μC/OS-II实时操作系统中任务延时的研究与改进

范胜[1], 殷志祥[2]

(1. 安徽理工大学 电气与信息学院, 淮南 232001; 2. 安徽理工大学 理学院)

摘要: 通过介绍μC/OS-II实时操作系统中的任务延时功能, 分析了系统中的任务延时的优缺点。针对任务延时占用处理器时间与任务总数有关和扫描各个任务占用大量处理器时间的问题, 对操作系统中的任务延时机制进行改进。改进后的操作系统在基于第二代Cortex-M3内核的LPC1768处理器上测试, 通过软件仿真, 得出此方法可以提高系统的实时性, 降低系统的额外开销。

关键词: μC/OS-II; 任务延时; Cortex-M3; 软件仿真

中图分类号: TP311.1　　文献标识码: A

Research and Improvement on Task Delay of μC/OS-II

Fan Sheng[1], Yin Zhixiang[2]

(1. College of Electric and Information Engineering, Anhui University of Science and Technology, Huainan 232001, China;
2. College of Science, Anhui University of Science and Technology)

Abstract: By introducing the function of the task delay in μC/OS-II real time system, task delays' strengths and weaknesses in the system are analyzed. For task delay has taken to processor time about the total numbers of task problems and scanned the task has significant processor time problems. The task delay mechanism in the operating systems is improved. Improved system is tested on the second generation Cortex-M3 LPC1768 processor. Software simulation shows that this method can improve the system real time performance and lower the overhead costs.

Key words: μC/OS-II; task delay; Cortex-M3; software simulation

引　言

μC/OS-II是一种源代码公开、结构小巧、具有可剥夺实时内核的实时操作系统[1]。绝大部分代码是用C语言编写的, 便于移植到各种内核上。μC/OS-II使用时钟节拍完成任务的延时功能, 每个时钟对所有的任务控制块进行扫描。时钟节拍率越高, 系统的额外负荷就越重, 而且会随着任务总数的增加而增加。

本文详细分析μC/OS-II中的任务延时功能, 对任务延时作适当改进。新创建一个任务延时链表, 把需要延时的任务链接到延时列表中, 这样每个时钟节拍只对延时任务的控制块进行扫描即可, 由此降低了系统负荷, 而且系统的开销不会随着任务总数的增加而增加, 而仅仅与同时延时的任务数有关。

1　μC/OS-II任务延时

μC/OS-II系统中任务延时是时间管理功能的主要部分, 而在μC/OS-II 2.81及以后的版本中, 增加了软件定时器功能。不管是任务延时还是软件定时器, 都需要一个硬件产生一个周期中断, 也就是时钟节拍。

μC/OS-II系统的时钟节拍的频率一般在10~100 Hz之间, 时钟节拍率越高, 系统的额外负荷就越重, 影响系统的实时性。任务延时是在时钟节拍中断函数中实现的, 时钟节拍中断函数调用时钟节拍函数OSTimeTick(), 此函数的工作主要是扫描每一个任务控制块中的时间延时项OSTCBDly, 完成任务的延时。由于OSTimeTick()要对每个任务都进行一遍同样的工作, 因此它的运行时间和任务数的多少成正比[2]。如果任务数比较多的话(现在μC/OS-II可以支持256个任务, 而基于Cortex-M3内核可以支持1024个任务), 函数OSTimeTick()占用大量的系统时间。

本文使用的操作系统版本为μC/OS-II 2.86, 此版本中与任务延时相关的函数包括:

①与任务延时设置相关的函数——任务延时函数OSTimeDly()与OSTimeDlyHMSM(), 位于time.c文件中, 用于任务自身调用, 无条件的挂起自己延时一段时间;

© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

请求资源函数 OS_FlagBlock()(请求事件标志)、OSMboxPend()(请求邮箱)、OSMutexPend()(请求互斥量)、OSQPend()(请求消息队列)、OSSemPend()(请求信号量)等,当资源请求不成功时,任务延时挂起;还有一个挂起其他任务的函数 OSTaskResume(),但没有挂起其他函数一段时间的函数。

②与任务恢复有关的函数——恢复任务延时函数 OSTimeDlyResume(),恢复因资源不满足而挂起任务的函数 OS_EventTaskRdy()、OS_FlagTaskRdy(),把等待列表中的占用位清除,清任务延时值;任务删除函数 OSTaskDel()、任务恢复函数 OSTaskResume()等。

③时钟节拍处理函数 OSTimeTick(),用于处理任务延时[3]。

2 任务延时的改进

首先在 uCOS_II.H 头文件中定义任务延时链表 OSTCBDlyList,延时任务总数变量 OSTCBDlyNum,由于记录延时任务数。设置任务延时,首先把任务从任务链表中删除,然后加入都任务延时链表 OSTCBDlyList 中,最后 OSTCBDlyNum 加 1;延时结束或任务恢复时,任务控制块从任务延时链表中删除,加入任务链表;删除任务时,首先判断任务是否处于延时中,再决定从哪个链表中删除。

2.1 修改与任务延时设置相关的函数

与任务延时设置相关函数中,在其代码 OSTCBCur->OSTCBDly= ticks(或 timeout)后面加入延时设定函数函数 OSTCBDlySet(),如 OSTimeDly()函数修改成如下形式:

```
void OSTimeDly (INT32U ticks) reentrant{
  if (ticks > 0) {
    ……              //参考 μC/OS-II 2.86版的源代码
    OSTCBCur->OSTCBDly= ticks;
    OSTCBDlySet(OSTCBCur);
                    //把当前任务加入任务延时链表
    ……
  }
}
```

函数 OSTCBDlySet()的形参为需要任务延时的控制块,函数的伪代码如下所示:

```
void OSTCBDlySet( OS_TCB * ptcb){
  ……                //参数检测
  //从任务链表 OSTCBList 中删除任务控制块
  if (ptcb->OSTCBPrev= = (OS_TCB * )0) {
    ptcb->OSTCBNext->OSTCBPrev= (OS_TCB * )0;
    OSTCBList= ptcb->OSTCBNext;
  }
  else {
    ptcb->OSTCBPrev->OSTCBNext= ptcb->OSTCBNext;
    ptcb->OSTCBNext->OSTCBPrev= ptcb->OSTCBPrev;
  }
  //任务控制块加入任务延时链表 OSTCBDlyList 中
  ptcb->OSTCBNext= OSTCBDlyList;
  ptcb->OSTCBPrev= (OS_TCB * )0;
  if (OSTCBDlyList ! = (OS_TCB * )0) {
    OSTCBDlyList->OSTCBPrev= ptcb;
  }
  OSTCBDlyList= ptcb;
  OSTCBDlyNum+ + ;   //延时任务数加 1
}
```

2.2 从任务延时链表中删除任务控制块

当任务延时完成、任务延时恢复、任务恢复、任务删除时,需要把任务从任务延时链表中删除。使用函数 OSTCBDlyDel(),此函数无返回,形参有两个:第一个为任务控制块指针,第二个为操作类型 opt,函数原型为 void OSTCBDlyDel(OS_TCB * ptcb, INT8U opt)。opt 的取值为:OS_Dly_OPT_NONE(正常执行,从任务延时链表删除,加入任务链表),OS_Dly_OPT_DEL(把任务从任务延时链表删除,OSTaskDel()函数使用)。函数 OSTCBDlyDel 的流程如图 1 所示。

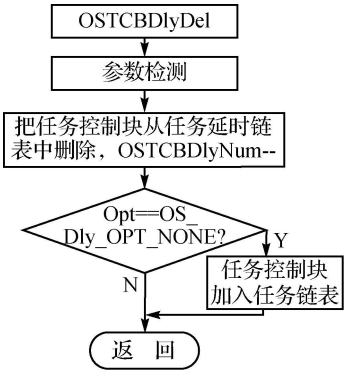

图 1 OSTCBDlyDel()流程

在函数 OSTimeDlyResume()、OS_EventTaskRdy()、OS_FlagTaskRdy()以及 OSTaskResume()中,在其代码 ptcb->OSTCBDly= 0 后面(下一行)加入 OSTCBDlyDel(ptcb, OS_Dly_OPT_NONE)。然后在时钟节拍处理函数 OSTimeTick()中作适当修改,修改后的伪代码如下:

```
void OSTimeTick (void) {
  ……                  //与原程序相同
  ptcb= OSTCBDlyList;  //修改为扫描任务延时链表
  for( i= 0; i< OSTCBDlyNum; i+ + ){
                       //使用 OSTCBDlyNum 控制循环次数
    OS_ENTER_CRITICAL();
    if (ptcb->OSTCBDly ! = 0) {
      if (--ptcb->OSTCBDly= = 0) {
        OSTCBDlyDel(ptcb, OS_Dly_OPT_NONE);
        ……            //与原程序相同
      }
    }
    ptcb= ptcb->OSTCBNext;
```


© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
    OS_EXIT_CRITICAL();
  }
}
```

对于任务删除函数 OSTCBDlyDel()，首先判断任务控制块中的任务延时值是否为0，如果不等于0，调用函数 OSTCBDlyDel(ptcb, OS_Dly_OPT_DEL)，把任务从任务延时链表中删除；如果等于0，把任务从任务链表中删除。最后，把任务控制块回收到空闲任务链表中。

3 实验测试

3.1 测试环境

本次实验使用软件开发环境 Keil4，把 μC/OS-II 操作系统移植到基于 Cortex-M3 内核的 LPC1768 处理器上。对实时操作系统 μC/OS-II 2.86 进行改进，并对改进后的操作系统进行软件仿真测试。

ARM Cortex-M3 内核采用3级流水线和哈佛结构，带独立的本地指令和数据总线以及用于外设的稍微低性能的第三条总线，还包含一个支持随机跳转的内部预取指单元。LPC1700 系列微控制器主要用于处理要求高度集成和低功耗的嵌入式应用，最高工作频率可达 100 MHz。内部有高达 512 KB 的 Flash 存储器、64 KB 的数据存储器，片内外设包括以太网 MAC、USB 主机/从机/OTG 接口、8 通道的通用 DMA 控制器、4 个 UART、2 条 CAN 通道、2 个 SSP 控制器、SPI 接口、3 个 I^2C 接口、2 输入和 2 输出的 I^2S 接口、8 通道的 12 位 ADC、10 位 DAC、电机控制 PWM、正交编码器接口、4 个通用定时器、6 输出的通用 PWM、带独立电池供电的超低功耗 RTC 等众多功能，方便系统的开发，节约成本[4]。

本次测试为：在 μC/OS-II 中创建 25 个用户任务，其中的 10 个任务延时一段时间(for 循环延时)，并发送信号量，已激活等待此信号量的任务，而后调用函数 OSTimeDly()，任务延时。而另外 15 个任务无限期等待另外几个任务的信号量。Keil4 的软件仿真中，LPC1768 的外部时钟设定为 22.1184 MHz。

3.2 实验结果

下面通过 Keil4 软件仿真中的 Performance Analyzer 功能[5]，观察 μC/OS-II 原操作系统与改进后的操作中函数 OSTimeTick() 的运行时间。性能测评图如图 2、图 3 所示。

可以看出，改进后 OSTimeTick() 函数的系统占用率只有原先的 50%，当然在实际使用环境中，改进后 OSTimeTick() 函数的系统占用率与系统中延时任务的多少、系统的运行速度等因素有关，不可能降低这么多，至少从理论上与仿真测试中可以验证此方法可以降低系统的额外开销，提高系统的实时性。

Performance Analyzer

Reset　Show: Functions

Module/Function	Calls	Time(Sec)	Time(%)
OSTimeTick	5244	53.751 ms	2%
SysTick_Handler	5244	44.574 ms	1%
os_cpu_a		31.785 ms	1%
OSIntExit	5244	22.287 ms	1%
OSSemPost	1048	13.100 ms	0%

图 2　原操作系统中函数 OSTimeTick() 的运行情况图

Performance Analyzer

Reset　Show: Functions

Module/Function	Calls	Time(Sec)	Time(%)
OSTimeTick	5071	51.978 ms	1%
SysTick_Handler	5071	43.104 ms	1%
os_cpu_a		30.740 ms	0%
OSIntExit	5071	21.552 ms	0%
OSSemPost	1014	12.675 ms	0%

图 3　改进后操作系统中函数 OSTimeTick() 的运行情况图

结　语

随着技术的进步，处理器芯片的内存不断增大，运行速度不断提高，而且应用系统设计越来越复杂，系统需要运行越来越多的任务，时钟节拍处理函数将占用大量的处理器时间，影响系统的实时性。通过本文对原操作系统中任务延时机制的改进与优化，使时钟节拍处理函数的运行时间仅与同时延时的任务数有关。通过 Keil4 开发环境下的软件仿真可以看出，改进后系统开销大大降低。

参考文献

[1] 任哲. 嵌入式实时操作系统 μC/OS-II 原理及应用[M]. 北京：北京航空航天大学出版社，2005.

[2] Jean J. Labrosse. MicroCOS-II The Real-Time Kernel Second Edition[M]. USA: CMP Media LLC, 2002.

[3] μC/OS-II Company Limited. μC/OS-II 2.86, 2009.

[4] NXP Semiconductors Limited. LPC17xx User manual, 2010-04.

[5] 陈瑶，李佳，宋宝华. Cortex-M3+μC/OS-II 嵌入式系统开发入门与应用[M]. 北京：人民邮电出版社，2010.

范胜(硕士生)，主要研究方向为智能控制、嵌入式系统设计；殷志祥(教授)，主要研究方向为 DNA 计算。

(收稿日期：2010-12-15)

© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net