

# 中国科学技术大学

# 实验报告



## 计算机系统详解

## Cache Lab

学生姓名： 朱云沁

学生学号： PB20061372

完成时间： 二〇二二年五月二十八日

# 目录

一、 简介	2
1. 实验目的	2
2. 实验要求	2
3. 实验环境	2
二、 实验成果	3
三、 实验过程	3
1. Part A: 工作原理	3
2. Part A: 程序流程	4
3. Part A: 处理命令行参数	5
4. Part A: 分配内存空间	7
5. Part A: 模拟 Cache 访问	7
6. Part B: 第一次尝试	9
7. Part B: 针对 32 x 32 情形优化	9
8. Part B: 针对 64 x 64 情形优化	9
9. Part B: 针对 61 x 67 情形优化	9
四、 总结	9
附录 A 代码清单	10
1. csim.c	10
2. Dockerfile	13

## 一、简介

### 1. 实验目的

- 理解高速缓存 (Cache) 的工作原理, 掌握常见的映射方式和替换策略.
- 学会针对 Cache 对程序代码进行优化.

### 2. 实验要求

- Part A:  
编写一个 Cache 模拟器 `csim`, 支持任意参数的组相连映射 (Set associative), 采用最近最少使用 (Least recently used) 替换策略, 功能应当与给定的参考程序 `csim-ref` 一致.
- Part B:  
优化一个矩阵转置函数, 尽可能减少 Cache misses, 提高 Cache 命中率. 考虑  $32 \times 32$ ,  $64 \times 64$  和  $61 \times 67$  三种情形. 要求遵守给定的编程规则.

### 3. 实验环境

本实验 Part A 在以下环境完成:

---

Machine	MacBook Pro 13"
SoC	Apple M1, 基于 ARM, 含 8 核 CPU、8 核 GPU 及 16GB RAM
OS	macOS Monterey 12.4
IDE	Visual Studio Code 1.68.1
Docker	Docker 20.10.13
Image	Ubuntu 22.04, amd64
Packages	GCC 11.2.0, Make 4.3, Python 2.7.18, Valgrind 3.18.1

---

容器配置参见附录 [A Dockerfile](#).

## 二、实验成果

由于时间不足与设备问题<sup>1</sup>, Part B 尚未完成. Part A 的测试结果如图 1 所示.

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PB20061372 at cac3dcd26711 in /workspaces/Repositories/Computer System/Labs/cache/cacheLab-handout on masterxxx 22-05-27 - 8:04:55
- make
gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c
gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o
gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c
# Generate a handin tar file each time you compile
tar -cvf -handin.tar csim.c trans.c
csim.c
trans.c
PB20061372 at cac3dcd26711 in /workspaces/Repositories/Computer System/Labs/cache/cacheLab-handout on masterxxx 22-05-27 - 8:05:03
- python2 ./driver.py
Part A: Testing cache simulator
Running ./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

Part B: Testing transpose function
Running ./test-trans -M 32 -N 32

```

图 1: 在终端中运行 driver.py, 验证 Part A 的正确性

完整源文件, 参见附录 A csim.c.

## 三、实验过程

### 1. Part A: 工作原理

采用组相连映射方案, Cache 结构如图 2 所示.

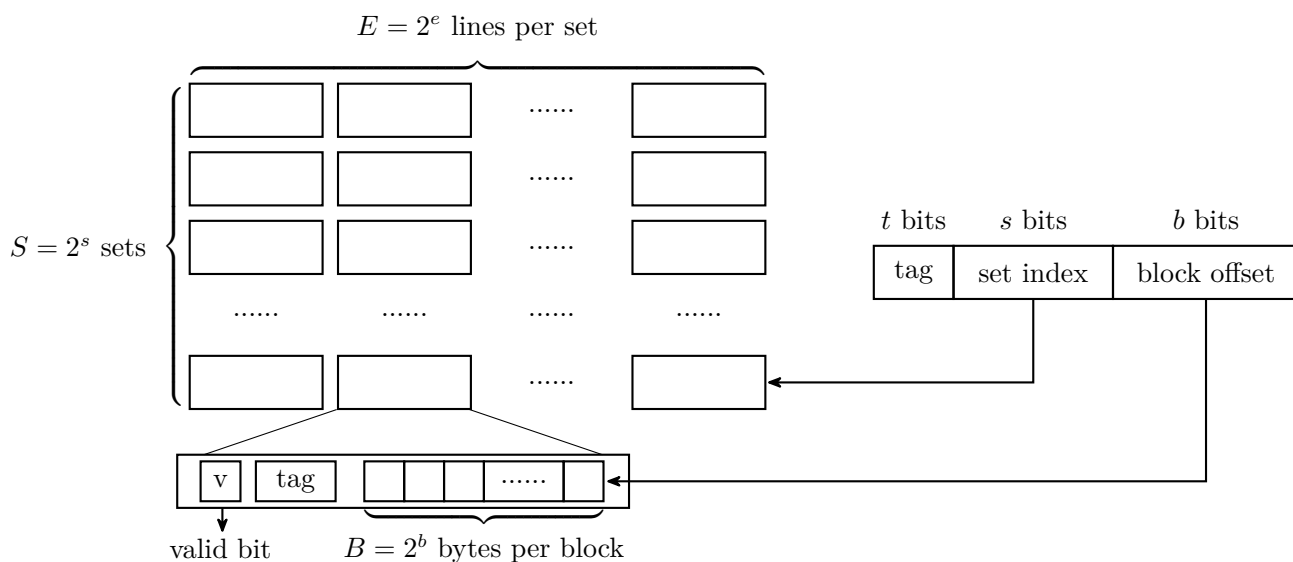


图 2: Cache 的组成结构

Cache 空间被分为  $2^s$  个组 (set), 每个组含有  $E$  个行 (line), 称作  $E$  路组相连. cache line 由一个有效位 (valid bit), 标签 (tag) 以及  $2^b$  个字的数据块 (block) 组成.

<sup>1</sup>在 ARM 机器上使用 QMEU 仿真 amd64 架构, 无法正常运行 valgrind, gdb 等跟踪调试工具.

在 Cache 的硬件实现中, CPU 每次访问内存, 内存地址被划分为 tag, set index, block offset 三部分. 在 set index 对应的组中, 逐个比较有效 cache line 的 tag 与地址中的 tag, 若匹配则 Cache 命中, 根据 block offset 访问数据; 若未命中, 按照某种策略替换 cache line 内容.

通常采用 LRU 替换策略. 为每个 cache line 增设一个计数值, 当一个 cache line 被访问时, 其计数值归零, 其余的有效 cache line 计数值加一. 每次 Cache 未命中而 Cache 已满时, 优先替换计数值最大的 cache line.

## 2. Part A: 程序流程

根据 Cache 工作原理以及实验材料的提示, 设计得程序流程大致如图 3 所示.

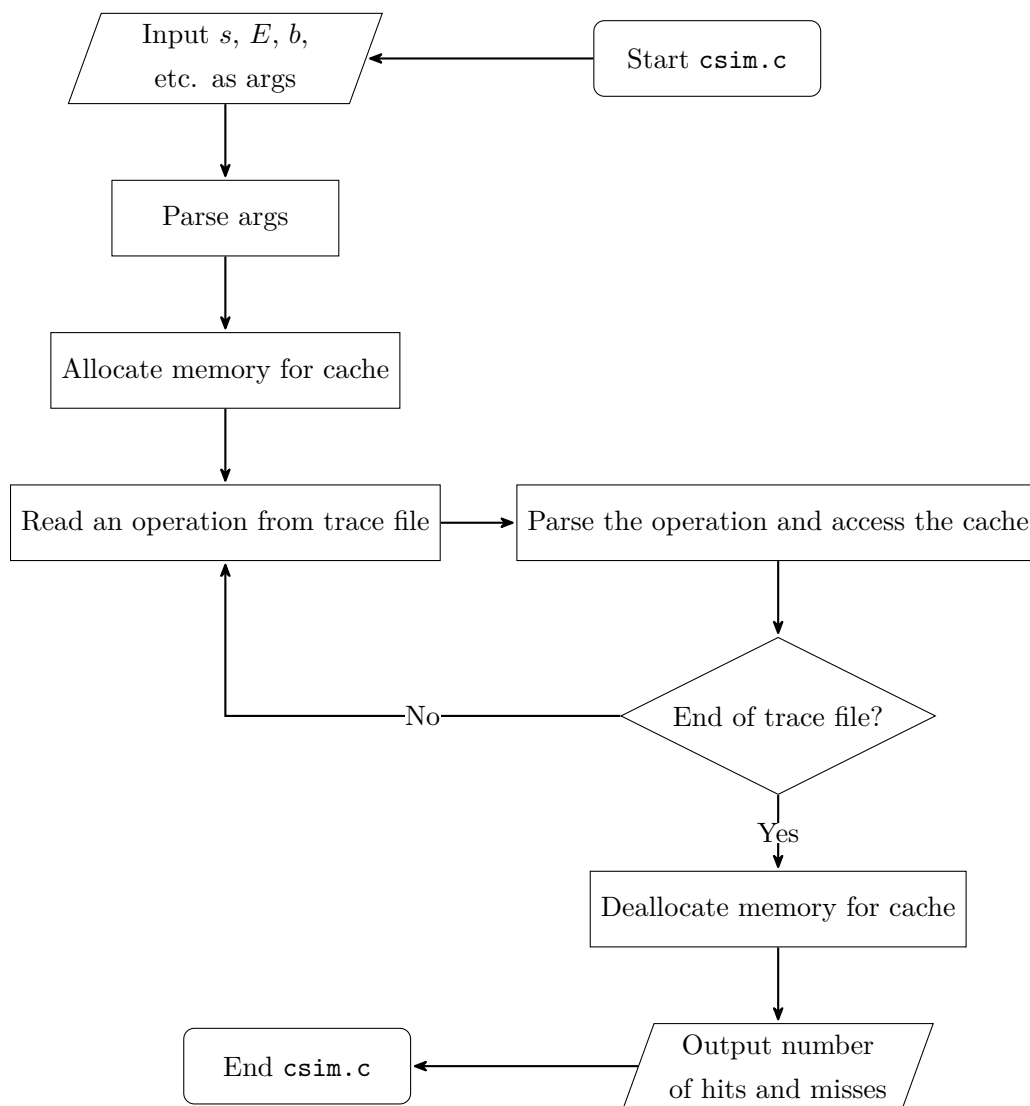


图 3: Cache 模拟程序的流程图

主函数的代码如下:

```
int main(int argc, char *argv[]) {
    parseArgs(argc, argv);
    initCache();
}
```

```

FILE *trace_fp = fopen(trace_file, "r");

char line[32];
while (fgets(line, 32, trace_fp)) {
    /* Ignore instruction cache accesses */
    if (line[0] == 'I')
        continue;

    /* Parse operation */
    char op;
    unsigned int addr;
    int size;
    sscanf(line, " %c %x,%d", &op, &addr, &size);

    accessCache(op, addr, size);
}

fclose(trace_fp);
deinitCache();
printSummary(num_hits, num_misses, num_evictions);
return 0;
}

```

### 3. Part A: 处理命令行参数

首先考虑如何处理命令行参数. 根据参考程序 `csim-ref` 的功能, 并遵循 `test_trans.c` 的编程风格, 不妨用如下 `usage` 函数打印帮助信息:

```

*argv[]) {
    printf("Usage: %s [-hv] -s <num> -E <num> -b <num> -t <file>", argv[0]);
    printf("Options:\n");
    printf("  -h          Print this help message.\n");
    printf("  -v          Optional verbose flag.");
    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("\n");
    printf("Examples:\n");
    printf("  linux> %s -s 4 -E 1 -b 4 -t traces/yi.trace\n", argv[0]);
    printf("  linux> %s -v -s 8 -E 2 -b 4 -t traces/yi.trace\n", argv[0]);
}

```

程序支持 `-h`, `-v` 两个可选项, 分别用于显示帮助信息和设置是否打印详细信息. `-s`, `-E`, `-b` 为必需参数, 已在[Part A: 工作原理](#)中解释. `-t` 也为必需参数, 用于指定跟踪文件.

设置如下全局变量用于存储上述参数:

```
/* Globals set on the command line */
static int s = 0; // Number of set index bits
static int E = 0; // Number of lines per set
static int b = 0; // Number of block offset bits
static int verbose_mode = 0;
static char *trace_file = NULL;
```

根据提示, 可用 `getopt` 库函数解析命令行参数. 遵循 `test_trans.c` 的编程风格, 编写如下 `parseArgs` 函数:

```
void parseArgs(int argc, char *argv[]) {
    char c;

    while ((c = getopt(argc, argv, "hvs:E:b:t:")) != -1) {
        switch (c) {
            case 's':
                s = atoi(optarg);
                break;
            case 'E':
                E = atoi(optarg);
                break;
            case 'b':
                b = atoi(optarg);
                break;
            case 't':
                trace_file = optarg;
                break;
            case 'v':
                verbose_mode = 1;
                break;
            case 'h':
                usage(argv);
                exit(0);
            default:
                usage(argv);
                exit(1);
        }
    }
}
```

#### 4. Part A: 分配内存空间

视命令行参数不同, Cache 大小也不同, 需要动态分配内存. 首先, 给出用于存储 Cache 的结构体:

```
/* Struct for cache lines */
typedef struct cache_line {
    int count; // Number of times this line has been accessed
    int valid;
    unsigned int tag;
} cache_line_t, *cache_set_t, **cache_t;
static cache_t cache;
```

本实验仅要求模拟 Cache 的命中与否, 无需对数据进行操作, 因此 cache\_line 结构体不含数据块. 此外, 为了实现 LRU 替换策略, 增设了一个计数值 count.

每个 set 视为 cache line 数组, Cache 视为 set 数组. 不难编写 initCache 和 deinitCache 函数, 分别用于创建与销毁 Cache.

```
void initCache() {
    int S = 1 << s; // Number of sets
    cache = malloc(sizeof(cache_set_t) * S);
    for (int i = 0; i < S; i++)
        cache[i] = calloc(E, sizeof(cache_line_t));
}

void deinitCache() {
    int S = 1 << s; // Number of sets
    for (int i = 0; i < S; i++)
        free(cache[i]);
    free(cache);
}
```

#### 5. Part A: 模拟 Cache 访问

Cache 的访问和更新, 由以下 accessCache 函数实现:

```
void accessCache(char op, unsigned int addr, int size) {

    /* Calculate tag in address */
    unsigned int tag = addr >> (s + b);
    /* Get cache set by index */
    cache_set_t set = cache[addr >> b & ((1 << s) - 1)];

    /* 'M' operation always results in an extra hit */
```



```

if (op == 'M')
    num_hits++;

if (verbose_mode)
    printf("%c %x,%d ", op, addr, size);

/* Check if there is a hit */
int empty = -1, lru = 0;
for (int i = 0; i < E; i++) {
    if (!set[i].valid)
        empty = i; // Search for an empty line
    else if (set[i].tag == tag) {
        if (verbose_mode)
            printf(op == 'M' ? "hit hit\n" : "hit\n");
        num_hits++;
        set[i].count = 0; // Update time count using LRU policy
        return;
    } else if (++set[i].count >= set[lru].count)
        lru = i; // Search for the LRU line
}

/* If there is no empty line, evict the LRU line */
cache_line_t *line;
if (empty >= 0) {
    if (verbose_mode)
        printf(op == 'M' ? "miss hit\n" : "miss\n");
    line = &set[empty];
} else {
    if (verbose_mode)
        printf(op == 'M' ? "miss eviction hit\n" : "miss eviction\n");
    line = &set[lru];
    num_evictions++;
}
line->valid = 1;
line->tag = tag;
line->count = 0;
num_misses++;
}

```

根据图 2, 不难得到由地址获取 tag 与 set index 的位操作表达式:

$$\begin{aligned} \text{tag} &= \text{addr} \gg (\text{s} + \text{b}) \\ \text{set\_index} &= \text{addr} \gg \text{b} \& ((1 \ll \text{s}) - 1) \end{aligned}$$

随后循环匹配 tag, 与此同时查找空行与最近最少使用的行. 使用三个全局变量 (`num_hits` 等) 记录统计数据, 适时更新. 对于跟踪文件中的数据调整操作 (以 ‘M’ 开头), 本质为先读后写, 应当额外考虑.

至此, 实现了 Cache 模拟器, 测试结果参见图 1.

## 6. Part B: 第一次尝试

To be continued...

## 7. Part B: 针对 $32 \times 32$ 情形优化

To be continued...

## 8. Part B: 针对 $64 \times 64$ 情形优化

To be continued...

## 9. Part B: 针对 $61 \times 67$ 情形优化

To be continued...

## 四、 总结

完成 Cache Lab, 主要有以下收获:

- 深入理解了内存访问的时空局部性, Cache 的原理和意义;
- 掌握了 Cache 的组成结构, 与主存的映射方式;
- 掌握了组相连映射的思想, 能够定量分析映射过程;
- 掌握了 LRU 替换策略的思想, 能够自行通过软件实现;
- 体会了 Cache 对程序性能的影响, 明白了代码优化的重要性;
- 学会了矩阵转置算法中, 针对 Cache 进行优化的几种途径;
- 学会了使用 `getopt` 库函数解析命令行参数;

本实验的所有材料已上传至 GitHub:

<https://github.com/HasiNed/Computer-System>

## 附录 A 代码清单

## 1. csim.c

---

```

1  #include "cachelab.h"
2  #include <getopt.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6
7  /* Globals set on the command line */
8  static int s = 0; // Number of set index bits
9  static int E = 0; // Number of lines per set
10 static int b = 0; // Number of block offset bits
11 static int verbose_mode = 0;
12 static char *trace_file = NULL;
13
14 /* Struct for cache lines */
15 typedef struct cache_line {
16     int count; // Number of times this line has been accessed
17     int valid;
18     unsigned int tag;
19 } cache_line_t, *cache_set_t, **cache_t;
20 static cache_t cache;
21
22 /* Globals for cache statistics */
23 static int num_hits = 0;
24 static int num_misses = 0;
25 static int num_evictions = 0;
26
27 /*
28  * usage - Print usage info
29  */
30 void usage(char *argv[]) {
31     printf("Usage: %s [-hv] -s <num> -E <num> -b <num> -t <file>", argv[0]);
32     printf("Options:\n");
33     printf("  -h          Print this help message.\n");
34     printf("  -v          Optional verbose flag.");
35     printf("  -s <num>   Number of set index bits.\n");
36     printf("  -E <num>   Number of lines per set.\n");
37     printf("  -b <num>   Number of block offset bits.\n");
38     printf("  -t <file>  Trace file.\n");
39     printf("\n");
40     printf("Examples:\n");
41     printf("  linux> %s -s 4 -E 1 -b 4 -t traces/yi.trace\n", argv[0]);
42     printf("  linux> %s -v -s 8 -E 2 -b 4 -t traces/yi.trace\n", argv[0]);
43 }
44
45 /*
46  * parseArgs - Parse the command line arguments

```

```
47  */
48  void parseArgs(int argc, char *argv[]) {
49      char c;
50
51      while ((c = getopt(argc, argv, "hvs:E:b:t:")) != -1) {
52          switch (c) {
53              case 's':
54                  s = atoi(optarg);
55                  break;
56              case 'E':
57                  E = atoi(optarg);
58                  break;
59              case 'b':
60                  b = atoi(optarg);
61                  break;
62              case 't':
63                  trace_file = optarg;
64                  break;
65              case 'v':
66                  verbose_mode = 1;
67                  break;
68              case 'h':
69                  usage(argv);
70                  exit(0);
71              default:
72                  usage(argv);
73                  exit(1);
74          }
75      }
76  }
77
78  /*
79   * initCache - Initialize the cache with s, E and b
80   */
81  void initCache() {
82      int S = 1 << s; // Number of sets
83      cache = malloc(sizeof(cache_set_t) * S);
84      for (int i = 0; i < S; i++)
85          cache[i] = calloc(E, sizeof(cache_line_t));
86  }
87
88  /*
89   * deinitCache - Deinitialize the cache
90   */
91  void deinitCache() {
92      int S = 1 << s; // Number of sets
93      for (int i = 0; i < S; i++)
94          free(cache[i]);
95      free(cache);
96  }
```

```

97
98  /*
99   * accessCache - Simulate to access cache
100  */
101 void accessCache(char op, unsigned int addr, int size) {
102
103     /* Calculate tag in address */
104     unsigned int tag = addr >> (s + b);
105     /* Get cache set by index */
106     cache_set_t set = cache[addr >> b & ((1 << s) - 1)];
107
108     /* 'M' operation always results in an extra hit */
109     if (op == 'M')
110         num_hits++;
111
112     if (verbose_mode)
113         printf("%c %x,%d ", op, addr, size);
114
115     /* Check if there is a hit */
116     int empty = -1, lru = 0;
117     for (int i = 0; i < E; i++) {
118         if (!set[i].valid)
119             empty = i; // Search for an empty line
120         else if (set[i].tag == tag) {
121             if (verbose_mode)
122                 printf(op == 'M' ? "hit hit\n" : "hit\n");
123             num_hits++;
124             set[i].count = 0; // Update time count using LRU policy
125             return;
126         } else if (++set[i].count >= set[lru].count)
127             lru = i; // Search for the LRU line
128     }
129
130     /* If there is no empty line, evict the LRU line */
131     cache_line_t *line;
132     if (empty >= 0) {
133         if (verbose_mode)
134             printf(op == 'M' ? "miss hit\n" : "miss\n");
135         line = &set[empty];
136     } else {
137         if (verbose_mode)
138             printf(op == 'M' ? "miss eviction hit\n" : "miss eviction\n");
139         line = &set[lru];
140         num_evictions++;
141     }
142     line->valid = 1;
143     line->tag = tag;
144     line->count = 0;
145     num_misses++;
146 }

```

```
147
148  /*
149   * main - Main routine
150   */
151  int main(int argc, char *argv[]) {
152      parseArgs(argc, argv);
153      initCache();
154      FILE *trace_fp = fopen(trace_file, "r");
155
156      char line[32];
157      while (fgets(line, 32, trace_fp)) {
158          /* Ignore instruction cache accesses */
159          if (line[0] == 'I')
160              continue;
161
162          /* Parse operation */
163          char op;
164          unsigned int addr;
165          int size;
166          sscanf(line, " %c %x,%d", &op, &addr, &size);
167
168          accessCache(op, addr, size);
169      }
170
171      fclose(trace_fp);
172      deinitCache();
173      printSummary(num_hits, num_misses, num_evictions);
174      return 0;
175  }
```

---

## 2. Dockerfile

---

```
1  # Emulate x86 architecture
2  FROM --platform=linux/x86_64 ubuntu:latest
3
4  # Switch apt source to mirror
5  RUN sed -i "s/archive.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list
6  RUN sed -i "s/security.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list
7
8  # Install packages
9  RUN apt-get update \
10     && DEBIAN_FRONTEND=noninteractive apt-get install -y build-essential sudo git locales zsh vim
11     && perl curl gdb valgrind python2\
12     && apt-get clean -y
13
14  # Generate locale
15  RUN locale-gen --no-purge en_US.UTF-8
```

```
16 # Create a user to show my student ID
17 ARG USERNAME="PB20061372"
18 RUN useradd $USERNAME -m \
19     && echo "$USERNAME ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/$USERNAME \
20     && chmod 0440 /etc/sudoers.d/$USERNAME
21 USER $USERNAME
22
23 # Install oh-my-zsh and set theme to fino-time (my favorite)
24 RUN sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
    ↪ ""
25 RUN sed -i "s/robbyrussell/fino-time/g" ~/.zshrc
```

---