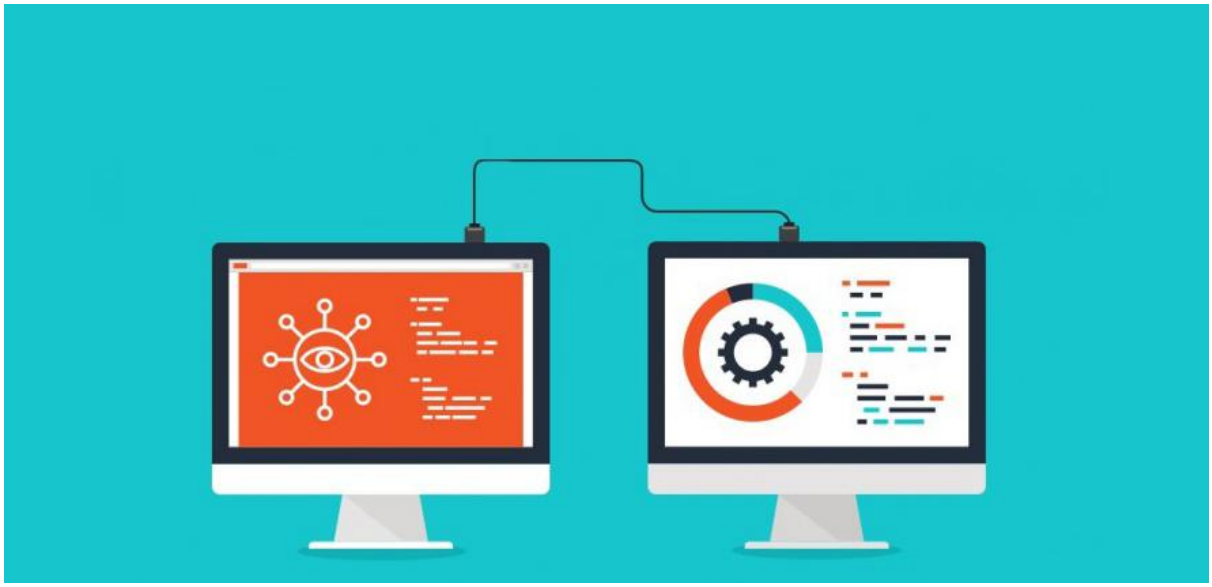
Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Socket Programming - Group 35

Report



José Javier Díaz González
(alu0101128894@ull.edu.es)

Kevin Solís Sierra
(alu0101243344@ull.edu.es

Index:

1. A brief description of the developed application **2**
2. A description of the developed protocol **2**
3. A guide for the compilation of the source code and the necessary steps to execute the server program..... **3**
4. Test cases **4**
5. Appendix: Source code **6**

1. A brief description of the developed application

An FTP Server is a piece of software that is running on a computer and uses the File Transfer Protocol to store and share files. The application aims at interaction between a client and the server through a series of commands implemented in standard 959. The commands we will use are the following: USER, PASS, PWD, PORT, PASV, CWD, STOR, SYST, TYPE, RETR, QUIT, LIST, and in order to control the result of this commands, we need to print a number associated with it. For example, if we are succeed with USER, we will get the 331 code, as we can see in next screenshots.

In rfc959 we can find the explanation on which we base to implement each of the commands. However, here we have a little description about them.

- ***USER***: Start a session in the FTP with a username given by us. After that, we are asked for a password.
- ***PASS***: This command receives a password that we wrote and check if it is correct.
- ***PWD***: Print the current working directory.
- ***PORT***: Makes an active connection with a port and an address given by the user.
- ***PASV***: Requests the server to "listen" on a data port and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.
- ***CWD***: Permits to change the directories where we want to
- ***STOR***: Permits to the user transfer a file to the server.
- ***SYST***: Tells the type of the remote host operating system.
- ***TYPE***: Indicates how is the data representation type used for data transfer and storage.
- ***QUIT***: Closes the current FTP session.
- ***RETR***: Permits to the user get a file from the server.

2. A description of the developed protocol: FTP

FTP or file transfer protocol is executed above of TCP. It uses two TCP parallel connections(sockets) to transfer a file, a control connection and a data connection. The control connection sends information between two hosts, as the user identification, its password, commands to modify the remote directory and to put/get files as we'll see with the development of the report. Two different file transfer modes can be distinguished:

• **Active mode:** The client connects from a unprivileged port N to the server command port (port 21). Then, the client starts listening to port M and sends the FTP command PORT M to the FTP server. The server will then connect back from port 20 to port M of the client.

The main problem with this procedure is that the server connects back to the client, which often is a problem with firewalls that drop not known incoming connections.

• **Passive mode:** To solve the issue that arises when using active mode, the clients support passive mode. In passive mode, the client initiates both connections (control and data). When opening an FTP connection, the client opens two random unprivileged ports locally. The first port contacts the server on port 21. Then the client issues a PASV command and the server responds with a port number. After that the client connects to that port in order to transfer the data.

3. A guide for the compilation of the source code and the necessary steps to execute the server program

- *Step1: Compilation*

Firstly we have to open the terminal on the directory where we have our source code. Then, we only need to write "make" and push return. The command "make" will take the .mk file and it will compile the code files, generate its associated object files (.o) and the executable if there are not mistakes in our project.

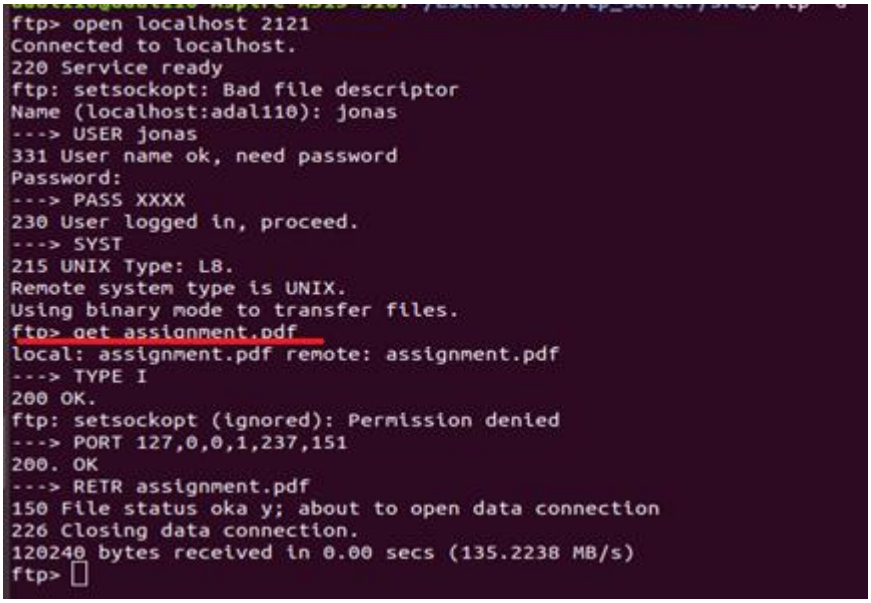
- *Step 2: Execution*

In order to get a real connection client-server, we need to open a new shell, where we have to write the ftp command with the debug flag (-d) to be able to read some messages sent by the client, that normally are hidden.

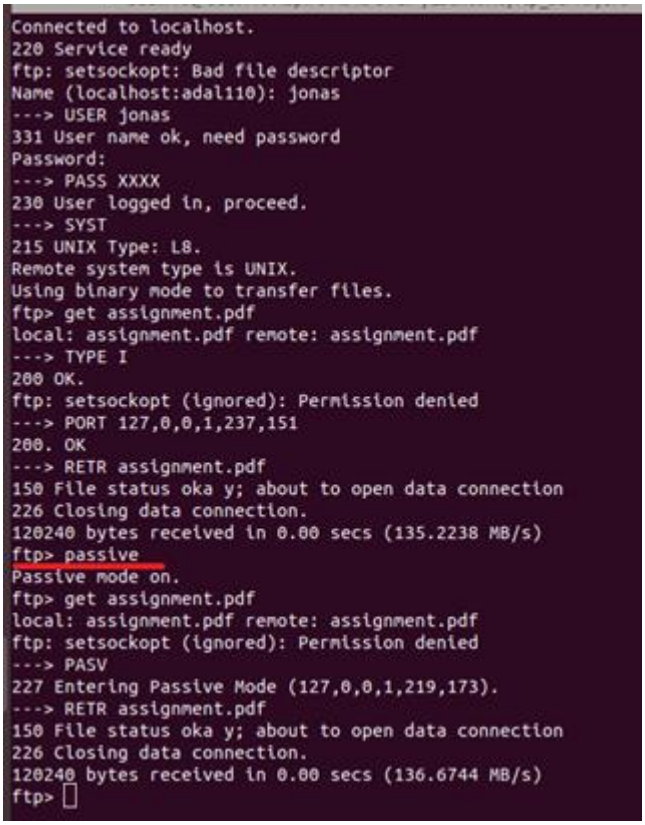
So now we have to run our executable writing "*./ftpserver*". After that, in the shell where we run ftp -d, we must to write "*open localhost 2121*" to set the connection between client and server with the port *2121*. Once we do this steps, we can run our application.

4. Test Cases

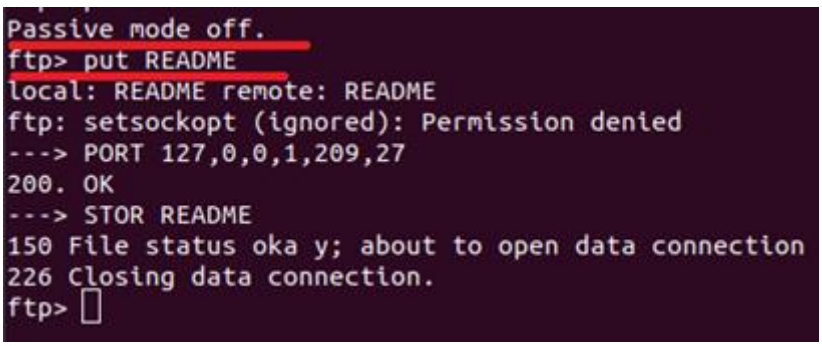
Case 1: Getting a file in “active” mode



Case 2: Getting a file in “passive” mode



Case 3: Putting a file in “active” mode



Case 4: Putting a file in “passive” mode

```
ftp> passive
Passive mode on.
ftp> put README
local: README remote: README
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,149,191).
---> STOR README
150 File status oka y; about to open data connection
226 Closing data connection.
ftp>
```

Case 5: ls in “active” mode

```
Passive mode off.
ftp> ls
---> TYPE A
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PORT 127,0,0,1,228,245
200. OK
---> LIST
125 List started OK.
FTPServer.cpp
ftp_server
Makefile
FTPServer.h
common.h
..
ClientConnection.cpp
ClientConnection.h
README
.
assignment.pdf
ftp_server.cpp
WARNING! 12 bare linefeeds received in ASCII mode
File may not have transferred correctly.
250 List completed successfully.
ftp>
```

Case 6: ls in “passive mode

```
ftp> passive
Passive mode on.
ftp> ls
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,156,97).
---> LIST
125 List started OK.
FTPServer.cpp
ftp_server
Makefile
FTPServer.h
common.h
..
ClientConnection.cpp
ClientConnection.h
README
.
assignment.pdf
ftp_server.cpp
WARNING! 12 bare linefeeds received in ASCII mode
File may not have transferred correctly.
250 List completed successfully.
```

5. Appendix: Source code

- **common.h:** Common functions used by other files of the source code.
- **ftp_server.cpp**: Main program.
- **FTPServer.h**: C++ class definition of the FTP server.
- **FTPServer.cpp**: C++ class implementation of the FTP server.
- **ClientConnection.h**: C++ class definition of the ClientConnection class that manages the information between the server and each client.
- **ClientConnection.cpp**: Implementation of the ClientConnection class. Here we have the implementation of the commands that we run during the execution of the FTP simulation.
- **Makefile**