CS456 Fall 2019

Assignment 1 - Introduction to Socket Programming

Due: October 7th, Midnight
Language: Any

1 Purpose

To gain experience with basic socket programming, sending and receiving TCP and UDP packets.

2 Problem

Implement a simple message board server and client. The message server will store all client messages, and, upon request send those messages to a client. First, the server and client will negotiate, using TCP, a port over which all other communications will take place. Then, the server will send all of its stored messages, over UDP, to the client who will display them. Next, the client will send its own message (think of it as a reply) to the server---who will add this message to its list of stored messages.

2.1 Specifications

Client-server communication should happen as follows:

1. Server and client negotiate, over TCP, a communications port.
    a. The client creates a TCP socket and connects to the server.
    b. The client sends a message to the server containing a numerical code (req_code).
    c. The server receives the message from the client and verifies that the req_code is valid.
        a. if the req_code is invalid, the server should reply to the client "**0**", and the client should terminate with an error "**Invalid req_code.**"
    d. The server sends the client a message with a port number, which will be used for communication.
2. Retrieving the stored messages from the server.
    a. The client sends a message "**GET**", over UDP, to the server.
        i. Note that this message should not be displayed.
    b. The server should then send all stored messages over UDP to the client.
    c. When there are no more messages, the server sends "**NO MSG.**", over UDP, to the client.
    d. The client will display each message on its own line.
3. Adding a message to the server.
    a. The client sends its text message, over UDP, to the server.
    b. The server stores this message as:

```
[CLIENTPORT]:  <msg>
```

Where CLIENTPORT is the port used by the client to send the message.

c. If the client sends the message "**TERMINATE**", the server shuts down.
d. The client waits for keyboard input before exiting.

2.2 Message Server

Implement a program, named **server**, which will be the message server. The server requires a single parameter, **req_code**, which is used to validate clients attempting to connect. When the server starts, it must print out the negotiation port number, <n_port>, in the following format:

```
SERVER_PORT=<n_port>
```

For example, if the negotiation port of the server is 65535, then the server should print:

```
SERVER_PORT=65535
```

The server must accept multiple, concurrent connections. It must also store all of the messages received from every client. Each client message should be stored in the following format:

```
[CLIENT_PORT]:  <msg>
```

The server should send these messages to a client if the client sends the message "**GET**". If there are no stored messages, or, after all message have been sent, the server sends "**NO MSG.**" to the client. The server should terminate if any client sends "**TERMINATE**" as a <msg>.

2.3 Client

Implement a program, **client**, which will send and receive messages from the message server. There are four parameters: <server_address>, <n_port>, <req_code>, and <msg> in that order.

```
client <server_address> <n_port> <req_code> <msg>
```

<server_address> is the address of the server, <n_port> is the negotiation port selected by the server and described above, <req_code> is the validation code, and <msg> is the message to send to the server.

The client should receive messages from the server after sending the "**GET**" command and then print the received messages, one per line. Afterward, the client should send <msg> to the server and shutdown. If <msg> == TERMINATE, the server should shut down.

2.4 Example Execution

Two shell scripts named **server.sh** and **client.sh** are provided. Modify them according to your choice of programming language. You should execute these scripts which will then call your client and server programs.

Server

```
./server.sh <req_code>
```

Client

```
./client.sh <server address> <n_port> <req_code> ‘Hello!’
```

Below is the output for a three client example (done in Java). Note that only the client is shown.

```
churro:A1 lanortha$ java ReaderClient localhost 65535 57 "How are you?"
NO MSG.

Press any key to exit.

churro:A1 lanortha$
```

```
churro:A1 lanortha$ java ReaderClient localhost 65535 57 "I'm awesome."
[56492]: How are you?
NO MSG.

Press any key to exit.
```

```
churro:A1 lanortha$ java ReaderClient localhost 65535 57 "TERMINATE."
[56492]: How are you?
[62805]: I'm awesome.
NO MSG.

Press any key to exit.
```

3 Hints

- You can use the Java sample code provided in the Module 2 slides (pages 92-112).
- Use ports greater than 1024, since [0,1023] are reserved.
- If you are experiencing difficulties establishing a connection, check for firewalls---the port may be blocked, etc.
- The server must be running before starting the client.
- If server and clients are running on the same system, 127.0.0.1 or localhost may be used as the address.
- You may use guides, books, etc., so long as you cite your sources appropriately.
- You may NOT work in groups or share your code.
- If your code is not readable or documented, you WILL lose marks.

4 Hand-in Instructions

Submit all your files in a single compressed file (.zip, .tar, etc.) using LEARN, in the dedicated Dropbox. You must hand in the following files/documents:

1. all source code files
2. Makefile (your code **must** compile and link by typing **make** or **gmake**)

3. README file containing compile and runtime instructions
4. Modified server/client.sh scripts

Your implementation will be tested on the undergrad environment machines. **Please compile and test your code on these machines prior to submission.**

4.1 Late Policy

10% penalty for each day (24hrs) late up to a maximum of 3 days. Submissions will not be accepted beyond 3 late days.

5 Evaluation

	Client and server compile and run.
	Server and one client on same machine. Incorrect <req_code>.
	Server and one client. Client 1 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and one client on the same machine. Client 1 Message: 'Hello!'
	Server and two clients on the same machine. Second client is started after the first client shuts down. Client 1 Message: 'Hello!' Client 2 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and two clients on the same machine. Clients alive at the same time. Client 1 Message: 'Hello!' Client 2 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and one client on different machines. Client 1 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and two clients on different machines. Second client is started after the first client shuts down. Client 1 Message: 'Hello!' Client 2 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and two clients on different machines. Clients alive at the same time. Client 1 Message: 'Hello!' Client 2 Message: 'TERMINATE' Server: shuts down/terminates.
	Server and four clients on different machines. Clients alive at the same time. Client 1 Message: 'Hello!' Client 2 Message: 'How are you?' Client 3 Message: 'Thank Mr. Goose.' Client 4 Message: 'TERMINATE' Server: shuts down/terminates.
	Code style/documentation

Total (out of 10):

Expected Output

- If there are no messages stored on the server:
    - NO MSG.
- If there are messages stored on the server (messages can be in any order, except for NO MSG.):
    - [some port]: message
    - …
    - NO MSG.
- The client sending **‘TERMINATE’** will be started after all other clients.

Comments: