设备级实时多任务嵌入式内核WebitV的设计与实现*)

林 涛 赵 海 王济勇 韩光洁 王金冬
(东北大学信息科学与工程学院 沈阳110004)

摘 要 使用实时内核来进行多任务的管理是目前嵌入式应用的一个趋势,面向设备的嵌入式应用由于其资源受到严重限制,具有针对性强的特点,本文在一个8位嵌入式芯片上设计并实现了一个面向设备的抢占式实时多任务内核WebitV,它在总体上保留了传统内核的主要特性,但在任务调度,优先级分配以及存储器管理等几个方面进行了改进,使它更适合于设备计算,为普适计算提供了一个底层平台。

关键词 嵌入式,内核,普适计算,实时,抢占式,多任务

The Design and Implementation of a Real-Time Multitask Embedded Device Level Kernel WebitV

LIN Tao ZHAO Hai WANG Ji-Yong HAN Guang-Jie WANG Jin-Dong
(School of Information Science & Engineering, Northeastern University, Shenyang 110004)

Abstract It is a trend to use real-time kernel in embedded application. Because of its serious scarcity resources the application of device level is different from other embedded application. A new preemptive multi-task kernel-WebitV based on an 8-bit embedded chip is presented in this paper. It remain the main character of traditional embedded kernel but have been improved in several portion, such as task schedule, priority assignment and memory management. After the redesign WebitV are more fit for device level computing and it can provide a low platform for ubiquitous computing.

Keywords Embedded, Kernel, Ubiquitous computing, Real-Time, Preemptive, Multitask

1 引言

随着嵌入式技术的不断发展,日常的各种设备具有了很强的处理能力,分布在各地的嵌入式芯片给普适计算提供了一个绝好的平台,而这必将导致普适计算技术的成熟[1,2]。普适计算是融入日常生活的、无所不在的计算,这也正是嵌入式技术发展的目标。早期的嵌入式Internet技术大都是基于通用计算机的嵌入式Internet服务器,由于体积大、成本高等缺点,已逐渐被淘汰。近年来出现了基于MCU的嵌入式Internet产品,由于体积小、成本低的优势,使其具有良好的应用前景,但此类产品大都是基于传统的前后台系统来开发的,系统的可调度利用率很低,对于这种前后台系统可调度性分析也缺乏相应的理论,难以保证系统的实时性和可靠性[3,4]。对于复杂的控制功能和数据采集功能,只能采取现场总线的通信协议与设备内部的MCU通信,由设备内部的MCU解析并执行控制命令,完成数据采集功能,系统运行效率很低。对系统开发人员来说,开发难度大,开发效率低。于是出现了实时操作系统(Real-Time Operating System,RTOS),这是解决以上问题的一个好方法。但是现在面临的问题是,所有的设备都不是基于某一种标准的,它有可能是一个家用电器,也有可能是一个工业设备,接口千差万别,要想使它们成为一个能够协同工作的集合,一定要给它们提供一个通用的接口,为此我们设计了一个面向设备的实时多任务内核——WebitV,并在此基础上设计了新型的嵌入式Web服务器。WebitV是一个基于优先级的抢占式多任务内核,为了使其适合普通设备,将内核进行了优化与裁减,该内核通过抢占式任务调度策略确保了嵌入式Internet应用程序的实时性和可靠性;通过提供任务创建、任务间同步与通信等系统函数(API)能大大方便用户应用程序的设计,提高了开发效率。

2 设备级嵌入式内核的特点

多数嵌入式系统有一个共同的特点:对系统的实时性和可靠性有严格的要求,每个应用对时限(或速度)的要求是千差万别的,所以设计实时系统的时候,应该明确目标系统的实时要求是什么,不可盲目追求高速度,以免浪费资源。面向设备的嵌入式内核的特点是功能单一,针对性强,对实时性和稳定性要求比较严格。针对这样的特点,如果将传统的嵌入式内核应用在设备控制上,必然会有一些资源上的浪费,以及功能上的不完善,为了满足设备级应用的要求,应该对传统嵌入式内核进行合理的改进和裁减。下面给出几个相关定义,WebitV系统的设计就是围绕着这几个主要问题展开的。

定义1(实时系统) 是指能在确定的时间内执行其功能并对外部的异步事件做出响应的计算机系统。一个系统具有实时性并不是说该系统的响应和处理速度非常快;而是说该系统的响应和处理速度对于特定的应用来说足够快。一个高速系统也未必是实时系统[5]。

定义2(软实时系统和硬实时系统) 软实时系统要求任务运行的越快越好,并不要求某一任务必须在限定的时间内完成。硬实时系统则要求任务的运行不仅要正确无误而且要准时,如果任务在时限到来之前未能完成,就会发生无法预测

*)本课题得到国家自然科学基金资助(69873007)。林 涛 博士研究生,主要研究领域为嵌入式Internet。赵 海 教授,博士生导师,研究方向嵌入式Internet,数据融合。

的灾难性后果。大多数实时系统是这两者的结合。而且大多数的实时系统都是嵌入式的。

定义3(任务) 在嵌入式系统中,一个任务也称作一个线程,是指一个程序分段。这个程序分段被操作系统当作一个基本单元来调度。它大致相当于一般计算机操作系统中进程的概念。每个任务都是整个应用的一部分,都被赋予了一个优先级,有它自己的一套CPU寄存器和自己的堆栈空间。一般情况下,每个任务都是一个无限的循环。不同的RTOS可能会给任务定义了不同种类的状态。典型的,每个任务会处于一下5种状态之一:就绪、休眠、运行、挂起和中断态。多任务的实现实际上是靠CPU在许多任务之间转换、调度。CPU按照某种调度算法,轮番服务于一系列任务中某一个。每个任务都象一个独立的前后台系统。多任务的运行使CPU的利用率得到最大的发挥,并使应用程序模块化。多任务的最大特点是,开发人员可以将很复杂的应用程序层次化,应用程序更容易设计和维护。

定义4(任务切换时间) 当由于某种原因使一个任务退出运行时,RTOS保存它的运行现场信息、插入相应队列、并依据一定的调度算法重新选择一个任务使之投入运行,这一过程所需时间称为任务切换时间。

定义5(最大中断禁止时间) 当RTOS运行在核态或执行某些系统调用的时候,是不会因为外部中断的到来而中断执行的。只有当RTOS重新回到用户态时才响应外部中断请求,这一过程所需的最大时间就是最大中断禁止时间。

3 WebitV的主要设计思路

RTOS的实时性和多任务能力在很大程度上取决于它的任务调度机制。从调度策略上来讲,分优先级调度策略和时间片轮转调度策略;从调度方式上来讲,分可抢占、不可抢占、选择可抢占调度方式[6]。在WebitV的设计上,采取了优先级调度的可抢占式的多任务内核,因为这种内核的时间性保证得比较好,实时性要求较高的任务可以抢占到系统资源,相应速度快,并且,为了简化调度算法,WebitV中任何两个任务的优先级都不相同。

3.1 多任务的调度算法

在多任务系统中,上下文切换指的是当处理器的控制权由运行任务转移到另外一个就绪任务是所发生的事件序列。当运行的任务转为就绪、挂起、或删除时,另外一个被选定的就绪任务就成为当前运行任务,上下文切换包括保存当前任务的状态,决定哪个任务运行,恢复将要运行的哪个任务的状态,保护和恢复上下文是依赖相关处理器的。因此,上下文切换事件是影响RTOS性能的一个重要指标。对于最高优先级任务的判断,不同的内核有不同的处理方式。最简单的是顺序检索:将所有任务按照优先级排序,优先级最高的排在队前,优先级低的排在队后。查找时从队列头开始检索,第一个遇到的就是最高优先级的就绪任务[7]。这种查找方式思路直接,便于实现,缺点是不同优先级任务在查找时所花费的CPU时间不同,如果系统中任务数目较多,查找将非常慢,将会大大影响任务调度的速度。而且,任务调度属于系统的临界段代码,调度中需要独占CPU,不允许中断或者任务切入,如果调度速度缓慢,就会大大影响整个系统的相应速度和处理能力。

为了实现就绪任务的快速查找,WebitV中采用了一种非常独特的方式来标识任务的就绪状态:在内存中设置一个任务就绪表,就绪表由一个变量KernelReady和一个4字节数组KernelReadyTable[4]表示。将16个任务分为4组,每组对应于数组KernelReadyTable[4]中的一个字节;每组4个任务,对应于KernelReadyTable[n]中的0～3bit;同时每一组(KernelReadyTable[0]～KernelReadyTable[3])的就绪状态有对应于一个8位变量KernelReady中的0～3bit。

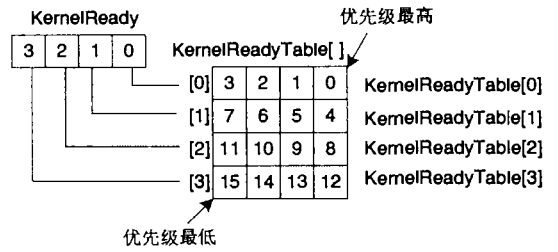

图1 任务就绪表

每个字节的后4位表示对应的4个任务是否就绪,如果就绪,则对应位就置1,否则清零。此外,只要每组中有一个任务就绪,则对应的KernelReady中的对应为也要置1。也就是说,KernelReadyTable[n]中有一位为1,则KernelReady的第n位就置1,只有当KernelReadyTable[n]中所有位都为0,则KernelReady的第n位才为0

KernelReadyTable[0]的0～3bit任何一位为1时,KernelReady中的第0位为1。

KernelReadyTable[1]的0～3bit任何一位为1时,KernelReady中的第1位为1。

KernelReadyTable[2]的0～3bit任何一位为1时,KernelReady中的第2位为1。

KernelReadyTable[3]的0～3bit任何一位为1时,KernelReady中的第3位为1。

KernelReady中4～7bit置为0。

3.2 任务状态切换策略

在任务调度算法使高优先级的任务优先执行的同时,任务的状态也在不断地切换。在本文设计的内核中每一任务都只有三种运行状态,就绪态,运行态,阻塞态。这样设计的好处是,任务状态清晰,减少内核设计的复杂性,由于本内核是针对设备的,只要在实时性上满足一定要求就可以完成用户任务了,所以在设计任务状态时,将传统的五个状态约简为三个。另外由于本内核是抢占式内核,当中断发生时可以将实时性比较高的任务转为运行态,如图2所示。在一个任务创建(creat)并起动(start)之后,就可以在系统中竞争一定的资源,在任一时刻,任务的状态一定是在这三种状态之一。由于CPU资源是唯一的,所以某一时刻只能有一个任务为运行态。就绪的任务是可以运行的,它与运行态的区别仅在于它没有占用CPU,就绪态任务一直在等待高优先级的运行态任务释放CPU后才能进入运行态。阻塞态是由于任务在运行态时进行的系统调用使它处于等待状态,因为就绪态的任务不能进行系统调用,任务不可能从就绪态迁移为阻塞态。

当一个任务由运行状态切换到另外一个状态后,它的各种信息需要进行完整的保存,以便它再次回到运行状态后可以继续运行,在本内核中设计了一个任务控制块,WebitV用它来保存该任务的状态。当任务重新得到CPU使用权时,任务控制块能确保任务从当时被中断的那一点丝毫不差地继续执行。由于在设备级应用中,任务都是预先设定的,不会动态改变,所有在WebitV中将任务控制块设计成固定的数据结

构，在存储器中分配固定的位置，这样，所有的任务都可以快速地找到自己状态的位置，迅速地加载。

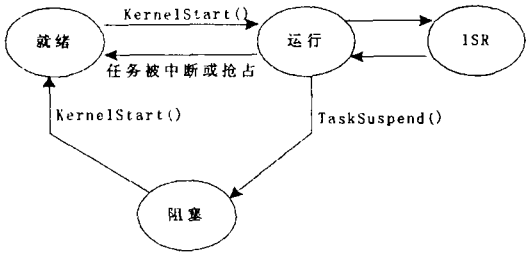

图2 任务状态转换图

3.3 任务间同步与通信

WebitV 提供信号量(Semaphore)和邮箱机制来实现任务间的同步和通信。信号量是被实时内核普遍使用的管理共享资源、实现任务间同步与互斥的机制。在 WebitV 中，信号量由两部分组成：一个是信号量的计数值，它是一个16位的无符号整数(0 到65，535之间)；另一个是由等待该信号量的任务组成的等待任务表，对信号量的操作有五种，分别是信号量的创建、等待、发送、请求和查询。在创建一个信号量时，要对这个信号量的初始计数值赋值，该初始值为0到65，535之间的一个数。

一个任务或者中断服务程序可以通过事件控制块 ECB (Event Control Block)来向另外的任务发信号/消息，事件控制块也是嵌入式内核常用的一个数据结构，在 WebitV 中它的设计与其他普通事件控制块没有区别。当任务调用函数取一个信号量时，其结果依赖于当时信号量是否可用，如果此时信号量可用，调用函数使信号量变为不可用，任务继续运行；如果此时信号量不可用，调用函数的任务进入阻塞状态，该任务被加入到该信号量的等待任务列表中，等待另外的任务或中断服务子程序发出该信号量，使该信号量变为可用。

由于本文设计的内核是抢占式的实时内核，如果进入就绪态的任务比当前运行的任务优先级高(假设，是当前任务释放的信号量激活了比自己优先级高的任务)，则内核做任务切换，高优先级的任务开始运行。当前任务被挂起。直到又变成就绪态中优先级最高任务。

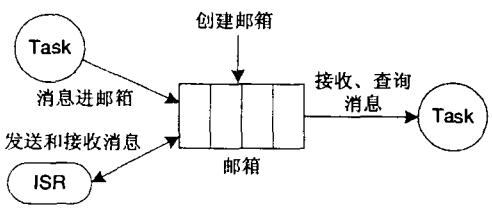

图3 邮箱模型

现代实时应用通常构造成一套相互独立的但又相互协调工作的任务集合。虽然信号量提供告诉的任务间同步和互斥机制，但常常还需要一种较高级的允许合作任务之间相互通信的机制。邮箱是实时内核中提供的另一种通信机制，它可以使一个任务或者中断服务子程序向另一个任务发送一个指针型的变量，该指针指向一个包含了特定“消息”的数据结构[8]。任务或者 ISR 调用 MsgPost()来向邮箱发送消息，此时如果没有任务在等待该邮箱中的消息，那么该消息进入邮箱。如果有任务在等待该邮箱的消息，那么这个消息立即提交给等待任务列表中优先级最高的任务。任务调用 MsgReceive()从消息邮箱总接收消息，如果没有消息可用，调用者将被阻塞，进入等待该消息的等待任务列表中。这个函数带有一个超时时间参数。超时的含义是指当邮箱总没有消息立即可用是，可以消息邮箱变为可用的时钟节拍数。

3.4 定时及存储器管理

定时管理是也是实时内核的一个重要功能，它为用户提供任务定时等系统服务。要实现定时功能，系统内核需要有一个周期性的定时中断来作为记时的单位，这个定时中断叫做时钟节拍。中断之间的间隔取值一般在10ms 到100ms 之间。时钟节拍式中断的内核，可以将任务延时若干个整数个时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍越快，系统的额外开销就越大。WebitV 中的时钟节拍是通过设置单片机内部的16位定时/计数器1获得的，中断间隔设为10ms。

在 WebitV 中，提供了一个这样的系统服务：申请该服务的任务可以延时一段时间，这段时间的长短是用时钟节拍的数目确定的。实现这个系统服务的函数叫做 KernelTimeDelay()。调用该函数会使内核进行一次任务调度，执行下一个优先级最高的就绪任务。任务调用 KernelTimeDelay()后，一旦规定的时间期满，它就会马上进入就绪状态。但只是进入就绪状态，只有当该任务在所有的就绪任务中具有最高优先级时，它才会立即运行。

由于 WebitV 是一个多任务内核，因此，对存储器容量的需求不仅仅取决于应用程序代码，内核本身还需要额外的代码空间(ROM)。内核的大小取决于多种因素，取决于内核的特性，Webit 的设计是面向一个8位嵌入式处理器，只提供任务调度、任务切换、信号量处理、延时及超时服务，大概需要的空间是3k 代码空间。在 WebitV 中，每个任务都是独立运行的，必须给每个任务提供单独的堆栈空间(RAM)。为了简化内核的复杂性，在 WebitV 中将每个任务的堆栈空间设计成固定大小，在这部分空间内，不仅仅包括任务本身的需求（局部变量、函数调用等等），还有最多中断嵌套层数(保存寄存器、中断服务程序中的局部变量等)。另外，在 WebitV 中，为了将每个任务所需的堆栈空间降到最低，将任务栈和系统栈分开来设计，系统栈专门用于处理中断级代码。

4 系统性能分析及内核评价

为了评价一个内核的效率，需要对其主要参数进行测试，并能够和同类别的系统进行比较，AVRX 是与 WebitV 系统具有相同硬件平台商业内核，下面就对这两种内核在同样的测试环境下采用相同的测试依据和方法进行性能分析。

评价一个内核的好坏主要从以下两个方面，任务切换时间(包括任务上下文切换和任务调度时间)，最大中断禁止时间。为了测试 WebitV 的性能指标，在程序的关键位置，设置对外部 I/O 口的输出语句，使外部 I/O 口的输出状态有所变化，通过分析存储示波器或者逻辑分析仪上的记录波型计算时间。

(1)任务切换时间的测试方法 创建两个任务，一个为低优先级任务 Task1，它循环输出一个高电平 P 和一个低电平 L，另外一个高优先级 Task2的任务则是持续的输出高电平 H，测试方法为，让低优先级任务首先获得处理器资源，然后，触发高优先级任务 Task2，抢占当前控制权输出高电平 H，然后挂起，低优先级任务 Task1激活后继续执行。这样的方法可以通过在逻辑分析仪上输出波形来进行分析，假设任务

Task1的循环周期为 T_P+T_L，而任务 Task2的一个周期为 T_H，假设任务切换时间用 T_{sw}来表示，如图4中所示，则在 T_2-T_1中包含两个任务切换时间，由此可以得出如下的计算公式：

$$T_{SW}=(2T_P+T_L-T_H)/2$$

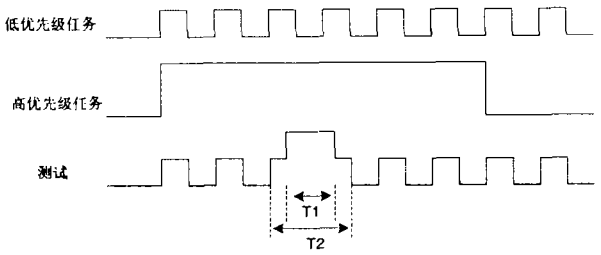

图4 任务切换时间测试图

(2)最大中断禁止时间的测量方法 设计两个任务，第一个任务重复如下步骤：使 PA0置1，然后调用任务切换，使系统切换为第二个任务。第二个任务重复如下步骤：使 PA0置0，然后调用任务切换，使系统切换为第一个任务。这样，PA0端口会以一定频率交替变换状态，这个周期就是两次任务调度时间。由于在任务调度期间会禁止中断相应，并且已经知道任务调度期间的中断禁止是最大的中断禁止时间，所以只要测量在中断调度期间的中断禁止时间就是内核的最大中断禁止时间。在这个过程中将一固定频率的信号接到单片机的外部中断引脚，会使系统不断产生中断响应，在中断服务程序中在端口 PA1输出一个脉冲。但由于任务调度期间禁止响应中断，会使中断响应产生一定的延时，测量中断信号和这个脉冲之间的时间间隔，就能测得本次的中断延时时间。多次测量这个延时时间，取其平均值然后乘以2，可以近似为最大中断禁止时间。

用以上两种测试方案，利用逻辑分析仪(TLA603)和仿真器等工具，可以测得多组数据，为了消除波动误差，对每种类别的试验反复测试，得出100组数据，然后求其平均值，得到下面的最终数据：

表1 性能测试结果表

	WebitV	AVRX
任务上下文切换时间(μs)	28	48
最大中断禁止时间(μs)	33	75

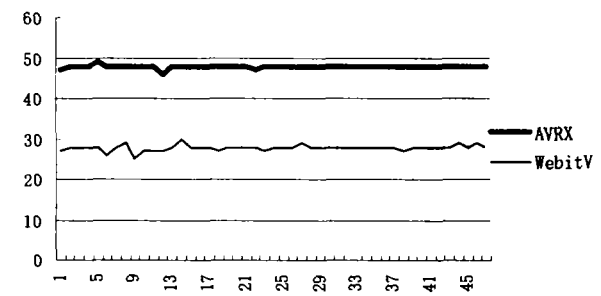

图5 任务切换时间对比图

为了明显看出两个内核的性能差别，将两组数据分别做成曲线图，如图5，图6所示，图中，较粗曲线代表 AVRX 内核，较细曲线代表本文设计的内核 WebitV，横坐标表示数据序列，纵坐标轴为时间，图5中 Y 轴表示任务切换时间，图6中 Y 轴表示最大禁止时间，单位都为微秒(μs)。从图中，可以看出，WebitV 的任务切换时间以及最大中断禁止时间都小于 AVRX 的相应的时间，这对于嵌入式应用来说是很重要的性能改进，可以使任务的实时性得到更好的保证，图中 AVRX 的曲线波动比较小，而 WebitV 的波动较大，这是因为，在测试 AVRX 时，调试语句无法进入系统调用内部，只能用程序来模拟，而对于 WebitV 则可以在它内部的任何一个位置查处调试输出语句，这样的时间测试得比较准确，总体上不会影响曲线的趋势。

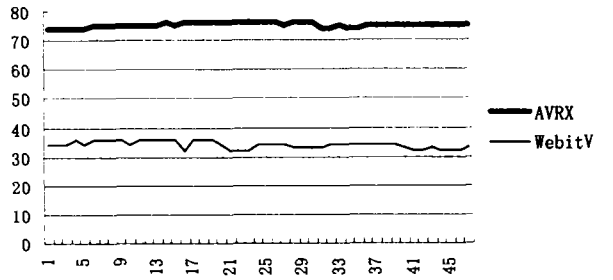

图6 最大中断禁止时间对比图

结论 WebitV 的使用使得实时应用程序的设计和扩展变得容易，不需要大的改动就可以增加新的功能。通过将应用程序分割成若干独立的任务，RTOS 使得应用程序的设计过程大为减化。使用可剥夺性内核时，所有时间要求苛刻的事件都得到了尽可能快捷、有效的处理。通过有效的服务，如信号量、邮箱、队列、延时、超时等，RTOS 使得资源得到更好的利用，尤其是在8位单片机这样的平台上面资源严重受限的情况下，更是有必要对原有内核进行改进，以适应面向设备的需求。

参考文献

1 Bergamaschi R, Bhattacharya S, Wagner R. Automating the design of SoCs using cores. IEEE Design & Test of Computers, 2001, 18(5): 32～45

2 Keutzer K, Malik S, Newton R. System level design: Orthogonolization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design, 2000, 19(12)

3 Klein E. RTOS Design: How is Your Application Affected?[J]. Embedded System Conference, 2000

4 Buttazzo G C. Hard Real-Time Computing Systems. Kluwer Academic Publishers, 1999

5 Gauthier L, et al. Automatic generation and targeting of application-specific operating systems and embedded systems software. IEEE Trans. On CAD, Nov. 2001

6 Dave B P, Lakshminarayana G, Jha N K. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. IEEE Trans. on Software Engineering, 1999, 7(1)

7 Labrosse J J. μC/OS-Ⅱ, The Real-Time Kernel [M]. R&D Publications, Inc, 1998

8 Balarin F, Sangiovanni-Vincentelli A. Schedule validation for embedded reactive real-time systems. In: 34th ACM/IEEE Design Automation Conf., Anaheim, USA, 1997. theory. In IEEE Int. Conf. on Computer Design, Texas, Sept. 1999