KEIL RTX51 TINY内核的分析与应用

■ 河海大学 刘玉宏

摘 要 简要介绍RTX51 TINY的基本情况和使用方法；详细分析这个内核的任务管理和内存管理的运行机制，并给出其主要代码流程图。

关键词 单片机 实时操作系统 RTX51

1 RTX51简介

1.1 RTX51 TINY特性

RTX51是KEIL公司开发的用于8051系列单片机的多任务实时操作系统。它有两个版本，RTX51 FULL和RTX51 TINY。

RTX51 TINY是RTX51 FULL的子集，仅支持按时间片循环任务调度，支持任务间信号传递，最大16个任务，可以并行地利用中断。具有以下等待操作：超时、另一个任务或中断的信号。但它不能进行信息处理，不支持存储区的分配和释放，不支持占先式调度。RTX51 TINY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，可以在没有外扩数据存储器的8051系统中运行，但应用程序仍然可以访问外部存储器。RTX51 TINY下文简称为内核。

1.2 RTX51 TINY的使用

内核完全集成在KEIL C51编译器中，以系统函数调用的方式运行，因此可以很容易地使用KEIL C51语言编写和编译一个多任务程序，并嵌入到实际应用系统中。内核提供以下函数供应用程序引用：

① char os_create_task (task_id);

② char os_delete_task (task_id);

③ char os_send_signal (task_id);

④ char isr_send_signal (task_id);

⑤ char os_clear_signal (task_id);

⑥ char os_running_task_id (void);

⑦ char os_wait (event_sel,ticks,dummy)。

各函数的函数原型和具体意义，可以阅读参考文献[1]。

2 RTX51 TINY内核分析

2.1 任务状态

RTX51 TINY的用户任务具有以下几个状态。

◇ RUNNING：任务处于运行中，同一时间只有一个任务可以处于“RUNNING”状态。

◇ READY：任务正在等待运行，在当前运行的任务时间片完成之后，RTX51 TINY运行下一个处于“READY”状态的任务。

◇ WAITING：任务等待一个事件。如果所等待的事件发生的话，任务进入“READY”状态。

◇ DELETED：任务不处于执行队列。

◇ TIME OUT：任务由于时间片用完而处于“TIME OUT”状态，并等待再次运行。该状态与“READY”状态相似，但由于是内部操作过程使一个循环任务被切换而被冠以标记。

图1所示为任务状态转换图。

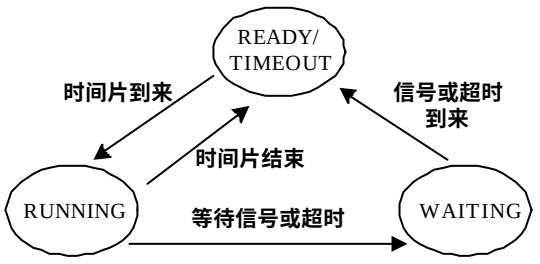


图1 任务状态转换图

2.2 同步机制

为了能保证任务在执行次序上的协调，必须采用同步机制。内核用以下事件进行任务间的通信和同步。

① SIGNAL：用于任务之间通信的位，可以用系统

函数置位或清除。如果一个任务调用了 os_wait 函数等待 SIGNAL 而 SIGNAL 未置位，则该任务被挂起直到 SIGNAL 置位，才返回到 READY 状态，并可被再次执行。

② TIMEOUT：由 os_wait 函数开始的时间延时，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，并可被再次执行。

③ INTERVAL：由 os_wait 函数开始的时间间隔，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，并可被再次执行。与 TIMEOUT 不同的是，任务的节拍计数器不复位。

2.3 调度规则

RTX51 TINY 使用 8051 内部定时器 T0 来产生定时节拍，各任务只在各自分配的定时节拍数（时间片）内执行。当时间片用完后，切换至下一任务运行，因此，各任务是并发执行的。

调度规则如下：如果①任务调用了 os_wait 函数，且特定事件还没有发生，②任务执行比循环切换所规定的时间长，则运行任务被中断；如果①没有其它任务正在运行，②任务处于“READY”或“TIMEOUT”状态下等待运行，则另一个任务开始。

2.4 任务控制块

为了能描述和控制任务的运行，内核为每个任务定义了称作任务控制块的数据结构，主要包括三项内容：

① ENTRY[task_id]：task_id 任务的代码入口地址，位于 CODE 空间，2 字节为一个单位。

② STKP[taskid]：taskid 任务所使用堆栈栈底位置，位于 IDATA 空间，1 字节为一个单位。

③ STATE[taskid].timer 和 STATE[tasked].state：前者表示任务的定时节拍计数器，在每一次定时节拍中断后都自减一次；后者表示任务状态寄存器，用其各个位来表示任务所处的状态。位于 IDATA 空间，以 2 字节为一单位。

2.5 存储器管理

内核使用了 KEIL C51 编译器的对全局变量和局部变量采取静态分配存储空间的策略，因此存储器管理简化为堆栈管理。内核为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，所以各个任务所用的堆栈位置是动态的，并用 STKP[taskid] 来记录各任务的堆栈栈底位置。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。

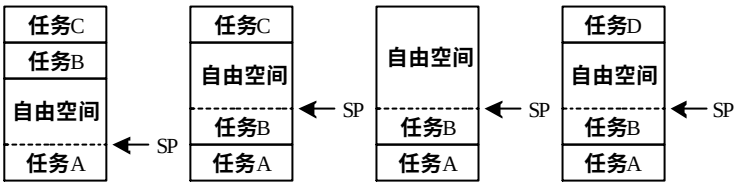
在以下情况会进行堆栈管理：

◇任务切换，将全部自由堆栈空间分配给正在运行的任务；

◇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 task_id，并将 ENTRY[task_id]，放入其堆栈；

◇任务删除，回收被删除的任务 task_id 的堆栈空间，并转换为自由堆栈空间。

堆栈管理如图 2 所示。



注：(a) 任务 A 正在运行；(b) 切换至任务 B 运行；(c) 删除任务 C 后，自由空间增加；(d) 创建任务 D 后，自由空间减少 2 字节

图2 堆栈管理

3 代码分析

内核代码用汇编语言写成，可读性差，但代码效率较高，主要由两个源程序文件 conf_tny.a51 和 rtxtny.a51 组成。前者是一个配置文件，用来定义系统运行所需要的全局变量和堆栈出错的宏 STACK_ERROR，这些全局变量和宏，用户都可以根据自己的系统配置灵活修改；后者是系统内核，完成系统调用的所有函数。

3.1 主程序 main

主程序 main 的主要任务是初始化各任务堆栈栈底指针 STKP、状态字 STATE 和定时器 T0，创建任务 0 并将其导入运行队列。这个过程加上 KEIL C51 的启动代码 CSTARTUP 正是一般嵌入式系统中 BSP 所作的工作。

3.2 定时器 T0 中断服务程序

内核使用定时器 T0 作为定时节拍发生器，是任务切换、时间片轮转的依据。中断服务程序有三个任务。

① 更新各个任务节拍数：将 STATE[taskid].timer 减 1，如果某任务超时（STATE[taskid].timer = 0），并且该任务正在等待超时事件，则将该任务置为“READY”状态，使其返回任务队列。

②检查自由堆栈空间：若自由堆栈空间范围小于 FREESTACK（默认为 20 字节）时，可以调用宏

STACK_ERROR，进行堆栈出错处理。

③ 检查当前任务（处于RUNNING状态）的时间片是否到时。若当前任务的时间片到时，将程序转到任务切换程序段（taskswitching）切换下一任务运行。

程序流程如图3所示。

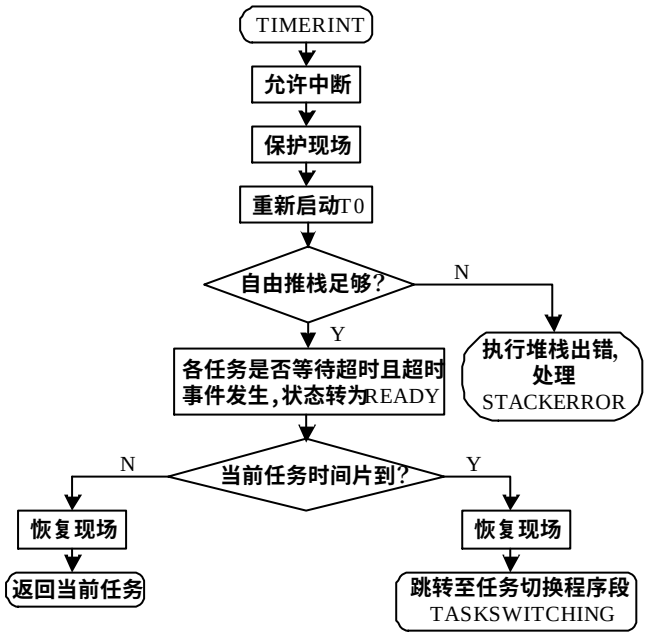


图3 T0中断服务程序

3.3 任务切换程序段

这个程序段是整个内核中最核心的一个，主要功能是完成任务切换。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。相应也有两个不同的出口。

其基本工作流程是首先将当前任务置为“TIMEOUT”状态，等待下一次时间片循环，其次找到下一个处于“READY”状态的任务并使其成为当前任务。然后进行堆栈管理，将自由堆栈空间分配给该任务。清除使该任务进入“READY”或“TIMEOUT”状态的相关位后，执行该任务。流程框图如图4所示。

3.4 os_wait 程序段

主要完成os_wait函数。任务调用os_wait函数，挂起当前任务，等待一个或几个间隔（K_IVL）、超时（K_TMO）、信号（K_SIG）事件。如果所等待的事件已经发生，继续执行当前任务；如果所等待的事件没有发生，则置相应的等待标志后，挂起该任务，转任务切换程序段（switchingnow）切换到下一任务。

3.5 其它程序段

其它程序段主要完成os_create_task、os_delete_task

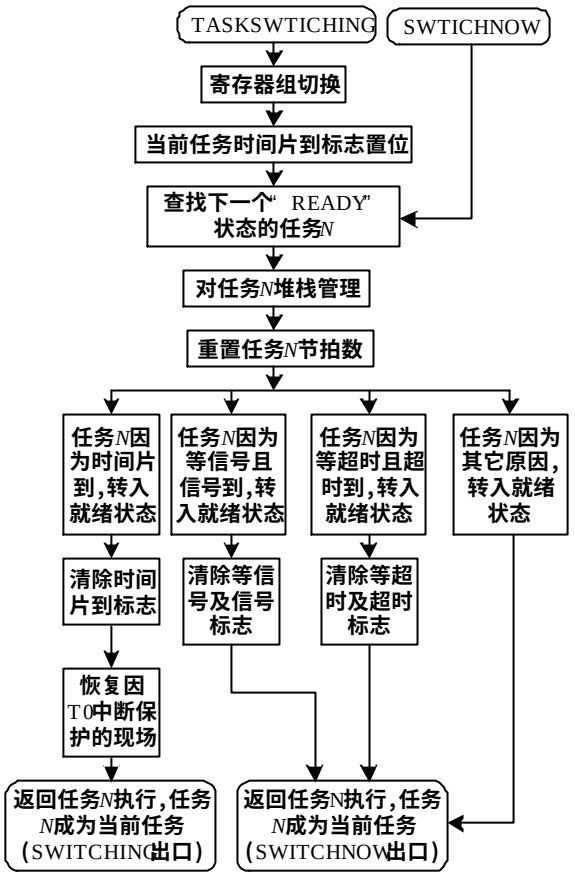


图4 任务切换程序

函数和有关信号处理的os_send_signal、isr_send_signal、os_clear_signal函数。这些函数功能相对比较简单，主要是根据上述存储器管理策略进行堆栈的分配和删除，并改变任务的状态字STATE[tasked].state，使任务处于不同的状态。

以上所有程序段，若涉及到任务状态字操作，必须关中断，以防止和定时器T0同时操作任务状态字。

结 语

以上分析可以看到这个内核简洁高效，非常适合于运行在资源较少的单片机上。根据其设计思想，我们也很容易把它移植到其它单片机上。但是它也有缺陷，例如：不支持外部任务切换；不支持用户使用定时器T0等。这些缺陷的存在，限制了任务切换的灵活性。在本文的结尾我要特别感谢C51BBS的各位网友给我提供了很大的帮助，使得本文能顺利成文。

参考文献

1 徐爱钧，彭秀华. 单片机高级语言C51 Windows环境编程与应用. 北京: 电子工业出版社，2001

（收修改稿日期：2003-04-18）