CS211 Summer 2020

Assignment 3 - Cache Simulator

Due: 08/11/2020 11:55pm

Points: 100

1. Overview

The goal of this assignment is to help you understand caches better. You are required to write a cache simulator using the C programming language. The programs have to run on iLab machines. We are providing real program memory traces as input to your cache simulator. The format and structure of the memory traces are described below.

2. Memory Access Traces

The input to the cache simulator is a memory access trace, which we have generated by executing real programs. The trace contains memory addresses accessed during program execution. Your cache simulator will have to use these addresses to determine if the access is a hit or a miss, and the actions to perform in each case. The memory trace file consists of multiple lines. Each line of the trace file corresponds to a memory accesses performed by the program. Each line consists of two columns, which are space separated. The second column reports 48-bit memory address that has been accessed by the program while the first column indicates whether the memory access is a read (R) or a write (W). The trace file always ends with a #eof string. We have provided you three input trace files (some of them are larger in size). You can safely assume the trace files always have proper format. Here is a sample trace file.

```
R 0x9cb3d40
W 0x9cb3d40
R 0x9cb3d44
W 0x9cb3d44
R 0xbf8ef498
#eof
```

3. Cache Simulator

You will implement a cache simulator to evaluate different configurations of caches. Your program should be able to support traces with any number of lines. The followings are the requirements for your cache simulator:

- You simulate one level cache.

- The cache size, associativity, the replacement policy, and the block size are the input parameters. Cache size and block size are specified in bytes.
- Replacement algorithm: First In First Out (FIFO). When a block needs to be replaced, the cache evicts the block that was accessed first. It does not take into account whether the block is frequently or recently accessed.
- You have to simulate a write through cache.

4. Cache Simulator Interface

You have to name your cache simulator first. Your program should support the following usage interface: ./first <cache size><block size><cache policy><associativity><trace file>

where:
A) <cache size>is the total size of the cache in bytes. This number should be a power of 2.
B) <block size>is a power of 2 integer that specifies the size of the cache block in bytes.
C) <cache policy>Here is valid cache policy is fifo.
D) <associativity>is one of:
direct - simulate a direct mapped cache.
assoc- simulate a fully associative cache.
assoc:n - simulate an n way associative cache. n will be a power of 2.
E) is the name of the trace file.

NOTE: Your program should check if all the inputs are in valid format, if not print error and then terminate the program.

5. Sample Output

As the output, your program should print out the number of memory reads (per cache block), memory writes (per cache block), cache hits, and cache misses. You should follow the exact same format shown below (pay attention to case sensitivity of the letters), otherwise, the autograder can not grade your program properly.

```
$./first 32 4 fifo assoc:2 trace2.txt
Memory reads: 3499
Memory writes: 2861
Cache hits: 6501
Cache misses: 3499
```

In this example above, we are simulating 2-way set associate cache of size 32 bytes. Each cache block is 4 bytes. The trace file name is trace2.txt.

NOTE: Some of the trace files are quite large. So it might take a few minutes for the autograder to grade for all the testcases.

6. Other Details

1 (a) When your program starts, there is nothing in the cache. So, all cache lines are empty (invalid).
(b) you can assume that the memory size is 2pow48 . Therefore, memory addresses are 48 bit (zero extend the addresses in the trace file if theyre less than 48-bit in length).
(c) the number of bits in the tag, cache address, and byte address are determined by the cache size and the block size;

2 For a write-through cache, there is the question of what should happen in case of a write miss. In this assignment, the assumption is that the block is first read from memory (one read memory), and then followed by a memory write.
3 You do not need to simulate the memory in this assignment. Because, the traces doesnt contain any information on data values transferred between the memory and the caches.
4 You have to compile your program with the following flags: -Wall -Werror -fsanitize=address

7. Extra credit (20 points)

As an extra credit, you should implement LRU (Least Recently Used) cache policy. Your program should output exactly the same format output as it shown before. Here is an example of running your program with LRU policy.

```
$./first 32 4 lru assoc:2 trace2.txt
Memory reads: 3292
Memory writes: 2861
Cache hits: 6708
Cache misses: 3292
```

8. Submission

You have to e-submit the assignment using Sakai . Put all files (source code + Makefile) into a directory named first, which itself is a sub-directory under pa3 . Then, create a tar file (follow the instructions in the previous assignments to create the tar file). Your submission should be only a tar file named pa3.tar. You have to e-submit the assignment using Sakai. Your submission should be a tar file named pa3.tar. To create this file, put everything that you are submitting into a directory named pa3. Then, cd into the directory containing pa3 (that is, pa3s parent directory) and run the following command:

```
$tar cvf pa3.tar pa3
```

To check that you have correctly created the tar file, you should copy it (pa3.tar) into an empty directory and run the following command:

```
$tar xvf pa3.tar
```

This is how the folder structure should be.

- pa3
  - first
    * first.c
    * first.h
    * Makefile

Source code: all source code files necessary for building your programs. e.g. first.c and first.h.
Makefile: There should be at least three rules in your Makefile:
all: make a complete build of your program (first).
first: build the executables (first).
clean: prepare for rebuilding from scratch.

9. Autograder

First mode
Testing when you are writing code with a pa3 folder.
1. Lets say you have a pa3 folder with the directory structure as described in the assignment.
2. Copy the folder to the directory of the autograder
3. Run the autograder with the following command

```
$python auto grader.py
```

It will run the test cases and print your scores.

Second mode
This mode is to test your final submission (i.e, pa3.tar)
1. Copy pa3.tar to the autograder directory
2. Run the autograder with pa3.tar as the argument as below:

```
$python auto grader.py pa3.tar
```

10. Grading guidelines

1 We should be able build your program by just running make.
2 Your program should follow the format specified above for the usage interface.
3 Your program should strictly follow the input and output specifications mentioned above. (Note: This is perhaps the most important guideline: failing to follow it might result in you losing all or most of your points for this assignment. Make sure your programs output format is exactly as specified. Any deviation will cause the automated grader to mark your output as incorrect. REQUESTS FOR RE-EVALUATIONS OF PROGRAMS REJECTED DUE TO IMPROPER FORMAT WILL NOT BE ENTERTAINED.)