第33卷 第16期 计 算 机 工 程 2007年8月
Vol.33 No.16 Computer Engineering August 2007
·开发研究与设计技术· 文章编号：1000—3428(2007)16—0246—02 文献标识码：A 中图分类号：TP316

实时嵌入式操作系统μC/OS-II内核的分析与改进

季 虹，付少锋，车向泉，周利华
（西安电子科技大学多媒体研究所，西安 710071）

摘 要：基于源码公开的实时嵌入式操作系统μC/OS-II及对内核的分析，该文对μC/OS-II的调度算法提出了改进，即扩充了任务数目，采用了任务分类的方法，使其能支持多于64个任务的调度。并可根据实际要求，对任务分组采用不同的调度算法，同时在整体上保持优先级调度模式。该文给出了局部时间片轮转调度和优先级调度算法的实现。

关键词：嵌入式操作系统；实时；多任务；调度算法

Analysis and Improvement of Real-time Embedded Operating System μC/OS-II Kernel

JI Hong, FU Shao-feng, CHE Xiang-quan, ZHOU Li-hua
（Multimedia Technology Institute, Xi'an Electronic & Technology University, Xi'an 710071）

【Abstract】 Based on the analysis of a real-time embedded operating system μC/OS-II which is open source, some improvements about schedule mechanism are put forward to support more than 64 scheduling tasks. Divided into different groups, different schedule algorithms can be adopted according to the requirements of practical application. Implementation of the round-robin scheduling and priority schedule in local are put forward. Meanwhile, priority-based schedule algorithm is kept in the whole.

【Key words】 embedded operating system; real-time; multi-task; schedule algorithm

随着嵌入式操作系统μC/OS-II在航天、医疗、科研等方面的广泛应用，其易移植性、高实时性、可裁剪等优点得到了日益广泛的肯定。但相比其他商业嵌入式操作系统，其在任务调度个数和调度机制上还有不足。在许多实际的应用中，任务个数都超过64个，且需要对任务分组采用不同的调度方法。

1 系统内核的分析

μC/OS-II的大部分代码是用ANSI C编写的，只有与处理器硬件相关的一部分代码用汇编语言编写，且与CPU相关的代码都放在与Os_cpu.h、Os_cpu_c.h、Os_cpu_a.s这3个文件中，当进行移植的时候，只需对这3个文件进行相应改动，而其他与CPU无关的代码则可直接编译、链接，这样，移植的工作量相对减少，工作相对集中。所以，μC/OS-II的移植性较好，且可以在绝大多数8位，16位和32位处理器上运行，用户可以根据实际硬件环境自己编写移植部分的代码。由于对操作系统内核的修改主要集中在进程调度方面，因此本文结合μC/OS-II操作系统的特点，对内核调度机制作了如下分析：

(1)μC/OS-II的调度

调度是指在有限的处理单元上对具有某些已知特征的任务集执行顺序的设计。μC/OS-II的任务调度是按抢占式多任务系统设计的，即它总是将处理机分配给处于就绪条件下优先级最高的任务，并允许调度程序根据某个原则，去打断某个正在执行的进程，将已分配给进程的处理机重新分配给另一个进程。例如中断服务程序可以抢占CPU，直到中断服务完成；其后，内核将让此时优先级处于最高的任务运行(不一定是那个被中断了的任务)；这样使任务级系统响应时间得到了最优化，同时是可知的。内核调度过程如图1所示。

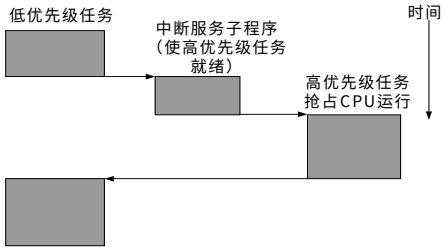

图1 内核调度

(2)μC/OS-II任务调度的实现

系统通过2种方法进行任务调度：1)时钟中断或其他硬件中断时，系统调用函数OSIntCtxSw( )执行任务切换功能；2)任务主动进入挂起或等待状态时，系统通过发软中断命令或依靠处理器执行陷阱指令来完成任务切换，中断服务例程或陷阱处理程序的向量地址必须指向函数OSCtxSw( )。

2 系统内核的改进

2.1 问题的提出

在μC/OS-II目前的版本中，系统最多可以管理64个任务，且保留了优先级标号为0,1,2,3,OS_LOWEST_PRIO-3,OS_LOWEST_PRIO-2,OS_LOWEST_PRIO-1,OS_LOWEST_ RIO的任务，因此，用户仅能同时拥有56个任务。这在日益变化

作者简介：季 虹(1983－)，女，硕士研究生，主研方向：计算机系统结构，嵌入式系统设计；付少锋，工程师；车向泉，高级工程师；周利华，教授

收稿日期：2006-08-21 **E-mail：**jihong@mti.xidian.edu.cn

的实际应用中是不充分的。

uCOS-II 规定所有任务的优先级必须不同，任务的优先级唯一地标识了该任务。即使 2 个任务的重要性是相同的，它们也必须有优先级上的差异，这是实现实时调度的必要条件。但对于一些实时性要求不高的普通服务进程，希望将其归在一组，与其他任务相区别时存在优先级差异，而在组内希望其轮流执行，因此，需要区分不同的任务类型以指定(分配)相应的调度策略。

2.2 问题的解决

为能较好地解决 2.1 节提出的问题，笔者采用多队列调度，即根据任务性质和类型的不同，将就绪队列再分为若干个子队列，所有的任务按其性质排入相应的队列中，而不同的就绪队列采用不同的调度算法。这里对任务分组后将原先的 64 个任务扩展为 64 个任务集合，每个集合可扩展 0 或多个任务，用户建立任务时可以将紧迫程度类似的任务或是需要采用相同调度策略的任务放在一组，根据分组定制任务调度策略，同时保留组间优先级差异。这里仅实现组内时间片轮转和优先级调度机制，以后还可根据实际需要扩展其他调度机制。

2.2.1 数据结构的改动

新建数组 SchedM[64]标识各优先级组采用的调度策略，为今后扩充，数组各元素为枚举类型，数值为 1/0 代表对应的组/优先级调度策略。

(1)TCB 控制块中增加分量。OSTCBHPrio 为同优先级组任务的 ID 号，用以区别各任务控制块，同时作为实时任务调度的组内优先级号；Counter 标示同优先级组所剩时间片大小，用于组内时间片调度的时间片分量。

(2)利用扩展分量 struct os_tcb *OSTCBExtPtr 建立同优先级组中 TCB 块之间的单向链表。

2.2.2 改进后的调度机制的实现

μC/OS-II 操作系统中，主要有以下几个进程调度时机：

(1)进程状态转换的时刻，进程终止，进程睡眠；

(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。

在前 2 种情况下，系统通过调用函数 OS_Sched( )实现进程调度；在第 3 中情况时，系统通过调用函数 OSIntExit( )实现进程调度。这 2 个函数在选择当前优先级最高的进程时采用了非常类似的方法，应对这 2 个函数中的进程调度算法也作类似修改，以 OS_Sched( )函数为例描述：

```
void   OS_Sched(void){
//从就绪表中找出当前优先级最高的组 ;对不同的调度策略采取
//相应的调度算法，选取其中需要调度的进程，保存指向该任务控制
//块的指针；
…
ostcbh = OSTCBPrioTbl[OSPrioHighRdy];
ostcb = OSTCBPrioTbl[OSPrioHighRdy];
maxid=0;
switch(SchedM[OSPrioHighRdy]) {
case SCHED_RT:
while (ostcb->OSTCBExtPtr != (OS_TCB *)0) {
if (ostcb->OSTCBHPrio > maxid) {
maxid=ostcb->OSTCBHPrio;
Postcb=ostcb;
}
ostcb = ostcb->OSTCBExtPtr;
}
if (ostcb->OSTCBHPrio > maxid) {
maxid=ostcb->OSTCBHPrio;
Postcb=ostcb;
}
//处理链表尾;
…
//根据实际确定是否进行任务切换;若必要将 Postcb 赋给
//OSTCBHighRdy;
case SCHED_OTHER:
//以权值 Counter 作为判断依据，算法与上同;
…
//如果可运行进程的时间片都用完了，则给它们的 Counter 重新
//赋值，并选取队首任务执行;
if (maxid==0) {
while (ostcb->OSTCBExtPtr != (OS_TCB *)0) {
ostcb->Counter= SchedT[OSPrioHighRdy];
                ostcb = ostcb->OSTCBExtPtr;
                }
ostcb->Counter=SchedT[OSPrioHighRdy];
                OSTCBHighRdy=ostcbh;
            }
…
}
```

同原来的调度机制相比，OS_Sched( )首先搜索就绪表，确定就绪表中优先级最高的任务组的 Prio。若此时该优先级位的任务组存在≥2 个任务，则对该组进行组内的调度，调度策略由用户定制。对于实时任务组，根据分量 OSTCBHPrio 找出优先级最高的进程运行；对于非实时任务组，采用时间片轮转法调度算法获取其中时间片 Counter 的值的任务，并且使其获得 CPU 使用权。如果在遍历过程中发现所有任务的时间片都已归 0，再将它们赋一定的初值，设置数组 SchedT[]存放不同优先级组初始分配的时间片大小，用户可以根据优先级组号定制时间片。

时间片轮转法算法实现的介绍：为使任务组采用局部时间片轮转的任务调度算法，TCB 控制块中增加了一项 Counter 作为时间片权值，同时调用时钟中断服务程序完成对每个任务的时间片更新的功能。本文对时钟中断服务函数改进如下：在中断处理函数中，将当前运行任务的 Counter 值减 1，并且在时钟中断返回的时候，调用 OSIntExit( )对任务进行调度。以下是时钟节拍中断服务程序伪代码：

```
时钟节拍中断服务子程序示意代码—OSTickISR( )
void OSTickISR(void)
{
  保存处理器寄存器的值;
调用 OSIntEnter( )或是将 OSIntNesting 加 1;
调用 OSTimeTick( );
调用 OSIntExit( );
恢复处理器寄存器的值;
执行中断返回指令;
}
时钟节拍函数的一个节拍服务——OSTimeTick( )
void OSTimeTick(void)
```

(下转第 250 页)