嵌入式操作系统中非阻塞的同步机制

张 丽，李仁发，彭蔓蔓，凌纯清
（湖南大学 计算机与通讯学院，湖南 长沙 410082）

摘 要：传统阻塞同步的缺点在于容易引起进程死锁和优先级的翻转，为了有效解决此类问题，出现了一种新的同步机制——非阻塞同步，该同步机制主要包括锁自由同步和等待自由同步。介绍了非阻塞同步在操作系统中的实现前提——同步原语和类型稳定的存储器管理以及它的实现技术，最后根据嵌入式系统的特点，提出了把非阻塞同步用于嵌入式操作系统的思想。

关键词：非阻塞同步；嵌入式操作系统
中图法分类号：TP316.2 **文献标识码**：A **文章编号**：1001-3695(2004)03-0088-03

Non-blocking Synchronization Mechanism for Embedded Operating System

ZHANG Li, LI Ren-fa, PENG Man-man, LING Chun-qing
(*College of Computer & Communication, Hunan University, Changsha Hunan* 410082, *China*)

Abstract: The shortcoming of traditional blocking synchronization is too easily leading to process deadlock and priority inversion. A new synchronization mechanism is proposed to solve above-mentioned issue effectively, which is non-blocking synchronization. Non-blocking synchronization comes in two favors: lock-free and wait-free synchronization. This article introduces implementation premises, which covers synchronization primitive and type-stable memory management, and implementation technologies of non-blocking synchronization on OS. At last, according to the characteristic of embedded system, we put forward an idea that non-blocking synchronization is applied to embedded operating system.

Key words: Non-blocking Synchronization; Embedded Operating System

1 引言

随着嵌入式系统应用范围的日益广泛，系统功能越来越强大，嵌入式操作系统的复杂性也与日俱增。由于应用环境的需求和硬件条件的限制，嵌入式操作系统一般都有专用性高、实时性强、占用空间小和效率高等特点。因此，近年来对嵌入式操作系统的研究成为该领域的一个热点。与台式计算机的操作系统一样，嵌入式操作系统中的同步问题成为系统软件设计中的一个关键问题。

传统同步方法最容易引起的问题是进程死锁和优先级翻转，在对时间限制要求严格的实时系统中，这两个问题造成的危害尤为严重，它会让系统崩溃。因此如何设计一种可以从根本上解决问题的同步机制成为嵌入式操作系统研究的主要问题之一。非阻塞同步是一种可满足上述需求的高效的同步机制，它允许多个进程同时访问共享对象（即共享资源），进程在临界区内不被阻塞，有效地避免了死锁和优先级翻转的问题[2]。

按照实现机制的不同，非阻塞同步可分为锁自由（Lock-Free）同步和等待自由（Wait-Free）同步两种。在操作系统中实现锁自由的非阻塞同步需要两个基本条件的支持——修改内存的原子指令和类型稳定的存储器管理（TSM）。由于很多有效的锁自由同步算法需要一种原子修改两个不连续存储器字的原语 CAS2（Two-words Compare-And-Swap），而这种原语在多种通用的处理器中都不提供，所以并没有很多实现该同步机制的操作系统实例。到目前为止，只有 Synthesis 内核和 Cache 内核有效地实现了锁自由的非阻塞同步。另外，由于等待自由同步算法的复杂性比较高，所以很少有操作系统使用这种同步机制。但文献[5]中把带有优先级继承策略的锁机制看作一种有效的等待自由同步的思想却大大简化了该同步机制的实现。Fiasco 内核就是锁自由同步和等待自由同步联合实现的实例。

收稿日期：2003-03-09；**修返日期**：2003-04-06

2 非阻塞同步的相关研究

2.1 总体背景

非阻塞同步方法的研究最早始于 20 世纪 90 年代初[3,4]，它属互斥型同步机制（传统的同步机制可分为互斥型和协作型），最初该同步方法是为共享存储区的多处理器设计的。由于传统同步方法中的锁机制在多处理器系统中成为各进程竞争访问的热点，造成系统性能的下降，于是出现了锁自由（Lock-Free）的非阻塞同步方法[3]。该方法的精髓在于它把对临界区访问的互斥性概念和对临界区的写操作集中在一条原子指令上（CAS，LL-SC），从而保证了访问数据对象的一致性。锁自由的同步方法使得进程在不阻塞的前提下，部分进程在有限的步骤内完成临界区的操作，但是却有可能导致另一部分进程饥饿（算法框架如图 1(a) 所示）。然而重要的是该同步机制有很好的特性：①由于它不使用锁来保证访问的互斥性，自然而然就避免了死锁的问题；②由于不用锁使得访问临界区的进程间的相互联系减少，避免正常的进程受到死进程的破坏性影

响,提高了系统的健壮性和容错性。

文献[4]提出了另外一种能力更强的非阻塞同步方法——等待自由(Wait-Free)。该方法的特点是在访问临界区之前如果进程检测到冲突,它并不像传统的阻塞策略一样,先等待临界区内的进程完成操作,而是继续运行并通过某种帮助策略先帮助临界区内的进程完成操作,然后再执行自己的临界区操作。等待自由的同步方法保证了所有进程都能在有限的步骤内完成其临界区的操作,因此很好地避免了进程的饥饿(算法框架如图1(b)所示)。文献[5]中还指出可以把等待自由看作添加了非阻塞帮助策略的锁机制。比如说,只要临界区不阻塞,锁机制再加上优先级继承策略就可认为是等待自由的同步机制。

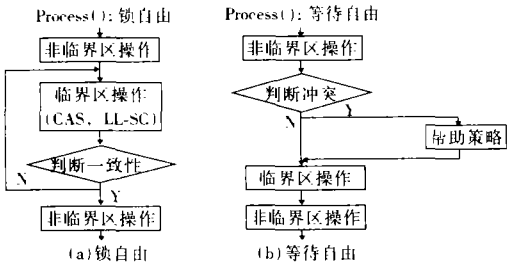

图1　算法框架

2.2　非阻塞同步的实现前提——同步原语和类型稳定的存储器管理

(1)原子内存修改指令(即同步原语)

原子内存修改指令CAS(Compare-And-Swap)和LL-SC(Load-Linked/Store-Conditional)是实现非阻塞同步的基础,原语的伪代码如图2所示。它们通过锁住存储器总线或其他一些方法保证在指令执行期间对存储区的原子性访问。最早引入CAS原语的是IBM370的处理器,LL-SC原语最早出现在文献[2]中。随着非阻塞的同步机制日益受到人们的关注,越来越多的处理器开始支持这些原子指令。如MIPS、ALPHA、Power PC处理器提供LL-SC原语,Pentium、SPARC、x86支持CAS原语[2]。

```
boolean CAS(val * addr,val old,val new)        val LL(val * add)
  atomically                                     return( * add)
if( * addr = = old)                              boolean SC(val * add,val new)
 * addr = new;                                     atomically
return true;                                     if( * add ! = new)
                                                 * add = new;
else return false;                               return true;
                                                 else return false;
```

图2　CAS和LL-SC原语的伪代码

一些数据结构(像计数器、堆栈和FIFO队列)可以在CAS或LL-SC原语的基础上实现非阻塞的同步操作,如文献[6]就介绍了一个基于优先级的单链表队列的非阻塞实现算法。该算法使用的是CAS原语,但由于它要求在进行队列操作的时候添加一辅助性节点,用于解决删除操作被插入操作抢占时造成的节点丢失问题,增加了算法的复杂性,浪费了系统资源,所以在实现操作系统内核的数据结构时该算法是不可取的。

有实验证明用DCAS(Double-Compare-And-Swap)原语实现大多数数据结构比CAS或LL-SC的实现更加有效[7],如文献[2]中第193-194页介绍的普通单链表队列的实现用的就是DCAS原语。由于DCAS原语可以同时更新两个不相邻的存储器字,所以在定义每个队列时添加一个版本信息,用DCAS原语来修改就能满足插入操作和删除操作并行执行的要求。该算法简化了实现步骤,提高了系统资源利用率。但是目前直接支持DCAS原语的处理器类型很少(只有Motorola 680x0系列处理器[8,9]),于是研究如何用软件模拟DCAS原语来代替其硬件实现成为当前非阻塞同步领域的一个热点[2,7]。

(2)类型稳定的存储器管理(TSM)

TSM是指对存储器的分配和再分配进行管理,使得被分配出去的部分在一定的时间期限内T_{stable}不改变其类型。它是对传统方法的一种扩展,比如说,在很多操作系统中,进程描述符都在系统初始化的时候静态分配,它的类型在系统执行的整个时间内都是不变的[2,9]。由于TSM在T_{stable}时间内保持类型不改变,大大简化了非阻塞同步算法的实现。

如果没有TSM,就会出现访问类型不一致的问题。该问题是指当我们准备用CAS或DCAS原语对共享对象进行操作时,该对象所占的空间已经被释放,但紧接着它又被重新分配给其他类型的对象,这时再用CAS或DCAS原语修改就会不成功(因为类型不一致),从而增加了非阻塞同步的开销。

2.3　非阻塞同步在操作系统中的实现方法

到目前为止,我们所知道的在操作系统的内核中实现非阻塞同步的例子只有三个:Synthesis内核、Cache内核和Fiasco内核。前两个系统运行于Motorola 680x0的处理器之上,用的是DCAS原语;后一个系统搭建在x86的处理器上,它的实现依赖于CAS原语,因为x86机不直接提供DCAS原语。

(1)Synthesis内核[8]是最早一个实现非阻塞同步的操作系统。该系统只使用了锁自由的同步机制,它首先介绍了简单数据结构的锁自由实现(如计数器、堆栈和队列),然后在此基础上构建更复杂的同步对象(像线程、虚拟存储器管理和文件系统)。实验证明在多处理器内核中实现非阻塞同步有效地减少了同步开销,增加了系统的并行性,同时也避免了进程的死锁,消除了优先级的翻转。

(2)Cache内核[9]是另外一个在操作系统内核中实现锁自由同步的例子。该内核的实现与Synthesis内核不同,它是先设计出系统所需的数据结构,然后再从基本的非阻塞算法中选出一种来实现。其中有代表性的非阻塞方法有:直接实现、单服务线程、部分拷贝、帮助策略以及增加辅助信息位。由于信号是该系统中进程间通信的基本实现形式,所以选择使用非阻塞同步有效地避免了进程死锁,减少了维护信号所需的开销,使得该操作系统的性能大大提高。

(3)由于非阻塞同步在实时系统中的表现更为突出,文献[5]的作者就抓住这一特点实现了一个非阻塞的实时操作系统内核Fiasco,该系统采用的是面向对象的设计方法。首先把内核中的对象分为局部和全局两种,然后根据对象的不同特点以及访问频度的差异,分别采用锁自由和等待自由的非阻塞同步方法来实现它们,以达到实时性和系统开销的平衡。该文章的一个创新之处在于提出了一种基于优先级继承的锁机制结合适当的帮助策略来实现等待自由的非阻塞同步的思想,该同步方法的使用前提是进程不能等待临界区内的事件。实验证明非阻塞同步在实时系统中的实现不仅减少了同步开销,最重要的是使系统的实时性也得到了明显的提高。

3 在嵌入式操作系统中实现非阻塞同步的可行性分析

上述三种操作系统内核的非阻塞实现有力地证明了非阻塞同步在操作系统中实现的可行性,但是我们的目标是要在嵌入式操作系统中实现非阻塞同步。嵌入式系统由于其应用环境及硬件条件的限制,决定了其操作系统在占用的硬件资源和提供的基本功能等方面与通用操作系统不同,所以我们要从非阻塞同步的前提条件——同步原语和类型稳定的存储器管理两方面出发来分析非阻塞同步在嵌入式操作系统中实现的可能性。

(1)由于我们的嵌入式操作系统最终将运行于 Power PC 微处理器上,而 Power PC 处理器只提供 LL-SC 原语,但是大多数高效的非阻塞算法都是基于 CAS 或 DCAS 原语的。虽然运行嵌入式系统的微处理器在同步原语的提供类型上有这种差距,但幸运的是,CAS 或 DCAS 原语是可以用 LL-SC 原语进行软件模拟的(实现代码如图 3 所示)[10]。同时在单处理器上可以用关中断的方法来保证操作的原子性,在这种情况下,关中断的时间不会很长,所以不会影响到对外界中断的响应。

```
boolean CAS(val * add,val old,val new)
{atomically
  {do
  {temp = LL(add);
  if(temp ! = old) return false;
  }while (!SC(add,new));
return true;
  }
}
```

图 3 CAS 原语的软件实现(DCAS 原语的软件实现既可以 LL-SC 为基础也可以 CAS 为基础)

(2)我们的嵌入式操作系统使用的是 Linux 2.4.17 的内核,该内核的存储管理部分有关存储空间分配与回收机制中有一个叫做 Slab 的算法。该算法与类型稳定的存储器管理方法很相似,它采用面向对象的分配原则,其中最主要的思想被称为 Object-Caching,即对象缓存,它把一种对象的所有实例都存在同一个缓冲区中。当需要申请新的对象空间时,就从缓冲区中查找空闲的对象空间,如果没有就从通用的存储器中为该对象分配一新的空间;当释放时就把该空间收归缓冲区所有。因此可以把这种算法看作是 T_{stable} 为无穷大的类型稳定的存储器管理。当然,在系统资源不足的时候,Slab 算法也允许将一部分未用的缓冲空间释放,以缓解系统压力[9]。利用该算法,可以大大降低我们实现非阻塞同步的难度。

(3)我们再来深入讨论一下其他方面的可行性。①由于锁自由的同步开销比关中断的同步开销还要小,所以很适合于像嵌入式这种硬件资源严重缺乏的系统。②嵌入式系统一般都是基于进程优先级的系统,因此把优先级继承的锁机制当作等待自由的同步算法来实现也是有基础的。③嵌入式系统的实时性使得高优先级进程的任何操作对低优先级进程来说都是原子的,这时我们就可以用通用的读写操作来代替同步原语,从而减小了系统开销。④由于嵌入式系统是以应用为中心的专用系统,所以我们可以更早地确定系统内的任务数,以便于根据任务特点合理选择锁自由或等待自由的同步算法。

另外需要特别强调的一点是:非阻塞同步并不意味着进程永远不会阻塞,而是说整个系统不会停顿在不进行任何有效操作的状态。它的优势主要在于可以避免死锁与优先级翻转,同时有效隔离死进程,提高系统的健壮性和容错性。

4 结束语

为了克服传统的阻塞同步机制易造成进程死锁和优先级翻转的问题,本文介绍了一种新的同步机制——非阻塞同步;在详细讨论了它的实现条件及其实现技术的基础上,提出了要把非阻塞同步用于嵌入式操作系统的思想,这是与以往所有的非阻塞同步研究不同的地方。由于嵌入式操作系统和通用操作系统的不同,本文又深入分析了非阻塞同步在嵌入式操作系统中实现的可行性问题及其在性能方面的优势。下一步的工作,我们将会从实现的角度进一步证明该思想的正确性。

参考文献:

[1] 李善平,刘文峰,李程远,等. Linux 内核 2.4 版源代码分析大全[M]. 北京:机械工业出版社,2001.147-173.

[2] Michael Greenwald. Non-blocking Synchronization and System Design, PhD Thesis, Stanford University Technical Report STAN-CS-TR-99-1624[EB/OL]. http://citeseer.nj.nec.com/greenwald99nonblocking.html,1999-08.

[3] T E Anderson. The Performance of Spin Lock Alternatives for Shared-memory Multiprocesssors[J]. IEEE Transactions on Parallel and Distributed System,1990,1(1):6-16.

[4] P M Herlihy. Wait-Free Synchronization[J]. ACM Transactions on Programming Languages and Systems,1991,13(1).

[5] Michael Hohmuth, Hermann Hartig. Pragmatic Non-blocking Synchronization for Real-time Systems[C]. Proceedings of the 2001 USENIX Annual Technical Conference.

[6] Valois John. Lock-Free Linked Lists Using Compare-And-Swap[C]. Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ont. Canada, 1995. 214-222.

[7] Timothy L Harris, Keir Fraser & Lan A Pratt. A Practical Multi-word Compare-And-Swap Operation[C]. Proceedings of the 2002 IEEE Symposium on Distributed Computing.

[8] Massalin Henry, Carleton Pu. A Lock-Free Multiprocessor OS Kernel, Technical Report CUCS-005-91, Columbia University[EB/OL]. http://citeseer.nj.nec.com/massalin91lockfree.html,1991-10.

[9] Michael Greenwald et al. The Synergy between Non-blocking Synchronization and Operating System Structure[C]. 2nd Symposium on Operating Systems Design and Implementation(OSDI'96), Seattle, WA, Berkeley, CA, USA, 1996. 123-136.

[10] Maged M Michael, Michael L Scott. Non-blocking Algorithms and Preemption-safe Locking on Multiprogrammed Shared Memory Multiprocessors[J]. Journal of Parallel and Distributed Computing, 1998, 51(1):1-26.

作者简介:

张丽,女,硕士生,主要研究方向为嵌入式操作系统;李仁发,男,教授,主要研究方向为分布式计算与虚拟技术、嵌入式系统、计算机体系结构、计算机网络;彭蔓蔓,女,副教授,主要研究方向为计算机体系结构、片上系统(SOC)研究与设计、嵌入式系统;凌纯清,女,硕士生,主要研究方向为嵌入式系统。