Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Sockets Programming

Practice 3 Report

Subject: Networks and Distributed Systems

Sebastián André Porto Specht
alu0101494265@ull.edu.es
Marco Napierski
alu0101469317@ull.edu.es
Enrique Reyes Baute
alu0101471990@ull.edu.es

Index:

Brief description of developed application

Based on the code snippets provided, it appears that the developed application is a simple FTP client that allows users to log in to an FTP server, send various commands to the server (e.g. "LIST", "STOR", "RETR"), and transfer files to and from the server.

The client uses a TCP connection to communicate with the server and sends commands to the server using the "SendCommand" function. The client also listens for data from the server over the data connection using the "ReceiveData" function.

The "LIST" command is used to list the files in the current directory on the server, and the "STOR" and "RETR" commands are used to upload and download files, respectively. The "PORT" and "PASV" commands are used to establish a data connection between the client and server for file transfers.

The client also supports basic authentication by sending the username and password to the server using the "USER" and "PASS" commands. If the username and password are correct, the server allows the user to perform various actions, including file transfers.

Overall, the developed application is a basic FTP client that provides users with a simple interface for transferring files to and from an FTP server.

Description of developed protocol (FTP)

The FTP (File Transfer Protocol) is a standard network protocol used for transferring files between servers and clients over a TCP/IP network. The protocol allows users to transfer files in both directions, upload and download, and supports both text and binary files.

FTP works on the client-server architecture, where the client initiates the connection with the server to transfer files. The client sends a request to the server to establish a connection, and once the connection is established, the user can send commands to the server to perform various operations such as listing files, changing directories, uploading files, and downloading files.

FTP uses two separate channels to transfer files: the control channel and the data channel. The control channel is used to send commands and responses between the client and server, while the data channel is used to transfer the actual files.

FTP provides several authentication methods to secure the data transmission, including anonymous FTP for public file sharing and user-authenticated FTP for private file transfers. Additionally, FTP supports different transfer modes such as ASCII, which is used for text files, and binary mode, which is used for non-text files such as images or executable files.

FTP supports two modes of data transfer: Active mode and Passive mode. These modes determine how the data connection is established between the client and server.

In Active mode, the client initiates the connection to the server's port 21 (command port), and the server responds by establishing a data connection back to the client on a port number greater than 1023. This mode requires the client to have a publicly accessible IP address and a firewall configuration that allows incoming connections on the data port. This can be problematic in situations where the client is behind a NAT (Network Address Translation) firewall or a proxy server.

In Passive mode, the client initiates both the command and data connections to the server. The client sends a PASV command to the server, which responds by providing an IP address and a port number for the client to use to establish the data connection. The client then initiates the data connection to the server on the specified IP address and port number. This mode is generally preferred over Active mode because it is more firewall-friendly and avoids issues with NAT.

Guide for the compilation of source code & necessary steps to execute the server program

For the compilation of the source code we will use Makefile. Before compiling make sure that you have the following packages installed on your system:
- gcc
- g++
- make

The Makefile file will look like this:
all: ftp_server

ftp_server: ClientConnection.cpp FTPServer.cpp ftp_server.cpp
g++ -g -std=gnu++0x ClientConnection.cpp FTPServer.cpp ftp_server.cpp -o ftp_server -lpthread -std=c++17 -lstdc++fs
(This last two are for the correct functionality of std::experimental::filesystem library and instances)

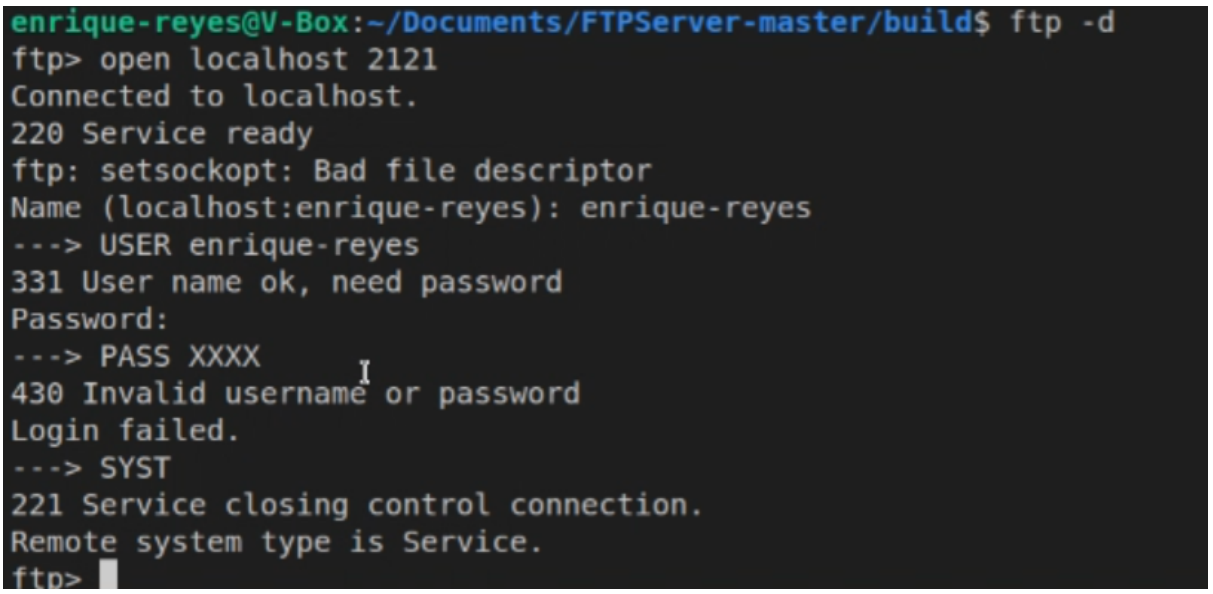
clean:
*$(RM) ftp_server *~*

The steps for the compilation are accessing through the cd command to the directory where the source code is and then type make to compile it. If you want to delete it just type make clean.

For the execution of the server program, you will have to type /ftp_server when the program is already compiled to open and use it.
Now **in another terminal** you would type **ftp -d** as shown in the "Test cases" part and connect to the ftp server previously created with **open localhost 2121**.
Input a user, then input the password which will be **passwd** and use commands such as, ftp> **passive** , ftp> **get README** ...

Test cases

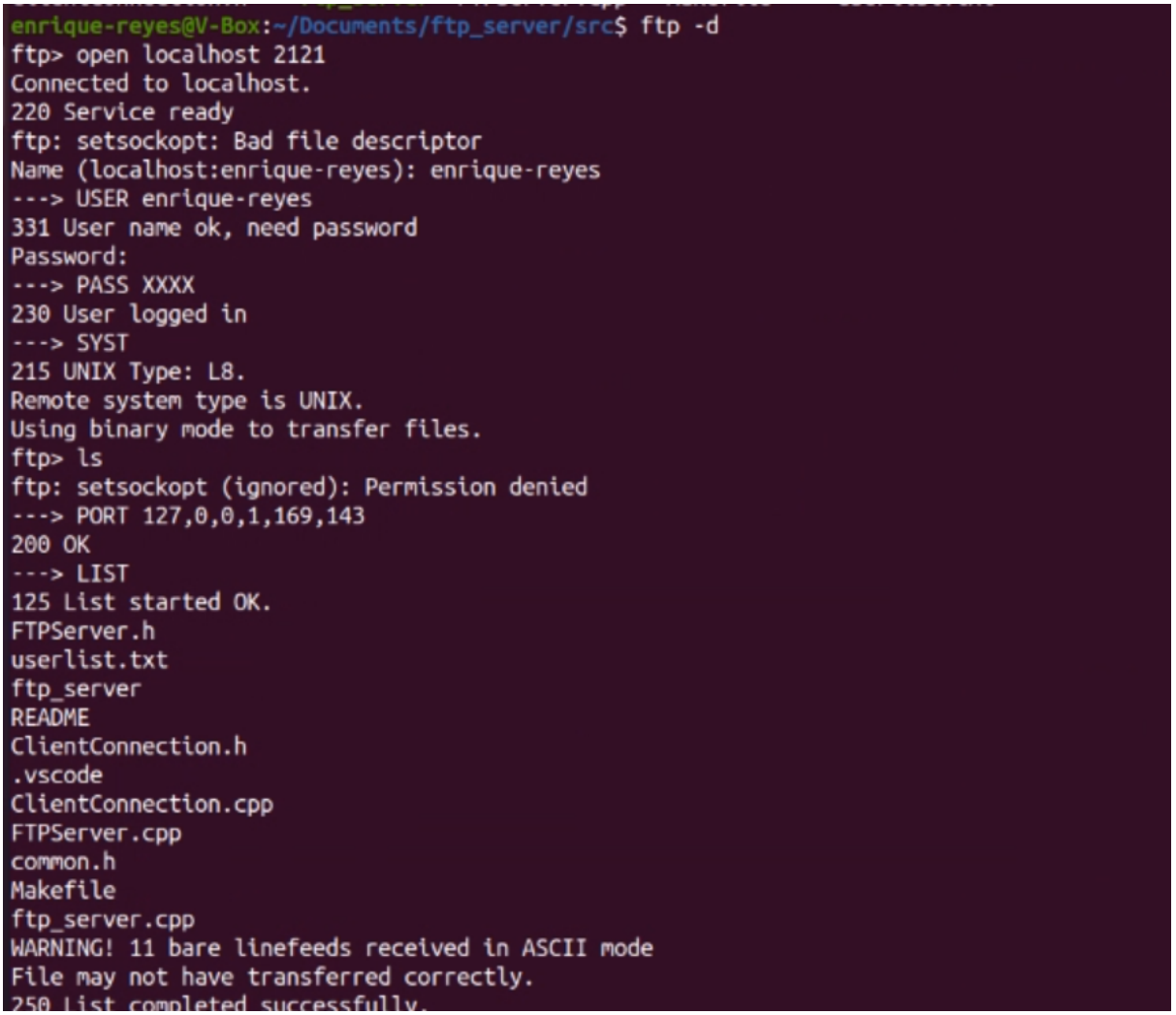
LOGIN FAILED



PUT A FILE IN PASSIVE MODE

```
ftp: setsockopt: Bad file descriptor
Name (localhost:enrique-reyes): enrique-reyes
---> USER enrique-reyes
331 User name ok, need password
Password:
---> PASS XXXX
230 User logged in
---> SYST
215 UNIX Type: L8.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pasive
?Invalid command
ftp> passive
Passive mode on.
ftp> put README
local: README remote: README
---> TYPE I
200 OK
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,174,23).
---> STOR README
150 File status okay; about to open data connection.
226 Closing data connection.
ftp>
```

COMMAND LS IN ACTIVE MODE



Appendix: Source code

Google Drive link: [ftp_server.tgz]