CSC 374 Computer Systems II
Karen Heart, Instructor
Cache Lab: Understanding Cache Memories

1 Logistics

This is an individual project, although you may certainly discuss the concepts of the project with others.

2 Overview

This lab will help you understand the impact that cache memories can have on the performance of your C programs. You will write a small C program (about 200-300 lines) that simulates the behavior of a cache memory. Your code will be tested under various circumstances and compared to a reference model.

3 Downloading the assignment

The handout for this assignment is named `cachelab-handout.tar`; it is on D2L.

Start by copying `cachelab-handout.tar` to a protected Linux directory in which you plan to do your work. Then give the command

```
linux> tar xvf cachelab-handout.tar
```

This will create a directory called `cachelab-handout` that contains a number of files. You will be modifying the file `csim.c`. To compile this file, type:

```
linux> make clean
linux> make
```

WARNING: Do not let the Windows WinZip program open up your `.tar` file (many Web browsers are set to do this automatically). Instead, save the file to your Linux directory and use the Linux `tar` program to extract the files. In general, for this class you should NEVER use any platform other than Linux to modify your files. Doing so can cause loss of data (and important work!).

4 Description

Your task is to implement a cache simulator.

4.1 Reference Trace Files

The `traces` subdirectory of the handout directory contains a collection of *reference trace files* that we will use to evaluate the correctness of the cache simulator you write. The trace files were generated by a Linux program called `valgrind`. `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 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). There is never a space before each “I”. There is always a space before each “M”, “L”, and “S”. The *address* field specifies a 64-bit *hexadecimal* memory address. The *size* field specifies the number of bytes accessed by the operation.

4.2 Writing a Cache Simulator

You will write a cache simulator in `csim.c` that takes a `valgrind` memory trace as input, simulates the hit/miss 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.

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 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 page 597 of the CS:APP2e textbook. For example:

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

The same example in verbose mode:

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

Your job is to fill in the `csim.c` file so that it takes the same command line arguments and produces the identical output as the reference simulator. The code to parse the command line has been supplied, and comments indicate the nature of the remaining tasks. You will have to complete the remainder of the needed code.

Programming Rules

- Include your name in the header comment for `csim.c`.
- Your `csim.c` file must compile without warnings in order to receive credit.
- Your simulator must work correctly for arbitrary s, E, and b. This means that you will need to allocate storage for your simulator's data structures using the `malloc` function. Type "man malloc" for information about this function.
- For this lab, we are interested only in data cache performance, so your simulator should ignore all instruction cache accesses (lines starting with "I"). Recall that `valgrind` always puts "I" in the first column (with no preceding space), and "M", "L", and "S" in the second column (with a preceding space). This may help you parse the trace.
- In order to receive credit, you must call the function `printSummary`, with the total number of hits, misses, and evictions, at the end of your `main` function:

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

- For this this lab, you should assume that memory accesses are aligned properly, such that a single memory access never crosses block boundaries. By making this assumption, you can ignore the request sizes in the `valgrind` traces.

5 Evaluation

5.1 Evaluation for Cache Simulator

The correctness of your code is determined automatically. You must run your cache simulator using different cache parameters and traces. There are eight test cases, each worth 3 points, except for the last case, which is worth 6 points:

```
  linux> ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
  linux> ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
  linux> ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
  linux> ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
  linux> ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
  linux> ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
  linux> ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
  linux> ./csim -s 5 -E 1 -b 5 -t traces/long.trace
```

You can use the reference simulator `csim-ref` to obtain the correct answer for each of these test cases. During debugging, use the `-v` option for a detailed record of each hit and miss.

For each test case, outputting the correct number of cache hits, misses and evictions will give you full credit for that test case. Each of your reported number of hits, misses and evictions is worth 1/3 of the credit for that test case. That is, if a particular test case is worth 3 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 2 points. If your code is entirely correct, it will be awarded 27 points.

5.2 Grade

Your grade will be calculated as follows:

```
ceiling[(points/27)*5]
```

6 Working on the Lab

We have provided you with an autograding program, called `test-csim`, that tests the correctness of your cache simulator on the reference traces. Be sure to compile your simulator before running the test:

```
linux> make
linux> ./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
```

For each test, it shows the number of points you earned, the cache parameters, the input trace file, and a comparison of the results from your simulator and the reference simulator.

Here are some hints and suggestions for working on the Cache Simulator:

- Structure your code well:
  1. You should begin by inputting the command line variables and processing them.
  2. You will then allocate a cache and clear all values therein.
  3. Your code should read the trace file one line at a time and determine whether the data address exists in the cache. If the data is not yet in the cache, add it. If an empty line is not present in the set where the data is to be placed, select a line that satisfies the LRU (Least Recently Used) algorithm.
  4. Store whether the data was found in the cache (hit) or added to it (miss).
  5. Also note whether an existing line of data was overwritten (eviction).
  6. Keep in mind that a Modify instruction involves two accesses of the same data; therefore, the second access will always be a hit.
- In order to implement the LRU algorithm, consider that some type of data must be maintained that tracks the last time a cache line was accessed. The time value from the system time() function is measured in seconds; therefore, it is too imprecise for this task. Although you may try another system measure, you may also define your own "clock", which you can easily implement with an `int` variable. Keep in mind that you must record every access of a cache line in order to implement the LRU algorithm correctly.
- Do your initial debugging on the small traces, such as `traces/dave.trace`.
- The reference simulator takes an optional `-v` argument that enables verbose output, displaying the hits, misses, and evictions that occur as a result of each memory access. You are not required to implement this feature in your `csim.c` code, but we strongly recommend that you do so. It will help you debug by allowing you to directly compare the behavior of your simulator with the reference simulator on the reference trace files.
- Each data load (L) or store (S) operation can cause at most one cache miss. The data modify operation (M) is treated as a load followed by a store to the same address. Thus, an M operation can result in two cache hits, or a miss and a hit plus a possible eviction.

7 Technical Support

If you have questions or get stuck, you may upload **ONLY** your `csim.c` file to your Assistance Submission folder on D2L and email me a note to take a look at it.

8 Handing in Your Work

Upload **ONLY** your `csim.c` file to the cachelab Submission folder on D2L.

IMPORTANT: Do not submit your file in any other format, such as a `.zip`, `.gzip`, or `.tar` file.