The George Washington University

CSCI 3410 Systems Programming

Week 9: Key-Value Server

This is an *individual assignment*, you may not collaborate or share code with other students.

Background Information The concept of a key-value store is critical to implementing many larger systems. You have likely already implemented one in Architecture (as a hash table). We are going to take that one step further in this exercise by building a remote API to a hash table that is accessible over a socket. This is a common way to "cache" data for fast future access. A well-known implementation of this technique is called memcached.

In this exercise, the socket code has been written for you, although you should be familiar with the various system called used (specifically `getaddrinfo`, `socket`, `bind`, and `listen`). If you are unclear on what a function does, refer to its `manpage`. Additionally, you should refer to this guide to socket programming if you have further questions or desire more background.

For this application, we will need to run two programs. First, is our server. This program starts and binds to a socket on the machine so other programs can communicate with it. It reads data sent to it (using the same UNIX APIs we already know and love), processes the request, and writes its response back over the socket. A second program can then start and write a string command to the "other end" of the socket and read the response.

System Specification

Find all `TODO`s in the existing code and complete the implementation of the server. In broad strokes, the server binds a client and reads/processes incoming commands (read over a socket) until the client disconnects. The following functions should be supported:

- `GET x`: Fetches the key x from the backing hashtable and return its value over the socket
- `SET x y`: Updates the value for key x to be y. Returns 'ok' over the socket on successful completion
- `CREATE x`: Puts an item with key x in the backing hashtable. Returns 'ok' over the socket on successful completion
- `DELETE x`: Removes an item with key x from the backing hashtable. Returns 'ok' over the socket on successful completion

Testing

Build and run the server using the provided `Makefile`. Run the server and pass the desired port as an argument. For example, to listen on port 8080, run `./kv-server 8080`.

Use the `telnet` program in an additional shell to connect to the server. For example, to connect to the server running on port 8080 on the local machine, run `telnet localhost 8080`. Refer to `man telnet` for more details.

Deliverables and Grading

- Push your code to GitHub (a link will be posted on Blackboard) before the deadline.
- All of your code (including your tests) should compile with the following `gcc` flags: `-std=c99 -Wall -pedantic -Werror`. You will lose credit if your code compiles, but only without those extra flags.
- Be sure to write clear and concise commit messages outlining what has been done.

- Write clean and simple code, using comments to explain what is not intuitive. If the grader cannot understand your code, you will lose credit on the assignment.
- Be sure your code compiles! If your program does not compile, you will receive **no credit**. It is better to submit a working program hat only does a subset of the requirements than a broken one that attempts to do them all.

Category	Percentage
Instructor Tests Pass	70%
Code Quality & Implementation	20%
Compilation with `-Wall -pedantic -Werror`	10%

Table 1: Grading Rubric