6(2):164-170.

[13] 唐功友，雷靖，孙亮. 控制时滞系统基于观测器的最优扰动抑制. 控制理论与应用，2009,26(2):210-215.

[14] 张冬梅，俞立. 线性时滞系统稳定性分析综述. 控制与决策，2008,23(8):842-850.

[15] 杨益群，项国波. 单变量时滞系统优化控制. 武汉理工大学学报，2007,29(5):125-129.

[16] 项国波. 时滞系统的优化控制[M]. 北京：中国电子出版社，2008.4.

[17] Oscar Camacho, Rub'en Rojas, Winston Garc?? a-Gab?? n. Some long time delay sliding mode control approaches. ISA Transactions, 2007.46,95-101.