Sock() It to Me

by Marika Swanberg and Jillian James

1 Compiling and Running the files

There is no need to download any libraries to run our code. Both client and server are made with sockets. We have the following server files:

- cache.hh/cache.cc: Our cache implementation. Handles `get()`, `set()`, and `del()` methods on the actual key/value cache
- server.hh/server.cc: This sets up a server with sockets that accepts and parses messages from client, executes the proper cache operations, and sends the results to the client.

To compile the server, run `g++ -o server cache.cc server.cc`

We have the following client files:

- cache.hh/client.cc: The cache.hh defines the API for client.cc. When the client calls `get()`, for example, client.cc sends the appropriate message to the server and parses the response from the server to return to the client.
- catch2.hpp: a testing framework for our test.cc file
- test.cc: our tests, written using the Catch2 framework

To compile the client, run `g++ -o client client.cc test.cc catch2.hpp`. This will take a minute because catch2.hpp has like 14,000 lines.

Running the files:

- Before running the files, it is important that you read this ReadMe's "Problems" section. To run the files, please run ./server (with optional arguments -m maxmem and -t portnum) on one terminal window and after that is running, run ./client in another terminal window.

2 Design Choices

We struggled to install and understand a lot of the networks libraries because they were very poorly documented (or not at all, in some cases). Instead, we scrapped that work, and wrote our server and client using just sockets and we generally recommend this approach. We got a lot of help from this example "hello world" client and server: https://www.geeksforgeeks.org/socketprogrammingcc/. We modified this code to our use case and read up on some of the socket functions. We also got some initial help/consultation from Marika's family during a very timely wedding that immeasurably helped our progress on the socket implementation.

3 Problems

We encountered a few problems that we were unable to fix gracefully at this time. First, while server's portnum can change, client always communicates on portnum 8080, so if you want the client and the server to communicate do not give server a different portnum. Second, the main problem we had is that our tests, while they pass individually, **any two tests cannot be run in succession** because the server will shut down after the first cache is destructed.

Because of this, in order to run a particular test, you must comment out **ALL** the other tests. Additionally, while creating a cache instance, Cache mycache(32), on the client side will initialize a cache with the given maxmem=32, there is nothing in place to give the server's cache instance the same maxmem. This does not effect most of our tests. Just test_get_evicted. Therefore if you are running the test, test_get_evicted (which tests get on a value that has already been evicted and relies on maxmem to do this) you must initialize the server with the specific maxmem given to cache in that test. So when running test_get_evicted run this as the server executable: ./server 32 8080.

Despite all these difficulties, all but one test (test_set_evict) passes. The way a string does not have variable size dependent on its length is responsible for the failure of this test. (In our original test.cc we used char arrays for this test.)

Lastly, two things were not implemented:

1. We do not properly shut down the server and free the cache memory
2. We did not implement the "HEAD" message/command

While we would have wanted to implement these with the given time, we thought it was better to focus on getting the tests to pass for the items we did implement, because these tests test basic functionailty of the cache.