Cache Lab

Description

In this lab, will you will write a small cache simulator that simulates the behavior of a cache memory. The input to that cache simulator is a cache configuration and a trace file; and the output is a set of statistics:
hits: how many cache hits
misses: how many cache misses
evictions: how many blocks have been kicked out of the cache. You need to implement Least Recently Used (LRU) replacement policy. (Hint: Do not forget the valid bit!)

The trace files (provided) contains memory accesses like the following example:

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.
You cache simulator will take care of "M", "L", and "S" but neglects the "I" instructions.

How will you work on this lab?

1. Download the file cachelab.tar
2. In a terminal, in virtualbox, type: *tar -xvf cachelab.tar*
3. A directory with the name "cachelab" will be crested.
4. When you go to this directory you will find several trace files, a file called Makefile, a file called cachelab.c, and a file called cache-ref. Here is the description of each file:
    a. Makefile: don't worry about it! But do not erase/modify it.
    b. Files *.trace contains the memory accesses of different programs with the format explained above.
    c. cache-ref is the executable of a correct cache simulator. Your program must produce the same statistics. "cache-ref" has an extra switch, "-v", that allows use to see for each memory access whether it is hit/miss. In your program, this is not needed. It is provided in the reference program only for your convenience.
    d. cachelab.c: This is the main file that you need to plug your code in. Please read the comments in that file before you add/modify anything.

5. You need to add your code the file cachelab.c. To compile the program after you are done type: *make cachesim*
6. This will produce an executable called cachesim. Its statistics must be similar to the ones generated by cache-ref for the same trace file. If you want to see the format of the command line, just type: *./cachesim* or type: *./cache-ref*

The command line looks like this:
./cachesim -s <num> -E <num> -b <num> -t <file>
Options:
-s <num> Number of set index bits
-E <num> Number of blocks per set (i.e. associativity).
-b <num> Number of block offset bits.
-t <file> Trace file.

So

./cachesim -s 10 -E 8 -b 6 -t ls.trace

means this cache has $2^{10} = 1024$ sets and each set has 8 blocks (i.e. 8-way set associative) and the block size is $2^6 = 64$ bytes. The simulator will produce hits, misses, and evictions for the trace file ls.trace.

What to submit?

Just the file cachelab.c and add your name and NetId, as a comment, at the top of the file.

How will we grade this lab?

We provided you with 3 traces files. We will test your code with those 3 files as well as 3 non-distributed file.
For the distributed files: 5 points each. → total of 15 points
For the non-distributed files: 15 points each → total of 45 points
This makes the whole lab worth of 60 points.

Enjoy!