实时操作系统中任务间通信的一种方法

郭　鹏, 罗　浩, 廖明宏

(哈尔滨工业大学 计算机学院, 黑龙江 哈尔滨 150001)

摘　要: 在实时操作系统中任务间通信是影响系统性能的 1 个关键因素. 当前多数的实时内核中所提供的邮箱机制在一对多的任务间通信中效率不高, 实时性不强. 针对实时内核这个缺点引入固定消息邮箱机制的通信方法. 这种方法的优点是: 占用内存少, 实时性强. 将固定消息邮箱机制与传统的邮箱机制配合使用可以极大的提高任务间通信的效率.

关键词: 嵌入式; 任务间通信; 邮箱机制; 固定消息
中图分类号: TP393　　文献标识码: A　　文章编号: 1672-0946(2003)05-0561-04

Study on a method for Inter task communication in RTOS

GUO Peng, LUO Hao, LIAO Ming-hong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: It has been proved that the performance of the inter task communication is one of the bottlenecks in RTOS. The traditional mechanism for exchange of information between tasks is message-passing using mailboxes. But it performs not well in the environment of one task communicate with multiple tasks. Presented State Message Mailbox, a new method of inter task communication. There are 2 advantages in the new method, firstly, less memory consumption; secondly, better performance on real-time. If the State Message Mailbox cooperates with the traditional mailbox, the inter task communication will work more efficient.

Key words: embedded; inter task communication; mailbox; state message

在实时操作系统中, 有时需要在任务间或中断服务与任务间交换信息. 这种信息传递被称为任务间的通信(inter task communication). 有经验表明, 任务间的通信的好坏是影响实时操作系统实时性的 1 个关键因素. 因此, 改进任务间通信的方法是一条提高实时操作系统实时性的 1 个途径.

本文在源代码公开的嵌入式实时操作系统 μC/OS-II 的传统邮箱机制基础上进行改进, 提出一个固定消息邮箱机制, 以其配合原传统消息邮箱机制而获得更好的性能.

1　传统的邮箱机制及其缺点

1.1　传统邮箱机制

传统的邮箱(message mail box)也称作消息交换(message exchange). 消息在实时操作系统通常用 1 个指针型变量表示. 1 个任务或 1 个中断服务程序通过内核服务可以把 1 个消息(即 1 个指针)放到邮箱里去. 同样, 1 个或多个任务可以通过内核服务接受这个消息. 发送消息的任务和接受消息的任务约定, 该指针所指向的内容就是该消息.

可见, 邮箱就相当于消息的中转站. 每个邮箱

收稿日期: 2003-04-06.
基金项目: 该课题得到哈尔滨工业大学跨学科交叉性研究基金资助, 课题代号: HITMD. 2001.01.
作者简介: 郭　鹏(1971-), 男, 硕士研究生, 研究方向: 嵌入式操作系统.

有相应的正在等待消息的任务列表,要得到消息的任务会因为邮箱是空的而被挂起,且被记录到等待消息的任务表中,直到收到消息.一般地说,内核允许用户定义等待超时,如果在规定时间内收到该消息,则该任务接受消息,进入就绪态,并且邮箱删除该消息;若等待消息的时间超过了,仍然没有收到该消息,这个任务进入就绪态,并返回出错信息,报告等待超时错误.

2.2 传统邮箱机制的缺点

传统的邮箱机制在一对一的任务间通信非常有用,特别是在当作只取两个值的信号量来用时,表示某事件已发生,通知任务进行状态切换.但是,实时系统中的很多应用环境(例如,信号采集)常常要用到一对多的任务间通信,这时用邮箱机制进行信息交换就存在如下两个严重的缺点.

·在任务间传递1个消息需要30~60 μs(在奔腾133 MHz上),而在实时系统中,应用程序常常需要在任务间每秒钟传送数千个消息,这个延时是不能被接受的.

·如果1个任务要给多个任务发相同的消息,它必须每个任务都发1次.

由于以上2个缺点,程序员被迫采用全局变量来在任务间交换信息.这是一种不安全的设计方法,首先,这些全局变量是临界区,读写必须互斥;其次,容易导致难以跟踪的软件错误.这在传统的邮箱机制中是无法避免的缺点,要解决这个问题有必要引入新的通信方法.

2 固定消息邮箱机制(state message mailbox)

固定消息机制是对传统邮箱机制的1个改进.使用固定消息邮箱在任务间交换信息时,实时嵌入式操作系统自动产生1个全局变量在任务间进行信息交换.这个全局变量不是用户定义的,而是系统自己产生和管理的.实际上,用户甚至并不知道他已经使用了全局变量,而且,固定消息邮箱机制的系统服务接口和传统的消息邮箱机制完全一样.

固定消息邮箱机制可以很好的解决上面提到的信号采集时的生产者消费者问题.可以想象,在固定消息的"邮箱"只跟消息的发送方(写者)有关而与消息的接受方(读者)无关时,只有1个任务可以往邮箱中加入消息(称之为写任务)而多个任务可以从邮箱中获取消息(称之为读任务).在这种情况下,固定消息邮箱机制和传统的邮箱机制有如下几个方面的不同:

·固定消息邮箱只与写任务有关(也可能写任务是中断程序),只有1个任务或者1个中断程序能给邮箱加入消息,而多个读任务可以从邮箱中接收消息.

·1个新来的消息覆盖掉上1个消息.

·读任务接收到消息后,邮箱中的该消息保留,而在传统的邮箱机制中,读任务在接收到消息后邮箱自动删掉这条消息.

·写任务和读任务都可以无阻塞的运行,这大大减少了任务状态的切换的次数,可以极大的提高内核的实时性.

可以从上面的特点看出来,固定消息机制和共享内存很相似,但是前者所占的内存更少,更符合实时操作系统的要求.

3 固定消息邮箱机制的有效性

在实时嵌入式系统中,有些信息只是在一段时间内有效,一段时间过后又会来新的数据.比如说就像上文提到的实时信号的采集,在这种情况下,假如任务 T_1 采集数据并将数据提供给任务 T_2.如果任务 T_1 给任务 T_2 发送了2个消息,则第1个消息是无效的,因为这时候第2个消息明显是最新的,而且比第1个更有效.如果在这种情况下使用传统的消息邮箱机制,任务 T_2 必须先把第1个消息取走,然后才能读取第2个消息.如果多个任务需要这样的消息的话,任务 T_1 必须给每1个任务 T_n 都发送1个这样的消息.这明显是效率不高的,而固定消息邮箱机制能很好的解决这个问题.

在采集数据的时候,任务 T_1 建立1个固定消息邮箱 SM_1 且只有 T_1 能向 SM_1 写入消息,其他的需要这些消息的任务都知道 SM_1 包含了它们所需要的消息.任务 T_1 每采集一次数据都把它放入邮箱 SM_1,其他需要消息的任务则在需要数据的时候对 SM_1 进行读操作取走最新的数据.

由于任务 T_1 每写入一条新消息都会覆盖掉上一条消息,这使得每一次读任务都能获得当前最新的数据,而且读写双方都是无阻塞的运行,这可以大大的提高应用程序运行的效率.

在实时系统中,这种生产者消费者问题非常普遍.1个任务产生数据(可能是传感器传来的数据,也有可能是实时计算产生的即时数据)发送给其他多个任务.在这种情况下,固定消息邮箱机制就非常适用.但是在一些情况下,阻塞1个任务还是非常有必要的,例如1个任务在等待某个事件的发

生. 这时候, 就要用到传统的邮箱机制或者是信号量机制了. 所以说, 固定邮箱机制并不能完全替换掉传统的邮箱机制而是作为传统邮箱机制的 1 个补充. 但是固定邮箱机制在很多情况下还是适用的.

4　固定消息邮箱机制的相关研究

固定消息邮箱机制由 LAMPORT[1] 在 1977 年他的论文中提出. 最先将固定消息邮箱机制应用于实际的系统是 MARS OS[2], 另 1 个嵌入式系统 ERCOS[3] 也在系统中加入了这一机制. 这些系统中运行固定消息邮箱机制的详情请参看[4].

在上述固定消息的实际应用中存在 1 个问题, 当读写消息考虑到同步问题时, 读任务可能只是读到 1 个只写了一半的消息, 因为可能被写任务抢占. 这两个系统通过设置 1 个 N 槽缓冲池来解决这个问题, 1 个槽存放一条消息. 1 个指针指向当前最新的消息, 读任务循环读消息直到确保消息完整的时候才取走这个消息. 在 N 足够大的时候, 这种方法是安全的, 在 MARS OS 和 ERCOS 中 N 取 100. 但是, 当 N 过大时, 系统内存占用太多, 而且循环读取使读任务有可能花许多时间来等待消息可用. 这在一些运行条件苛刻的嵌入式环境中是不合适的, 我们期望 N 取值尽可能小而且读任务不要循环等待消息完整.

5　问题的解决

本文做如下假设

• 实时操作系统的任务调度是基于优先级的抢占式调度配合以最后期限调度.

• 读任务和写任务是在不同的线程中运行.

• 读任务的优先级低于写任务.

在抢占式的内核中, 任务级别低的任务不可能抢占任务级别高的任务, 低级别任务必须在高级别任务完成以后才有可能执行. 下面我们根据读任务的最后期限来确定 N 的大小.

N 的计算

设 CPU 带宽为 B, 消息的长度为 L, 单位均为字节.

当 $L \leqslant B$ 时计算非常简单, 因为 CPU 可以在一条命令内完成对消息的读写操作, 此时设 $N = 1$ 即可放心的操作而不会出现问题. 这时候这个 N 槽缓冲池就是 1 个 L 字节变量, 可以编写 1 个宏专门对其进行读写操作.

当 $L > B$ 时情况稍微复杂, 如果还沿用 $L \leqslant B$ 时的处理就会出现问题. 因为此时 CPU 无法在一条指令内处理完这个消息, 当 1 个读任务正在读取消息的时候可能正读到一半, 这时候新的消息到来了, 因为写任务的优先级高, 它立刻剥夺了读任务的 CPU 占有权, 然后写入新消息. 这样当读任务恢复执行的时候, 消息已经变了, 此时的消息是错误的消息. 要解决这个问题, 可以设置 1 个索引变量 I, I 总是指向当前最新的消息, 每次来新消息的时候, 写任务总是把消息写入 $I+1$ 的地方. 这样, 就总能确保读任务总能得到当前最新的消息.

当 $L > B$ 时计算 N 方法

考虑最坏情况, 当读任务每读 B 字节消息就有 1 个新的消息来到, 这样, 为了保证消息的完整性, 如果不考虑最终期限, 则

$$N = x_{\max} + 1.$$

式中: $x_{\max}$ 是 CPU 完成一次消息操作要执行的指令条数, $x_{\max} + 1$ 是因为还要对索引变量进行加 1 操作.

考虑任务的最终期限(deadlines), 此时读任务不可能总被写任务抢占, 任务期限一到必须马上运行(这是由任务调度保证的). 设 maxreadtime 为读任务直到可以读消息时的最大等待时间(包括任务被抢占的等待时间), d 为读任务的期限, e 为读任务的运行时间, 这样读任务可被抢占的时间为 $(d - e)$. 如果 e_r 为任务读消息所消耗的时间, 则

$$\text{maxreadtime} = d - (e - e_r).$$

在 1 个读任务的可等待时间内新消息到来的可能的最大次数如图 1 所示.

此时, 第 1 个写任务尽可能迟地到来(在写任务的最终时限之前)其后每个写任务都在它的周期刚开始就运行, 可以得到在最终期限下写任务发生次数的限制条件

$$x_{\max} - 1 - \left[\frac{\text{maxreadtime} - (p_w - d_w)}{p_w}\right].$$

这时候只要取 $N = x_{\max}$ 就能保证读写任务安全无阻塞运行.

最后考虑一种情况, 当 1 个读任务或者多个读任务的周期和最终期限很长(可以叫做 slow reader), 这样的话 $x_{\max}$, 就非常大(可能 10 以上), 这样缓冲池的内存占用量就过大, 不符合实时嵌入式系统的要求. 这个问题可以通过设置 1 个原子操作来完成 slow reader 的拷贝内存工作, 运用这个方法就可以节省下系统内存. 这个方法对系统性能稍有影响, 好在 slow reader 在系统中不可能频繁发生, 这

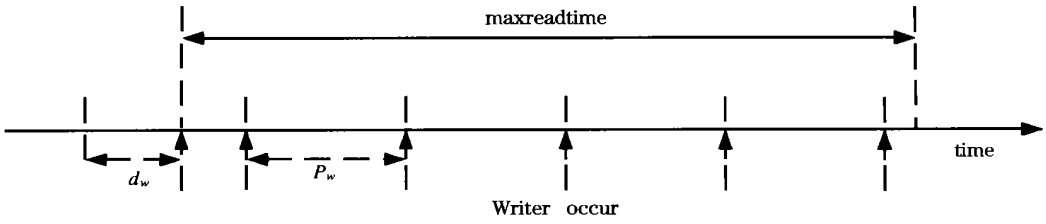

图1 等待时间内新消息到来的最大次数

Figure 1 Most of new messages during the waiting time

种影响基本上可以忽略不计. 注意,不论是否是原子操作,写任务的操作不变.

6 实验证明

笔者在μC/OS - II 实现了上述固定邮箱机制[5],下表是分别在固定邮箱机制和传统邮箱机制下1个任务给20个任务发送1 000个消息的实验数据比较(运行平台为奔腾133).

表1 固定邮箱与传统邮箱机制下实验数据比较

Table 1 Experimental data comparison between state mailbox and traditional mailbox

	固定邮箱机制	传统邮箱机制
Send(8字节)	2.1μs	14μs
receive(8字节)	2.3μs	7μs
Slow receive(8字节)	4.4μs	……

从比较中,可以看出固定邮箱机制在一对多的情况下确实比传统的邮箱机制性能高.

7 结语

上述固定消息机制是针对实时操作系统中任务间一对多的通信问题,满足其对于操作系统微小化、实时性强、可配置指标等要求. 它和传统的邮箱机制配合应用在各个方面可以极大的提高RTOS的性能[6].

参考文献:

[1] LAMPORT L. Concurrent reading and writing[J]. Communications of the ACM 20, 1977(11): 806-811.

[2] KOPETZ H, DAMM A, KOZA C, *et al*. Distributed fault-tolerant real-time systems: the MARS approach[J]. IEEE Micro 9, 1989(1): 25-40.

[3] POLEDNA S, MOCKEN T, SCHIEMANN J. ERCOS: an operating system for automotive applications[C]. In Society of Automotive Engineers International Congress and Exposition (February 1996), pp. 55-65. SAE Technical Paper Series 960623.

[4] KOPETZ H, REISINGER J. The non-blocking write protocol NBW: a solution to a real-time synchronization problem[C]. In Proc. Real-Time Systems Symposium (1993), 131-137.

[5] 罗 宇, 商临锋. 操作系统多线程实现技术研究[J]. 小型微型计算机系统, 2000(5):500-503.

[6] 王书达, 王肖林. RS-232C接口与单片机串行口通信的研究[J]. 哈尔滨商业大学学报:自然科学版, 2003, 19(4):411-414.