Program Assignment 2

CMSC 417 Spring 2015

February 26, 2015

1 Deadline

March 12, 2015.

2 Objective

In this assignment you will write the server program which will communicate using sockets with the client program provided by us. Your server program will operate according to the protocol described in the next chapter. Obviously, the protocol is trivial/useless, however, this exercise will get familiarize you with client/server programming and communication protocol.

3 Protocol

The server runs on the machine SERVER HOSTNAME and listens for requests on a TCP socket bound to port SERVER PORT. Both constants are defined in the header file, "common.h", provided for you.

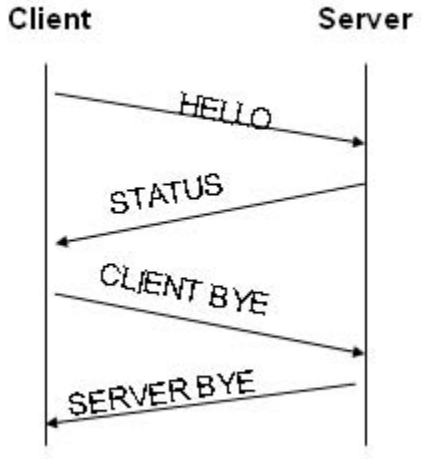

Figure 1: Protocol

The protocol has four types of messages: **HELLO, STATUS, CLIENT_BYE and SERVER_BYE.** Each message is an ASCII string, and consists of multiple fields **separated by whitespace (space (0x20) or newline (0x0a) character)**. The MAXIMUM length of the string is 255.

The protocol outline is given in Fig 1. The client initiates the protocol by sending a HELLO message to the server. The server replies with a STATUS message. The client then sends a CLIENT_BYE message, and the server terminates the connection by sending a SERVER_BYE message. A connection is successful if and only if all of these messages are correctly sent and received. Since we are using TCP for communication in this assignment, you do not have to worry about lost messages; you only need to ensure that all messages are sent correctly (and that you receive and parse messages correctly).

The details of each message are as follows:

1. **HELLO (From the client to the server: Client → Server)**

   The HELLO message has 4 fields EXACTLY in the following order:

   - Magic String
     It MUST set to be MAGIC STRING which is a constant defined in the header file ("cmsc417spring2015"). If the client sends a message which does not start with this magic string, the message should be ignored by your server program.
   - Message Type
     The type string is HELLO to indicate a message type HELLO. The server should be case-sensitive.
   - Login ID
     This field is a string that you choose. You will put your login ID as a argument when you execute the client program.
   - Name
     The last field is your first name. Please do NOT put spaces in your name, even if it contains spaces. You will also put your name as a argument when you execute the client program.

   An example HELLO message might look like this:
   cmsc417spring2015 HELLO cs417000 Alice

2. **STATUS (Server → Client)**

   The STATUS message has 4 fields in the following order:

   - Magic String
     Same as above.
   - Message Type
     Must be set to STATUS.
   - Cookie
     An integer randomly generated by the server (represented in ASCII). Cookie generation is explained later.

- IP Address and Port number
  A string of the form a.b.c.d:e, representing the IP address and port number of the client.

An example STATUS message might be:
cmsc417spring2015 STATUS 42 128.8.128.153:48522

3. **CLIENT_BYE (Client → Server)**

   The CLIENT_BYE message has 3 fields in the following order:

   - Magic String
     The same as above.
   - Message Type
     Must be set to CLIENT_BYE.
   - Cookie
     A string of an integer, set to the value of the cookie sent by the server in the STATUS message for this connection.

   An example CLIENT_BYE message would be:
   cmsc417spring2015 CLIENT_BYE 42

4. **SERVER_BYE (Server → Client)**

   The SERVER_BYE message has 2 fields in the following order:

   - Magic String
     The same as above.
   - Message Type
     Must be set to "SERVER_BYE".

   An example SERVER_BYE message would be:
   cmsc417spring2015 SERVER_BYE

4 Server Program

The command line syntax for a minimal server is given below. The server will take the port as an argument.

```
USAGE:
 ./server [<port>]
```

- The cookie should be generated using the formula:
  (a + b + c + d) 13 mod 1111, where a.b.c.d is the IP address of the client, $0 \leq$ a, b, c, d $\leq$ 255.

- After successful communication, the server MUST print the cookie it generates along with the clients login id, first name, IP address and port number. All this information should be in a single line. An example is:
  555 cs417050 Alice from 128.8.126.208:48542

  Notice that ((128+8+126+208) 13 mod 1111) is 555.

- Your server should not accept spurious input from the clients.

  – We will test your server with non-conforming clients; the server should print out an error message containing the clients IP address and port number also in a single line, as such:
  ****Error** from 128.8.126.133:48522**

  and immediately close the connection when it finds a bad message from the client. It should not breakdown, but continue operating after servicing misbehaving clients. Bad messages are ones that have an incorrect magic string, incorrect message type or too many fields.

  Remember, the cookie sent in the STATUS message has to match the cookie in the CLIENT BYE message for a communication to be successful.

- The server should be able to serve multiple clients. It is acceptable if this is done serially.

- Do NOT print out any other debugging messages. They are useful for you, but not for your TA to grade.

- All output should be printed to stdout. You may use **fflush(stdout);** after every output to stdout.

5 Client Program

The client program is provided. The command line syntax for the client is given below. The client program takes command line arguments corresponding to the login id and first name. The hostname and port specifications are optional. If included, they override the default definition of SERVER HOSTNAME and SERVER PORT in "common.h".

```
USAGE:
 ./client [<hostname> [<port>]] <login id> <first name>
 ./client heaving.csic.umd.edu 9999 csic417050 Alice
```

6 Requirement

- Your code must be -Wall clean on gcc. (For example, when you compile your code, try like this:"gcc -o -Wall server server.c")

- The TA will answer general questions/confusions only, and is not supposed to debug for you. The "This is my code, what could be the problem" type of questions will be ignored.

7 Shell Script

You must also write a shell script that demonstrates functionality of your code. In general, your shell script must do **exactly** the following:

1. Compile your code using your Makefile.
2. Read in **TWO** comma separated lists.
3. The first list should be a list of $< port >, < name >$. Your script should invoke a server instance on all of the ports (one for each port). This file will be the first argument.
4. The second list should be a list of $< host >, < port >, < loginid >, < firstname >< name >$ (with no header) from the file specified as the first argument. This script will be the second argument. Your script should launch a client for each line in this file. When all clients are finished your script should kill all server instances.
5. Log the output of each client to $client. < name > .out$ and the output of each server to $server. < name > .out$

Your shell script must be named $tests.sh$.

8 Hints

- Your server and client programs must all run in the CORE environment.
- Your shell script is free to invoke a ruby script to the the bulk of the work. Ruby is much easier to work with than bash (in my opinion) and it will make killing the server instances much easier.

9 Submission

- Please submit your code to the Submit Server (https://submit.cs.umd.edu/).
- You should upload a zip file which contains the files server.c and common.h (and possibly any other .h files you are using). You can create this file in Unix using:
  zip server.zip server.c common.h test.sh