一种嵌入式微调度器的实现方法*

■重庆邮电大学　曾素华　蒋建春

摘　要　常用的嵌入式操作系统不能很好地满足中低端仪器仪表小内核、高实时性、高可靠性的要求。本文通过分析常见嵌入式操作系统的任务调度算法，提出一种新的应用于智能仪表的实时任务调度算法，并在典型的8位、16位SoC上进行设计，实现了基于这种算法的嵌入式操作系统。

关键词　微调度器　仪器仪表　调度算法

智能仪表是自动控制技术的重要组成部分。随着智能仪表在工业控制、通信和汽车电子中的广泛应用，智能仪表逐渐向数字化、网络化、智能化方向发展；同时，智能仪表复杂度不断增加，对实时性要求几乎达到了苛刻的程度。在编程方式和代码重复利用等方面，超循环方式的智能仪表越来越不能满足资源管理和系统的实时要求，迫切需要在中低端智能仪表中加入一些轻量级的多任务管理的调度器或实时操作系统。本文根据智能仪表对嵌入式操作系统的特殊要求设计了一种新的任务调度算法，并实现了一个应用于中低端仪器仪表的嵌入式微调度器。

1　实时任务调度的一般方法和策略

在实时操作系统中，系统把应用分为行为可以预知的、功能确定的多个任务。每个任务一般处于3种状态：执行状态、就绪状态和等待状态(有的操作系统还具有挂起和休眠状态)。为了满足实时性要求，系统根据一定的原则选择合适的任务执行。

常见的任务调度算法分为静态算法和动态算法两类：

① 静态算法：在系统在运行前(即系统初始化阶段)，就为所有的任务分配固定的优先级别，在系统执行过程中优先级保持不变。当一个事件发生时，调度程序只需要查就绪表，就可以调度哪个任务处于运行状态。

② 动态算法：在系统初始化时初步分配一个优先级。每一个任务在运行时可以改变它的优先级。

当前的嵌入式操作系统一般采用静态算法，只在处理优先级反转时临时采用动态优先级算法。

2　仪器仪表对调度算法的要求

为了提高仪表的可靠性，实现高性能、多功能应用，应用于智能仪表的调度器必须满足以下要求：

① 良好的实时性。智能仪表必须实时地对通过现场总线采集的数据进行数字编码，通过人机界面进行显示，并把用户对被监控系统的参数设置实时地传送给执行部件。

② 基于优先级的任务调度策略。在复杂的大规模应用中需要使用大量的传感器、执行器和控制器等，对其数据显示和传输控制需要通过不同优先级的任务来控制。

③ 低消耗要求。随着应用环境的复杂化，对智能仪表的计算能力要求越来越高，势必要求调度器必须占用较少的系统资源。

④ 低成本要求。为了降低成本，在硬件设计上，存储器的大小是成本控制的一个方面。因此，要求提供的调度器必须具备小内核以减小存储空间。

此外，还要求调度器必须有精确定时的功能，也就是事件驱动和时钟驱动相结合，以满足智能仪表中周期性任务执行和突发性任务执行的需要。

3　嵌入式微调度器的设计与实现

根据智能仪表对调度算法实时性、多任务、低消耗的要求，本文提出了一种新的静态优先级，单任务队列、具有4种任务状态的非抢占式调度的轻量级任务调度算法，并根据这种算法实现了应用于智能仪表的调度器。该算法的特点是以任务在任务控制块数组中的相对位置表示优先级高低，任务的状态和延时量使用统一的任务状态字，在少量任务的轻量级应用中具有很好的时间和空间性能。

* ① 国家高技术研究发展计划(863计划)(2006AA11A1C1－3，汽车节气门控制软件及电控系统支撑技术开发)；
② 国家高技术研究发展计划(863计划)(2006AA11A107－2 长安混合动力汽车标定系统开发)。

3.1 任务的状态

在本调度器中任务有4种状态:就绪状态、运行状态、等待状态和挂起状态。内存中的任务必须处于这4种状态之一。

就绪状态:指任务运行的时间条件和资源条件都满足,等待调度算法选择最合适的任务进入就绪状态。任务一旦建立就处于就绪状态,这一点和μC/OS-Ⅱ相同。

运行状态:是当前时刻任务占有CPU资源正在运行的状态。本调度算法选择进入就绪任务队列中优先级最高的任务运行。任何时刻只能有一个任务处于运行状态。

等待状态:如果任务需要等待一段时间才能运行,那么这个任务当前处于等待状态。使任务延迟一段时间可通过调用Os_Task_Delay()函数实现。调度器在每个系统时钟节拍检查任务延迟时间,一旦任务定义的延迟时间到,就使任务进入就绪状态。

挂起状态:正在运行的任务需要等待某一事件的发生,如果该事件没有发生那么任务就处于挂起状态。事件的发生可能来自另外一个任务,也可能来自中断服务程序。

除此之外,系统还可能处在中断服务状态。这是一种特殊的运行状态,当系统响应中断时,正在执行的任务被挂起,中断服务程序控制了CPU的使用权,系统就进入中断服务状态。

其中,空闲任务优先级最低,而且永远处于就绪状态,而且当所有的任务都在等待事件发生或者延迟时间结束时,操作系统就会执行空闲任务。

3.2 调度器核心数据结构

3.2.1 任务控制块和任务控制块列表

任务控制块由任务堆栈、任务入口地址、任务状态字和任务优先级4个部分组成。任务堆栈用于保护被中断的现场数据;任务入口地址是指向任务程序的指针,用于指定任务所进行的操作;任务状态字用来表示任务当前的状态和延迟的时间间隔;任务优先级表示就绪列表中的哪个任务可以优先进入运行状态。

在整个调度过程中使用一个全局的任务控制块数组来表示任务控制块列表。每个任务使用唯一一个任务控制块表示,任务的优先级通过任务控制块在任务控制块数组中的相对位置来表示。每个任务有且仅有一个优先级,所以任务的优先级也可以用任务的ID号来表示。任务控制块结构如下:

```
typedef struct {          //tsk_tcb结构定义
    pStack stack;         //tsk_tcb堆栈入口
    pTASK task;           //tsk_tcb指向的任务
    U8 state;             //tsk_tcb任务目前的状态
    U8 prior;             //任务优先级
}TCB;
```

3.2.2 任务调度算法及实现

这种算法已在16位单片机Motorola MC9S12DP256B和8位单片机AT89C52上实现。一些与硬件相关的算法,主要给出在MC9S12DP256B上的算法实现。

① 建立任务Os_Task_Create()算法。任务创建函数代码如下:

```
void Os_Task_Create(OS_STACK  *task_stack,
uWord task_id, pTASK task_func){
    os_tcb[task_id].task=task_func;
    os_tcb[task_id].stack=task_stack;
    os_tcb[task_id].prior=task_id;
}
```

该程序表示了系统建立任务的过程。如上节所述每个任务对应一个优先级,所以任务ID也可表示任务的优先级。建立任务的过程就是,把任务控制块数组的任务入口地址对应ID(即任务优先级)的任务控制块的任务入口地址指向任务函数的地址,并初始化该任务的任务堆栈。

② 任务调度算法的功能是找到当前就绪列表中优先级最高的任务,并把这个任务切换到运行状态。在任务控制块列表中使用任务在列表中的相对位置表示优先级的高低,并不需要实际地对任务优先级进行比较。算法流程如图1所示。

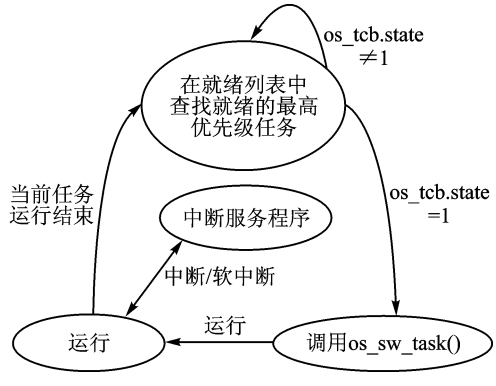

图1 任务调度算法状态描述

从任务控制块队列的头部(即任务优先级为0的任务)开始依次检查任务就绪标志(os_tcb.state),如果当前任务标志≠1,表示当前任务为非就绪状态,继续检查下一优先级的任务。如果当前任务标志为1,则找到最高优先级任务退出循环,调用任务调度函数进行任务状态切换。

任务的调度算法如下:

```
void os_schedule_task (void){
  int i;
  pCur_task=pHi_task;
  for(i=0;i<TASKNUM &&os_tcb[i].state !=1;i++){
```

```
    }
    Hi_task=i;
    if (pHi_task != &os_tcb[i]) {
      pHi_task= &os_tcb[i];
      os_sw_task();
    }
  }
```

任务级切换函数需要改变程序计数器(PC),所以必须通过软中断实现。在软中断服务函数中改变当前运行任务的 TCB 指针到最高优先级就绪任务,执行中断返回指令在新的任务堆栈中弹出最高优先级任务的 PSW 和 PC 指针,从而完成任务切换。

③ 任务状态转换主要是激活任务 os_Task_Active()、挂起任务 os_Task_Suspend()和延迟任务 os_Task_Delay()。挂起任务使任务进入挂起状态,延迟任务使任务进入等待状态,而激活任务函数可以使任务从挂起状态或者等待状态直接进入就绪状态。任务的状态由任务控制块中的任务状态字(os_tcb.state)给出。当 os_tcb.state=1 时表示任务进入就绪状态;当 os_tcb.state=0 时表示任务处于挂起状态;当 os_tcb.state>1 时表示任务等待 os_tcb.state-1 个系统时钟间隔之后进入就绪状态。任务状态切换示意图如图 2 所示。

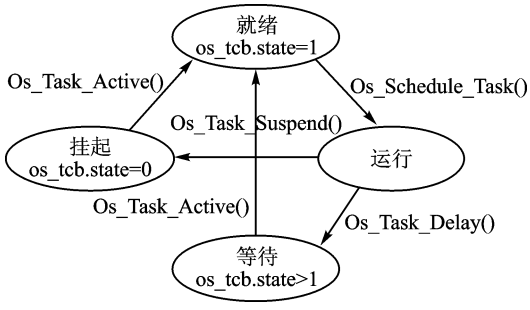

图 2 任务状态切换示意图

④ 由于这些中低端的仪器仪表每个任务的执行时间都比较短,为了避免优先级反转和死锁,采用非抢占式调度方式,进入就绪态的任务必须在当前任务执行完成后才能被调度。调度时处于就绪表中优先级最高的任务进入运行。

3.2.3 调度算法的时钟驱动

时间驱动需要硬件提供时钟节拍来实现任务的定时。时钟节拍信号源可以是专门的硬件定时器,比如 AT89C52 中的 Timer2。也可以使用其他更精确的方式提供系统时钟节拍。在这里使用 MC9S12DP256B 捕获器的第 7 个通道来实现,时钟中断处理函数如下:

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
interrupt void RTI_ISR(void){
    absoluteTime++;
    /*CRGFLG = 0x80; */
    __asm{
        ldab    #$80
        stab    TFLG1
        ldd     TC7
        addd    #OS_TICK_OC_CNTS
        std     TC7
    }
    os_time_isr();
}
```

#pragma CODE_SEG DEFAULT 在 MC9S12DP256B 的捕获器中有一个自动增长主时钟,每一个硬件周期驱动 TCNT+1,并与 TC7 相比较。设置 TC7=TCNT + OS_TICK_OC_CNTS(在系统配置文件中定义),当度过 OS_TICK_OC_CNTS 个硬件周期时,TCNT=TC7 则产生中断。在中断中调用系统时钟节拍函数提供精确的系统时钟节拍,并再次初始化 TC7=TCNT+OS_TICK_OC_CNTS,产生下一个时钟节拍。

系统时钟节拍函数自动检查每个被延迟的任务,当任务的延迟周期结束后,自动将任务切换到就绪状态。具体算法如下:

① 从任务控制块列表头部开始顺序检查各任务状态字,将所有延迟任务的任务状态字减 1。

② 当前延迟任务的状态字变为 1 时,该任务延时结束,置就绪任务列表改变标志位。

③ 恢复被中断任务状态,返回中断。

系统时钟驱动代码如下:

```
void os_time_isr(void){
  uByte i;
  for (i=0;i<=TASKNUM;i++){
    if (os_tcb[i].state>1){
      os_tcb[i].state--;
    }
  }
}
```

结 语

本文提出的任务调度算法是一个应用于智能仪表系统的中间件,目的是良好地管理 CPU 资源,提供方便的用户应用接口,具有良好的可移植性、时间性能和空间性能。在具有大量周期性任务的轻量级智能仪表的应用中,性能和易用性的提高是非常明显的。该算法已经成功应用于车载智能仪表的图形操作系统中。

基于构件技术的嵌入式系统复用软件设计

■东北电力大学　杨胜春　曲朝阳

摘　要　提高软件生产率成为软件产业的当务之急；基于构件的软件复用是当前复用研究的焦点，被视为实现成功复用的关键因素之一。本文主要讨论基于构件的嵌入式系统软件复用技术，提出工业嵌入式系统的系统级软件设计方法，以软件构件形式对其进行封装，以标准接口形式暴露给用户级应用软件对其进行操作，方便了系统的集成和维护。

关键词　软件复用　嵌入式系统　构件标准接口　软件构件平台

引　言

对嵌入式软件构件平台而言，其支撑平台首先是一个嵌入式实时多任务操作系统，其次为整个软件构件的设计提供开发工具和集成环境。在支撑平台的设计过程中，可以借鉴领域工程的思想，将整个嵌入式实时多任务操作系统设计成一个系统级的软件构件库。这样不但实现了嵌入式操作系统的可裁剪性，而且由于从嵌入式操作系统到应用程序的设计都是基于离散化的软件构件，因此方便了嵌入式控制应用软件设计时的集成和调试。为了方便软件构件的管理，可以将系统级和应用级的软件构件库综合成一个功能完备的软件构件库。它包括从嵌入式控制系统的系统层、支撑层和应用层所需的一切软件构件，因而具有功能的完整性[1]。

1　嵌入式软件构件平台的体系结构

嵌入式软件构件平台的体系结构如图 1 所示，它包括系统层、连接层（支撑层）和应用层 3 个部分。系统层属于领域工程的范畴，它利用领域工程的分析方法对嵌入式控制系统进行分析、抽象和提炼，并分解成相应的系统类和应用类功能模块。连接层是一个嵌入式软件构件平台，它实现系统层和应用层之间的无缝连接，即提供软件构件一个集成开发平台。应用层属于应用工程的范畴，用户根据实际的嵌入式控制系统的控制要求和目标，从软件构件库中选取所需软件构件，经集成后生成实际的嵌入式控制应用程序。

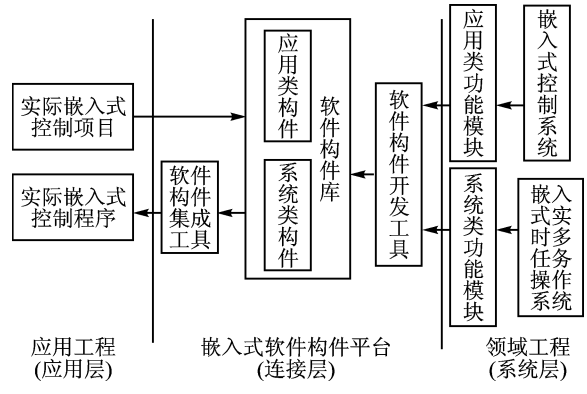

图 1　嵌入式软件构件平台的体系结构

2　嵌入式软件构件平台的设计

该软件构件平台是基于 TMS320F2812 DSP 芯片构建的一个构件化的嵌入式实时多任务操作系统。在设计时充分利用了平台体系结构所述的设计思想，程序的可读性和裁剪性非常好。其特点是：

① 设计了操作系统和系统两个数据结构。为了方便

参考文献

[1] 李传锋，沈安文，赵方亮. 基于 MSP430 的智能仪表与组态王的通讯设计[J]. 微计算机信息，2007(3)：91－92.

[2] 胡宁，张德运，史宏锋. 一种低开销的多任务调度模型[J]. 微电子学与计算机，2005.

[3] 彭芳，张茂青，钱伟清，等. 基于 MSP430 的智能仪表的 LCD 驱动设计[J]. 电子工程师，2006(10).

[4] Jean Labrosse J. 嵌入式实时操作系统 μC/OS－II[M]. 邵贝贝，等译. 第 2 版. 北京：北京航空航天大学出版社，1998.

曾素华（讲师、硕士），蒋建春（讲师）：主要研究方向为计算机控制、嵌入式系统。

（收稿日期：2007-12-03）