Sockets Programming

Networks and Distributed Systems

2016
*Practice last year

INDEX

1) FTP protocol description:

FTP (“File Transfer Protocol”) in computer science, is a network protocol for transferring files between computers connected to a TCP (“Transmission Control Protocol”) based on client--server architecture systems. From a client computer you can connect to a server to download files from it or to send files, regardless of operating systems use on each computer.

FTP service is offered by the application layer model layer TCP/IP network to the user, usually using the network port “20” an “21”.

In the model, the interpreter protocol (IP) user initiates the control connection on port 21. The standard FTP commands are generated by the IP user and transmitted to the server process via control connection. Standard replies are sent from the IP address of the user IP connection control in response to the command server.

These FTP commands specify parameters for the data connection (data port, transfer mode, type of representation and structure) and the nature of the operation on the file system (to store, retrieve, add, delete, etc.). The data transfer process (DTP) of user or other process in place, you must wait for the server to initiate the connection to the specified data port (port 20 in standard mode or active) and transfer data according to the parameters they have been specified.

2) Implementation of the socket description

We will implement a simple FTP server using the TCP protocol. Recall that the socket is the combination of the IP address plus the TCP port.

The server is defined in the "define_socket_tcp" function in this role structure "sockaddr_in" which contains data concerning the socket is defined. You must use the "socket" function that you miss the family stream socket and to return us to the identifier being less than zero wrong socket implementation.

Once the structure we have created is filled, we use the 'bind' function to associate the socket system. This "define_socket_tcp" function returns the socket handle.

The "define_socket_tcp" function is implemented in the "FTPServer.h" header that corresponds to the server implementation. This FTPServer class mounts the server and is responsible for monitoring, connect, etc stop the client class can be seen below. To perform these functions contains procedures such as "stop" "run" as we have said customer controls.

The client is implemented in the "ClientConnection" which will contain a function to connect to the server, which we call "connect_tcp" class. In this

role structure "sockaddr_in" is defined. With "memset" function we clean the structure. With the "memcpy" function reserve N bytes in memory.

The customer class contains procedures "stop" and "WaitForRequest" and go into detail on the latter.

In the "WaitForRequest" are implemented all commands that the client can use.

Commands:

1. USER = read on screen a number of user and if applicable proceeds to ask the password, otherwise it returns an error.
2. PASS = read on screen and the user's password to login proceeds if applicable, otherwise it returns an error.
3. PWD = displays the current directory, for this we use the "getcwd" function that gives us the directory where we are
4. PORT = is used for active mode through port 21. We read from file the IP and port. We logical operations and logical shift. We connect to the server.
5. PASV = is used for passive mode. We use the "getsockname" for which returns the address of the socket. We pass by the socket handle parameter, the sockaddr_in structure and socklen_t. If the result returned by the function is zero, enter the passive mode, otherwise shows an error message.
6. CWD = change directory. As we have only one directory, the root, never change. We use the "getcwd" function to read the current directory will be passed as argument to the function "chdir" used to move between directories.
7. = STOR command to upload files to the server if the file does not exist create it and if not overwritten. We used block size of 512 bytes and created a buffer to that block size. Flow control file using "fopen", "fwrite", "fclose" We will write to the file block by block with "fwrite".
8. SYST = command that simply indicates the type of UNIX
9. TYPE = message OK
10. DELAY = command to download files. Ask the user to enter the argument, which is the name of the file, and if there proceed to download. Again we use Blocks 512 and a buffer with this size. With "fread" We will read the file and send the blocks with "send" We again use flow control files with corresponding flow errors.
11. QUIT = command to the customer leaves.
12. LIST = command to list directories in the root directory, use the same flow control but with FILE * Dir--specific files. With the function "readdir" read the directory and until we get to the end screen printing for struct dirent that stores the names of the directories.
13. FEAT = command not implemented. Necessary for MAC machines
14. SIZE = command not implemented. Necessary for MAC machines
15. EPSV = command not implemented. Necessary for MAC machines

3) Guide for the compilation and steps to execute the server program.

First of all, to make use of the server, we have to compile the code. For that, we need to extract the source files and save them in the same folder. Then, we will need to open two terminals, one for the server and another for the client. On the terminal server, we will write "make" to compile the file and then run it as shown below:

```
$ make
g++   -g   -std=gnu++0x      ClientConnection.cpp   FTPServer.cpp
ftp_server.cpp -o ftp_server -lpthread
$ ./ftp_server
```

Now, we have the server running and we have to connect with the client. To do this, write the following at the client terminal:

```
$ ftp -d
```

Then, we have to open the port connectio with the server:

```
ftp> open localhost 2121
```

At this time, the service is already available and we have to enter the username and password. In our case it is "root" and "12345", respectively.

```
Name (localhost:user): root
---> USER root
331 User name ok, need password
Password: (12345)
---> PASS XXXX
230 User logged in, proceed.
ftp>
```

Finally, we have everything ready and functioning properly, and you can run the commands, such as "get", to download a file, or "put" to upload a file, or maybe "quit", to close the connection.

4) Practical cases and tests

In the previous section we explained theoretically some commands, now illustrate a practical case.

Connecting to the server:

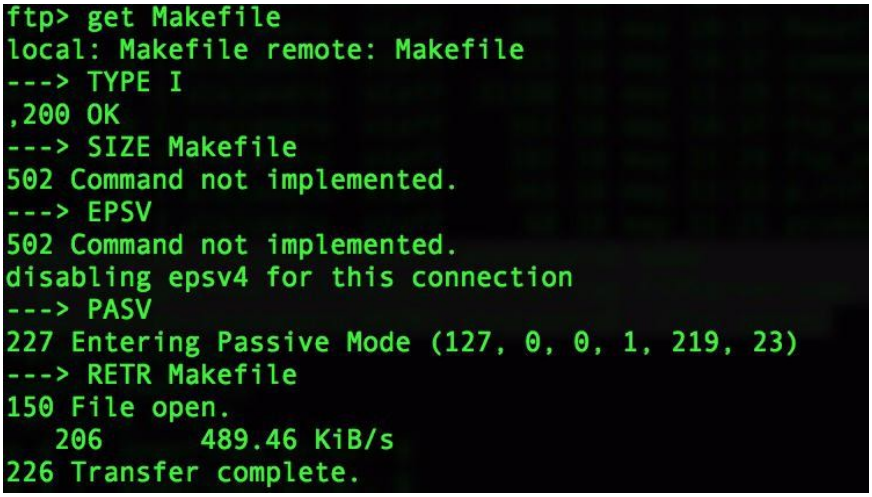
```
MacBook-Pro-de-Alejandro:REDES alejandro$ ftp -d
ftp> open localhost 2121
Trying ::1...
ftp: Can't connect to `::1': Connection refused
Trying 127.0.0.1...
Connected to localhost.
220 Service ready
ftp_login: user `<null>' pass `<null>' host `localhost'
Name (localhost:alejandro):
```

Service authenticating:

```
Name (localhost:alejandro): root
---> USER root
331 User name ok, need password
Password:
```

```
Name (localhost:alejandro): root
---> USER root
331 User name ok, need password
Password:
---> PASS XXXX
230 User logged in, proceed.
---> SYST
215 UNIX Type: L8
Remote system type is UNIX.
Using binary mode to transfer files.
---> FEAT
502 Command not implemented.
```

Downloading a file of the server:



Uploading a file to the server:

5) Source Code

Source code's included in the folder.

6) Working Group

Practice carried out by the students:

-- Néstor García Moreno