一种改进的双令牌互斥算法

吴小惠[1],洪 龙[1],杨 扬[2]
(1.南京邮电大学 计算机学院,江苏 南京 210003;
2.中兴通讯股份有限公司,江苏 南京 210012)

摘 要:互斥是解决分布式系统中资源申请的相互冲突、实现资源共享的一种有效方法。文中对目前的一些互斥算法进行了介绍,并对集中式互斥算法和分布式互斥算法进行了讨论,并分析了其特点。在简要介绍令牌环算法和双令牌算法的思想和步骤后,提出一种改进的双令牌算法。详细叙述了改进算法的设计思想和实现步骤,对性能进行了分析并给出算法示例。讨论结果表明,与原有的双令牌算法相比,新算法可以以更高效率有效检测令牌丢失并实现令牌重构。

关键词:互斥;双令牌;令牌丢失;令牌重构
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2011)04-0040-04

An Improved Mutual Exclusion Algorithm with Dual Tokens

WU Xiao-hui[1], HONG Long[1], YANG Yang[2]
(1. College of Computer, Nanjing University of Posts & Telecommunications, Nanjing 210003, China;
2. ZTE Corporation, Nanjing 210012, China)

Abstract: Mutual exclusion in distributed systems is an effective method of solving the conflict with the application of resources and sharing resources. Some of the current mutual exclusion algorithms are introduced, and centralized mutual exclusion algorithm and distributed mutual exclusion algorithm and their characteristics are discussed. An improved algorithm for token loss detection was proposed after the simple token ring algorithm and the algorithm for token loss detection were raised. The improved algorithm's design and the steps of the implementation have been described, moreover, the analysis of the performance and an example was given. The discussion showed that the new algorithm can be more effective than the dual tokens algorithm in detecting token loss and token reconstruction.

Key words: mutual exclusion; dual-tokens; token loss; token reconstruction

0 引 言

分布式系统的基础是多进程的并发和协作,这要求进程能够在同一个时段访问相同的资源。为了保证这种访问的有效性,需要采用互斥访问方法。保证在一个时刻只能有一个进程访问共享资源是互斥的主要目标。

实现互斥的算法一般分为两种不同类型:基于令牌的解决方法和非令牌的解决方法。非令牌的算法主要分为集中式算法和分布式算法。目前,国际上已有一些分布式的互斥算法,如 Lamport 算法、Ricart and Agrawala[1]算法、Maekawa[2]算法等,国内也有很多人对分布式互斥算法做了详细的研究[3-8]。还有一些基于令牌的算法,如简单令牌环算法、基于树的令牌算法[9]和基于图的令牌算法[10]等。

文中在简单介绍了集中式互斥算法和分布式互斥算法,令牌环的基本算法和双令牌互斥算法后,改进了双令牌互斥算法,使得性能更优。

1 集中式互斥算法和分布式互斥算法

集中式算法采用协调者的方法实现互斥,只有获得协调者许可的进程才可以进入临界区。算法要点如下:(1)选一个进程为协调者。(2)每个要求进入临界区的进程向协调者发出请求,协调者对所有的请求进行排队和授予许可。若临界区中已有一个进程,协调者便不能同意新的请求。(3)当占有临界区的进程从临界区退出时,向协调者发送释放临界区消息,允许其它进程进入临界区。集中式算法保证了互斥的实现——协调者在某一时刻只能让某一进程进入临界区。但是,协调者显然是算法的瓶颈所在。

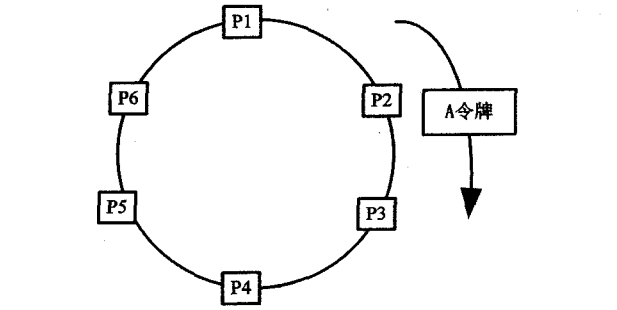
不同于集中式算法的是,在分布式算法中没有控制节点(即协调者)的概念,所有节点都是平等的。它

收稿日期:2010-10-21;**修回日期**:2010-12-27
基金项目:中兴通讯重大科研基金(nj200909080002)
作者简介:吴小惠(1984-),女,硕士研究生,主要研究方向为智能计算技术;洪 龙,教授,研究员级高级工程师,主要研究领域为分布式系统、非经典逻辑及应用。

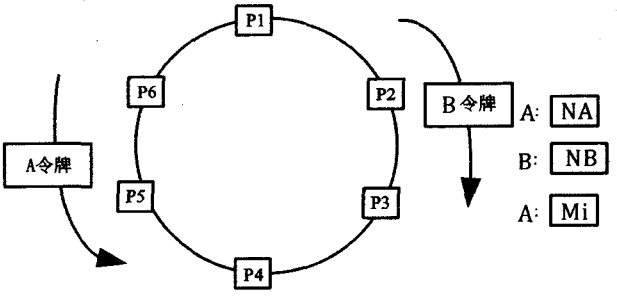
的基本思想是:一个节点如果想进入临界区,就必须得到其它节点的同意。假设所有事件都是全序的,当进程欲进入临界区时,应向其他进程广播请求,任一进程收到此请求时若:①不在临界区中也不想进入临界区,向发送者发送 OK 消息;②已在临界区中,不回答,负责将发送者排入请求队列;③要进入但尚未进入临界区,要比较自己和发来请求的时间戳的大小,如果发来的时间戳小,则向发送者发送 OK 消息,否则同②。如果所有回答均为 OK,请求进程可进入临界区,从临界区退出后向请求队列中所有进程发 OK,释放临界区[8]。

2　令牌环算法简介

在基于令牌的解决方法中,互斥是通过在进程之间传递一个特殊的消息来实现的,这个消息称为令牌。在所有的进程里只有一个令牌。一个进程当且仅当拥有令牌时可以访问临界区。当令牌被传递到某一个进程时,如果这个进程不需要访问临界区,则把该令牌传递给下一个进程,否则在访问完临界区才将令牌传递到下一个进程。简单令牌环算法如图 1(a)所示。进程有序构成一个逻辑环,令牌有序环绕前进,使得每个进程都能拥有令牌。简单令牌环算法的主要问题是难以检测令牌的丢失。虽然可以采用超时机制判定令牌的丢失,但因令牌在同一进程两次出现的时间间隙是不定的,所以超时机制有时也会导致误判令牌丢失的现象[11]。双令牌互斥算法可以解决这个问题。双令牌互斥算法[12]如图 1(b)所示。



(a)进程构成的逻辑环



(b) 双令牌互斥算法

图 1　令牌环算法

该算法使用两个令牌,其中每一个令牌负责检测另一个令牌可能的丢失,而其中的一个令牌用来控制访问共享资源,两个令牌按照相反方向沿着环访问进程。在该算法中,如果某一个进程被同一个令牌连续两次访问,则令牌的丢失被检测到。

算法描述[12]:双令牌互斥算法使用两个令牌(A, B),每个令牌各自带有一个变量(NA,NB)。环中一共有 n 个进程($P1$ 到 Pn),每个进程 Pi 带有一个变量 $Mi(1 \leqslant i \leqslant n)$。当两个令牌相遇时,更新 NA,NB。

(1)算法刚开始时:$NA := 1$,$NB := -1$;$Mi := 0$,$1 \leqslant i \leqslant n$。

(2)当进程 Pi 接收到令牌 A 时(当进程 Pi 接收到令牌 B 时步骤一样):

if $Mi == NA$

then {令牌 B 丢失:令牌 A 在没有改变 NA 的情况下绕令牌环网转了一周,也同时表示在令牌 B 这段时间内没有访问过 Pi;} 重构令牌 B;

else {令牌 B 无丢失} $Mi := NA$;

(3)当两个令牌相遇时:

$NA := NA + 1$;$NB := NB - 1$

(4)当进程 Pi 重构令牌 B 时(当进程 Pi 重构令牌 A 时步骤一样):

$NA := NA + 1$;$NB := -NA$;

{这个时候进程 Pi 持有两个令牌}

该算法同简单令牌环算法一样,可以避免死锁和无饥饿现象。此外,该算法还可以很容易地扩展到 K 令牌的情形。双令牌互斥算法可以有效地检测到令牌的丢失并重构,但是算法的性能可以进一步改善。

3　改进的双令牌互斥算法

提出的双令牌算法中的两个令牌按相反方向沿着环访问进程。A 令牌有两个功能:(1) 持有 A 令牌的进程可以访问共享资源;(2) 检测 B 令牌是否丢失。B 令牌也有两个功能:(1) 在每个进程上保存离自己最近的(沿 A 令牌访问方向) 请求进入临界区的进程;(2) 检测 A 令牌是否丢失。A 令牌里有一个变量 NA,它保存连续访问 Mi 值为 1 的进程的个数。B 令牌里有两个变量 NB 和 PB,NB(用负值) 的功能与 NA 一样,PB 保存最新请求访问共享资源的进程的进程号(如果是 $n + 1$,表示无请求访问共享资源的进程)。进程 Pi 均带有一个变量 Mi,当最近访问的为 A 令牌,Mi 赋值为 1;反之,赋值为 $-PB$。

算法的检测令牌丢失机制为:当 A 令牌访问进程 Pi 时,将 Mi 置为 1;当 B 令牌访问进程 Pi 时,将 Mi 置为负数。则当 A 令牌检测连续 n 个进程的 Mi 值(即所有进程的 Mi 值) 均为正数时,可判定 B 令牌丢失;反之,则当 B 令牌检测到连续 n 个进程的 Mi 值为负数时,

可判定 A 令牌丢失。

3.1 算法步骤

(1)算法刚开始时:$NA=0$,$NB=0$,$PB=n+1$;$Mi=n+1$,$1\leqslant i\leqslant n$。

(2)当进程 Pi 接收到令牌 B 时,步骤如下:

①if Mi 为负数或者 $Mi=n+1$

then {上一次访问该进程的令牌为 B 令牌} $NB=NB-1$;

else $NB=0$;

②if $NB=-n$,

then{令牌 A 丢失} 重构令牌 A;

③$Mi=-PB$;

④if 进程请求进入临界区

then $PB=i$(i 为该进程的进程号,PB 保存最新请求访问共享资源的进程的进程号);

⑤将令牌 B 发往进程 $P((i+1)\bmod n)$。

(3)当进程 Pi 接收到令牌 A 时,步骤如下:

①if Mi 为正数

then {上一次访问该进程的令牌为 A 令牌} $NA=NA+1$;

else $NA=0$;

②if $NA=n$

then{令牌 B 丢失} 重构令牌 B;

③if 进程请求进入临界区

then 进程访问资源,访问完退出临界区;

④temp = Mi,$Mi=1$;

⑤if temp 为正数或者为 $-(n+1)$,

then 下一个访问的进程为 $P((i-1+n)\bmod n)$;

else 下一个访问的进程为 $P(-Mi)$;

将令牌 A 发往下一个访问进程。

重构 A 令牌时:令 $NA=0$,$NB=0$;重构 B 令牌时:令 $NB=0$,$NA=0$,$PB=n+1$。

3.2 算法分析

由于只有拥有 A 令牌的进程才能访问共享资源,而该算法在某个时间点上只能使一个进程拥有 A 令牌,所以可以实现互斥;由于令牌环单一控制点的特性,不存在两个进程间相互等待的现象,也即不存在死锁;由于 B 令牌按照顺时针方向将请求资源的进程号放在下几个访问到的进程里直到遇到新的请求资源的进程,而 A 令牌按照一个方向访问进程,所以该算法可以保证无饥饿现象。

算法的特点:在 B 令牌经过某一进程时,首先将 B 令牌的 PB 记录在进程里,接着如果进程请求访问共享资源,则将 PB 置为该令牌的令牌号。因而每个进程里记录的是 A 令牌传递方向上离自己最近的请求令牌的进程号,这样 A 令牌就可以直接从某一进程跳到(沿 A 令牌传递方向上的)离自己最近的请求令牌的进程,从而提高了效率。

3.3 算法示例

(1)令牌环初始的时候(此时令牌环网有 6 个进程),令 $PB=n+1=9$,$Mi=n+1=9$;$P1$ 产生令牌 A、B,令 $NA:=0$,$NB:=0$,如图 2 所示。

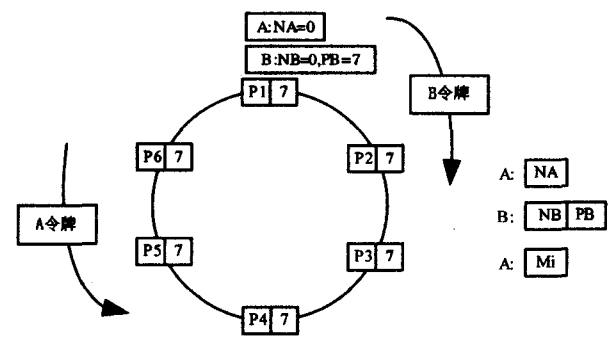


图 2 令牌环初始状态

(2)假设该令牌环里 $P2$ 请求令牌,接下来的步骤如图 3 所示。

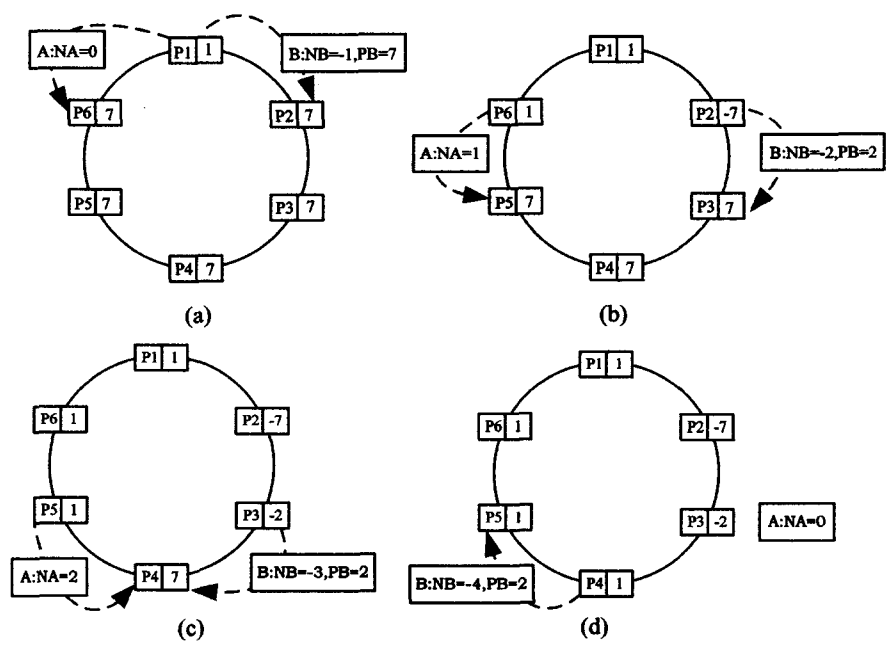


图 3 算法实例步骤

步骤 1,如 a 图所示:假设 $P1$ 先生成 B 令牌:① $M1=7=>NB=NB-1=-1$;②$M1=-PB=-7$;③将令牌 B 发往进程 $P2$。接着 $P1$ 生成 A 令牌:① $M1=7=>$

$NA=0$;②temp = $M1=-7$, $M1=1$;③令牌A下一个访问的进程为$P6$;将令牌A发往下一个进程。

步骤2,如b图所示:进程$P2$接收到令牌B:① $M2=7 => NB=NB-1=-2$;② $M2=-PB=-7$;③该进程请求访问共享资源 => $PB=2$;④将令牌B发往进程$P3$。与此同时进程$P6$接收到令牌A: ① $M6=7 => NA=1$;②temp = 7, $M6=1$;③令牌A下一个访问的进程为$P5$,将令牌A发往下一个进程。

步骤3,如c图所示:进程$P3$接收到令牌B:① $M3=7 => NB=NB-1=-3$;② $M3=-PB=-2$;③将令牌B发往进程$P4$。与此同时进程$P5$接收到令牌A:① $M5=7 => NA=0$;②temp = 7, $M5=1$;③令牌A下一个访问的进程为$P4$,将令牌A发往下一个进程。

步骤4,如d图所示:假设进程$P4$先接收到令牌B: ① $Mi=7 => NB=NB-1=-4$;② $Mi=-PB=-2$;③将令牌B发往进程$P5$。这时进程接收到令牌A: ① $Mi=-2 => NA=0$;②temp = $Mi=-2$, $Mi=1$;③根据temp可知令牌A下一个访问的进程为$P2$,将令牌A发往下一个进程。

在令牌环中等负荷的时候,该改进算法可以加快A令牌沿着环传递的速度,从而提高算法的有效性,但是要求环里的每个进程知道环里的所有进程的地址。

4　结束语

文中分析了基本的令牌环算法和能够检测令牌丢失的双令牌算法,提出了一种改进的双令牌算法。该算法在延续原算法的检测令牌丢失的基础之上,特别是在中等负荷的情况下,提高了原算法的有效性。

参考文献:

[1] Ricart G, Agrawala A K. An optimal algorithm for mutual exclusion in computer networks [J]. Communications of the ACM, 1981, 24(7): 9-17.

[2] Maekawa M. A square-root(N) algorithm for mutual exclusion in decentralized systems[J]. ACM Transactions on Computer Systems, 1985, 2(6): 145-159.

[3] 邢　雁. 关于分布式系统进程互斥算法的研究[J]. 哈尔滨商业大学学报(自然科学版), 2003(1): 43-46.

[4] 王昕晔,叶慧娟,李　新. 分布式系统设计中的互斥问题[J]. 海军工程大学学报, 2003(5): 105-112.

[5] 李旭芳. 分布式系统中进程的同步与互斥算法讨论[J]. 计算机工程与设计, 2004(6): 935-937.

[6] 李美安,刘心松,王　征. 非稳定环境下基于竞争消息复杂度的分布式互斥节点容错算法[J]. 微计算机信息, 2005(26): 145-147.

[7] 石　敏,金　辉. Recart&Agrawla 同步互斥算法的改进[J]. 计算机应用研究, 20046): 32-34.

[8] 孙辰军,王翠茹. 分布式系统进程互斥算法的研究与改进[J]. 微计算机应用, 2005, 26(2): 139-141.

[9] Raymond K. A distributed algorithm for multiple entries to a critical section [J]. Information Processing Letters, 1989, 30(2): 189-193.

[10] Nishio S, Li K F, Manning E G. A resilient mutual exclusion algorithm for computer networks [J]. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(7): 344-355.

[11] Lynch N A. 分布式算法[M]. 舒继武,李国东,余华山,等译. 北京:机械工业出版社, 2004.

[12] Misra J. Detecting termination of distributed computations using markers[C]//Proc. ACM Symposium on PODC. Montreal: [s. n.], 1983: 290-294.

(上接第39页)

下一步的工作将围绕这部分内容展开,考虑如何减少人工的参与,最大程度地提高自动化测试的效率[12]。

参考文献:

[1] 王　璞,张臻鉴,王玉玺. 基于覆盖的软件测试技术在实时嵌入式软件中的应用研究[J]. 计算机工程与设计, 1998, 19(6): 45-49.

[2] 严建峰,李伟华,刘　明. 多 Agent 系统任务分配的研究[J]. 计算机工程, 2009, 35(11): 221-222.

[3] 郑明辉,周慧华,马光致. 面向对象的软件测试多 Agent 系统研究[J]. 计算机应用, 2004, 24(5): 94-97.

[4] Landi W. Undecidability of Static Snalysis[J]. ACM Lett. Programming Languages Syst, 1992, 1(12): 32-46.

[5] 裘杭萍,肖登海,连向磊,等. 一种新的基于 Agent 的体系机构[J]. 计算机技术与发展, 2010, 20(1): 50-53.

[6] 周　清,林　拉. 基于 Agent 技术的在线测试系统研究与设计[J]. 计算机技术与发展, 2007, 17(10): 184-188.

[7] 樊　玮,陈增强,袁著祉. 基于 Agent 的分布式数据库设计及其在飞行计划系统中的应用[J]. 南开大学学报, 2004, 37(3): 49-54.

[8] 刘　军,刘卓军. 改进 lex 和 yacc 以开发基于事件驱动的语法分析类软件[J]. 计算机工程与设计, 1999, 20(1): 17-23.

[9] 乔文军. 嵌入式软件测试平台的研究与实现[D]. 南京:南京航空航天大学, 2007.

[10] Zhu H. A formal analysis of the subsume relation between software test adequcy criteria [J]. IEEE Trans. on Software Eng., 1996, 22(4): 248-255.

[11] Frankl P. A formal analysis of the fault-detecting ability of testing methods[J]. IEEE Trans. on Software Eng., 1993, 19(3): 202-212.

[12] 赵欣培,李明树,王　青,等. 一种基于 Agent 的自适应软件过程模型[J]. 软件学报, 2004, 15(3): 348-359.