Hyerim Yong (hy1602)
Mathilde Simoni (mps565)

Test Code for Project 1: File Transport Protocol

Outline

- Initialization
- PORT
- Login Authentication (USER, PASS)
- STOR
- RETR
- LIST/!LIST
- PWD/!PWD
- CWD/!CWD
- QUIT
- Special Case: Multiple concurrent Users

INITIALIZATION

After compiling the programs with the makefile, the server program and the client program are initiated by typing the following:
./server
./client

Server

```
[(base) mathildesimoni@x86_64-apple-darwin13 ComputerNetworks_Project1 % make      ]
 gcc -c ftp_server.c -w
 gcc ftp_server.o -o server -w
 gcc -c ftp_client.c -w
 gcc ftp_client.o -o client -w
[(base) mathildesimoni@x86_64-apple-darwin13 ComputerNetworks_Project1 % ./server  ]
 Server started and is listening...
```

Client

```
[(base) mathildesimoni@x86_64-apple-darwin13 ComputerNetworks_Project1 % ./client
 To display available commands, enter "commands"
 220 Service ready for new user.
 ftp> commands
 Available commands:
 - USER username: to start authentification
 - PASS password: to finish authentification (after USER command)
 - STOR filename: upload a local file from current client directory to current server dir
 ectory
 - RETR filename: download a file from current server directory to current client directo
 ry
 - LIST: list all the files under current server directory
 - !LIST: list all the files under current client directory
 - CWD foldername: change current server directory
 - !CWD foldername: change current client directory
 - PWD: display current server directory
 - !PWD: display current client directory
 - QUIT: quit the FTP session and closes the control TCP connection
 ftp>
```

PORT

This command is automatically sent to the server when the client types a **LIST**, **STOR** or **RETR** command. For example, if the user types LIST, the client will first send the command **PORT N + i** to the server, when N is the client port for the control connection (randomly assigned) and i characterizes the ith data connection. The server replies with **200 PORT command successful** before starting the data transfer on port 20 (for data connection)

Client:

```
ftp> LIST
200 PORT command successful.
150 File status okay; about to open data connection.
. .. filename_client.txt file2.txt
226 Transfer completed.
```

Server:

```
PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.
```

Login Authentication (USER, PASS)

If the user types **USER bob**, the server checks within its users.txt file that the username exists. If it does exist, it sends to the client **331 Username OK, need password**. Else, it sends **530 Not logged in.**

If the username is checked, the user sends **PASS donuts**, in which the server checks if there is a password with the previous username that was sent. If it exists, the server sends to client **230 User logged in, proceed.** Else, it sends **530 Not logged in.**

Client:

```
cailynyong@Cailynui-MacBook-Pro ComputerNetworks_Project1 % ./client
To display available commands, enter "commands"
220 Service ready for new user.
ftp> USER asdf
530 Not logged in.
ftp> USER bob
331 Username OK, need password.
ftp> PASS aaaa
530 Not logged in.
ftp> PASS donuts
230 User logged in, proceed.
ftp>
```

Server:

```
New client connected.

USER command received.
Checking for usernames...
Failed login. Try again.

USER command received.
Checking for usernames...
Valid username.

PASS command received.
Checking for passwords...
Failed login. Try again.

PASS command received.
Checking for passwords...
User has successfully logged in.
```

Only after the user is logged in, the user is able to handle other commands such as LIST, STOR, RETR, etc.

Error cases

If the PASS command is typed before the USER command, the server sends to the client **503 Bad sequence of commands.**

If the user is not logged in and tries to send other commands, the client gets an error message **530 Not logged in.**

Server:

```
New client connected.

PASS command received.
Bad sequence of commands.
```

Client:

```
cailynyong@Cailynui-MacBook-Pro ComputerNetworks_Project1 % ./client
To display available commands, enter "commands"
220 Service ready for new user.
ftp> PASS donuts
503 Bad sequence of commands.
ftp> LIST
530 Not logged in.
ftp>
```

STOR

The STOR command is to send a file from the client directory to the server directory. For example, for a file **filename_client.txt** that exists in **client_directories/bob/filename_client.txt**, if user types **STOR filename_client.txt** :

1. First check that this file exists within the client directory. Using !LIST, we observe filename_client.txt.
2. Then do STOR filename_client.txt, in which the server replies with PORT establishment first, gets ready for data connection with **150**, and then **226 Transfer completed**.
3. To check if the file is actually copied, check LIST to check the files within the server directory and see that filename_client.txt has been updated.

Client:

```
ftp> !LIST
.
..
filename_client.txt
dummy-clientfiles
ftp> LIST
200 PORT command successful.
150 File status okay; about to open. data connection.
. .. file2.txt file3.txt dummy-serverfiles .DS_Store filename_server.txt
226 Transfer completed.
ftp> STOR filename_client.txt
200 PORT command successful.
226 Transfer completed.
ftp> LIST
200 PORT command successful.
150 File status okay; about to open. data connection.
. .. filename_client.txt file2.txt file3.txt dummy-serverfiles .DS_Store filename_
server.txt
226 Transfer completed.
```

Server:

```
PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.

PORT command received.
Connected to client on new port
STOR command received.
Opening the file

End of file now
File transferred to client.

PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.
```

Error Cases

If the file doesn't exist, **550 No such file or directory** error is sent to the client.

```
ftp> STOR nofile.txt
200 PORT command successful.
client path: client_directories/bob/nofile.txt
fopen-file: No such file or directory
550 No such file or directory.
```

RETR

Similar to the STOR command, it sends a file from the server to the client directory. For example, for a file2.txt that exists in **server_directories/bob/file2.txt**, if user types **RETR file2.txt:**

1. The server first checks if the file exists within the directory and starts sending the content line by line to the client. We can check with LIST to see if file2.txt exists.
2. Then send **STOR file2.txt**, in which the server establishes a PORT, gets ready for data connection with **150 file status ok. About to open data connection**, and sends the file to the client. If transfer complete, server sends to client **226 Transfer Completed.**
3. Check by using **!LIST** to see that file2.txt has been copied into the client directory.

Client:

```
ftp> !LIST
.
..
filename_client.txt
dummy-clientfiles
ftp> LIST
200 PORT command successful.
150 File status okay; about to open. data connection.
. .. filename_client.txt file2.txt file3.txt dummy-serverfiles .DS_Store filename_
server.txt
226 Transfer completed.
ftp> RETR file2.txt
200 PORT command successful.
150 File status okay; about to open. data connection.
226 Transfer completed.
```

Server:

```
PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.

PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.

PORT command received.
Connected to client on new port
RETR command received.
sending data now
File transferred to client.
```

Error Cases

If the file doesn't exist, **550 No such file or directory** error is sent to the client.

```
ftp> RETR file4.txt
200 PORT command successful.
550 No such file or directory.
```

Other Cases

Both STOR and RETR support files other than txt files, such as binary files. An example is **RETR binary_file.bin**, which exists within the server directory.

1. First check with LIST that binary_file.bin exists in server directory.
2. Doing **RETR binary_file.bin**, server establishes PORT connection, gets ready for data connection with 150, then sends **226 Transfer completed.**
3. To check that binary file is copied in client directory, check with !LIST and see that the file has been updated.

Server:

```
PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.

PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.

PORT command received.
Connected to client on new port
RETR command received.
sending data now
File transferred to client.
```

Client:

```
ftp> LIST
200 PORT command successful.
150 File status okay; about to open. data connection.
. .. binary_file.bin resume.txt homework
226 Transfer completed.
ftp> !LIST
.
..
resume.txt
new_homework
ftp> RETR binary_file.bin
200 PORT command successful.
150 File status okay; about to open. data connection.
226 Transfer completed.
ftp> !LIST
.
..
binary_file.bin
resume.txt
new_homework
```

LIST, !LIST

LIST command connects to the port and sends all the files within the server directory to the client. !LIST command prints all the files within the client directory to the client.

Server:

```
PORT command received.
Connected to client on new port
LIST command received.
File transferred to client.
```

Client:

```
ftp> LIST
200 PORT command successful.
150 File status okay; about to open. data connection.
. .. filename_client.txt file2.txt file3.txt dummy-serverfiles .DS_Store filename_
server.txt
226 Transfer completed.
ftp> !LIST
.
..
filename_client.txt
file2.txt
dummy-clientfiles
```

PWD, !PWD

For PWD, the server sends to the client the current server directory.

For !PWD, the client prints the current client directory.

Server:

```
PWD command received.
Current server dir: server_directories/bob/
```

Client:

```
ftp> PWD
257 server_directories/bob/
ftp> !PWD
Current client directory: client_directories/bob/
```

CWD, !CWD

CWD is to change the current server directory.
For instance, say alice logged it and is in the server directory server_directories/alice/. She wants to change the current directory to server_directories/alice/homework/. She must type **CWD homework** and the server will reply with **200 directory changed to server_directories/alice/homework/**.
In addition, if Alice wants to go back to the previous directory server_directories/alice/, she types **CWD ..** and the server replies with **200 directory changed to server_directories/alice/**.
Using **PWD** command, we can check that the server directory was updated

Client

```
230 User logged in, proceed.
ftp> PWD
257 server_directories/alice/
ftp> CWD homework
200 directory changed to server_directories/alice/homework/
ftp> PWD
257 server_directories/alice/homework/
ftp> CWD ..
200 directory changed to server_directories/alice/
ftp> PWD
257 server_directories/alice/
ftp>
```

Server

```
PWD command received.
Current server dir: server_directories/alice/

CWD command received.
Server directory updated.

PWD command received.
Current server dir: server_directories/alice/homework/

CWD command received.
Server directory updated.

PWD command received.
Current server dir: server_directories/alice/
```

Error Cases

If the user enters a directory which doesn't exist, the server replies with **550 No such file or directory**. Moreover, Alice is not allowed to navigate in the server directory of other users. Therefore, if she is in the directory server_directories/alice/ and types **CWD ..** the server will also reply with **550 No such file or directory.**

Client

```
ftp> PWD
257 server_directories/alice/
ftp> CWD invalid_dir
550 No such file or directory.
ftp> CWD ..
550 No such file or directory.
ftp>
```

Server

```
PWD command received.
Current server dir: server_directories/alice/

CWD command received.
Error: could not change current server directory.

CWD command received.
Error: could not change current server directory.
```

The !CWD behaves similarly to change the current client directory

Client

```
ftp> !PWD
Current client directory: client_directories/cam/
ftp> !LIST
.
..
4-year-plan
cv
ftp> !CWD cv
Client directory updated: client_directories/cam/cv/
ftp> !PWD
Current client directory: client_directories/cam/cv/
ftp> !CWD ..
Client directory updated: client_directories/cam/
ftp> !PWD
Current client directory: client_directories/cam/
ftp> !CWD ..
Error: could not change current client directory
ftp> !CWD invalid_directory
Error: could not change current client directory
ftp>
```

Server
There is no server interaction for the **!CWD**, **!PWD** and **!LIST** commands because they are implemented locally in the client program,

QUIT

If the user wants to disconnect, they must type the **QUIT** command. The server replies with **221 Service closing control connection.** The TCP connection is closed and the client program terminates.

Client

```
ftp> QUIT
221 Service closing control connection.
(base) mathildesimoni@x86_64-apple-darwin13 ComputerNetworks_Project1 %
```

Server

```
QUIT command received.
Client disconnected
```

Special Case: Multiple Concurrent Users

This FTP Application Protocol handles multiple concurrent users. The server is able to maintain multiple FTP sessions and provide access to multiple users as it can be seen below:

See the video submitted on Brightspace.