第 26 卷 第 2 期 华侨大学学报(自然科学版) Vol. 26 No. 2
2005 年 4 月 Journal of Huaqiao University (Natural Science) Apr. 2005

文章编号 1000-5013(2005)02-0168-04

嵌入式实时操作系统 μC/OS 中高精度定时的实现

洪 健

(华侨大学机电及自动化学院, 福建 泉州 362011)

摘要 实时系统对定时服务的精度有较高的要求, μC/OS-Ⅱ原有的定时服务实现较为简单, 精度不高. 因此提出一种能够提供更高精度定时服务的改进方法. 通过采用动态更新的时间表, 对任意随机发生的定时请求进行插值及记录, 时间的属性以邮件方式传递. 从而, 使 μC/OS-Ⅱ在原有时钟节拍不变的基础上, 提供对微秒级精度的任务级定时请求的支持, 大幅度提高系统时间响应的准确性. 文中给出具体的实现过程.

关键词 嵌入式实时操作系统, μC/OS-Ⅱ, 定时, 时钟节拍

中图分类号 TP 316.2; TH 714.8 文献标识码 A

在工业控制领域, 各种控制设备、仪器仪表的软件设计中, 广泛地使用嵌入式实时操作系统. μC/OS-Ⅱ作为一种源码公开的嵌入式实时操作系统, 由于其系统结构简单、效率高、稳定可靠, 得到了广泛的应用. 对于实时系统, 精确、快速的定时服务是十分重要的性能指标. 标准的 μC/OS 操作系统所提供的时钟节拍(Clock Tick)为 10~ 100 次·s^{-1}. 这时, 定时精度最高只有 10 ms, 这在很多需要高速、精确定时的强实时性应用中很难满足要求. 虽然时钟节拍可以通过修改用户配置参数的方法进行调整提高, 但单纯使用这个方法将导致操作系统由于频繁的任务调度而产生很大的负荷. 本文提出的方法, 能够在不提高系统原有时钟节拍的情况下, 通过对与移植相关代码的改进, 提供微秒级的高精度定时服务.

1 μC/OS-Ⅱ的时钟节拍工作原理

在 μC/OS-Ⅱ操作系统中, 与定时相关的功能均基于系统的时钟节拍[1]. 时钟节拍是特定的周期性定时器中断. 对于不同的应用, 该时钟节拍一般在 10~ 100 次·s^{-1}的范围内选取, 其对应的中断间的时间间隔为 200 ~ 10 ms, 这是 μC/OS-Ⅱ所能提供的最小时间分辨单位. 例如, 若时钟节拍为 100 次·s^{-1}, 则系统将不支持任何要求精度小于 10 ms 的定时需求. 在许多实时控制场合, 这样的定时精度过于粗糙, 不能满足使用要求.

定时精度为时钟节拍的倒数. 因此, 适当选取较高的时钟节拍可以提高定时精度. 但由于操作系统对每次发生的时钟节拍的处理均包含复杂的任务运算及切换调度, 相当耗时. 因此, 时钟节拍越高, 操作系统占用的运算时间就越多, 这在运算能力普遍不高的嵌入式应用中是不可行的. 所以, 过高的时钟节拍将不具有实用价值.

在 μC/OS-Ⅱ操作系统中, 定时器使用固定的时间常数, 周期性地产生中断, 系统响应中断, 产生时钟节拍. 中断服务程序完成的处理为

```
void OSTickISR(void) {
    ⋮
    重置时间常数;
```

收稿日期 2004-10-09

作者简介 洪 健(1970-), 男, 讲师, 主要从事自动检测与控制的研究. E-mail: jhong@hqu.edu.cn

基金项目 泉州市工业计划基金资助项目(2003G09)

```
    OSTimeTick( );
    OSIntExit( );
    ⋮ }
```

其中 OSTimeTick 是操作系统的节拍服务函数, 主要实现通过对任务控制块 OS _ TCB 链表的遍历, 跟踪所有任务的定时及超时时限. OSIntExit 用于进行中断级的任务调度. 考虑到这是一个较复杂的运算过程, 导致在中断服务程序中执行的代码过长, μC/ OS- Ⅱ还建议了另外一个从任务级执行 OS- TimeTick 的方法, 即建立一个具有最高优先级的任务. 由其处理与时钟节拍相关的事务, 通过时间中断服务程序以发送邮件的方式激活.

2　时钟节拍的改进

对用于时钟节拍的定时器产生时间中断的策略上, 有两种方法[2]. 一种是采用固定时间间隔, 定时时间是一个常数. 其优点是实现简单, 但功能单一, 灵活性差, μC/ OS- Ⅱ即是采用这种方法. 另一种方法, 采用可变定时时间, 定时器每产生一个中断后, 下一次所使用的定时时间再被动态地取出并写入定时器, 该定时时间可动态修改. 本方案提出的方法就是采用这种策略. 在这里, 定时器必须同时承担下列两方面的任务. (1) 产生操作系统所要求的时钟节拍. 这是一个相对较长时间的周期性定时服务, 且定时精度要求不高. (2) 在任意一个不确定的时间点上接受精确延时请求并产生相应的一次性定时服务. 在实时系统中, 对该定时精度要求较高.

方案的核心是建立一个动态更新的定时器时间表 TTbl, 用于记录及跟踪定时服务的请求. 表结构如图 1 所示. 时间表 TTbl 用结构数组实现, 与指针 (CurTP) 一起构成一个循环队列. CurTP 为循环队列指针, 总是指向当前的定时时间设定值. Tset 为定时时间长度, Prio 为任务优先级. 由于每个任务均有唯一的优先级, 此数值可用作任务的识别号. 在本方案中, TickTask 任务使用优先级 0, 且其不会产生精确延时请求. 因此, 0 被用于代表系统周期定时, 其他数值对应请求精确延时的任务的优先级.

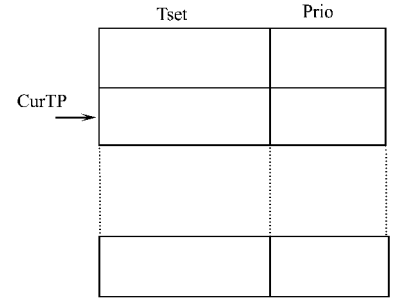

图 1　TTbl 表结构

由下列操作实现对时间表 TTbl 的使用及维护. (1) 定时器中断. 当定时器中断发生时, 首先向 TickTask 发邮件将其激活, 此时 CurTP 所指向的为当前表项. 该邮件将 TTbl[CurTP]. Prio 作为邮件消息发送. 当前表项用默认的系统周期时间 Tcyc 及优先级 0 更新, 形成默认的系统周期定时. 然后修正 CurTP 指针, 使其指向下一个表项, 取出 Tset 值并据此设置定时器的定时时间. (2) 各任务通过调用 ExTimeDlySet 请求精确定时服务. 通过计算得出插入点及时间长度并插入 TTbl. 每次插入必将多出一个表项, 因此需要将插入点后的表项逐个后移一位, 表中的最后一项按溢出抛弃的原则处理. 采用选取适当的表长度、不直接使用长精确定时等方法可防止一次性定时项被溢出. 为方便表述, 以当前 CurTP 指向的表项为第 0 项, T_n 为第 n 项的时间长度, 请求定时时间记做 T_d. 考虑到定时请求的随机性, 当请求发生时, 当前的准确时间在设定时间的基础上已经有了一定的时间流逝. 因此, T_0 不能使用表中的数值, 必须通过读取定时器的当前时间获取. 如图 2 所示, 设系统的周期定时为 1 ms, 任务 1 请

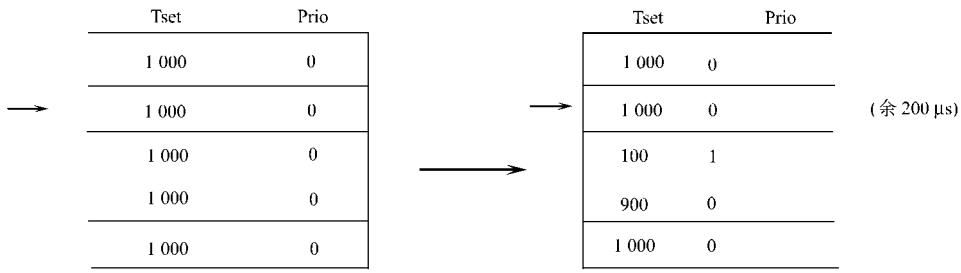

图 2　插值结果示例

求定时时间 T_d = 300 μs. 当请求发生时, 定时器已经计时 800 μs, 则插值前后对照如图示. 本算法流程

图, 如图 3 所示.

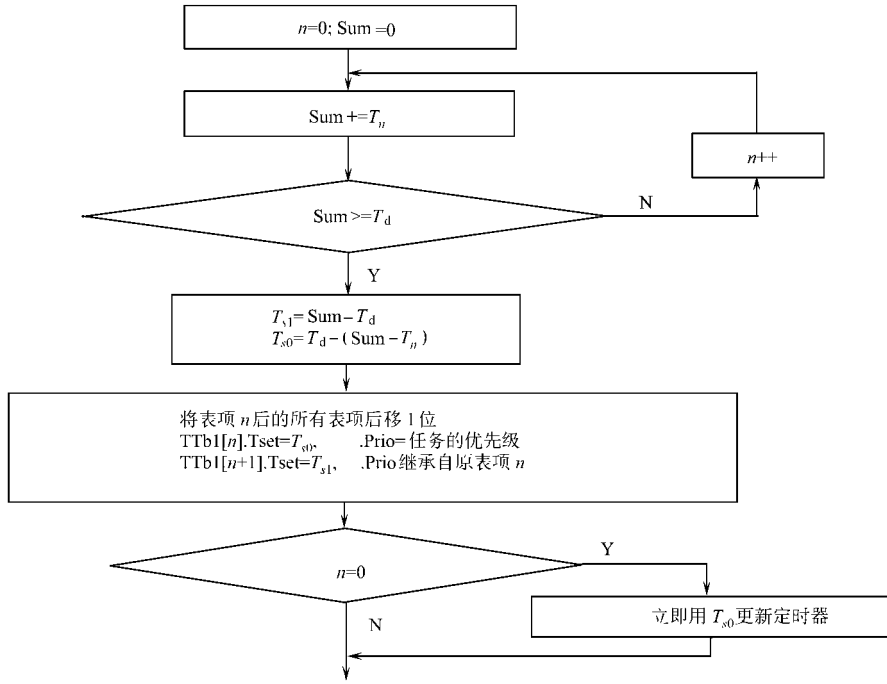

图 3 插值算法流程图

3 实现方法

在使用本方案移植 μC/OS-Ⅱ时[3], 需要作下列工作. (1) 建立任务 TickTask, 此任务分配以最高优先级 0, 其程序示意性代码如下[4].

```
void TickTask (void * pdata)  {
    ⋮
    while (1) {
        OSMBoxPend( …);
        if (Prio = = 0)  {       /*  Prio 为取自邮箱的邮件消息数据 */
            OSTimeTick();        /*  调用操作系统的节拍服务函数 */
            OS_Sched();          /*  进行任务级切换 */
        } else {
          OSTimeDlyResume(Prio); /*  任务精确定时时间到 */
        }
      }
    }
```

(2) 编制 ExTimeDlySet 函数. 函数只实现对时间表 TTbl 的插值运算, 任务进入等待状态将通过调用 OSTimeDly 实现. 需要注意的是, 为防止冲突, 在函数开始和结束时, 要调用 OS_ENTER_CRITICAL 和 OS_EXIT_CRITICAL 以进入及离开临界段. 其流程图如图 2 所示. (3) 编制 TimerISR 定时器中断服务程序, 其示意性代码为

```
void OSTickISR(void)  {
    保存处理器寄存器的值;
    调用 OSIntEnter, 或是将 OSIntNesting 加 1;
    if (OSIntNesting = = 1) {
        OSTCBCur- > OSTCBStkPtr = SP ; }
    以 TTbl[ CurTP] . Prio 为邮件消息调用 OSMboxPost;
```

```
TTbl[CurTP].Tset= Tcyc; TTbl[CurTP].Prio= 0;
指针 CurTP 指向下一表项;
自 TTbl 取出 Tset, 重写定时器;
清发出中断设备的中断, 重新允许中断;
调用 OSIntExit;
恢复处理器寄存器的值;
执行中断返回指令;  }
```

该子程序用汇编语言实现, 代码在 OS_CPU_A.ASM 中书写. (4) 时间表初始化. 时间表 TTbl 必须在定时器打开前完成初始化, 将所有表项的 Tset 设置为周期时间 Tcyc, Prio 设置为 0. 此操作可在 OSInitHookBegin 中完成. 一般情况下, 各任务采用获得改进后的高精度定时服务方式为

```
void MyTask(void * pdata)  {
    ⋮
    while (1) {
    ⋮
    ExTimeDlySet(Td); /* 设定精确定时时间 */
    OSTimeDly(-1); /* 延时等待, 使用-1 确保任务只被精确定时时间到唤醒*/  }
    ⋮
    }
```

4 结束语

本方案通过对 μC/OS-Ⅱ的改进, 能够提供更为精确的定时服务. 同时, 不对 μC/OS-Ⅱ的源码进行改动, 确保了其可移植性及对后续版本的兼容能力. 改进方案的代码全部在与移植相关的文件 OS_CPU_A.ASM 及用户的任务级编程中实现. 本方案对强实时性的嵌入式系统设计有普遍的应用价值, 已在 MCS51 系列的 8 位机及 ARM7 系列的 32 位机上成功移植. 其运行稳定, 效果良好.

参　考　文　献

1 Labrosse J J. 嵌入式实时操作系统μC/OS-Ⅱ[M]. 第 2 版. 北京: 北京航空航天大学出版社, 2003. 103~111
2 Hollabaugh C. 嵌入式 LINUX——硬件软件与接口[M]. 北京: 电子工业出版社, 2003. 236~269
3 周立功. ARM 微控制器基础与实战[M]. 北京: 北京航空航天大学出版社, 2003. 391~414
4 刘云生, 吴　飞. 一种改进的嵌入式实时 OS 消息机制[J]. 计算机工程与应用, 2002, 13: 105~107

Realization of Highly Precise Timing in an Embedded System of Real-Time Operation

Hong Jian

(College of Mechanical Engineering and Automation, Huaqiao University, 362021, Quanzhou, China)

Abstract Real-time system requires a fairly high precise timing service. However, original timing service of the simple μC/OS-Ⅱ is not high enough in precision. Therefore, the author puts forward an improved method which is able to offer a timing service with higher precision. By adopting a time schedule of dymamic renewal, the arbitrary timing requirement occurring randomly can be interpolated and registered while the attribute of time can be transfered in the form of postal item. Thus the support can be given to a timing requirement which is at task level and is of microsecond precision while the μC/OS-Ⅱ remains to be in original clock rate; the precision of the time response of system can be greatly improved. The exact processes of realization are given in the text.

Keywords embedded system of real-time operation, μC/OS-Ⅱ, timing, clock rate