Name: Aditya Arun Rudrawar

Instructions: Install the required modules from requirements.txt

Run server command: python server.py

Run client command: python client.py

# 1. Description and Details

## Design

Diagram

Description automatically generated

The .env file has the host, port for TCP connection.

Implemented Memcached-lite. The server supports multi-threaded operation. One or more clients can be connected to the server where each client does a single request and closes the connection after the request is completed. The Basic server and client uses the socket module to connect to the host and listen to the specific port number. Based on the memcache protocol of set and get; we send the key with its flag and expiry and the value while set operation and the operation is defined by ‘SET’ keyword, similarly for ‘GET’ operation as well.

The client.py initializes 200 (arbitrary no of clients) clients where each client does a either a SET or GET, we are closing the client connection after operation. The testing is explained in the later section.

## A. Advanced: Concurrency

The server handles concurrency by using the python multithreading module, where the server waits for a connection and for each request spawns a new non-blocking thread. This way each request from each client will run on their respective threads. The above figure explains the structure of concurrency. That thread will respond to the clients after processing ends. The write requests acquires a global lock on the file, and does not allow any other read/write until the lock is release. This mimics the concept of critical sections and allows us to be consistent. While reading we just need to make sure that other thread is not writing to it, so we check if the lock is locked or not.

## B. Advanced: Memcache compatibility

To make the server compatible with memcache client, I had to format the set and get request that are in line with the specification. For the set request I had to pass the expiry, flags, no\_reply parameters as well, these parameters are persisted, and the use case of these are depending on the server. For now, there is no use case. In the future we can use them for our purposes.

## C. Limits

In the code there is not specific limit set for the key or value. But the server/client fails to respond when the length of the value becomes greater than or equal to 64000 characters, they probable cause might be the packet limit set by TCP connection.

# 2. Performance and Testing

The server is tested against three types of test cases.

## A. 200 Concurrent SET requests.

Initialized 200 clients on the client.py code, each randomly generating key of length 10, a value of length 50. Setting the no\_reply flag to False, to measure the performance of the server, and the time it takes to set the value of the key. The keys that are used again for further test cases. The graph below shows for 200 concurrent requests.

## B. 200 Concurrent GET requests.

The keys generated previously are then used to test the GET request. 200 clients are used to connect and request the value of those keys.

## C. 200 SET and 200 GET requests together.

For better real-world simulation I simultaneously send GET and SET request in a random order by creating a thread for each client and starting them. The performance of them is shown through the graph below.

# 3. Errors (not handled)

When the key has any special characters, the memcache client raises an exception before sending the request to the server, but not all special characters are considered for example @ is passed to the server even though it is a special character.

# 4. Limitations

For now, the server does not fully supports the memcache protocol, it only supports GET/SET operation with only one key. It does not support multiple keys for those operation in a single request. In addition to which it also does not support all the Retrieval/Storage/Error commands.

# 5. Future Improvements

A. We can use better file system as just one source of file will soon become bottleneck as the number of GET/SET request start increasing exponentially.

B. We can add support for the full suite of memcache commands to make it completely operable.