Assignment 1: Memcached-lite

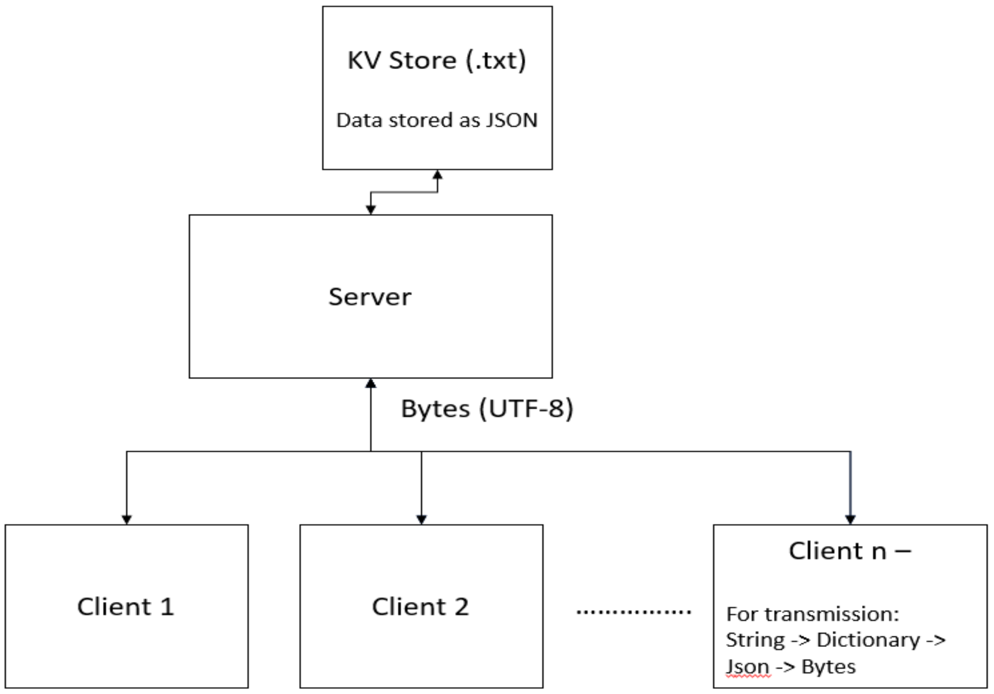
The memcache system consists of two files, one server file and one client file. The server runs on port 5050 and has IP address 149.159.214.71. It listens and accepts incoming valid client connections. Once the clients connects to the server, the server starts a thread to keep track of parallel multiple client connections if any. For every incoming client connection, a new thread is opened which is processed by the **handle_client** function. The communication between the server and the client is done using sockets where data is transferred between the respective sockets as bytes. The byte format is the UTF – 8 standard for communication. While setting the data in memcache, the key, value and the byte length is first extracted from the string and stored in a Python object which is then converted to a json string. Following that, the json string is encoded with the UTF-8 standard and then transferred over the socket to the client. On the server side, exactly the reverse process occurs where the data from the client is first decoded and then converted from json string to a dictionary.

Storing the Data

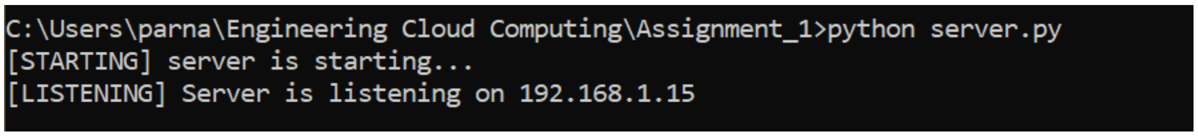
When the data from the client first reaches the server, it extracts the key, value and the byte length from the loaded JSON string. The storage file system named **"memcache.txt"** is opened and then dictionary which is converted to JSON string is stored in the file. If there's a pre-existing key in the file system, it is overwritten with the new value.

Fetching the Data

Fetching is a straight forward process where the entire content of the storage file is stored in a temporary dictionary on the server and then it's parsed through to search for the key that the user asked for. If found, the key, value and the bytes length is transmitted back to the client followed by the word END indicating the end of the message from the server. If not found, -1 is sent back which means that the key was not found.

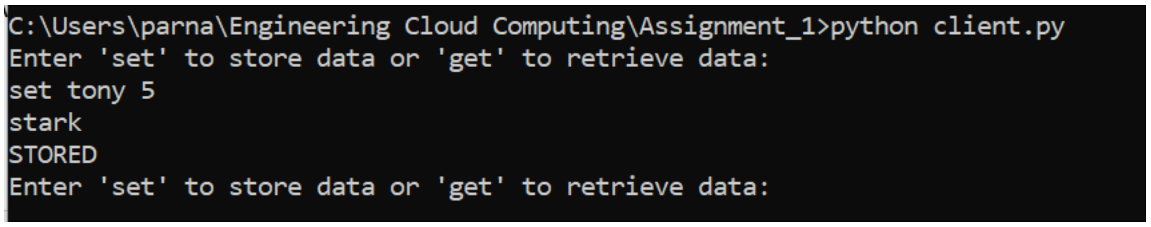


Server Listening:



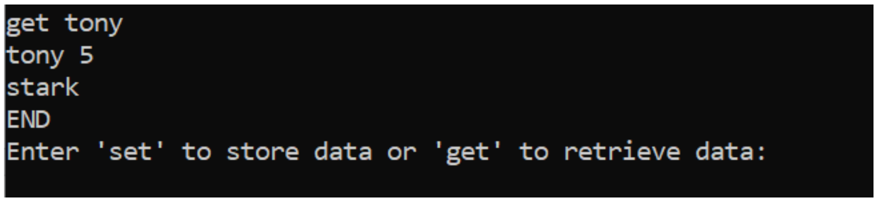
```
C:\Users\parna\Engineering Cloud Computing\Assignment_1>python server.py
[STARTING] server is starting...
[LISTENING] Server is listening on 192.168.1.15
```

Setting Data:



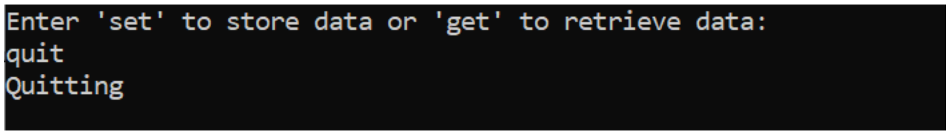
```
C:\Users\parna\Engineering Cloud Computing\Assignment_1>python client.py
Enter 'set' to store data or 'get' to retrieve data:
set tony 5
stark
STORED
Enter 'set' to store data or 'get' to retrieve data:
```

Getting Data:



```
get tony
tony 5
stark
END
Enter 'set' to store data or 'get' to retrieve data:
```

Shutting down client:



```
Enter 'set' to store data or 'get' to retrieve data:
quit
Quitting
```

Performance of the system:

The system was tested by manually connecting 5 concurrent clients from the command terminal and then storing and retrieving data at the same time without any of the client being disconnected. This is handled with threading as a new thread is maintained for every client connection with the server and the thread function performs specific operations such as storing and retrieving data based on the particular client that was connected. The system is yet to be tested for the maximum concurrency limit.

Error Handling:

There are multiple test cases which have been looked into. There are still a few additional cases which haven't been accounted for but they have been listed under future improvements

1) If the client enters anything other than get, set or quit, the client should know that these are the only commands that the client and the server understands. The server prints 'Invalid input' to let the client know expected commands
2) When the client disconnects from the server with Ctrl + Z or Ctrl + break, it is smoothly handled on the server side with a message indicating that the client got disconnected. For multiple clients, the server specifies which client got disconnected.
3) While setting values, the server accepts values only till the specified byte length entered by the client before storing in the file system.

Future Improvements:

1) The server has been tested for 5 clients connected at the same time but the maximum concurrency limit can be tested. It will give a good idea about server handling capacity and parallel processing
2) The socket can handle a maximum of 2MB of data but this can be improved upon and extended conditionally for specific use cases where a large stream of data is to be transmitted.
3) Error handling can be added on the client side for abrupt shutting down of the server so that the end user knows what the issue was.
4) When the message side exceeds 2 MB, the client should know that the data was not stored in the file system because the socket message size limit was exceeded.