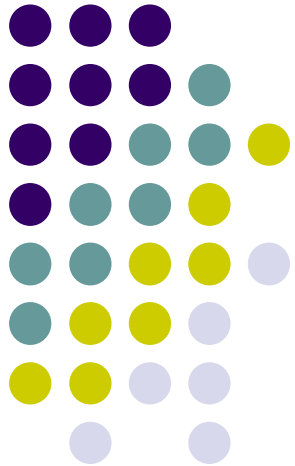
cache模拟器的设计与实现

《计算机系统结构》课程组



实验概述

- **实验目的**
  - 理解cache工作原理；
  - 如何实现一个高效的模拟器。
- **实验内容**
  - 编写一个200-300行的C程序来模拟Cache缓存的行为
- **实验环境**
  - Linux 64-bit ， C语言

实验数据与文件

- 实验数据包：**cachelab-handout.tar**
- 解压命令：**tar xvf cachelab-handout.tar**
- 数据包中的重要文件与目录：
  - csim.c：实验中需要修改和提交的Cache模拟程序
  - csim-ref ：供参考的二进制可执行Cache模拟器（模拟一个具有任意大小、关联度和LRU（least-recently used）替换策略的Cache）
  - traces子目录：包含一组参考内存访问轨迹文件（reference trace files，由valgrind程序生成），用以评估Cache模拟器的正确性
  - test-csim：测试程序，用以验证Cache模拟器在上述参考内存访问轨迹上的正确性

实验数据与文件

- 内存访问轨迹文件
  - 位于traces子目录中，用以评估Cache模拟器的正确性
  - 记录了某一程序在运行过程中访问内存的序列及其参数（地址、大小等）
  - 每行记录1或2次内存访问的信息，格式为：

    [0-1个空格] **operation** **address**,**size**

    operation（操作）：内存访问的类型。I - 指令装载，L - 数据装载，S - 数据存储，M - 数据修改（即数据装载后接数据存储）

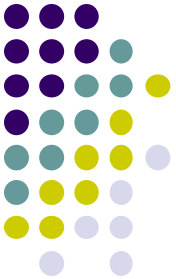
    address：所64-bit十六进制内存地址

    size：访问的内存字节数量
  - 示例：

```
I    0400d7d4,8
 M   0421c7f0,4
 L   04f6b868,8
 S   7ff005c8,8
```

注意：I符号前没有空格，而每个M、L、S符号前总有一个空格，代表对应的数据访问是由指令（执行）引起的

实验内容：编写Cache模拟器



- 任务：在**csim.c**提供的程序框架中，编写实现一个**Cache**模拟器：
  - 输入：内存访问轨迹
  - 操作：模拟缓存相对内存访问轨迹的命中/缺失行为
  - 输出：命中、缺失和（缓存行）淘汰/驱逐的总数
- 具体要求：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。
- 命令行格式：**csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>**
  - -h: 显示帮助信息（可选）
  - -v: 显示轨迹信息（可选）
  - -s <s>: 组索引位数
  - -E <E>: 关联度（每组包含的缓存行数）
  - -b <b>: 内存块内地址位数
  - -t <tracefile>: 内存访问轨迹文件名

示例：

```
$>./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
```

实验内容：编写Cache模拟器

- **csim.c编程要求：**
  - 模拟器必须在输入参数s、E、b设置为任意值时均能正确工作——即需要使用malloc函数（而不是代码中固定大小的值）来为模拟器中数据结构分配存储空间。
  - 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cache访问（即轨迹中“I”起始的行）
  - 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块的边界——因此可忽略访问轨迹中给出的访问请求大小
  - main函数最后必须调用printSummary函数输出结果，并如下传之以命中hit、缺失miss和淘汰/驱逐eviction的总数作为参数：

    printSummary(hit_count, miss_count, eviction_count);

csim.c代码框架

```
#include “cachelab.h”
int main() {
    int hit_count = 0, miss_count = 0, eviction_count = 0;
    … …
    printSummary(hit_count, miss_count, eviction_count);
    return 0;
}
```

- 每一数据装载(L)或存储(S)操作可引发最多1次缓存缺失(miss)
- 数据修改操作(M)可认为是同一地址上1次装载后跟1次存储，因此可引发2次缓存命中(hit)，或1次缺失+1次命中外加可能1次淘汰/驱逐(evict)

实验内容：编写Cache模拟器

- **Cache性能测试：**
  - **8个测试用例——**不同Cache参数和访问轨迹

```
linux> ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
linux> ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
linux> ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
linux> ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
linux> ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
linux> ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
linux> ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
linux> ./csim -s 5 -E 1 -b 5 -t traces/long.trace
```

```
 L 10,1
 M 20,1
 L 22,1
 S 18,1
 L 110,1
 L 210,1
 M 12,1
```

```
I  0040056d,3
 L 7ff00038c,4
I  00400570,3
 S 00600a70,4
I  00400573,4
 M 7ff000388,4
I  00400577,4
 L 7ff000388,4
I  0040057b,2
I  0040053c,3
 L 7ff000384,4
```

实验内容：编写Cache模拟器

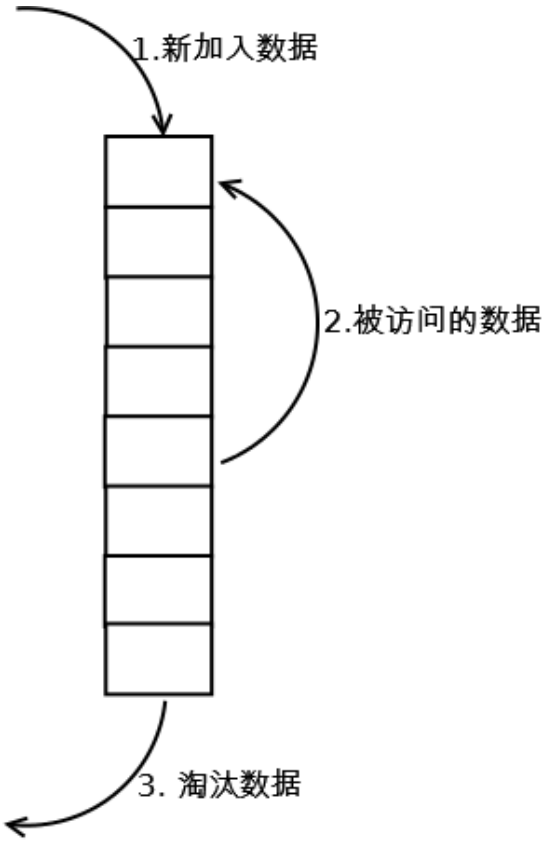
- **Cache性能测试：**
  - **test-csim测试程序：**依次使用上列每一测试用例对csim进行测试
    - 对每一测试，test-csim从缓存的Hits（命中）/Misses（缺失）/Evicts（淘汰/驱逐）数量三个指标比较了所实现csim模拟器和参考Cache模拟器csim-ref的性能，
    - 计算csim实现获得的分数：每个用例的每一指标1分（最后一个用例2分）——与参考csim-ref模拟器输出指标相同则判为正确：

```
linux> ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts     Hits  Misses  Evicts
3      (1,1,1)    9     8       6          9     8       6       traces/yi2.trace
3      (4,2,4)    4     5       2          4     5       2       traces/yi.trace
3      (2,1,4)    2     3       1          2     3       1       traces/dave.trace
3      (2,1,3)    167   71      67         167   71      67      traces/trans.trace
3      (2,2,3)    201   37      29         201   37      29      traces/trans.trace
3      (2,4,3)    212   26      10         212   26      10      traces/trans.trace
3      (5,1,5)    231   7       0          231   7       0       traces/trans.trace
6      (5,1,5)    265189 21775  21743      265189 21775  21743   traces/long.trace
27
```

缓存替换算法 - LRU

LRU（Least recently used，最近最少使用）算法

- 根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。
- 最常见的实现是使用一个链表保存缓存数据：
  - 新数据插入到链表头部
  - 每当缓存命中（即缓存数据被访问），则将数据移到链表头部
  - 当链表满的时候，将链表尾部的数据丢弃



实验数据提交

- 修改完成实验的结果文件**csim.c**后，在实验数据的根目录中执行如下命令进行编译：

  linux> make clean

  linux> make

- 每次如上执行**make**命令时，相应**Makefile**将创建一个名为**"-handin.tar"**的文件，其中包含你需要提交的**csim.c**和**trans.c**文件。

- 将该**tar**文件重命名为“姓名+学号**.tar**”并提交。

谢 谢！