姓名： 杨润琦
学号： 121180144

1 问题与动机

本次作业我选择了在 [1]The black-white bakery algorithm and related bounded-space, adaptive, local-spinning and FIFO algorithms 这篇文章的基础上讨论直接基于消息传递模型的 Black-White Bakery Algorithm。Lamport 的 Bakery Algorithm 是共享存储互斥访问最经典的算法，然而其最原始的版本无法满足所需的公共存储空间有界，于是在最经典的算法版本之上有了 Black-White Bakery Algorithm，在引入票的颜色之后可以保证所需存储空间有界。并且 [1] 中还进一步讨论了 Adaptive and Local-spinning Black-White Bakery Algorithm，Adaptive 使得算法复杂度只和此时想进入临界区和正在临界区内的进程有关，和总的进程数无关；Local-spinning 则引入了 remote access 和 local access 的区别，在循环检查某个变量是否满足条件时只需要 local access，并且由其他进程很少的 remote write 来通知其他正在 local spinning 的进程。这篇文章是基于共享存储模型来讨论的，所用的语境也是进程访问内存临界区的互斥，然而把眼光放在分布式共享存储的互斥访问的时候，Adaptive 和 Local-spinning 这两个概念就显得十分不自然——很显然分布式的实现不可能出现一个循环反复请求远程变量的值再检查其是否满足条件，而且原文中基于共享存储来表述的算法涉及到大量的读写集合、区分全局和内部变量等等，显得十分冗杂。在分布式消息传递的语境下，很多问题讨论起来反而比共享存储的讨论要简单，比如上述两个特性使用事件驱动的设计模式可以完全避免反复检查某些变量值是否满足条件的循环，实现纯异步非阻塞的结构，并且基于消息传递模型来讨论也拉近了算法和实现之间的距离。因此在读了 [1] 之后，我打算基于消息传递模型重写其中的算法，满足 bounded-space, adaptive, local-spinning and FIFO 四个属性，同时也做了简单的实现来验证自己改写过后的算法的正确性。

2 算法与分析

首先简要回顾一下传统 Black White Bakery Algorithm 的过程。所有想要进入临界区的机器首先把自己的票设置成“流行色”，并且通过某种机制和同色的机器协商，从小到大依次选号。票的颜色不同于“流行色”的机器优先，如果颜色相同，则数字小的优先，如果颜色和数字都相同，则比较机器的 ID。访问完临界区的机器把流行色改成自己颜色的反色，确保自己这一批是优先的而新加入的机器优先级更低。通过以上分析可以看出如果有 n 台机器，票号的范围只会在 $0 \sim n$ 之间。

改写过后的算法分为三块，如果产生了访问临界区的需求，那么首先进入算法一；由于这些是非阻塞的异步算法，因此在一定的条件满足之后才会触发算法二和算法三（详见下页的算法伪码）。

本地变量有三个：myColor、myNum 和 doorwayFlag。doorwayFlag 负责记录目前这台机器目前是否想要访问临界区；myColor 和 myNum 则是算法中的票的颜色及票号。需要共同维护的全局变量除了临界区的变量以外只有一个 fashionColor。

算法一为一个机器宣告自己想要进入临界区、准备选号的过程。其流程非常简单，在设置完 doorwayFlag 和 myColor 之后向其他的机器广播自己的颜色，如果受到消息的机器发现自己的颜色和发送者不同，说明自己和发送者的选号无关，直接回复 0；否则回复自己的号数。集齐了所有回复即

Algorithm 1

initially fashionColor := 0 myNum := 0 doorwayFlag := false

1: doorwayFlag := true;
2: myColor := fashionColor;
3: BROADCAST: <1><myColor>

/** rules for handling broadcast I **/

1: REPLY myColor=MSG.myColor?myNum:0

If all peers have replied⇒Enter Algorithm 2

Algorithm 2

1: myNum := max{all replyMSG.myNum}+1
2: **for** peer in competing peers **do**
3: **if** (myColor = peer.color AND myNum >= peer.number) OR (peer.color != fashionColor AND myColor = fashionColor) **then**
4: SEND <key> → peer
5: remove peer from competing peers
6: **end if**
7: **end for**
8: BROADCAST: <2><myColor><myNum>

/** rules for handling broadcast II **/

1: **if** (NOT doorwayFlag) OR (fashionColor = myColor AND myColor != MSG.myColor) OR (myColor = MSG.myColor AND (myNum,myID) > (MSG.myNum,MSG.SENDER.ID)) **then**
2: REPLY <key>
3: **else**
4: add MSG.SENDER to competing peers
5: **end if**

If all <key>s are received⇒Enter Algorithm 3

Algorithm 3

1: visit CRITICAL SECTION
2: BROADCAST <new value of variables in CRITICAL SECTION>
3: fashionColor := (myColor=1?0:1)
4: BROADCAST <fashionColor>
5: SEND <key> → competing peers
6: clear competing peers
7: myNum := 0
8: doorwayFlag := false

表示可以选号，利用事件驱动机制触发算法二。必须所有机器都回复的原因是该算法属于异步算法，没有收到回复可能是消息尚未传达，不能提供任何有效信息，所以不能利用同步算法的假设来简化通信复杂度。

算法二乍看起来有些复杂，但其核心思想十分简单——即选好自己的号之后广播第二轮消息，提供自己的号和颜色，向其他所有人询问他们是否排在自己前面，如果对方发现自己不在排队或确定已经排在后面，就立刻回复“key”(表示确认的报文)；如果对方排在前面，则在访问完临界区后回复“key”，拿到所有的“key”即表示得到了所有其他人的确认，确认自己已经是目前最优先的了，因此可以进入临界区。然而有一些中间状态使得收到消息的机器暂时无法确定自己是否排在对方前面，因此引入一个叫做 competing peers 的集合，把待发送确认消息的机器都加入该集合。

理清基本思路以后就可以详细分析算法二的每一步了。算法二首先选好自己的号数，接下来检查 competing peers，如果确认对方优先则向其发送“key”并将对方移出集合。由于在 competing peers 中的机器肯定已经选好了自己的号了才会发送第二轮消息被选进该集合，因此立刻可以比较得出先后顺序。算法二的第三行即是判断方法的详细表述：如果发现对方颜色优先，可以确认对方优先；如果颜色相同，自己的号大于等于对方的号即可判断对方优先。这里可以取等号是因为如果对方出现在自己的 competing peers 集合里，说明对方在时间上是更先结束算法二的（算法二的最后才会广播第二轮消息)，因而理应更优先，并且由于本机的第二轮广播尚未发送，同时对方早已结束第三行的判定代码，因此不会出现两个机器在这一阶段同时发送“key”的情况。如果不取等号，就很可能因为相互等待确认而发生死锁。

检查完以后即可发送第二轮广播并且等待所有的“key”。拿到所有的“key”即进入算法三。

算法三一开始先处理临界区操作。处理完以后即同步临界区、更改并同步 fashionColor、向 competing peers 中的机器发送确认报文、清空 competing peers、清零 myNum 以及 doorwayFlag。从这里可以发现算法的正确性还有一个前提，即消息的延迟虽然可以任意长，然而却不能接受消息的乱序，即在本机上先发送的消息对方一定要先收到。如果 competing peers 中的机器先收到“key”再收到同步临界区的消息，那么其中某个收到“key”的机器可能已经进入临界区，基于同步前的值做了更改并且广播了它自己的更改，这样就会引起错误。

3 实验与评价

由于以上算法基于消息传递模型来讨论，因而可以比较直接地实现。由于实验中产生了大量冗杂的 log，因此为了可读性就不附在文章里了。实验中人为加大了各个环节的延迟（尤其是在临界区中的延迟)，因而能更方便地验证算法的正确性。在实验环境下该算法可以保证临界区访问的 safety、no starvation 和 FIFO。在很多程度上验证了算法的正确性。

该算法的优点在于没有一个循环检查某些变量值是否满足条件的循环，因而不存在内部的阻塞，并且采用异步触发，可以使机器内部的运行效率大大提高。从消息复杂度来看，一次对临界区的访问总共产生了两轮消息通信，加上访问结束后同步 fashionColor，总共涉及 $5(n-1)$ 条消息，这一不俗的表现源于算法极大地简化了 Black White Bakery Algorithm 中所有在异步通信中不必要的流程，只进行必要的消息通信。并且该算法属于纯异步的分布式算法，没有任何对同步时间的假设，消息延迟可以任意长，不会影响结果的正确性。

算法的缺点也十分明显，即容错性不佳。如果出现某台机器故障或消息丢失，可能导致整个系统

进入无限的等待。并且如前所述，消息传递中的序错乱也会导致系统错误。实际运行中可能需要进一步设计容错检测机制来应对复杂的可能存在各种错误的实际情况。

4 总结

最终做出的算法比一开始的预想要复杂得多。在将许多理想化的假设翻译成实际情况的过程中需要应对许许多多的问题。通过这一次实践性的设计也可以看出理论和实践之间的差距。在这次设计中，笔者也尝试过深入讨论容错性和故障处理的问题，然而因为过于复杂而作罢，只好先在不存在错误的情况下先迈出一步。在这样的过程中也感受到分布式系统最基本的共享存储一致性的要求也需要大量的理论和精巧的实践设计才能良好地实现并在实际中投入使用。水滴石穿非一日之功，在此也向所有分布式研究者们致敬。

参考文献

[1] Taubenfeld G. The black-white bakery algorithm and related bounded-space, adaptive, local-spinning and FIFO algorithms[M]//Distributed Computing. Springer Berlin Heidelberg, 2004: 56-70.