

**计算机系统结构实验报告**

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

姓名：

班级：

计算机科学与技术学院

2020 年 4月 21 日

**目 录**

[1. Cache模拟器实验 3](#_Toc38388508)

[**1.1.** **实验目的** 3](#_Toc38388509)

[**1.2.** **实验环境** 3](#_Toc38388510)

[**1.3.** **实验思路** 3](#_Toc38388511)

[**1.4.** **实验结果和分析** 7](#_Toc38388512)

[2. 总结和体会 8](#_Toc38388513)

[3. 对实验课程的建议 8](#_Toc38388514)

# Cache模拟器实验

* 1. **实验目的**

理解cache的工作原理

用C语言实现cache模拟器，并统计hit、miss和eviction的次数

* 1. **实验环境**

Ubuntu18.04 - 64 bit , c语言

* 1. **实验思路**

1. 整体设计思路：

本实验模拟缓存的行为，内存的数据不用存储，不使用块偏移，因此b并不重要，简单的计算hits、miss、evictions的值。模拟器需要适用于不同的（s，b，E）。最后就是要求使用LRU最近最少使用策略进行行的覆盖，因此cache发生miss的时候，从下一层存储读入数据，假设在空间已满的情况下（未满的时候不需要替换行）,新读取的数据需要存储在cache里面，但空间已满，只能选择覆盖某一行，依据LRU策略进行。LRU策略就是：选择最后被访问的时间距离现在最远的块。具体实现是给cache的每一行分配一个Lrunumber，这个值越大表示它最后一次被访问的时间距离现在最远，可以作为覆盖行。对于L，M，S三种操作，M等同于两次访存，S等同于一次访存，记录所有命中，丢失以及行覆盖的次数输出即可。

2.具体实现：

（1）处理输入参数：

本程序是在linux下以附加命令参数的方式来执行的，所以要对命令行的参数进行处理。这里要用到getopt函数，函数原型为：

int getopt(int argc,char\*\* argv,const char\* opstring);

getopt函数的返回值即为当前调用它所读取到的那个参数（int对应其ASCII码值），其中的opstring是一个参数集合的字符串，形如：

const char\* optstring = "hvs:E:b:t:";

其中每一个字母后面如果加一个冒号代表其必须带有一个附加的参数，不带就是必须没有附加的参数；具体实现代码如下图：

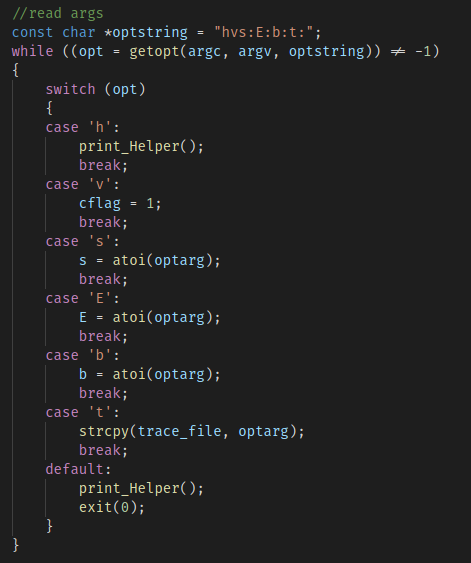


图1.1 read args

(2)初始化cache：

通过编程语言的形式来“模拟”出cache的简单模型，即S=2^s个set，每个set有E lines，每line有B=2^b个存储单元，加上tag标记位和valid有效位，具体结构如图所示

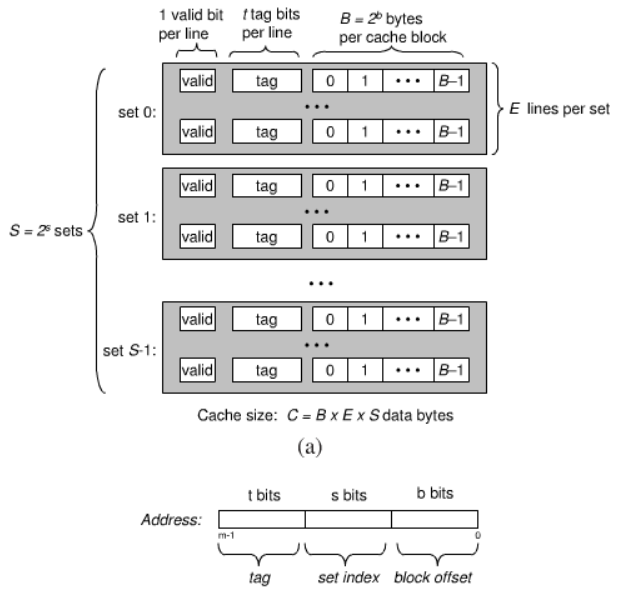


图1.2 Cache结构

依据其物理结构设计数据结构如下：

typedef struct

{

int valid; //有效位

int tag; //标识位

int lru\_number; //LRU算法依赖

} Line;

typedef struct

{

Line \*lines; //一组中所有的行

} Set;

typedef struct

{

int set\_num; //组数

int line\_num; //行数

Set \*sets; //cache

} Cache;

依据数据结构和参数进行相应的内存空间分配即可：

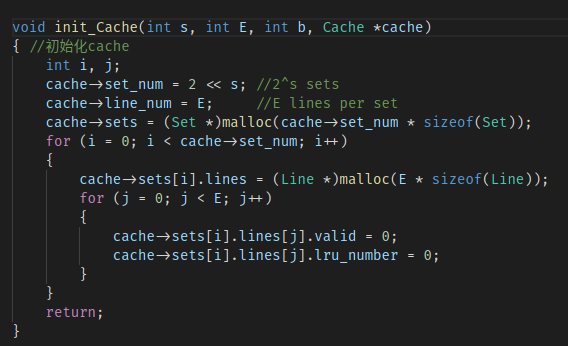


图1.3 Chche init

(3)内存访问：

首先操作分为I，L，M，S四部分，I默认忽略，M可等价为两次访存，S可等价为一次访存，下面主要分析L操作访存的过程：

对于给定的addr得到其set\_num和tag\_num，在cache对对应set中所有的line进行比较，寻找对应的tag\_num；如果找到且valid位为1则为hit，找不到或valid不为1但set未满则为miss，找不到或valid不为1但set已满则为miss && evict，依据LRU的规则进行相应的行覆盖即可。

计算set\_num和tag\_num:

set\_num = addr >> b & ((1 << s) - 1);

tag\_num = addr >> (s + b);

LOAD操作：遍历set\_num对应的set，依据结果对ismiss赋值

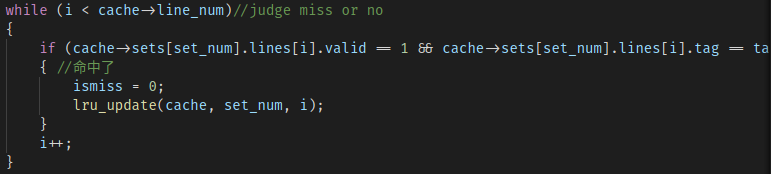


图1.4 Cache miss

当miss时，对相应的set进行插入行或覆盖行的操作：

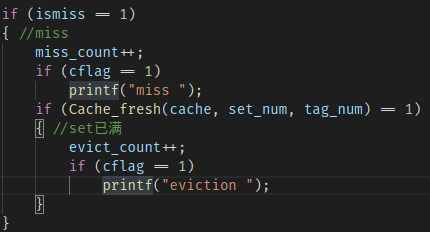


图1.5 cache miss处理

非miss时，hit\_count自增即可

(4)LRU实现：

每一行绑定一个Lrunum，初始化时置为0，每次方位set命中的行置为0，未命中的行加1，这个值越大表示它最后一次被访问的时间距离现在最远，当需要发生行替换时，选择lrunum最大的行作为覆盖行

* 1. **实验结果和分析**

标准程序执行结果与自己写的结果对比：

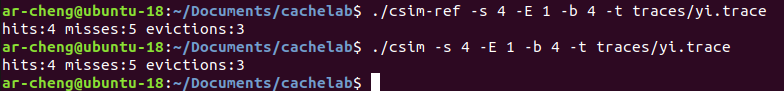


图1.6 yi.trace标准测试

结果基本一致

执行总的测试程序结果如下：

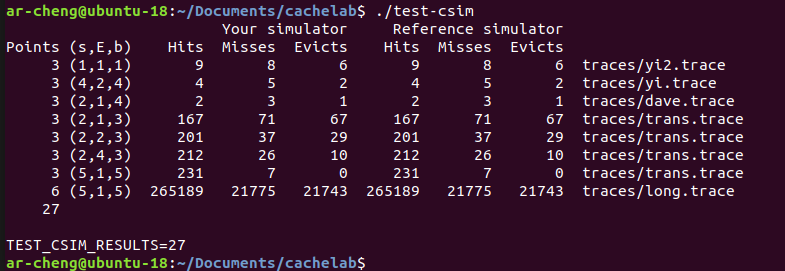


图1.7 总体测试结果

总体测试结果基本符合

在调试的过程中碰到了一个比较难解决的问题，各个函数经过单独测试没有明显问题，最后通过排查找到了问题在getopt 函数的使用上，const char string = "hvs:E:b:t:"，最初t后未加冒号，导致程序的运行与预期结果差距较大，花费了较多时间。

# 总结和体会

通过本次实验对于cache的工作原理有了更好的理解，在代码层次的实现更加深了对于cache工作过程的认识，是对课堂学习内容很好的实践。在实验过程中因为代码细节方面的问题花费了较多的时间，代码能力还需要提高。

# 对实验课程的建议

由于网上学习的限制，个人觉得对于相关内容的掌握效果不如学校课堂，实验很好的加强了这一点。但是感觉实验的内容较少，希望能够增加课时，增加实验的内容。