华中科技大学

计算机系统结构实验报告

姓　　名：　胡澳

学　　院：　计算机科学与技术学院

专　　业：　计算机科学与技术

班　　级：　CS1706 班

学　　号：　U201714761

指导教师：　万继光

分数	
教师签名	

2020 年 4 月 13 日

目 录

1 Cache 模拟器实验

1.1 实验目的

理解 Cache 的工作原理，实现一个高效的 Cache 模拟器。

1.2 实验环境

在 macOS 上使用 docker 搭建 Linux 环境完成实验。实验中使用的操作系统、编译器等软件的版本信息如下。

主机操作系统：macOS Catalina 10.15.4

Docker Engine：19.03.8

Docker 镜像：ubuntu:18.04

Ubuntu 版本：Ubuntu 18.04.4 LTS

Linux 内核：Linux 4bdef660d1b7 4.19.76-linuxkit #1 SMP Thu Oct 17 19:31:58 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

编译器：gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

构建工具：GNU Make 4.1

1.3 实验思路

本实验的内容主要分为两部分来完成，即参数的解析和 Cache 模拟器。

命令行参数的解析部分中需要完成的工作就是对用户启动程序时输入的命令行参数的数量、内容与要求的命令行参数格式进行匹配，若参数格式正确，则为 Cache 模拟器需要的组索引位数 s、关联度 E 和内存块内地址位数 b 这三个参数赋值，同时记录输入文件的路径以及是否需要显示轨迹信息，然后调用第二部分的 Cache 模拟器根据上述参数对给定文件中的内存访问轨迹进行 Cache 模拟。若参数格式不正确或包含-h 参数，我们就输出程序的帮助信息。该部分内容主要为字符串处理，与实验的主体内容 Cache 模拟器关系不大，因此不做过多的描述，下面主要针对第二部分，即 Cache 模拟器的实现进行分析。

内存访问轨迹文件中，每行提供了 1 或 2 次内存访问信息，其中包含了访问类型、访问地址以及访问数据长度。我们首先对这三个参数进行分析。

内存访问类型包含了取指令(I)、读(L)、写(S)和修改(M)，其中取指令的数据不进入我们需要模拟的 Cache，因此我们只需要考虑读、写和修改操作。由于我

们的模拟器关心的是 Cache 在访存的过程中是否命中，因此读操作和写操作在我们看来是相同的，均为判断待访问的数据是否在 Cache 中，如果在 Cache 中则表现为命中，如果不在 Cache 中，则需要对 Cache 进行更新。对于修改操作，其本质上是一次读操作和一次写操作的组合，但是与普通的读写不同，在修改操作中，我们执行写操作的地址就是上次读操作的地址，而读操作结束后，该地址对应的内存块必定在 Cache 中，因此写操作是必然会命中 Cache 的，故一次修改操作就等价与一次写操作加上一次必然命中的读操作。由上述分析可知，所有的操作对于 Cache 是否命中而言都可以通过对读操作的分析来实现，因此我们只需要设计一个判断对一个地址的读操作产生的结果的函数，其他的操作都可以通过调用该函数来实现。

对于内存地址，我们需要根据模拟器参数进行如图 1-1 的划分。其中我们需要关心的是 Index 字段和 Tag 字段，Index 字段用于表明该地址对应的内存块应该映射到 Cache 中的组数，Tag 字段则用于在该组中查找具体的 Cache 行。

图 1-1 地址划分

据此，我们可以设计如图 1-2 所示的数据结构来保存模拟器中的 Cache。图中每一个(count, list)结构代表一个 Cache 组，其中 count 用于保存该组中以及包含的有效的 Cache 行数，而 list 则指向一个有这些有效的 Cache 的 Tag 所组成的链表。使用单链表结构则可以让我们在一个组需要执行替换操作时方便的地使用 LRU 来实现 Cache 行的替换。借由此结构，我们就可以跟踪每经过一次操作 Cache 状态的变化，从而确定每次操作时所产生的命中/缺失事件。数据结构的 C 语言定义如下。

图 1-2 Cache 数据结构

```
struct cacheList {
    unsigned long long tag; // 保存内存地址中的 tag 字段
    struct cacheList *next;
};
struct cacheLine {
    int count;                // 保存该 cache 组中已经存在的块数量
    struct cacheList *first; // 保存第一个地址
};
```

由于我们约定一次数据访问不会越过内存块边界，因此访问数据长度对 Cache 模拟器的运行没有影响，我们也就无需关心该参数。

下面，我们来具体实现 Cache 模拟器。模拟器的主要工作流程如图 1-3 所示，模拟器不断执行该流程直到文件读取完毕。其中 cacheLoad 用于判断一个地址的命中情况，并对 Cache 数据进行相应的修改，该函数的具体流程如图 1-4 所示。在该函数中，我们首先根据图 1-2 所示的地址划分方式获取到该地址对应的 index 和 tag 字段的信息，根据得到的 index，我们就可以找到该地址对应的 Cache 组，从而获取到该组中以及包含的行数量以及这些行的 tag 值。我们对这个 tag 链表进行一次遍历，即可得知其中是否包含有当前地址的 tag 值，若包含，则表明访问命中，根据 LRU 算法，我们将该 tag 值对应的节点移动到链表的首部；若没有找到该 tag 值，则表明访问缺失，这时我们需要将这个 tag 值引入到链表中，根据 Cache 组中保存的当前行数量，我们可以判断是否需要发生替换，如果 Cache 组已满，那么我们需要根据 LRU 算法将链表尾部的节点从链表中移除，然后将新的 tag 节点加入到链表的首部，此时发生的事件为缺失和淘汰；如果 Cache 组未满，我们只需要将新的 tag 节点加入到链表的首部，同时更新 Cache 组中的 count 值，此时发生的事件仅为缺失。

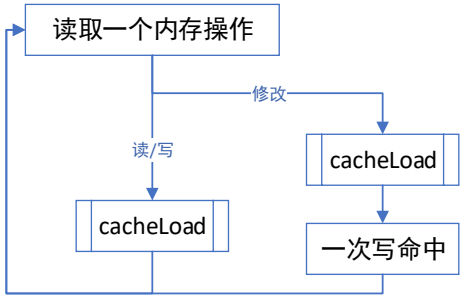

图 1-3 模拟器流程

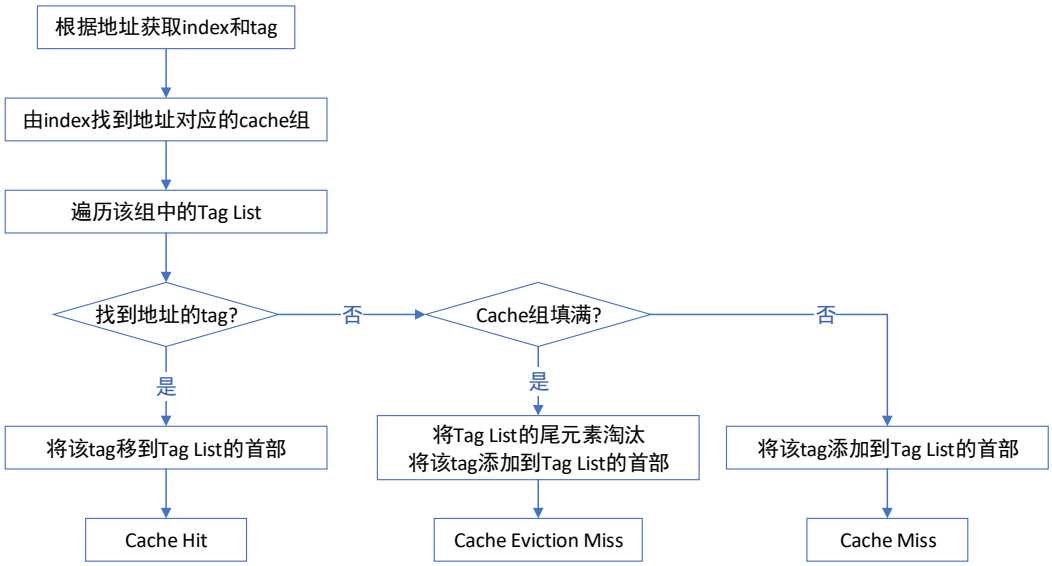

图 1-4 cacheLoad 流程

这样，我们就实现了对一次内存访问的模拟，将上述两个部分结合起来，即可构建出一个完整的 Cache 模拟器。

1.4 实验结果与分析

运行测试程序的结果如图 1-5 所示。由测试结果可知，我们正确的实现了对 Cache 行为的模拟。

```
root@4bdef660d1b7:/csapp/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
```

图 1-5 运行结果

附录 实验代码

程序代码如下。

```
#include "cachelab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct cacheList {
    unsigned long long tag; // 保存内存地址中的 tag 字段
    struct cacheList *next;
};

struct cacheLine {
    int count;               // 保存该 cache 行中已经存在的块数量
    struct cacheList *first; // 保存第一个地址
};

void printHelp();
int setPara(char **argv, int i, int *s, int *E, int *b, char **file);
int cacheLoad(struct cacheLine *cache, char *line, int s, int E, int b,
```

```
              int verbose, int *countHit, int *countMiss, int *countEvictio
n);
int cacheModify(struct cacheLine *cache, char *line, int s, int E, int b,
                int verbose, int *countHit, int *countMiss, int *countEvict
ion);

int main(int argc, char **argv) {
    int verbose = 0;
    int s, E, b;
    char *file;
    // 首先来判断参数
    if (argc < 9 || argc > 10) {
        printHelp();
        return 0;
    }
    if (strcmp(argv[1], "-h") == 0) {
        printHelp();
        return 0;
    }
    verbose = argc - 9;
    if (setPara(argv, argc - 8, &s, &E, &b, &file) != 0) {
        printHelp();
        return 0;
    }

    struct cacheLine *cache =
        (struct cacheLine *)malloc(sizeof(struct cacheLine) * (1 << s));
    memset(cache, 0, sizeof(struct cacheLine) * (1 << s));

    FILE *fp = fopen(file, "r");
    char mod[3];
    char line[256];
    int countHit = 0;
    int countMiss = 0;
    int countEviction = 0;
    fgets(line, 256, fp);
    while (!feof(fp)) {
        line[strlen(line) - 1] = 0;
        sscanf(line, "%s", mod);
        if (strcmp(mod, "I") == 0) {
            fgets(line, 256, fp);
            continue;
        }
        if (strcmp(mod, "L") == 0)
```

```
            cacheLoad(cache, line, s, E, b, verbose, &countHit, &countMiss,
                      &countEviction);
        else if (strcmp(mod, "S") == 0)
            cacheLoad(cache, line, s, E, b, verbose, &countHit, &countMiss,
                      &countEviction);
        else if (strcmp(mod, "M") == 0)
            cacheModify(cache, line, s, E, b, verbose, &countHit, &countMis
s,
                        &countEviction);
        fgets(line, 256, fp);
    }

    printSummary(countHit, countMiss, countEviction);
    return 0;
}

int cacheLoad(struct cacheLine *cache, char *line, int s, int E, int b,
              int verbose, int *countHit, int *countMiss, int *countEvictio
n) {
    unsigned long long addr;
    char mod[3];
    sscanf(line, "%s %llx", mod, &addr);
    unsigned long long tag = addr & ~((1 << (s + b)) - 1);
    unsigned long long index = (addr & ((1 << (s + b)) - 1)) >> b;
    struct cacheList *first = cache[index].first;
    struct cacheList *second = NULL;
    while (first != NULL) {
        if (first->tag == tag) {
            // 命中
            if (second != NULL) {
                // 将 first 的块调整为第一个块
                second->next = first->next;
                first->next = cache[index].first;
                cache[index].first = first;
            }
            (*countHit)++;
            if (verbose == 1)
                printf("%s hit\n", line);
            return 1;
        } else {
            second = first;
            first = first->next;
        }
    }
```

```
    // 如果到这还没有命中的话 那么我们就需要将该页添加到 cache 中了
    (*countMiss)++;
    if (cache[index].count == E) {
        // 说明 cache 满了 需要替换
        first = cache[index].first;
        if (E == 1) {
            first->tag = tag;
        } else {
            while (first->next != NULL) {
                second = first;
                first = first->next;
            }
            second->next = NULL;
            first->tag = tag;
            first->next = cache[index].first;
            cache[index].first = first;
        }
        (*countEviction)++;
        if (verbose == 1)
            printf("%s miss eviction\n", line);
        return 3;
    } else {
        // cache 没满 直接加入
        first = (struct cacheList *)malloc(sizeof(struct cacheList));
        first->next = cache[index].first;
        first->tag = tag;
        cache[index].first = first;
        cache[index].count++;
        if (verbose == 1)
            printf("%s miss\n", line);
        return 2;
    }
}

int cacheModify(struct cacheLine *cache, char *line, int s, int E, int b,
                int verbose, int *countHit, int *countMiss,
                int *countEviction) {
    int load =
        cacheLoad(cache, line, s, E, b, 0, countHit, countMiss, countEvicti
on);
    (*countHit)++;
    switch (load) {
    case 1:
        if (verbose == 1)
```

```
            printf("%s hit hit\n", line);
        break;
    case 2:
        if (verbose == 1)
            printf("%s miss hit\n", line);
        break;
    case 3:
        if (verbose == 1)
            printf("%s miss eviction hit\n", line);
        break;
    default:
        break;
    }
    return 0;
}

int setPara(char **argv, int i, int *s, int *E, int *b, char **file) {
    if (strcmp(argv[i++], "-s") != 0)
        return 1;
    *s = atoi(argv[i++]);
    if (strcmp(argv[i++], "-E") != 0)
        return 1;
    *E = atoi(argv[i++]);
    if (strcmp(argv[i++], "-b") != 0)
        return 1;
    *b = atoi(argv[i++]);
    if (strcmp(argv[i++], "-t") != 0)
        return 1;
    *file = argv[i++];
    return 0;
}

/*
 * 输出帮助信息
 */
void printHelp() {
    printf("Usage: ./csim-ref [-hv] -s <num> -E <num> -b <num> -t <file>");
    printf("Options:");
    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");
```

```
    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");
}
```