

准备阶段

这次实验要在 `csim.c` 文件中编码一个类似 cache 行为的模拟器，`trace` 里是输入测试用例。

构建结构体

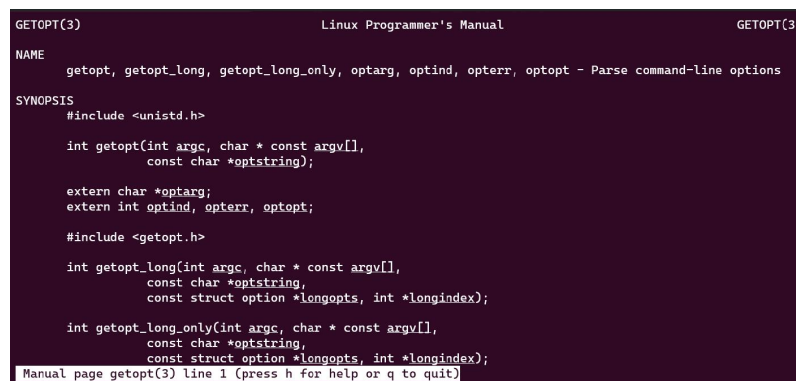
阅读实验任务要求时，发现读取指令格式是 `[space]operation address, size`（四种指令 I、L、S、M，I 不需要处理），并且使用 LRU 的方式替换 cache 存储，所以我计划利用三个结构体，来实现后面可能会用到的操作。首先定义 cache 中的存储行，包括有效位以及标记位；然后定义一个链表的节点，包括前驱、后继以及有效位；最后定义一个链表，包括链表头，以及包含若干行的组。

头文件导入

除了预设的 `#include "cachelab.h"` 之外，首先要 `#include <stdio.h>`、`#include <stdlib.h>`、`#include <string.h>`、`#include <math.h>`。然后根据文件提示接收到在 linux 命令行中输入的 `-v`、`-h`、`-s` 等参数，建议使用 `getopt()` 函数来解析命令行参数，因此还需要再加入一个 `#include <unistd.h>`。

Getopt() 函数

在 Linux 中，`man 3 getopt` 得到这个函数的具体应用：



```
GETOPT(3)                                Linux Programmer's Manual                                GETOPT(3)

NAME
  getopt, getopt_long, getopt_long_only, optarg, optind, opterr, optopt - Parse command-line options

SYNOPSIS
  #include <unistd.h>

  int getopt(int argc, char * const argv[],
             const char *optstring);

  extern char *optarg;
  extern int optind, opterr, optopt;

  #include <getopt.h>

  int getopt_long(int argc, char * const argv[],
                 const char *optstring,
                 const struct option *longopts, int *longindex);

  int getopt_long_only(int argc, char * const argv[],
                     const char *optstring,
                     const struct option *longopts, int *longindex);

Manual page getopt(3) line 1 (press h for help or q to quit)
```

利用到 synopsis 的前四行，并且声明里面的全局变量。

十六进制字符转十进制数字函数

方便后面读取字符串然后把十六进制字符按照 ASCII 码值表进行对应转化为十进制数，可以在 main 中反复调用这个函数，减少主函数的长度。

主函数构建

初始化输出值与 cache 相关参数值

从 pdf 中可以知道，输出 `printSummary()` 中有三个参数 `hit_count`、`miss_count`、`eviction_count` 初始置零；设置一个整数型 `opt` 用于后面接受 `getopt` 函数返回；cache 可能会用到组数 `s`、行数 `E`、块指数 `b` 初始置零；然后为后面解析命令行输入参数设置字符指针 `file_name` 以及文件指针 `file`，另外，为了标记是否需要查看该次输入 `hit`、`miss`、`eviction` 的具体信息，设置一个 `verbose` 初始置零。

解析命令行输入参数

根据实验任务 pdf 的 `usage:.....` 可以得到命令行输入所包含的参数以及信息，也根据 `usage` 设置一个帮助字符串，提示应该输入的字符串信息，也是后面输入 `-h` 之后要输出的信息。然后利用 `while+getopt` 函数进行命令行输入解析，这时套用 `switch` 分支进行判

断，” sEbt: hv” 中只有 -s, -E, -b, -t 是可接受字符连带着后续处理要用的信息，其中，-s, -E, -b 需要读入之后配置 cache 的信息，-t 用来读取文件名，并用于打开文件。-v 和 -h 是不可接收字符，判断是否返回 help 字符串以及 verbose 的依据。

初始化 cache 模拟器进行动态分配与释放、读取文件数据前准备工作

因为组 $S=2^s$, $B=2^b$ ，所以可以利用 1 左移代替 2^s 的幂运算。然后利用结构和动态分配初始化 cache 模拟器，S 组 LRURow，然后初始化链表头以及组行，提前写好对应的释放代码在结尾部分。同时，创建组索引的掩码、可以用来容纳每行数据的字符串、分隔符以及后续暂时存放字符串的字符串。

循环判断每行数据的参数初始化以及字符串分割

利用 strtok 分割每行数据，因为 pdf 中描述每行三个数据中间用 ” , ” 隔开。第一个分割出的是访问指令类型 instruction，假设是 ’ I ’ 不处理；第二个分割出的是地址字符串，这时要用到开头编写好的十六进制字符串转十进制长整数函数，得到对应长整型地址；最后剩下的子串是 size。然后利用地址和组索引掩码相与再右移可以得到对应 cache 的索引，再将地址经过右移可以得到标记。最后，判断是否是 -v 类输入，需要先打印出具体的信息，以及判断 instruction 是否是 M，需要循环两遍。

循环中遍历每行数据的不同情况

先获取 cache 中的每组以及这组的链表头，然后分情况：

(1) 命中 hit：有效且标记相同

如果 cache 中的这一行对应的有效位为 1，并且标记位与文件中的对应判断行相同，则利用 while 循环找到链表命中那一部分，准备放入表头，此时还应该判断是不是命中部分就是表头或者表尾，是的话都跳过，不是就把这部分放入表头，命中数+1，同时判断命令行输入含 -v 的话就打印出对应信息 hit，然后去到循环中的 end 部分。

(2) 无效 miss：写入无效缓冲区

如果这一行的有效位为 0，则准备一个新的节点，并把这个节点的标志位置为 tag，同时，考虑把这一节点放在哪个位置，假如此时链表还为空，没有存在的节点，就把这一节点设置为头节点，否则就把这一节点设置为头节点，miss_count+1，并将这一行的有效位置为 1，标记位置为 tag。仍需要判断是否 v，是的话要打印 miss 然后去 end。

(3) 移除 miss+eviction：既未命中也没有多余内存，链表最后一个移到开头

首先利用 while 循环找到链表最后一个节点，这时候判断这个链表是不是只有这一个节点就是头节点，不是的话把这个节点移到头节点的位置，更新头节点置为这个节点。然后找到缓存中对应这个节点的缓存替换并对这一行进行更新。仍需要判断是否 v，是的话要打印 miss eviction，对 miss_count 以及 eviction_count 都加 1。

循环结尾

end 后是前面三种情况符合情况下的跳转，此时需要判断是否同时满足 verbose==1 以及 TagM!=2，满足就可以打印换行符，然后换行了。

Test-csim 结果截图

```

root@LAPTOP-L9FRK1MI:~/10214602404/cachelab-handout# ./test-csim
Your simulator      Reference simulator
Points (s,E,b) Hits Misses Evicts Hits Misses Evicts
6 (1,1,1) 9 8 6 9 8 6 traces/yi2.trace
6 (4,2,4) 4 5 2 4 5 2 traces/yi.trace
6 (2,1,4) 2 3 1 2 3 1 traces/dave.trace
6 (2,1,3) 167 71 67 167 71 67 traces/trans.trace
6 (2,2,3) 201 37 29 201 37 29 traces/trans.trace
6 (2,4,3) 212 26 10 212 26 10 traces/trans.trace
6 (5,1,5) 231 7 0 231 7 0 traces/trans.trace
8 (5,1,5) 265189 21775 21743 265189 21775 21743 traces/long.trace
50
TEST_CSIM_RESULTS=50
root@LAPTOP-L9FRK1MI:~/10214602404/cachelab-handout#

```

实验收获与体会

通过本次实验,我复习了理论课上有关 cache 的全部内容还有数据结构学过的链表相关的操作,新学了 getopt()、fprintf()、atoi() 等的用法与代码。另外,我通过调试本地的 WSL 启动了 VScode 并复习了第一次上机课的内容重新配置了本地环境,通过反复尝试实现了水杉码园与本地的连接与文件传递。在本次模拟 cache 代码的书写过程中,因为结构体较多,判断情况复杂,所以思路梳理上花费了很多时间,LRU 涉及到链表节点遍历以及替换等操作,我又重新花时间复习了大二的知识,总之,收获很大,通过阅读代码风格的文件,我又重新修改了一些不规范的代码,希望下次自己解决实验的速度能够提升。