第31卷 第21期 计 算 机 工 程 2005年11月
Vol.31 № 21 Computer Engineering November 2005
·软件技术与数据库· 文章编号：1000—3428(2005)21—0084—03 文献标识码：A 中图分类号：TP316.2

非阻塞同步在嵌入式操作系统中的实现

张 丽[1,2]，王 兴[1]，郝身刚[1]，彭蔓蔓[2]
（1.南阳师范学院计算机科学系，南阳 473000；2.湖南大学计算机与通信学院，长沙 410082）

摘 要：提出了把非阻塞同步机制应用于嵌入式操作系统的新设想，同时通过修改嵌入式操作系统内核源码的方法对其进行了实现。针对内核中共享资源的不同特点，综合使用了锁自由和等待自由两种不同性质的非阻塞同步策略，同时改进了锁自由的同步算法，并对原有的等待自由同步算法进行简化，使新内核具有较小的同步开销和较好的实时性能。

关键词：非阻塞同步；嵌入式操作系统；锁自由；等待自由

Implementation of Non-blocking Synchronization in Embedded Operating System

ZHANG Li[1,2], WANG Xing[1], HAO Shengang[1], PENG Manman[2]
（1.Department of Computer Science, Nanyang Normal College, Nanyang 473000；
2.College of Computer and Communication, Hunan University, Changsha 410082）

【Abstract】 This article puts forward a new idea, which is to apply non-blocking synchronization to embedded operating system and realizes it through reworking source codes of kernel. In addition, due to the different characteristic of mutual resources in kernel, this paper combines lock-free synchronization with wait-free synchronization, improves the older lock-free synchronization arithmetic and simplifies the older wait-free synchronization, at last turns out that the new kernel has lesser synchronization overheads and better real-time performance.

【Key words】 Non-blocking synchronization; Embedded operating system; Lock-free; Wait-free

嵌入式系统的操作系统和应用程序位于同一个地址空间[1]使得存取同一个内存地址的可能性相对增加，这对系统的同步机制提出了更高的要求。如果同步问题处理不当的话，就会引起程序出错，导致系统失效。此外，由于嵌入式系统对时间期限的要求比较高，基于优先级的调度策略使用得较为普遍，因此如果处理不好进程间的协作关系，就会使系统错过时间限制，导致系统崩溃。

上述问题表明同步机制在嵌入式操作系统中的作用尤为重要。然而传统的锁同步机制容易引起进程的死锁以及进程优先级的翻转，严重影响了嵌入式系统的实时性和稳定性。

非阻塞同步机制是一种有效克服传统同步方法的上述缺点的新型同步机制，它不仅具有较小的同步开销，而且又避免了进程的死锁和优先级的翻转，使系统具有较好的健壮性和实时性[2]，因此本文提出了要把非阻塞同步应用于嵌入式操作系统的新观点（在文献[8]中分析了该想法的可行性）。

1 非阻塞同步及其相关研究

1.1 锁自由和等待自由同步

非阻塞同步机制的研究最早始于20世纪90年代初[3,5]，最初它是为共享存储区的多处理器系统而设计的。该同步算法太复杂，到目前为止，实现非阻塞同步机制的通用操作系统内核只有3例。它们分别是Synthesis内核、Cache内核和Fiasco内核。在嵌入式操作系统中，非阻塞同步机制的应用还没有开始。

非阻塞同步机制又细分为锁自由同步和等待自由同步两种。锁自由同步的实现完全不用锁，它把对共享资源的互斥访问操作和修改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[3]。该同步机制的好处是，它避免了进程死锁的发生，提高了系统的健壮性[3]。实现锁自由同步需要两个基本前提——修改内存的原子指令和类型稳定的存储器管理（TSM）[6]。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DCAS（double compare-and-swap）,而这种原语在多种通用的（微）处理器中都不提供，因此目前研究DCAS原语的软件实现问题成为非阻塞同步领域内的一个热点[2, 7]。

等待自由同步的特点是：在访问临界区之前如果进程检测到冲突，那么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然后再运行，而是通过某种帮助策略帮助临界区内的进程先完成操作，然后再执行自己的临界区操作。这样一来，不仅避免了低优先级进程的饥饿而且还不会出现优先级翻转的问题[5]。文献[4]指出可以把等待自由同步看作是带有帮助策略的锁机制（比如说带有优先级继承策略的锁机制），条件是临界区内的操作不会阻塞。

1.2 原子内存修改指令（同步原语）

原子修改内存指令CAS(compare-and-swap)和LL-SC(load-linked/ store-conditional)是实现非阻塞同步的基础。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。其中使用较为普遍的是CAS原语，它的compare部分用来检测不同进程同时访问临界区时的冲突，swap部分用来对临界区进行修改操作。一些数据结构（像

作者简介：张 丽（1978—），女，助教，主研方向为嵌入式操作系统；王 兴，讲师；郝身刚，助教；彭蔓蔓，副教授

收稿日期：2004-08-31 **E-mail：**nythhsg@163.com

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。随着非阻塞同步机制日益受到人们的关注，越来越多的（微）处理器开始支持这些原子指令。比如 MIPS，ALPHA，PowerPC 处理器提供 LL-SC 原语；Pentium, SPARC，X86 提供 CAS 原语[2]。

2 非阻塞同步在嵌入式操作系统中的实现分析

2.1 设计目标及设计原则

我们的目标是通过研究非阻塞同步在嵌入式操作系统中的实现，最终使系统获取较低的同步开销和较好的实时性能。

由于锁自由同步算法和等待自由同步算法在实现效率和实现策略上的不同，导致它们各自的应用范围也不同。经过对这两种同步算法实现机制的研究，我们总结出二者在操作系统中的应用原则：锁自由用于对全局性简单数据（如队列、堆栈、计数器）的同步；等待自由用于对局部数据、全局性的复杂数据和较长的临界区进行同步；特别要强调的是，短临界区还是可以用关/开中断的方法进行同步，该方法属于锁自由同步的范畴。

2.2 实现非阻塞同步所面临的难题

由于原有的非阻塞同步算法的复杂度高，不适用于嵌入式系统的实现，因此要对原有的非阻塞同步算法进行改写。首先拿锁自由同步来说，大多数有效的锁自由同步算法需要使用 DCAS 原语[2]，但是该原语的系统开销几乎是 CAS 原语的两倍，同时随着处理器速度的提高，这个倍数还会大大增加[2]，该特点与嵌入式系统要求低开销的宗旨相违背。因此只能选用低开销的 CAS 同步原语来实现锁自由的同步算法，但问题是目前还没有这种现成的算法。再者，原有的等待自由同步为了实现算法中的帮助策略，要求每个进程在执行临界区之前先把自己将要执行的操作登记在一个共享存储区内，以便高优先级的进程在抢占处理器时先帮助低优先级进程完成它的临界区操作。该算法思想大大增加了系统的资源开销和等待自由算法的复杂度，因此如何简化原有的等待自由同步算法使之具有较小的同步开销，成为在嵌入式操作系统中实现非阻塞同步时要解决的又一道难题。

3 非阻塞同步在嵌入式操作系统中的简化实现

我们选择实现非阻塞同步的嵌入式目标操作系统是 Montavista Linux，它本质上就是 Hardhat Linux 的嵌入式实现。之所以选用这一款嵌入式 OS，是因为 Linux 的源代码开放、相关资源非常丰富，为我们最终实现非阻塞的同步机制提供了方便。该操作系统使用的是 Linux2.4.17 内核，运行于 Power PC 的微处理器目标板上。

3.1 通用的简化方法

一般来说，嵌入式操作系统使用的都是基于优先级的调度策略，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，这时可以用通用的读写操作代替复杂的同步原语，减小系统开销，简化同步算法。

3.2 锁自由同步的改进

由锁自由同步的应用原则可知，锁自由适用于队列结构的同步操作。通过对 Linux2.4.17 内核中同步机制的研究发现：等待队列这一全局性数据结构在 linux 同步机制中的作用至关重要。又由于等待队列属于比较敏感的数据结构，容易引起临界区问题，因此使用锁自由同步对等待队列进行改写将会对系统性能产生很大影响。

Linux 中等待队列使用的是双向链表结构，使用原有的锁自由同步算法实现对等待队列的同步操作要用到 DCAS 原语。由 2.2 节的分析可知，嵌入式系统的低开销性要求系统不能使用代价昂贵的 DCAS 原语来实现等待队列的锁自由同步算法，而只能使用简单的低开销原语——CAS。由于目前没有此类现成的低开销算法，因此本文的任务之一就是要解决用 CAS 原语实现等待队列的锁自由同步算法的问题。通过对双向链表相关操作的锁自由同步原有算法的仔细分析，发现插入和删除节点的两端节点位置的不确定性是该算法使用 DCAS 原语对其进行同步保护的主要原因。DCAS 原语的原子性可以保证插入和删除操作的完整性和一致性。由于 Linux 中等待队列双向链表结构的入队列操作都集中在头指针节点处进行，即头指针节点是插入操作的一个固定点，因此，在每次入队列操作完成后只需要检查头指针的指向就能判断是否发生了冲突，于是只用 CAS 原语就可以完成等待队列插入操作的锁自由同步算法,算法框图如图 1。同时由于 Linux 中等待队列双向链表结构的首尾互连性，即该数据结构是一循环链表，因此不需要在队列操作中进行空队列的判断，从而可以简化等待队列的锁自由同步算法。至于等待队列的删除操作，同样是由于它的首尾相连性，再加上插入操作总在头指针处进行的特点，我们只需要用一个 CAS 原语就能实现其删除操作的锁自由同步算法。原因是只有当删除节点正好位于头指针后的相邻位置时，删除操作才有可能与插入操作发生冲突。此时只要检测头指针是否指向了删除节点就能判断是否有冲突发生，该算法框图如图 2。

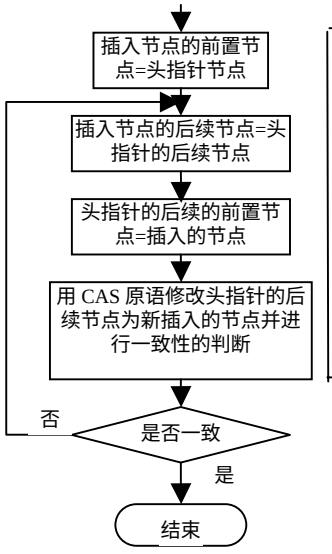

图 1 改进后的入队列操作的锁自由同步算法框图

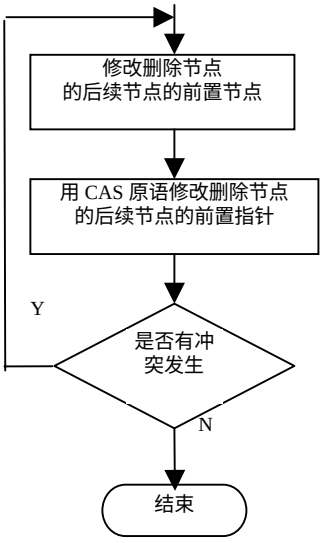

图 2 改进后的出队列操作的锁自由同步算法框图

3.3 等待自由同步的简化

由 2.2 节的分析知道，原有复杂的等待自由同步算法不能应用于嵌入式系统，因此需要一种简化的算法来代替它。经过研究发现，大多数嵌入式系统都有实时性强、单处理器的特点，而一种叫做“帮助锁”的算法[4]正好满足在嵌入式系统下实现简化的等待自由同步算法的要求，该算法在空间复杂度和时间复杂度上都远远小于原来的等待自由同步算法。

图 3 为帮助锁的算法框图。从帮助锁的算法框图中发现，帮助锁的算法实现与互斥锁（二进制信号量）的实现非常相似，二者都会在进程申请不到临界区时进入等待队列，交出 CPU 的控制权，由系统对其他进程进行调度。不同的是得不

到互斥锁的进程会阻塞，等待占有临界区的进程完成临界区操作，此时如果有中等优先级的进程抢占了拥有互斥锁的进程的 CPU，那么就会发生优先级的翻转；而得不到帮助锁的进程会把自己的优先级借给获取锁的进程并让它继续运行，此时只有更高优先级的进程才能抢占拥有锁的进程。否则，拥有锁的进程会一直运行，直到完成了临界区的操作才会把 CPU 再还给刚刚帮助它的进程，这样的话就没有优先级翻转的发生，从而使得帮助进程完成操作的时间提前。

经过上述分析，得出的结论是，用帮助锁算法来取代 Linux2.4.17 内核中二进制信号量的上锁 mutex_lock()和开锁 mutex_unlock()操作，并以此代替等待自由同步算法在 Linux 中的简化实现。

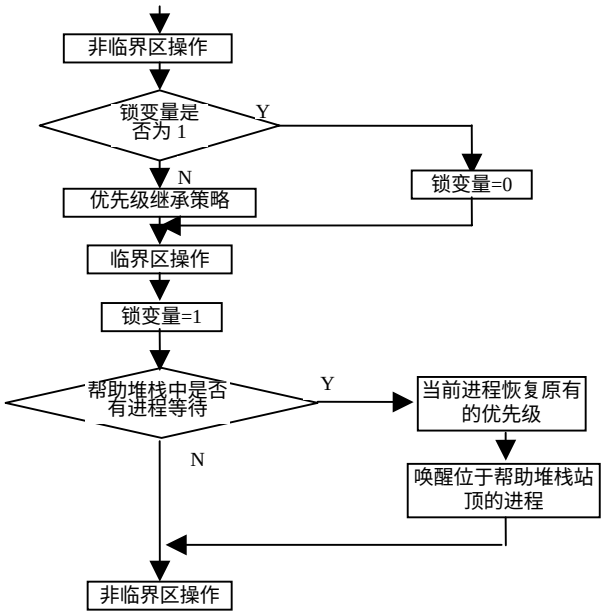

图 3 帮助锁算法框图

4 系统的性能指标

为了提供新内核的性能参数，先后做了两个实验：第 1 个实验测试所有同步技术的系统开销；第 2 个实验测试新旧内核的实时性。

4.1 同步技术的系统开销

我们编写了一系列的用户程序用以测试系统中所有同步技术（包括 atomic_t 型的原子操作、CAS 原语、DCAS 原语、开/关中断、帮助锁以及信号量）的开销。实验结果如表 1 所示。由该结果可以看出：atomic_t 型原子操作使用的系统时间最少，其次是 CAS 原语，然后才是关中断。由于原子操作只能用于单个内存单元的修改操作，使用范围受到限制，因此在对类似队列操作这样的短临界区进行保护时还是要选择 CAS 原语。在有竞争和没有竞争的情况下，帮助锁的系统开销都会比信号量的开销要大一点点，但是它可以避免优先级翻转，因此用帮助锁来代替信号量的工作还是有意义的。

表 1 各同步技术的开销

同步操作	系统时间(μs)
Atomic_t 型原子操作	6.38
CAS 原语	6.69
DCAS 原语	12.51
开/关中断	6.83
没有竞争条件的帮助锁	15.59
有竞争条件的帮助锁	1 217.87
没有竞争条件的信号量	14.05
有竞争条件的信号量	1 157.75

4.2 新旧内核的实时性

操作系统内核实时性的一个最重要的指标就是系统的中断延迟，它是指中断发生那一刻起到内核的中断服务子程序的第 1 条指令开始执行这段时间的时间间隔。为了测试新旧内核的中断延迟，用 PowerPC 目标板上的 1 个 32b 的 decrementer 异常来模拟外部中断源。我们的硬件平台是一个 50MHz 的 PPC860 的微处理器和 Nete860 达尔文的主板，软件是 Montavista Linux 的嵌入式操作系统。在用户模式下，分别运行了 Dhrystone1.1, Fibonacci, Sieve 3 个基准程序来测试锁自由同步的中断延迟，同时又通过自己编写的应用程序测试了等待自由同步的中断延迟。最后的实验结果如表 2、表 3 所示。虽然等待自由同步使内核的中断延迟有所增加，但是锁自由同步在更大程度上减小了内核的中断延迟。最终的结果是，非阻塞同步提高了内核的实时性，从而证明在嵌入式系统中使用非阻塞同步确实是有意义的。

表 2 3 个基准程序测试结果

基准程序	内核修改前	内核修改后
Dhrystone	5.612 8μs	5.254 4μs
Fibonacci	4.390 4μs	4.028 8μs
Sieve	8.265 6μs	7.721 6μs

表 3 用户程序测试的结果

	内核修改前	内核修改后
用户程序	4.896μs	4.953 6μs

5 结束语

本文研究了非阻塞同步的算法思想，给出了锁自由同步和等待自由同步的应用原则。同时针对嵌入式系统的特点，分别对原有的锁自由同步算法和等待自由同步算法进行了改进和简化。最后用实验证明把非阻塞同步应用于嵌入式操作系统新观点的有效性。目前我们的研究工作还存在一些不足，对实验结果分析还处于最初的阶段。因此下一阶段的任务是要继续探讨非阻塞同步机制的内核实现并通过对使用非阻塞同步的周期性任务集的调度分析从理论上说明非阻塞同步对系统实时性的影响，同时对实验结果作进一步的分析。

参考文献

1 Scheduling and Synchronization in Embedded Real-time Operating Systems.http://www.cs.ucsd.edu/classes/wi01/cse221/OSSurveyW01/papers, 2001-03

2 Greenwald M. Non-blocking Synchronization and System Design [PhD thesis]. http://citeseer.nj.nec.com/greenwald99nonblocking.html, 1999-08

3 A Lock-free Multiprocessor OS Kernel. http://citeseer.nj.nec. com/massalin91lockfree.html, 1991-06

4 Hohmuth M, Hartig H. Pragmatic Non-blocking Synchronization for Real-time Systems. In: Proc. of the 2001 USENIX Annual Technical Conference, 2001: 91-101

5 Herlihy P M. Wait-free Synchronization. ACM Transactions on Programming Languages and Systems, 1991, 13(1): 276-290

6 Greenwald M, Cheriton D. The Synergy Between Non-blocking Synchronization and Operating System Structure. In: 2nd Symposium on Operating Systems Design and Implementation, 1996: 123–136

7 Harris T L, Fraser K, Pratt I A. A Practical Multi-word Compare-and-Swap Operation. In: Proc. of the 2002 IEEE Symposium on Distributed Computing, 2002: 265-279

8 张 丽, 李仁发, 彭蔓蔓等. 嵌入式操作系统中非阻塞的同步机制. 计算机应用研究, 2004,21(3)