计算机系统结构

Lab1 Cache替换策略设计与分析 实验报告

计63 董硕华 2016011295

一. 实验目的

1. 理解学习LRU及其他已经提出的Cache替换策略
2. 在提供的模拟器上实现自己设计的Cache替换策略
3. 通过benchmark测试比较不同的Cache替换策略
4. 在实验报告中简要说明不同Cache替换策略的核心思想和算法
5. 在实验报告中说明是怎样对不同的Cache替换策略进行测试的
6. 在实验报告中分析不同替换策略下，程序的运行时间、Cache命中率受到的影响

二. 不同Cache替换策略的核心思想和算法

1. LRU

把CPU近期最少使用的块替换出去。这种替换算法需要随时记录Cache中各块的使用情况，以便确定哪个块是近期最少使用的块。每块设置一个计数器，Cache每命中一次，命中块计数器清零，其他各计数器增加1。当需要替换时，将计数值最大的块换出

2. LFU

将一段时间内被访问次数最少的那个块替换出去。每块设置一个计数器，从0开始计数，每访问一次，被访问块的计数器就加1。当需要替换时，将计数值最小的块换出，同时将所有块的计数器清零

3. 随机替换

最简单的替换算法是随机替换。随机替换算法完全不管Cache的情况，简单地根据一个随机数选择一块替换出去。随机替换算法在硬件上容易实现，且速度也比前两种算法快。缺点则是降低了命中率和Cache工作效率

4. FIFO

先进先出。在FIFO Cache设计中，核心原则就是：如果一个数据最先进入缓存中，则应该最早淘汰掉。也就是说，当缓存满的时候，应当把最先进入缓存的数据给淘汰掉

5. RRIP

核心思想为将访问间隔较小的块留在cache中，从而提高命中率。维护Mbits记录PPRV，选择PPRV值为2的M次方减一的项替换出去，如果没有该项，则每项PPRV值+1，之后重复扫描。新进PPRV置为2的m次方-2，命中块清零

6. 我使用的Cache替换策略 Clock

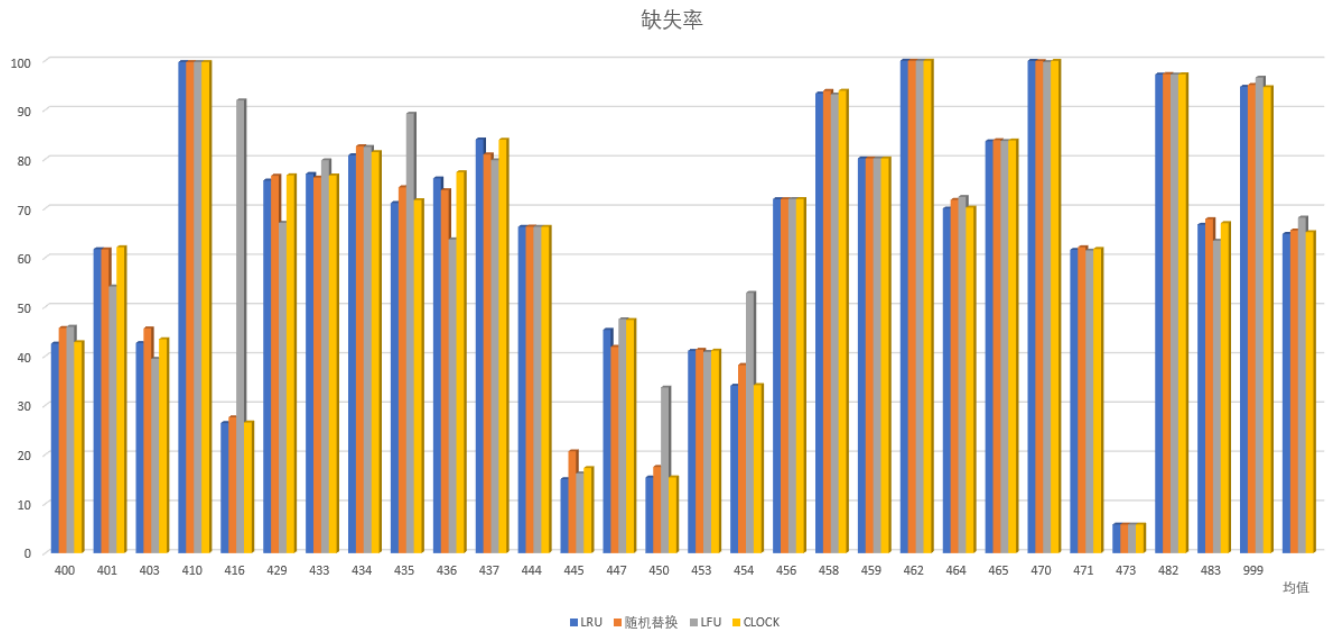
时钟算法是FIFO与LRU两种算法的综合。FIFO不考虑过去，而LRU又考虑了太多，于是综合考虑两种方法，取其中间，得到时钟算法

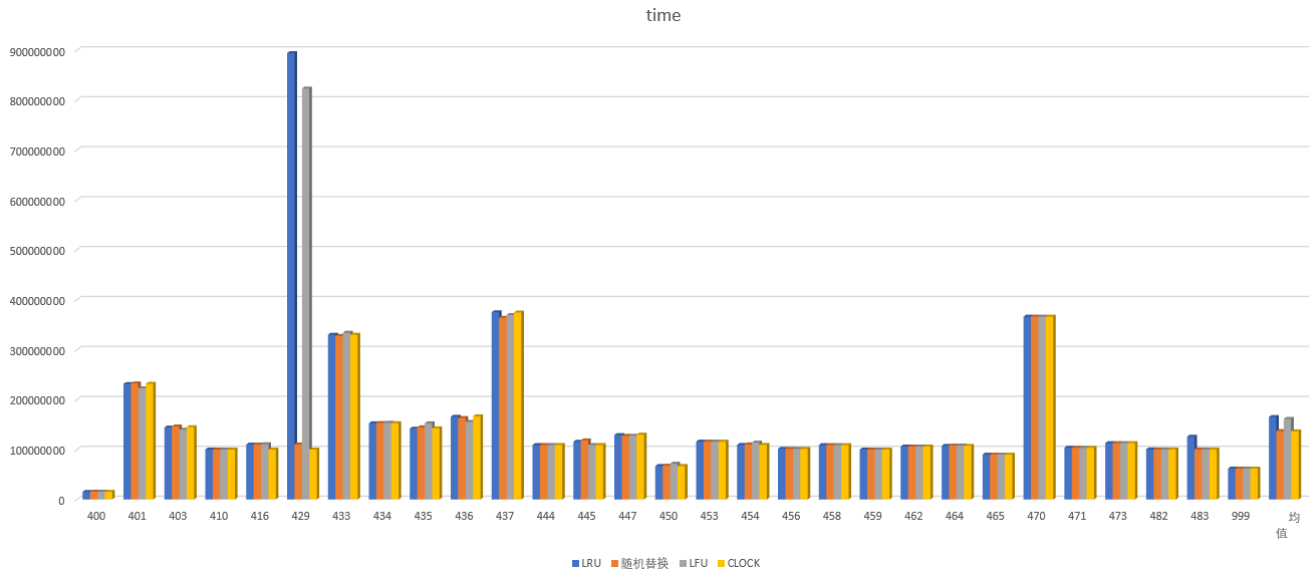
实现思路：定义一个环形页表，指针指向初始位置。命中时，将其中的访问位置1；缺失时，从指针位置开始进行判断，若当前位置访问位为1，将其置为0，指针往下走一位，直到遇到一个访问位为0的，进行替换，并将指针指向下一位

三. 测试结果

利用Python脚本实现性能测试，分别测试了LRU，随机替换算法，LFU算法，CLOCK算法

测试结果见Excel文件





可以看出：LRU优于CLOCK优于随机替换优于LFU

四. 结果分析

LRU

LRU算法实现起来比较复杂，系统开销较大。通常需要对每一块设置一个称为计数器的硬件或软件模块，用以记录其被使用的情况。这种算法保护了刚调入Cache的新数据块，故具有较高的命中率。由于需要维护将元素调往栈顶，其余元素一次往下的过程，所以会使得程序的运行时间变长

随机替换

没有任何参考，随机替换，时间影响不大，运行时间会比较短，速度比较快

LFU

根据数据的历史访问频率来淘汰数据，其核心思想是"如果数据过去被访问多次，那么将来被访问的频率也更高。需要维护一个队列记录所有数据的访问记录，每个数据都需要维护引用计数。需要记录所有数据的访问记录，内存消耗较高；需要基于引用计数排序，性能消耗较高

CLOCK

是LRU与FIFO算法的综合，命中率有所提升，但是不会高于LRU，但是速度会比LRU比起来快

五. 实验小结

在本次实验中，我能够在实际操作中了解各种Cache替换策略的实现，并且能够较为直观的感受到他们对程序运行的影响。并且，在阅读已存在的实现后，我可以依照样例自己动手实现Cache替换策略，尽管环境的配置可能花费了大量的时间且出现了各种问题，但是在助教和同学的帮助下，也终于是能够顺利的自己实现自己的方法。

由于五一期间没能按时回到学校，所以没能按时完成实验，在此向助教表示抱歉，感谢老师和助教一直以来的帮助，让我能对系统结构有一个全面而系统的了解。