CSE 238/2038/2138 Systems Programming
Project 3
Cache Lab: Understanding Cache Memories
Due: 17.05.2017 17:00 PM

1 Logistics

This is an individual project. You must run this lab on a 64-bit x86-64 machine.

2 Downloading the assignment

You can download the handout for the project through the autolab: `autolab.cse.eng.marmara.edu.tr`

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!).

3 Description

In this project, you will write a cache simulator in `csim.c` that takes a 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.

Your simulator will also take the following command-line arguments:

```
Usage: ./your_simulator -s <s> -E <E> -b <b> -t <tracefile>
```

- `-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 trace file (see Reference Trace Files part below)

The command-line arguments are based on the notation (s, E, and b) from page 652 of the CS:APP3e textbook.

For example, if you want to simulate a direct-mapped cache (E=1) of 16 sets (s=4) and 16 blocks (b=4) in each set, and see the results for the trace file `cachelab-handout/traces/yi.trace`, you will run your program with the arguments given in the following example. Also, the results should be in the given format.

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

The details for the `traces/yi.trace` file are given below. You can also include these details in your program.

```
     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 for Part A is to fill in the `csim.c` file so that it takes the given command line arguments and produces the identical output. Notice that this file is almost completely empty. You'll need to write it from scratch.

Programming Rules for Part A

- Include your name and ID 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 in the trace files, "I" is always 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 Part A, 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 traces.

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 in Part A. The memory trace files 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.

5 Working on the Project

Here are some hints and suggestions for working on the project:

- Do your initial debugging on the small traces, such as `traces/dave.trace`.
- We recommend that you use the `getopt` function to parse your command line arguments. You'll need the following header files:

  ```
  #include <getopt.h>
  #include <stdlib.h>
  #include <unistd.h>
  ```

  See "`man 3 getopt`" for details.
- 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.

6 Handing in Your Work

Each time you type `make` in the `cachelab-handout` directory, the Makefile creates a tarball, called `userid-handin.tar`, that contains your current `csim.c` file.

You will submit your handin through autolab.

IMPORTANT: Do not create the handin tarball on a Windows or Mac machine, and do not handin files in any other archive format, such as `.zip`, `.gzip`, or `.tgz` files.