Assignment 7

Due:Monday, May 14 before 11:59PM

Updates:

1. Collaboration Policy

For this assignment, you may work in pairs (2 people). All students (whether working in a pair or not) must individually turn in the assignment. Therefore, 2 copies of the assignment will be turned in for those working as a pair. The grader will choose to grade only one copy of a pair's work, and assign the same score to both students.
If one member of a pair does something unexpected with respect to turning in the assignment (for example: bad copy of the source code, late, or perhaps not turning in portions of the assignment), then *both* of the pair get the same penalty and the same score.
If working in a pair, the names and sections of *both* students must appear as comments at the top of all files that will be turned in. Please read the handin section (section 4) of this specification carefully for details about **one extra file that should be submitted by people working in a pair**.

2. Learning Goals

You will work on developing a small cache simulator. This will make you an expert in basics of cache and strengthen your C programing skills.
Please read the entire assignment specification before starting.

3. Cachelab

Copy the file
`/u/g/e/gerald/public/html/cs354/Spring18/projects/p7/cachelab.tar.gz` to your private working directory. Enter the following commands (without "$>"):

```
$> tar -zxvf cachelab.tar.gz
```

This should create a directory named **cachelab**.

3.1 Reference trace files

The `cachelab/traces` directory contains a collection of reference trace files that we will use to evaluate the correctness of the cache simulator you write in this part. The trace files are generated by a Linux program called *valgrind*. For example, typing

```
$> valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l
```

on the command line runs the executable program **"`ls -l`"**, captures a trace of each of its memory accesses in the order they occur, and prints them on stdout. Valgrind memory traces have the following form:

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

Each line denotes one or two memory accesses. The format of each line is
`[space]operation[space]address,size`

The operation field denotes the type of memory access: "I" denotes an instruction load, "L" a data load, "S" a data store, and "M" a data modify (i.e., a data load followed by a data store).
You should consider only memory accesses to data (L/S/M) and ignore instruction fetch (I) while working on your simulator. The address field specifies a **64-bit** hexadecimal memory address. The size field specifies the number of bytes accessed by the operation. In this course we have dealt with 32-bit computers. Only for this assignment, we use 64-bit computer traces - all the addresses for accessing memory are 64-bit addresses.

NOTE: You will NOT have to generate any traces on your own. You need to work only with the traces provided to you.

3.2 Writing the cache simulator *csim*

You will write a cache simulator in **csim.c** that takes a valgrind memory trace as input, simulates the hit/miss/eviction behavior of a cache memory on this trace, and **outputs the total number of hits, misses and evictions.**
We have provided you with the binary executable of a reference cache simulator, called **csim-ref**, that simulates the behavior of a cache with arbitrary size and associativity on a valgrind trace

file. It uses the **LRU (least-recently used) replacement policy** when choosing which cache line to evict. You can use csim-ref to compare your implementation.

The reference simulator takes the following command-line arguments:
Usage: `./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>`

`-h:` Optional help flag that prints usage info
`-v:` Optional verbose flag that displays trace info
`-s <s>:` Number of set index bits (`S = 2^s` is the number of sets)
`-E <E>:` Associativity (number of cache lines per set)
`-b <b>:` Number of block bits (`B = 2^b` is the block size)
`-t <tracefile>:` Name of the valgrind trace to replay

The command-line arguments are based on the notation (s, E, and b) from pages 597 and 598 of the CS:APP2e textbook.

For example, enter the following command:

```
$> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:3
```

The same example in **verbose** mode:

```
$> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace -v
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
```

You can use this verbose output from csim-ref while debugging your code (csim.c). The "hit"/"miss"/"eviction" in the verbose output above indicates if that particular memory access (L/S/M) specified by the trace file led to hit/miss/eviction in cache.

We have provided you with skeleton code in file csim.c. Your job is to complete the implementation so that it outputs the correct number of hits, misses and evictions. **You NEED NOT support the verbose output (using the -v option) as mentioned above.**

Once you have made changes to file csim.c and want to test your implementation, do the following:

```
$> make clean
$> make
$> ./csim -s 4 -E 1 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:3
```

The **SAMPLE OUTPUT** should look like as shown below:

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

NOTE: The number of hits, misses and evictions will vary with different cache configurations and different traces. However, the output should be single line similar to what is shown above.

YOU SHOULD NOT PRINT ANYTHING EXTRA IN THE OUTPUT.

Your simulator must work correctly for arbitrary values of `s, E, and b`.

Some important points regarding the implementation in csim.c:

You should carefully read the skeleton code in csim.c. Starting from main() function, the flow is described below in brief:

- Parse the command line arguments. This is already done for you.
- **`void initCache() -`** This function should allocate the data structures to hold information about the sets and cache lines using malloc depending on the values of parameters S (S = 2^s) and E. You need to complete this function.
- **`void replayTrace(char* trace_fn)`** This function parses the input trace file. This part is already done for you. It should call the **`accessData()`** function. You need to complete the missing code in this function.
- **`void accessData(mem_addr_t addr)`** This function is the core of implementation which should use the data structures that were allocated in `initCache()` function to model the cache hits, misses and evictions. You need to complete this function. The most crucial thing is to update the global variables **`hit_count, miss_count, eviction_count`** inside this function appropriately. You should implement Least-Recently-Used (LRU) cache replacement policy.
- **`void freeCache()`** This function should free up any memory you allocated using `malloc()` in `initCache()` function. This is crucial to avoid memory leaks in the code. You need to complete this function.
- **`printSummary(hit_count, miss_count, eviction_count)`** This function prints the statistics in the desired format. This is already implemented for you.

You **MUST READ COMMENTS in the file csim.c** which provide additional details.
To check your implementation in csim.c, you can run the following command:

```
$> ./test-csim
```

It will tell you how your implementation in csim.c compares against csim-ref.

4. Handing in the assignment

Copy csim.c into your handin directory. Your handin directory for this project is
/p/course/cs354-gerald/public/handin/login/p7
where login is your cs-login.
If you are working as part of a pair, you must turn in an extra file. This file will contain the names and email of **both** students in the pair. As an example, if Gerald worked with Ancy on this assignment, the contents of the extra file for both Gerald and Ancy would be
Adalbert Gerald Soosai Raj, gerald@cs.wisc.edu
Ancy Philip, aphilip@cs.wisc.edu
The **name** of this file is specialized to help the 354 automated grading tools identify who worked together. This file name is composed of the CS logins of the partners separated by a period. The file name is of the form <login1>.<login2>. Gerald's CS login is gerald, and Ancy's CS login is aphilip. The file name that both use will be **`gerald.aphilip`**. Please have both partners use the same file name. It does not matter which partner's name is first within this file name.

Requirements

1. Include a comment at the top of the source code of all programs with your *name* and *section* (and your partner's name and section, if working in a pair).
2. Use the instructional Linux machines for this assignment! A penalty will be imposed for any program that was obviously edited on a Windows machine and transferred to the Unix machines for turning in. It is annoying to see Windows line endings (^M) at the end of every line. Points will be taken off for such mistakes.
3. Your programs must compile on an instructional Linux machine as indicated in this specification *without warnings or errors*.