一种基于消息槽的 K 资源互斥算法①

A K – Mutual Exclusion Algorithm Based on Message – Slot

赵淑梅　（郑州铁路职业技术学院 河南郑州 450052）
秦　杰　（河南工业大学信息科学与工程学院 河南郑州 450052）

摘　要： 在银行系统等有多个实体同时活跃的分布式应用中，必须妥善解决不同实体对资源的需求，即同步与互斥。本文基于令牌的 K 资源互斥算法，提出了基于消息槽的 K 资源互斥算法，该算法能有效满足 K 资源分布式环境下同步与互斥的要求。

关键词： 实体 分布式应用 同步与互斥 K 资源互斥 消息槽

1　引言

日常生活中有很多分布式应用，如银行系统、售票系统、图书预定系统等[1]。在这些应用中不可避免的会涉及到不同实体对共享资源的访问，而且往往是对多个资源的访问。此时必须处理好这些实体对资源的访问请求，即进行互斥；否则，系统就有可能进入混乱状态甚至崩溃。

分布式应用的前提是需要一个好的互斥算法的支撑[2]。关于 1 资源互斥以及 K 资源互斥的问题，已经有很多算法[3][4]，其中有些具有良好的实用性，如基于令牌的 K 资源互斥算法和基于仲裁集的 K 资源互斥算法等[5][6]。本文在分析吸取现有成果的基础上，提出了基于消息槽的 K 资源互斥算法，并进行了分析，分析结果表明，该算法能有效满足 K 资源分布式环境下同步与互斥的要求。本文后续内容安排如下：第 2 部分描述 K 资源互斥算法；第 3 部分给出基于该算法的一个应用实例。第 4 部分结合应用实例给出算法的性能分析，对该算进行讨论。

2　K 资源互斥算法

首先描述消息槽[6]的含义。所谓消息槽，就像一列运送货物的火车，车上有多节车厢，每节车厢可以容纳一批货物；当发货人要把自己的货物给收货人时，发货人就在站台等着货车的到来，然后在车上找到一节空闲的车厢，把货物放上去；火车继续往前，到了收货人那里之后，收货人把货物从存放货物的车厢中卸下来，则原来存放货物的这节的车厢也就再次空闲可用。这里所说的消息槽，就像火车车厢，槽中共分出 K 槽位，每个槽位代表对一个资源的操作情况，即该资源目前是否被使用。当一个节点要使用资源时，它就等着消息槽的到来，然后在其中找出若干空闲的槽位，并声明，这些资源已被占用。使用完之后，再将这些槽位释放。

2.1　前提与假设

为了描述本文的算法，结合实际应用对算法作出以下约定[6]：

1. 系统中所有节点组织成一个逻辑连接为环形的结构，消息槽，就像一个令牌，沿着这个逻辑环在系统中循环往复的传送。

2. 消息槽中共有 K 槽位，相应的，系统中有 K 资源，每个槽位代表对一个资源的使用情况。初始时，每个槽位的内容都为空。

3. 在消息槽的末尾，另设一项数据，记录当前别的某个节点所需要的资源数。平常该项数据固定为 0，当某个节点发现消息槽中空闲的资源数不能满足自己的要求时，就把这一位填上自己所需要的资源数，等到释放资源时，再将这项数据重新清 0。

4. 为防止某些节点长期占用资源，导致另一些节

① 基金项目：863 课题高性能网络服务器评测（No. 2003AA111020）；河南工业大学科研项目（No2006BS009）

点被饿死，并为提高资源的利用率，采用时间片（比如消息槽转了 n 圈）方法[7]，当一个节点使用资源一定时间后，必须将资源释放，若还要使用，则需重新申请。

2.2 算法的执行过程描述

算法的执行主要包括以下三个过程：请求资源、释放资源、不申请也不释放资源。

1. 请求资源

若节点 i 申请使用 a 个资源，则 i 等到消息槽到达本节点后，然后如果消息槽中空闲的槽位数能满足自己的要求，即空闲槽位数 f > =a，并且消息槽的末尾数据项为 0；或者消息槽末尾数据项为 b，但 f > =a + b，则节点 i 从这 f 个资源中任选 a 个，并在他们对应的消息槽槽位中填上本节点的进程号，表明这些资源已经被节点 i 占用了。

同时，如果节点 i 在早些时候曾经将消息槽的末尾项数据填上的话，那么，将这项数据清 0；否则直接使用资源，不必考虑消息槽的末尾项数据。

如果 f < a，并且消息槽的末尾数据项为 0，则表明系统中空闲的资源数不能满足节点 i 的要求，且没有其他节点预定资源，则节点 i 把消息槽的末尾项数据填上自己所需要的资源数，并等待消息槽的下一次到达。

否则应该有 f < a，并且消息槽的末尾数据项已经置上，则节点 i 不能做什么动作，只能把消息槽传送给下一个逻辑节点，并等待消息槽的下一次到达。

2. 释放资源

如果节点 i 完成了对资源的使用，那么它等到消息槽到达后，将自己所申请资源所对应的消息槽中的槽位清空，表明这些资源又成为可用的了。当一个节点连续使用某些资源达一定时间后，该节点必须进行资源释放过程，若还需使用，需要再次进行资源申请过程。

3. 不申请也不释放资源

如果收到消息槽，只是简单的把消息槽往下一个逻辑节点传送。

3 算法运作实例

上面给出了该算法的描述，为便于理解，下面结合一个例子具体说明算法的实际执行过程。

假设在一个系统中，共有 7 个进程节点，如图 1 所示。6 个共享资源。这 7 个进程采用上述算法，对这 6 个资源进行互斥访问。

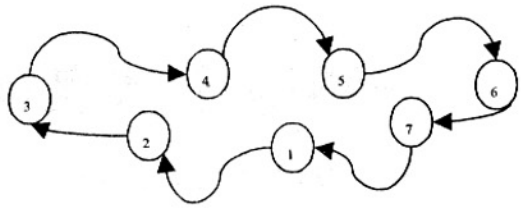

图 1　分布式系统节点的逻辑环结构

系统中所用的消息槽的格式如表 1 所示。

表 1　消息槽格式

消息槽槽位	1	2	3	4	5	6	尾项数据
内容							

消息槽首先由 1 号节点产生并发送出去。之后，消息槽到达 3 号节点，而 3 号节点需要 2 个资源已完成某项工作。而此时，消息槽中有六项资源空闲，并且尾项数据并未置上，于是节点 3 在消息槽的前 2 个槽位中填上自己的进程号 3，表明这些资源已经被自己占用了。此时消息槽的内容如下：

消息槽槽位	1	2	3	4	5	6	尾项数据
内容	3	3					0

消息槽继续往下传，并到达了 5 号节点。5 号节点需要 3 个资源已完成任务，类似的，经过检查之后，5 号节点选择了 3、4、5 三个槽位所对应的资源，并填上自己的进程号 5。此时消息槽的内容如下：

消息槽槽位	1	2	3	4	5	6	尾项数据
内容	3	3	5	5	5		0

消息槽继续传递并到达了 7 号节点这里。7 号节点也需要 3 个资源以完成作业。7 号节点检查过消息槽后，发现只有一个资源可用，于是在尾项数据中填上 (7,3)，表示 7 号节点需要 3 个资源，并把消息槽往下传。此时消息槽的内容如下：

消息槽槽位	1	2	3	4	5	6	尾项数据
内容	3	3	5	5	5		7,3

消息槽再次传到 2 号节点，2 号节点此时需要 1 个资源。此时消息槽中有一个空闲资源，但尾项数据中填的是 (7,3)，而 1 < 1 + 3，所以 2 号节点并不能使用这个空闲的资源，只得把消息槽往下传。这个过程中消息槽的内容不变。

消息槽再次传到 3 号节点，此时 3 号节点已经完成了对资源的使用，于是释放了资源，把消息槽的前两

项数据清空。此时消息槽的内容如下：

消息槽槽位	1	2	3	4	5	6	尾项数据
内容	3	3	5	5	5		7,3

然后消息槽到了节点4，节点4需要1个资源，但空闲资源只有3个，而3<3+1，所以节点4也只能把消息槽往下传。消息槽内容不变。

后来消息槽就又到了节点7，此时有3个空闲资源，刚好满足节点7的需要，于是节点7把这3个资源占用，并清除尾项数据。此时消息槽内容如下：

消息槽槽位	1	2	3	4	5	6	尾项数据
内容	7	7	5	5	5	7	0

系统按照上述算法，不断的运作下去，直到断电或者人为的切断。

4 分析与总结

该算法满足了互斥的要求。因为只有拿到资源的节点才能进入临界段，当系统剩余资源不能满足新的请求时，节点将申请不到资源，即同时处于临界段的节点所占用的资源总数不会大于K，所以算法可以满足互斥的要求。

该算法不存在死锁。因为一个节点要么一次拿到所需的所有的资源，要么一个资源也拿不到，不会出现占有了一些资源，却还在等待另一些资源的情况，也就是说，不可能同时具备发生死锁的所有条件，所以不可能死锁。

另外，算法也是不存在饿死的情况。因为，如果一个进程需要的资源数较多而无法立即得到满足的话，那么它可以在消息槽中加以声明，这样别的节点就会给它“让路”，等保证了它的请求得到满足后，别的节点才会去申请资源。所以，任何需要较多资源的进程，都会在一定的时间后得到所需要的资源，不会出现一个需要较多资源的进程一直处于等待别的进程释放资源的状态，也即不存在饿死。

此外，该算法虽然存在一点点的资源浪费，但由于有时间片限制的存在，不会出现很长时间内有大量空闲资源却不能使用的情况。而且，与几种较成功的K资源互斥算法相比，资源利用率相差不大。

综上所述，该算法实现了K资源的互斥，且不存在死锁与饿死，并且资源利用率较高。

参考文献

1 Divyakant Agrawal and Amr El Abbadi. An efficient solution to the distributed mutual exclusion problem. In Principles of Distributed Computing, 1989:193 – 200.

2 Roberto Baldoni. Mutual Exclusion in Distributed Systems. PhD thesis, Universita di Roma "La Sapienza", 1994.

3 O. S. F. Carvalho and G. Roucairol. On mutual exclusion in computer networks. Communications of the ACM, February 1983 , 26(2):146 – 147.

4 Sukumar Ghosh. Binary self – stabilization in distributed systems. Information Processing Letters, November 1991, 40(3):153 ~ 159.

5 Kenichi Hagihara. Algorithms for fault – tolerant distributed systems. Journal of Information Processing Society of Japan, November 1993, 34(11):1336 – 1340.

6 Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self stabilizing mutual exclusion. In Proceedings of the 9th ACM Symposium on Principles of Distributed Computing, ACM, 1990: 119 – 131.

7 Yoshifumi Manabe and Shigemi Aoyagi. A distributed k – mutual exclusion algorithm using k – coterie. IEICE Japan, SIG Computation Record, May 1993, COMP91 – 13:11 – 18.

8 Glenn Ricart and Ashok K. Agrawala. An optimal algorithm for mutual exclusion in computer network. Communications of the ACM, January 1981 24(1):9 – 17.

9 Mukesh Singhal. A class of deadlock – free maekawa – type algorithms for mutual exclusion in distributed systems. Distributed Computing, April 1991:131 – 138.

10 Ichiro Suzuki and Tadao Kasami. A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems, November 1985,3(4):344 – 349.

11 尹俊文，邹鹏，王广芳，徐长梅. 分布式操作系统. 国防科技大学出版社，2004 年 1 月.