第35卷 第12期 计 算 机 工 程 2009年6月
Vol.35 No.12 Computer Engineering June 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)12—0266—03 文献标识码：A 中图分类号：TP273

μC/OS-Ⅱ系统中任务调度与监控机制改进

吴永明，罗海据
(广东工业大学机电工程学院，广州 510006)

摘 要：针对μC/OS-Ⅱ系统的任务调度机制局限于只能按照优先级顺序执行，其任务监控机制可能因外界接口环境异常而出现死锁等问题，提出一些改进方法，包括在系统中增加一个负责任务调度的优先级最低的任务，以便灵活控制任务的调度。增加时间限制变量来控制任务的执行时间，以避免任务发生死锁时导致整个系统的崩溃。该系统已应用于一款按摩椅产品的控制系统之中，结果证明这些方法是实用可行的。

关键词：任务调度；任务监控；嵌入式操作系统

Improvement of Task Scheduling and Supervision Mechanism in μC/OS-Ⅱ System

WU Yong-ming, LUO Hai-ju
(Faculty of Electro-mechanical Engineering, Guangdong University of Technology, Guangzhou 510006)

【Abstract】 In the μC/OS-Ⅱ system, the task scheduling mechanism is to execute tasks just according to the tasks' priority, and the task supervision mechanism may happen unexpected dead-lock accidents because of abnormities of interface to outside environment. In order to solve these problems, some improvement methods are proposed. A new task having the lowest priority is added to the system to take charge of task scheduling agilely, and a variable of time limitation is added to the task data structure to control the task running time for avoiding system breakdown when the task dead-lock happens. The improved system is applied to a massage-chair control system successfully. It illustrates the proposed methods are feasible.

【Key words】 task scheduling; task supervision; embedded OS

1 概述

μC/OS-Ⅱ是一个开源的微型嵌入式操作系统，具有多任务、实时性好和容易移植的特点，广泛应用于各种控制系统。在应用 μC/OS-Ⅱ系统开发按摩椅控制系统的过程中，发现μC/OS-Ⅱ系统在任务调度机制和系统的稳定性等方面还存在一些问题，不能完全满足一些实际要求。就其任务调度机制而言，它总是运行当前优先级最高的、已就绪的任务[1]，而在现实的应用中，不同的系统会因特定的要求需要不同的任务调度的方法[2]，例如在按摩椅控制系统里电机和键盘对时间响应的要求是不一样的，因此，增加控制电机任务的执行次数可以提高对电机控制的响应速度。考虑在保留 μC/OS-Ⅱ系统现有功能和优点的情况下增加其他的任务调度方法。另一方面，在嵌入式系统应用环境中，经常需要通过输入输出口和外界进行信息交互，以启动或监控任务的执行。例如，系统中正在执行的一个任务是要读取某一个输入或者等待一个外界信号输入来控制另外一个信号的输出，如果此时出现故障导致外界信号迟迟不能正确输入给系统，那么这个任务就有可能进入死锁状态。在μC/OS-Ⅱ系统中，如果这个任务是系统中优先级最高的任务，那么所有任务都会得不到响应，从而导致整个系统的瘫痪。

本文针对 μC/OS-Ⅱ系统在任务调度机制和监控机制等方面存在的上述问题，研究确定了在任务切换过程中时间消耗最小的切换方式。利用这种切换方式研究了一种简单实用的任务调度方法和任务监控功能，保证了即使某一任务出现死锁的状况而其他任务仍然能照常运行。

2 μC/OS-Ⅱ系统中的任务调度机制

2.1 任务调用的切换过程

μC/OS-Ⅱ的任务依照它们的优先级占据任务就绪表的位置，优先级可以在初始化时确立或者在运行过程中发生改变。μC/OS-Ⅱ的任务执行机制是首先查询任务就绪表，找出优先级最高的任务，然后运行它。如要进行任务切换，即切换到优先级比当前任务低的任务，只有把当前的任务主动挂起后，优先级低的任务才能运行，所以优先级低的任务总是被动地得到运行。在μC/OS-Ⅱ系统中，使优先级高的任务主动停止运行可以采用等待时钟节拍、等待信号量、等待消息或者主动挂起等办法。从这些办法的实现方式的函数来看，等待时钟节拍方式的效率比较慢；而等待信号量、等待消息需要创建多个信号量或消息，消耗系统的内存，效率也不高。相比之下，主动挂起的方式，效率比较高。

2.2 任务调度的实现

由μC/OS-Ⅱ的任务切换特点可知，只有低优先级的任务才能利用发射信号量、消息等机制唤醒高优先级的任务，而高优先级的任务在主动进入等待状态时才能放弃自己对CPU

基金项目：广东省科技计划基金资助项目(2005B50101011)
作者简介：吴永明(1966－)，男，副研究员、博士，主研方向：CIMS/CALS，绿色制造，机电控制系统；罗海据，硕士
收稿日期：2008-12-30 **E-mail：**ymwu@gdut.edu.cn

的控制权，并且高优先级的任务不能调度其他优先级比它低的任务。鉴于此，将整个系统的调度功能安排为在用户的所有任务里优先级最低的任务，当它调用切换函数OSTaskResume()时，能将 CPU 的控制权交付给那些等待信号量或者挂起来的优先级比当前任务高的任务。另一方面，优先级高的任务在扫描一周之后调用 OSTaskSuspend()将自己挂起，释放自己对 CPU 的控制权，因此，所有的任务都具有机会获得 CPU 的控制权。而且对于要求时间响应比较快的任务，可以在调度任务中调用多次以达到要求。

优先级最低的任务调度函数 TaskSwitch 的伪代码如下所示：

```
void TaskSwitch(void *dataptr)
{for(;;)
  {OSTaskResume(OS_PRIO_E1);
   //OS_PRIO_E1 表示任务 E1 的优先级
   OSTaskResume(OS_PRIO_E2);
   //OS_PRIO_E2 表示任务 E2 的优先级
   OSTaskResume(OS_PRIO_E3);
   //OS_PRIO_E3 表示任务 E3 的优先级
   OSTaskResume(OS_PRIO_E4);
   //OS_PRIO_E4 表示任务 E4 的优先级
   …    //省略了其他任务的调用
   OSTaskResume(OS_PRIO_En);
   //OS_PRIO_En 表示任务 En 的优先级
  }
}
```

函数 TaskSwitch 的实现流程如图 1 所示。μC/OS-Ⅱ系统初始化时，除 TaskSwith 任务外，所有的任务都处于挂起状态，当系统运行 OSStart()函数时，首先执行 TaskSwitch 任务。在 TaskSwitch 里用 OSTaskResume(OS_PRIO_E1)唤醒了任务 E1。E1 的优先级比 TaskSwitch 高，因而 E1 获得执行。任务 E1 扫描一周后，调用 OSTaskSuspend(OS_PRIO_SELF)函数将自己挂起，然后系统在就绪表中查找优先级最高的任务进行调度。由于所有优先级的任务都进入等待态或挂起状态，因此 TaskSwitch 又重新获得 CPU 的控制权，把下一个任务 E2 从等待状态激活并进入运行状态。当 E2 扫描一周后自动挂起，TaskSwitch 又获得优先权，把任务 E3 从挂起状态激活进入运行状态。依此执行，直到系统中 En 挂起后，进入下一个循环，E1 又进入运行态，任务的调用机制就是如此循环往复地进行。如果任务之间存在信号量、邮箱等方法的通信而导致任务的挂起或激活，由于 TaskSwitch 处于最低的优先级，并不会对这些操作产生阻碍的作用，因此不影响原来系统的功能。

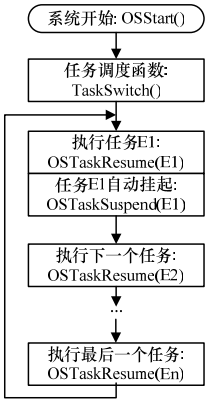

图 1　系统任务的运行与切换机制

由图 1 可知，在一个周期范围内，每个任务至少可以被调用一次，但每个任务被调度的次数又可以是不相同的，可以灵活地对任务进行组合和调用，这样的组合可以根据现实的要求而改变，与原有系统只是依靠优先级来进行任务调度相比，适应面更广。通常情况下，可以把时间响应紧迫的任务进行多次调用以达到任务的实时性要求。例如，在按摩椅控制系统里，键盘的扫描周期可能是 100 ms，而电机的运行控制扫描周期是 1 ms，那么就可以把控制电机的任务进行多次调用，而扫描键盘的任务只调用一次。合理安排任务的调度，可以提高系统的整体性能。

3　系统的任务监控

3.1　任务监控问题的提出

当系统中某一任务出现异常而没有放弃自己对 CPU 的控制权时，会造成整个系统崩溃。目前对这种故障的解决办法是采用“看门狗”技术，但是这种技术会造成整个系统的复位，当一些并非重要任务出现这种故障时会造成不必要的麻烦。为了解决这个问题，改善 μC/OS-Ⅱ系统的任务监控机制，本文改造了原来的任务结构体[1]，在其任务结构体上加上一个以系统节拍为单位的时间限制变量(TimeLimitation)和计数器(TimeLimitationClick)，改进的任务函数结构体代码如下所示：

```
typedef struct os_tcb {
OS_STK * OSTCBStkPtr;
void * OSTCBExtPtr;
…//省略了原来任务结构体的源代码
INT8U    TimeLimitation;          //定义任务运行限制时间变量
INT8U    TimeLimitationClick;
//定义任务运行限制时间计数器
} OS_TCB;
```

如果任务在设定的系统时间节拍内没有主动放弃 CPU 的控制权，那么系统就强制将此任务挂起，而进行其他任务的调度。利用此功能，可以创建一个系统的监控任务，此任务可以将故障代码通知用户，让用户决定复位系统或者进行其他处理。

3.2　任务监控的实现过程

要实现对任务的运行时间进行限制，在任务建立时赋予 TimeLimitation 一个初始值。每个任务根据本身占用的 CPU 时间而不同，其中系统内部的任务 OSTaskIdle 任务则不需要进行监控，可以在初始化时直接赋予 0。值得注意的是，如果 TimeLimitation 限制的时间较短而经常发生超时现象和相应的任务调度，则会影响系统的实时性。所以 TimeLimitation 这个时间限量，应该比任务扫描程序一周用的时间还要长，并且要包括 CPU 进入所有中断处理所用的时间。下面给出对 μC/OS-Ⅱ的分时任务调度方式改造的实现步骤：

首先，在 OSTaskCreate()对 TimeLimitation 赋初始值。

其次，在任务获得 CPU 的控制权之前将 TimeLimitation 的值复制到 TimeLimitationClick。系统每次在进行调度之前，都要在 OSTCBSWHook()函数里更新指向新任务的指针，那么在更新指针后就可以对其进行赋值。如果当前任务要等待系统节拍的延时或者等待一个信号量等消息而放弃 CPU 的控制权，那么系统在进行新任务调度之前要把当前任务的 TimeLimitationClick 清零。这个操作可以在 OSSched()函数里实现。

在系统进行任务调度的地方还有函数 OSIntExit()，在此

函数里也要在系统进行调度前将当前的 TimeLimitationClick 清零，表示不再监控当前任务运行的时间。

对于任务的监控是在系统的后台里操作的，μC/OS-Ⅱ为用户提供了后台操作的接口函数 OSTickISRHook()，主要在这个函数里实现任务监控的改造。首先，为了安全起见，程序先对正在运行的任务检查它的 OSRunning 标志，如果正在运行的任务没有发生其他错误，那么 OSRunning 为真。然后再检查 TimeLimitationClick 是否等于 0，如果等于 0 就直接退出这个程序。如果不为 0，那么将 TimeLimitationClick 进行减 1 操作，再判断是否为 0，如果为 0 就把当前的任务挂起，否则就退出。因为这个子程序是在中断里调用，在进入中断之前已经把中断的总开关关掉，这里就不再设置为临界区。图 2 是 OSTickISRHook()的流程。

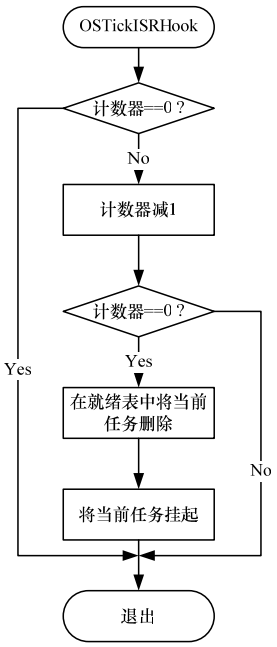

图 2　OSTickISRHook()的流程

4　仿真分析与实际应用

为了验证上述对 μC/OS-Ⅱ系统的任务调度机制和任务监控机制的改造方法，利用开发环境 KEIL 对这个改造的系统进行仿真，如图 3 所示，系统里每个任务运行的状态通过此开发环境可视化地表现出来。图 3 给出了 4 个任务的运行情况和任务调度情况，其中，TaskA, TaskB, TaskC 是具有自我挂起功能的任务，TaskD 是一个出现了死锁的任务，TaskS 是进行任务调度的任务。图中高电平方波表示任务执行一次，TaskA, TaskB, TaskC 由于没有故障，因此能在执行一次后自我挂起，然后由 TaskS 进行其他的任务调度。而 TaskD 出现了故障，不能正确地执行，当它运行的时间超过了自身的 TimeLimitation 时，就被系统强制挂起。从试验的结果来看，改进后的任务运行的情况达到了预期的效果。

图 3　μC/OS-Ⅱ的任务调度和监控机制改善后的试验结果

在对 μC/OS-Ⅱ系统的任务调度机制和任务监控机制进行改造和仿真分析之后，将其实际应用于一款按摩椅产品的控制系统之中，使按摩椅的按摩方式灵活多变，该产品上市之后受到用户的喜爱。

5　结束语

本文针对 μC/OS-Ⅱ系统的任务调度机制和任务监控机制在实际应用过程中所出现的一些问题，提出了一些改进方法。利用耗时最短的任务切换方式的特点，设计了一个负责任务调度的函数，将这个函数作为一种优先级最低的任务加入到系统原有的任务调度之中。这样既确保了系统的原有性能，又增加了系统的任务调度方法，满足了实际应用的需要。针对在嵌入式系统应用的外界环境中有可能发生的异常情况。通过增加时间限制变量和计数器等方式，改善了任务监控功能，可以避免任务发生死锁时导致整个系统崩溃的可能性。通过 KEIL 开发环境运行改进的 μC/OS-Ⅱ系统，用图示化的仿真结果验证了本文所研制出来的改进方法的正确性。这个通过仿真验证的改进方法已成功应用于按摩椅产品的控制系统之中。

参考文献

[1] Labrosse J J. μC/OS-II 源码公开的实时嵌入式操作系统[M]. 邵贝贝, 译. 北京: 中国电力出版社, 2001.

[2] Krishna C M, Shin K G. 实时系统[M]. 戴琼海, 译. 北京: 清华大学出版社, 2004.

编辑　顾逸斐

(上接第 265 页)

台上，通过对 160 GB 数据进行存储测试，系统的存储速度可以达到 190 MB/s。本系统的应用还可扩展到图像处理、雷达、声纳等对数据存储速度有较高要求的领域。

参考文献

[1] 黄　进, 郭立红, 李　岩, 等. 一种高速 CCD 视频实时存储方案的速度分析[J]. 光学技术, 2005, 31(1): 149-150.

[2] 张昆帆, 王　展, 皇甫堪. 高速数据采集和存储[J]. 现代雷达, 2004, 26(4): 14-15.

[3] Corbet J. Linux 设备驱动程序[M]. 魏永明, 耿　岳, 钟书毅, 译. 北京: 中国电力出版社, 2007.

[4] Bovet D P, Cesati M. 深入理解 Linux 内核[M]. 陈莉君, 张琼声, 张宏伟, 译. 北京: 中国电力出版社, 2007.

[5] 马庆军, 舒　嵘. 基于 PXI 平台的高速数据记录系统[J]. 科学技术与工程, 2008, 8(3): 664-665.

编辑　顾姣健