

一、学会 valgrind 的安装

开始将给出的实验文件发送到虚拟机，使用 tar xvf 指令解压两个文件压缩包。在 valgrind-3.12.0 内找到 configure 文件，安装 valgrind。

安装后输入 valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l 来查看是否安装成功，发现提示 valgrind 未安装。这就有些麻烦了。于是就按照提示的指令，使用 sudo apt-get install valgrind 来用系统自带的工具进行安装。之后再次检测是否安装成功，出现了应有的指令，安装成功

二、csim-ref 模拟器的使用

-h 是输出帮助信息，-v 是输出详细的运行过程，也就是不仅给出 hit miss eviction 的数，也给出每一行指令产生的行为，-s 后接一个数表示 cache 的组数，-E 接数字表示每一个组里有 E 行，-b 接数字表示每行由一个 2^b 的字节的数据块组成，-t 加文件表示输入的数据文件的路径。

三、认识各项 I、L、S、M 操作含义，并运行在缓存模型上，能逐条分析 trace 文件操作且对 LRU 替换策略，能分析 hit、miss、eviction 结果产生过程

I 为指令加载，I 之前没有空格，并且在处理的时候可以忽略 I。L 表示数据加载，S 表示数据存储，这两个操作都对 cache 进行一次操作，M 表示数据更改，进行一次 L 进行一次 S。下面以 yi.trace 为例说明。

```
ubuntu@ubuntu:~/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
```

给出的地址为 16 进制，m 为 32 位，命令行参数中给定 s 为 4，b 为 4，E 为 1。

在第一行，由于 cache 由于是空的，所以有一次冷不命中。

第二行，进行 M 操作，0x20=0.....0010 0000，第一次访问 miss，但由于第一次将其写入，所以第二次 hit

第三行，进行 L 操作，0x22=0.....0010 0001，由于在第二次操作对这个 cache 进行了写入，所以这次操作 hit

第四行，进行 S 操作，0x18=0.....0001 1000，第一行操作写入了，所以 hit

第五行，进行 L 操作，0x110=0.....0001 0001 0000 根据 4 位的 s 找到组 2，组 2 的有效位为 1，但是标记位却为 0000，与地址的不匹配，导致 miss 并且 eviction

第六行，与第五行类似，找到组 1，标记位不匹配，导致 miss 并且 eviction

第七行，进行 M 操作，0x12=0.....0001 0010，找到组 1，标记位不匹配，导致第一次 miss 并且 eviction，第二次则 hit

四、原型一

get_Opt 函数的编写，调用了 getopt 函数，s，E，b 就是三个参数，tracefileName 应当接受一个字符串，表明读取的文件位置，isVerbose 则控制详细信息的输出。

如果是 h 或其他则输出帮助信息。

```

int get_Opt(int argc, char **argv, int *s, int *E, int *b, char *tracefileName, int *isVerbose) {
    int opt;
    while (-1 != (opt = getopt(argc, argv, "hvs:E:b:t:")) {
        switch(opt) {
            case 'v':
                *isVerbose=1;
                break;
            case 's':
                *s=atoi(optarg);
                break;
            case 'E':
                *E=atoi(optarg);
                break;
            case 'b':
                *b=atoi(optarg);
                break;
            case 't':
                strcpy(tracefileName, optarg);
                break;
            case 'h':
            default:
                printHelpMenu();
                exit(0);
                break;
        }
    }
    return 1;
}

```

检查输入是否合法，不合法则打印输出信息。这里的判断条件是这四个参数都是设置的初始值，也就是在输入时有任意一个没输入，就输出错误信息。

```

void checkOptarg(int s, int E, int b, char *fileName) {
    /*if (curOptarg[0]!='-') {
        printf("/csim-ref: Missing required command line argument\n");
        printHelpMenu();
        exit(0);
    }*/
    if (s==0 || E==0 || b==0 || strlen(fileName)==0) {
        printf("/csim: Missing required command line argument\n");
        printHelpMenu();
        exit(0);
    }
}

```

参考 csim-ref -h 写出的帮助信息，将第一行的 ./csim-ref 改为了 ./csim

```

void printHelpMenu(){
    printf("Usage: ./csim [-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");
    printf("Examples:\n");
    printf("  linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace\n");
    printf("  linux> ./csim-ref -v -s 8 -E 2 -b 4 -t traces/yi.trace\n");
}

```

五、原型二

首先定义给出的三个结构体：分别是一个 cache 模拟器的总体，包含组数，每个组的行数，以及对每个组的定义；在组里定义了行；在行中则定义了有效位，符号位还有 LruNumber 用于 LRU 规则的处理。

```

typedef struct{
    int valld;
    int tag;
    int LruNumber;
}Line;
typedef struct{
    Line *lines;
}Set;
typedef struct{
    int set_num;
    int line_num;
    Set *sets;
}Sim_Cache;

```

接下来就是对内存的分配和释放，依照在 get_opt()中接受的输入给出的 s, E 来对定义的 cache 进行分配，首先设置组数，根据公式 $S=2^s$ ，将 1 左移 s 为得到 S，赋给 set_num，而 line_num 就直接等于 E，有了组数就可以用 malloc 对 cache 模拟器的 sets 进行大小的分配。之后就是对每个组里的行用 malloc 进行分配，并且对每一行分配过后给有效位和

LruNumber 进行初始化为 0。

接下来就是释放的过程，先对内部的行进行释放，再对组进行释放，通过与 malloc 配对的 free 来完成。

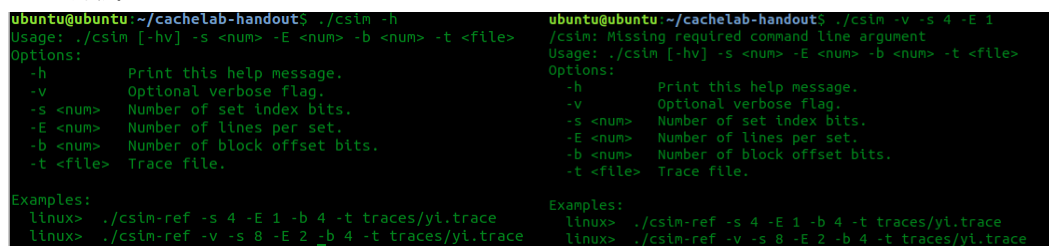
```
void init_SimCache(int s,int E,int b,Sim_Cache *cache) {
    cache->set_num=1<<s;
    cache->line_num=E;
    cache->sets=(Set*) malloc (cache->set_num * sizeof(Set));
    int i,j;
    for (i=0;i<cache->set_num;i++) {
        cache->sets[i].lines=(Line*) malloc (E * sizeof(Line));
        for (j=0;j<cache->line_num;j++) {
            cache->sets[i].lines[j].vaild=0;
            cache->sets[i].lines[j].LruNumber=0;
        }
    }
    return;
}

int free_SimCache(Sim_Cache *cache) {
    int i;
    for (i=0;i<cache->line_num;i++)
        free(cache->sets[i].lines);
    free(cache->sets);
    return 1;
}
```

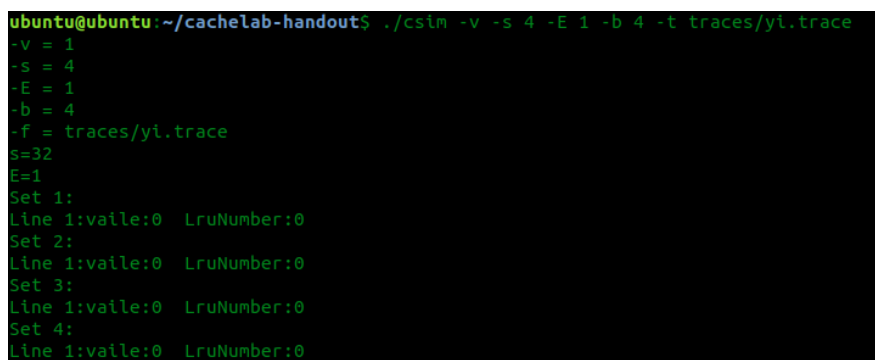
最后就是要进行 putSets 来显示这一个 cache 模拟器的信息，在前两行显示 cache 的组数和行数，接下来的显示方式就是组号：行号：有效位的值和 LruNumber

```
void putSets(Sim_Cache *cache) {
    printf("s=%d\n",cache->set_num);
    printf("E=%d\n",cache->line_num);
    int i,j;
    for (i=0;i<cache->set_num;i++) {
        printf("Set %d:\n",i+1);
        for (j=0;j<cache->line_num;j++) {
            printf("Line %d:",j+1);
            printf("vaile:%d\tLruNumber:%d\n",cache->sets[i].lines[j].vaild,cache->sets[i].lines[j].LruNumber);
        }
    }
    return;
}
```

六、测试



左图是-h 打印帮助信息，右图是在输入不合法时输出错误信息以及帮助信息



这是输出这一个 cache 的信息，其中前五行的输出是为了验证这几个输入的值，接下来就是 cache 信息的显示。