第 22卷第 6期 计算机应用与软件 Vol. 22, No. 6
2005年 6月 Computer Applications and Software Jun. 2005

RTX51嵌入式实时操作系统分析

蔡林骥 李清宝
(解放军信息工程大学信息工程学院计算机科学与技术系 河南 郑州 450002)

摘 要 传统的微控制应用大都采用结构化编程思想,对于单任务控制能很好地达到编程简单、思路清晰、开发周期短的要求。但面对任务较多、控制复杂的问题时,往往难以满足要求。为此,本文专门介绍了在嵌入式系统软件设计中引入现代操作系统的设计思想,并以实时操作系统 RTX51为例,对其原理和运行机制进行了较深入的分析。

关键词 RTX51 嵌入式操作系统 内核

THE ANALYSIS OF RTX51 EMBEDDED RTOS

Cai Linji Li Qingbao
(Department of Science & Technology, PLA University of Information Engineering, Zhengzhou Henan 450002, China)

Abstract The traditional structured programming can do well in the microcontrol application with single task. But it is not satisfying sometimes in the application with multi_task or complex control. This paper discuss the thinking that introduce modern operation system to the software design of embedded system. And, with the example of RTX51, we analyze its theory and give an example program.

Keywords RTX51 Embedded RTOS Kernel

0 引 言

在许多复杂的嵌入式系统应用中,不但要求能够及时响应随机发生的外部事件,并对其作出快速处理,通常还需要同时执行多个任务。对于这样的应用,采用实时操作系统 (Realtime Operating System, RTOS)作为软件基础平台是一个良好的选择。

RTOS中有一个核心,负责处理器专项任务,例如 CPU的分配与调度、寄存器上下文变换和存储器管理。核心的周围是完成 RTOS服务的例行程序库,它们执行各种系统级功能,在应用程序运行时发挥一定的作用。应用程序被分解为一组任务,RTOS调度器根据某些多任务调度算法让这些任务轮流占有 CPU。一个应用任务 (通常使用汇编语言以外的其他语言写成) 为了得到 RTOS服务,需调用相应的应用程序接口 (API)功能。RTOS及其 API库实际上掩盖了处理器的内部工作机理,因而应用软件开发人员不必太多地考虑实际使用的处理器。

下面以 Keil公司开发的嵌入式实时操作系统 RTX51 Tiny为例,简要地分析其原理和工作机制,并举例说明其在实时多任务控制中的应用。

1 Keil RTX51实时操作系统简介

RTX51是德国 Keil公司开发的一种应用于 MCS51系列单片机的实时多任务操作系统,它可以工作在所有 8051单片机以及派生家庭中。它有两个版本,RTX51 Full和 RTX51 Tiny。

RTX51 Full支持事件驱动的抢先式多任务调度策略,允许4个优先权任务的循环和切换,并且还能并行的利用中断功能。它支持信号传递,以及与系统邮箱和信号量进行消息传递。RTX51 Full的 os_wait函数可以等待以下事件:中断、时间到、来自任务或中断的信号、来自任务或中断的消息、信号量。

RTX51 Tiny是 RTX51 Full的一个子集。RTX51 Tiny可以很容易地运行在没有扩展外部存储器的单片机系统上。但是,使用 RTX51 Tiny的程序可以访问外部存储器。RTX51 Tiny允许最大 16个任务循环切换,并且支持信号传递,还能并行的利用中断功能。但它不能进行消息处理,不支持存储区的分配和释放,不支持抢先式调度。RTX51 Tiny是一个很小的内核,完全集成在 Keil C51编译器中。更重要的是,它仅占用 800字节左右的程序存储空间。RTX51 Tiny的 os_wait函数可以等待以下事件:时间到、时间间隔、来自任务或者中断的信号。

2 Keil RTX51 Tiny实行操作系统分析

2.1 RTX51 Tiny任务状态

RTX51 Tiny的用户任务具有以下几个状态。

® 运行 (RUNNING):任务正处于运行中:同一时刻只有一个任务可以处于“RUNNING”状态。

® 就绪 (READY):等待运行的任务处于“READY”状态。在当前运行的任务退出运行状态后,就绪队列中的任务根据调度策略被调度执行,进入到运行状态。

® 阻塞 (BLOCKED):等待一个事件的任务处于“BLOCKED”状态。如果等待的事件发生,则此任务进入“READY”状态,等待被调度。

® 休眠 (SLEEPING):被申明过但没有开始运行的任务处于休眠状态。运行过但已经被删除的任务也处在休眠状态中。

® 超时 (TIMEOUT):任务由于时间片用完而处于“TIMEOUT”状态,并等待再次运行。该状态与“READY”状态相似,但

收稿日期:2004 - 04 - 28。蔡林骥,硕士生,主研领域:数字系统设计自动化。

由于是内部操作过程使一个循环任务被切换,因而单独算作一个状态。

处于“READY/TIMEOUT”、“RUNNING”和“BLOCKED”状态的任务被认为是激活的状态,它们的转换关系如图 1。“SLEEPING”状态的任务是非激活的,不能被执行或认为已经终止。

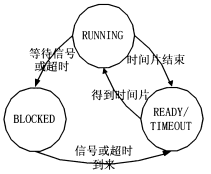


图 1　用户任务状态转换图

2.2　同步机制

RTX51 Tiny内核用以下事件进行任务间的通信和同步。

® 超时(TIMEOUT):由 os_wait函数调用引发的时间延时,其持续时间可由定时节拍数确定。带有 TIMEOUT值调用 os_wait函数的任务将被挂起,直到延时结束,才返回到“READY”状态。

® 间隔(INTERVAL):由 os_wait函数调用引发的时间间隔,其间隔时间可由定时节拍数确定。带有 INTERVAL值调用 os_wait函数的任务将被挂起,直到间隔时间结束,然后返回到 READY状态。与 TIMEOUT不同的是,任务的节拍计数器不复位,典型应用是产生时钟。

® 信号(SIGNAL):系统定义的位变量,可以由系统函数置位或清除。可以调用 os_wait函数暂停一个任务并等待从另一个任务发出的信号,这可以用于协调两个或更多的任务。如果一个任务在等待一个信号并且信号标志为 0,则在收到这个信号之前,这个任务将一直处于挂起状态。如果信号标志已经被置 1,则当任务查询信号时,信号标志会被清除,任务将可以被继续执行。

2.3　调度策略

RTX51 Tiny采用时间片轮转算法,不支持任务优先级策略。时间片非常短,通常只有几个毫秒。一个时间片的持续时间也可以预先用配置变量 TIMESHARING定义。系统每次调度时,把 CPU分配给一个就绪的任务,并令其执行一个时间片,构成微观上轮流运行、宏观上并行执行的多任务效果。

如果出现以下情况,则当前运行任务中断:

(1) 任务调用 os_wait函数并且指定事件没有发生;

(2) 任务运行时间超过定义的时间片轮转超时时间。

如果出现以下情况,则开始另一个任务:

(1) 没有其他的任务运行;

(2) 将要开始的任务处于“READY”或“TIMEOUT”状态。

2.4　存储器管理

RTX51 Tiny利用 Keil C51编译器对全局变量和局部变量静态分配存储空间,将存储器管理转化为堆栈管理。每个任务都保留一个单独的堆栈区,全部堆栈管理都在 IDATA空间进行。为了给当前正在运行的任务分配尽可能大的栈区,各个任务所用的堆栈位置是动态的。每个任务都有各自的数据结构记录所用的堆栈位置。当堆栈自由空间小于 FREESTACK(默认为 20)个字节时,就会调用宏 STACK_ERROR,进行堆栈出错处理。

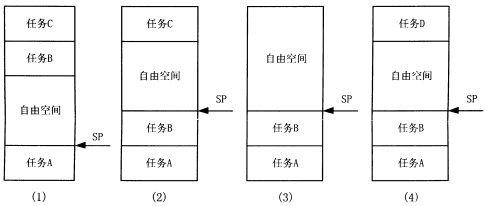
以下情况会启动堆栈管理:

(1) 任务切换,将全部自由堆栈空间分配给正在运行的任务;

(2) 任务创建,将自由堆栈空间的 2个字节分配给新创建的任务;

(3) 任务删除,回收被删除任务的堆栈空间,并转换为自由堆栈空间。

堆栈管理如图 2所示。



(1)任务 A正在运行;(2)切换到任务 B运行;
(3)删除任务 C后,自由空间增加;
(4)创建任务 D后自由空间减少 2字节

图 2　堆栈管理

2.5　中断处理

RTX51 Tiny提供两种方法来处理中断:一种是 C51中断函数,包括软件中断和硬件中断。对于硬件中断,用户不能再使用定时器 T0。当中断发生的时候,程序跳到相应的中断处理函数。中断处理过程与正在运行的任务是相互独立的,即中断处理过程在 RTX51系统内核之外和任务切换规则没有关联。但中断处理过程可以同 RTX51任务互发信号或交换数据;另一种是 RTX51的任务中断。如果中断发生,等待中断的任务就从“等待”状态进入到“就绪”状态,并按照任务切换规则进行切换。这种中断处理是完全集成在 RTX51内部,与硬件中断事件的处理以及信号、消息的处理是完全相同的。

两种方法可以在应用程序中混合使用。在系统响应时间上 C51中断函数是最快的。RTX51必须完全控制中断使能寄存器,这样才能遵守任务的切换规则并保证中断程序的无误进行。必须注意中断使能寄存器是由 RTX51完全控制的,用户不能在程序中修改,否则可能会影响内核的正常运行。

RTX51使用 8051内部定时器 T0来产生定时节拍,是任务切换、时间片轮转的依据。内核中的定时器 T0中断服务例程有三个主要任务:

(1) 更新任务节拍数。每个任务都有一个定时节拍计数器变量,在每一次定时节拍中断中都自减一次。

(2) 检查自由堆栈空间。如果自由堆栈空间范围小于 FREESTACK(默认为 20字节,用户可以自定义)时,则调用宏 STACK_ERROR,进入堆栈出错处理。

(3) 检查当前处于“RUNNING”状态任务的时间片是否到时。如果超时,则将程序转到任务切换程序段(taskswitching)切换下一任务运行。

T0中断服务例程流程如图 3。

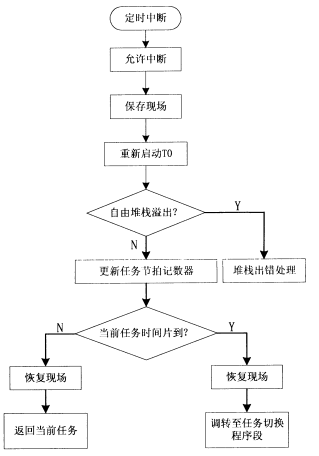


图 3　T0中断服务流程图

2.6　任务切换

RTX51 Tiny的任务切换是整个内核中最核心的部分。它共有两个入口 TASKSWITCHING和 SWITCHINGNOW。前

者供定时器 T0的中断服务程序调用,后能供系统函数 os_delete 和 os_wait调用。相应地也有两个不同的出口。

系统首先将当前任务置为“TIMEOUT”状态,等待下一次时间片循环。然后找到下一个处于“READY”状态的任务,通过堆栈管理,将自由堆栈空间分配给该任务,使其成为当前任务。清除使该任务进入“READY”或“TIMEOUT”状态的相关位后,执行该任务。

3 RTX51 Tiny应用举例

首先对例子中用到的函数作一简要说明。RTX51 Tiny提供的 os_create_task函数和 os_delete_task函数用于创建和删除任务。os_wait函数可以停止当前任务,等待一个或几个事件,比如一个时间间隔、一个超时或从一个任务或中断发送给另一个任务或中断的信号。系统定义的事件常数有 K_IVL(等待一个报时信号间隔)、K_SIG(等待一个信号)、K_TMO(等待一个超时 TIMEOUT)。os_send_signal函数向指定 id的任务发送一个信号。如果等待中的任务收到这个信号,就会转入 READY状态。信号保存在任务的信号标志位中。

下面来看一个简单的例子。程序被划分为 4个任务,任务 0完成初始化工作,它负责创建了任务 1和 2,然后删除自己。任务 3在任务 2的首次运行中被创建。任务 1更新记数器后,被 os_wait函数挂起,需要等待 5个时钟滴答后才能再次被调度执行。任务 3被创建后并不能马上执行,它需要等待任务 2发来的信号。在收到信号后,任务 3可以被调度执行,执行时信号被清除,所以下一次执行还需要收到新的信号,幸好任务 2每隔一段时间便会向任务 3发送一次信号。该程序在 KeilC51 V7上编译运行通过,清单如下。

```
#include〈rtx51tiny. h〉/ * RTX-51 tiny functions & defines * /
unsigned long counter1, counter2, counter3, count;
void job0 (viod) _task_ 0 {
    os_create_task (0);
    os_create_task (1);
    os_create_task (2);
    os_delete_task (0);
}
void job1 (void) _task_ 1 {
    while (1) {
    counter1 ++;
    os_wait (K_TMO, 5);
  }
}
void job2 (void) _task_ 2 {
  os_create_task (3);
  while (1) {
    counter2 ++;
    count ++;
    if ((count & 0xFF) == 0) {
      count = 0;
      os_send_signal (3);
    }
  }
  }
  void job3 (void) _task_3 {
    while (1) {
      os_wait (K_SIG, 0);
      counter3 ++;
  }
}
```

4 结束语

在许多复杂的嵌入式应用中,实时性和并行性是非常重要的,但两者之间有一定的矛盾。协调实时性和并行性的重要手段就是在软件设计中引入现代操作系统思想,对有限的资源和处理器进行合理的分配和调度。实践证明在引入 RTX51 Tiny实时操作系统后,软件开发周期缩短,程序结构更加清晰,系统实时性和并行性大大增强,开发出的程序具有较高的可维护性和可移植性。

参 考 文 献

[1]《RTX51 Real-time Kernel》http://www.keil.com/rtx51/default.htm.
[2]徐爱钧,单片机高级语言 C51 Windows环境编程与应用,电子工业出版社.
[3]刘明路、王亮生、李世煜,“基于 RTX51的单片机软件设计”,《单片机与嵌入式系统应用》,2002.12.

(上接第 13页)

表 1 部分字母手势的计算结果

标准手势		a	b	c	d	e
输入手势	a	0.0125	0.3261	0.4043	0.3241	0.6352
	b	0.3398	0.0236	0.2519	0.5624	0.6409
	c	0.4011	0.2174	0.0147	0.1234	0.6241
	d	0.3171	0.5358	0.1447	0.0253	0.6127
	e	0.6363	0.6452	0.6245	0.6135	0.0445

(Inter P4 1.8G CPU,256MB DDR),软件环境为 Windows 2000、Visual C++。实验结果表明,归一化的傅立叶描述子能够更加鲁棒地识别和区分具有旋转、平移和尺度变化的手势图像,其识别率达到 89.6%,是快速识别和分析物体形状的一种有效方法。

5 结束语

本文利用八邻域搜索算法提取了手势边界点的坐标,利用傅立叶描述子鲁棒提取了手势图像的特征向量,并利用欧氏距离进行了模式匹配。实验对中国手语中 30个字母手势进行了识别,实验结果表明傅立叶描述子具有旋转、平移和尺度不变性,并且与曲线的起点位置的选择无关。傅立叶描述子具有简单、高效的特点,已经成为识别物体形状的重要方法之一,但同时它也具有计算量大等缺点,需要我们进一步研究。

参 考 文 献

[1]王林泉、章文怡、郑刚,“区域特征的乐谱识别系统”,《软件学报》,1994,5(11):44.
[2]何海峰、王林泉、葛元,“灰度图像中基于像素分布特征的人脸定位”,《计算机工程》,2002,28(6):158～160.
[3]Vladimir Pavlovic, R. Sharma, T. Huang Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. IEEE PAMI, 1997, 19(7):156.
[4]Starner T. Pentland A., Real-Time American Sign Language Recognition from Video Using Hidden Markov Models MIT Media Lab, 1995, TR: 375.