Cache Simulation

By: Adam J. Caudle
ID: 011774063

Creating the Structure:

With the given code, I decided to implement three structs for my cache simulation. These three structures mirror how caches are organized and were a good starting point for the project. I created a struct for the overall cache object (which included variables for its settings such as a set number of set bits, block bits, and the associativity), a struct for a set that would contain multiple cache lines, and then a struct for an individual line which contained variables to track the tag, how recently the tag was used, and whether it was valid or not. With these structs completed, I created the shape of the program by parsing command line input with getopt and could run the simulation. This was all done within main().

Initializing and running the Simulation:

To initialize the simulation, I individually used malloc() on the cache, each line, and each set based on user-provided parameters for the number of block bits, the number of set-index bits, and associativity. This lets me dynamically allocate memory based on parameters provided by the user. This completes the initialization of the cache and ensures it is ready for the simulation.

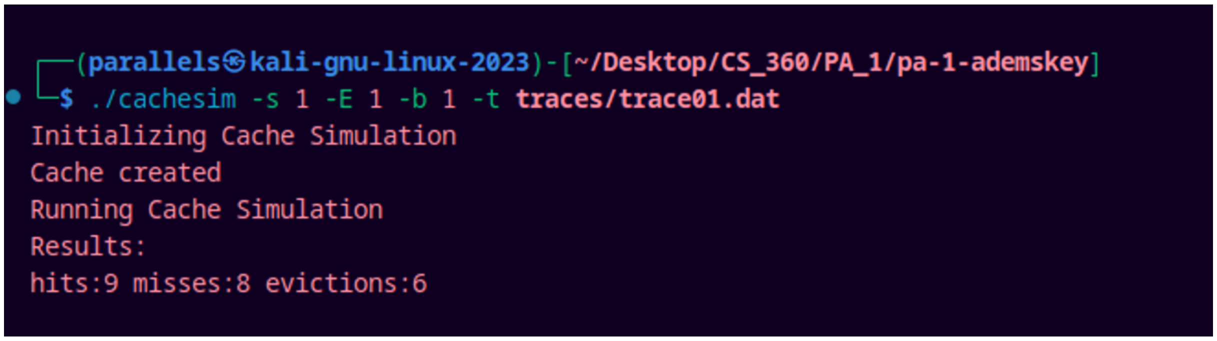
To run the simulation, I just read through the provided Valgrind trace file until the end of the file. Based on the operation type, I would then call an "access_cache()" function once or twice to access the cache based on the operation type. The access cache function would calculate the tag and set index, then search the cache set for the tag. Depending on the current state of the cache, access_cache() would complete the access while handling hits, misses, evictions, and replacements with the Least Recently Used (LRU) algorithm. I created helper functions "find_lru()" and "lru_update()" to help with LRU operations when accessing. These functions would all print their results when the verbose flag was provided giving a more detailed output during simulation.

Finishing Simulation and Printing Usage:

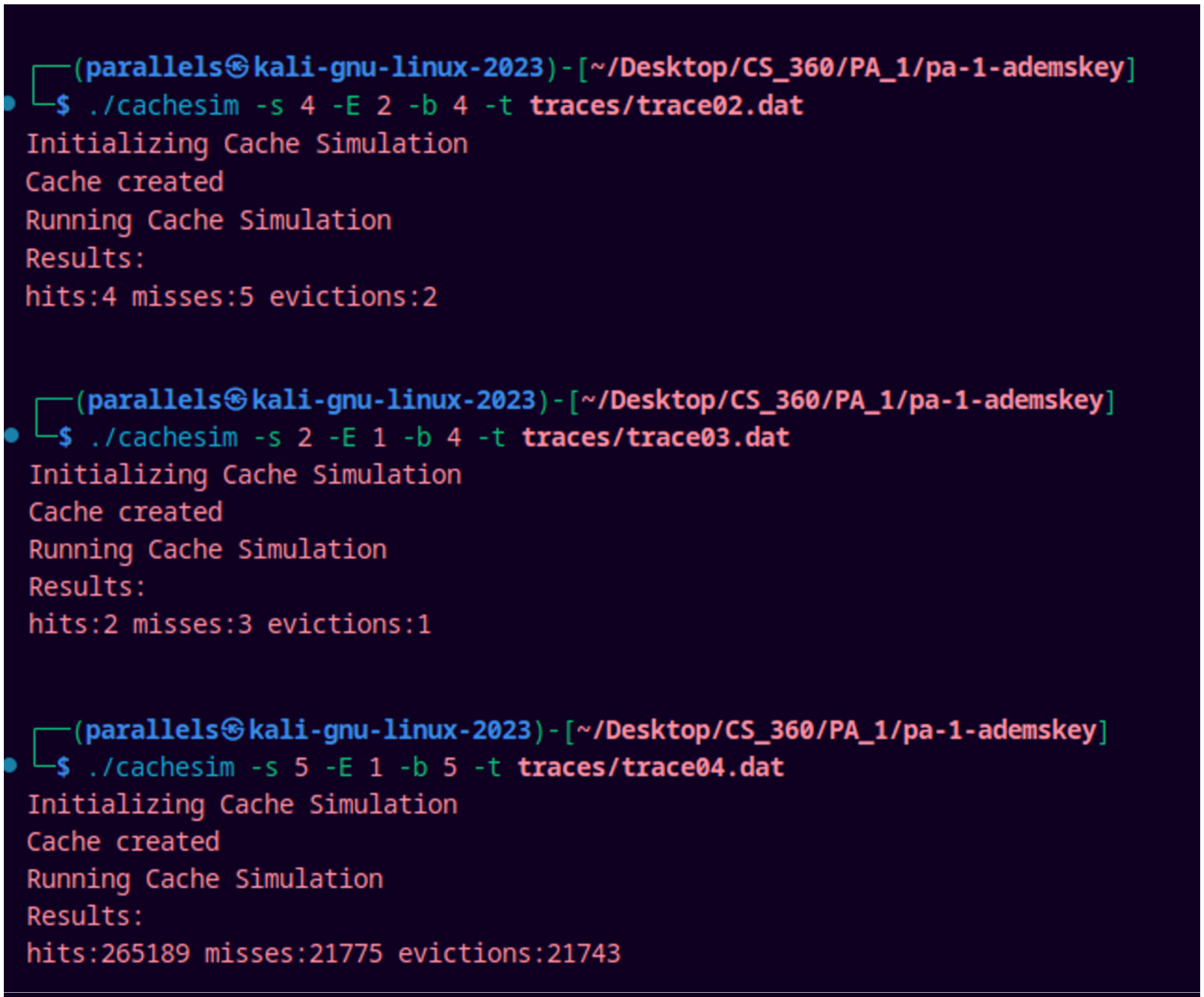
Pointers passed into my run_simulation function tracked hits, misses, and evictions. These are then passed into "print_results" to display the simulation results. In the event the incorrect or impossible criteria are passed into the command line, the "print_usage" function is called which prints the correct program usage to the terminal. With error handling, simulation code, and results all completed, I began testing the simulation.

Testing:

To test my simulation, I used the makefile without any errors or warnings and ran each of the test cases provided in the assignment PDF. An example of my correctly functioning tests and my verbose output for trace02 can be seen below.



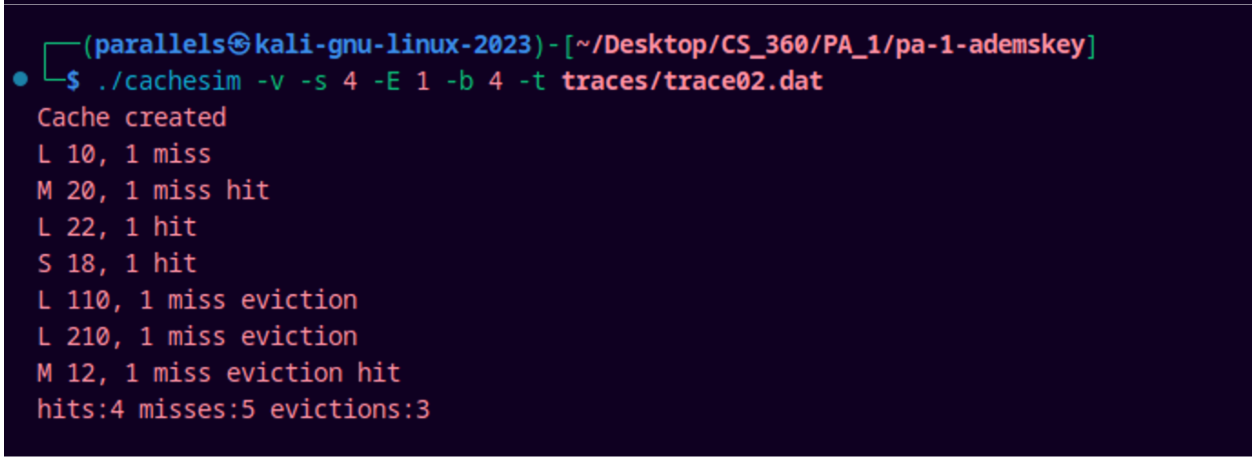
```
┌──(parallels㉿kali-gnu-linux-2023)-[~/Desktop/CS_360/PA_1/pa-1-ademskey]
└─$ ./cachesim -s 1 -E 1 -b 1 -t traces/trace01.dat
Initializing Cache Simulation
Cache created
Running Cache Simulation
Results:
hits:9 misses:8 evictions:6
```



```
┌──(parallels㉿kali-gnu-linux-2023)-[~/Desktop/CS_360/PA_1/pa-1-ademskey]
└─$ ./cachesim -s 4 -E 2 -b 4 -t traces/trace02.dat
Initializing Cache Simulation
Cache created
Running Cache Simulation
Results:
hits:4 misses:5 evictions:2


┌──(parallels㉿kali-gnu-linux-2023)-[~/Desktop/CS_360/PA_1/pa-1-ademskey]
└─$ ./cachesim -s 2 -E 1 -b 4 -t traces/trace03.dat
Initializing Cache Simulation
Cache created
Running Cache Simulation
Results:
hits:2 misses:3 evictions:1


┌──(parallels㉿kali-gnu-linux-2023)-[~/Desktop/CS_360/PA_1/pa-1-ademskey]
└─$ ./cachesim -s 5 -E 1 -b 5 -t traces/trace04.dat
Initializing Cache Simulation
Cache created
Running Cache Simulation
Results:
hits:265189 misses:21775 evictions:21743
```



```
┌──(parallels㉿kali-gnu-linux-2023)-[~/Desktop/CS_360/PA_1/pa-1-ademskey]
└─$ ./cachesim -v -s 4 -E 1 -b 4 -t traces/trace02.dat
Cache created
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
```

Conclusion:

This shows that my Cache simulator is complete and working as expected. I learned a lot about how set-associativity works, and how caches work in general. This has been a very informative assignment that was completed according to all criteria.