CSCI 4211 Fall'20 Project: Gopher Chat Room

Last update: 9/21/2020, Copyright: Feng Qian, University of Minnesota

A Deadline

November 30 2020 23:59 CST. A penalty will incur for late submissions (see the syllabus).

B Video Lecture

A video lecture on this project can be found on Canvas: `0923_socket_programming.mp4`.

C Problem Description

In this project, you will practice socket programming by developing a chat room application called GopherChat. It allows multiple users on different hosts to join a virtual chat room where a user can send messages or files to other user(s). GopherChat employs the client-server architecture. A message/file from a client will be first sent to the server, which will then distribute the message/file to the corresponding recipient(s). You will need to implement both the client and server applications on Linux. Your program must be written in C/C++, and must use the TCP socket interface provided by Linux. You are not allowed to use any third-party library or any operating system other than Linux. You may use the standard C library and the C++ Standard Template Library (STL).

This project accounts for 27% of your final grade, plus 10% (as bonus points) of your final grade.

C.1 Required Features

Your GopherChat application must support the following features.

1. *Account registration.* A new user connects to the server and registers a new account. The user will need to provide a username and a password which will be stored on the server side. The lengths of both a valid username and password are between 4 and 8 characters consisting of only letters (case-sensitive) and digits.

2. *Login and logout.* A user with an existing account can log onto the chat room by providing her username and password. The server admits the user if the credential is correct. A user in the chat room can log out at any time. We say a user is *online* if she is currently in the chat room. Your server needs to support up to 32 online users.

3. *Sending and receiving public messages.* An online user can send a *public message*, which will be delivered to *all* online users except the sender herself. All the recipients will see the sender's username and the message content. Alternatively, the sender can choose to make the message *anonymous*. In that case the sender's username will not be sent to or shown to the recipients. The maximum length of a message is 256 bytes. A public message must be delivered and displayed in real time.

4. *Sending and receiving private messages.* An online user can also send a *private message* that is delivered to only one recipient specified by the sender (through the username). The sender can also choose to make a private message anonymous. One cannot send a private message to herself. A private message must also be delivered and displayed in real time. An error message will show up to the sender if the recipient does not exist or is not online. The maximum length of a private message is also 256 bytes. Sending/receiving a message should not block sending/receiving another message.

5. *File transfer.* An online user can send files such as an image or a video clip. A file transfer can be either public (sent to all other online users) or private (deliver to a particular recipient). One cannot send a file to herself. The size limit of a file is 10 MB (10,000,000 bytes). A file transfer cannot be anonymous. Your program should allow multiple transfers from/to the same client to take place simultaneously. The limit of concurrent file transfers (including both upload and download) per client is 8 (*i.e.,* we will not test a scenario where more than 8 concurrent file transfers are taking place). A file transfer should not block a message delivery or another file transfer.

6. *List all online users.* An online user can ask the server to provide a list of all online users (their user names).

7. *Server-side monitoring.* The GopherChat server program should display in real time all messages/files received and sent by the server. Important information to be displayed include the timestamp, the sender, the recipient(s), the message content, and the file size. The server program should also display (both successful and unsuccessful) user registration, logging in, and logging out events.

C.2 Add-on Feature for Bonus Points

To earn bonus points, you are encouraged to implement one add-on feature to GopherChat. Some examples are listed below. Feel free to brainstorm other interesting features.

1. *Message/File Encryption.* Messages/files transferred in plain text are vulnerable to the packet sniffing attack by tcpdump or Wireshark. Consider encrypting the messages, files, and commands to prevent this.

2. *Group chat.* Allow a subset of online users to form a group where a group member can send messages/files to all other online users in the same group. You will need to work out the details such as how to create, join, and leave a group.

3. *Offline Messaging.* Allow sending a message to an offline user. The message will be stored (buffered) on the server. The message will be delivered as soon as the recipient becomes online.

4. Other cool features you can think of.

The add-on feature must not conflict with the required features described in §C.1. You need to describe your implemented add-on feature in the documentation (§C.5). Implementing more than one add-on feature will not give you additional bonus points.

C.3 User Interface and Command Line Arguments

GopherChat uses a command-based UI *i.e.,* users issue a set of simple commands to interact with the application. Your program must support all the following commands. Each command may have zero, one, or more arguments separated by a single whitespace. To support your add-on feature, you may need to further expand the command set.

Command	Description
`REGISTER␣[username]␣[password]`	Register a new account
`LOGIN␣[username]␣[password]`	Log in with an existing account and enter the chat room
`LOGOUT`	Log out and leave the chat room
`SEND␣[msg]`	Send a public message
`SEND2␣[username]␣[msg]`	Send a private message to a user
`SENDA␣[msg]`	Send an anonymous public message
`SENDA2␣[username]␣[msg]`	Send an anonymous private message to a user
`SENDF␣[local_file]`	Send a file publicly
`SENDF2␣[username]␣[local_file]`	Send a file to a user privately
`LIST`	List all online users
`DELAY␣[N]`	A special command that delays for N seconds before executing the next command in the script (see below)

In `SENDF` and `SENDF2`, `local_file` denotes the name of a file that is under the same folder where the client program runs. `local_file` only contains the filename (e.g., `pipe.png`) but not any pathname (e.g., `./pipe.png` or `/home/mario/pipe.png`). In other words, the client cannot send a file that is under a different directory than that of the client program. A received file must also be stored in the same directory where the client program runs, with the same name as specified by the sender. If a file with the same name already exists, overwrite it.

To facilitate our test and your debugging, all commands will be loaded from a script file (instead of being entered from the keyboard) on the client side. Each line in the script corresponds to a

command (the line separator is \n). The commands will be launched by the client sequentially without any pause between them. You must also implement a special command called `DELAY` that makes the client delay for N seconds before executing the next command in the script. N is an integer between 1 and 9999. Note that such a delay does not affect the client's other activities such as performing network transfers and displaying received messages. Below shows an example script. A script may contain up to 10,000 lines (commands).

```
REGISTER␣mario␣foobar00
DELAY␣2
LOGIN␣mario␣foobar00
DELAY␣1
SEND␣hello␣everyone␣this␣is␣mario.
DELAY␣1
SEND2␣luigi␣hi␣luigi,␣look␣at␣this␣cool␣pipe!
SENDF2␣luigi␣pipe.png
DELAY␣9999
```

Your client program takes three arguments: the server's IP address, port number, and the script file name. For example:

```
./client 11.22.33.44 6001 test␣script.txt
```

Your server program takes one argument: the port number. For example:

```
./server 6001
```

Your server program must also be able to clean up its database containing users' login credentials, using a special argument `reset`. After resetting the server, all data generated by the server will be wiped off. This will help you debug your code as you can run the same script multiple times. We will also use this feature to test your code:

```
./server reset
```

Please remember that the client always takes input from a script file and outputs to the screen. In other words, the client program never interacts with the keyboard. The server also outputs to the screen (for server-side monitoring).

C.4 Error Handling

Your program should be able to handle various error situations such as the following.

1. When a client cannot reach the server (*e.g.,* due to a loss of the Internet connectivity or unexpected termination of the server), the client application should print out some error message and exit gracefully, instead of crashing directly.
2. Your server should never crash. When a network I/O error occurs on one client, the server

can assume that client goes offline. However, this should not affect other clients, *i.e.,* the server should continue serving other clients if possible.

3. A file transfer may take a long time to complete. Errors that occur during a transfer must be properly handled. A file transfer must not block messages or other transfers.

4. There are other errors or corner cases you need to consider, such as sending a message/file to a non-existing user, sending a message/file to an offline user (unless you implement the "offline messaging" add-on feature), sending a file that is not in the current directory, providing a wrong password when logging in, or issuing an unknown command or a command with incorrect arguments.

C.5 Documentation

You need to prepare a document describing the following.

1. Provide a short summary of the features (both required and add-on) you have implemented. Describe any known issues/problems with your implementation. Please be honest when describing the limitations of your implementation – our tests will reveal them anyway.

2. If you introduce any additional commands for your add-on feature, give the usage descriptions for those commands (like the table on Page 3).

3. A `Makefile` is preferred if you know how to create one. Otherwise, give instructions on how to compile your code.

D Submit Your Program

Compress all source code files (.c/.cpp/.h), the documentation (.pdf/.txt/.doc/.docx), and the optional `Makefile` into a single file called `chatroom.zip`, `chatroom.bz2`, or `chatroom.tar.gz` (depending on the compression program you use). Do not include any executable in your submission.

We strongly recommend you work with an update-to-date GNU C/C++ compiler (gcc/g++ version 4.3 or above). Do not use other C/C++ compilers that might be incompatible with GNU gcc/g++.

E Test and Grading

We will test your software over the real Internet. The server will be running on an Amazon EC2 instance with Ubuntu 18. We will launch multiple client instances to test your application by letting clients "chat" using various scripts. Note that you do not need to use multiple physical machines to test multiple clients – they can (and should be able to) run on the same host.

You may also be asked to give a demo of your code to the TAs (detailed logistics will be announced later). The following table describes how your final grade will be calculated.

Test Cases	Points
User registration and login/logout	2
Sending and receiving messages	8
Sending and receiving files (no concurrency)	4
Sending and receiving files (with concurrency)	4
List all users	1
Error handling	4
System performance	3
Documentation	1
Multiple users sending files and messages concurrently	6 (bonus)
One add-on feature	4 (bonus)
Total	27 + 10 (bonus)

F Sample Test Cases

You can find the test case files (`*.user*.txt`) in the `testcases` folder.

• **Test Case A.** One new user registers an account with the server and logs in. Then another user registers an account with the server and logs in.

Run this first (`a.user1.txt`):

```
REGISTER mario foobar00
DELAY 1
LOGIN mario foobar00
DELAY 9999
```

After 3 seconds, run this (`a.user2.txt`):

```
REGISTER luigi foobar00
DELAY 1
LOGIN luigi foobar00
DELAY 9999
```

• **Test Case B.** Two users are online. One user sends a short public message.

Run this first (`b.user1.txt`):

```
REGISTER mario foobar00
DELAY 1
LOGIN mario foobar00
DELAY 9999
```

After 3 seconds, run this (`b.user2.txt`):

```
REGISTER luigi foobar00
DELAY 1
LOGIN luigi foobar00
DELAY 1
SEND HELLOWORLD
DELAY 9999
```

• **Test Case C.** Two users are online. One user sends one 1KB file privately to the other user.

Run this first (`c.user1.txt`):

```
REGISTER mario foobar00
DELAY 1
LOGIN mario foobar00
DELAY 9999
```

After 3 seconds, run this (`c.user2.txt`):

```
REGISTER luigi foobar00
DELAY 1
LOGIN luigi foobar00
DELAY 1
SENDF2 mario 1kb.user2
DELAY 9999
```

• **Test Case D.** Three users are online. One user sends seven 1MB files publicly.

Run this first (`d.user1.txt`):

```
REGISTER mario foobar00
DELAY 1
LOGIN mario foobar00
DELAY 9999
```

After 3 seconds, run this (`d.user2.txt`):

```
REGISTER luigi foobar00
DELAY 1
LOGIN luigi foobar00
DELAY 9999
```

After 3 seconds, run this (`d.user3.txt`):

```
REGISTER wario foobar00
DELAY 1
LOGIN wario foobar00
DELAY 1
SENDF 1mb.1.user3
SENDF 1mb.2.user3
SENDF 1mb.3.user3
```

```
SENDF 1mb.4.user3
SENDF 1mb.5.user3
SENDF 1mb.6.user3
SENDF 1mb.7.user3
DELAY 9999
```

G Honor Code

Students must follow the University of Minnesota honor code (`https://regents.umn.edu/sites/regents.umn.edu/files/policies/Student_Conduct_Code.pdf`). This project is an individual assignment, and no collaboration among students is allowed. In no case may your code be copied from another student or a third-party source on the Internet. We will use an anti-plagiarism software to detect code "shared" among students. Any violations of the honor code will be dealt with strictly, including but not limited to receiving no credit for the entire project.