CS3205 - Introduction to Computer Networks
Even Sem. 2019, Dr. Manikantan Srinivasan
Assignment 3: File Transfer Protocol
Individual Assignment
Due date: March 21, 2019, 11PM, On Moodle

Extension: 15 % penalty for each 24-hr period; Max. of 48-hrs past the original deadline

March 7, 2019

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: When <dirname> is not provided, print the list of files in the current directory, of the

server.
Semantics: When <dirname> is provided, Print the list of files in the directory given in <dirname> at the server if valid directory name. If invalid directory name, print error message to screen.

- cd <dirname>
  <dirname> is optional;
  Semantics: When <dirname> is not provided, change to the user's home directory of the server.
  Semantics: When <dirname> is provided, change to directory given in <dirname> at the server if valid directory name and print the current directory. If invalid directory name, print error message to screen.

- lcd <dirname>
  <dirname> is optional;
  Semantics: When <dirname> is not provided, change to the user's home directory at the *client*.
  Semantics: When <dirname> is provided, change to directory given in <dirname> at the *client* if valid directory name and print the current directory. If invalid directory name, print error message to screen.

- get <filename> <dirname>
  <filename> is required
  <dirname> is optional;
  Semantics: When <dirname> is not provided, Retrieve the specified file, if it exists at the server. If file does not exist, then print error message to screen.
  Semantics: When <dirname> is provided, 1) change to directory in the server given in <dirname>. If directory does not exist, Print error message. 2) if directory is valid retrieve the specified file, if it exists at the server. If file does not exist, then print error message to screen.

- put <filename> <dirname>
  <filename> is required
  <dirname> is optional;
  Semantics: When <dirname> is not provided, Store the specified file in the current working directory at the server, if the file exists in the client directory. If client indicates file does not exist, then print error message to screen.

  Retrieve the specified file, if it exists at the server. If file does not exist, then print error message to screen.
  Semantics: When <dirname> is provided, 1) change to directory in the server given in <dirname>. If directory does not exist, Print error message. 2) Store the specified file in the changed directory at the server, if the file exists in the client directory. If client indicates file does not exist, then print error message to screen.

- pwd
  Semantics: Print Working Directory on to the screen.

- 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 (simple socket example server that was shown in the class, available in the book).

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", <dirname> is provided and is valid, then Change Working Directory, and send the new directory name to the client
  else If current directory is not "root", <dirname> is provided and is invalid Send "Illegal CWD Operation" Message to client.

  Semantics: When <dirname> is not provided, and a) the current directory is not "root", change to "root" directory, and send the directory name to client
  b) the current directory is "root", send the directory name 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.

NOTE: In your report, you must indicate what are the codes defined by you.

2 Organization

In your implementation, you 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 `$` the end of data. If `$` appears as part of the data, then it will be replaced by `\$`; and if a `\` will be replaced by `\\`.
- Likewise for the STOR message, the client has to send the bytes of the specified file followed by `$`. The client has to parse and deal with the special characters above appearing in the middle of the data.

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 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 ASSIGN3 directory. Please use a suitable unique port number for your server. The directory ASSIGN3/FTP-SERVER will be the "root" directory for ftp server purposes.

Under the directory FTP-SERVER, create files assign1, assign2, assign3, and three sub-directories: SD1, SD2, SD3. Under each sub-directory, 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. (Be creative. Creativity will carry credits.)

Under the directory ASSIGN3 in the client side, 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 ASSIGN3/FTP-SERVER
% ../ftpserver 25678 &              -- Server running on host-1 <IP-ADDR1>. \\
                                    --   Both on same machine assume Server \\
                                    -- running on 127.0.0.1 (local interface)

% cd ASSIGN3
% ./ftpclient <IP-ADDR1> 25678      -- Client running on another host,
                                        -- host-2 <IP-ADDR2>
.... Server's initial response.
(ftp) ls
.... Server's response.
(ftp) get assign1
...
(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
...
%
```

Note: - - xxxx given above is a comment to explain/clarify.

4 What to Submit

The platform for this project will be Linux and C/C++ / Java / Python. Create a tar-gz file with name: Assignment3-RollNo.tgz (e.g. Assignment3-CS16B099.tgz) that will contain a directory named Assignment3-RollNo with all relevant files.

The directory should contain the following files:

- Client Software Files
- Server Software files
- A text file obtained by running UNIX command *script* which will record the way you have finally tested your program. One for each modes. i.e., server and client enabled in the same machine, server and client on different machines.
- a README file containing what port number to use, and instructions to compile, run and test your program. NOTE: In the read me file, list the MAC address and IP address of your Client Machine and Server machine.

- Two WIRESHARK capture files. 1) File when the execution is done in local machines. 2) File when the execution is done using different machines.
- A WIRESHARK-README file related to the captures. Which packets constitute the flow for the corresponding exchanges. During evaluation, you would be requested to explain two or three flows using the wireshark, as well as do it real time.
- 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. Ask questions EARLY and start your work NOW. Take advantage of the help of the TAs and the instructor.
2. Submissions PAST the extended deadline SHOULD NOT be mailed to the TAs. Only submissions approved by the instructor or uploaded to Moodle within the deadline will be graded.
3. Demonstration of code execution to the TAs MUST be done using the student's code uploaded on Moodle.
4. NO sharing of code between students, submission of downloaded code (from the Internet, Campus LAN, or anywhere else) is allowed. Code copying will result in a 'U' Course Grade. Students may also be reported to the Campus Disciplinary Committee, which can impose additional penalties.
5. Please protect your Moodle account password. Do not share it with ANYONE. Do not share your academic disk drive space on the Campus LAN.
6. Implement the solutions, step by step. Trying to write the program in one setting may lead to frustration and errors.

6 Grading

Successful working of both Client and Server is observable with the implementation at the Server. Hence weights given based on Server aspects.

- NLST: 20 points
- RETR: 20 points
- STOR: 20 points
- CWD: 6 points
- PWD: 6 points
- QUIT: 3 points
- Text output file, captured using 'script', showing how you tested and output for local, two machine setup - 5 Points.

- WIRESHARK Capture - 10 Points, and
- WIRESHARK README - 5 points
- README and COMMENTS file: 5 points

No Script File: -5 points; No Output File: -5 points Incomplete Compilation: -10 points; NO README: -3 points; NO COMMENTS: -2 points