Assignment 1: Memcached-Lite

Problem Statement

- Implement a TCP server, that stores and retrieves data for different clients.
- Server should be able to store data using *set* command and retrieves data using *get* command.
- Server should handle more than one request at a time.
- The server should run on the SICE Linux servers.

Understanding Problems

- First problem is when requesting *set* and *get* command, input should in specific format so that server can parse it as per the request.
    - Intermediate Problem-
        - *set* command should be two lines request, while *get* command will be one line.
- Second problem is that server will be assigned one port, in case of multiple requests, all requests will be coming from same port, if the request will be processed on that port, processing will be slow, and we need to make this processing concurrent.

Design Details

This architecture has a server and multiple clients which are tested on Linux server burrow.luddy.indiana.edu, below are the design details:

Server

- The server is implemented in Python, server.py.
- For first problem, two separate functions *get_data()* and *set_data()* are created for parsing the request and execute accordingly.
    - This gives us flexibility to parse request individually.
    - *set* request will be parsed line by line:
        - Syntax is as follows-
          `set <key> <value-size-bytes> \r\n`
          `<value> \r\n`
        - Here `<value-size-bytes>` should be same as length of `<value>`. The server will send following message based on this condition-
          `STORED` if matches
          `NOT STORED` if not matches
        - If key is already present, then value will be updated.
    - *get* request is a single line request
        - Syntax is as follows-
          `get <key>\r\n`
        - The server will send below message to the client
          `VALUE <key> <bytes> \r\n`
          `<data block>\r\n`
- For second problem, a function *handle_client()* is created which is assigning new port and address to the request as soon as it is arriving on the server port. This will create different threads for each request and execute it accordingly.

Client

- The client is implemented in python
    - client.py will open a terminal where a user can write and send request to server.
    - client_multiple_req.py will send a series of requests to the server automatically.
- For first problem, designed a function *take_input()* for taking the input unless a blank enter is entered.
    - This gives flexibility such that we don't have to define a function for any additional request (if added) separately.

Test Cases

S No	Test Case	Script
1	Server should start running on SICE linux server	run_server.sh
2	Client should connect with Server	run_single_client.sh
3	The get Command should retrieve value based on Key	run_single_client.sh
4	The set Command should store key-value in the file system	run_single_client.sh
5	If key is already present, value should be updated with new	run_single_client.sh
6	The server should be able to process series of requests	run_single_client_multi_req.sh
7	Server should be able to process concurrent client requests	Run (run_single_client_multi_req.sh) multiple times

Examples

Let's evaluate the working of this code by sending following requests and see how the server responds:

Input	Output
get S520(Double Enter)	VALUE S520 26 INTRODUCTION TO STATISTICS
set E516 27 (Enter) ENGINEERING CLOUD COMPUTING (Double Enter)	STORED
get E516 (Double Enter)	VALUE E516 27 ENGINEERING CLOUD COMPUTING

Solution Evaluation and Review

- The server is running on the SICE linux server burrow.luddy.indiana.edu as expected.
- The client is connecting to the server.
    - Multiple clients are getting connected to the same server.
- When the input request is given in expected format, *get* command is retrieving the values from the file system as expected.
    - If the key is not present in the system, then the server will not return any value.

    - After all items have been transmitted, the server sends 'END'.
    - The server returns ERROR if the request is invalid.
- When the input request is given in expected format, *set* command is storing the values in the file system as expected.
    - The server will send STORED after successfully storing the key value.
    - If the length of the value is not equal to the value size bytes, the server will send NOT STORED.
    - The server returns ERROR if the request is invalid.
- When series of commands were given to the server through client, it can execute them.
- Multiple clients are getting connected to the server
    - Each request is getting assigned to different Ports and address.
    - Server is executing the requests concurrently.
    - The server was tested by running more than 20 clients simultaneously.

Limitation & Future Improvement

- For this project, the data was stored in json file. This file needs to be created with one key value initially, so that the server can open and dump other key values in the file.

  This can be overcome by inserting the if file exist condition. This is not included in current project because there was a blocker which was not resolved due to time constraint.

- The size of the request (key, value and req type) is 2048 bytes. This can be increased in future depending on the requirement. But currently, it is working fine.

References

- To create a TCP client and server in Python. (https://realpython.com/python-sockets/)
- For running the requests in parallel. (https://www.geeksforgeeks.org/socket-programming-multi-threading-python/)