Online Message Board Client using C/C++, Java or Python

Table of Contents

1. Project Description
2. Development Environment
3. Project Demo
4. List of Tasks
5. Project Requirements and Tutorials
6. Code Quality
7. Assessment Tool
8. Submission
9. Getting Help

1. Project Description

In this project, you will implement the client of an online message board system, where an online message board is a client-server application, clients can POST, GET and DELETE messages, and the server runs on a specified port and supports the mentioned four commands received from the client: **POST, GET, DELETE,** and **QUIT**.

(i) **POST:** It allows the client to send a message to the server. The client can type the message (using the standard input) line-by-line, the first line must be the command (i.e. POST) and subsequent lines are treated as part of the message, and the message will be ended by a special symbol #.

For Example: suppose a user wants to post "some text!\nmore text." to the server. The example is shown below.

```
POST
some text!
more text.
#
```

The server will acknowledge the receipt of the message with "OK".

(ii) **GET:** upon received this command, the server will send the client all previously received messages (in addition to the original messages, the server also sends the associated message IDs (by the server) and their received times.). In the client screen, these messages need to be print out. Note that the server will also send any messages that were previously sent by other clients if existent.

(iii) **DELETE:** the client can use the DELETE command to remove any message with the message ID (the client can find its message IDs through the **GET** command) from the server. The first line is the command line (i.e. DELETE), followed by typing message ID per line, which the message to be removed. Once all to-be-removed message IDs have been typed, the last line must be the "#" symbol to finish the delete operation

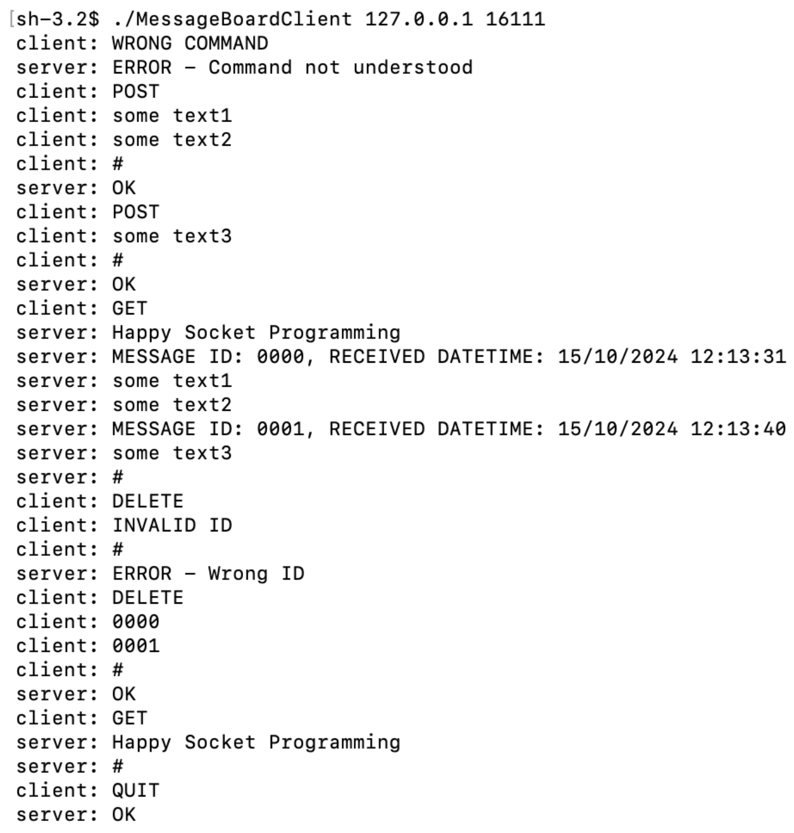
For Example: suppose a client wants to delete messages with IDs 0000, 0001 and 0002, respectively. The example is shown below:

```
DELETE
0001
0000
0002
#
```

If the deletion operation is successful, the server will respond with "OK." If any of the input message IDs is incorrect, the server responds with an ‘ERROR’ message back to the client.

(iv) **QUIT:** the xlient informs the server to close the current session. The server will acknowledge the receipt of the command with "OK", and disconnect the session with the client.

We here make use of an example to illustrate how to design a client socket by meeting the specified requirements. For exmaple, assume that the server is running on IP address 127.0.0.1 with Port number: 16111. After establishing a TCP connection to the server socket, the client program will repeatedly ask its client to input a command line for execution. For commands **GET** and **QUIT**, the client will just send the command name to the server, whereas for **POST** and **DELETE**, the client will also send a message in addition to the first line of the command name (i.e. **POST** or **DELETE**), and the rest lines include the message itself. Below is a possible communication scenraio between a client and the server:



```
[sh-3.2$ ./MessageBoardClient 127.0.0.1 16111
client: WRONG COMMAND
server: ERROR - Command not understood
client: POST
client: some text1
client: some text2
client: #
server: OK
client: POST
client: some text3
client: #
server: OK
client: GET
server: Happy Socket Programming
server: MESSAGE ID: 0000, RECEIVED DATETIME: 15/10/2024 12:13:31
server: some text1
server: some text2
server: MESSAGE ID: 0001, RECEIVED DATETIME: 15/10/2024 12:13:40
server: some text3
server: #
client: DELETE
client: INVALID ID
client: #
server: ERROR - Wrong ID
client: DELETE
client: 0000
client: 0001
client: #
server: OK
client: GET
server: Happy Socket Programming
server: #
client: QUIT
server: OK
```

The **server program has been provided** to you, which can be downloaded in the project demo file.

You **only need to implement the client program.** Name your client **MessageBoardClient**.

2. Development Environment

Software:

You can choose to code the client in either C/C++, Java, or Python, based on your own preference and familiarity.

- If you use **MS Windows**:
  - for C/C++:

    Visual Studio: *https://visualstudio.microsoft.com/downloads/* (Links to an external site.)
    Install C++ support in Visual Studio: *https://docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation?view=vs-2017* (Links to an external site.)

  - for Java:

    IntelliJ IDEA: *https://www.jetbrains.com/idea/download/#section=windows* (Links to an external site.)

  - for Python:

    PyCharm: *https://www.jetbrains.com/pycharm/download/#section=windows* (Links to an external site.)

- If you use **Linux Ubuntu System** or **Mac OS X**:
  - for C/C++:

    The gcc/g++ compiler is usually installed by default.

  - for Java:

    IntelliJ IDEA: (Mac OS) *https://www.jetbrains.com/idea/download/#section=mac* (Links to an external site.)
    IntelliJ IDEA: (Linux) *https://www.jetbrains.com/idea/download/#section=linux* (Links to an external site.)

  - for Python:

    PyCharm: (Mac OS) *https://www.jetbrains.com/pycharm/download/#section=mac* (Links to an external site.)

PyCharm:
(Linux) *https://www.jetbrains.com/pycharm/download/#section=linux* (Links to an external site.)

Remark: it is suggested that computers involved in the same working group should use the **same operating system** and apply the **same software tool** for the project design.

3. Project Demo

Apart from the server program, we have also provided the demo programs to demonstrate the application's operations:

- For Windows System, the demo program is provided as: Demo program (Note that it is for the Windows 10 (64 bit) operating system)

- For Linux System, the demo program is provided as: Demo program (Note that it is developed under Ubuntu 18 (64 bit) operating system)

- For Mac OS, the demo program is provided as: Demo program (Note that it is developed under Mac OS High Sierra)

Here are some tips about how to run the demo program. To run the server program, you should use the following command:

for Linux/Mac:

```
./MessageBoardServer
```

for Windows:

```
MessageBoardServer.exe
```

To run the client program, you should use the following command:

for Linux/Mac:

```
./MessageBoardClient <server_ip_address> <server_port>
```

for Windows:

```
MessageBoardClient.exe <server_ip_address> <server_port>
```

Some notes about the provided **server program**:

- The default port number of the server is **16111**, so <server_port> is typically set to 16111.
- It will print out information about errors or commands that it receives from the clients; otherwise, it remains silent.
- When the server receives a POST command, it will print out all stored messages.
- The server keeps listening even if the client disconnects from the server.
- You can run the server program using the following command.

4. List of Tasks

Task 1: Socket initialization

Description: initialize and create the TCP socket.

Task 2: Initiate the connection request

Description:

- Input the server IP and port.
- Initiate a connection on a socket.

Task 3: Send command to the server

Description: send the command to the server.

- **POST**: the program continues to accept the user input until entering a line that consists only of a "#" followed by a newline character '\n'.
- **GET**: send this command to the server.
- **DELETE**: the program continues to accept the message IDs for deletion until entering a line that consists only of a "#" followed by a newline character '\n'.
- **QUIT**: send this command to the server.
- Other cases: send this command to the server.

Task 4: Receive the message from the server.

Description: receive the message from the server and perform the corresponding action.

- **POST**: if "OK" is received, it indicates that the server has successfully received all of the messages from clients. Print out "OK" and then allow the user to input the next command.
- **GET**: receive all messages from server and print them to the screen. Then, allow the user to input the next command.
- **DELETE**: if "OK" is received, it indicates that the deletion is successful. If "ERROR - Wrong ID
- " is received, it indicates that the deletion is unsuccessful. You should print "OK" or the error information to the standard output.
- **QUIT**: if "OK" is received, it indicates that the server has successfully received this command. Then, the socket will be closed (see Task 5).
- Other cases: receive the error message "ERROR - Command not understood" that is sent by the server. Print out the error message and allow the user to input another command.

Task 5: Close socket

Description: when the client has sent the "QUIT" command and received "OK" from the server in response, close the connection socket.

The input must include the `IP Address` and `Port Number` of the server.

How to find your IP address:

- Linux or Mac: Type `ifconfig` in terminal

- Windows: Network & Internet settings -> network properties

5. Project Requirements and Tutorials

5.1. Project Tutorials

To provide you some starting points for your project implementation, we provide tutorials in *C/C++*, *Java* and *Python*. Basically, you need to implement the five tasks (explained above) to complete the assignment.

The links to the tutorials are here:
C/C++_tutorial
Java_tutorial
Python_tutorial

5.2. Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

- Proper decomposition of a program into subroutines (and multiple source code files when necessary) -- A 500 line program as a single routine won't suffice.
- Comment---judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:
  - Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments and will result in reduction of marks.
  - Comment your code in English. It is the language of instruction of the university.
- Proper variable names -- `leia` is not a good variable name, it never was and never will be.
- Use a small number of global variables, if any. Most programs need a very small number of global variables if any. (If you have a global variable named `temp`, think again.)
- **The return values from all function calls should be checked and all values should be dealt with appropriately.**

6. Assessment Tool

- You **must make it clear how to compile and run your code** by explaining it in your `readme` text file. The development environment should also be indicated in this file.
- TAs are not supposed to fix bugs, neither in your source code nor in your `Makefile`.
- If an executable file cannot be generated or executed successfully, it will be considered as unsuccessful.
- Any program that does not print the output in the format required in Section 1, will be considered as unsuccessful.
- If the program can be run successfully and the output is in the correct form, your program will be graded according to the following marking scheme in *Table 1*.

Table 1: Marking scheme.

Components	**Weight**
`Initialize socket`	5%
`Connect to server`	5%
`POST command`	15%
`GET command`	15%
`DELETE command`	15%
`QUIT command`	15%
`Other commands`	10%
`Error handling`	10%
`programming style and in-program comments`	10%

7. Submission

- This assignment is to be done **individually or by a group of two students**. You are encouraged to discuss the high-level design of your solution with your classmates but you must implement the program on your own. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of two files: a source program file (e.g. .cpp, .c, .py or .java file), a readme (.txt) file telling us how to compile your code, and a `Makefile` if applicable.
- Use your student ID to name your submitted files, such as 5xxxxxxx-5xxxxxxx.py, 5xxxxxxx-readme.txt for submission.
- Submit the files to Canvas.
- No late submission will be accepted.

8. Getting Help

If you have any programming issues, please feel free to contact the following TA:

HUANG Zian: huang.za@cityu-dg.edu.cn