Doi:10.3969/j.issn.1003-5060.2010.11.010

# 基于工控单片机的嵌入式操作系统核心的设计

陈 康[1], 黄彩虹[2], 何明华[1]
(1.福州大学 自动化与电气工程学院,福建 福州 350007; 2.华侨大学 信息科学与工程学院,福建 厦门 361021)

摘 要:由于常用的 RTOS 对 CPU 硬件要求较高,导致其在用于工业控制的单片机上使用效果不好。文章采用 C 语言设计了一种小型嵌入式实时操作系统内核,而其对硬件的要求低,容量不超过 3 K 字节,能根据优先级进行任务调度,并提供了信号量与任务间通信功能优先级反转,可以较好地应用在小内存的单片机上。
关键词:RTOS 核心;优先级反转;信号量;堆栈切换
中图分类号:TP316 文献标志码:A 文章编号:1003-5060(2010)11-1639-04

# Design of RTOS core based on single-chip computer in industrial control

CHEN Kang[1], HUANG Cai-hong[2], HE Ming-hua[1]
(1.School of Electrical Engineering and Automation, Fuzhou University, Fuzhou 350007, China; 2.School of Information Science and Engineering, Huaqiao University, Xiamen 361021, China)

**Abstract**: Normal real-time operating system(RTOS) does not have good performance on single-chip computer in industrial control due to its strict requirement for hardware. In this paper, a small-sized RTOS core based on C language is designed, which is no more than 3 KB and has less requirement for hardware. The operating system can schedule the tasks in term of priority and make priority reversal as communication function between semaphore and task is conducted, so it is suitable to the single-chip computer with small memory.
**Key words**: real-time operating system(RTOS) core; priority reversal; semaphore; stack switch

嵌入式实时操作系统(real-time operating system,简写为 RTOS)具有比较好的扩展性,且通过任务这个概念把原有的应用程序分割成若干部分。由于采用可剥夺型的内核,与前后台系统相比,实时性得到了更好的保证,所以 RTOS 在嵌入式系统中得到了广泛的应用[1]。但现有的几个 RTOS(Linux, VxWorks, uC/OS-Ⅱ)代码量比较大,占用较多的内存资源,对 CPU 硬件要求较高,目前在工业控制领域中广泛应用的 CPU 如 8051 系列由于其内存较小(片内存储器仅 128 个字节),速度较慢(外接晶振为 12 M),无法使用这些操作系统。为提高工控软件的开发效率,考虑用 C 语言为其设计一种 RTOS。

## 1 实 现

### 1.1 抢先式算法

由于目前 RTOS 常用算法有抢先式与非抢先式 2 种。考虑到自动控制领域对实时性要求较高,使用抢先式算法,它的主要特点是处于就绪态的最高优先级任务始终运行并占用着 CPU,优点在于能够迅速对外部事件做出响应。在这个内核里,可以把所有的任务分为挂起、就绪、运行、休眠、被中断态 5 种状态,这些状态可以进行相互间的转换,如图 1 所示。某一时刻任何任务都处于 5 种状态之一,操作系统就是要根据设计者的需要实现这些状态之间的转化,并查找就绪任务中

收稿日期:2009-11-19;修回日期:2010-03-28
基金项目:福建省自然科学基金重点资助项目(2007J0003);泉州市科技计划基金资助项目(2008G9)
作者简介:陈 康(1974—),男,福建福州人,福州大学讲师;
何明华(1972—),男,福建福州人,博士,福州大学教授,博士生导师.

优先级最高的使其迅速进入运行状态。

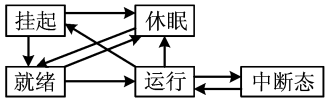


图 1 任务状态机

由于单片机内存容量较小,只有 128 字节,为尽可能地节省内存,在这个操作系统中使用函数数组定义作为基本的数据结构[2],其代码为:

Void ( *processTable[ TOTALTASK] )();

这个数据结构里定义了各个任务的函数入口地址,主要是为了减少内存占用,精简代码量,还可通过各个任务在数组中的索引来确定它的优先级,索引值越大,优先级越高[3]。另外,为区分任务的不同状态,建立了 2 个状态表。

(1) 休眠状态表。用若干个字节来表示(取决于任务数),每位表示一个任务,0 表示休眠状态,1 表示非休眠状态(包括就绪和挂起态)。

(2) 挂起任务状态表。用若干个字节来表示,每位表示一个任务,休眠状态表的该位为 1 的情况下,挂起任务状态表中的 1 表示就绪状态,0 表示挂起状态。可通过状态表中任务所在字节的位置来确定其优先级,位数越高的,优先级越高。

实时操作系统运行时需要找到优先级最高的就绪态任务并让其运行,这就是任务调度,可以通过休眠状态表与挂起状态表相与得到任务就绪状态表,在此表中 1 为就绪态,0 为非就绪态。位数越高的任务代表的优先级越高(例如,bit 7 所代表任务的优先级高于 bit 6),可以通过函数 taskSched()查找就绪状态表中位数最高的 1 以确定下一个进入运行的任务,其实现如图 2 所示。框图中的变量 NextRunningTask 做为全局变量代表下一个即将运行的任务。此外还需按照图 1 根据实际需要对相关的任务进行切换。就是通过程序把休眠状态表或挂起状态表相应的位设为 1 或 0 来实现状态的转换,以 setReady (int *i*)为例画出框图说明如何将挂起态的任务转换为就绪态,*i* 代表想要变成就绪态的任务号,如图 3 所示。此外还有些状态转换的函数与此类似,8051 有 2 个外部中断,在中断中可以调用这些程序来改变任务的状态,由于篇幅的原因不具体介绍。

### 1.2 上下文切换方法

运行的任务切换为其它状态或其它状态切换成运行态,不仅仅需要状态表的变换,还需要堆栈进行上下文切换。所谓的上下文切换是指将当前任务的现场数据推入堆栈,将要运行任务的现场数据从堆栈里恢复。根据不同 CPU 以及片内存储器大小的差异用不同的方法建立堆栈,一种是为每个任务建立大小相同的任务堆栈,其容量按现场数据的最大值计算。另一种是每个任务的堆栈大小根据实际需要来确定。这 2 种方法都需要把每个任务的栈底或栈顶的位置保存在堆栈数组 stackPos 中。针对 2 种建立堆栈的方法,其上下文切换的方法也是不同的。

方法 1 是推入当前任务的现场数据后,根据栈底位置+(要运行的任务号 *每个任务堆栈占用的字节数),直接找到将要运行任务的堆栈地址,将堆栈指针指向该处即可推出数据。该种方法的优点是不需要移动其它任务的现场数据,其任务切换较快,缺点也很明显,内存浪费较大,所以它比较适合 TI 的 DSP2407 等大内存系统。

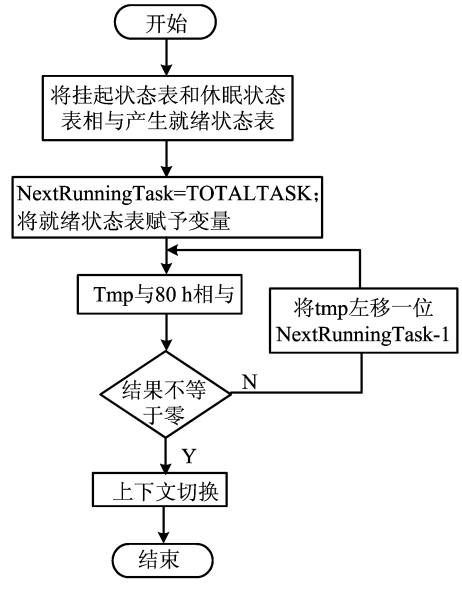


图 2 任务调度

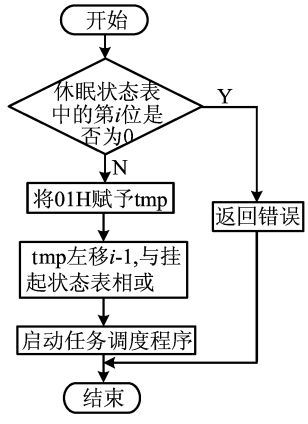


图 3 setReady 框图

方法 2 则将多余的内存保存在当前任务堆栈中，推入现场数据后，移动当前任务与将要运行任务之间所有的现场数据，使堆栈中的多余空间保存在将要运行的任务上，将堆栈指针指向将要运行任务的堆栈[4]。具体实现如图 4、图 5 所示。由于现场数据与 CPU 的型号有很大的关系，所以这一段需要用汇编语言编写[5]。使用方法 2 时内存利用率较高，但是由于需要移动多个任务的现场数据，切换速度较慢，比较适合 8051 等小内存系统。这 2 种方法在系统中是通过编译开关来实现选择编译的。

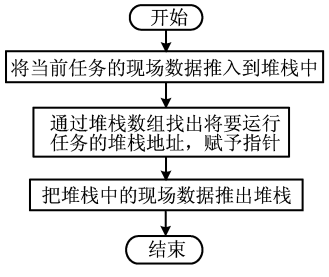


图 4　任务堆栈切换方法 1

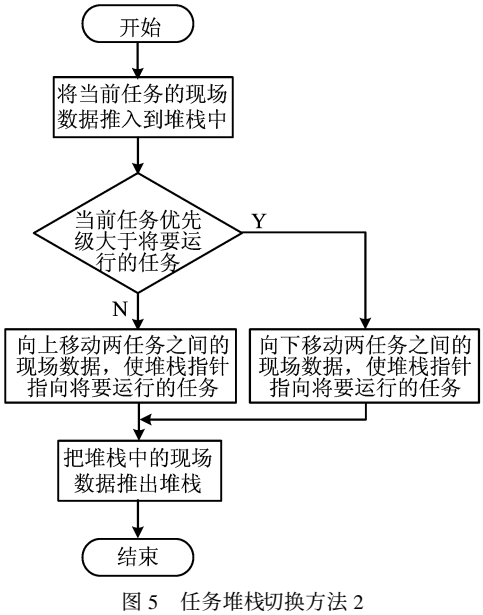


图 5　任务堆栈切换方法 2

### 1.3　优先级反转问题

对于共享设备与共享资源，信号量的操作是不可避免的。在进入共享资源前，任务必须获取一个信号量；一旦共享设备使用完成，那么该设备必须释放信号量[6]。其它想进入的任务必须等待，直到某个任务释放信号量。在信号量使用时经常会遇到优先级反转的问题。所谓优先级反转是指高优先级任务需要等待低优先级任务释放资源，而低优先级任务又正在等待中等优先级任务的现象叫做优先级反转。

举个例子，任务 1 优先级高于任务 2，任务 2 优先级高于任务 3。任务 1 和任务 2 处于挂起状态，等待某一事件的发生，任务 3 正在运行。此时，任务 3 要使用其共享资源。使用共享资源之前，首先必须得到该资源的信号量(Semaphore)。任务 3 得到了该信号量，并开始使用该共享资源。由于任务 1 优先级高，它等待的事件到来之后剥夺了任务 3 的 CPU 使用权，任务 1 开始运行。运行过程中任务 1 也要使用任务 3 正在使用着的资源，由于该资源的信号量还被任务 3 占用着，任务 1 只能进入挂起状态，等待任务 3 释放该信号量，任务 3 得以继续运行。

由于任务 2 的优先级高于任务 3，当任务 2 等待的事件发生后，任务 2 剥夺了任务 3 的 CPU 的使用权并开始运行，处理它该处理的事件，直到处理完之后将 CPU 控制权还给任务 3。

任务 3 接着运行，直到释放该共享资源的信号量。直到此时，实时内核知道有个高优先级的任务在等待这个信号量，内核做任务切换，使任务 1 得到该信号量并接着运行，在这种情况下，任务 1 优先级实际降到了任务 3 的优先级水平。因为任务 1 要等，等到任务 3 释放占有的共享资源。由于任务 2 剥夺任务 3 的 CPU 使用权，使任务 1 的状况更加恶化，任务 2 使任务 1 增加了额外的延迟时间。任务 1 和任务 2 的优先级发生了反转[1]。

为解决此问题，可采用优先级继承算法来实现，就是将任务 3 的优先级提高到任务 1 来。由于是在单片机上运行这个操作系统，为减少代码量及其内存，通过修改备份后的就绪状态表与堆栈位置数组 stackPos 来实现，就是当一个任务因为信号量进入挂起状态时，检测是否有低优先级的任务正在占用该信号量，如果有修改堆栈位置数组中的高优先级任务的堆栈位置指向，使其指向占用该资源低优先级任务的堆栈位置，实现优先级继承，在任务运行完后通过信号量释放来恢复原有的堆栈位置数组，恢复原有的优先级，重新设定休眠状态表与就绪状态表，具体如图 6、图 7 所示，该种方法只需占用极少的内存，很适合在 8051 这种小内存的系统上运行。另外在该操作系统中各个任务间的通信可通过信息队列或邮箱来实现，这与信号量的实现相似。需要指出的是上述介绍的都是操作系统的临界代码，进入临界代码需要关中断，完成临界代码后再打开中断，只有这样才能保证系统的正常运行[7,8]。

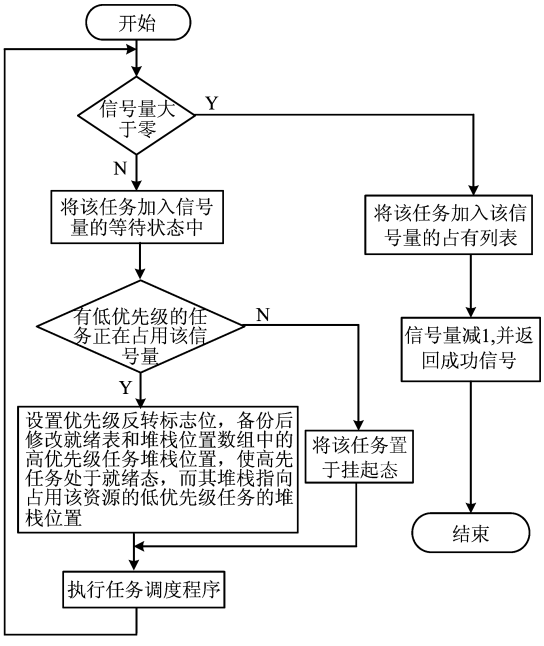


图 6 带有优先级继承的信号量挂起

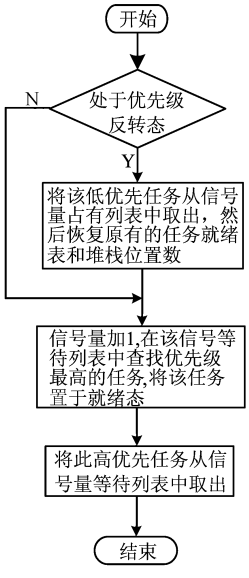


图 7 信号量释放

### 1.4 可重入函数

抢先式内核需要可重入的函数,所以在编写前,还需要了解编译环境是否易于产生可重入函数。如果采用 8051 的 C 语言编译器 KEil C 作为编译环境, 一般情况下会产生不可重入函数,因为它把局部变量也放在系统内存的固定位置中,就相当于全局变量一样,在这种情况下必须采取一些方法来产生一个可重入函数,例如减少函数自变量个数,使得 KEil C 将每个自变量放入寄存器,而不是放在内存中来产生可重入函数。

## 2 测试与结果

在 8051 系列单片机上使用该操作系统对房间温度湿度控制系统编写程序。该控制系统通过温度与湿度传感器检测房间中的实际温度,在出现偏差时通过空调与加湿机来保持房间温度湿度恒定,并显示温度与湿度的实际值。在此系统中将温度显示、湿度显示、温度控制与湿度控制分别作为控制任务,按照优先级从高到低写入到数组函数中。由于其中 2 个任务共用一个显示,所以必须使用信号量函数。通过在上下文切换与信号量函数中设置断点,观察挂起状态表与将要运行任务等变量,确定各个任务,可根据优先级的高低自动进行切换,堆栈中的现场数据推入与推出正确,温度与湿度可用 7 段代码依次显示。另外为测试优先级反转,修改了程序,将湿度显示与温度控制对调。在湿度显示时触发温度控制使其运行,通过在信号量等待函数中设置断点,这时观察到优先级发生转换,证明了优先级继承算法使任务的实时响应获得了极大的提高。由于该操作系统大小不到 3 k 字节,对硬件的要求极低,占用的系统资源较少,使该控制系统能够顺利运行,最关键的是由于在设计过程中使用 RTOS,设计调试时间由原来的 1 个月缩短为 1 周,提高了设计效率。

[参 考 文 献]

[1] Labrosse J J. 嵌入式实时操作系统 μC/OS[M]. 邵贝贝,译. 北京: 北京航天航空大学出版社, 2003: 120—125.

[2] 谭浩强. C 程序设计[M]. 第 2 版. 北京: 清华大学出版社, 2003: 102—105.

[3] 马忠梅, 籍顺心, 张 凯, 等. 单片机的 C 语言应用程序设计[M]. 第 3 版. 北京: 北京航空航天大学出版社, 2003: 45—47.

[4] 彭良清. μC/OS-II 任务堆栈处理的一种改进方法[J]. 单片机与嵌入式系统应用, 2008, (5): 115—120.

[5] 卡马尔. 嵌入式体系结构编程与设计[M]. 北京: 清华大学出版社, 2005: 89—98.

[6] Allworth S T. Introduction to real-time software design [M]. New York: Springer-Verlag, 1981: 31—32.

[7] Douglas C. Operating-system design: the XINU approach [M]. Englewood Cliffs, New Jersey: Prentice-Hall, 1984: 6—9.

[8] Wood M, Barrett T. A real-time primer[J]. Embedded Systems Programming, 1990, 3(2): 20—28.

(责任编辑 张 镅)