一. 安装 valgrind

首先下载实验文件里面的 valgrind-3.12.0.tar 和 cachelab-handout 压缩包进入 ubuntu21.04 解压安装(如果在 windows 下解压会造成文件缺失)。进入压缩包所在文件夹后使用命令 tar xvf valgrind-3.12.0.tar.bz2 进行解压，之后进入该文件夹，先后依次使用./configure;make;sudo make install 三个指令完成安装，初次安装时 gcc 安装出现问题，请教同学得知可以使用直接安装 3.15.0 版本。最终安装版本如下图。

```
ichibanmikan@ichibanmikan:~/CSAPP/ShareForUbuntu/CSex/cachelab-handout/cachelab-
handout$ valgrind --version
valgrind-3.15.0
```

二. csim-ref 模拟器命令行参数及使用方法

使用方法: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>

-h：可选参数，打印使用说明

-v：可选参数，展示追踪信息

-s <s>：组索引的位数

-b <b>：块的位数

-E <E>：结合性（缓存集缓存行数）

-t <tracefile>：用于追迹的 valgrind trace 文件。使用方法如追踪 yi.trace 文件

三. yi.trace 模拟器命令行参数及使用方法

初始时使用./csim-ref -v -s4 -E1 -b4 -t traces/yi.trace 命令追踪 yi.trace 得到结果如下图

```
ichibanmikan@ichibanmikan:~/CSAPP/ShareForUbuntu/CSex/cachelab-handout/cachelab-
handout$ ./csim-ref -v -s4 -E1 -b4 -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
```

以 L 10,1 miss 为例，第一行的 L(M S I)代表操作名称,其中 L 指数据加载(load) I 代表指令加载(instructions load) M 代表数据修改 S 代表数据存储。然后接下来的 10,1 中，10 代表地址，1 代表数据大小。miss 是该指令的命中状态，miss 代表未命中(第一行为冷不命中) hit 代表命中 eviction 代表替换(驱赶)

上述过程的具体分析如下

本实验取 4-7 位为组索引 s，0-3 位为块偏移 b，其余为 tag 值。分析时简化,不必要时仅取二进制串后 8 位

L 10,1 miss: 0x10=0001 0000,s=1,b=0,t=0 初始时 cache 为空,发生冷不命中 miss。

M 20,1 miss: 0x20=0010 0000,s=2,b=0,t=0 第二组无内容,故读取时发生 miss 并从下一层存储器中读入数据,写入时无冲突,故此时命中 hit

L 22,1 hit: 0x22=0010 0010,s=2,b=2,t=0.在上一个操作中在 0x20 处写入数据,该命令读出时第 2 组的 tag=0 的位置有内容，故命中 hit

S 18,1 hit: 0x18=0001 1000,s=1,b=8,t=0 执行此操作时 1 组的 tag=0 行有数据,故命中 hit

L 110,1 miss eviction: 0x110=0001 0001 0000,s=1,b=0,t=1 第一组 tag=1 的行无数据,故未命中 miss,由于模拟缓存只有一行，只能将 1 组中的数据替换发生 eviction

L 210,1 miss eviction: 0x210=0010 0001 0000,s=1,b=0,t=2 第一组 tag=2 的行无数据,故未命中 miss,由于模拟缓存只有一行，只能将 1 组中的数据替换发生 eviction

M 12,1 miss eviction hit: 0x12=0001 0010,s=1,b=2,t=0,上个操作第一组发生了驱逐 eviction，因此未命中，

然后重新读取再次发生 eviction 驱逐原有的块，之后存储也因此命中 hit。

四． 模拟创建与释放缓存

相关函数如下图

定义 get_Opt 函数解析输入的参数，init_SimCache 函数初始化 cache，printHelp 打印帮助信息，putSets 打印每行信息。

最终运行结果和 main 检测结果如下图。打印出了每行信息;输入错误时打印出错误信息和帮助列表

```
int get_Opt(int argc,char **argv,int *s,int *E,int* b,char *tracefileName,int *isVerbose){
    int c;
    while((c= getopt(argc,argv,"hvs:E:b:t:"))!=-1){
        switch(c){
            case 'v':
                *isVerbose=1;
                break;
            case 's':
                checkOptarg(optarg);
                *s =atoi(optarg);
                break;
            case 'E':
                checkOptarg(optarg);
                *E =atoi(optarg);
                break;
            case 'b':
                checkOptarg(optarg);
                *b =atoi(optarg);
                break;
            case 't':
                checkOptarg(optarg);
                strcpy(tracefileName,optarg);
                break;
            case 'h':
            default:
                printHelp();
                exit(0);
        }
    }
    return 1;
}//get_Opt解析输入参数
```

```
void init_SimCache(int s,int E,int b,Sim_Cache *cache){
    if(s<0){
        printf("invald cache sets number!\n");
        exit(0);
    }
    cache->SetNumber=1<<s;
    cache->LineNumber=E;
    cache->sets=(Set *)malloc(cache->SetNumber*sizeof(Set));
    if(!cache->sets){
        printf("Set Memory error!\n");
        exit(0);
    }
    int i,j;
    for(i=0; i<cache->SetNumber; i++){
        cache->sets[i].lines=(Line *)malloc(E*sizeof(Line));
        if(!cache->sets[i].lines){
            printf("Line Memeory error!\n");
            exit(0);
        }
        for(j=0; j<E; j++){
            cache->sets[i].lines[j].valid=0;
            cache->sets[i].lines[j].LruNumber=0;
        }
    }
}//init_SimCache初始化cache
```

```
void printHelp(){
    printf("Z Usage: ./csim-ref [-hv] -s <num> -E <num> -b <num> -t <file>\n");
    printf("Options:\n");
    printf("-h Print this help message.\n");
    printf("-v Optional verbose flag.\n");
    printf("-s <num> Number of set index bits.\n");
    printf("-E <num> Number of lines per set.\n");
    printf("-b <num> Number of block offset bits.\n");
    printf("-t <file> Trace file.\n\n\n");
    printf("Examples:\n");
    printf("linux> ./csim -s 4 -E 1 -b 4 -t traces/yi.trace\n");
    printf("linux> ./csim -v -s 8 -E 2 -b 4 -t traces/yi.trace\n");
}//printHelp帮助打印信息
void checkOptarg(char *curOptarg){
    if(curOptarg[0]=='-') {
        printf("./csim :Missing required command line argument\n");
        printHelp();
        exit(0);
    }
}//checkOptarg检查命令行输入参数合法性
```

```
int free_SimCache(Sim_Cache *cache){
    int i;
    for(i=0; i<cache->SetNumber; i++) {
        free(cache->sets[i].lines);
        cache->sets[i].lines=NULL;
    }
    free(cache->sets);
    cache->sets=NULL;
    return 0;
}//free_SimCache释放内存
```

```
ichibanmikan@ichibanmikan:~/ShareForUbuntu/c
t$ ./csim -v -s4 -E1 -b4 -t traces/yi.trace
set 0: line 0: valid=0, LruNumber=0
set 1: line 0: valid=0, LruNumber=0
set 2: line 0: valid=0, LruNumber=0
set 3: line 0: valid=0, LruNumber=0
set 4: line 0: valid=0, LruNumber=0
set 5: line 0: valid=0, LruNumber=0
set 6: line 0: valid=0, LruNumber=0
set 7: line 0: valid=0, LruNumber=0
set 8: line 0: valid=0, LruNumber=0
set 9: line 0: valid=0, LruNumber=0
set 10: line 0: valid=0, LruNumber=0
set 11: line 0: valid=0, LruNumber=0
set 12: line 0: valid=0, LruNumber=0
set 13: line 0: valid=0, LruNumber=0
set 14: line 0: valid=0, LruNumber=0
set 15: line 0: valid=0, LruNumber=0
```

```
int putSets(Sim_Cache *cache){
    int i,j;
    for(i=0; i<cache->SetNumber; i++){
        for(j=0; j<cache->LineNumber; j++) {
            printf("set %d: line %d: valid=%d, LruNumber=%d\n",i,j,cache->sets[i].lines[j].valid,cache->sets[i].lines[j].LruNumber);
        }
    }
    return 0;
}//putSets显示各组信息
```

```
int main(int argc, char *argv[]) {
    int s, E, b, isVerbose=0;
    char tracefileName[100];
    Sim_Cache cache;
    get_Opt(argc,argv,&s,&E,&b,tracefileName,&isVerbose);
    init_SimCache(s,E,b,&cache);
    putSets(&cache);
```

```
./csim :缺少需要的参数
用法: ./csim [-hv] -s <num> -E <num> -b <num> -t <file>
参数:
-h 打印帮助信息.
-v 显示详细信息.
-s <num> 索引位大小.
-E <num> 缓存集行数.
-b <num> 偏移位大小.
-t <file> Trace文件.

样例:
linux> ./csim -s 4 -E 1 -b 4 -t traces/yi.trace
linux> ./csim -v -s 8 -E 2 -b 4 -t traces/yi.trace
```

五． 替换行方法

实验相关截图如下

runLru 检测 Lru 运行。isMiss 表示如果未命中返回 1，否则返回 0。未命中使用 updateCache 更新高速缓存。此时如果已满就根据 LRU 最近最少使用算法选择牺牲行(LRU 最小)进行替换。

LRU 算法用 updateLruNumber 处理，hit 就将最新使用的缓存置 max 其他-1.

主函数测试时就在 putSets(&cache);上加 runLru(&cache,1,1);即可，最终 main 检测结果如下图所示，显示出了 LruNumber 的正确值。

```
int runLru(Sim_Cache *sim_cache,int setBits,int tagBits) {
    if(isMiss(sim_cache,setBits,tagBits))
        updateCache(sim_cache,setBits,tagBits);
    return 0;
}//lru运行脚本
int isMiss(Sim_Cache *sim_cache,int setBits,int tagBits){
    int i;
    int isMiss=1;
    for(i=0; i<sim_cache->LineNumber; i++){
        if(sim_cache->sets[setBits].lines[i].valid==1 && sim_cache->sets[setBits].lines[i].tag ==tagBits){
            isMiss=0;
            updateLruNumber(sim_cache,setBits,i);
            break;
        }
    }
    return isMiss;
}//isMiss判断是否命中
int updateCache(Sim_Cache *sim_cache,int setBits,int tagBits) {
    int i;
    int isfull=1;
    for(i=0; i<sim_cache->LineNumber; i++){
        if(sim_cache->sets[setBits].lines[i].valid==0) {
            isfull=0;
            break;
        }
    }
    if(isfull==0){
        sim_cache->sets[setBits].lines[i].valid=1;
        sim_cache->sets[setBits].lines[i].tag=tagBits;
        updateLruNumber(sim_cache,setBits,i);
    }//未满，使用新的
    else{
        int evictionIndex=findMinLruNumber(sim_cache,setBits);
        sim_cache->sets[setBits].lines[evictionIndex].valid=1;
        sim_cache->sets[setBits].lines[evictionIndex].tag=tagBits;
        updateLruNumber(sim_cache,setBits,evictionIndex);
    }//满了，使用LRU
    return isfull;//0未满
}//updateCache更新高速缓存数据
```

```
void updateLruNumber(Sim_Cache *sim_cache,int setBits,int hitIndex){
    sim_cache->sets[setBits].lines[hitIndex].LruNumber=MAX_NUM;
    int j;
    for(j=0; j<sim_cache->LineNumber; j++){
        if(j!=hitIndex)
            sim_cache->sets[setBits].lines[j].LruNumber--;
    }
}//updateLruNumber更新LruNumber
```

```
ichibanmikan@ichibanmikan:~/ShareForUbuntu/CS
t$ ./csim -v -s4 -E1 -b4 -t traces/yi.trace
set 0: line 0: valid=0, LruNumber=0
set 1: line 0: valid=1, LruNumber=2147483647
set 2: line 0: valid=0, LruNumber=0
set 3: line 0: valid=0, LruNumber=0
```

六. 处理脚本命令

实验相关截图如下

```
int getSet(int addr,int s,int b){
    addr=addr>>b;//舍去低b位
    int mask = (1<<s)-1;
    return addr &mask;//保留低s位
} //getSet获取地址中的组号
int getTag(int addr,int s,int b){
    int mask=s+b;
    return addr>>mask;
}//获取地址中的标签号
```

```
void loadData(Sim_Cache *sim_cache,int addr,int size,int setBits,int tagBits,int isVerbose){
    if(isMiss(sim_cache,setBits,tagBits)==1){
        misses++;
        if(isVerbose==1)
            printf("miss ");
        if(updateCache(sim_cache,setBits,tagBits)==1){
            evictions++;
            if(isVerbose==1)printf("eviction ");
        } //该组需要牺牲行
    }//没有命中
    else{
        hits++;
        if(isVerbose==1)printf("hit ");
    }
}//L操作(数据加载操作)
void storeData(Sim_Cache *sim_cache,int addr,int size,int setBits,int tagBits,int isVerbose){
    loadData(sim_cache,addr,size,setBits,tagBits,isVerbose);
}//S操作(数据存储操作)
void modifyData(Sim_Cache *sim_cache,int addr,int size,int setBits,int tagBits,int isVerbose){
    loadData(sim_cache,addr,size,setBits,tagBits,isVerbose);
    storeData(sim_cache,addr,size,setBits,tagBits,isVerbose);
}// M操作(数据修改操作)
```

```
while(fscanf(tracefile,"%s %x,%d",opt,&addr,&size)!=EOF){
    if(strcmp(opt,"1")==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);
    free_SimCache(&cache);
```

相关函数操作及主函数代码相关情况如上图所示。

L 加载命令是基础操作。其中 misses 计数未命中的情况，hits 计数命中情况，evictions 计数驱逐情况。isVerbose 为-v 参数的标志位，updateCache 函数判断缓存是否已满。

4-7 位为组索引 s，0-3 位为块偏移 b，其余为 tag 值

main 函数内部完成的读取文件与 lsm 指令分支处理等操作由 while 循环完成。循环判断条件即为 fscanf 读入文件并存储到相应位置，根据 lsm 指令调用不同操作。并使用 printSummary 打印相关信息,free_SimCache 释放内存。

最终使用 make clean,make 指令将 csim.c 文件编译。使用 S4 E1 b4 模型验证(./csim -v -s4 -E1 -b4 traces/yi.trace 指令)可得与 csim-ref 所得结果相同。再使用./test-csim 对所有 trace 进行追踪得结果与 csim-ref 相同，得分为 27.

```
ichibanmikan@ichibanmikan:~/ShareForUbuntu/CSex/cachelab-handout/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
```

```
ichibanmikan@ichibanmikan:~/ShareForUbuntu/CSex/cachelab-handout/cachelab-handout$ ./csim -v -s4 -E1 -b4 -t traces/yi.trace
setBits:1  tagBits:0
L 10,1miss
setBits:2  tagBits:0
M 20,1miss hit
setBits:2  tagBits:0
L 22,1hit
setBits:1  tagBits:0
S 18,1hit
setBits:1  tagBits:1
L 110,1miss eviction
setBits:1  tagBits:2
L 210,1miss eviction
setBits:1  tagBits:0
M 12,1miss eviction hit
hits:4 misses:5 evictions:3
```