

**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 姓 名： | 王铭璇 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1705 |
| 学 号： | U201714742 |
| 指导教师： | 万继光 |

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

2020 年 4 月 20 日

**目 录**

[1. Cache模拟器实验 4](#_Toc23963)

[1.1. 实验目的 4](#_Toc3705)

[1.2. 实验环境 4](#_Toc865)

[1.3. 实验思路 4](#_Toc25045)

[1.4. 实验结果和分析 6](#_Toc21474)

[2. 总结和体会 8](#_Toc18091)

[3. 对实验课程的建议 8](#_Toc32753)

# Cache模拟器实验

* 1. **实验目的**

1. 理解Cache工作原理；
2. 实现一个高效的模拟器。
   1. **实验环境**

编码环境：MacOS Catalina 10.15.1 XCode 11.3.1

编译器：gcc version 5.4.0

运行环境：Linux version 4.15.0-88-generic

* 1. **实验思路**

1. Cache模拟数据结构：

由于采用组相联的Cache组织方式，结合LRU淘汰算法，Cache的每个组都需维护一个链表，链表的一个结点存储Cache的一个块，用顺序线性表的一个元素表示Cache的组，作为链表头节点指向对应的组链表，通过指针数组的下标可以快速定位到访问的数据应存放的组链表。块结构定义为：

typedef struct Block{

int tag;

int valid;

Block\* nextBlock;

}CacheRow;

其中tag是主存块的标识，valid标记存储数据是否有效，为0时该块为空块，Cache组链表的存储规则定为：当有多个空结点即一个组内有多个空块时，优先使用最靠近链表尾的空结点；当一个组存储块全部被占用时，淘汰链表尾的块；

1. 访问主存地址到Cache地址的转换：

主存地址通过读取trace文件获得，主存地址可以由高位到低位依次分割为标识位tag，组号index，和偏移量offset。由二进制数的按位与和左移右移操作可以获取存取数据在Cache中对应存储的组号index，再根据tag与Cache块结构中存放的tag标记对比得知要访问的数据块是否在该组中；

1. Cache的更新函数UpdateCache：

首先由计算获得的内存块对应Cache组号定位到该组链表的头指针，遍历该链表对比Cache块内存储的有效tag和访问的内存块tag是否相等，如找到了该块，则说明Cache命中，hit总次数加一，并更新链表，将命中块的链表结点置于链表第一个空结点后；

如果遍历指针指向链表尾仍未找到该数据块，说明数据块缺失，miss总次数加一；便利过程中如每发现块的valid标记位为0，则用一个空块指针LastEmptyBlock指向该块，则遍历结束时该指针指向的是链表内最靠近链表尾的一个空块。

如链表内存在空块，令该块的tag值为访问数据块的tag值，表示从主存移入该数据块；如链表存储已满，说明需淘汰链表尾的数据块移入新块，eviction总次数加一，链表尾的块tag值修改为本次访问数据块的tag值，并将其移至链表头；

1. Cache模拟器整体工作流程：

解析命令行输入获得模拟Cache的组数，组内块数，和内存块的大小，从而调用CreateCache(CacheRow \*Cache[])函数创建模拟Cache数据结构；

调用ReadTestFile()函数，读取trace文件，获得访问模式和访问的内存地址，计算得到对应的Cache组号及数据块标记Tag；

调用UpdateCache函数，查找Cache对应组内数据块，并更新链表状态；

调用printSummary函数，输出hit，miss和eviction的统计总数。

* 1. **实验结果和分析**

1. 编译csim文件，生成可执行文件csim.o，如图1.所示：



（a）



（b）

图1.1

1. 命令行输入“./csim -h -s 2 -E 2 -b 3 -t traces/trans.trace”，输出帮助信息，如图1.所示：

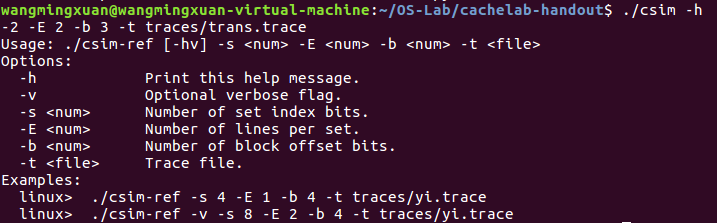


图1.2

1. 命令行输入“./csim -v -s 2 -E 2 -b 3 -t traces/yi.trace”，输出轨迹信息，如图1.所示：

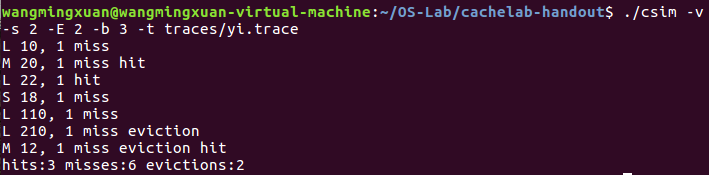


图1.3

1. 命令行输入“./csim -s 2 -E 2 -b 3 -t traces/trans.trace”，直接输出统计结果，如图1.所示：



图1.4

1. 运行测试程序test-csim，对比两个simulator输出结果，输出一致，如图1.所示，Cache模拟器程序正确。

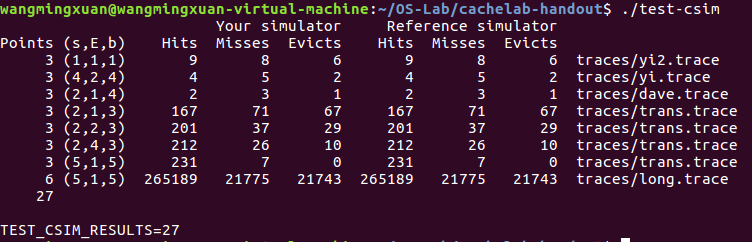


图1.5

# 总结和体会

本次实验通过实现一个简化的Cache模拟器，复习了Cache的组相联组织方式和主存地址间的映射关系，加深了我对Cache和主存映像的理解；通过实现LRU淘汰算法，对LRU机制的理解和链表数据结构的特点的体会都得到了提高，实验开始时想过直接用一个大小为组数\*组内块数的数组实现模拟Cache，但是后来考虑到如果用数组表示，则涉及LRU命中及淘汰的块淘汰优先级变化产生移动时，需要对整个组对应的数组存储内容都进行移动，一方面数组类型在数据移动时时间效率低，另一方面这种数据的移动也无法体现Cache存储的特点，反而是为了实现LRU而改变了Cache的存储原理。因此最终采用了临接链表结构，既可以快速地定位到访问主存块对应的Cache组，又可以方便地实现LRU需要的块的移动。最终实现了符合实验要求的Cache模拟器。

# 对实验课程的建议

实验内容比较少，如果增加课时的话或许可以学习更多内容加深对计算机系统结构的理解。