**Project 2: Key-Value Network Service**

CSC 6712 Distributed Storage Systems

**Overview**

In this project, you’re going to implement a TCP/IP server on top of your key-value library from project 1 to create a key-value network service.

**Part I: Design a Protocol**

You will need to design a protocol for clients to interact with the server. Your key-value store should support the following operations:

* PUT key value -> value? : Puts the key-value pair in the database. If the key is already in the database, return the old value. Otherwise, return null.
* GET key -> value? : Get the value associated with the given key from the database. If the key is absent, return null.
* GET PREVIOUS key -> (key, value)? Get the value associated with the key preceding the given key from the database. If there is no preceding key, return null.
* GET NEXT key -> (key, value)?: Get the value associated with the key following the given key from the database. If there is no succeeding key, return null.
* CONTAINS key -> boolean : Return true if the key is in the database and false otherwise.
* SHUTDOWN : Starts a clean shutdown procedure of the network service. All operations received before the shutdown command should finish. New connections and new operations should not be accepted. All connections should be closed gracefully once all outstanding operations for each connection are completed.

Your protocol will need the ability to respond with an error. For example, if an operation is received after SHUTDOWN is initiated, an error code should be returned indicating that the server is not accepting new requests.

In your report, describe the byte layout of the protocol in sufficient detail for someone else to implement it.

**Part II: Implement a Network Server**

Once you've designed a protocol, implement a network server.

Implement your server using multiplexing I/O (e.g., select() or asyncio in Python) to support multiple concurrent connections. You'll be using blocking I/O so only one request can be handled at a time. Nonetheless, when waiting on a client to initiate a request, the server can receive serve requests from other clients. Since no parallel processing is involved, you don't need to worry about concurrent modifications. Connections should be maintained until the client disconnects (instead of establishing a new connection for each operation). Ensure that your multiplexing is set up so that requests from multiple connections are interweaved. Meaning that your implementation should not handle all requests from connection before moving on to the next connection.

**Part III: Benchmarks**

You want to benchmark your service to determine whether the non-blocking I/O (multiplexing) allows for achieving a higher the total throughput (average number of requests served per second) with multiple connections. You will focus on read requests.

Create a database with 1 million key-value pairs.

You will need to implement a benchmark client that performs N read requests where N is given on the command-line. The client should simulate gaps between requests by sleeping for 10 ms after reach request.

Evaluate the amount of time required for a client to complete 10,000 read operations with random keys. Then re-measure with 5, 10, 20, 50, and 100 total clients running concurrently. Compute the average number of requests per second by summing the total number of requests made across all clients divided by the run time (in seconds). Plot the average requests per second against the number of clients.

Hints:

* [Supervisord](http://supervisord.org/index.html) can be used to start and stop multiple copies of a program using the numprocs keyword.

**Part IV: Written Report**

You will need to write a 3-4 page report that includes:

* A description of your network protocol at the byte or text level. sufficient for someone to create a compatible implementation. Use tables, visuals, etc. as needed.
* Your benchmark results (as a graph) for read throughput and your interpretations of their meaning. Up to how many clients does throughput continue to improve?

**Submission Instructions**

Create a new branch to capture the state of your GitHub repository at submission time. Submit a link to the GitHub repository and your report (in PDF format) on Canvas.

**Rubric**

|  |  |
| --- | --- |
| **GitHub Repository:** Has a README.md file with instructions sufficient for building and running the code. | 5% |
| **Network Service:** Implementation is functionally correct. Multiplexing is implemented so that requests are handled in a round-robin manner. Closed connections (expected or not) are handled gracefully rather crashing the server. Graceful shutdown is performed without interrupting queued operations. New connections and operations are rejected. | 40% |
| **Written Report (Overall):** Report is written in a professional manner using proper grammar and spelling. Report is a useful standalone document that can be shared with another student. Report contains a detailed description of the implementation, experimental designs for benchmarks, benchmark results, and interpretations of the results. | 10% |
| **Plots:** Appropriate types of plots were chosen for each analysis. Axes are properly labeled. Used legends if appropriate. Chose appropriate axis limits to make plots readable and avoid misleading interpretations. Font sizes are legible. Figures are saved at high resolutions. | 5% |
| **Protocol Description:** Description of your network protocol is sufficient for someone to create a compatible implementation. | 15% |
| **Benchmarks:** Experimental designs are correct. Benchmark results are interpreted correctly. | 25% |