基于事件驱动的单片机多任务程序设计①

周富相，陈德毅，郑晓晶

(总参通信训练基地，宣化 075100)

摘 要：利用单片机进行嵌入式系统开发时，经常会面临同时处理多个任务的要求。为了在资源紧缺的单片机系统中获得资源与性能平衡的条件下，实现多任务处理机制，提出了一种基于事件驱动的单片机多任务程序设计方法。该方法内核设计简单、代码少、移植性好，可用于大多数带有定时中断的单片机系统，使一些复杂的单片机程序设计简单化，逻辑控制具有实时性。

关键词：单片机；多任务；程序；事件

Design of Single-Chip Multi-Duties Program Based on Event Driving

ZHOU Fu-Xiang, CHEN De-Yi, ZHEN Xiao-Jing

(Communication Education Institute General Staff, PLA, Xuanhua 075100, China)

Abstract: Developing embedded system using single-chip microprocessor, we are often faced with dealing with multi-duties at the same time. In order to implement multi-duties in sing-chip system of fewer resource, a new method of program design based on event driving is put forward. This method is simple, which can be applied for most single-chip application system with timer interruption, making some complex programming simple, logical control with real-time.

Key words: single-chip; multi-duties; program; event

嵌入式系统最常见的软件架构一般有两种[1]：前后台系统和多任务实时操作系统，单片机应用系统广泛使用的是前后系统。在这种架构下，应用程序一般是一个无限循环，循环中调用相应的函数完成相应的操作。这种程序设计方法就是单任务机制，单任务系统具有简单直观、易于控制的优点。然而由于程序只能按顺序依次执行，缺乏灵活性，只能使用中断函数实时地处理一些较短的任务，在较复杂的应用中使用极为不便，嵌入式多任务实时操作系统的出现解决了这个问题[2]。在多任务系统中，可以同时执行多个并行任务。但是，嵌入式操作系统在提供强大功能的同时， 也带来了代码量大、结构复杂、对硬件要求较高、开发难度大且成本高等问题。对于资源本身紧缺的单片机而言，采用多任务实时操作系统在很大程度上对于系统性能的提升并没有带来显著的改善。更多情况下只需要实现简单的多任务操作就可以满足实际需要，将多任务机制引入单片机系统，可以大大提高现有单片机系统的工作效率，满足多任务要求。

1 单片机多任务系统内核设计

单片机多任务系统分为两个部分，多任务内核和用户应用程序。多任务内核是整个单片机软件系统的核心部分，负责进行任务管理、任务调度及事件处理；用户应用程序在多任务内核的调度下通过解析事件来执行相应处理，完成相应功能。因此，主要对多任务内核的设计进行论述，而用户应用程序这里不再详细论述。

1.1 任务控制块设计

一个任务也称为一个线程[3]，是一个具有特定功能的程序，每个任务都是一个无限循环。在嵌入式实时操作系统里，每个任务具有多种状态，如 uC/OS-II 系统中的任务状态定义为 5 种。为了内核调度简单，

① 收稿时间:2011-10-08;收到修改稿时间:2011-11-10

这里对每个任务状态只定义两种：运行态和等待态。当任务被执行的时候处于运行态，当任务被挂起不被执行时处于等待态。借鉴嵌入式实时操作系统中对任务管理的思想，本文也引入了任务控制块（Task Control Block, TCB）的概念，TCB 是一种特殊的数据结构，用来对任务的管理，如实现任务优先级的标记，驱动事件的接收等功能。TCB 在单片机多任务系统内核中起重要作用，其结构设计如下：

```
Typdedef   struct
{
    INT16U   wTimerPeriod;
    INT16U   wTimerCount;
    INT16U   wEvent;
}OS_TCB;
```

wTimerPeriod：定义任务运行的周期，也是单片机定时器的定时周期，一般在 ms 级，当定时中断发生时，如果此任务优先级最高，且处于等待状态，那么此时该任务得到执行。

wTimerCount：定义任务已经等待的时间，用一时间计数器来实现，任务设置运行周期后，每个系统定时中断都会更新所有任务的等待时间计数器。当等待时间计数器达到设定的任务运行周期时，任务将收到定时事件（Timer）。

wEvent：定义任务接收到的事件，事件是任务与任务之间通信的唯一方式，每个任务可以事先定义若干个事件。任务的执行以事件为驱动，内核只有在查询到任务接收到事件后任务才开始执行相应的处理，否则任务挂起。

1.2 任务切换

在多任务实时操作系统中，每个任务都有自己独立的堆栈空间，用于存放任务的当前状态。当多任务内核调度其它优先级更高的任务时，堆栈空间保存正在运行任务的当前状态，然后从要调度的任务堆栈中恢复将要运行的任务，实现任务切换[4,5]。多任务实时操作系统的任务切换是建立在高性能处理器的基础上的，对资源非常有限的单片机来说，RAM 只有几 K 字节，难以用这种方式实现任务切换。因此，为了节省 RAM，执行任务切换时不保存当前任务状态，直接切换到下一个任务。

在任务切换过程中也引入优先级设计思想[6]，也就是每个任务都被赋予特定的优先级，任务的调度是基于优先级的，任务越重要，就应设计越高的优先级，这样当多个任务在等待 CPU 时，优先级最高的任务总是最先被执行。任务的执行总是通过事件来触发的，每个任务都事先定义了若干个事件，当多任务内核查询到任务管理块接收到任务触发事件时，如果任务的优先级是最高的，那么调度该任务开始执行。

任务切换的函数设计如下：

```
INT16U OS_Task_Switch(INT8U wTaskPrio)
{
  INT8U   wTemp;
  for( wTemp=0;wTemp < wTaskPrio; wTemp++)
 {
   if(OSTCB[wTemp].wEvent != 0)
    {
      return(true);
    }
 }
 return(false);
}
```

其中，wTaskPrio 是任务优先级参数， OSTCB 是任务控制块数组定义。函数检查所有比当前任务优先级更高的任务，如果查询到有更高的优先级任务，那么再检查该任务是否接收到事件，然后等待执行。

任务切换的程序流程如图 1 所示。

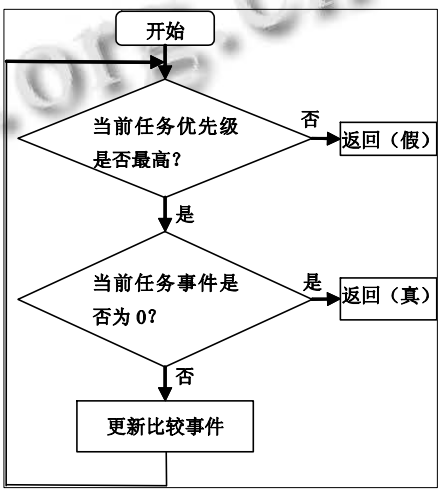

图 1 任务切换流程图

2 单片机多任务内核主要函数设计

单片机多任务内核的主要功能由任务创建、事件查询以及发送事件等 8 个函数来实现，这 8 个内核功

能函数是单片机多任务内核的核心，其全部代码都存放在文件 Kernel.c 中。单片机多任务内核主要函数的具体设计如下：

2.1 事件查询与发送

单片机多任务内核的事件查询与发送函数具体设计如下：

```
INT16U OSEventPend(INT8U wTaskPrio)
{
  INT16U wTaskEvent;
  wTackEvent = OSTCB[wTaskPrio].wEvent;
  OSTCB[wTaskPrio].wEvent &= ~wTaskEvent;
  return(wTaskEvent);
}
Void  OSEventSend(INT8U  wTaskPrio,
                  INT16U wTaskEvent)
{
  OSTCB[wTaskPrio].wEvent!=(1<< wTaskEvent);
}
```

其中，OSEventPend(…)是事件查询函数，任务查询时以其优先级为识别特征，通过读取任务控制块 TCB 中优先级对应索引的事件变量来获得当前任务的事件，事件在读取之后立即被清除。OSEventSend(…)是发送事件函数，通过对任务控制块 TCB 中优先级对应索引的事件变量进行写操作来实现对此优先级对应的任务发送事件。

2.2 任务创建、开始与状态更新

单片机多任务内核的任务创建、开始与状态更新函数的具体设计如下：

```
Void OS_TaskCreate(INT8U wTaskPrio,
                   INT16U wTimerPeriod)
{
  OSTCB[wTaskPrio].wTimerPeriod=
                        wTimer  Period;
}
void OS_TaskStart(INT8U wTaskPrio,
INT16U wTimerCount)
{
  OSTCB[wTaskPrio]. wTimerCount=
                         wTimerCount;
}
void OS_TaskUpdate(void)
{
  INT8U wTemp;
  for(wTemp=0 ; wTemp<cMaxTash ; wTemp++)
  {
    OSTCB[wTemp].wTimerCount ++;
    if(OSTCB[wTemp].wTimerCount
      ==OSTCB[wTemp].wTimerPeriod)
    {
      OSTCB[wTemp].wTimerCount = 0;
      OSEventSend(wTemp,0);
    }
  }
}
```

其中，OS_TaskCreate(…)是单片机多任务内核的任务创建函数，功能是创建一个任务，创建任务的工作只是设定一下任务的执行周期。OS_TaskStart(…)是任务开始函数，该函数设定当前任务已经延时等待的周期。OS_TaskUpdate(…)是任务状态更新函数，是在内核时钟中断中执行，每一次系统时钟中断时，任务更新函数会对所有任务已经等待的计数器进行更新，检查并向已经超时等待的任务发送 Timer 事件。

2.3 内核初始化与运行

单片机多任务内核初始化函数具体设计如下：

```
void OS_Init(void)
{
  INT8U wTemp;
  for(wTemp = 0; wTemp < cMaxTask; wTemp++)
  {
    OSTCB[wTemp].wTimerPeriod = 0;
    OSTCB[wTemp].wTimerCount = 0;
    OSTCB[wTemp].wEvent = 0;
  }
}
```

其中，cMaxTask 是应用程序最大任务数，是用户在应用程序中以全局变量的形式给出。内核初始化函数主要功能是对任务控制块 TCB 状态进行清零处理。

单片机多任务内核运行函数的具体设计如下：

```
void OS_Start(void)
{
  Task_Init();
  Task_Start();
```

}

内核运行函数的功能连续调用两个函数：Task_Init(…)函数执行所有任务的初始化代码；Task_Start(…)函数开始多任务运行。

3 程序流程

基于事件驱动的单片机多任务程序设计流程如图 2 所示。

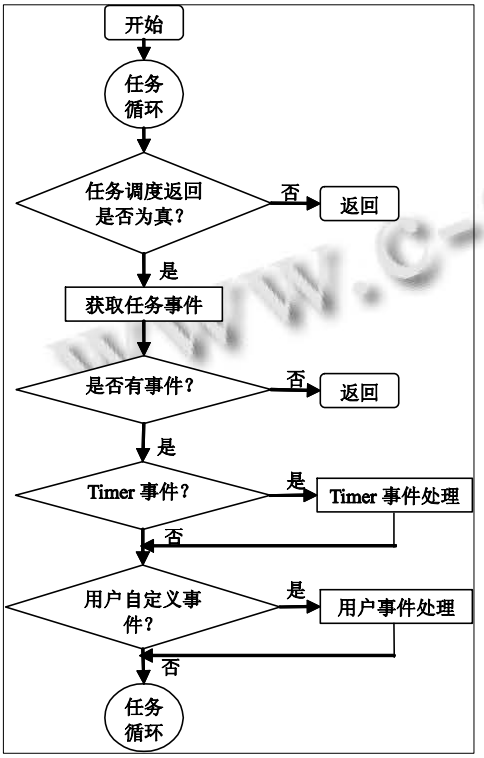

图 2 任务流程图

程序开始时进行多任务内核初始化操作，如定义最大任务数、定义各任务的优先级和任务事件等，通过调度内核函数来创建用户自定义的任务。初始化工作完成后就进入任务循环，所有任务的结构都是一个无限循环，在无限循环体中解析事件并执行相应的事件处理。在任务调度过程中每个任务总是最先检查任务调度情况，如有更高优先级的任务等待处理器的执行，则退出当前任务，否则任务将获取事件，如有事件则对事件进行逐一解析并处理。

4 结语

单片机在嵌入式低端市场应用较为普遍，其程序设计越来越复杂，为了在单片机系统中实现类似实时多任务操作系统的机制，借鉴了主流操作系统中的多任务和多线程机制，提出了一种新的单片机系统程序设计方法：基于事件驱动的单片机多任务程序结构设计。这种程序架构通过一个独立的定时器来模拟实时操作系统中的内核时钟，仅需要使用几个简单的函数来完成多任务系统的调度。这种设计方法没有汇编代码的移植，具有很好的通用性。同时，为了适应资源较少的单片机系统应用，此架构保留了实时操作系统中事件通信方式的思想和简化的任务控制块结构，将内核对处理器资源的占用减到最少，获得资源与性能之间的平衡。

参考文献

1 罗江,户永清.51 单片机多任务机制的实现策略研究.四川文理学院学报(自然科学),2008,18(2):82–85.
2 高国胜,陈俊杰.单片机系统的实时多任务机制研究.舰船电子工程,2009,29(8):137–140.
3 方志雄.分时操作系统思想在单片机编程中的实现.邯郸学院学报,2005,15(3):28–32.
4 潘永雄.基于过程的单片机多任务程序结构及实现方法.单片机技术及应用,2005,31(3):5–8.
5 杨金,孙世新.基于消息机制的单片机多任务系统的开发.计算机时代,2007,2007(4):55–57.
6 朱彩霞,徐江海.单片机多任务延时程序设计.邯郸学院学报,2009,18(1):54–58.