事件/时间触发嵌入式操作系统内核的设计

陈 曦,吕伟杰,刘鲁源
CHEN Xi,LV Wei-jie,LIU Lu-yuan

天津大学 电气与自动化工程学院,天津 300072
School of Electrical Engineering and Automation,Tianjin University,Tianjin 300072,China
E-mail:chenxi@tju.edu.cn

CHEN Xi,LV Wei-jie,LIU Lu-yuan.Design and implementation of real time kernel supporting event/time mixed trigger. Computer Engineering and Applications,2008,44(16):87-89.

Abstract: With the development of real-time distributed system,the real time kernel should meet new requirements such as event/time mixed trigger.To minimize the jitter because of the interference of time-triggered activities,an improved schedule of time-triggered tasks is proposed.A new embedded real time kernel named TTuCOSII (Time-Triggered Micro Operating System II) is implemented by adding the time-trigger module to the uCOSII which is an open source operating system kernel.Unlike other real-time kernels,TTuCOSII supports both event trigger and time trigger.Simulation shows that TTuCOSII is of high precision of time-trigger and high performance of scheduling,which fulfills the requirement of event/time mixed triggered system.
Key words: event/time mixed triggered;scheduler;jitter

摘 要:针对分布式实时系统对操作系统内核的新需要,为避免因时间触发任务相互影响而造成的抖动,研究了时间触发任务的调度器设计,提出了一种改进的调度策略;在开放源代码的 uCOSII 嵌入式操作系统内核的基础上扩展了时间触发功能,设计了支持事件/时间混合触发的嵌入式实时操作内核 TTuCOSII(Time-Triggered Micro Operating System II)。仿真实验表明 TTuCOSII 具有较高的时间触发精度,良好的调度性能,可以满足事件/时间混合触发的要求。
关键词:事件/时间混合触发;任务调度;抖动
DOI:10.3778/j.issn.1002-8331.2008.16.026 **文章编号**:1002-8331(2008)16-0087-03 **文献标识码**:A **中图分类号**:TP336

1 引言

事件触发任务是指实时操作系统中的一个任务只有在与之相关的特定事件发生的条件下才能开始运行。时间触发任务是指实时操作系统中的一个任务在某一预先决定的时间点上才能开始运行。目前常见的嵌入式实时操作系统内核并不能满足事件/时间混合触发的要求。有些操作系统如 uCOSII、Power PAC、FreeRTOS 等只支持事件触发而不支持时间触发;有些操作系统如 RTX 支持事件触发和时间片轮转调度,但时间片轮转调度并不等同于时间触发。原因在于时间片轮转调度中的某个时间片可以分配给当前任何一个就绪的任务,而时间触发中的一个时间片(时间槽)只能分配给特定的一个任务,如图 1(a)所示,例如时间槽 2 只能分配给时间触发任务 2,而不能分配给除时间触发任务 2 以外的任何其它时间触发任务。

支持事件/时间混合触发的嵌入式实时操作系统内核在分布式系统中有着广泛的应用前景。例如分布式系统整体调度[1,2]要求操作系统内核不仅要支持目前常见的事件触发,而且要支持时间触发。又如 TTCAN[3,4]是一种同时支持事件/时间触发的通讯协议,目前只有 BOSCH 公司提供 FPGA 形式的 TTCAN 接口器件,一种可行的、工程化的、低成本实现方案就是以运行支持事件/时间混合触发的嵌入式实时操作系统内核的单片机为控制器,以支持点发模式的器件如 MCP2515 为 CAN 接口器件构成 TTCAN 系统。

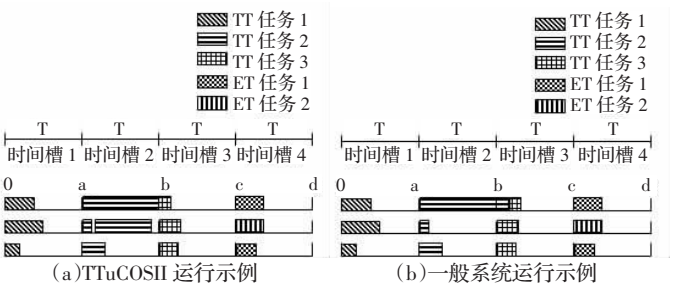

图 1 支持事件/时间混合触发的实时操作系统运行示例

支持事件/时间混合触发的嵌入式实时操作系统内核需满足下述要求:(1)时间触发任务要在预先决定的时间点上开始运行,如果在此时有事件触发任务,那么该事件触发任务将被抢占。(2)时间触发采用本地时钟作为时间基准,本地时钟间的

基金项目:天津大学青年教师基金(the Foundation of Young Teachers of Tianjin University)。
作者简介:陈曦(1980-),男,博士生,助教,主要研究领域为嵌入式系统,现场总线;吕伟杰(1975-),女,博士后,副教授,主要研究领域为网络控制系统,分布式系统;刘鲁源(1941-),男,教授,博士生导师,主要研究领域为计算机控制系统,网络控制系统。
收稿日期:2007-11-13 **修回日期**:2007-12-27

同步通过通讯协议来实现。(3)事件触发的任务采用基于优先级的调度策略。uCOSII[5]是目前常见的基于事件触发机制、开放源代码的嵌入式实时操作系统内核,本文以该内核为基础扩展时间触发功能,研究了时间触发任务的调度器,设计了支持事件/时间混合触发的 TTuCOSII(Time-Triggered Micro Operating System II)嵌入式实时操作系统内核。

2 时间触发机制的设计

2.1 TTuCOSII 的任务状态及任务控制块

TTuCOSII 中时间触发(Time-Triggered,以下简称为 TT)任务共有休眠、就绪、运行、挂起和中断五种状态,状态之间的转换关系如图 2 所示。

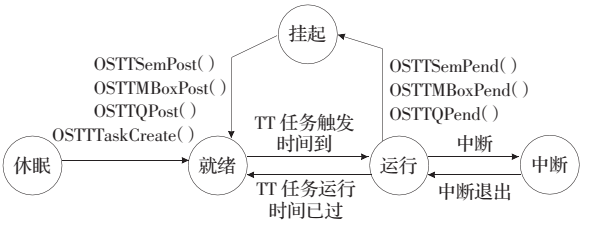

图 2 TTuCOSII 中 TT 任务的状态及其转换关系

一个 TT 任务的 C 语言形式伪代码如图 3 所示。

```
void TTTask1(void)
{
    while(DEF_TRUE) {
        OSTTBegin();
        TT 任务代码;
        OSTTEnd();
    }
}
```

图 3 TT 任务的 C 语言形式伪代码

一个处于休眠态的 TT 任务是指该任务驻留在程序空间中,还没有被 TT 任务调度器识别和调度。把 TT 任务交给 TT 任务调度器管理的方法是通过建立 TT 任务的系统函数 OSTTTaskCreate(),之后 TT 任务由休眠状态变为就绪状态。由图可知与事件触发(Event-Triggered,以下简称为 ET)任务调度算法不同,一个 TT 任务由就绪态变为运行态的前提条件是与该任务对应的时间到,当 TT 任务运行时间过去后,该 TT 任务立即变为就绪态,而且 TT 任务由运行态变为就绪态并不受 ET 任务影响。TT 任务之间可以通过信号量、邮箱、消息队列等进行通信。如果一个 TT 任务为等待某一事件发生而调用 OSTTSemPend()、OSTTMBoxPend()或 OSTTQPend(),那么该TT 任务就进入挂起状态。当该 TT 任务等待的事件发生后,该任务由挂起状态变为就绪状态。一个处于运行状态的 TT 任务可以被中断打断,中断服务程序执行完成后处于运行状态的 TT 任务将继续执行。

TT 任务与其触发时间紧密相关,为了描述触发时间需要确定时间的起点(如图 1(a)中的时刻 0)和时间额度(如图 1(a)中的 T)。TT 任务有两种确定时间起点的方式:一是主动启动方式,即当系统上电复位后,当定时器第一次产生中断时就认为是时间起点,TT 任务调度器以此时间起点为开始进行调度;二是被动启动方式,也就是当系统上电复位后,TT 任务并不马上运行,而是等到收到同步消息后,TT 任务以此时间起点为开始进行调度。根据前文所述事件/时间混合触发的要求(2),一般情况下采用被动启动方式。但当某些特殊情况下,如该任务作为 TTCAN 时间基准发送时间同步帧时需要采用主动启动方式。TTuCOSII 支持上述两种启动方式。

利用硬件定时器可以产生精确的时间额度,其值由 1/OS_TT_TICKS_PER_SEC 决定,OS_TT_TICKS_PER_SEC 表示每 1 秒包含的时间槽个数,根据时间触发的定义,每个任务的触发时间必须各不相同,所以每个 TT 任务可以用其触发时间来唯一表示。由于触发时间一定是时间额度的整数倍,如图 1 中,TT 任务 1 的触发时刻为 0T,TT 任务 2 的触发时间为 1T,所以采用整数来唯一标示每个 TT 任务。TT 任务采用如图 4 所示的 TT 任务控制块来描述。

```
typedef struct os_tt_tcb {
    OS_TT_STK    *OSTTTCBStkPtr;
    struct os_tt_tcb  *OSTTTCBNext;
#if OS_TT_EVENT_EN || (OS_TT_FLAG_EN>0)
    OS_TT_EVENT          *OSTTTCBEventPtr;
#endif
    INT8U        OSTTTCBStat;
    INT8U        OSTTTCBCont;
    INT32U       OSTTTCBTime;
} OS_TT_TCB;
```

图 4 TT 任务控制块

TT 任务控制块 OS_TT_TCB 中的 OSTTTCBTime 是用来标示每个 TT 任务的整数,如 TT 任务 1 的 OSTTTCBTime 为 0,TT 任务 2 的 OSTTTCBTime 为 1。

当 TTuCOSII 初始化 OS_TT_TCB 后,通过调用 OSTT-TaskCreate()函数把 TT 任务插入到 TT 任务列表中。该任务列表利用单向链表把 OS_TT_TCB 链接在一起,OS_TT_TCB 中的 *OSTTTCBNext 指向单向链表的后继。根据 TT 任务调度的特点,如果当前 TT 任务完成后,在没有接受到时间同步帧的情况下,下一个即将运行的 TT 任务的 OSTTTCBTime 的值一定大于当前 TT 任务,所以为了加快 TT 任务调度,该任务列表根据 OSTTCBTime 的值由小到大依次进行排序,如图 5 所示,OSTTTCBList 是单向链表的表头。

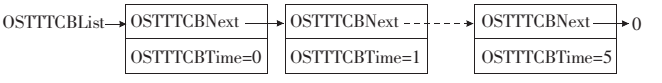

图 5 TT 任务列表

每当建立一个 TT 任务后,该任务就要插入到单向链表中,在插入链表时采用了基于插入排序的方法来实现 OSTTTCBTime 由小到大的排序。

2.2 TT 任务调度器

TTuCOSII 利用定时器产生时钟节拍中断。在时钟节拍中断服务程序中 TT 任务调度器完成 TT 任务的切换。根据前面所述的事件/时间混合触发的要求,TT 任务调度器应实现下述过程。

(1)检测是否有时间同步帧,其代码如下:

```
OS_ENTER_CRITICAL();
if(OSTTRestart == OS_TRUE){
    OSTTRestart = OS_FALSE;
    OSTTTick = 0;
```

```
        OSTTTCBCur = OSTTTCBList->OSTTTCBNext;
    }
    OS_EXIT_CRITICAL();
```

因为时间同步帧通常由通讯中断服务程序捕获，而 TT 任务调度器也是在时钟节拍中断服务程序中运行的，两个中断程序之间交换信息的方法是通过全局变量，所以利用全局变量 OSTTRestart 来判断是否有时间同步帧，如果有即 OSTTRestart ==OS_TRUE，则要更新当前任务指针 OSTTTCBCur 使它重新指向 TT 任务列表的第一个任务且把计时变量 OSTTTick 清零。

(2)检测计时变量 OSTTTick 是否到达最大值 OS_TT_TICKS_MAX,其代码如下:

```
if(OSTTTick == OS_TT_TICKS_MAX) {
    OSTTTick = 0;
    OSTTTCBCur = OSTTTCBList->OSTTTCBNext;
}
```

如果 OSTTTick 已达到最大值，那么意味着要开始新一轮的调度,所以更新当前任务指针 OSTTTCBCur 使它重新指向任务列表的第一个任务且把计时变量 OSTTTick 清零。

(3)取 TT 任务列表中的当前任务,如果当前任务的 OSTTTCBTime 和 OSTTTick 相等，那么首先调用 OSETSchedLock 来禁止 uCOSII 的任务切换,抢占 ET 任务,然后更新计时变量 OSTTTick,最后进行 TT 任务切换;如果不等,那么在更新计时变量 OSTTTick 后通过调用 uCOSII 的时钟节拍函数 OSTimeTick 来允许 ET 任务切换。其代码如下:

```
if(OSTTTick == OSTTTCBCur-> OSTTTCBTime) {
    OSETSchedLock ();
    OSTTTick++;
    OSTTCtxSw();
} else {
    OSETSchedUnlock();
    OSTTTick++;
    OSTimeTick();
}
```

其中的 TT 任务切换函数 OSTTCtxSw()因为要涉及到堆栈指针的操作,所以需要用汇编语言实现。

2.3　TT 任务切换函数的实现与抖动的处理

理想情况下 TT 任务应该能够在时间额度规定的时间内完成,但是在 TT 任务实际运行过程中,可能会出现 TT 任务运行时间超过时间额度的情况,如果采用如图 1(b)所示的调度策略,由于 TT 任务 2 的运行时间超过了时间额度规定的时间,造成 TT 任务 3 产生了启动抖动。因为抖动的传递性,所以在最坏情况下会造成所有的 TT 任务产生抖动。为了避免这种情况,TTuCOSII 采用了如图 1(a)所示的调度策略,即无论 TT 任务在时间额度规定的时间内是否完成任务,TT 任务调度器都会在指定的时间点上进行切换,如图 1(a)中的 TT 任务 2 虽然在时间槽 2 内并未完成，但是在时间点 b 处 TT 任务调度器仍旧进行 TT 任务调度,转而执行 TT 任务 3。TT 任务 2 未完成的部分将在下一次指定的时间槽到后继续执行,当任务执行完成后再开始执行新一轮 TT 任务 2。这种调度策略的优点是使 TT 任务的启动抖动只取决于该任务本身，消除了 TT 任务之间的相互影响而造成的 TT 任务抖动。为了实现这种调度策略,一方面在 TT 任务的结束处要调用系统函数 OSTTEnd,另一方面要对 TT 任务切换函数进行处理。

TT 任务代码中的 OSTTEnd()函数其主要作用是使 OSTTTCBCur->OSTTTCBCont=OS_FALSE,即使 TT 任务“继续执行的状态”为假。如果 TTTask 在时间额度规定的时间内完成,OSTTEnd()函数一定会被执行,从而使 OSTTTCBCur->OSTTTCBCont=OS_FALSE,TT 任务切换函数 OSTTCtxSw()检测到 OSTTTCBCur->OSTTTCBCont 为 OS_FALSE 就会从头开始执行 TTTask；如果 TTTask 并未在规定的时间内完成,OSTTEnd()函数就不会执行,TT 任务切换函数 OSTTCtxSw()检测 OSTTTCBCur->OSTTTCBCont 为 OS_TRUE,就会继续执行未完成的部分。

TT 任务切换函数 OSTTCtxSw()的 C 语言形式伪代码如下所示:

```
void OSTTCtxSw(void)
{
    if(OSTTTCBCur->OSTTTCBStat==OS_TT_STAT_RDY
            且 OSTTTCBCur->OSTTTCBCont=OS_FALSE) {
        取当前任务的堆栈 OSTTTCBCur->OSTTTCBStkPtr;
        TT 任务指针指向下一个要运行的任务 OSTTTCBCur=
OSTTTCBCur->Next;
        进行 TT 任务切换;
    } else if(OSTTTCBCur->Status==OS_TT_STAT_PEND){
        if(任务通讯或同步事件==就绪) {
            TT 任务指针指向下一个要运行的任务 OSTT-
TCBCur=OSTTTCBCur->Next;
            从挂起处继续执行代码;
        }
    } else if(OSTTTCBCur->OSTTTCBCont=OS_TRUE){
        取当前任务的堆栈 OSTTTCBCur->OSTTTCBStkPtr;
        TT 任务指针指向下一个要运行的任务 OSTTTCBCur=
OSTTTCBCur->Next;
        从原任务切换处执行代码;
    }
}
```

3　仿真实验

为了验证 TTuCOSII 的系统性能，使用 Keil MDK 以 AT91SAM7X256 芯片为对象进行了仿真。实验设置为:TT 任务 1,TT 任务 2,TT 任务 3 分别使 AT91SAM7X256 的三个 I/O 口定时产生单脉冲。ET 任务 1 和 ET 任务 2 使 AT91SAM7X256 的两个 I/O 口产生单脉冲,且 ET 任务 1 的优先级高于 ET 任务 2,时间额度为 25 ms。使用 CAN 总线通讯产生时间同步消息。Keil MDK 的 Logic Analyzer 观察到的波形如图 6，测得的 TTuCOSII 主要时间参数如表 1 所示。

表 1　TTuCOSII 的主要时间参数

时间参数	时间/us
TT 任务建立时间	17.1
ET 任务建立时间	13.0
TT 任务切换时间	12.3
ET 任务切换时间	11.1
IRQ 相应时间	0.9

通过表 1 可以看到 TTuCOSII 的各项性能指标能够满足大多数嵌入式系统的需要。TT 任务的建立时间长于 uCOSII 的建

(下转 144 页)