UC/OS-II 中优先级抢占的时间片调度算法的实现①

Implementation of Round-Robin Based on Preemptive Prior on UC/OS-II

何海涛 (四川理工学院 计算机学院 四川 自贡 643000)

摘 要: UC/OS-II 是一种针对嵌入式设计的实时多任务操作系统，采用基于优先级的抢占式任务调度算法，虽然效率高但调度策略比较单调。在实际应用中，常需要多个任务以时间片轮转方式调度。分析了 UC/OS-II 操作系统内核任务调度算法，在不破坏原有内核机构，保持现有的 UC/OS-II 任务的可运行基础上提出了一种改进 UC/OS-II 内核调度的方法，使其能实现抢占式时间片轮转调度。

关键词: UC/OS-II 实时操作系统 时间片轮转 任务调度

1 引言

UC/OS-II 是源码开放的实时嵌入式操作系统，其主要特点如下：源码开放，系统透明化，容易将操作系统移植到其他平台；采用抢占式优先级调度算法，总是运行就绪条件下优先级最高的任务，这种模式使得系统任务及响应时间得以优化。最高优先级任务一旦就绪，总能得到 CPU 控制权。如果在一个中断处理过程中某优先级高的任务就绪，那么在中断程序完成时，优先级高的任务得以运行；支持多任务：UC/OS-II 可以运行 64 个任务，8 个系统任务和 56 个应用程序任务[1]。同时，UC/OS-II 系统不支持时间片轮转调度法，其任务调度的依据是通过查就绪优先级表，所以每个任务的优先级必须不同，这也限制了系统中的任务个数。在实际的嵌入式系统中(如网络服务程序，多个电机的控制程序等)，系统多个任务地位平等，执行顺序上也没有先后的要求，任务的调度是按照时间片轮转方式进行的。虽然在 UC/OS-II 中通过信号量，事件等机制也可以实现类似时间片轮转调度，但在编写程序时会增加难度，并且对于调度而言也会降低效率。再者，由于任务优先级不能相同，也限制了系统的总任务数。故有必要对 UC/OS-II 的内核做出修改，使其能适应同优先级任务的调度。

2 UC/OS-II内核调度策略分析

UC/OS-II 操作系统中，主要有以下几个任务调度时机：(1)进程状态转换时刻，进程终止、睡眠时；(2)可运行队列中增加一个进程时；(3)内核处理完中断后，由中断返回时。

其任务调度的依据是：根据最高就绪优先级值 OSPrioHighRdy 在任务控制块数组 OSTCBPrioTbl 中找到最高就绪优先级任务 OSTCBHighRdy，然后进行任务切换调度。

为了让任务主动让出 CPU，在没有同步关系的任务中需要使用 OSTimeDly 进行延时以便任务切换。OSTimeDly 通过调用任务切换函数 OS_Sched 切换任务。对于延时任务，其关键的处理在系统时钟中断处理函数 OSTimeTick 中。此函数每个时钟 tick 被调用一次，把延时的任务置为“非就绪”状态，并且对任务延时时间进行计时。而任务的调度并不在 OSTimeTick 中，而是在退出中断后 OSIntExit 进行的。在 OSIntExit 中判断当前任务的优先级和最高就绪任务优先级是否一致，如果一样，表示是同一个任务，不调度。否则，就暂停当前任务而切换到高优先级任务。

① 基金项目:人工智能四川省重点实验室项目(2008RK011)
收稿时间:2009-02-28

3 时间片调度算法实现的条件

时间片调度算法的调度依据是“任务的运行时间”，基本方法是为每个任务分配一个时间片，当任务运行时，对任务进行计时，当任务时间片运行完后进行任务调度，即切换到下一个任务[2]。

如上所述，时间片调度算法的核心是计时，而在 UC/OS-II 系统中的时钟中断处理函数 OSTimeTick 本身就有对任务的计时功能，利用现有的条件，可以对任务进行控制。此外，还需要任务的时间片参数，以及连接多个优先级相同任务的链表。至于同优先级任务的切换，则需要内核调度算法中按照链表轮转切换，而且此调度和任务无关，不需要任务感知，在任务中也不需要进行延时等处理，任务的表现形式通常是一个无限循环。

4 基于优先级抢占的时间片调度实现

首先是对任务控制块 OS_TCB 的修改：为了让任务仍能保持优先级抢占的调度(这也是实时操作系统的基本特征，在绝大多数应用中都需要保留)，增加同优先级任务的双向循环链表结构：OSTimeSchNext，OSTimeSchPrev，及任务剩余运行时间 OSTSCurLen。

因为新加入了时间片轮转调度，允许创建任务时优先级相同，故在初始化任务时，把相同优先级的任务链入到同优先级任务链表中。为统计同优先级任务个数以及判断同优先级是否存在，增加全局数组 SamePrioNum，此数组的大小等于系统配置的最大优先级值。即在本文讨论的方案中，系统所有任务的优先级也限制为 64 级，但每个优先级可以有多个任务。创建任务时，通过修改 SamePrioNum[prio]的值可以方便的得到同优先级任务个数。

任务的时间片通过配置文件 Os_cfg.h 设定。在 OS_Cfg.h 中增加常量 TIMESLICE，其值表示任务运行的时间片。在本文讨论的时间片轮转调度算法中，所有任务的时间片都是相同的。采用这种方式的原因是基于保持 UC/OS-II 的上层函数 OSTaskCreate 接口不变，以保持系统的兼容性和稳定性。在扩展版本 OSTaskCreateExt 函数中，如果没有使用到参数 id，也可以把 id 的值作为时间片，这样，亦可以保持接口不变且可以指定任务各自的时间片。

时间片调度计时的工作就放在 OSTimeTick 中，每个时钟 tick，任务的时间片做相应的调整。其代码如下：

```
if(--OSTCBCur->OSTSCurLen == 0){ //当前任务时间片是否用完
OSTCBCur->OSTSCurLen = TIMESLICE; //使用完后重修赋值时间片
.....}
```

接下来关键的工作是如何让系统能对时间片任务进行轮转调度。因为 UC/OS-II 的调度依据是优先级，而时间片任务的优先级相同，所以必须进行“主动”调度，即调用其任务切换函数 OSIntCtxSw()或者 OS_Sched()。同优先级任务切换的基本条件是：当系统存在多于 1 个的同优先级任务，且任务优先级是当前就绪任务中最高的。

首先通过同优先级循环链表的指针(OSTCBCur->OSTimeSchNext != OSTCBCur)可以知道同优先级任务数是否多于 1 个。若当前任务时间片减 1 后为 0，则表示时间片用完，退出中断时可能需要进行主动调度。退出中断也可能不进行当前同优先级的任务调度，如果在中断期间有更高优先级任务就绪，那么系统应该暂停当前优先级任务而去调度高优先级任务。

为进行主动调度，引入全局标志 samePrioSche：表示是否有同优先级任务调度的需要。若 samePrioSche 为 1，表示当前优先级任务有多个，并且当前任务时间片已耗尽，需要切换到下一个同优先级任务。

下面是对退出中断 OSIntExit 函数进行修改，以使其能对同优先级进行调度：

```
if (OSPrioHighRdy != OSPrioCur) {
//原系统的代码不需要做任何修改，当有高优先级任务就绪时进行优先级抢占调度 .....
}else{ //如果是同优先级，可能需要调度
if(samePrioSche--){ // 如果需要进行同优先级任务切换
OSIntCtxSw(); // 主动任务调度
}}
```

要能在进行同优先级任务调度，还有一步关键的工作：设置设置当前任务控制块指针 OSTCBCur 和就绪的最高优先级任务控制块 OSTCBHighRdy 指针为欲调度的任务，即同优先级链表中的下一节点：

```
OSTCBPrioTbl[OSTCBCur->OSTCBPrio]=OSTCBCur ->OSTimeSchNext;
```

这是任务调度的关键之处，因为在系统最底层的汇编代码中，调度的方式是对 OSTCBHighRdy 进行操作的(MOV AL, BYTE PTR DS:_OSPrioHighRdy)，而原系统优先级抢占调度方式中，OSTCBHighRdy 是通过查表自动得到的，在同优先级任务调度时，由于没有查表的操作(同优先级任务存在在链表中，没有存放在原系统的任务表中)，在此需要进行手动设置，否则将不能进行任务的切换。这部分工作放在 OSTimeTick 函数中，当需要进行同优先级任务切换时进行。

至此，同优先级下一个任务已经就绪，等待退出时钟中断后系统进行调度。调度发生的时机和原系统一样：在退出中断后 OSIntExit 函数中进行。如果当前任务优先级和就绪的最高优先级任务不一致，则系统按照优先级抢占方式进行高优先级调度；若一致，则检查同优先级调度标志 samePrioSche，若为 1，则进行主动任务调度 OSIntCtxSw()；对于非中断的任务切换函数 OS_Sched，也做相同的处理。这样的处理方式，即保证了优先级抢占方式，也实现了同优先级任务的时间片轮转。

5 系统的调试验证及性能分析

UC/OS-II 系统本身可以在 DOS 系统下运行，但在 Win32 的保护模式下无法进行调试。为方便进行调试验证，使用虚拟机安装 Windows98 系统，在其纯 DOS 模式下使用 BC45 进行调试。调试时，创建 3 个任务，一个高优先级任务 TaskStart：

OSTaskCreate(TaskStart,(void*)0,&TaskStartStk[TASK_STK_SIZE - 1], 0);此任务获取键盘输入以结束程序，任务中需要调用延时函数 OSTimeDly，否则根据优先级抢占原则，其他任务不能得到调度。另外创建 2 个优先级都为 1 的任务 Task1 和 Task2：

```
OSTaskCreate(Task1, (void *)&TaskData[1],
&TaskStk[1][TASK_STK_SIZE - 1],1);
OSTaskCreate(Task2, (void *)&TaskData[2],
&TaskStk[2][TASK_STK_SIZE - 1],1);
```

创建任务接口和原 UC/OS-II 系统一致，保证了兼容性和稳定性。2 个测试任务的代码如下：

```
void Task1 (void *data)
{
for(;;)
PC_DispStr(1,  2,"*task1*",
DISP_FGND_WHITE);
}
void Task2 (void *data)
{
for(;;)
    PC_DispStr(1,  2,  "#task2#",
DISP_FGND_WHITE);
}
```

即在同一位置显示不同的字符串。在仿真环境下运行，可见交替显示字符串，通过配置 TIMESLICE 为不同的值，可以改变字符串显示的时间。

同样，把系统代码移植到 ARM 系统上，使用多个同优先级任务控制 GPIO 端口的 LED 灯，经测试正常；而原来基于 UC/OS-II 系统编写的任务移植到新系统上不用做任何修改也可以正常运行。

本文中对系统核心代码的修改仅限于 OSTime Tick 和 OSIntExit 等少数函数，其中 OSIntExit 最大运行时间仅仅增加了 30 个时钟周期左右，而关中断时间也小于系统内核中的最长关中断函数时间，对系统的实时响应特性没有影响。

6 结语

本文在 UC/OS-II 的优先级抢占式调度基础上添加了同优先级的时间片轮转调度策略，和其他的修改内核以改进调度算法的方法相比[3]，本文中的做法最大程度上保持了操作系统上层接口的一致性，原来在 UC/OS-II 上运行的应用任务可以不做任何修改即能运行在新系统之上，并且可以在原系统上添加更多的同优先级任务，使得 UC/OS-II 系统不再受到 64 个任务的限制，在实际应用中也不必在为相同重要程度的任务安排优先级而困扰，从而减少因多任务制约而带来的低效率编程复杂度。

参考文献

1 Labrosse JJ. uC/OS The Real-Time Kernel. Kansas: R & D Publication Inc, 1992.

2 Tanenbaum AS, Woodhull AS. Operating Systems Design and Implementation. NJ, Englewood Cliffs: Prentice-Hall International Inc, 1997.

3 成后发,杨春金.UC/OS-II 操作系统内核的改进.通讯和计算机, 2006,6(3):52－54.