

Part A

12/28

相关文件均被找到并被放在此处，服务器内基本配置也已经就绪

- `make`用以编译, `make clean`用以清除旧文件
-

阅读

- 使用`valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l`来验证正确
- `./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace`给予了参考的可执行二进制文件`csim-ref`
- `./csim-ref -h` to see the help information
- trace文件格式 [操作] [地址] [大小]

i表示指令加载, l表示数据加载, s表示数据存储, m表示数据修改

l前面无空格, LSM前有空格

函数

参考网站

仅列部分

- `atoi(const char * nptr)`

函数说明 `atoi()`会扫描参数`nptr`字符串, 跳过前面的空格字符, 直到遇上数字或正负符号才开始做转换, 转换为int

- `itoa`

整数转换为字符串

- `double atof(const char *nptr)`

字符串转换为double。`atof()`会扫描参数`nptr`字符串, 跳过前面的空格字符, 直到遇上数字或正负符号才开始做转换, 而再遇到非数字或字符串结束时(`\0`)才结束转换, 并将结果返回。

- `long atol(const char *nptr)`

将字符串转换成长整型数

参考网站

- `int getopt(int argc, char * const argv[], const char * optstring);` 头文件`unistd.h`, linux c文件

- opstring是选择字符串

形如a:b:cd:e:, 两个:表示参数可选, 一个表示后面有一个参数, 连续的字母表示这两个参数可以放在一起, 比如-rf。

- oprang时当前argc中的参数, 比如-b 'sad', 程序运行到-b时, 然后参数会被存储在oprang中。

过程

- 总共有如下参数, cache地址tag, set index, block。 然后被读取文件中的op, addr, opblock(没用)。
- cache的结构类似二次数组。
- 本题结构体cache中有三个参数, s, E, b。 E是每个set的行的数量, b表示每个Block的大小, s表示set index bits, 与实际地址的关系如下。每个set含有一个line, 有valid(表示是否可用), tag(地址的高位, 区别不同组映射), lru(处理Lru, 最近最少使用算法)。
- 实际提供的参数位E, b, op, addr, opblock; 要用到的数据为tag, set index, block
- 本处中间的存储在addr(long unsigned int)中, $\text{set index} = (\text{addr} \gg b) \& (s - 1)$; $\text{tag} = \text{addr} \gg (s+b)$; $\text{cacge.s} = 2 \ll s$ 。
- 当l为0时, 直接跳过, 如Ppt所言。
- 然后按照读取的op数据来调用函数

检查

- 运行./compare.sh
- 运行./test-csim
- ./drive.py

Part B

1/02

相关文件均被找到并被放在此处, 服务器内基本配置也已经就绪

- make用以编译, make clean用以清除旧文件

阅读

- 主要任务是编写一个转置矩阵的函数, 在trans.c中的transpose_submit函数。
- 对于int类型的局部变量有限制, 一般不超过12个, 不可以使用递归。
- 不能使用malloc函数

- 后来发现M == 64时不太行，最终Miss超过2000。
- 认为第三种情况下比第二种宽松得多，(矩阵大小类似)在之前的思想下采用暴力求解的方式算出来16符合要求。(2-32中2的倍数)。
- 由网上的思路可以知道，把A中前四个放在B之后，A最后4个先存放在B，减少miss。

A[0][0]			A[0][4]
A[0][1]			A[0][5]
A[0][2]			A[0][6]
A[0][3]			A[0][7]

MISS计算

- 32时A每行2次(对角线一次, 初次读入), B读取每行8次.

检查

- ./compare.sh
- ./test-trans
- ./driver.py