

**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 姓名： | 尹卯 |
| 学院： | 计算机科学与技术 |
| 专业： | 计算机科学与技术 |
| 班级： | 计算机1706班 |
| 学号： | U201714770 |
| 指导教师： | 万继光 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2020 年. 4月. 28日

**目 录**

[1. Cache模拟器实验 2](#_Toc23963)

[1.1. 实验目的 2](#_Toc3705)

[1.2. 实验环境 2](#_Toc865)

[1.3. 实验思路 2](#_Toc25045)

[1.4. 实验结果和分析 4](#_Toc21474)

[2. 总结和体会 6](#_Toc18091)

[3. 对实验课程的建议 6](#_Toc32753)

# Cache模拟器实验

* 1. **实验目的**

1. 理解cache的工作原理
2. 理解LRU算法的工作原理
3. 如何实现一个高效的cache模拟器
   1. **实验环境**
4. Linux 64-bit
5. C语言
   1. **实验思路**

#### 1.3.1 实验内容

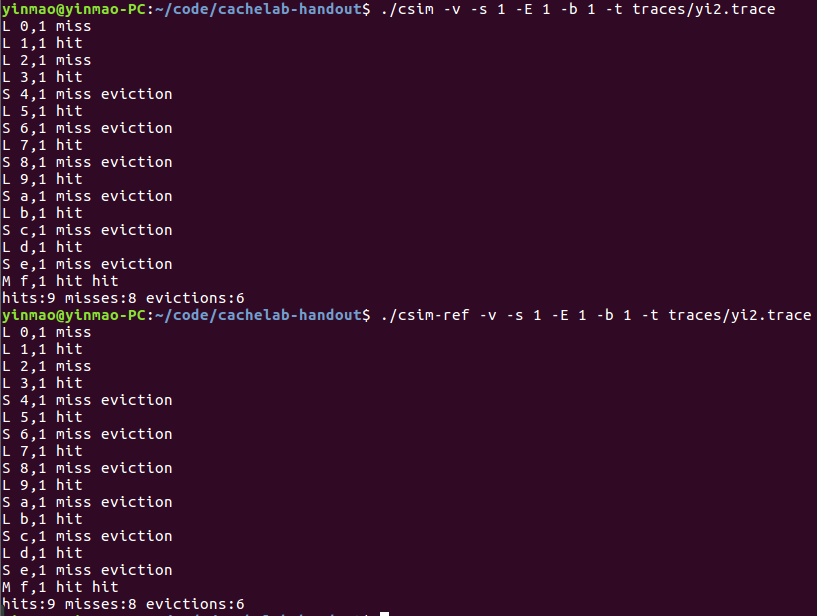
1. 在csim.c提供的程序框架中，编写实现一个cache模拟器
2. 输入：内存访问轨迹文件trace
3. 操作：模拟缓存想对内存访问轨迹的命中缺失行为
4. 输出：命中、缺失和缓存行淘汰的总数
5. 完成的csim.c文件应能接受与参考缓存模拟器csim-ref相同的命令行参数并产生一致的输出结果。
6. 命令行格式：csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
7. 模拟器必须在输入参数s、E、b设置位任意值时均能正确工作，所以需要使用malloc函数（而不是代码中固定大小的值）来为模拟器中数据分配存储空间
8. 由于实验仅关心数据cache的性能，因此模拟器应忽略所有指令的cache访问（即轨迹文件中“i”起始的行）
9. 假设内存访问的地址总是正确对齐的即一次内存访问从不跨越内存块的边界，因此可忽略访问轨迹中给出的访问请求大小
10. main函数必须调用printSummary函数输出结果，并显示命中hit、缺失miss和淘汰eviction的总数作为参数

#### 1.3.2 实验数据结构设计

1. CacheLine：Cache行的结构
   1. valid：有效位
   2. tag：标记位
   3. lru：LRU算法的计数器
2. CacheGroup：Cache组的结构，\*line指向Cache组中的多个Cache行
3. Cache：整个Cache的空间结构，\*groups指向Cache中的多个Cache组
4. 用来计数的变量：
   1. miss\_count：缺失次数
   2. hit\_count：命中次数
   3. eviction\_count：替换次数
   4. lru\_count：为1，表示lru计数器的初始值
5. 组索引和标记位：indexMark，tagMark

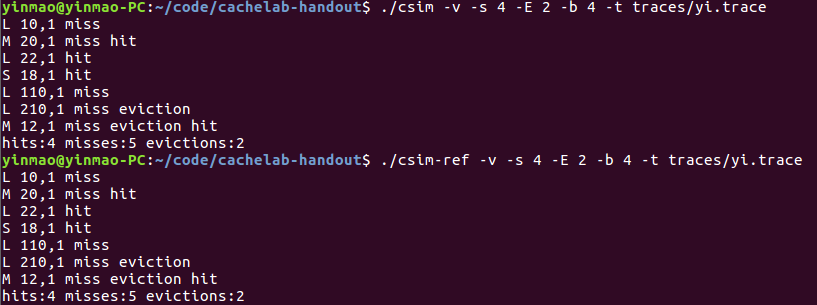
#### 1.3.3 实验函数功能设计

1. initCache：初始化Cache，为每行Cache分配内存，将每行Cache的有效位、标记位和lru计数器设置为0
2. visit(addr)：根据所给的operation和地址对Cache进行访问
   1. 如果Cache中存在该地址所对应的数据，命中数hit\_count加一。
   2. 如果Cache中不存在该地址所应的数据，缺失数miss\_count加一。
   3. 判断该地址所在组的Cache行是否已满，如果Cache行未满，找到一个空闲的Cache行，根据该行的组索引位将该行的有效位valid设置为1，标记位设置值为tagMark，lru设置为初始值1，其他Cache行的lru计数器减1。
   4. 如果该地址所在组的Cache行已满，找到改Cache组中lru最小的Cache行，将该Cache行进行替换，将该行的有效位valid设置为1，标记位设置值为tagMark，lru设置为初始值1，其他Cache行的lru计数器减1。
3. traceFile(tracePaht)：读取trace轨迹文件中的信息，根据信息中的operation和地址来访问Cache
   1. 使用fopen打开文件
   2. 一次读取每一行，获取每一行中的operation、address和size等信息，如果该行中的operation为S、L或M，则调用visit()函数，将address作为参数传递给visit函数，如果operation为M，则还要调用一次visit函数。
   3. 一次对读取到的每一行trace轨迹信息都访问Cache，直到读完所有的trace信息。
4. freeCache：释放Cache空间，依次释放Cache行空间，然后释放Cache组空间。
5. main函数：解析用户输入的命令行，并根据s、E、b提取出Cache的组索引位数、组内Cache行数、块地址位数、trace文件等信息，然后调用initCache初始化Cache行，调用traceFile读取trace文件进行测试，输出访问结果，释放Cache内存空间。
   1. **实验结果和分析**
6. 测试用例./csim -v -s 1 -E 1 -b 1 -t traces/yi2.trace：



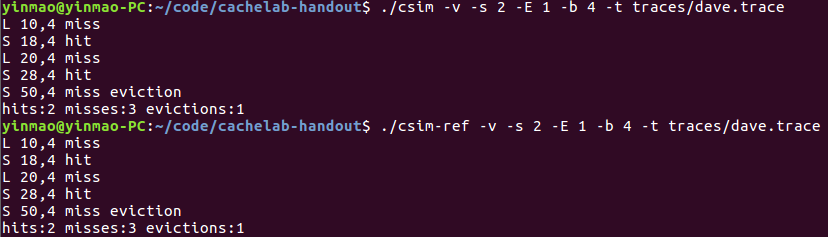
**图1.4.1 测试用例1**

1. 测试用例./csim -v -s 4 -E 2 -b 4 -t traces/yi.trace：



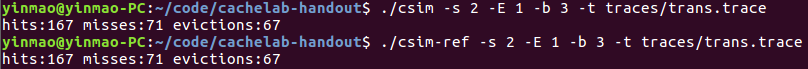
**图1.4.2 测试用例2**

1. 测试用例./csim -v -s 2 -E 1 -b 4 -t traces/dave.trace：



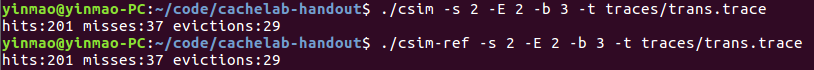
**图1.4.3 测试用例3**

1. 测试用例./csim -s 2 -E 1 -b 3 -t traces/trans.trace：



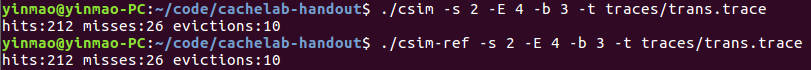
**图1.4.4 测试用例4**

1. 测试用例./csim -s 2 -E 2 -b 3 -t traces/trans.trace：



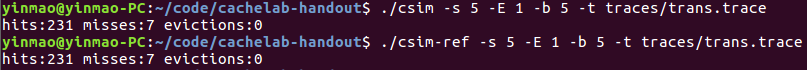
**图1.4.5 测试用例5**

1. 测试用例./csim -s 2 -E 4 -b 3 -t traces/trans.trace：



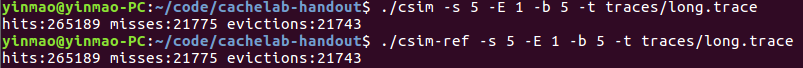
**图1.4.6 测试用例6**

1. 测试用例./csim -v -s 5 -E 1 -b 5 -t traces/trans.trace：



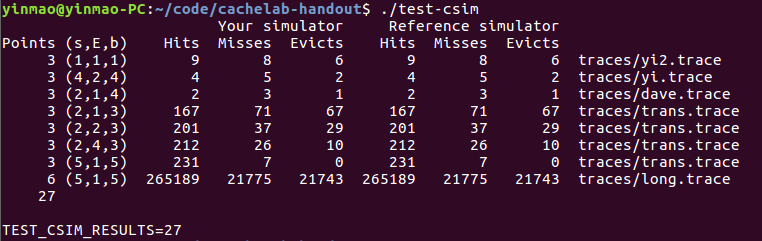
**图1.4.7 测试用例7**

1. 测试用例./csim -v -s 5 -E 1 -b 5 -t traces/long.trace：



**图1.4.8 测试用例8**

1. 使用test-csim测试程序进行测试：



**图1.4.9 test-csim测试**

# 总结和体会

1. 此次实验加深了对Cache的基本概念和工作原理的理解，特别是Cache的映像规则及其变换。
2. 通过改变Cache模拟器中s、E和b的大小，认识到了Cache容量、关联度、块大小对Cache性能的影响。
3. 掌握了降低Cache不命中率的一些方法，以及这些方法给Cache带来的优缺点。比如适当增加块大小可降低不命中率。
4. 深刻理解了LRU算法的基本思想和实现方法，以及LRU算法的优缺点。

# 对实验课程的建议

1. 可以要求学生在实现Cache模拟器时除了使用LRU替换策略外，再实现其他的替换策略，并对比每种替换策略带来的优缺点。
2. 可以再增加一个简单的互联网络相关的实验，让学生对互联网络的理解更深刻。