1 Logistics

This is an individual project. This lab must be completed on the machines in the Asprey Lab or the machines in SP 309. All submissions are done using the submit224 program. Any clarifications and revisions to the assignment will be posted to the class website.

2 Overview

This lab will help you understand the impact that cache memories can have on the performance of your C programs.

For this lab you will complete a small C program that simulates the behavior of a cache memory.

3 Downloading the assignment

1. Start by downing the lab starter code from the class website to a directory in which you plan to do your work.

2. Then give the command: tar xvf cachelab.tar. This will cause a number of files to be unpacked in the directory cachelab.

3. You will be doing all of your work inside this directory. You will be modifying the file: csim.c. To compile this file, type:

   ```
   linux> make
   ```

4 Description

For this lab you will implement a cache simulator.
4.1 Reference Trace Files

The traces subdirectory of the cachelab 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 are generated by a Linux program called valgrind. For example, typing

```
linux> 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 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`: Help flag that prints usage info
- `-v`: 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 617 of the CS:APP3e textbook. For example:

```
linux> ./csim-ref -s 2 -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 2 -E 1 -b 4 -t traces/yi.trace
L 10,1
Hits: 0, Misses: 1, Evictions: 0
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:0 LRU:1
S:2 L:0 V:0 T:0 LRU:0
S:3 L:0 V:0 T:0 LRU:0
M 20,1
Hits: 1, Misses: 2, Evictions: 0
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:0 LRU:1
S:2 L:0 V:1 T:0 LRU:2
S:3 L:0 V:0 T:0 LRU:0
L 22,1
Hits: 2, Misses: 2, Evictions: 0
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:0 LRU:1
S:2 L:0 V:1 T:0 LRU:3
S:3 L:0 V:0 T:0 LRU:0
S 18,1
Hits: 3, Misses: 2, Evictions: 0
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:0 LRU:4
S:2 L:0 V:1 T:0 LRU:3
S:3 L:0 V:0 T:0 LRU:0
L 110,1
Hits: 3, Misses: 3, Evictions: 1
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:4 LRU:5
S:2 L:0 V:1 T:0 LRU:3
S:3 L:0 V:0 T:0 LRU:0
```
L 210,1
Hits: 3, Misses: 4, Evictions: 2
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:8 LRU:6
S:2 L:0 V:1 T:0 LRU:3
S:3 L:0 V:0 T:0 LRU:0
M 12,1
Hits: 4, Misses: 5, Evictions: 3
S:0 L:0 V:0 T:0 LRU:0
S:1 L:0 V:1 T:0 LRU:7
S:2 L:0 V:1 T:0 LRU:3
S:3 L:0 V:0 T:0 LRU:0

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.

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 storage for your simulator’s data structures uses the malloc function. Type “man malloc” for information about this function.
- When placing a line in the cache, if there is an empty line in the cache for a set, you should place it in the first free line. This is what the reference simulator does, and it is helpful to match this behavior so you can compare your cache to the reference simulator.
- 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.
- To receive credit for this lab, 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 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

This section describes how your work will be evaluated. The full score for this lab is 100 points:

- Simulator Correctness: 81 Points
- printCache function implementation: 10 Points
- Style: 9 Points

5.1 Evaluation for Correctness

We will run your cache simulator using different cache parameters and traces. There are eight test cases, each worth 9 points, except for the last case, which is worth 18 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 9 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 6 points.

5.2 Implementation of printCache

In addition to correctness, there are 10 for implementing a printCache function. This function prints out the entire contents of your cache. This can be very helpful when you are trying to debug your code. It is invoked when you run the simulator with the `-v` option.

5.3 Evaluation for Style

There are 9 points for coding style. These will be assigned manually by me. I am looking for code that is well documented and easy to understand.
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
```

<table>
<thead>
<tr>
<th>Points (s,E,b)</th>
<th>Your simulator</th>
<th>Reference simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hits</td>
<td>Misses</td>
</tr>
<tr>
<td>9 (1,1,1)</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>9 (4,2,4)</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>9 (2,1,4)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9 (2,1,3)</td>
<td>167</td>
<td>71</td>
</tr>
<tr>
<td>9 (2,2,3)</td>
<td>201</td>
<td>37</td>
</tr>
<tr>
<td>9 (2,4,3)</td>
<td>212</td>
<td>26</td>
</tr>
<tr>
<td>9 (5,1,5)</td>
<td>231</td>
<td>7</td>
</tr>
<tr>
<td>18 (5,1,5)</td>
<td>265189</td>
<td>21775</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td></td>
</tr>
</tbody>
</table>

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 lab:

- 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 as well as printing out the contents of the cache. Implement this feature first! 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.
- Since you don’t know the size of your cache at compile time (as it depends on what options the user runs the program with) you must dynamically allocate any data structures you are using with the `malloc()` library call. We have implemented this for you, but go over and make sure you understand how this code works.

7 Handing in Your Work

To hand in your work, type `make submit` in the `cachelab` directory.