μC/OS-Ⅱ基于动态优先级的时间片轮转任务调度策略

巩思亮[1,2,3]，左秀婷[4]，梁庆伟[1,2,5]，邢　涛[1]

（1. 中国科学院 上海微系统与信息技术研究所 无线传感网与通信重点实验室，上海　200050；
2. 中国科学院 研究生院，北京　100049；3. 公安部 第三研究所，上海　200031；
4. 中国科学院 上海天文台，上海　200030；5. 无锡物联网产业研究院，江苏 无锡　214135）

摘　要：针对 μC/OS-Ⅱ仅支持高优先级独占内核，不支持任务时间片轮转调度的缺陷，提出了一种基于动态优先级方案的时间片轮转任务调度策略。该方案在没有改变内核源代码的前提下，仅在应用层面就能实现任务的时间片轮转调度，具有安全可靠、简单实用的特点。

关键词：嵌入式操作系统；μC/OS-Ⅱ；任务调度；时间片轮转

中图分类号：TN911.7-34；TP393　　**文献标识码**：A　　**文章编号**：1004-373X(2012)15-0123-04

Time slice rotation task scheduling strategy based on dynamic priority

GONG Si-liang[1,2,3], ZUO Xiu-ting[4], LIANG Qing-wei[1,2,5], XING Tao[1]

(1. Key Laboratory of Wireless Sensor Network & Communication, Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Science, Shanghai 200050, China; 2. Graduate School, Chinese Academy of Science, Beijing 100049, China; 3. The Third Research Institute, Ministry of Public Security, Shanghai 200031, China; 4. Shanghai Astronomical Observatory, Chinese Academy of Science, Shanghai 200030, China; 5. Wuxi Sensing Net Industrialization Research Institute, Wuxi 214135, China)

Abstract: To solve the problem that the kernel of μC/OS-Ⅱ does not support the round robin scheduling, a time slice rotation task scheduling strategy based on dynamic priority is proposed. The strategy realizes the task scheduling based on time slice rotation in the application level without changing the kernel source code. The analysis and experiment show that the strategy has the advatages of safety, reliability, simpleness and practicability.

Keywords: embedded operating system; μC/OS-Ⅱ; task scheduling; time slice rotation

0　引　言

在嵌入式系统中使用嵌入式操作系统，可以将简单的单任务单线程应用系统改进为多任务多应用系统，同时大大简化嵌入式系统的设计[1-2]。μC/OS-Ⅱ是由 Jean J. Labrosse 开发的开源的微型嵌入式操作系统，具有结构简练、多任务和实时性好等特点[2]。μC/OS-Ⅱ具有极好的稳定性和可靠性，2000 年 7 月，μC/OS-Ⅱ获得了美国联邦航空管理局(Federal Aviation Administration)对用于商用飞机的符合 RTCA DO-178B 标准的认证，是一种安全级别较高的实时操作系统，另外，μC/OS-Ⅱ可以方便地在各类单片机平台上进行移植[3-4]。因此 μC/OS-Ⅱ在嵌入式系统中得到了广泛的应用[2,5]。但是由于 μC/OS-Ⅱ是一种基于抢占式的内核，当某个优先级较高的任务处于运行状态时，优先级相对低的任务就会一直处于等待状态。假若当前任务由于某种原因导致阻塞，所有任务都会得不到运行。但是在现实的应用场景中，许多系统要求多个任务同时进行。比如在危险信号监测应用中，多个任务同时监测相应的危险信号，由于优先级最高的任务迟迟捕捉不到危险信号，其他任务都会得不到响应，从而导致系统失效，这一点大大限制了 μC/OS-Ⅱ的应用。

针对 μC/OS-Ⅱ的缺陷，国内外许多研究人员都提出了 μC/OS-Ⅱ的改进方案。文献[6]提出了一种增加时间限制变量和计数器，使用最低优先级任务实现高优先级任务调度的方法。优点是可以有效地避免任务发生死锁时导致系统瘫痪的可能性。缺点是将应用层任务限定在具有周期性特点的检测任务中，不具有一般性，另外，需要改进内核，实现相对比较复杂，不适合对

收稿日期：2012-04-15
基金项目：国家重大科技专项(2010ZX03006-002)；国家“九七三”重点基础研究发展计划(2011CB302901)；国家重大科技专项(2009ZX03006-003)；江苏省自然科学基金项目(BK2011035)

μC/OS-Ⅱ内核不熟悉的开发者使用。文献[7]提出的改进 μC/OS-Ⅱ内核使之支持同优先级任务和时间片轮转调度的方案同样存在实现复杂、难以实现的缺点。本文针对上述缺陷,研究确定了一种在应用层实现时间片轮的方案,提出了基于任务层面的时间片轮调度策略,该策略具有独立于内核、实现简单的优点,利于开发人员使用。

1 μC/OS-Ⅱ基于动态优先级的时间片轮调度策略

1.1 多任务实时操作系统任务调度机制分析[2]

在嵌入式实时系统中使用多任务开发应用程序,可以使应用层模块化,并且使 CPU 的利用率达到最高,同时多任务也使应用程序更容易设计和维护。调度是操作系统内核决定由哪个任务进入运行状态的方式和机制,一般来说大多数的嵌入式多任务实时操作系统内核都是基于优先级调度算法,即 CPU 总是执行处于就绪态的优先级最高的任务。为了满足严格的实时性要求,μC/OS-Ⅱ和绝大多数嵌入式实时操作系统内核都是可剥夺性内核,当中断、消息或者信号量等原因导致一个更高优先级任务处于就绪态时,就会"剥夺"当前优先级较低的任务的 CPU 使用权。也就是说基于优先级算法的可剥夺型内核总是运行处于就绪态任务中的优先级最高的任务。

基于优先级算法的可剥夺型内核的优势是具有非常高的实时性,μC/OS-Ⅱ在事件发生时总能保证优先级最高的任务首先获得执行,从而使得任务级的响应时间得以最优化。但是缺点是不方便处理并行的操作,比如系统需要查询式监控串口接收数据,同时需要轮询某个 GPIO 端口,另外还需要从网口不断发送数据,这些操作都需要并行执行,但是如果当前运行的最高优先级的任务没有主动放弃 CPU 时,基于优先级抢占方式的内核无法实现上述操作。这仅是一个简单的例子,在实际情况中遇到的问题会复杂的多,多数较复杂的嵌入式系统应用一般都要求并行操作,因此不支持时间片轮转调度的嵌入式多任务操作系统的应用受到了很大的限制。

以下以 μC/OS-Ⅱ为例,给出了一种基于任务层面动态优先级算法的时间片轮转调度策略,该方法的特点是简单实用,可以不用改变内核源代码,仅在应用层面就能实现任务的时间片轮转调度。

1.2 μC/OS-Ⅱ基于任务层面的时间片轮转调度初步改进

在给出算法之前首先给出如下三个定义:

时间片调度任务组(Group):调度任务组是所有需要以时间片轮转方式运转的任务的集合。

调度任务(SW_TASK):调度任务是一个具有调度功能的任务,他可以调度时间片调度任务组里面的所有任务。要求调度任务的优先级要高于时间片调度任务组中任务的最高优先级,即:

$$\text{prio}(\text{SW\_TASK})>\max(\text{prio}(t_n));\quad t_n\in\text{Group}$$

保留优先级(Prio_preserve):保留优先级是一个介于调度任务的优先级和时间片轮转调度任务组中任务的最高优先级之间的优先级。也就是满足以下条件:

$$\text{prio}(\text{SW\_TASK})<\text{Prio\_preserve}<\max(\text{prio}(t_i));\quad t_i\in\text{Group}\qquad(1)$$

基于动态优先级的时间片轮转策略基本思想如下:

(1) 调度任务(SW_TASK)的优先级高于时间片调度任务组 Group 中所有任务的最高优先级,因此 SW_TASK进入 delay 状态时,系统中优先级次之的任务就会得到执行。

(2) 由于保留优先级 Prio_preserve 满足条件(1),那么调度任务可以通过将时间片调度任务组 Group 中某个任务的优先级设置为 Prio_preserve,当调度任务进入 delay 状态,同时系统中又没有优先级高于 Prio_preserve 的任务时,该任务就可以得到执行。

(3) 调度任务(SW_TASK)可以通过设置 delay 的时间长短来决定让(2)中所述任务执行时间的长短。

(4) 调度任务(SW_TASK)通过循环地改变和复原 Group 中任务的优先级并设置一定时间长度的delay就实现了以时间片方式循环地调度 Group 中的任务。

综上所述,SW_TASK 执行时间片调度的伪代码如下:

```
void SwitchTask(void *pparam)
{
……
do{
    OSTaskChangePrio (TASK_PRIO1,Prio_preserve );
    OSTimeDly(T1);
    OSTaskChangePrio (Prio_preserve, TASK_PRIO1);

    OSTaskChangePrio (TASK_PRIO2,Prio_preserve );
    OSTimeDly(T2);
    OSTaskChangePrio (Prio_preserve, TASK_PRIO2);
        ……
    OSTaskChangePrio (TASK_PRIOn,Prio_preserve );
    OSTimeDly(Tn);
    OSTaskChangePrio (Prio_preserve, TASK_PRIO2);
}while(1)
}
```

通过这种由优先级最高的任务执行任务调度的方式,可以有效地实现对其他任务的时间片轮转调度,使用者不必更改 μC/OS-Ⅱ内核源代码,仅通过在任务层动态改变任务优先级、设置不同的延时时间,就可以循环调度 Group 中任务以时间片方式有条不紊地循环运

行。经过测试，该方案完全可行。但是，由于调度任务仅完成简单的时间片分配和任务切换，这种方案也存在以下两个缺陷：

（1）当某个任务在运行过程中忽然进入延时或等待状态，会将自己的时间片让给其他已经进入就绪状态却没有分到时间片的优先级最高的任务，造成任务分配时间片比例的无法预测；

（2）因为任务时间片轮调度以任务挂起和唤醒为基础，在任务中就不能使用任务挂起唤醒函数对 OSTaskSuspend 和 OSTaskResume，限制了 μC/OS-Ⅱ的功能。

下文将针对这两个问题对方案进行进一步改进。

1.3　μC/OS-Ⅱ基于任务层面的时间片轮转调度进一步改进

为实现任务调度方式的灵活设置和任务所占时间片长度的灵活选择，本文设计了一种独立于任务和 μC/OS-Ⅱ内核的能够统一管理需要时间片轮调度任务的任务调度器（Task_Scheduler）。任务调度器的工作原理与系统架构如图 1 所示。任务调度器的核心是一个任务和时间片查询表（TaskScheduleTable），TaskScheduleTable 中涵盖了 Group 中的所有任务和相应的时间片，每个任务可以动态地选择加入时间片调度任务组 Group，也即添加进 TaskScheduleTable；在 Group 中的任务也可以选择退出。同时，每个已经加入 Group 的任务可以动态改变自己希望分配到的时间片长度。由于在 μC/OS-Ⅱ中，任务的优先级惟一，因此可以使用优先级作为任务的惟一标示。实际上，在 μC/OS-Ⅱ的源代码中也是这样做的。

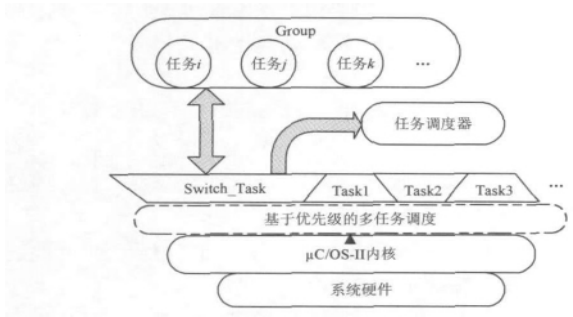

图 1　调度器的工作原理与系统架构

根据上述分析，TaskScheduleTable 中对于每个任务必不可少的两项为任务的优先级和预分配的时间片。也即：

```
Struct TaskScheduleItem{
  ……
  INT8U prio;
  INT8U time_slot;
……
};
```

为便于在 TaskScheduleTable 中进行索引，系统初始化时会对 TaskScheduleTable 进行清零，因为优先级 0 是 μC/OS-Ⅱ系统中最高的优先级，不可能作为优先级比 SW_T 任务优先级还低的时间片轮转任务的优先级，因此，可以通过判断 TaskScheduleItem 的成员 prio 是否为 0 来决定 TaskScheduleTable 中的某项是否为有效项，从而便于任务对于 Group 的加入和删除。SW_TASK 也会根据 TaskScheduleTable 执行任务调度和时间片分配。SW_TASK 的程序代码改进如下：

```
void SwitchTask(void * pparam)
{
……
for(int i;i< Sched_Task_Num;i++)
     //Sched_TASK_NUM 为 TaskScheduleTable 中的可
       用表项数
{
  if(TaskScheduleTable [i]. prio ! =0)
  {
    OSTaskChangePrio (TaskScheduleTable[i]. prio,Prio_
preserve );
    OSTimeDly(TaskScheduleTable[i]. time_slot);
     OSTaskChangePrio (Prio_preserve, TaskScheduleT-
able[i]. prio);
    }
  }
……
}
```

每个任务都可以选择在一开始或者在执行过程中加入到 Group（即添加进 TaskScheduleTable），或者动态改变自己期望分配的时隙长度。这一过程通过函数 Update_SchedTable 实现。Update_SchedTable 的源代码如下：

```
bool Update_SchedTable(int8U prio,INT8U time_slot)
{
……
  for(int i=0;i< Sched_TASK_NUM;i++)
  {
  if(TaskScheduleTable [i]. prio==prio)
  {
  TaskScheduleTable [i]. time_slot=time_slot;
  return true;
  }
}

for(int i=0;i< Sched_TASK_NUM;i++)
{
  if(TaskScheduleTable [i]. prio==0)
  {
    TaskScheduleTable [i]. prio=prio;
    TaskScheduleTable [i]. time_slot=time_slot;
    return true;
    }
  }
return false;
}
```

当任务不希望继续实行时间片轮转方式调度，而是回到抢占式调度方式的时候，可以将自己的优先级从 TaskScheduleTable 中删除。需要注意的是，由于可能出现多任务同时操作 TaskScheduleTable 的情况，需要做好互斥。对于实现互斥的方法可以参见文献[2]，为方便起见，此处简略。

1.4 多种调度方式共存情况分析

采用时间片任务管理调度器之后，大大简化了任务实现时间片轮转调度的设计。同时也为多种调度方式共存提供了可能。在系统设计中就需要考虑多任务多种调度方式共存的情况。

首先给出时间片轮转调度过程中休眠时间片和活跃时间片的定义：

休眠时间片：当选择时间片轮转调度方式的某个任务在其时间片到来时处于挂起状态或者等待状态，称该时间片为休眠时间片。相反，称该时间片为活跃时间片。为简单起见，将任务在其时间片中运行过程时突然挂起或者等待状态也归于休眠时间片的范畴。

当不存在休眠时间片时，就可以将时间片调度任务组 Group 看作是一个任务 GROUP_TASK，该任务的优先级即为调度任务 SW_TASK 的优先级。这样 GROUP_TASK 与其他任务之间仍实行占先式调度，但当 GROUP_TASK 占用 CPU 时，时间片调度任务组 Group 中的任务实行时间片轮转方式调度，即 Group 中的任务轮流占用 CPU。

但是当存在休眠时间片时，情况就变得比较复杂，当 Group 中的某个任务进入休眠时间片，优先级为低于保留优先级 Prio_preserve 的最高优先级的任务会进入运行态。尽管这种方式会使实际的时间片分配无法预测，但是在实际应用中大大提高了 CPU 的效率。

2 测试和验证

为了验证上述基于动态优先级的时间片轮转任务调度策略，在 Microchip 公司生产的 PIC18F97J60 单片机评估板上建立四个不同优先级的任务 $T_1 \sim T_4$，每个任务都是循环向串口打印 ASCII 字符 $i(i='1','2','3','4')$，串口的波特率设置为 115 200。然后将这四个任务加入 Group，设置时间片长度分别为 t，$2t$，$3t$，$4t(t\approx 1\ \text{ms})$，在串口调试助手的显示结果如图 2 所示。

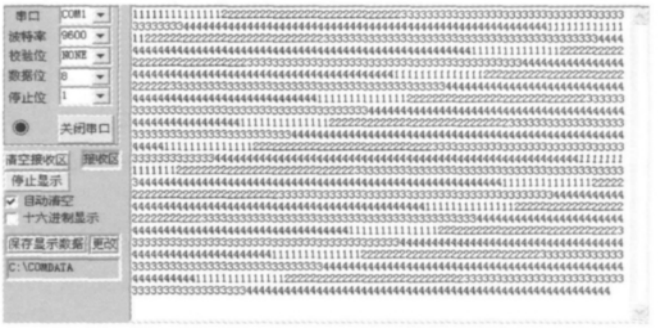

图 2 任务执行过程中的串口助手显示结果

通过实验结果可以看出，字符'1'，'2'，'3'，'4'出现的数量大概为 1∶2∶3∶4。由此证明本文所述的方案是正确的。

3 结 语

本文针对在工程项目中广泛应用的开源微型嵌入式操作系统 μC/OS-Ⅱ仅支持高优先级独占内核，不支持任务时间片轮转调度的缺陷，提出了一种基于动态优先级方案的时间片轮转任务调度策略。使用该方案无需改变内核源代码，仅在应用层面就能实现任务的时间片轮转调度。具有安全可靠、简单实用的特点。

参 考 文 献

[1] 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，2011.

[2] 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.

[3] 鄢仁辉. 嵌入式实时操作系统 μC/OS-Ⅱ的移植实例[J]. 现代电子技术，2011，34(22)：71-74.

[4] 杨立杰，王广生. μC/OS-Ⅱ在 Philips ARM7 上的移植[J]. 现代电子技术，2007，30(10)：77-81.

[5] 冉汉政. 嵌入式实时操作系统 μC/OS-Ⅱ在控制工程中的应用[J]. 现代电子技术，2003，26(13)：84-86.

[6] 吴永明，罗海据. μC/OS-Ⅱ系统中任务调度与监控机制改进[J]. 计算机工程，2009，35(12)：266-268.

[7] 高富强，秦昌硕，游纪元，等. μC/OS-Ⅱ内核扩充时间片轮转调度算法的设计[J]. 计算机应用，2009，29(4)：1128-1130.

作者简介：巩思亮 男，1985 年出生，山东章丘人，博士研究生。主要研究领域为嵌入式系统、无线传感器网络、信任机制。