单片机系统中的多任务多线程机制的实现

■ 武汉化工学院 冉全 章涤峰

摘 要 单片机系统的开发很多情况下不是在嵌入式操作系统平台上进行的，而是直接基于处理器编写。在多任务并行执行的要求下，可以借鉴操作系统中的任务和线程机制，对资源和处理器合理进行调度。本文以实例对此进行讨论。

关键词 单片机 任务 线程 并行处理

引 言

首先要指出的一点是，我们不是讨论嵌入式实时多任务操作系统（RTOS）的设计。我们讨论的是，在不使用RTOS的控制系统中，如何体现多任务多线程机制的程序设计思想。

一些嵌入式设备可能需要操作系统，例如掌上电脑、PDA、网络控制器等高性能的手持设备和移动设备。它们往往和无线通信、互联网访问和多媒体处理等复杂而强大的功能联系在一起；对CPU要求也很高，往往是以通用CPU为原型的各种高端嵌入式处理器。

作为一个完整的操作系统，RTOS有一个可靠性很高的实时内核，将CPU时间、中断、I/O、定时器等资源都包装起来，留给用户一个标准的应用程序接口（API）；根据各个任务的优先级，合理地在不同任务之间分配CPU的时间，保证程序执行的实时性、可靠性。内核一般都能提供任务调度与管理、时间管理、任务间同步与通信、内存管理和中断服务等功能，部分高档商业化产品，如Windows XP Embedded，甚至支持32位地址空间、虚拟存储管理、多进程以及嵌入式操作系统中不多见的动态链接库（DLL）[1]。对于这些RTOS来说，多任务实时处理不是件困难的事情。

但更多的情况下，用户使用的是另一类CPU——微控制器，即单片机，往往是按照某一流程执行单一任务。出于成本和技术上的原因，这类软件开发多数还是基于处理器直接编写，没有选配实时多任务操作系统作为开发平台，也不需要将系统软件和应用软件分开处理。但是在实际应用中，有时也会面临同时处理多个并行任务的要求，这就需要安排一种运行机制，来模拟RTOS中的处理办法。

1 RTOS中的设计思想

单处理机多道程序系统具有如下特征：

① 从宏观上看，几道程序“同时运行”。即它们先后开始了各自的运行，且均未结束。

② 从微观上看，几道程序“交替执行”。对于单处理机系统而言，它们只能轮流地占用CPU[2]。

其实质是指几道程序在处理机中“交替执行”。我们按照现在常用的方法，把一道程序和一个任务对应，把任务中的每个分开的、独立执行的部分称之为线程。

具体到RTOS来说，一方面，实时操作中的多任务引起的并发性和实时性，要求操作系统对资源分配具有更强的控制能力。通常的办法是采取设立前台与后台两个作业的分配办法。前台作业中包含实时采集、控制、处理有关的任务，任务优先级较高；后台作业一般是对数据进行分析、输出数据、响应操作员请求等任务，优先级较低。后台作业中的任务只能在前台任务运行的间隙运行。前台作业与后台作业并非完全孤立的：后台作业所需数据由前台作业存储在共享内存区内，作业之间通过共享存储区进行数据交换。

另一方面，实时任务总是由某个事件发生或时间条件满足来激活。事件有两种：内部事件和外部事件。时间驱动也有两种：按绝对时间驱动和按相对时间驱动。内部事件驱动是指某一程序运行的结果导致另一任务的启动，这个结果可能是数据满足一定条件，也可能是释放了某一资源；而最典型的实时任务是由外部事件驱动的。在实时系统中，外部事件发生有时是不可预测的，由外部事件驱动的

任务一般是需要立即执行的任务，它的优先级最高。绝对时间驱动是指在某指定时刻执行的任务，也就是在自然时钟的绝对时间执行。相对时间驱动是指周期性执行的任务，总是相对上一次执行时间计时，执行时间间隔一定。除了周期性任务外，还有一些同步任务也可能由相对时间驱动，如等待某种条件到来。等待时间是编程设定的。相对时间可用计算机内部时钟或软时钟计时。

我们在实际设计当中，这两方面的问题都有所体现，所有的事件驱动和时间驱动都体现在设置相应的任务标识和线程标识。从后面的讨论中可以看出，当硬件环境一定时，依据这些标识，通过安排系统内中断响应方式和调整任务调度算法，可以有效解决多任务并行问题，因为系统的实时性主要取决于这两点[3]。

2 多任务多线程机制的实现

我们设计的对象是双通道和四通道测试的某型医用检验设备。每个通道可以置入样本，设置不同的测试项目，完成测试后输出不同的测试结果和附加的计算结果。

常规的处理方法是这样的：各个通道只能测试同一个项目，按统一步骤同步执行各任务的相同阶段，其处理示意如图1。为简化起见，我们用双通道进行说明。

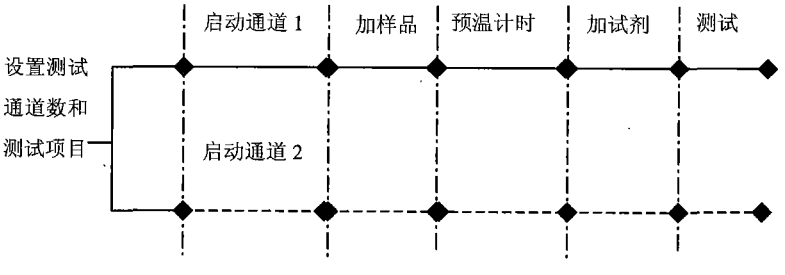

图1 多任务简单重叠

显然，这样做不仅会失去测试的灵活性，例如不能同时测量不同项目，不能随意在不同通道中测试不同样本，即使有空余通道也不能在上一样本测试过程中启动下一样本的测试；而且还牺牲效率，浪费时间，因为要等每个阶段最慢的一个处理完才能进入下一阶段。这其实是单任务的多次简单重复，设计也容易。国内很多类似产品采用了这种方案，但我们放弃了。

我们选择了完全并行的设计，即要求所有通道可以完全独立工作；任意启动和停止；彼此没有约束；时间上可以任意重叠；是几个独立的任务，如图2。

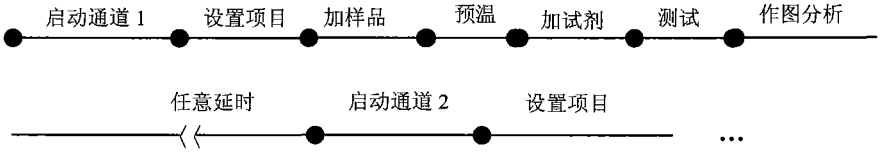

图2 独立并行任务

这里我们把每一个启动通道进行测试的程序叫做一个任务，把各自任务下的每一个单独的、分开处理的程序段叫做一个线程，每个线程依靠自己的标识来识别。一个通道的测试任务可分为启动、设置、加样品、预温计时、加试剂与搅拌、通道轮流采样、数据处理和作图打印等多个线程。另外，有一个温度的实时监控独立线程，它的优先级要次于通道的测试采样。

这些线程可分属于前台和后台两类：前台主要是一些中断的处理，例如两路温度的实时监控、每100ms内的各通道循环检测一遍、采用中断方式的键盘干预等；后台主要是扫描方式下的响应操作员的按键请求、数据处理、图形显示、打印报告等内容。

整个实现机制可以简单地概括如下：前台通过合理安排中断的响应和服务方式来对多个任务的实时线程进行处理；后台操作主要以循环方式扫描各个任务的线程标识，满足条件的线程被激活予以处理。

限于篇幅，不可能详细介绍整个设计方案，在此只能给出各测试通道工作任务的前台和后台线程划分及流程，供参考。然后，给出一个中断退出后返回到任意地址的函数，它比C51自己的setjmp和longjmp全程跳转函数的使用要方便很多。实时任务中，中断服务结束后不是返回到断点地址执行原有程序，而是强制返回到某一地址执行新程序的情况非常普遍。我们采用设置环境变量的方法，使中断退出后可以任意返回到多个设置入口中的某一个去执行，有效地解决了前台和后台任务线程的灵活切换这一关键问题。我们使用的CPU是97C52，编程语言为Keil C51 6.0版。

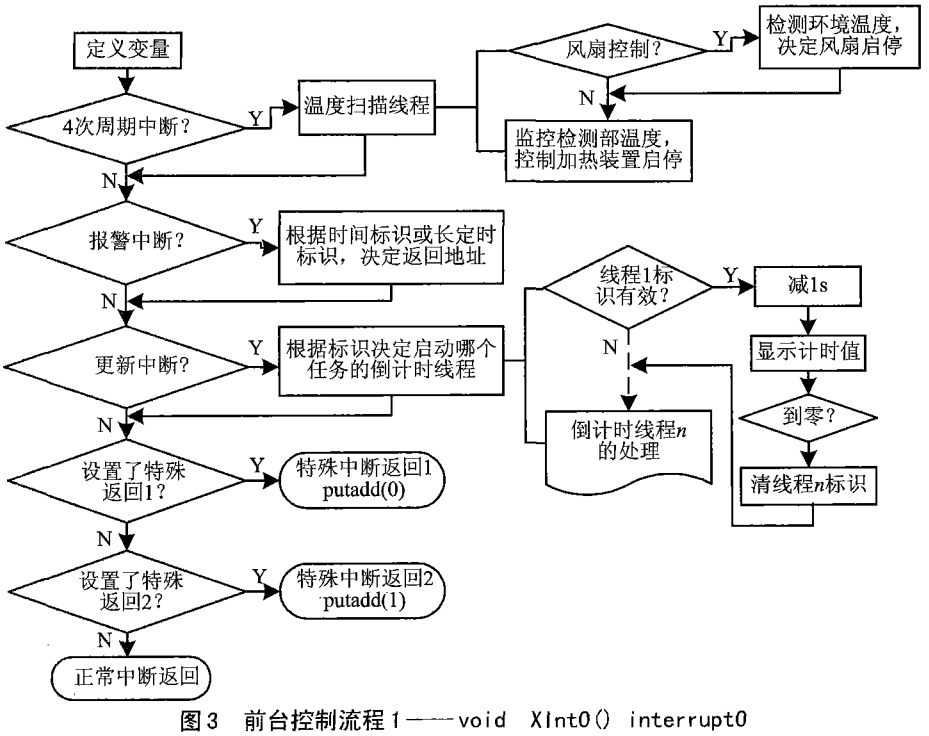

图3 前台控制流程1——void XInt0() interrupt0

图3是主定时器中断服务，12C887提供中断请求信号至int0。12C887的三个中断触发服务中，温度扫描是独立线程，四次500ms“周期中断”（即每2s）后执行一遍；需要屏幕显示预温倒计时的时候使用“更新中断”，每秒一次，各测试任务，其倒计时线程依靠各自的标识启动和停止；“报警中断”需要时设置为每分钟1次，用于主菜单界面显示当前时间和长定时的返回。

图4是CPU内部定时器0的中断服务，用于A/D转换。每个测试任务的A/D分为两个线程：检测试剂加入和测试试剂样品的反应曲线，虽然都是通过对光学传感器的输出进行检测的，但处理方法完全不同，数据量也很不一样。定时器0设定为每100ms中断1次，因为采用高精度Σ-Δ转换器件，CPU必须直接控制器件的整个转换过程，所以，要注意所有通道轮扫一遍A/D的时间不能超过100ms。

图5为后台流程。后台程序依靠通道按键启动一个测试任务，然后进行该任务预处理，类似初始化的一些功能。如果这期间又启动别的任务，则未初始化完的先前任务中止。

初始化完成后进入多任务所属线程的循环处理阶段，其间可以随时由通道按键引起的中断来加入新的任务，每个线程的调度标识可以由相关的前台线程给出，也可来自相关的后台线程。配合Getadd()和Putadd()从中断强制返回某地址后，使用跳转语句到真正的目标地址。

最后给出强制返回程序代码（供参考）：

```
/* 保存当前地址信息到环境变量JMPEnv[env1][]中，每个变量由三项组成，env1是二维下标参数 */
void getadd(unsigned char env1)
{   unsigned char temp;
    temp=SP;
    JMPEnv[env1][0]=(*((unsigned char idata*)SP));
    temp--;
    JMPEnv[env1][1]=(*((unsigned char idata*)temp));
    JMPEnv[env1][2]=SP-2;
}
```

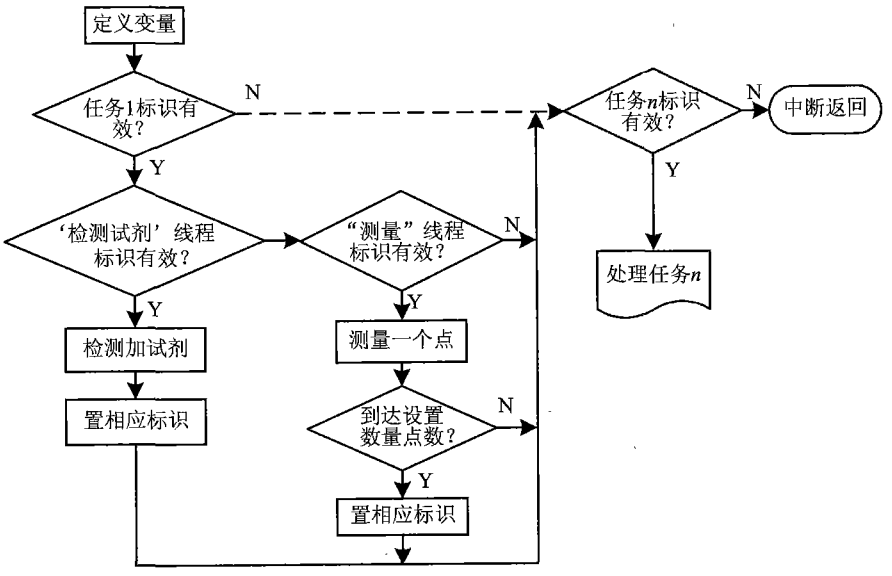

图4 前台控制流程2——void time0()interrupt 1

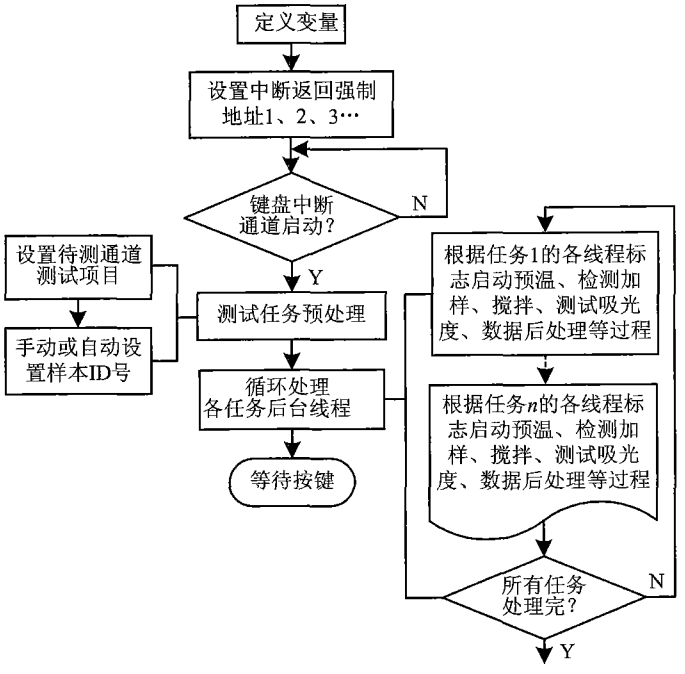

图5 后台分析流程——void analyze()

```
/*置中断返回的任意跳转地址*/
void putadd(unsigned char env1) reentrant
{   unsigned char temp[15]; char i;
    /*下面保存进入中断程序时的压栈值*/
    for(i=0;i<15;i++)
    {  temp[i]=(*((unsigned char idata*)SP));
    SP--)
    }
    /*放置新地址*/
    SP=JMPEnv[env1][2]; SP++;
    (*((unsigned char idata*)SP))=JMPEnv[env1][1]; SP++;
    (*((unsigned char idata*)SP))=JMPEnv[env1][0];
    /*恢复中断开始时的那些压栈值*/
    for(i=14;i>=0;i--)
    {   SP++;
        (*((unsigned char idata*)SP))=temp[i];
    }
}
```

结 语

限于篇幅，不可能详述任务、线程和标识的细节，仅提出一种单片机等嵌入式控制系统对多任务进行实时处理的一种思路；借鉴了主流操作系统中的多任务和多线程机制。实践证明，这种想法是行之有效的，并且取得了很好的效果。

虽然我们研制的系统是对多个相同的任务进行并行处理，但该种设计方法应该可以推广到多种不同性质的实时任务的并行处理当中去。

参考文献

1 刘艺平，陈宏林. 嵌入式操作系统的航向. 微电脑世界，2002(1)

2 朱继生，宗大华. 最新操作系统教程. 北京：电子工业出版社，1997

3 蔡德聪. 工业控制计算机实时操作系统. 第1版. 北京：清华大学出版社，1999

（收稿日期：2003-01-03）

新书介绍

现场总线CAN原理与应用技术

饶运涛 邹继军 郑勇芸 编著

北京航空航天大学出版社出版 出版日期：2003年6月

CAN是一种具有国际标准而且性能价格比较高的现场总线，它在当今自动控制领域的发展中发挥着重要的作用。本书内容包括：计算机网络技术与现场总线的基本原理概念和它们之间的密切关系；CAN 2.0规范和几种功能典型且流行的CAN控制器和驱动器的详细资料；在作者实验和开发应用CAN技术的成果基础上，详细介绍了CAN的应用开发技术，从硬件的设计到各个基本软件功能模块的编写，其中包括CAN控制器与单片机、PC机不同方式的接口技术等。这些资料可供读者直接参考使用（含源程序代码从汇编语言到Windows下的VxD和DLL），以便尽快进入实践阶段。书中还介绍了作者已完成并投入使用的一个CAN系统设计的实例。本书力求理论密切联系实际，重点突出，学以致用。主要对象是现场总线CAN的初学者，也可以作为大专院校电子技术和自控专业类师生的参考书以及相关专业技术人员的培训材料。