Jefferson Kline

011734323

CPTS 360 programming assignment 1

9/27/25

My program’s link is: https://github.com/CPTS360FA25/pa-1-JeffLepp.git

This programming assignment focused on simulating a data cache using valgrind traces. The program accepts the input of a trace file generated with valgrind's Lackey tool. To start it we are using command line flags, which are “-s, -E, -b, -t” . These each stand for a given set’s index bits, number of lines per set, and the block offset bits. There is also an option flag “-v” which stands for verbose. The outputs of the program will be the total hits, misses, and evictions given no -v.

Data structures and functions:

- Line struct: stores a valid bit, tag, LRU counter for each cache line
- Set struct: stores an array of Line structs for each cache set
- Cache struct: stores an array of Set structs, global time counter
- AccessResult enumeration: Set of results for cache code to return.

    I dynamically allocate sets and lines using calloc, which works with arbitrary values. This happens for each line/set upon initialization. Num_sets_from_bits() is used to find S = 2^s.

    For each progression of the cache, I use access_cache() which increments the global time counter and then loops through lines in each set until a hit is made. Upon hit we increment global hits and update the line’s lru to the current time. If we miss then we increment global_misses and fill an empty line. If no empty line we find the smallest lru by comparing the times. We then update global_evictions and update the line. I decode addresses with decode_address() which extracts the set index and tag using bitshifting and a mask. The global counters are the overall result of the cache simulation.

    For the trace file I use fgets to read each line in a loop. To skip the ‘I’, I simply skip the line if it begins with ‘I’. I’ll parse lines that begin with L, S, or M, with if statements for each fgets loop. To parse the inputs I use getopt which handles the respective flags, and called print_usage() when invalid arguments are given.

Testing and verification:

To test my program I first made sure that the makefile compiles cleanly using ‘make clean’ then ‘make’. After, I tested the tests that were going to be used in the documentation provided. For example, I used “./cachesim -s 1 -E 1 -b 1 -t traces/trace01.dat” to test for (1, 1, 1,). These all resulted in the correct outputs that were also provided.

To check for memory leaks or undefined behavior I ran valgrind to test these using “valgrind --leak-check=full --track-origins=yes ./cachesim -s 1 -E 1 -b 1 -t traces/trace01.dat” These showed that these issues were not taking place in my code.

Generated trace file made with valgrind’s tool=lackey functions and successfully simulates how a cache would run under those circumstances. The command used was “valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l > traces/mytrace.dat” to generate and run my program with the trace data.

Information sources that helped me:

Used https://community.unix.com/t/c-code-1ull-and-bit-calculation/330493 for bitshifting and looping logic.

Used https://coffeebeforearch.github.io/2020/12/16/cache-simulator.html for helping understand how to wrap my head around this, particularly with access_cache logic.