

cache 模拟实验

实验描述

在本章实验当中，我们将要实现一个 cache 的模拟器，这个 cache 的规格如下：

- 1. 索引位数 index： s ；
- 2. 每组行数 association： E （ E 路组相连）；
- 3. 块位数 offset： b （ $B = 2^b$ ，每一块大小为 B 字节）。

本实验 cache 的映射方式为上文提到的**组相连映射**，即模拟的是一个共 2^s 组， E 路组相连，块大小为 2^b 的 cache。在本单元实验当中，你需要模拟 CPU 访问内存的过程，采用 **LRU（最近最少使用）替换策略**，并且输出 cache 命中（hit）、未命中（miss）、被逐出（eviction）的次数。

你的任务是完成 cache 模拟器的主体部分，已经在源代码文件（可从题目附件下载）中标出。

本次 Project 建议在 linux/macOS 操作系统中进行实验，课程组提供的虚拟机即可满足需求，相关环境配置操作见预习章节。

程序运行与输入

从源代码编译模拟器程序后，使用终端运行程序并指定运行参数，命令与参数格式如下：

```
co-eda@co-eda:~/cache$ ./csim -s <s> -E <E> -b <b> -t <filename>
```

- 1. csim：编译得到的模拟器程序名
- 2. -s <s>：设置索引位数；
- 3. -E <E>：每组行数（ E 路组相连）；
- 4. -b ：块位数（ $B = 2^b$ ，每一块大小为 B 字节）；
- 5. -t <filename>：输入的 trace 文件名。

trace 是一种文本文件格式，名为**引用跟踪文件**，是工具 valgrind 的输出。valgrind 工具能够分析程序对内存的访问情况，进而形成 trace 文件。在此，我们使用 trace 文件来**模拟 CPU 的访存行为**。文件的每一行指的是一次或两次的内存访问，其基本格式为： operation address,size 。 * operation 字段表示内存访问的类型： * I 表示指令加载； * L 表示数据加载，若未命中，则记为1次 miss； * S 表示数据存储，若未命中，则需要先将数据加载到 cache 中，同时进行存储操作，记为1次 miss； * M 表示数据修改（即，先是数据加载，然后是数据存储）。 * address 字段指定 **64 位十六进制**内存地址。 * size 字段指定操作访问的字节数。

举个例子，下面是某 trace 文件的一部分：

```
I 0400d7d4,8
M 0421c7f0,4
L 04f6b868,8
S 7ff0005c8,8
```

对这部分解释如下：

- 1. 从 0400d7d4 这个内存地址取指令，指令的长度是 8 个字节；
- 2. 对内存地址为 0421c7f0 的数据进行修改（先加载，再存储，需要访问两次 cache），修改的长度为 4 个字节；
- 3. 对内存地址为 04f6b868 的数据进行加载，加载的长度为 8 字节；
- 4. 对内存地址为 7ff0005c8 的数据进行存储，长度是 8 字节。

 **注意**

valgrind 工具不支持在 windows 操作系统中运行。

在本实验当中，我们需要将 trace 文件作为输入，但是我们只对数据缓存性能感兴趣，因此你的模拟器应**忽略所有指令缓存访问（以 I 开头的行）**。

对于程序的参数解析，我们在下发的代码文件中提供了解析接口，你需要调用函数 parse_cmd 来接收参数。

```
int parse_cmd(int args, char **argv, int *s, int *E, int *b, char *file);
```

对于文件读入，我们也提供了读入接口，你需要调用函数 `readline` 来按行读取 `trace` 文件。

```
int readline(FILE *trace, char *op, unsigned long long *address, int *request_length)
```

程序输出

你**必须**在主函数的末尾调用函数 `printSummary`（该函数在下发代码文件中已经给出），该函数将对模拟 `cache` 的命中、未命中和逐出总数进行输出：

```
void printSummary(hit_count, miss_count, eviction_count);
```

运行和输入样例

```
co-eda@co-eda:~/cache$ ./csim -s 4 -E 1 -b 4 -t traces/yi.trace
```

其中，`traces/yi.trace` 文件的内容如下：

```
L 10,1
M 20,1
L 22,1
S 18,1
L 110,1
L 210,1
M 12,1
```

输出样例

```
hits:4 misses:5 evictions:3
```

样例解释

s 的值为 4， E 的值为 1， b 的值为 4——模拟的是一个直接映射，每一组有一个块，块大小为 16 字节的 `cache`。在执行完 `traces/yi.trace` 文件中的指令之后，命中 4 次，缺失 5 次，逐出 3 次，每一指令的执行的情况如下：

```
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
```

在本地测试过程中，你可以输出类似的中间信息方便自己进行调试。在线评测仅对 `printSummary` 函数的输出进行检查。

提交窗口

Cache 模拟

注意事项

- 你可能需要使用 `malloc` 等函数为模拟 cache 的数据结构分配内存
- 数据保证单次内存访问不会跨越块边界，即地址 `[address, address + size)` 均在一个块内
- 涉及乘方操作（如：获取 2^n 的值）请使用位运算实现，不能使用 `pow` 等数学库函数

提交测试说明


请提交编写完成后的 `csim.c` 文件进行评测。

对于 100% 的测试点：

- 数据范围： $0 < s \leq 4$, $0 < E \leq 5$, $2 \leq b \leq 5$
- 时间限制：1000 ms
- 内存限制：16 MB
- 编译参数：-O2

[CSIM.C](#)

提交 P7X_L0_csim_weak_2024

 点击/拖拽选择文件

 提交

提交记录

[查看提交历史](#) 

ID	957320
提交时间	2024-12-11 10:57:05
评测结果	 

Cache 课下中测

注意事项

- 你可能需要使用 `malloc` 等函数为模拟 cache 的数据结构分配内存
- 数据保证单次内存访问不会跨越块边界，即地址 `[address, address + size)` 均在一个块内
- 涉及乘方操作（如：获取的 2^n 值）请使用位运算实现，不能使用 `pow` 等数学库函数
- 你的 cache 模拟器需要支持 64 位地址的读取

提交测试说明

请提交编写完成后的 `csim.c` 文件进行评测。**请关注与弱测相比，数据范围的变动。**

对于 100% 的测试点：

- 数据范围： $0 \leq s \leq 6, 0 < E \leq 6, 2 \leq b \leq 5$
- 时间限制：`1000` ms
- 内存限制：`16` MB
- 编译参数：`-O2`

提交 P7X_L0_csim_middle_2024

 点击/拖拽选择文件

 提交

提交记录

查看提交历史 

ID	957321
提交时间	2024-12-11 10:57:13
评测结果	<div> </div>