华中科技大学

计算机系统结构实验报告

姓　　名：　　范唯

学　　院：　　计算机科学与技术

专　　业：　　计算机科学与技术

班　　级：　　CS1703

学　　号：　　U201714670

指导教师：　　王芳　陈俭喜

分数	
教师签名	

2020 年　4 月　26 日

目　录

1. Cache模拟器实验

1.1. 实验目的

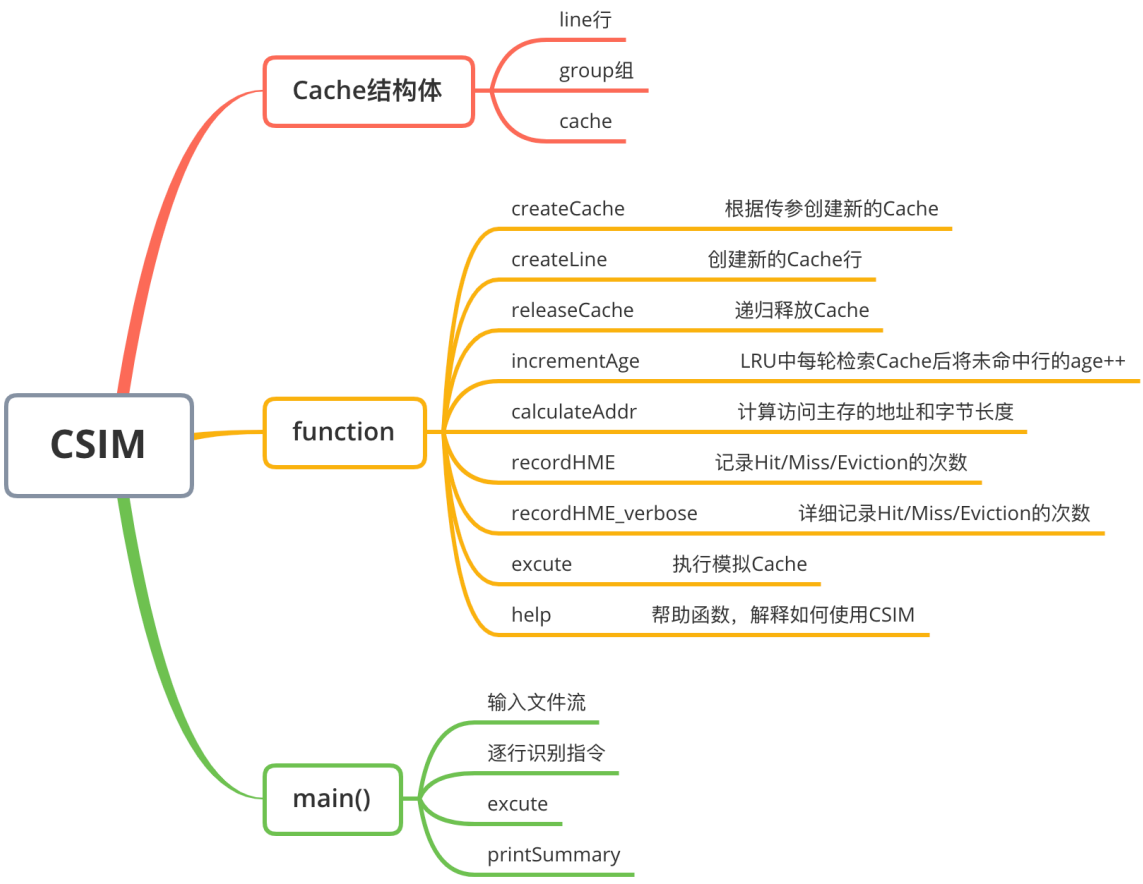
- 理解cache工作原理
- 如何实现一个高效的模拟器

1.2. 实验环境

- 硬件：华为云ECS弹性云服务器
- 系统：Linux
- 软件：SSH、Visual Studio Code

1.3. 实验思路

实验思路思维导图



- **Cache结构体的实现**

valid	tag	age	block
有效位	组号	LRU替换算法age	存储块

```
typedef struct line{
  int age;           // 使用次数
  char valid;      //有效位
  int tag;       //组号
  int* block; //块
  } line;

typedef struct group{
    line* lines; // 每一组有多少行
} group;

typedef struct cache{
  group* groups;     // 每一个cache有多少组
} cache;
```

- **createCache**

  根据传入的参数groupNum、lineNum来创建Cache并分配空间，并且把每一行的valid位置为0。

  等待插入line时再将valid置为1

- **createLine**

  创建新的cache的line，并且根据块的大小分配地址空间，

  并将valid位置为1，age置为-1。

- **recordHME & recordHME_verbose**

代码配合注释描述整个流程

```
/* 计算出Hit/Miss/Eviction的次数 */
void recordHME(group* cur_group, int lines, int tag, int blockSize, long
memAddr,int* hits, int* misses, int* evictions){
    int InvalidLine = -1;//用于记录没有被填充的Line
    int evictionLine = -1;//用于记录需要被替换的Line
    int max_age = 0;
    for(int i = 0; i < lines; i++){//实现多路选择器，在组号相同的情况下，对比该cache组
中有无line拥有相同的区号。如果命中，那么将其age置为-1，并且该组所有行的age++1。
        if((cur_group->lines[i].valid == 1) && (cur_group->lines[i].tag ==
tag)){
```

```
            *hits += 1;
            cur_group->lines[i].age = -1;
            incrementAge(cur_group, lines);//增加该组中其他没命中的line的age
            return;
        } else if((InvalidLine == -1) && (cur_group->lines[i].valid == 0)){
            InvalidLine = i;//记录没被用的line,准备在这一行插入新的cache行
        } else if((cur_group->lines[i].valid == 1) && (cur_group->lines[i].age
>= max_age)){//寻找需要被替换的行，age最大的行将被替换
            evictionLine = i;//当前需要被替换的行
            max_age = cur_group->lines[i].age;
        }
    }
    *misses += 1;//没有命中
    if(InvalidLine != -1){//先选择填满valid==0的行
        cur_group->lines[InvalidLine] = createLine(tag, blockSize, memAddr);
        incrementAge(cur_group, lines);
    } else{//或者把年纪最大的行替换
        *evictions += 1;
        free(cur_group->lines[evictionLine].block);//释放年纪最大的行
        cur_group->lines[evictionLine] = createLine(tag, blockSize,
memAddr);//放入未命中的块
        incrementAge(cur_group, lines);
    }
    return;
}
```

- **excute 模拟执行cache**

代码配合注释描述整个流程

```
void excute(int s, int E, int b, FILE* file, int v,int* hits, int* misses,
int* evictions){
    *hits = 0;//命中次数
    *misses = 0;//未命中次数
    *evictions = 0;

    int groups = pow(2,s),//组的数量
        blockSize = pow(2,b);//块的大小
    cache* simCache = createCache(groups, E);//创建新的cache

    char* fileLine = NULL;//文件行指针
    size_t len = 0; //每一个line的长度
    ssize_t read;
    if(v == 1){// 当命令行参数包含-v时，显示详细信息
        while((read = getline(&fileLine, &len, file)) != -1){//逐行解析
            char* printLine = fileLine;//解析每一行
            printLine[strlen(printLine) - 1] = '\0';//每一行末尾加上终止符
            if(fileLine[0] == 'I'){continue;}//I指令需要被忽略
```

```
            long memAddr;
            int addrlen = calculateAddr(fileLine, &memAddr);
            int groupNum = (memAddr >> b) & (groups - 1);
            group* cur_group = &(simCache->groups[groupNum]);
            int maxTag = pow(2,(4 * addrlen) - s - b) - 1;
            int tag = (memAddr >> (s + b)) & maxTag;
            char acctype = fileLine[1];
            if(acctype == 'M'){ //M指令需要两次访存
                recordHME(cur_group, E, tag, blockSize, memAddr, hits, misses,
evictions);
              //计算出Hit/Miss/Eviction的次数
                recordHME_verbose(&oldHits, &oldMisses, &oldEvictions, hits,
misses, evictions);//详细打印出Hit/Miss/Eviction
                recordHME(cur_group, E, tag, blockSize, memAddr, hits, misses,
evictions);//计算出Hit/Miss/Eviction的次数
                recordHME_verbose(&oldHits, &oldMisses, &oldEvictions, hits,
misses, evictions);//详细打印出Hit/Miss/Eviction
            } else if(acctype == 'S' || acctype == 'L'){//S和L指令一次访存
                recordHME(cur_group, E, tag, blockSize, memAddr, hits, misses,
evictions);//计算出Hit/Miss/Eviction的次数
                recordHME_verbose(&oldHits, &oldMisses, &oldEvictions,hits,
misses, evictions);//详细打印出Hit/Miss/Eviction
            } else {printf ("Error");}
            printf("\n");
        }
    } else{
        //当命令行参数没有包含-v时，显示简要信息
    }
    fclose(file);//关闭文件
    free(fileLine);//释放文件行指针
    releaseCache(simCache, groups, E);//释放Cache
    return;}
```

整个流程的实现按照思维导图的过程来实现。其中main函数中主要包括了execute的执行与printsummary的调用。

1.4. 实验结果和分析

- 测试帮助函数

命令：./csim -h

```
root@Huaweiyun:~/Desktop/计算机系统结构实验/cachelab-handout# ./csim -h
Usage: ./csim-ref [-hv] -s <num> -E <num> -b <num> -t <file>
Options:
        -h      Print this help message.
        -v      Optional v flag.
        -s <num>        Number of group index bits.
        -E <num>        Number of lines per group.
        -b <num>        Number of block offgroup bits.
        -t <file>       Trace file.

Examples:
        linux>  ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.file
        linux>  ./csim-ref -v -s 8 -E 2 -b 4 -t traces/yi.file
Segmentation fault (core dumped)
```

- 测试verbose版本模拟cache。

命令./csim -v -s 1 -E 1 -b 1 -t traces/yi2.trace

```
root@Huaweiyun:~/Desktop/计算机系统结构实验/cachelab-handout# ./csim -v -s 1 -E 1 -b 1 -t traces/yi2.trace
 L 0,1 miss
 L 1,1 hit
 L 2,1 miss
 L 3,1 hit
 S 4,1 miss eviction
 L 5,1 hit
 S 6,1 miss eviction
 L 7,1 hit
 S 8,1 miss eviction
 L 9,1 hit
 S a,1 miss eviction
 L b,1 hit
 S c,1 miss eviction
 L d,1 hit
 S e,1 miss eviction
 M f,1 hit hit
hits: 9             misses: 8           evictions: 6
```

- 编写shell脚本自动编译与运行test-csim

```
make clean
make
echo -e "\n********************CS1703-范唯-U201714670********************"
./test-csim
```

```
root@Huaweiyun:~/Desktop/计算机系统结构实验/cachelab-handout# ./exc.sh
rm -rf *.o
rm -f *.tar
rm -f csim
rm -f test-trans tracegen
rm -f trace.all trace.f*
rm -f .csim_results .marker
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c
gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o
gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c
# Generate a handin tar file each time you compile
tar -cvf root-handin.tar  csim.c trans.c
csim.c
trans.c

********************CS1703-范唯-U201714670********************
                        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

TEST_CSIM_RESULTS=27
```

综上所述可以发现已经实现以下功能

命令	功能
-h	显示帮助信息(可选)
-v	显示轨迹信息(可选)
-b	内存块内地址位数
-t	内存访问轨迹文件名
-s	组索引位数
-E	关联度(每组包含的缓存行数)

最终模拟结果正确，圆满完成任务🎉

2. 总结和体会

先说一说我对计算机系统结构的心得体会吧，计算机系统结构就是计算机的机器语言程序员或编译程序编写者所看到的外特性。所谓外特性，就是计算机的概念性结构和功能特性。用一个不恰当的比喻一，比如动物吧，它的"系统结构"是指什么呢? 它的概念性结构和功能特性，就相当于动物的器官组成及其功能特性，如鸡有胃，胃可以消化食物。至于鸡的胃是什么形状的、鸡的胃部由什么组成就不是"系统

结构"研究的问题了。

然后对于Cache的设计，之前的课程一直停留在组成原理中MIPS CPU的流水设计，一直都没有用代码完整的实现过Cache的结构和LRU替换算法。所以计算机系统结构能设计这样一个实验还是十分令人欣喜的。

主存和Cache都分组，主存中一个组内的块数与Cache中的分组数相同，组间采用直接映射，组内采用全相联映射。也就是说，将Cache分成2^S组，每组包含2^E块，主存块存放到哪个组是固定的，至于存到该组哪一块则是灵活的。Line、group、Cache都是以指针的形式进行设计与实现的，整个Cache的实现比较灵活。

而因为长期以python为主要语言，所以在本次实验中用C语言具体实现的时候，有一点不习惯，特别是对文件流和字符串的处理有一点点生疏。但是对于指针的把握加强了不少，也体会到了C家族的强大。

3. 对实验课程的建议

- 希望可以在指导书加入整个实现过程的大致的流程图
- 希望可以介绍一下I、L、S、M的具体差异，这样可以让实验的时候少走很多弯路。
- 希望可以多设置几种替换算法，这样可以帮助同学们更好的理解Cache的工作机制。