CS3210 – Computer Networks Lab
Prof. Krishna Sivalingam, Dept. of CSE, IIT Madras
Jan-May 2015
Due Date: **Feb 2, 2015, 11PM**. On-Line Submission via Moodle
NO LATE SUBMISSIONS ALLOWED.

1 Assignment description

The objective of this assignment is to implement an FTP client, and an FTP server using the Socket Interface. We will be implementing a minimal client and server in this assignment. The objective of the assignment is to familiarize yourself with writing a network server and a client. **This is an individual assignment – please refer to the honor code regarding assignment work; any violation of this will result in 'U' grade, and other penalties.**

There are two major components to the software: (i) the FTP server, and (ii) the FTP client.

The Internet FTP protocol standard definition is provided in RFC-959 available through the class webpage. This RFC will form the main reference for our protocol implementation. The FTP implementation will be *similar* to the minimal implementation outlined in Section 5.1 of the RFC. That is, it will support ASCII Non-print TYPE, Stream MODE, File STRUCTURE and the commands listed below. Note that we are implementing a really simplified subset of the RFC requirements.

1.1 FTP client

An end user will be running the FTP client to communicate with the server. The client software can be partitioned into these major pieces:

(i) USER INPUT INTERFACE: The user-input interface accepts user commands, processes them, and passes appropriate data to the network interface.

(ii) NETWORK INTERFACE: The network interface is responsible for establishing the required socket(s) with the remote server. It is responsible for accepting the data provided by the USER INPUT INTERFACE above and transmitting it over the socket. It is responsible for reading the responses transmitted by the server, and either storing the data on file locally, or displaying it to the screen.

The client software interface will accept the following commands:

- ls <dirname>
  <dirname> is optional;
  Semantics: Print the list of files in the current directory, of the server.
- cd <dirname>
  <dirname> is optional;
  Semantics: change current working directory of the server.
- lcd <dirname>
  <dirname> is optional;
  Semantics: change current working directory at the *client* if valid directory name and print the current directory. If invalid directory name, print error message.

- ▷ get <filename>
  <dirname> is required;
  Semantics: Retrieve the specified file, if it exists at the server. If server indicates file does not exist, then print error message to screen.

- ▷ put <filename>
  <dirname> is required;

  Semantics: Store the specified file in the server, if it exists in the client directory. If client indicates file does not exist, then print error message to screen.

- ▷ pwd
  Semantics: Print Working Directory

- ▷ quit
  Exit the client (closing sockets, files, etc.)

The role of the client interface is thus to accept the user commands, parse the commands, and generate the appropriate FTP commands that will be passed to the server.

Note that since the server is YOUR process, you do not have to support the USER and PASS commands. That is, when the client connects to the server, the server is ready to accept client commands.

1.2 FTP server

The purpose of the FTP server is to respond to the requests generated by the client. The FTP server is expected to be up and running on the remote machine, listening to a specific port. The FTP server will be similar to the echo server.

The server software contains these major components:

(i) NETWORK INTERFACE: This piece is responsible for receiving requests from the client, and passing it to the COMMAND PROCESSOR.

(ii) COMMAND PROCESSOR: This is responsible for processing the request from the client, running appropriate system commands (e.g. getcwd, readdir, chdir, etc.) and passing the output generated to the NETWORK INTERFACE.

When the server is started, the default directory is called the "root" directory of the server. All client requests will be handled starting from this directory.

The commands that the *server* will support are as follows:

- ▷ NLST <dirname>
  <dirname> is optional
  Directory listing of files in specified directory, or current directory; Send listing to client

- ▷ CWD <dirname>
  <dirname> is optional
  Semantics: If current directory is not "root" , then Change Working Directory, and send the new

directory name to the client
else Send "Illegal CWD Operation" Message to client.

- ▷ RETR <filename>
  Semantics: Send Contents of Specified File to client, if it exists. First, Send a "File Name OK." message to client.
  Then, Send Contents of Specified File to client.
  If file doesn't exist, send Error Message.

- ▷ STOR <filename>
  Semantics: First, Send a "OK to SEND File" message to client.
  Then, Receive Contents of Specified File from client and store it locally.
  After complete file is received, send a "File Received" message to client.

- ▷ PWD
  Semantics: Print working directory – send the directory name to the client

- ▷ QUIT
  Semantics: Logout

The codes generated by the server need not be in full compliance with the RFC specification. Since only YOUR client will communicate with YOUR server, you can simplify the codes.

2 Organization

In our implementation, we will use the same socket for both data and control communication. (In the RFC description, a separate socket is used for data communication. To keep things simple, we are using a single socket for both control and data.)

Here, *control* refers to commands sent by the client, and error codes sent by server. *Data* refers to information sent by the server (e.g, directory listing, file contents, etc.)

The server maintains one socket endpoint for communication with the client for both control and for data. The client, likewise maintains a single socket for both control and data communication with the server.

2.1 End of Data

When the client sends a request, the server may have one or more response messages:

- ▷ The standard response (e.g. 200 Command Successful)

- ▷ For NLST and RETR messages, the server has to send additional data. The question is: How do the client and server agree on End-of-Data? You may use the special string `@$` the end of data. What happens if `@$` appears as part of the data - we will assume for this assignment that this will not occur.

- ▷ Likewise for the STOR message, the client has to send the bytes of the specified file followed by `@$`.

2.2 System Calls

The following system calls will be required at the server side. On Linux, *man command* or *info command* will provide you with relevant information.

- ▷ chdir – man chdir
- ▷ readdir – man 3 readdir
- ▷ getcwd – man getcwd

3 Sample Session

Assume that you have created the files ftpclient.c and ftpserver.c and the corresponding executables in your LAB1 directory. Please use a suitable unique port number for your server. The directory LAB1/FTP-SERVER will be the "root" directory for ftp server purposes.

Under the directory FTP-SERVER, create files a1, a2, a3, and three subdirectories: SD1, SD2, SD3. Under each subdirectory, create four files a, b, c, and d (or whatever be your choice). Let each file created above have at least 3-4 lines of some text.

Under the directory LAB1, create 3 data files – clientdata1, clientdata2 and clientdata3 with 3 or more lines of random text.

There will be two cases that you will test for: (i) server and client running on the same machine. You can invoke the client as: ftpclient localhost <port>; (ii) server and client running on different machines.

```
% cd LAB1/FTP-SERVER
% ../ftpserver 25678 &                -- Server running on nif-c6

% cd LAB1
% ./ftpclient nif-c5 25678           -- Client running on another host
.... Server's initial response.
(ftp) ls
.... Server's response.
(ftp) get a1
...
(ftp) cd SD1
....
(ftp) get b
...
(ftp) get e
...
(ftp) pwd
...
(ftp) cd ..
...
```

```
(ftp) pwd
...
(ftp) cd ..
... (Illegal move?)
(ftp) pwd
...
(ftp) cd SD2
...
(ftp) put clientdata2
...
(ftp) ls
...
(ftp) quit
...
%
```

4 What to Submit

Name your assignment directory as LAB1 (Note: ALL UPPERCASE)

Once you are ready to submit, change directory to the directory above LAB1, and tar all files in the directory with the command:

`tar czf CS1xBabc-Lab1.tgz LAB1`

where CS1xBabc refers to your roll number.

The directory should contain the following files:

- ▷ Client Software Files
- ▷ Server Software files
- ▷ Makefile
  Typing command 'make' at the UNIX command prompt, should generate all the required executables.
- ▷ A Script file obtained by running UNIX command *script* which will record the way you have finally tested your program.
- ▷ a README file containing what port number to use, and instructions to compile, run and test your program.
- ▷ a COMMENTS file which describes your experience with the assignment, suggestions for change, and anything else you may wish to say regarding this assignment. This is your opportunity for feedback, and will be very helpful.

Make sure that all files have been correctly tar-red with the appropriate command. Then, submit the *tgz* file via Moodle.

5 Help

1. WARNING ABOUT ACADEMIC DISHONESTY: Do not share or discuss your work with anyone else. The work YOU submit SHOULD be the result of YOUR efforts. Any violation of this policy will result in an automatic ZERO on the assignment, a 'U' in the course, and other academic penalties.

2. Ask questions EARLY. Do not wait until the week before. This assignment is quite time-consuming.

   Please make use of the office hours of the TA to the fullest extent. If you need to meet him during other times, please send him an email for appointment.

3. The tricky part is to make sure that the client and server are receiving the correct response that they are expecting. Otherwise, client and server could get deadlocked.

4. Implement the solutions, step by step. Trying to write the entire program in one shot, and compiling the program will lead to frustration, more than anything else.

   For example, implement the server COMMAND PROCESSOR first, without any network interface. Make sure that textual input (NLST, PWD, etc.) generates the correct server responses.

   Then, implement the client USER INPUT INTERFACE to accept user level commands (ls, pwd, etc.) and generate commands that the server will understand (NLST, PWD, etc.).

   Now, implement the network sockets at both ends, and test **ONE COMMAND AT A TIME** between client and server.

6 Grading

- ▷ NLST: 25 points
- ▷ RETR: 25 points
- ▷ STOR: 25 points
- ▷ CWD: 8 points
- ▷ PWD: 8 points
- ▷ QUIT: 4 points
- ▷ README and COMMENTS file: 5 points

No Script File: -10 points; Incomplete Compilation: -10 points; NO README: -5 points; NO COMMENTS: -5 points