Creating our own FTP

Programming Assignment #4

Problem Description:

Write two separate programs, client.c and server.c, to implement a simplified version of FTP. The server process should work in connection-oriented and concurrent-server mode. You need to first start the server process in a server host and publish its host name and port number. A user on another host can then issue a command like

%myftp server-host-name server-port-number

to download a file from or upload a file to the server. After accepting the above **myftp** command, the client process should responds with prompt **ftp>**, waiting for user's ftp commands.

Requirements:

1) **myftp** is so named as to tell from the standard ftp command.
2) You need to consider the following ftp commands:

ftp>put filename	to upload a file named filename to the server,
ftp>get filename	to download a file named filename from the server,
ftp>ls	to list the files under the present directory of the server,
ftp>cd	to change the present working directory of the server,
ftp>pwd	to display the present working directory of the server
ftp>!ls	to list the files under the present directory of the client,
ftp>!cd	to change the present directory of the client,
ftp>!pwd	to display the present working directory of the client,
ftp>quit	to quit from ftp session and return to Unix prompt.

3) Other commands except the above ones are considered as invalid ftp commands. When a user inputs an invalid ftp command, your program should respond with "An invalid FTP Command."
4) When put or get a non-existed file, your program should respond with " filename: no such file."
5) Submit your program by emailing it to ece5650@ece.eng.wayne.edu .
6) Late submission is not accepted.

Due Day: November 14 (Thursday), 2002

Pseudocode for Client.c

```
sd=socket(AF_INET, SOCK_STREAM, 0)
get server host name from argv[1] and server port number from argv[2]
connect(sd, &server-socket-address,...)

While (1) {
show ftp>
fgets a ftp command line from keyboard

   if the command is "put a existed file"
        0. send the command to the server
        1. open the file
        2. read the file
        3. write the file to server
        4. close the file

   if the command is "put a non-existed file"
        display "filename: no such file on client"

   if the command is "get a file"
        1. send the command to the server
        2. fgets first line from the server:existed or nonexisted
        3. if existed
                3.1 read the file from the server
                3.2 write the file to the local directory
        4. if nonexisted
                display "filename: no such file on server"

   if the command is "cd ..." , "ls ...", or “pwd”
        1.  send the command to the server
        2.  fgets a reply line from the socket to see if the command is successfully executed
        3.  read from the socket and display correspondingly.

   if the command is "!ls ..." or “!pwd”
        1. call system(command) locally

   if the command is   "!cd   directory"
        1. call chdir (directory) locally. Note that system( ) cannot execute "cd ..."

   if the command is "quit"
        1. close the socket
        2. break or exit

   otherwise: show "An invalid ftp command."
}
```

Pseudocode for child process forked by server.c

```
While (1) {
  fgets a ftp command line from client via stream socket

    if the command is "put a file"
        1. create a file
        2. read the file from socket
        3. write to the file
        4. close the file

    if the command is "get a file"
        1. open the file
        2. if existed
                2.1 send "existed" to client
                2.2 read the file
                2.3 write the file to client
                2.4 close the file
        3. if nonexitested
                send "nonexisted" to client

    if the command is "ls ..." or "pwd"
        1. fp =popen(command,"r")
        2. read the result from fp
        3. if no result from fp, reply "wrong command usage!" to client, otherwise reply "successfully executed!" to client
        4.    send the result to client

    if the command is "cd directory…"
        1. call chdirr(directory).
        2.reply to the client if the command is successfully executed.

    if the command is "quit"
        1. close socket
        2. exit
}
```

Notes: 1) This is only a pseudocode. It doesn't intend to be all-inclusvie.
2) This is only a suggestion. You may follow your own style to write programs.

Some Tips

1) Your program should be able to display an error message when some error occurs. You need to consider such errors:

   client
   - put a file but the file is non-existed locally.
   - invalid ftp commands except for put, get, ls, cd, pwd, and quit.

   server
   - get a file but the file is non-existed on the server.
   - a valid ftp command (ls) but followed by invalid options such as "ls –z" (-z is a wrong options) or a valid ftp command (cd) but followed by non-existed directory, such as "cd abc" (abc is a non-existed directory)

2) This assignment requires you to use a stream socket that has no boundaries between consecutive messages. It is your responsibility to recognize each message from the stream socket.

3) You can use '\n' to as a message delimiter and thus use fgets() to get a message (or a line) from the stream socket. Or you can use the function readline() at page 280 of Stevens textbook, 1st edition.

4) In order to use fgets() to read a line from a stream socket, you need to follow the order:
   ….sd = socket (…);….sd1 = accept(…);…. dup2(sd1, 0);…..fgets(buf, size, stdin);….
   Note that fgets() can only be used for FILE pointer, not for file descriptor.

5) When transferring a file along a socket, you need to indicate the end of file transfer. There are several ways to do so such as: 5.1) server closes and reopen the socket: a bad way. 5.2) Use a special string pattern. But any string pattern may appear in the file text. This is not an encouraged way. 5.3) Tell the opposite process the size of the transferred file. 5.4) Use Ping-Pong protocol. That is, one write( ) corresponds to one read( ) and one reply write( ) on the other end.

6) Remember that a stream socket is bi-directional! After a client sends a chunk of data (say a file) by calling write() many times, the client can read() from the socket immediately to get the response from the server. Don't worry that the client will get back its own data---it is impossible. That is, if client writes something into a stream socket, only server can read it; and vice versa.

7) popen(" ls -l ", "r") is OK, but chdir(" temp") is not OK. That is, popen( ) allows white space in its arguments but chdir( ) doesn't.

8) You can not use popen("cd temp", "r") or system("cd temp"). Instead you can use chdir("temp").