cachelab实验日志

step1:

valgrind 安装

安装步骤:

1. tar -jxvf valgrind-3.12.0.tar.bz2
2. cd valgrind-3.12.0
3. ./configure
4. make 5.sudo make install

输入 valgrind –h 显示 valgrind 的参数及提示，说明安装成功

```
lh@lh-VirtualBox:~/valgrind$ valgrind --version
valgrind-3.15.0
```

csim-ref 模拟器使用命令行参数及使用方法

该模拟器有使用的命令行参数有：./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>

-h:打印使用信息的可选帮助标志

-v:显示跟踪信息的可选详细标志

-s:集合索引位数(S=2^s 是集合数)

-E :相关性（每组的行数）

-b :块位数（B = 2^b 是块大小）

-t :重新显示 valgrind 追踪函数的名称

使用方法：

linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace 可以打印出

hits:4 misses:5 evictions:3

linux> ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace 可以打印出详细信息

```
lh@lh-VirtualBox:~/cachelab-handout$ ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:3
lh@lh-VirtualBox:~/cachelab-handout$ ./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
```

认识yi.trace中各项(I,L,S,M)操作含义

“I”表示指令加载，“L”表示数据加载，“S”表示数据存储，M”表示数据修改（即数据存储之后的数据加载）每个“I”前面都没有空格。每个“M” “L”和“S”之前总是有空格。地址字段指定一个 32 位的十六进制存储器地址。大小字段指定操作访问的字节数**各种操作解释：**

①对于‘I’指定地操作，实验说明中提到，我们不需要考虑：意思就是 valgrind 运行地时候第一个指令总是为操作‘I’。

②对于‘L’以及‘S’指定的操作，我们简单地可以认为这两个操作都是对某一个地址寄存器进行访问（读取或者存入数据）

③对于‘M’指定的操作，可以看作是对于同一地址连续进行‘L‘和’S‘操作

```
star@star-VirtualBox:~/cachelab/cachelab-handout$ ./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
```

逐条进行分析：

1)L 10 对于地址0x10 进行访问：0x10=0000…0001 0000，块偏移值为最低四位，故组索引s=1；一开始的时候 cache 是空的，因此第一次访问的时候为 miss。

2)M 20，连续对地址 0x20 进行连续两次访问：0x20=000…0010 0000，组索引 s=2;所以第一次访问的时候没有要的内容，访问结果为 miss；然后cache 从低一级存储器读取第一次访问需要的内容， 第二 次访问的时候有需要的内容且标记位相等，所以第二次访问的时候结果为 hit。

3)L 22 对地址0x22 进行访问：0x22=000…0010 0100，组索引 s=2;由于操作 2 以将该块存入高速缓存且标记位都相等，故结果为 hit。

4)S 18 对地址0x18 进行访问：0x18=000…0001 1000,组索引 s=1；由于之前的操作，该块已存入高速缓存且标记位都为 0，故访问结果为 hit。

5)L 110 对地址0x110 进行访问：0x110=000…0001 0001 0000 故组索引 s=1；操作 4 将该块存入高速缓存了但标记位不相等，故访问结果为 miss,发生一次 eviction。

6)L 210 对地址 0x210 进行访问：0x210=000…0010 0001 0000，故组索引 s=1，这里标记位为 2 跟操作 5 读取新的行的标记位不匹配，故访问结果为 miss,cache 读取新的行，发生一次 eviction。7).M 12对地址0x12进行连续两次访问：0x12=000…0001 0010,组索引 s=1;第一次访问结果为miss，因为在操作 6 的时候发生了一次行替换把该块驱逐了即使标记位相等,cache 重新读取该块，发生一次 eviction，那么第二次肯定为 hit。（跟操作 1 类似的） 故总共：hits=4 ； miss=5；eviction=3；

step2:创建与缓存释放

命令解析函数 get_opt

```
int get_opt(int argc,char *argv[],int *s,int *E,int* b,char *tracefileName,int *isVerbose)
{
    int opt;
    while((opt=getopt(argc,argv,"hvs:E:b:t:"))!=-1){
        switch (opt) {
            case 'h'://打印帮助信息
                printHelpMenu();
                break;
            case 'v'://提示 trace info
                *isVerbose=1;
                break;
            case 's'://读取s
                checkOptarg(optarg);
                *s = atoi(optarg);//将字符拆转化位整型
                break;
            case 'E'://读取E
            checkOptarg(optarg);
                *E = atoi(optarg);
                break;
            case 'b'://读取b
                checkOptarg(optarg);
                *b = atoi(optarg);
                break;
            case 't'://读取t
                checkOptarg(optarg);
                strcpy(tracefileName,optarg);//C中char*不能直接赋值
                break;
            default://输入错误时，输出help信息
                printHelpMenu();
                exit(0);//退出程序
                break;
        }
    }
    return 1;
}
```

put_Set 函数显示各组信息

```
void put_Sets(Sim_Cache* cache)
{
    printf("S=%d,E=%d\n",cache->set_num,cache->line_num);
    int i,j;
    for(i=0;i<cache->set_num;i++)
    {
        printf("set%d: ",i);
        for(j=0;j<cache->line_num;j++) //显示各组信息
        {
            //printf("line%d ",j+1);
            printf("<%d,%d,%x> ",cache->sets[i].line[j].valid,cache->sets[i].line[j].tag,cache->sets[i].line[j].lruNumber);//显示<valid tag lruNumber>
        }
        printf("\n");
    }
}
```

错误输入时输出帮忙信息函数实现

```
void checkOptarg(char* curOptarg)
{
    if(curOptarg[0]=='-')//如果参数curOptarg='-'说明提取错误，退出程序
    {
        printHelpMenu();//打印帮助信息
        exit(0);
    }
}
```

输出错误命令行参数打印帮助信息

```
star@star-VirtualBox:~/cachelab/cachelab-handout$ ./csim -x
./csim: invalid option -- 'x'
The reference simulator takes the following command-line arguments:
Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
-h: Optional help flag that prints usage info
-v: Optional verbose flag that displays trace info
-s <s>: Number of set index bits (S = 2^s is the number of sets)
-E <E>: Associativity (number of lines per set)
-b <b>: Number of block bits (B = 2^b is the block size)
-t <tracefile>: Name of the valgrind trace to replay
star@star-VirtualBox:~/cachelab/cachelab-handout$
```

step3: 替换行方法

LRU 算法实现函数

```
void updateLruNumber(Sim_Cache* cache,int setBits,int index)//更新lru whice cache line to evict
{
    int i;
    for(i=0;i<cache->line_num;i++)
    {
        cache->sets[setBits].line[i].lruNumber--; //这一组所有行lru-1
    }
    cache->sets[setBits].line[index].lruNumber=Maxn;//如果某一组的某一行发生了hit，那么这一行的lru值置为maxn
}
```

不命中时调用更新 cache 的 Lru 运行脚本函数

```
int updateCache(Sim_Cache* cache,int setBits,int tagBit)
{
    int i,index=-1,evic=0;
    for(i=0;i<cache->line_num;i++)
    {
        if(!cache->sets[setBits].line[i].valid)//找到未满的行更新
        {
            index=i;
            break;
        }
    }
    //不存在未满的行
    if(index==-1)
    {
        index=findMinLruNumber(cache,setBits);//找到最小的lru所在行
        evic=1;
    }
    cache->sets[setBits].line[index].valid=1;
    cache->sets[setBits].line[index].tag=tagBit;
    updateLruNumber(cache,setBits,index);
    return evic;//返回是否冲突
}
```

put_Sets 函数显示各组信息

```
void put_Sets(Sim_Cache* cache)
{
    int i,j;
    printf("-display LRU info:----------------------\n");
    for(i=0;i<cache->set_num;i++)
    {
        printf("set%d: ",i);
        for(j=0;j<cache->line_num;j++)
        {
            printf("<%d,%d,%x>  ",cache->sets[i].line[j].valid,cache->sets[i].line[j].tag,cache->sets[i].line[j].lruNumber);
        }
        printf("\n");
    }
}
```

验证 LRU 主函数代码编写与结果分析

```
int main(int argc,char **argv){
    int s,E,b,isVerbose=0;
    char tracefileName[100]/*opt[10]*/;

    //int addr,size;
    misses = hits = evictions =0;

    Sim_Cache cache;

    get_Opt(argc,argv,&s,&E,&b,tracefileName,&isVerbose);
    init_SimCache(s,E,b,&cache);
    //FILE *tracefile = fopen(tracefileName,"r");
    runLru(&cache,0,4);
    put_Cache(&cache);
```

手动输入命令行语句测试，输出自定缓存模型结构每次操作后显示各组信息<valid tag Lru>

1.L 10,1 对 0x10 进行访问，组索引为 1，最开始为空， 故 miss

2.S 10,1 对 0x10 进行访问，组索引为 1，第一步操作已经处理 0x10，故本次为 hit

22,1 对 0x22 进行两次访问，组索引为 2，第一次为空，第二次不空，故第一次 miss，第二次 hit

从测试结果可以看到，若某一组某一行 hit，则 LRU 更新为定义的 MAX，该组其余行 LRU-1.

注意：LRU 初始化为 0，减 1 后为-1 即图中的 0xffffffff

结果验证正确，LRU 算法行为验证正确

```
star@star-VirtualBox:~/cachelab/cachelab-handout$ ./csim -v -s 2 -E 2 -b 4
S=4,E=2
set0: <0,0,0>  <0,0,0>
set1: <0,0,0>  <0,0,0>
set2: <0,0,0>  <0,0,0>
set3: <0,0,0>  <0,0,0>
struct of chache:s-2 e-2 b-4
L 10,1

L 00000010,1  | set 1    tag 0    block 0   miss
display LRU info:----------------------
set0: <0,0,0>  <0,0,0>
set1: <1,0,7fffffff>  <0,0,ffffffff>
set2: <0,0,0>  <0,0,0>
set3: <0,0,0>  <0,0,0>
S 10,1

S 00000010,1  | set 1    tag 0    block 0   hit
display LRU info:----------------------
set0: <0,0,0>  <0,0,0>
set1: <1,0,7fffffff>  <0,0,fffffffe>
set2: <0,0,0>  <0,0,0>
set3: <0,0,0>  <0,0,0>
M 22,1

M 00000022,1  | set 2    tag 0    block 2   miss
display LRU info:----------------------
set0: <0,0,0>  <0,0,0>
set1: <1,0,7fffffff>  <0,0,fffffffe>
set2: <1,0,7fffffff>  <0,0,ffffffff>
set3: <0,0,0>  <0,0,0>
hit
display LRU info:----------------------
set0: <0,0,0>  <0,0,0>
set1: <1,0,7fffffff>  <0,0,fffffffe>
set2: <1,0,7fffffff>  <0,0,ffffffff>
set3: <0,0,0>  <0,0,0>
```

step4:

编写加载数据的 L 命令处理函数代码

```
void loadData(Sim_Cache*cache,int E,int setBits,int tagBit,int verbose)
{
    if(!isMiss(cache,setBits,tagBit))//命中
    {
        hit++;
        if(verbose) printf("hit");
        //LruDisplay(cache);
        return;
    }
    miss++;//未命中
    if(verbose) printf("miss");
    //处理写入，可能会出现冲突
    if(updateCache(cache,setBits,tagBit))//冲突，发生驱逐
    {
        if(verbose) printf("eviction");
        eviction++;
        LruDisplay(cache);
    }
    LruDisplay(cache);
}
```

① getopt 详细模式-v 时，verbose=1，据此判断是否打印出相应结果 miss，hit，eviction.

② 若命中，则 miss++返回，否则 miss++，接着判断是否冲突，是则 eviction++

编写存储数据的 S 命令处理函数和修改数据的 M 命令处理函数

```
void storeData(Sim_Cache* cache,int E,int setBits,int tagBit,int verbose)
{//store直接调用load函数
    loadData(cache,E,setBits,tagBit,verbose);
}
void mmodifyData(Sim_Cache* cache,int E,int setBits,int tagBit,int verbose)
{//Modify是对地址的两次连续访问
    loadData(cache,E,setBits,tagBit,verbose);
    storeData(cache,E,setBits,tagBit,verbose);
}
```

① store 直接调用 load 函数

② Modify 是对地址的两次连续访问

编写获取 trace 脚本操作地址中的组索引与标记位的函数

```
int getSet(int address,int s,int b) //获取组索引
{
    return (address>>b)&(0xffffffff>>(32-s));
}
int getTag(int address,int s,int b) //获取标记位
{
    return (address>>(s+b));
}
int getBlock(int address,int b) //获取块内偏移量
{
    unsigned a=address;
    return (a<<(32-b))>>(32-b);
}
```

① 组索引获取函数 getSet：将 address 逻辑右移 b 位用于移除 block，然后将 0xffffffff 逻辑右移 32-s 位，最后二者位于，取出 address

② 标记位获取函数 getTag：将 address 逻辑右移 s+b 位， 从而去掉 set 和block，得到 tag 位

验证 LRU 主函数代码编写与结果分析

```
int main(int argc,char **argv){
    int s,E,b,isVerbose=0;
    char tracefileName[100],opt[10];

    int addr,size;
    misses = hits = evictions =0;

    Sim_Cache cache;

    get_Opt(argc,argv,&s,&E,&b,tracefileName,&isVerbose);
    init_SimCache(s,E,b,&cache);
    FILE *tracefile = fopen(tracefileName,"r");

    while(fscanf(tracefile,"%s %x,%d",opt,&addr,&size) != EOF){
        if(strcmp(opt,"I")==0) continue;
        int setBits = getSet(addr,s,b);
        int tagBits = getTag(addr,s,b);
        printf("------------------------------------\n");
        printf("setBits:%x  tagBits:%x\n",setBits,tagBits);
        if(isVerbose == 1) printf("%s %x,%d ",opt,addr,size);
        if(strcmp(opt,"S")==0) {
            storeData(&cache,addr,size,setBits,tagBits,isVerbose);
        }
        if(strcmp(opt,"M")==0) {
            modifyData(&cache,addr,size,setBits,tagBits,isVerbose);
        }
        if(strcmp(opt,"L")==0) {
            loadData(&cache,addr,size,setBits,tagBits,isVerbose);
        }
        if(isVerbose == 1) printf("\n");
    }
    printSummary(hits,misses,evictions);
    return 0;
}
```

```
star@star-VirtualBox:~/cachelab/cachelab-handout$ ./csim -v -s 4 -E 1 -b 4 -t traces/yi.trace
struct of chache:s-4 e-1 b-4

L 00000010,1  | set 1    tag 0    block 0    miss
M 00000020,1  | set 2    tag 0    block 0    miss   hit
L 00000022,1  | set 2    tag 0    block 2    hit
S 00000018,1  | set 1    tag 0    block 8    hit
L 00000110,1  | set 1    tag 1    block 0    miss   eviction
L 00000210,1  | set 1    tag 2    block 0    miss   eviction
M 00000012,1  | set 1    tag 0    block 2    miss   eviction   hit
Summary: hits:4 misses:5 evictions:3
```

```
star@star-VirtualBox:~/cachelab/cachelab-handout$ ./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

TEST_CSIM_RESULTS=27
```

验证结果正确，与csim-ref 运行结果完全一致，且通过 test 文件测试获得了和csim-ref 模拟器在所有模型下相同的结果。