

**实验报告**

**实 验（六）**

题 目 Cachelab

高速缓冲器模拟

专 业 计算机大类

学　　 号 1191000606

班　　 级 1903003

学 生 陈一帆

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.5.28

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc72915484)

[1.1 实验目的 - 3 -](#_Toc72915485)

[1.2 实验环境与工具 - 3 -](#_Toc72915486)

[1.2.1 硬件环境](#_Toc72915487) **[错误！未定义书签。](#_Toc72915487)**

[1.2.2 软件环境](#_Toc72915488) **[错误！未定义书签。](#_Toc72915488)**

[1.2.3 开发工具](#_Toc72915489) **[错误！未定义书签。](#_Toc72915489)**

[1.3 实验预习 - 3 -](#_Toc72915490)

[第2章 实验预习 - 5 -](#_Toc72915491)

[2.1 画出存储器层级结构，标识容量价格速度等指标变化（5分） - 5 -](#_Toc72915492)

[2.2用CPUZ等查看你的计算机Ca](#_Toc72915493)[che各参数，写出各级Cache的C S E B s e b（5分）](#_Toc72915493) **[错误！未定义书签。](#_Toc72915493)**

[2.3写出各类Cache的读策略与写策略（5分） - 6 -](#_Toc72915494)

[2.4 写出用gprof进行性能分析的方法（5分） - 7 -](#_Toc72915495)

[2.5写出用Valgrind进行性能分析的方法（5分） - 8 -](#_Toc72915496)

[第3章 Cache模拟与测试 - 11 -](#_Toc72915497)

[3.1 Cache模拟器设计 - 11 -](#_Toc72915498)

[3.2 矩阵转置设计 - 17 -](#_Toc72915499)

[第4章 总结 - 24 -](#_Toc72915500)

[4.1 请总结本次实验的收获 - 24 -](#_Toc72915501)

[4.2 请给出对本次实验内容的建议 - 24 -](#_Toc72915502)

[参考文献 - 25 -](#_Toc72915503)

# 第1章 实验基本信息

## 1.1 实验目的

## 理解现代计算机系统存储器层级结构

## 掌握Cache的功能结构与访问控制策略

## 培养Linux下的性能测试方法与技巧

## 深入理解Cache组成结构对C程序性能的影响

## 1.2 实验环境与工具

### 1.2.1 硬件环境

X64CPU；1.6GHz；8G RAM；512G SSD

### 1.2.2 软件环境

Windows10 64位；Ubuntu 18.04.5 LTS 64位

### 1.2.3 开发工具

Clion2020.3；Vscode；gedit+gcc；

## 1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT或PDF）

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

画出存储器的层级结构，标识其容量价格速度等指标变化

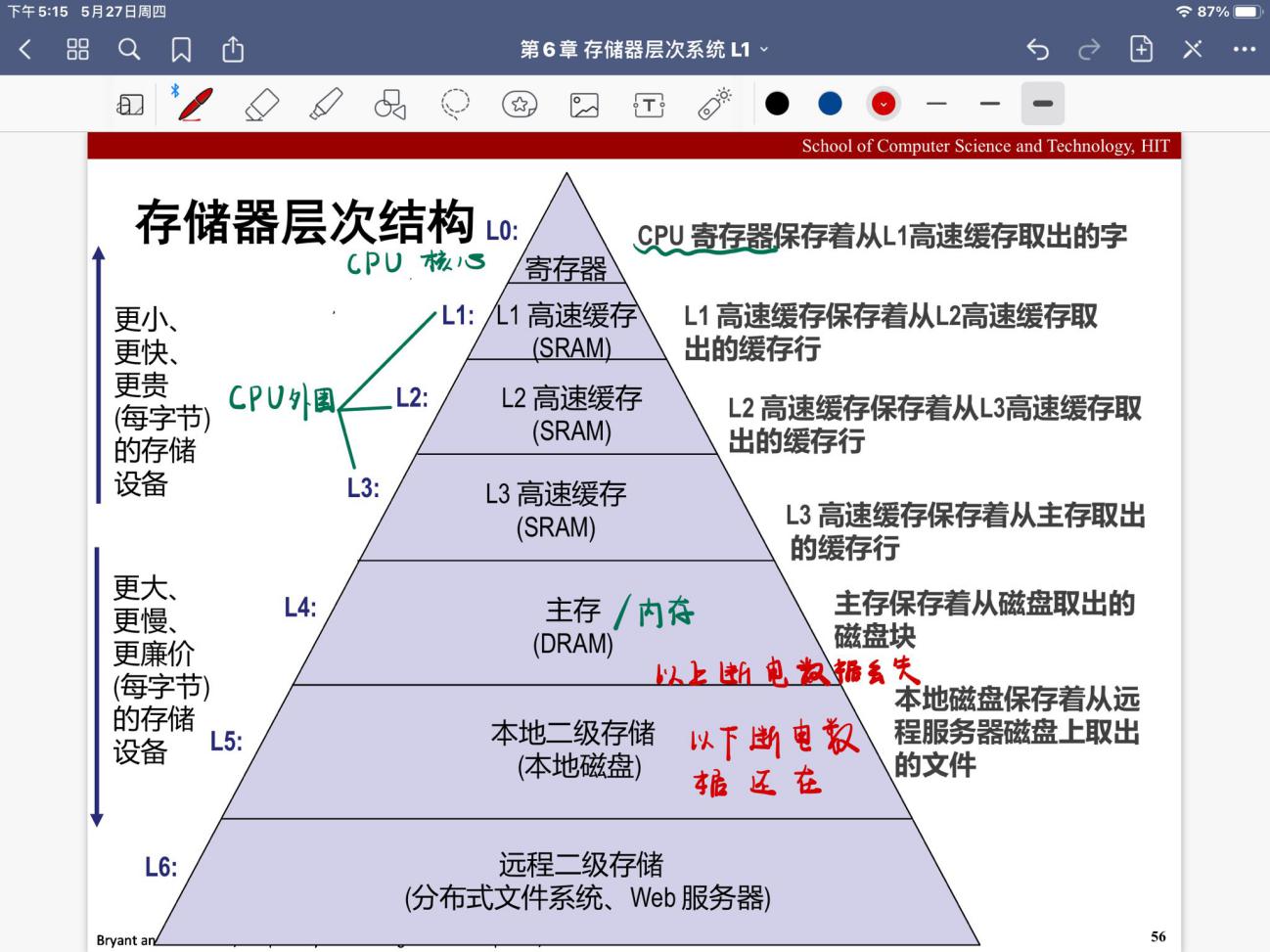
用CPUZ等查看你的计算机Cache各参数，写出Cache的基本结构与参数：缓存大小C、分组数量 S、关联度/组内行数 E、块大小 B，及对应的编码位数 ：组索引位数s、 e 、块内偏移位数b

写出Cache的各种读策略与写策略

掌握Valgrind、gprof的使用方法

# 第2章 实验预习

## 2.1 画出存储器层级结构，标识容量价格速度等指标变化（5分）



## 2.2计算机Cache的参数查看与分析（5分）

用CPUZ等查看你的计算机Cache的参数，写出各级Cache的C(大小)、 S(组数)、 E(路数)、 B(块大小)，并分析相应的s（组编码位数）、b（块内偏移地址位数）数值。

CPU-Z参数如下图，Windows下运行。



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C | S | E | B | s | b |
| L0数据缓存 | 128KB | 256 | 8 | 64 | 8 | 6 |
| L1指令缓存 | 128KB | 256 | 8 | 64 | 8 | 6 |
| L1 | 1024KB | 4096 | 4 | 64 | 12 | 6 |
| L2 | 6MB | 8096 | 12 | 64 | 13 | 6 |

## 2.3写出各类Cache的读策略与写策略（5分）

读策略

缓存命中，则从cache中读相应数据到CPU或上一级cache中

缓存不命中，则从主存或下一级cache中读取数据，并替换出一行数据

写策略

命中

直写：立即写入存储器

写回：推迟到缓存行要替换时才写回内存

不命中

写分配：加载低一层中的块到缓存

非写分配：直接写回到主存中，不加载到缓存

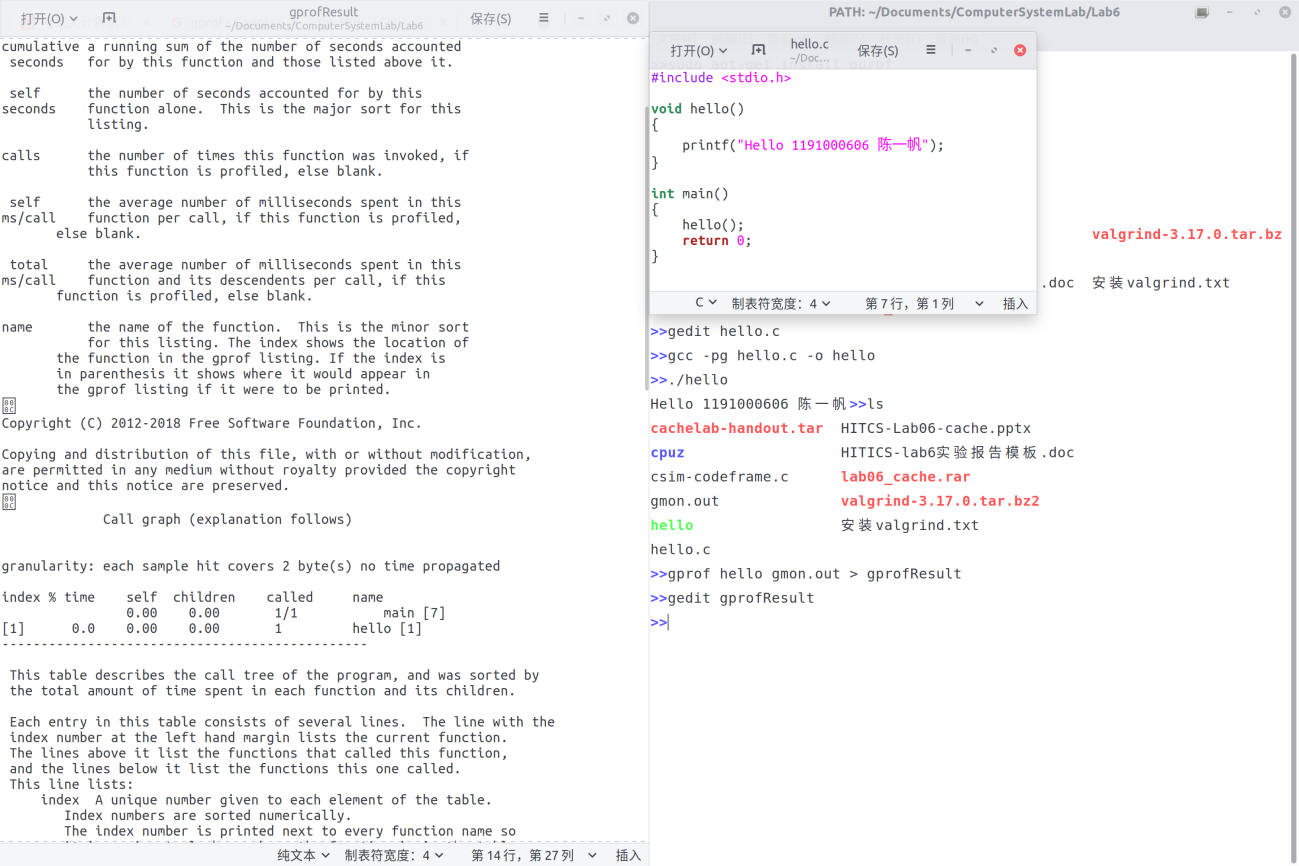
## 2.4 写出用gprof进行性能分析的方法（5分）

用gcc、g++、xlC编译程序时，使用-pg参数，如：gcc -pg hello.c -o hello。编译器会自动在目标代码中插入用于性能测试的代码片断，这些代码在程序运行时采集并记录函数的调用关系和调用次数，并记录函数自身执行时间和被调用函数的执行时间。

执行编译后的可执行程序，如：./hello。该步骤运行程序的时间会稍慢于正常编译的可执行程序的运行时间。程序运行结束后，会在程序所在路径下生成一个缺省文件名为gmon.out的文件，这个文件就是记录程序运行的性能、调用关系、调用次数等信息的数据文件。

使用gprof命令来分析记录程序运行信息的gmon.out文件，如：gprof hello gmon.out则可以在显示器上看到函数调用相关的统计、分析信息。上述信息也可以采用gprof hello gmon.out> gprofresult.txt重定向到文本文件以便于后续分析。

如下图。



## 2.5写出用Valgrind进行性能分析的方法（5分）

Valgrind是运行在Linux上一套基于仿真技术的程序调试和分析工具，它包含一个内核──一个软件合成的CPU，和一系列的小工具，每个工具都可以完成一项任务──调试，分析，或测试等。Valgrind可以检测内存泄漏和内存违例，还可以分析cache的使用等。

Valgrind包含以下工具：

1. Memcheck，用来检测程序中出现的内存问题，所有对内存的读写都会被检测到，一切对malloc()/free()/new/delete的调用都会被捕获。
2. Callgrind，收集程序运行时的一些数据，建立函数调用关系图，还可以有选择地进行cache模拟。在运行结束时，它会把分析数据写入一个文件，callgrind\_annotate可以把这个文件的内容转化成可读的形式。
3. Cachegrind，模拟CPU中的一级缓存I1，Dl和二级缓存，能够精确地指出程序中cache的丢失和命中。如果需要，它还能够为我们提供cache丢失次数，内存引用次数，以及每行代码，每个函数，每个模块，整个程序产生的指令数。
4. Helgrind，用来检查多线程程序中出现的竞争问题。
5. Massif，堆栈分析器，能测量程序在堆栈中使用了多少内存，告诉我们堆块，堆管理块和栈的大小。

Valgrind的使用非常简单，valgrind命令的格式如下：valgrind [valgrind-options] your-prog [your-prog options] 。一些常用选项如下：

-h --help 显示帮助信息

--version 显示valgrind内核和各个工具的版本

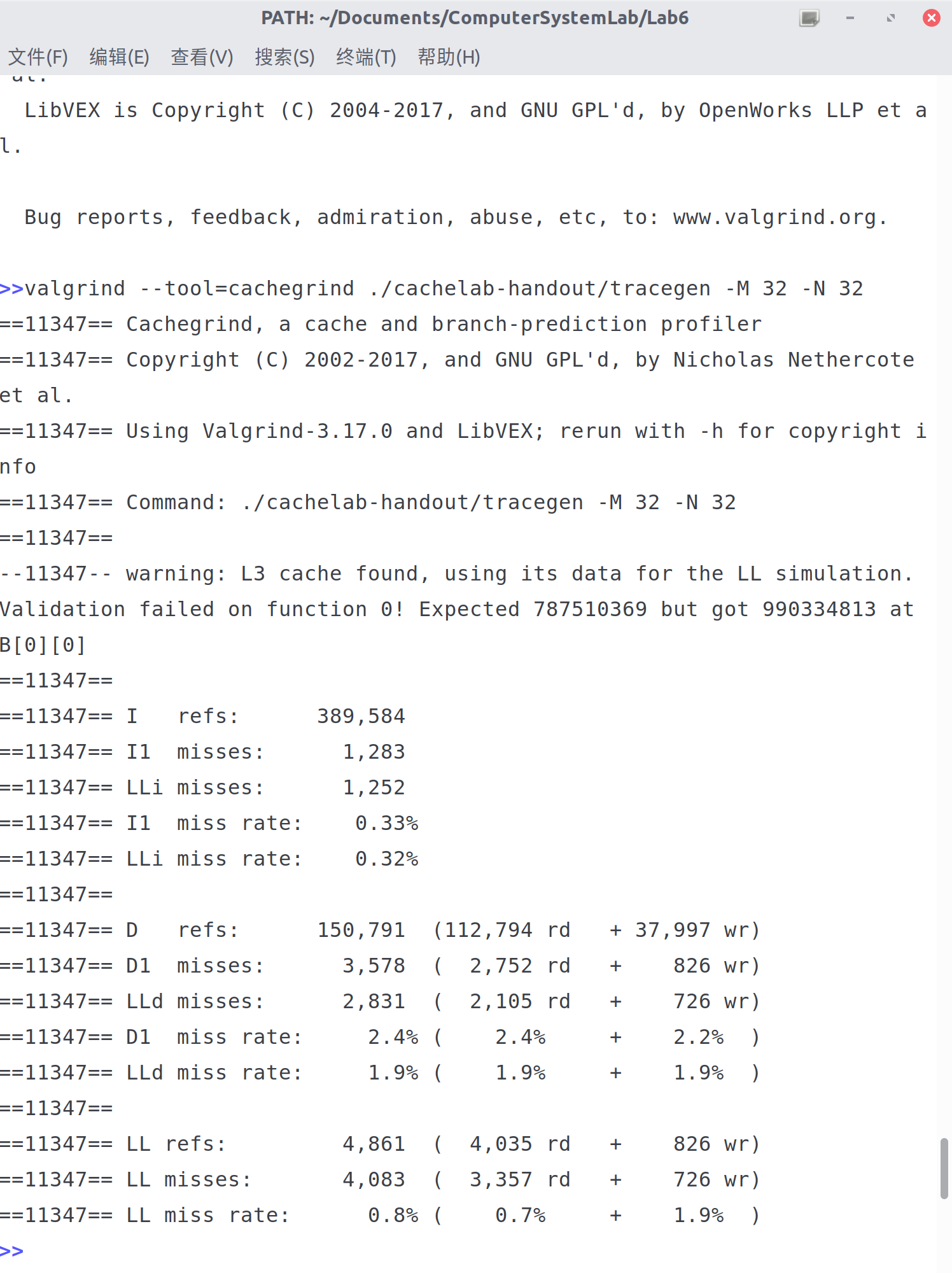
-q --quiet 安静的运行，只打印错误信息

-v --verbose 打印更详细的信息

--tool=[default:memcheck] 最常用的选项，运行valgrind中名为toolname的工具。省略则默认运行memcheck

-db-attach=[default:no] 绑定到调试器上，便于调试错误。

具体例子如下图。



# 第3章 Cache模拟与测试

## 3.1 Cache模拟器设计

csim.c见附件。

程序设计思想：

首先阅读整个目录文件，得知已经给出csim.c的框架csimframe.c，阅读csimframe.c，其中已经给出了cache行，组，整个cache的结构体定义，将S，s，B，b，E等定义为全局变量，定义了miss\_count，hit\_count，eviction\_count等统计数字，lru\_counter则记录执行每一行指令的时间点。verbosity用来记录用户是否想要输出轨迹。

由于本次模拟并不针对具体的数据，因此cache行的结构体定义中并没有数据块，同时根据lru策略加上一个记录最后被访问时间的lru。

csim.c中有main，initCache，replayTrace，accessDate，freeCache函数，用以模拟Cache。printSummary函数负责hit\_count，miss\_count，eviction\_count的输出。printUsage则是当用户没输入命令行参数时进行提示。

main()中读取命令行参数，记录下s，b，E，verbosity，文件名，然后计算出S，B。然后调用initCache初始化Cache，调用replayTrace用以模拟Cache执行指令。最后freeCache释放空间，然后调用printSummary进行输出。

initCache，根据S，E等申请内存空间，然后计算组索引的下标从第几个二进制位开始。

replayTrace，首先读取文件中的一行，如果是命令是S，L则进行一次数据读取（调用函数accessData），M进行两次，I不进行。每一行指令中含有读取数据的长度，但是PPT中已经提到，不会出现读取数据时过界的问题。因此len不作为accessData的参数，只有addr。如果用户输入的命令行参数中有-v，即verbosity=1，则打印这一行的内容。

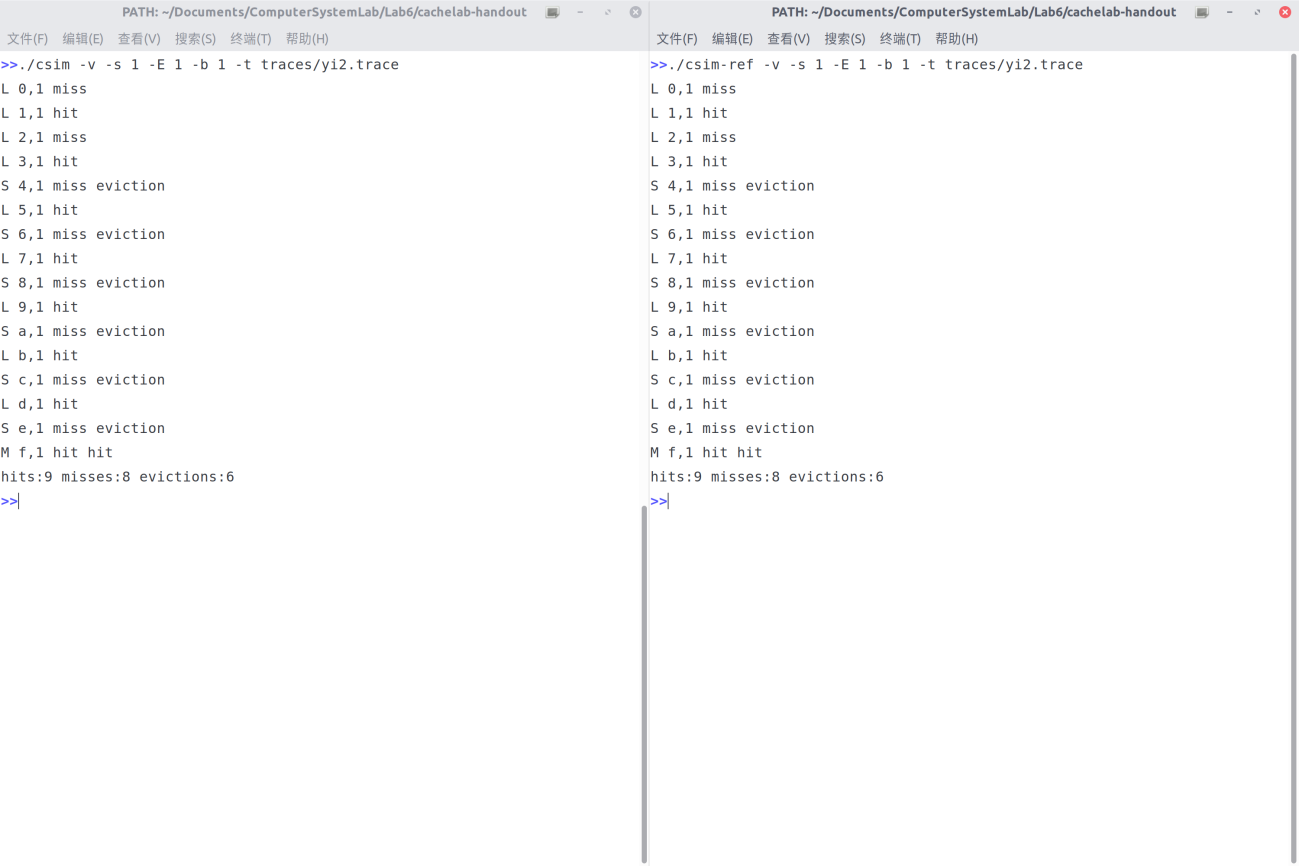
assessData，首先地址为64位。从最低位开始，b位是块偏移量。然后s位是组索引，然后剩下的位作为tag。使用第二章中位操作的知识，提取出组索引，tag的具体值。

首先从cache中找到缓存组，然后逐行遍历。如果找到了有效并且tag匹配的缓存行，则更新最后一次访问时间lru，hit\_count++，如果verbosity=1，则打印hit ，然后退出。如果没找到，则miss\_count++。然后看是否有无效的行，有则更新为新行，将新行的tag，最后一次访问时间逐一设置，并且设置为有效，然后依情况打印miss，退出。如果找不到无效的行，则将这一组里面/最后一次被被访问的时间/离现在最长的一行驱逐，eviction\_count++，替换，设置新行，依情况打印miss eviction，然后退出。

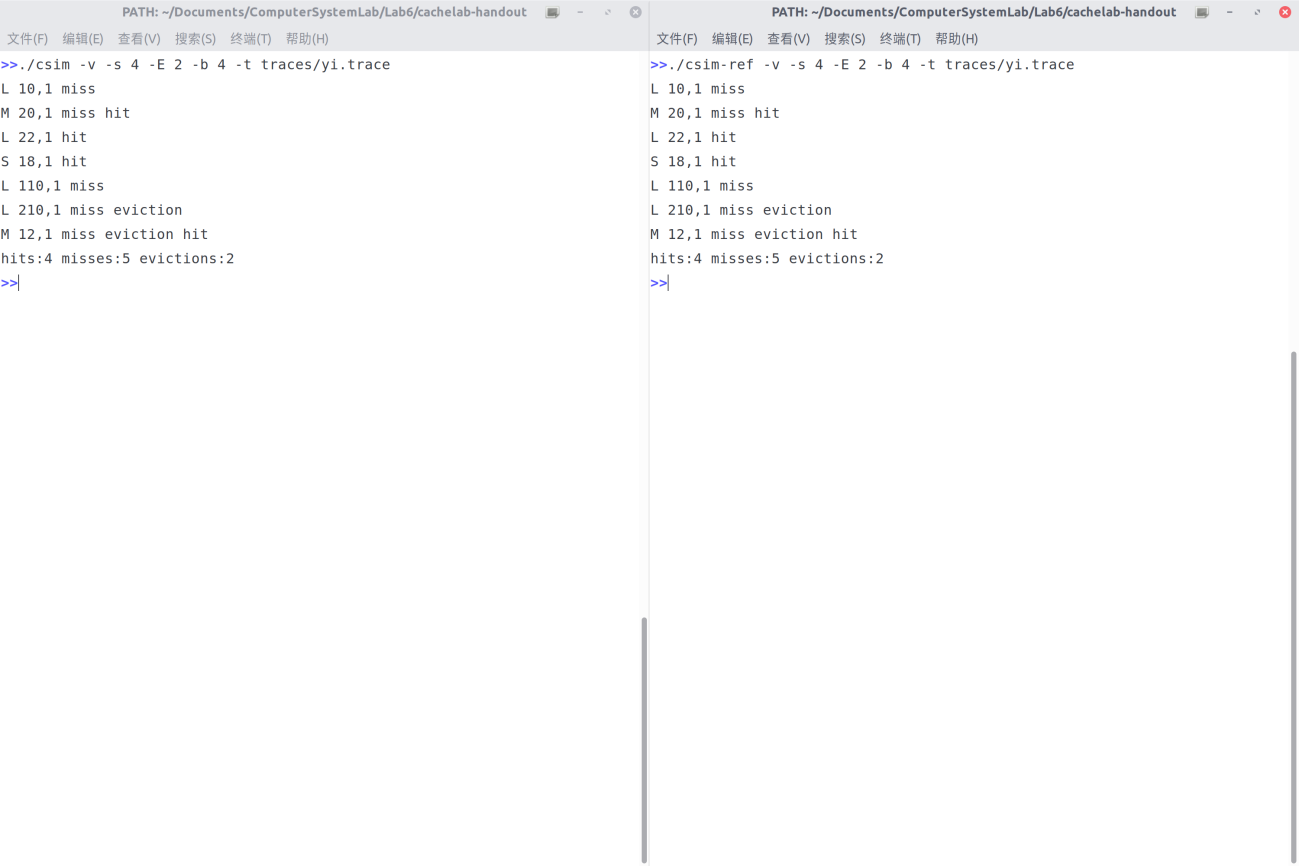
总的的来说，accessData的思路为：找到则hit。找不到就miss，载入新行。优先替换无效的行，找不到无效的行就淘汰最后一次访问最久远的一行。

freeCache，free即可。

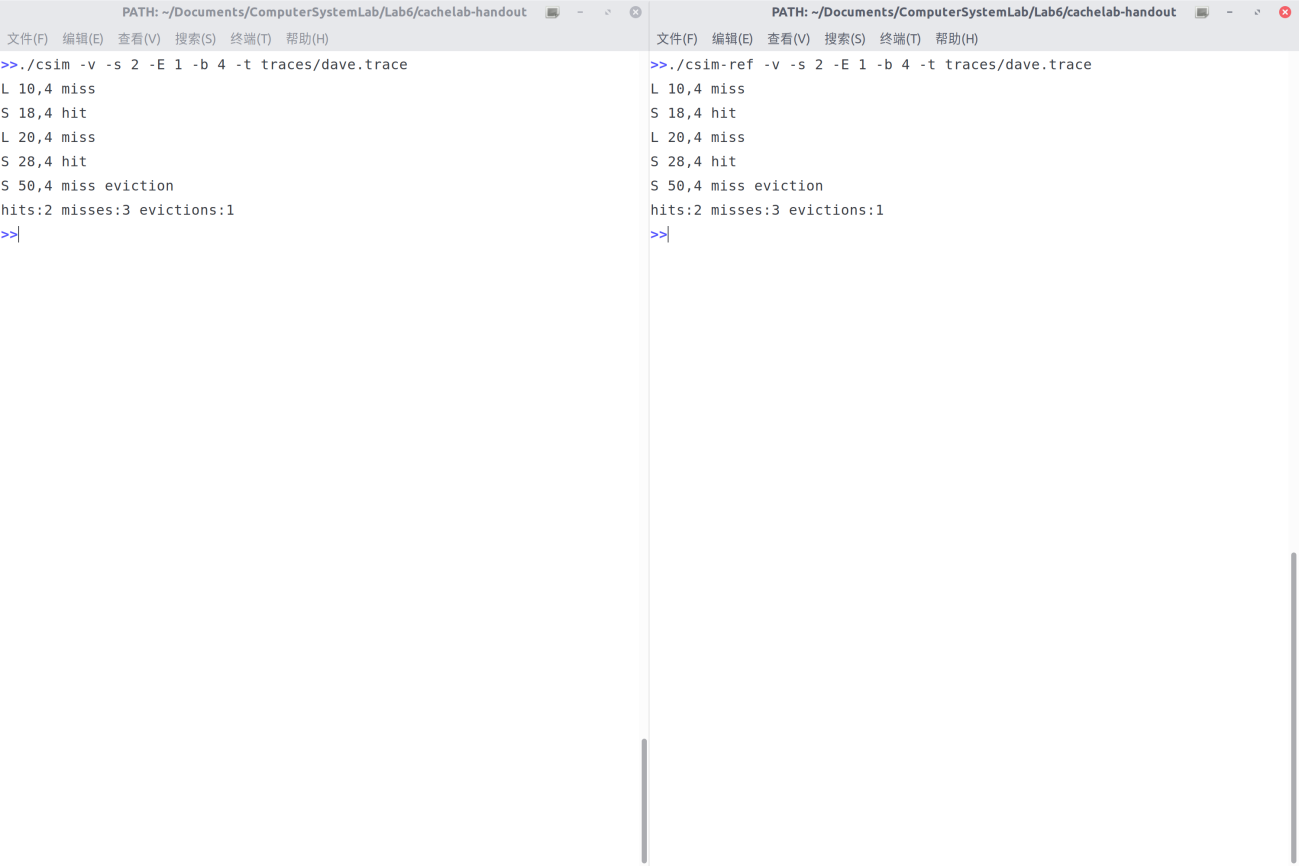
测试用例1的输出截图（5分）：



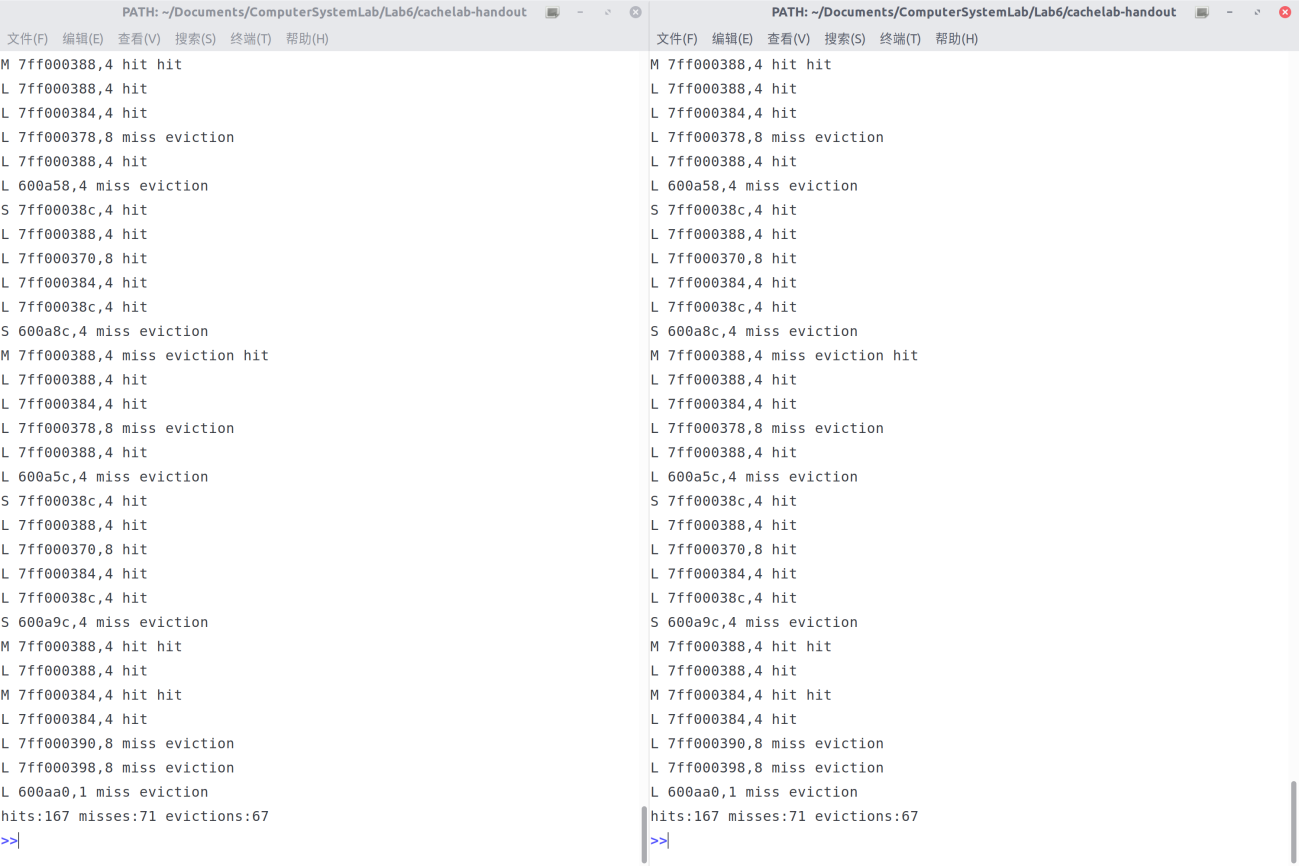
测试用例2的输出截图（5分）：



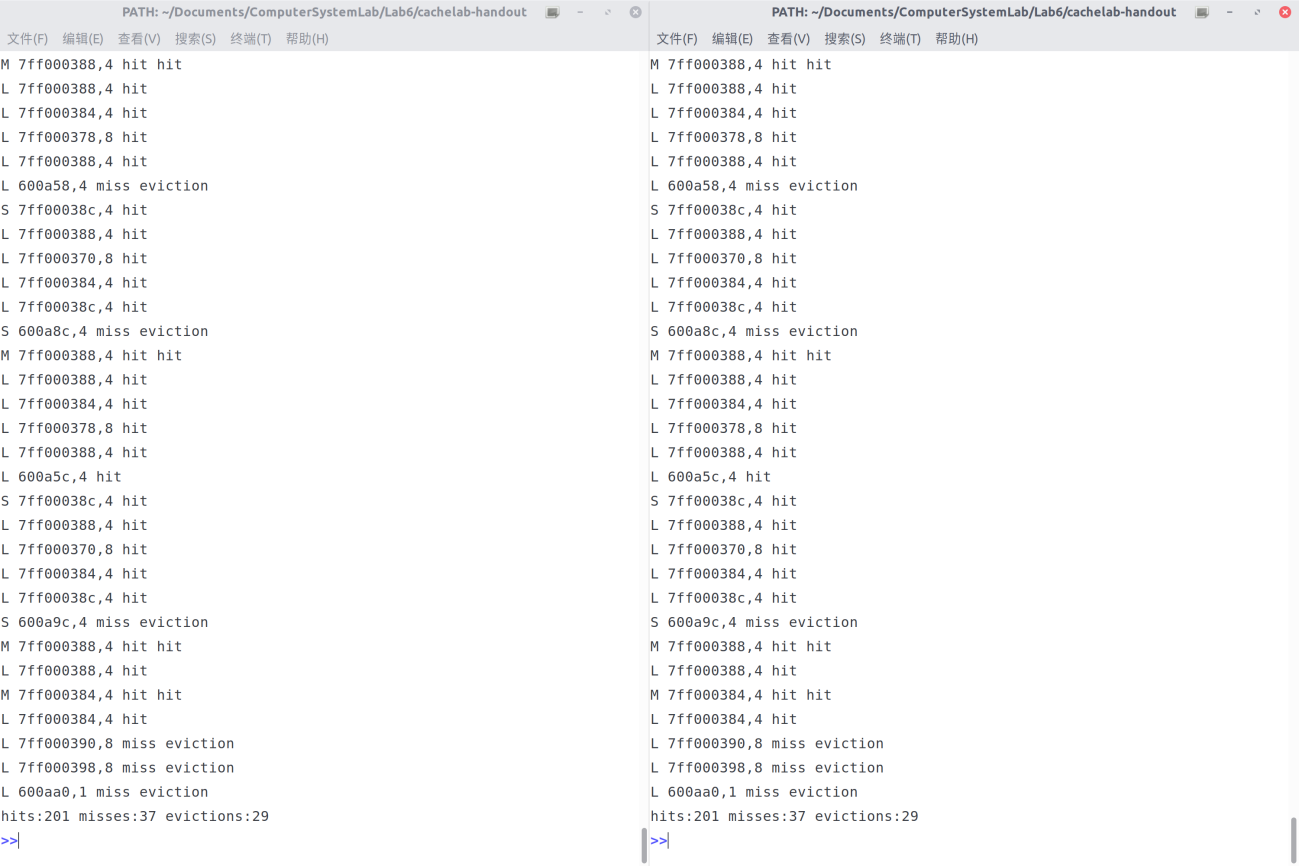
测试用例3的输出截图（5分）：



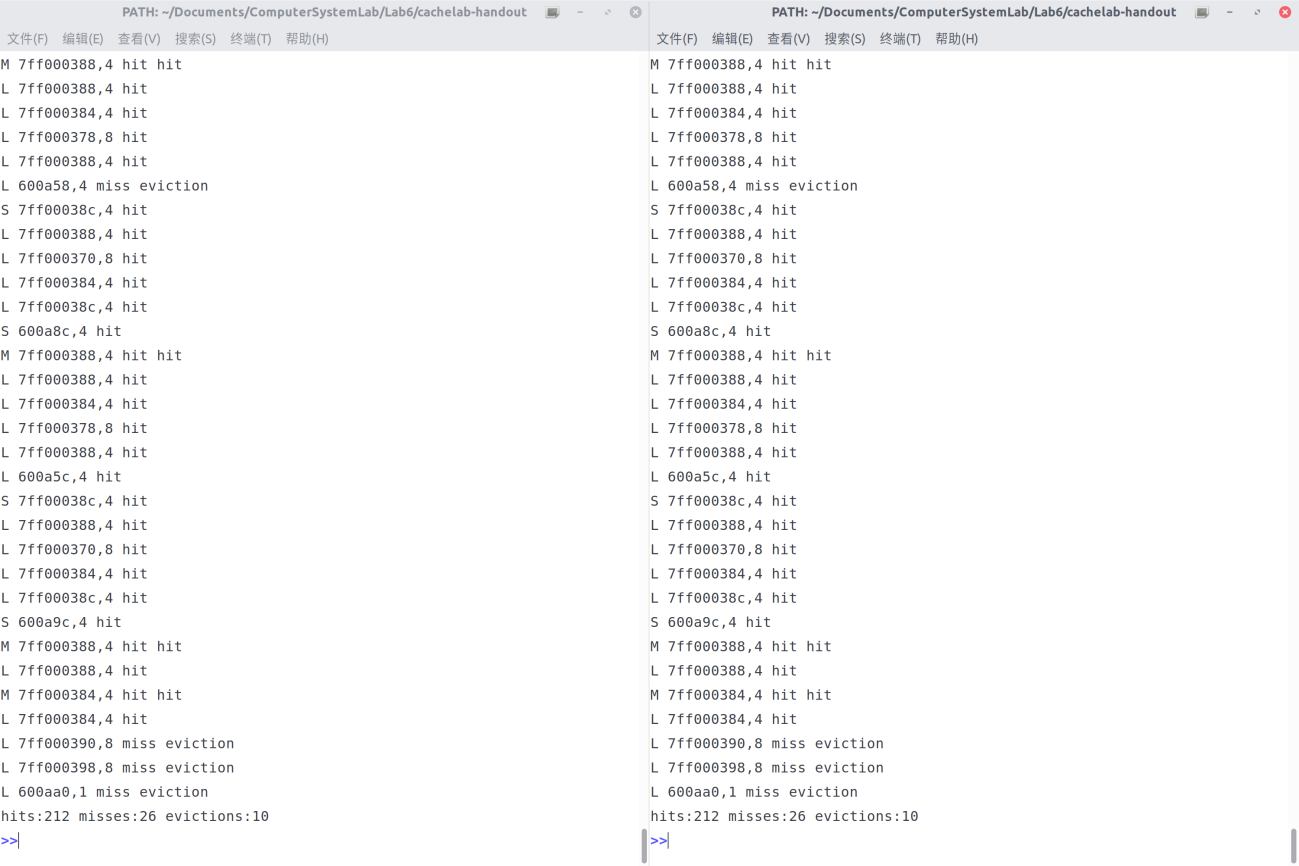
测试用例4的输出截图（5分）：



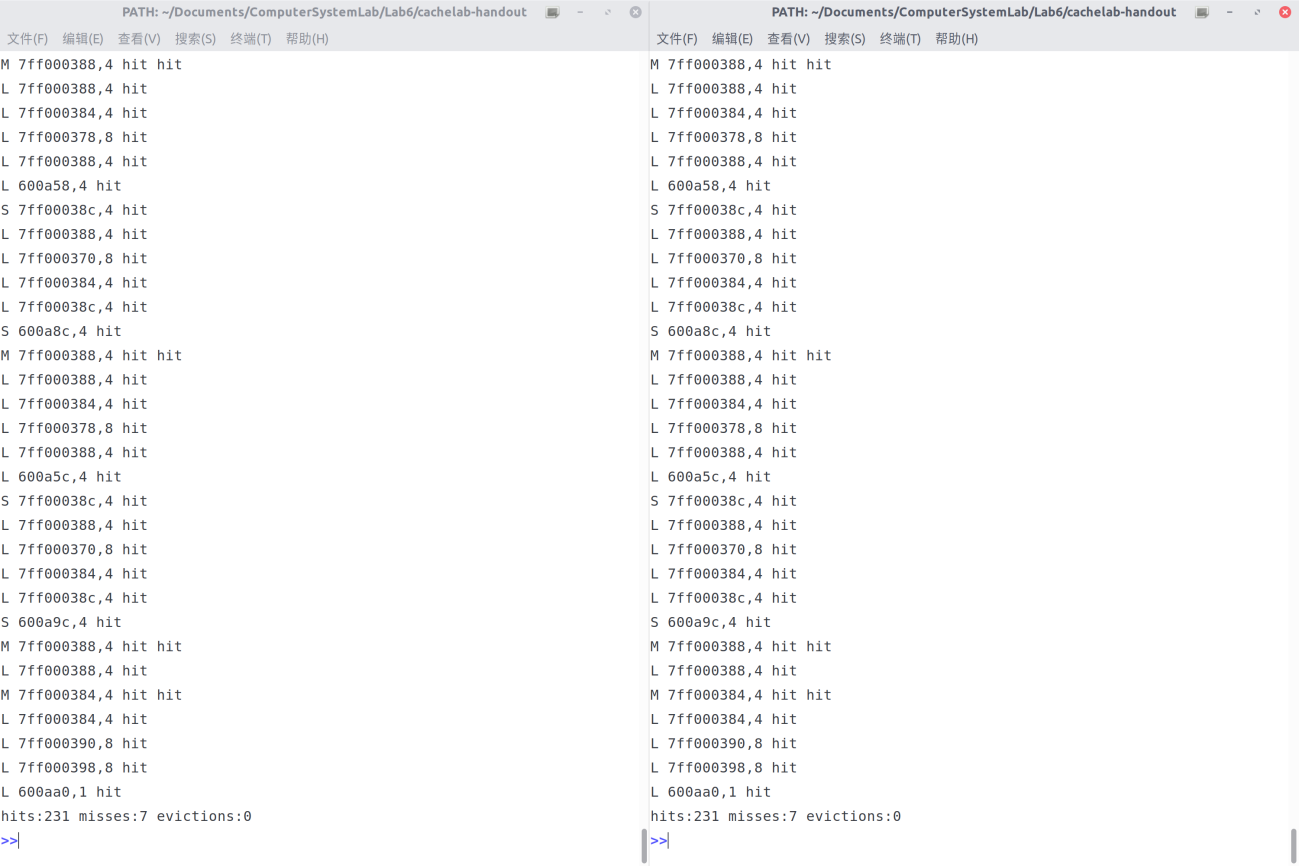
测试用例5的输出截图（5分）：



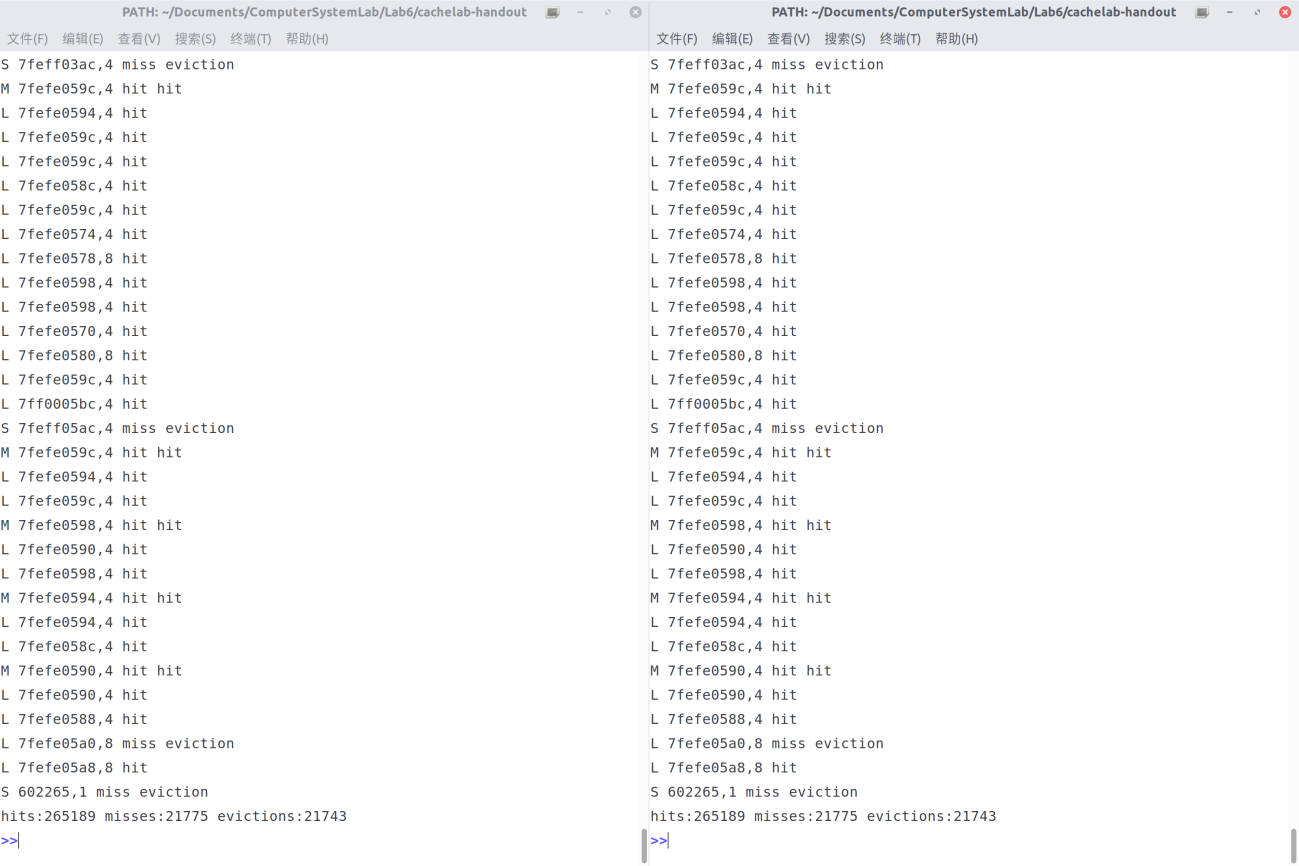
测试用例6的输出截图（5分）：



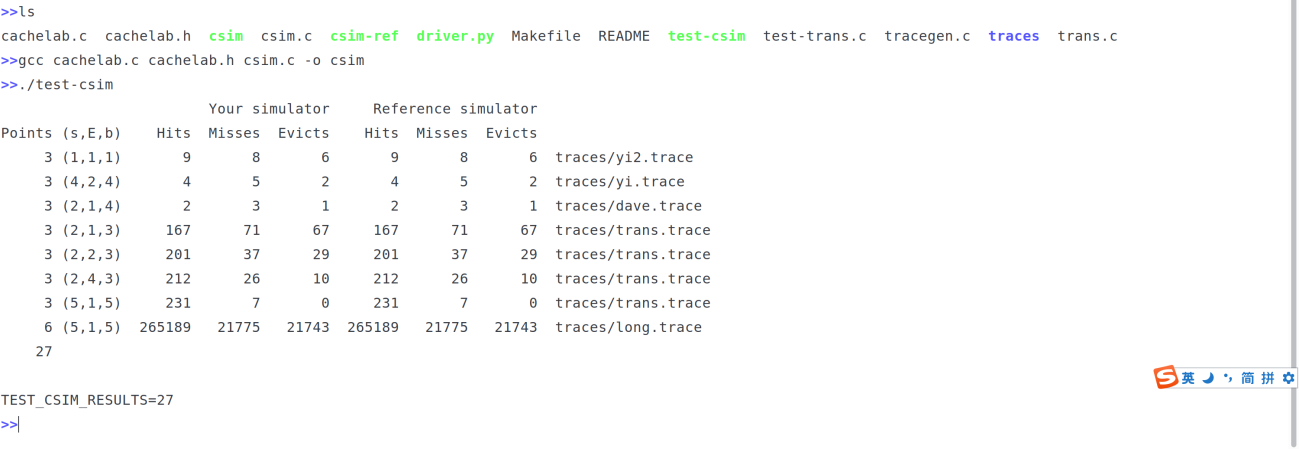
测试用例7的输出截图（5分）：



测试用例8的输出截图（10分）：



test-csim测试函数。



**注：每个用例的每一指标5分（最后一个用例10）**——与参考csim-ref模拟器输出指标相同则判为正确

## 3.2 矩阵转置设计

提交trans.c

见附件

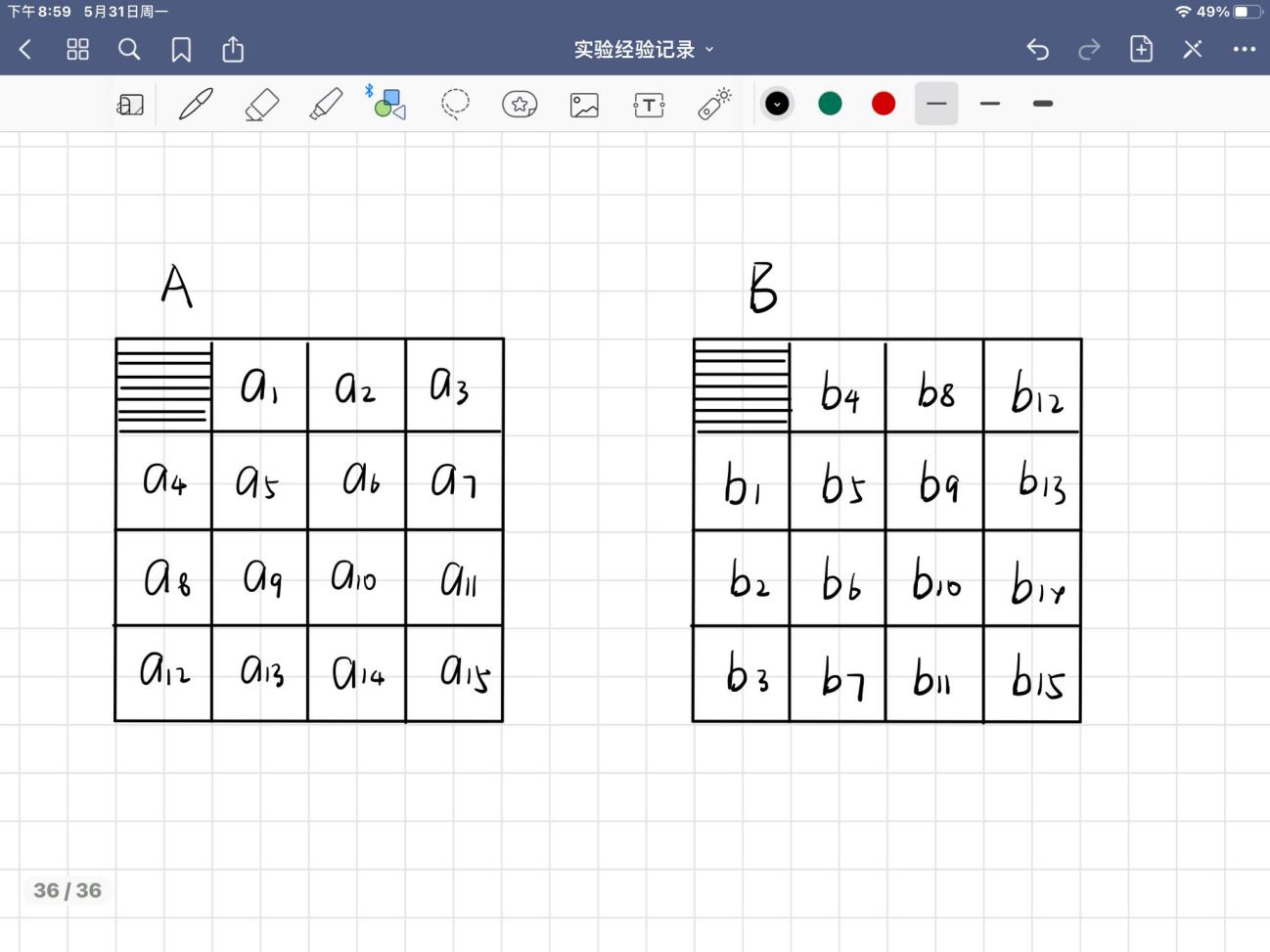
程序设计思想：

32×32：

首先分析，首先思考A，B矩阵的内存地址，观察程序，A，B的地址空间应该是连着的。而A矩阵的大小为4\*32\*32=4096，所以对于A[i][j]和B[i][j]可以认为他们的内存地址之差为4096byte，那么他们地址后10位是相同的，映射时会映射到相同的组，只是tag不同。其实由于s+b是10位，因此每隔1024字节，他们就会映射到同一个组。

然后思考如何才能最小化miss。

为方便思考，不妨将矩阵A，B划分为4\*4大小的块，每个块大小为8\*8，一行（32byte）恰好是一个数据块的长度。

不妨设首先读取的元素为A[0][0]，应该被放置到B[0][0]。A[0][0]和B[0][0]

会被映射到同一个组，所以A[0][0~7]被读取到cache中之后，应该将之保存下来，让这些元素不会再被读入到cache。这些元素就应该被放置在B[0~7][0]，所以这时候b0块全部元素都要进入cache，占据8个组，每个组（行）第一个元素被赋值。此时最好的方式是将A[1][0~7]中元素读入cache，原有B[1][0~7]被淘汰，将A中元素保存后，在将B[1][0~7]读入cache，再给B[0~7][1]赋值。如此处理剩下的a0块中元素，这个块共有1+8+2\*7=23次miss。

如上图，a0块成功赋值给了b0块，根据翻转的规则，剩下的ai块也对应的给bi块赋值。当ai不处于对角线时，ai和bi块中元素会映射到不同的缓存组。处于对角线时，则会映射到相同的缓存组。这么区分不会减少miss，只会减少参数的使用。

64×64：

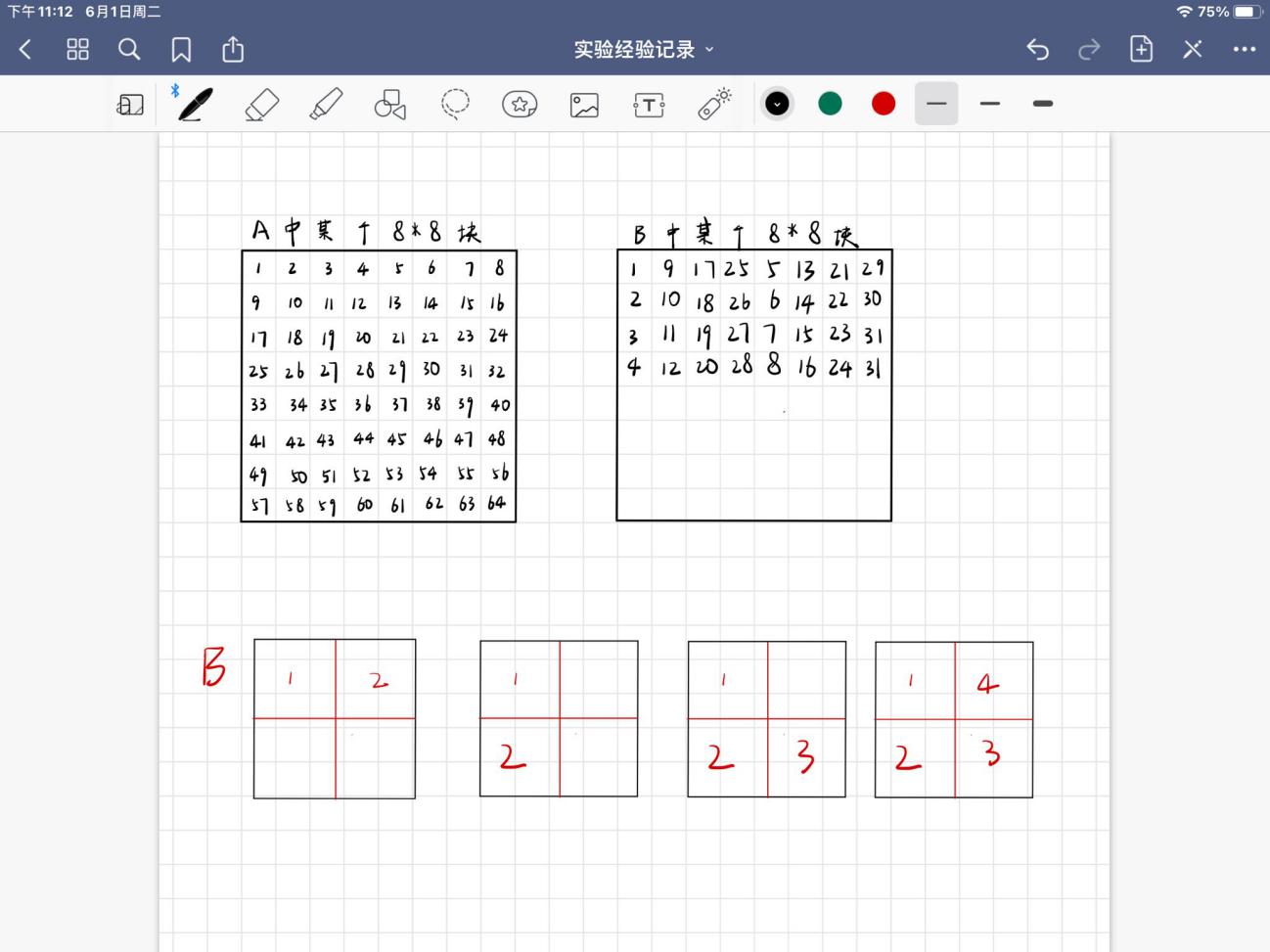
思考能否按同样的思路处理64\*64，即同样按8\*8分块，将4->8后发现，miss\_count为4723，思考后发现，此时不再是每隔4个块便映射到相同的组，而是每隔四行便映射到相同的组。同一个块，前四行，后四行会冲突。

为了减少miss，同样读取8个int，放入一个组，所以还是采用8\*8分块的方法来思考。分析不难得出，只要处理好一个8\*8块，其他的8\*8块采用同样思路即可。

关于8\*8块的处理，8\*8块即为8行8列，0~3列与4~7列会映射到相同的组里面，所以如果直接取A的一行为B的一列赋值，或取A的一列为B的一组赋值，都会导致很大的miss，也就是上面的4723。

那如何处理8\*8块？我想出了两种策略。

1. 如下图所示，为叙述方便，下一段中A,B均代指其中某一个8\*8块。并且将B中的8\*8块以4\*4分块。

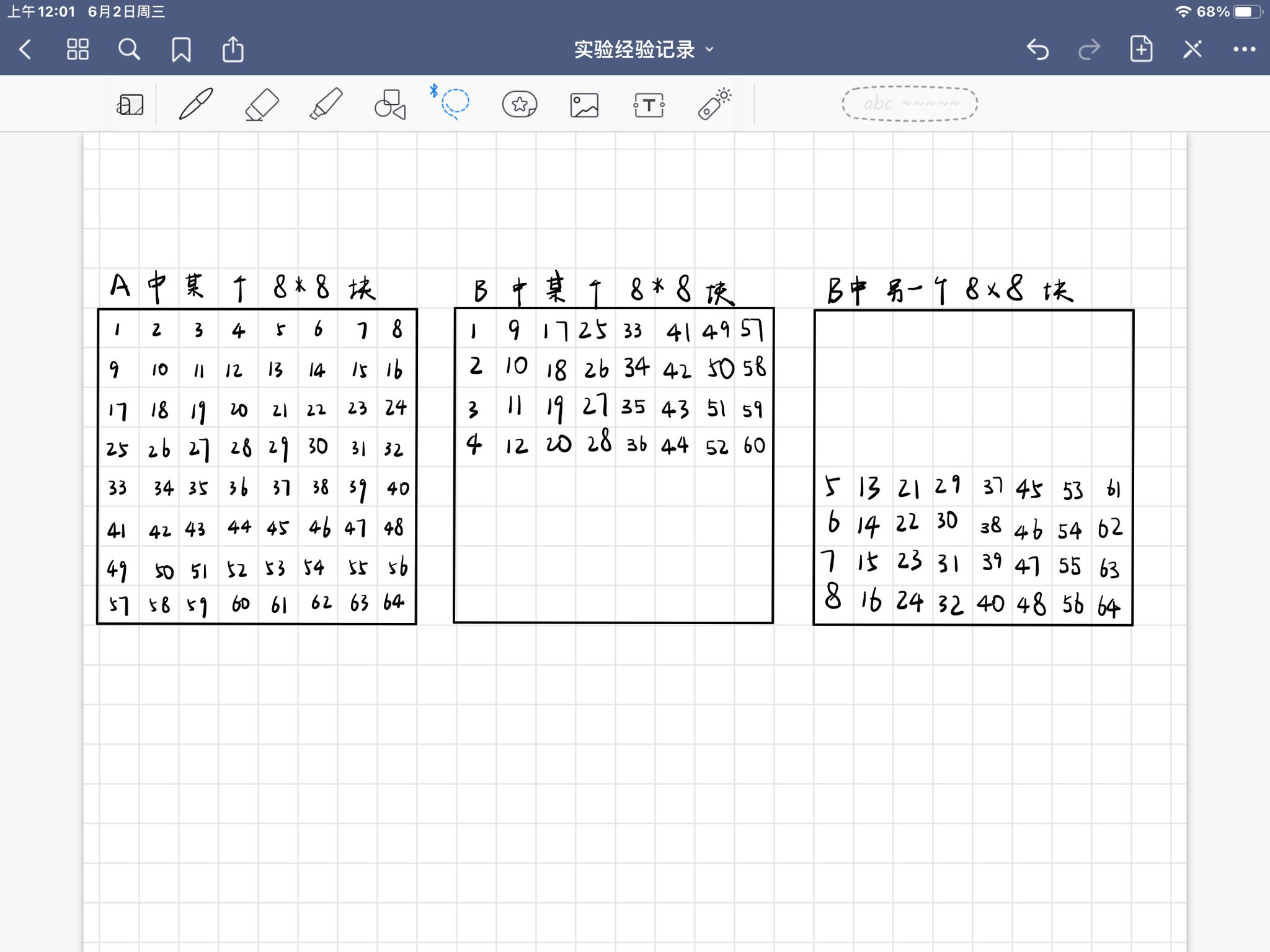


这种策略的思想是：将本来要放到B[0][4~7]的部分，放在B[4][0~3]中，A的0~3行都这样处理。如此填满了B上方两个4\*4块，然后将右上方4\*4块移动到左下角，然后取A中4~7行，依次填满B中右下角，右上角。

但是这样做效果不好，依次填满B中右下角，右上角的过程miss太多。最后miss次数为3635。

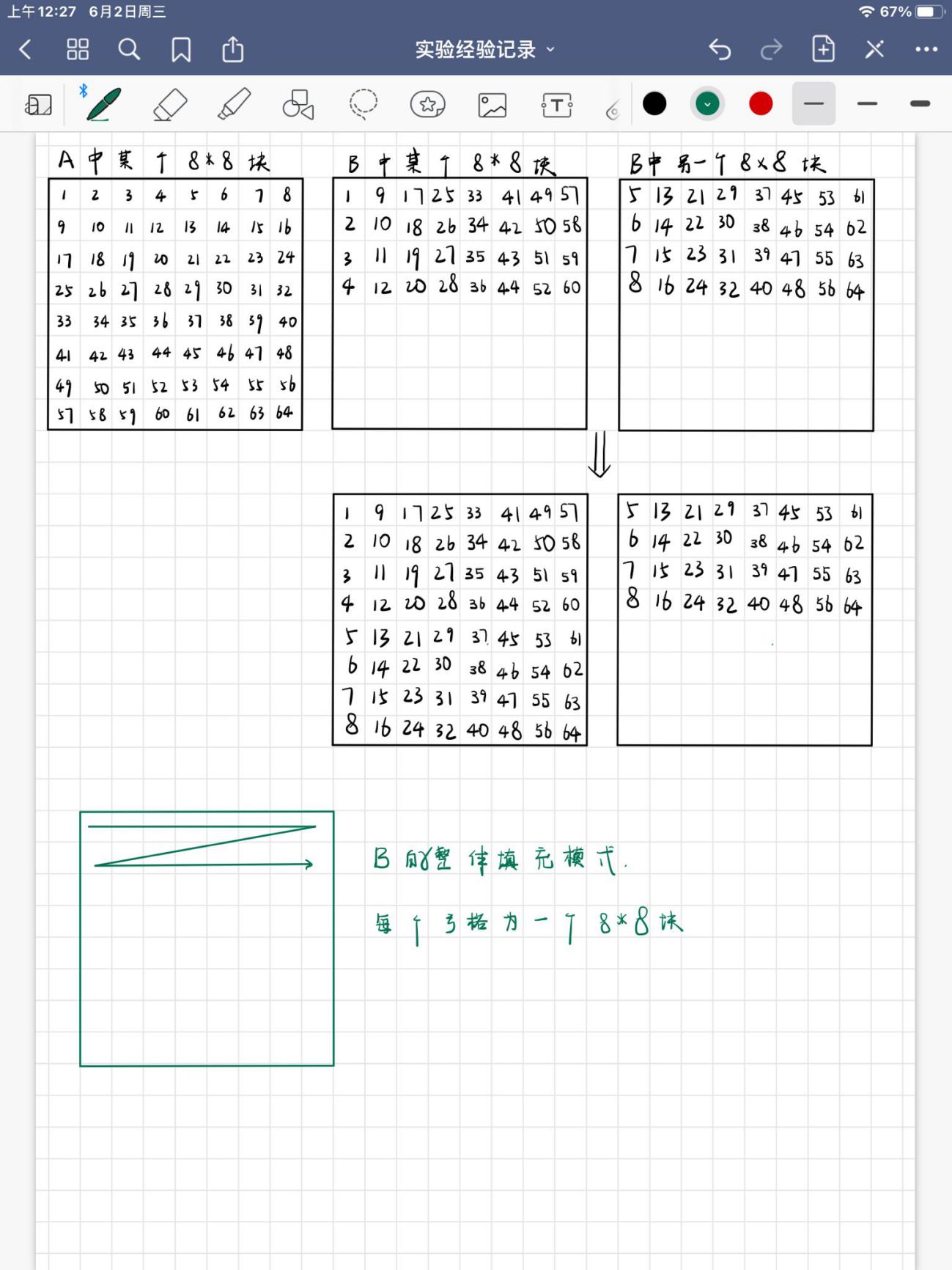
于是对这一策略进行优化，在B中右上角转移到左下角的过程中，同时取出A中4~7行中的一行，同时完成B中左下角，右上角的排列。最后再完成右下角的排列。同时完成B中左下角，右上角的排列，我想出了两种方法，一种方法不小心改变了A矩阵，或者说是在用A矩阵的空间进行存储。第二种方法是取A中的一行时，保存前四个，放入B的右上角。但在此之前，读取B的右上角，将其中的某一行也保存下来，放到B左下方，最后miss为1749。

1. 其实第二个策略与第一个策略交替进行，只不过第二个策略率先取得了突破，miss降低到了1300以下。



如图所示，每从A中取一行，前4个放在应该在的块中前四行，后四个放在另一个块的后四行，这两个块相邻，最后一个块则放入到第一个块中。这样读取A中全部的块之后，再对B中的元素重排列至应有位置。这样的话miss为1491。

对上述方法继续进行优化，原来读取A中8\*8块是先行后列的方法，这也导致了填充B中的8\*8块是先列后行。这样做下一个块总是会把前一个块挤出cache。因此将之变换顺序，B中块以先行后列的方式被填充。同时应该放在下一个块中的后四列，改为放在下一个块中前四列。



同时不在将所有的块都处理完之后再调整顺序，而是每摆完一个块，就调整顺序。

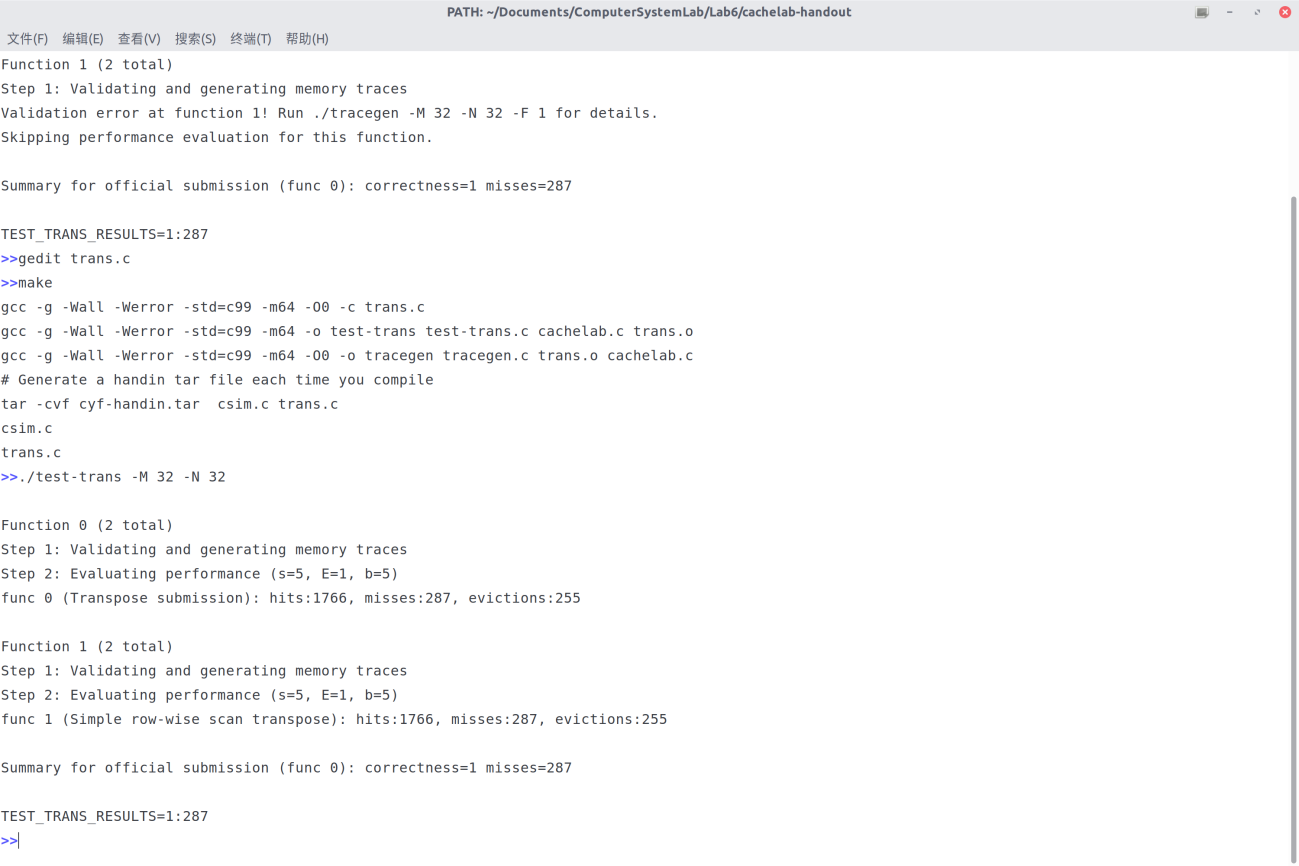
这样处理的好处是：趁着下一个块的前四列还在cache中的时候，将其中的数据转移到当前块的后四列，因为当前块的前四列已经摆放好了，因此可以顺利的将之淘汰。同时摆放好之后，下一个块的前四行仍然还在缓冲区，那么就再处理这个块，向其中填入应该填的数据。所以才有了对B中的块先行后列的填充。

如果是每行最后一个块，那就以下一行第一个块为下一个块，如果是最后一个块，就采用策略一中的方法处理。这样的话，块的miss减少到了1159。

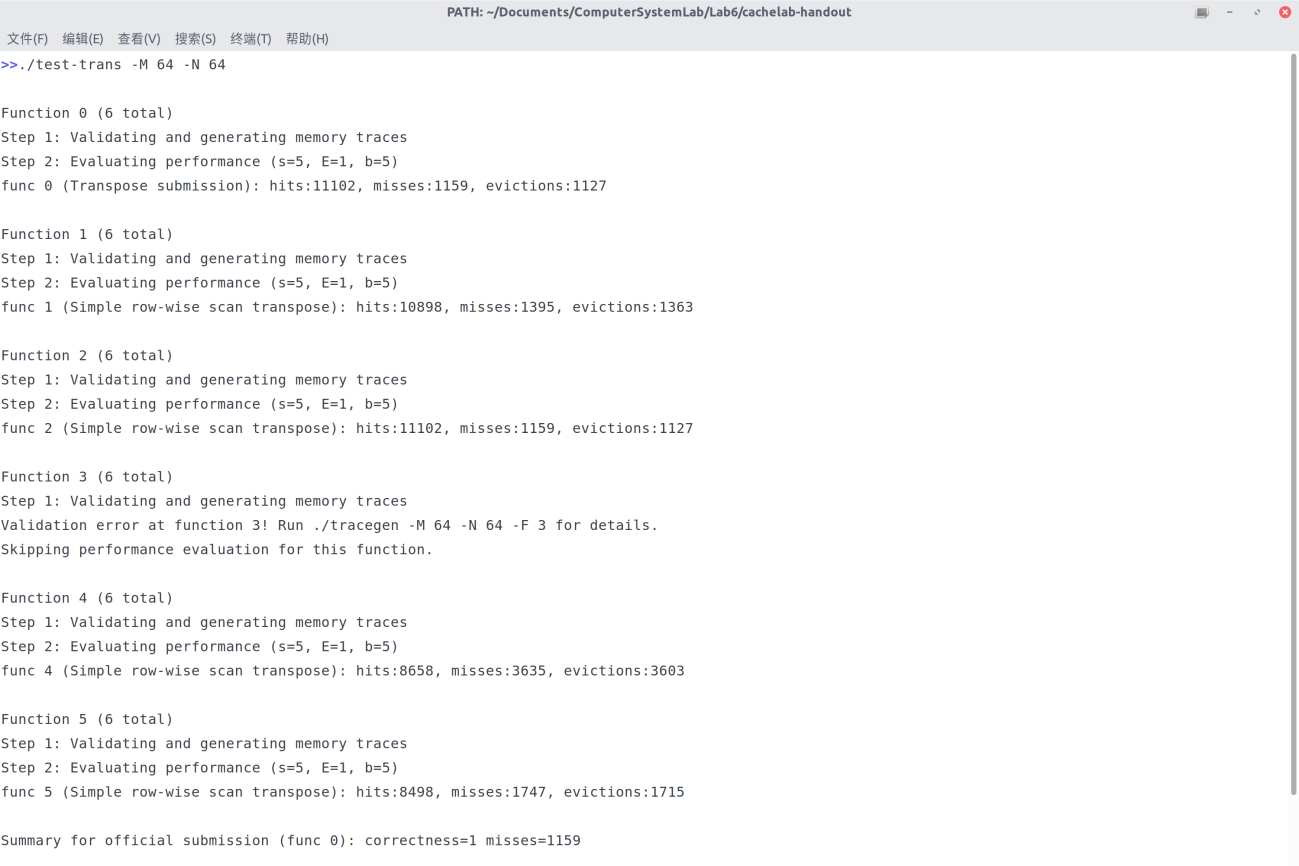
61×67

与之前的矩阵相比，这一矩阵不再规则，不能再以8\*8的模式恰好划分，但是还是可以采用同样的思路，进行了部分调整。

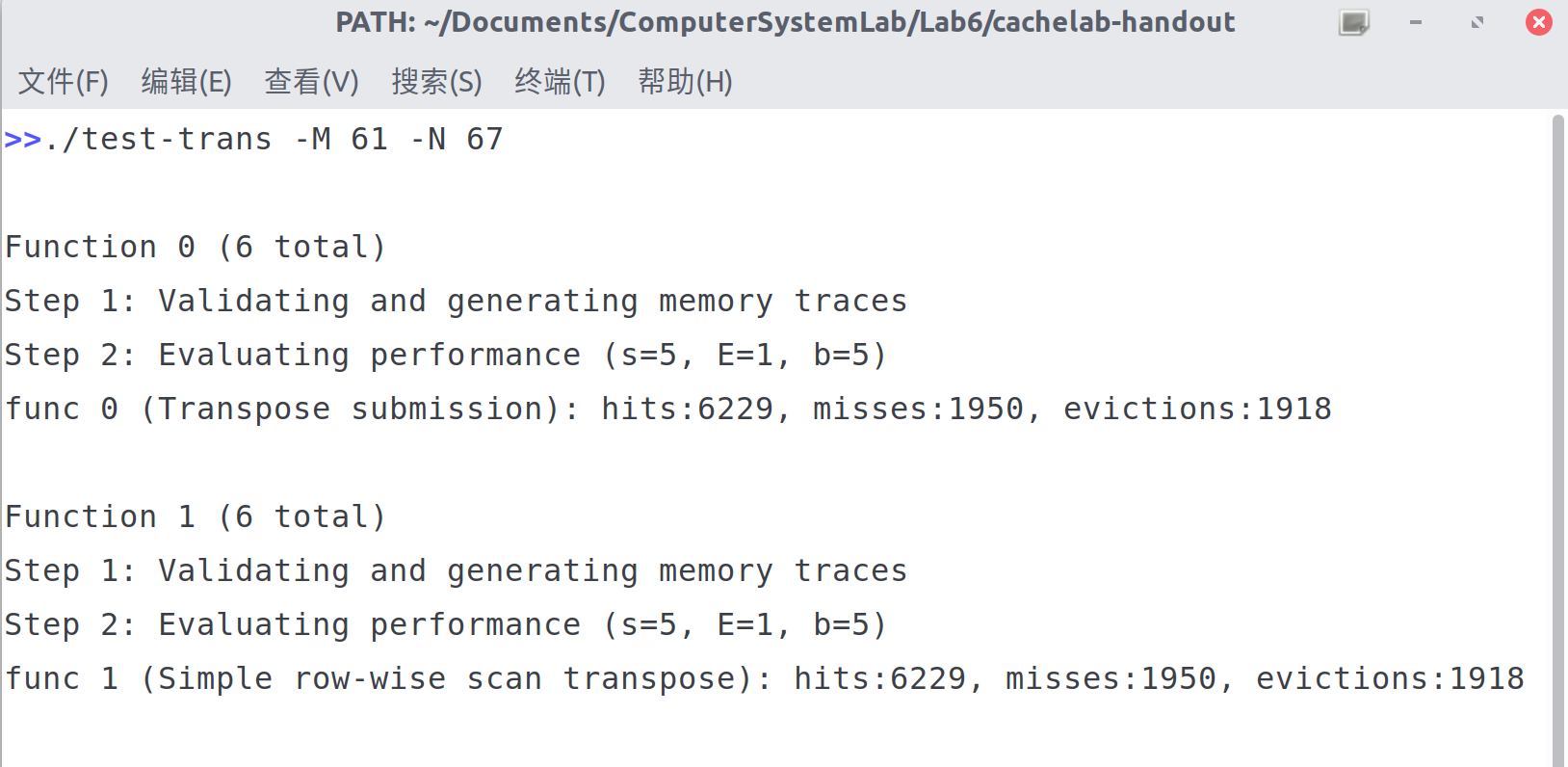
**32×32（10分）：运行结果截图**

****

**64×64（10分）：运行结果截图**

****

**61×67（20分）：运行结果截图**

****

# 第4章 总结

## 4.1 请总结本次实验的收获

增加了对cache，lru的理解。

对循环有了更深的认识。

对矩阵转置有了更深的理解

学会使用valgrind工具。

## 4.2 请给出对本次实验内容的建议

矩阵转置有点难

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/ collection/anatmorp.