

FILE TRANSFER APPLICATION

ELEN4017: Network Fundamentals

Kishan Y Narotam (717 931) & Matthew van Rooyen (706 692)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: The following report presents the design, implementation and testing of a File Transfer Application. The system consists of a server and a client that requires a simple graphical user interface. The system is deemed a success as the minimum requirements stated in RFC 959 were met. The client is able to display and traverse directories and the contents, upload and download various file types to and from the server and responds with the correct response message. The client is tested against multiple FTP servers and Wireshark is used to confirm the successful results. The overall system is critically analyzed and its limitations and problems are presented. Future development of the system includes improving the functionality of the client and server.

Key words: File Transfer Protocol, Client-Server modal, networking, Wireshark

1. INTRODUCTION

The File Transfer Protocol (FTP) is a set of systemic rules that allow networked computers to communicate with one another on the basis of a client-server model [1]. The protocol communicates over a TCP/IP network and is used to transfer files to and from one another [2]. The objectives of FTP are to promote sharing of computer programs and/or data; utilize the use of programs on remote computers; to protect a user from variations in file storage systems among hosts; and to have reliable data transfer that includes efficiency [3].

A File Transfer Application consists of two components, a FTP server and a FTP client. In the report that follows, the design, implementation and testing of a File Transfer Application is presented. This includes an overview of the system, how the client and server communicate with one another, a critical analysis of the overall system and results from various testing techniques and subsequently how the work was divided among the group.

2. SYSTEM OVERVIEW

The File Transfer Application is required to follow certain guidelines as listed in the Request for Comments: 959 document [3]. The minimum requirements for both the server and client are listed in the document with the corresponding list of commands and replies designated for FTP.

2.1 FTP Server

MATT

2.2 FTP Client

The FTP client is executed from a user's personal computer (local host) and allows the user to interact with a FTP server with the goal of transferring files both to and from the server. The client is im-

plemented with a GUI (graphical user interface) that makes the use of FTP easier.

The GUI allows the user to connect to the desired FTP server, but does not give the option to change the port number. The port number is restricted to 21 as this is the designated port number for FTP. Once connected to the desired server, the user is then allowed to login with a username and password.

The username and password used to log into the server must be authenticated by the server before the user can continue. Once authenticated, the user is given a chance to call various functions in the interface. Since the design of the GUI for the FTP client is kept simple, discussed later in Section 4.2.4, the client requires the user to type in designated commands that will prompt the relative response from the FTP server.

The user is allowed user to view the current directory in the server and in the local host. This is provided by listing each item in the respective directory, labeling each returned item as a folder or a file. Once the user knows the files and folders in both the server and the local host, they are allowed to change the directory in either using the respective command prompts. The user can traverse the server with ease, by allowing them to go down a level (into a new folder) or up a level (exit out of the current folder). Furthermore, the user can traverse the local host from the client interface, by changing folders or returning to the original directory when the connection between the client and server was established.

Subsequently, the user can upload (send) a file from the currently selected directory in the local host to the currently selected directory in the server. The files that a user can send, given that they have permission to write to the server directory, range from simple text files to a variety of image files, music files and video files.

The downloading (receiving/retrieving) a file from the server to the local host is also possible. As mentioned

before, depending on the user's currently selected directory, a variety of file types can be downloaded to the local host over the client interface and saved to the respective folder.

2.3 Unimplemented Features

Although the basic features of traversing directories in both the client and server, uploading and downloading files and presenting the current directories, many features have not been implemented in the client. The user is unable to create new directories, remove or rename them from either server or client directories. This extends to the inability to delete or rename files from the server or client.

Due to complexity in programming, the feature to change a file structure is not implemented. Along with the ability to change transmission mode, the client does not allow the user to change the transmission mode between *stream* and *block* mode. The *stream* mode is utilized as the default transmission mode in this implementation of FTP, since all data transferred is of *file* structure [3].

3. COMMANDS AND REPLIES

MATT, the table of our server's code implementations

4. IMPLEMENTATION

The server and client were both implemented using the programming language Python, specifically Python 3. Both client and server utilize the `socket` module that allows the user to access the BSD socket interface in the operating system [4]. The communication between the client and server utilizes this module. The `os` module allows the user to use the operating system dependent functionality [5]. This module gave the user the ability to traverse and view the directories and the respective files.

Since FTP communicates over a TCP/IP connection, this type of connection needs to be established. When commands are sent from the client to the server, these need to be formatted in a way that the server can understand and decode. For this reason, UTF-8 encoding was used for universal communication. An example of how the commands are formatted and sent can be seen in Figure 1, where the respective “[” “]” should be ignored and `\r\n` behave as a terminator of the command.

```
[Command] [Arguments] [\r\n]
```

Figure 1 : Command

4.1 Server Implementation

MATT, see how Jared did this part of the report

4.2 Client Implementation

The FTP client was first developed as a terminal-type system, with all responses being sent to and received from the server being presented in the command window. This made for easier debugging if errors arose before creating a GUI for the client.

4.2.1 Connection to the Server The first step in implementing the FTP client was to secure a connection with the server. In order for the client to connect, the IP address and port number of the server must be known before the connection can be initiated. Since the purpose of the connection is for FTP, the port number utilized is set to 21. In order to communicate with the server, a `send()` function was created that will send the commands to the server as per Figure 1.

With a half duplex communication channel set up, the client required a `receive()` function that could return the message sent from the server. These responses from the server are printed to the command window, and in the GUI the display window, so that the user can respond accordingly. To ensure that the message or command would be sent first then received a function that confined these functions to the correct order was created, aptly named `combinedSendReceive()`.

A `passive()` function was created in order for files to be transferred between the client and server. This function sends the code `PASV` to the server that initiates the passive mode connection. This means that the client initiates both connections to the server, therefore preventing the firewalls from filtering incoming data [6]. The client opens two random ports locally, whereby the first port contacts the server on port 21, and the `PASV` command will be issued. The second random port is opened (port number >1023), the connection is established and data can then be transferred [6].

4.2.2 Traversing the Directories The local host directory is obtained through the function `localDirectory()`. This function uses a function in the `os` module to go through every hierarchical directory before presenting the full directory the user is currently in. Listing the files and folders in the current directory is done through the function `browseDirectory()`. However, in the directory, all sub-directories (folders) are listed first, and then all subsequent files are listed.

The server directory is presented by the function `listDirectory()`, where FTP passive mode must be entered and the command `LIST` is sent to the server, and the client decodes the received data, and presents it accordingly. Once all of the folders and files in the remote directory have been listed, the `ABOR` command is sent to the server and the new passive socket is closed. Changing directories is done through user in-

put, where the command `CDUP` is sent to the server to go up one directory or `CWD` to change the directory and go into one of the listed folders on the server.

4.2.3 Transferring Files Sending or uploading a file to the server is done through the function `sendFile()`. However, before sending the file using this function, the server needs information on the type of file that it will receive. If the file is an ASCII file, the command `TYPE A` is sent to the server, or if the file is a binary (image) file, the command `TYPE I` is sent to the server. Passive mode is then entered, and the command `STOR` and the file name is sent to the server. While being uploaded, the file is broken down into smaller chunks of data and sent through.

Receiving or retrieving or downloading a file from the server is done through the function `receiveFile()`. Once again, the server needs to know the type of file that is being sent to the client, and since no automatic process of checking the file type is done, the user needs to enter either ASCII or binary (image) and respective commands are sent to the servers. Once passive mode is entered, the command `RECV` is sent to the server, and the new file is downloaded to the home directory of the local host.

4.2.4 GUI The `Qt` application framework is utilized as the graphical user interface. The specific one used is the `PyQt5` framework as it was designed for the `Python` programming language. The GUI provides the user with a simple interface in order to accomplish the desired tasks of an FTP client. There are various text boxes and corresponding buttons that will allow the user to enter the server address and press the connect button to initiate a connection. There is a display box that allows the user to view the corresponding codes from the server, current directories and relevant responses depending on the commands typed in the interface. Figure 2 in Appendix A shows the designed interface.

5. DIVISION OF WORK

The File Transfer Application has two components: the client and the server. All client related work, including all relevant report sections were written by Kishan Narotam. All server related work, including all relevant report sections were written by Matthew van Rooyen. The commands and replies information was handled by Matthew while Kishan handled the introduction and conclusion of the report. The results, critical analysis and abstract was done by both group members.

6. RESULTS

The testing was done in various ways in order to test the full functionality of client and server. First the im-

plemented client and server are tested together on two separate laptops connected to the same network. Second, the implemented client is tested with a FileZilla server on the same laptop, and lastly the implemented client is tested with a free online FTP server.

6.1 Implemented server and client

Due to certain limitations and issues, discussed in further detail in Section 7.2, the commands tested against the implemented server were focused more around uploading and downloading various files. The client (10.0.0.24) connected to the server (10.0.0.7) and confirmation was received. As seen in Figures 3 and 4 in Appendix C, the client requested the current working directories. The client then downloaded the file `TestFile.txt`, and once completed, the client uploaded the file `Devil.png`. Both files were respectively downloaded and uploaded correctly, thus confirming a successful test scenario.

6.2 FileZilla server and client

Increasing the amount of functionality seen in the previous test, the FileZilla server was connected to by the client on the same host. Figures 5 and 6 in Appendix C show the log from FileZilla and how the directories are presented in the interface respectively. As seen from the log, the remote directory was changed before the file `Obi-Wan.mp3` was uploaded. After a computer error, the client had to reconnect to the server, then was prompted to download the image `May4th.jpg`. The directory on the server was changed, before another download was prompted, and lastly a text file `lipsum.txt` was uploaded to the server. This confirmed yet another successful test scenario of the implemented client.

6.3 Free online server and client

The final test case made use of a public FTP server (`demo.wftpserver.com`). The reason this was chosen was to test the authentication of the user's login information, thus determining the amount of rights granted. Figures 7 and 8 in Appendix C show the Wireshark trace and respond codes and the implemented interface displays information after downloading a significantly large file. First the directories are displayed, then the remote directory was changed and download began for pdf file `manual_en.pdf` which took approximately 16 seconds to download. Once downloaded, the total number of bytes downloaded and the speed of the transfer was displayed in the interface. An attempt was made to upload a file to the public FTP server, however as seen from the trace in Figure 7 in Appendix C, the user was banned as they did not have the correct rights. This test scenario proved successful and showed that the client is indeed a successful implementation.

7. CRITICAL ANALYSIS

After testing and analyzing the implemented system, the success, limitations and possible future development ideas are presented below.

7.1 Successes

The File Transfer Application is functional and is a decent solution. As shown in Section 6., the requirements are met for a basic File Transfer Application. These requirements are:

- The client is able to communicate with the server, send commands and receive corresponding replies from the server.
- The client is provided with a simple graphical interface that communicates with different FTP servers.
- The client is able to upload and download various files, such as: images, audio, text, video.
- Wireshark traces indicated that the File Transfer Protocol was utilized and followed.
- The client and server are able to communicate with one another regardless of which host either is on, provided they are on the same network.
- No FTP libraries available in Python were used.

The system met all these requirements and yielded promising results. This is an indication of the success of the system.

7.2 Limitations and Problems

The client did not suffer from any or few problems. One issue that occurred was that should the server not respond in a timely manner, the client would crash. This could be due to an incorrect command being sent to the server or a computer problem causing the program to crash. The client was fairly limited as its functionality did not allow for creation and deletion of directories and files.

7.3 Future Development

The client interface can be given a more interactive and informative improvement. Along with this, the client can implement some of the missing features explained in Section 2.3.

8. CODE STRUCTURE AND PREREQUISITES

8.1 Code Structure

The client and server both make use of a class based code structure. The server has one class that allows the multithreading to occur, in order for multiple clients to connect. Subsequently, it contains all of the necessary indexing of the commands that could be sent by a client and the corresponding response codes. The

client has one class that extends on the initial implementation of a terminal-based client. This was done to reduce complexity and to ensure the success of the terminal based client was carried over into the client with a user interface.

8.2 System Prerequisites

The server and client require Python 3 and were implemented on Windows 10 operating systems. Many external modules were utilized to ensure a functional client interface from Python 3. The PyQt5 module was chosen as this is one of the available interface modules that gave minimal complexity and allowed for the simple interface implemented for the client. Furthermore, this module is compatible with Python 3. In order to execute the programs, one would need to install the relevant modules by typing the following in the terminal (cmd): `pip install pyqt5`

The server can be started by typing the following command in the terminal:

The client can be started by using the following command in the terminal: `python -u ClientInterface.py`

9. CONCLUSION

the design, implementation and testing of a File Transfer Application has been presented. The overall system was regarded as a success regardless of minor problems experienced. The minimum requirements of a File transfer Application were met and a simple user interface was created for the client. Wireshark was utilized to monitor and ensure that the correct protocol, the File Transfer Protocol, was being utilized and the correct commands were being sent to the server. The final client was tested against three different FTP servers and was proven to produce successful results. The system was analyzed and future development recommendations have been provided.

REFERENCES

- [1] Vlajin, B; *What Is FTP? Everything About File Transfer Protocol*; <https://www.cloudwards.net/what-is-ftp/>; Last Accessed: 04/05/2019
- [2] Statz, P; *FTP for Beginners—WIRED*; https://www.wired.com/2010/02/ftp_for_beginners/; Last Accessed: 04/05/2019
- [3] Postel, J; Reynolds, J; *File Transfer Protocol (FTP)*; Network Working Group; October 1985
- [4] Python; *socket Low-level networking interface Python 3.7.3 documentation*; <https://docs.python.org/3/library/socket.html>; Last Accessed: 04/05/2019
- [5] Python; *os Miscellaneous operating system interfaces Python 3.7.3 documentation*; <https://docs.python.org/3/library/os>.

html?highlight=os#module-os; Last Accessed:
04/05/2019

- [6] Ribak, J; *Active FTP vs. Passive FTP, a Definitive Explanation*; <https://slacksite.com/other/ftp.html>; Last Accessed: 04/05/2019

Appendices

A Designed GUI

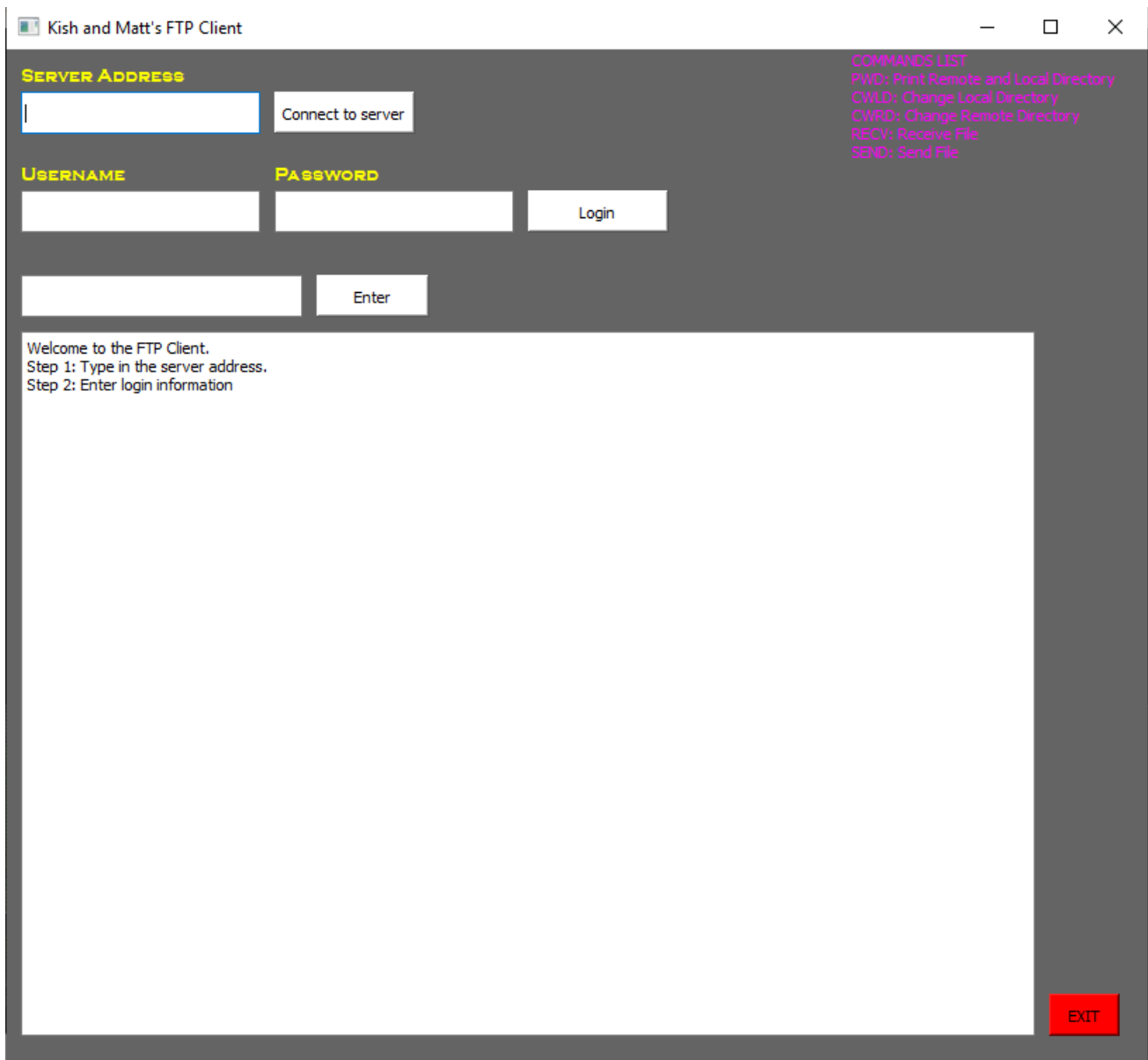
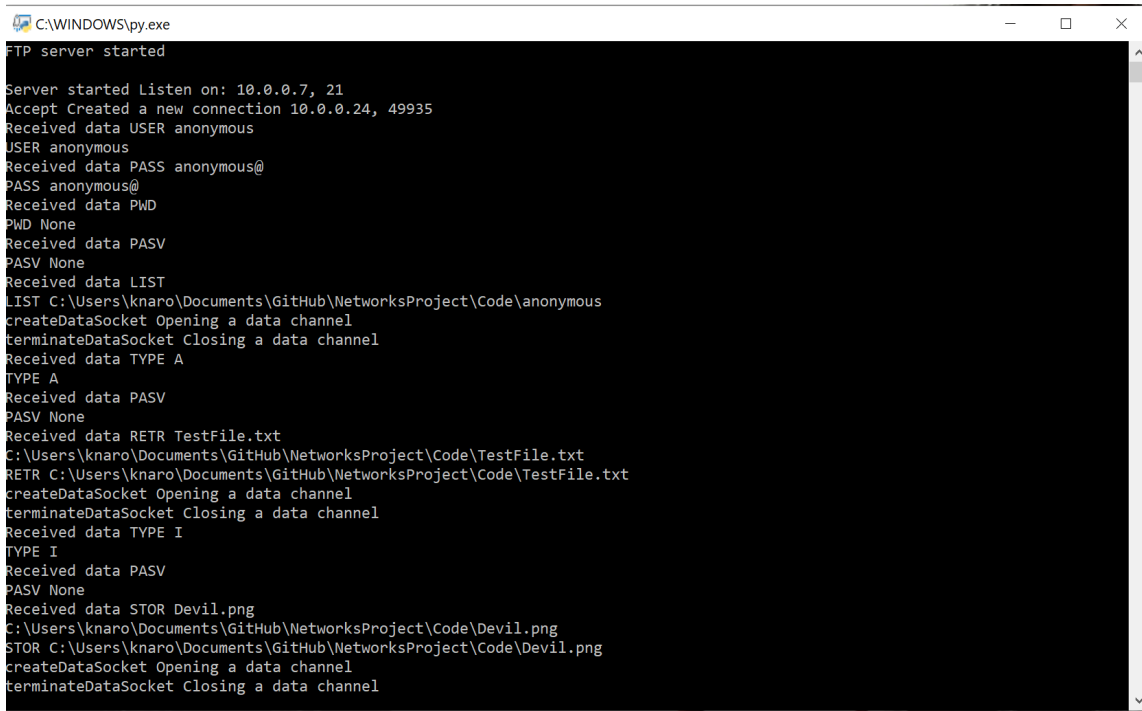


Figure 2 : Designed GUI in PyQt5

B Implemented Commands and Reply Codes

C Results

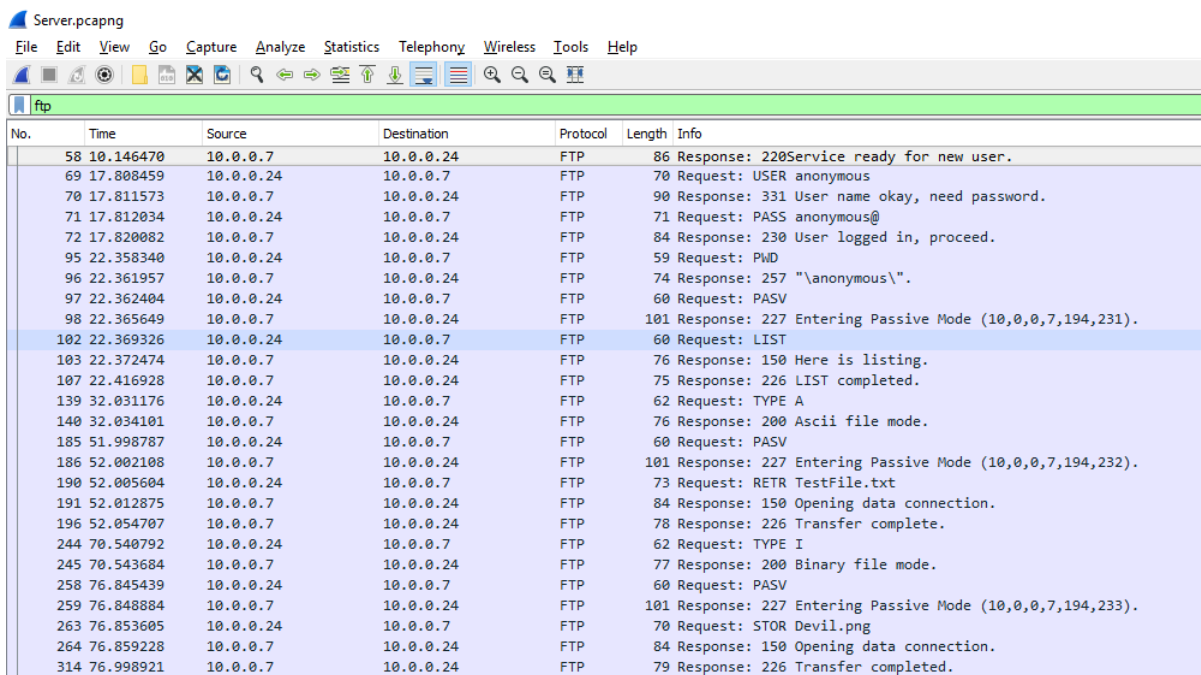


```

C:\WINDOWS\py.exe
FTP server started
Server started Listen on: 10.0.0.7, 21
Accept Created a new connection 10.0.0.24, 49935
Received data USER anonymous
USER anonymous
Received data PASS anonymous@
PASS anonymous@
Received data PWD
PWD None
Received data PASV
PASV None
Received data LIST
LIST C:\Users\knaro\Documents\GitHub\NetworksProject\Code\anonymous
createDataSocket Opening a data channel
terminateDataSocket Closing a data channel
Received data TYPE A
TYPE A
Received data PASV
PASV None
Received data RETR TestFile.txt
C:\Users\knaro\Documents\GitHub\NetworksProject\Code\TestFile.txt
RETR C:\Users\knaro\Documents\GitHub\NetworksProject\Code\TestFile.txt
createDataSocket Opening a data channel
terminateDataSocket Closing a data channel
Received data TYPE I
TYPE I
Received data PASV
PASV None
Received data STOR Devil.png
C:\Users\knaro\Documents\GitHub\NetworksProject\Code\Devil.png
STOR C:\Users\knaro\Documents\GitHub\NetworksProject\Code\Devil.png
createDataSocket Opening a data channel
terminateDataSocket Closing a data channel

```

Figure 3 : Server log on host 10.0.0.7



No.	Time	Source	Destination	Protocol	Length	Info
58	10.146470	10.0.0.7	10.0.0.24	FTP	86	Response: 220 Service ready for new user.
69	17.808459	10.0.0.24	10.0.0.7	FTP	70	Request: USER anonymous
70	17.811573	10.0.0.7	10.0.0.24	FTP	90	Response: 331 User name okay, need password.
71	17.812034	10.0.0.24	10.0.0.7	FTP	71	Request: PASS anonymous@
72	17.820082	10.0.0.7	10.0.0.24	FTP	84	Response: 230 User logged in, proceed.
95	22.358340	10.0.0.24	10.0.0.7	FTP	59	Request: PWD
96	22.361957	10.0.0.7	10.0.0.24	FTP	74	Response: 257 "\anonymous\".
97	22.362404	10.0.0.24	10.0.0.7	FTP	60	Request: PASV
98	22.365649	10.0.0.7	10.0.0.24	FTP	101	Response: 227 Entering Passive Mode (10,0,0,7,194,231).
102	22.369326	10.0.0.24	10.0.0.7	FTP	60	Request: LIST
103	22.372474	10.0.0.7	10.0.0.24	FTP	76	Response: 150 Here is listing.
107	22.416928	10.0.0.7	10.0.0.24	FTP	75	Response: 226 LIST completed.
139	32.031176	10.0.0.24	10.0.0.7	FTP	62	Request: TYPE A
140	32.034101	10.0.0.7	10.0.0.24	FTP	76	Response: 200 Ascii file mode.
185	51.998787	10.0.0.24	10.0.0.7	FTP	60	Request: PASV
186	52.002108	10.0.0.7	10.0.0.24	FTP	101	Response: 227 Entering Passive Mode (10,0,0,7,194,232).
190	52.005604	10.0.0.24	10.0.0.7	FTP	73	Request: RETR TestFile.txt
191	52.012875	10.0.0.7	10.0.0.24	FTP	84	Response: 150 Opening data connection.
196	52.054707	10.0.0.7	10.0.0.24	FTP	78	Response: 226 Transfer complete.
244	70.540792	10.0.0.24	10.0.0.7	FTP	62	Request: TYPE I
245	70.543684	10.0.0.7	10.0.0.24	FTP	77	Response: 200 Binary file mode.
258	76.845439	10.0.0.24	10.0.0.7	FTP	60	Request: PASV
259	76.848884	10.0.0.7	10.0.0.24	FTP	101	Response: 227 Entering Passive Mode (10,0,0,7,194,233).
263	76.853605	10.0.0.24	10.0.0.7	FTP	70	Request: STOR Devil.png
264	76.859228	10.0.0.7	10.0.0.24	FTP	84	Response: 150 Opening data connection.
314	76.998921	10.0.0.7	10.0.0.24	FTP	79	Response: 226 Transfer completed.

Figure 4 : Wireshark trace of implemented client (10.0.0.24) and implemented server (10.0.0.24)


```

(000001)04/05/2019 23:30:13 PM - anon (10.0.0.24)> 230 Logged on
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> PWD
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> 257 "/" is current directory.
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> PASV
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,246,110)
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> LIST
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> 150 Opening data channel for directory listing of "/"
(000001)04/05/2019 23:30:16 PM - anon (10.0.0.24)> 226 Successfully transferred "/"
(000001)04/05/2019 23:31:18 PM - anon (10.0.0.24)> CWD Upload
(000001)04/05/2019 23:31:18 PM - anon (10.0.0.24)> 250 CWD successful. "/Upload" is current directory.
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> PWD
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> 257 "/Upload" is current directory.
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> PASV
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,206,176)
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> LIST
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> 150 Opening data channel for directory listing of "/Upload"
(000001)04/05/2019 23:31:23 PM - anon (10.0.0.24)> 226 Successfully transferred "/Upload"
(000001)04/05/2019 23:31:38 PM - anon (10.0.0.24)> TYPE I
(000001)04/05/2019 23:31:38 PM - anon (10.0.0.24)> 200 Type set to I
(000001)04/05/2019 23:31:43 PM - anon (10.0.0.24)> PASV
(000001)04/05/2019 23:31:43 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,236,27)
(000001)04/05/2019 23:31:43 PM - anon (10.0.0.24)> STOR Obi-Wan.mp3
(000001)04/05/2019 23:31:43 PM - anon (10.0.0.24)> 150 Opening data channel for file upload to server of "/Upload/Obi-Wan.mp3"
(000001)04/05/2019 23:31:43 PM - anon (10.0.0.24)> 226 Successfully transferred "/Upload/Obi-Wan.mp3"
(000001)04/05/2019 23:32:02 PM - anon (10.0.0.24)> CDUP
(000001)04/05/2019 23:32:02 PM - anon (10.0.0.24)> 200 CDUP successful. "/" is current directory.
(000001)04/05/2019 23:32:08 PM - anon (10.0.0.24)> PWD
(000001)04/05/2019 23:32:08 PM - anon (10.0.0.24)> 257 "/" is current directory.
(000001)04/05/2019 23:32:08 PM - anon (10.0.0.24)> PASV
(000001)04/05/2019 23:32:08 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,236,214)
(000001)04/05/2019 23:32:11 PM - anon (10.0.0.24)> disconnected.
(000002)04/05/2019 23:32:36 PM - (not logged in) (10.0.0.24)> Connected on port 21, sending welcome message...
(000002)04/05/2019 23:32:36 PM - (not logged in) (10.0.0.24)> 220-FileZilla Server 0.9.60 beta
(000002)04/05/2019 23:32:36 PM - (not logged in) (10.0.0.24)> 220-written by Tim Kosse (tim.kosse@filezilla-project.org)
(000002)04/05/2019 23:32:36 PM - (not logged in) (10.0.0.24)> 220 Please visit https://filezilla-project.org/
(000002)04/05/2019 23:32:39 PM - (not logged in) (10.0.0.24)> USER anon
(000002)04/05/2019 23:32:39 PM - (not logged in) (10.0.0.24)> 331 Password required for anon
(000002)04/05/2019 23:32:39 PM - (not logged in) (10.0.0.24)> PASS ****
(000002)04/05/2019 23:32:39 PM - anon (10.0.0.24)> 230 Logged on
(000002)04/05/2019 23:33:02 PM - anon (10.0.0.24)> CWD Download
(000002)04/05/2019 23:33:02 PM - anon (10.0.0.24)> 250 CWD successful. "/Download" is current directory.
(000002)04/05/2019 23:33:23 PM - anon (10.0.0.24)> TYPE I
(000002)04/05/2019 23:33:23 PM - anon (10.0.0.24)> 200 Type set to I
(000002)04/05/2019 23:33:33 PM - anon (10.0.0.24)> PASV
(000002)04/05/2019 23:33:33 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,255,177)
(000002)04/05/2019 23:33:33 PM - anon (10.0.0.24)> RETR May4th.jpg
(000002)04/05/2019 23:33:33 PM - anon (10.0.0.24)> 150 Opening data channel for file download from server of "/Download/May4th.jpg"
(000002)04/05/2019 23:33:33 PM - anon (10.0.0.24)> 226 Successfully transferred "/Download/May4th.jpg"
(000002)04/05/2019 23:33:43 PM - anon (10.0.0.24)> CDUP
(000002)04/05/2019 23:33:43 PM - anon (10.0.0.24)> 200 CDUP successful. "/" is current directory.
(000002)04/05/2019 23:33:54 PM - anon (10.0.0.24)> CWD
(000002)04/05/2019 23:33:54 PM - anon (10.0.0.24)> 250 Broken client detected, missing argument to CWD. "/" is current directory.
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> PWD
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> 257 "/" is current directory.
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> PASV
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,198,77)
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> LIST
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> 150 Opening data channel for directory listing of "/"
(000002)04/05/2019 23:33:59 PM - anon (10.0.0.24)> 226 Successfully transferred "/"
(000002)04/05/2019 23:34:09 PM - anon (10.0.0.24)> TYPE A
(000002)04/05/2019 23:34:09 PM - anon (10.0.0.24)> 200 Type set to A
(000002)04/05/2019 23:34:12 PM - anon (10.0.0.24)> PASV
(000002)04/05/2019 23:34:12 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,202,174)
(000002)04/05/2019 23:34:12 PM - anon (10.0.0.24)> RETR user.txt
(000002)04/05/2019 23:34:12 PM - anon (10.0.0.24)> 150 Opening data channel for file download from server of "/user.txt"
(000002)04/05/2019 23:34:12 PM - anon (10.0.0.24)> 226 Successfully transferred "/user.txt"
(000002)04/05/2019 23:34:58 PM - anon (10.0.0.24)> TYPE A
(000002)04/05/2019 23:34:58 PM - anon (10.0.0.24)> 200 Type set to A
(000002)04/05/2019 23:35:04 PM - anon (10.0.0.24)> PASV
(000002)04/05/2019 23:35:04 PM - anon (10.0.0.24)> 227 Entering Passive Mode (10,0,0,24,209,152)
(000002)04/05/2019 23:35:04 PM - anon (10.0.0.24)> STOR ipsum.txt
(000002)04/05/2019 23:35:04 PM - anon (10.0.0.24)> 150 Opening data channel for file upload to server of "/ipsum.txt"
(000002)04/05/2019 23:35:05 PM - anon (10.0.0.24)> 226 Successfully transferred "/ipsum.txt"

```

Figure 5 : FileZilla log

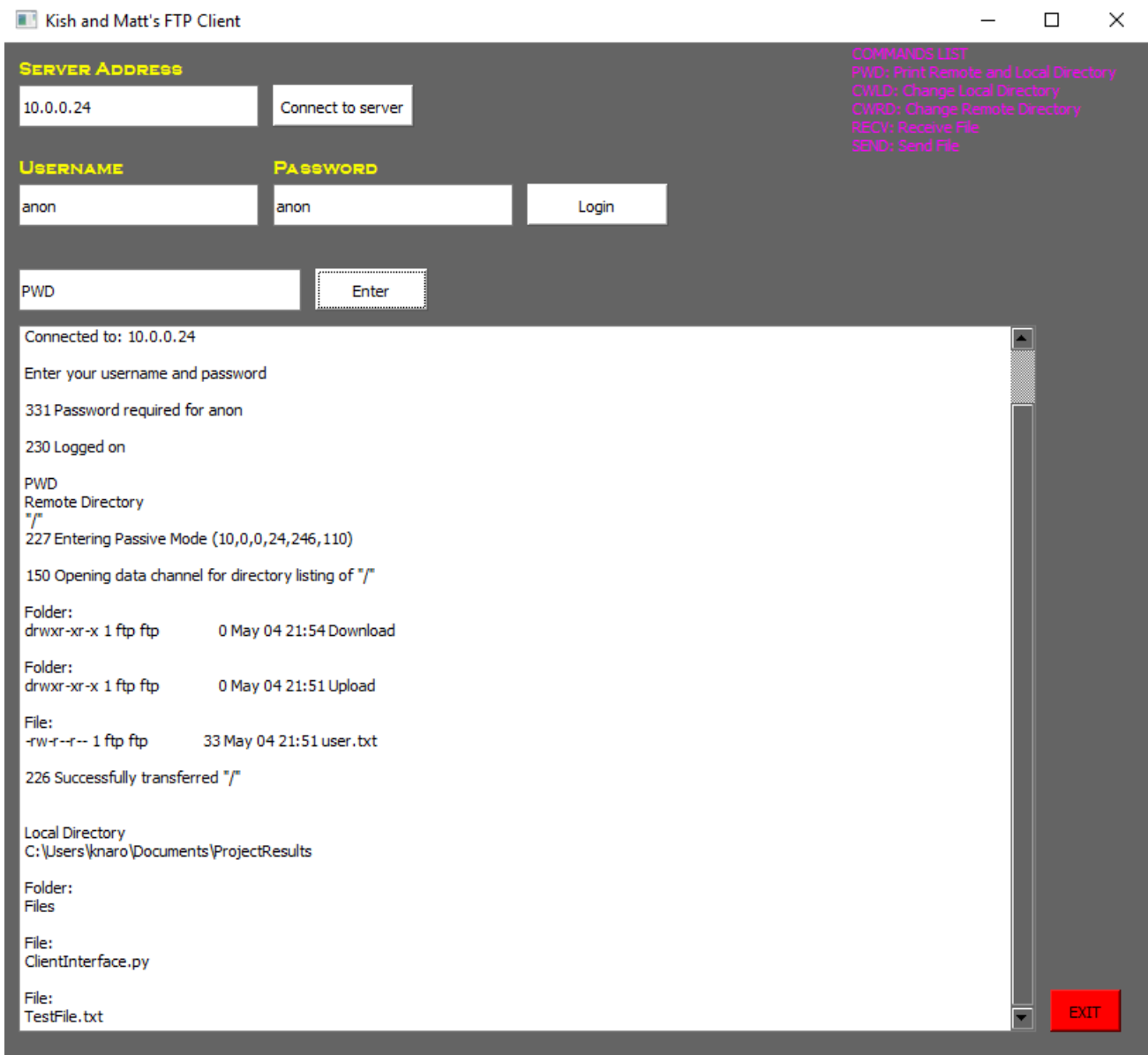


Figure 6 : GUI output of printing the current directories

Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ftp

No.	Time	Source	Destination	Protocol	Length	Info
60	9.027681	199.71.215.197	10.0.0.24	FTP	84	Response: 220 Wing FTP Server ready...
83	18.849344	10.0.0.24	199.71.215.197	FTP	70	Request: USER demo-user
85	19.171810	199.71.215.197	10.0.0.24	FTP	91	Response: 331 Password required for demo-user
86	19.172234	10.0.0.24	199.71.215.197	FTP	70	Request: PASS demo-user
87	19.495732	199.71.215.197	10.0.0.24	FTP	85	Response: 230 User demo-user logged in.
101	23.098067	10.0.0.24	199.71.215.197	FTP	59	Request: PWD
104	23.419506	199.71.215.197	10.0.0.24	FTP	85	Response: 257 "/" is current directory.
105	23.420061	10.0.0.24	199.71.215.197	FTP	60	Request: PASV
106	23.745400	199.71.215.197	10.0.0.24	FTP	103	Response: 227 Entering Passive Mode (199,71,215,197,4,16)
111	24.068889	10.0.0.24	199.71.215.197	FTP	60	Request: LIST
115	24.394819	199.71.215.197	10.0.0.24	FTP	100	Response: 150 Opening data channel for directory list.
118	24.756751	199.71.215.197	10.0.0.24	FTP	80	Response: 226 Transfer successful.
152	33.852162	10.0.0.24	199.71.215.197	FTP	68	Request: CWD download
153	34.175357	199.71.215.197	10.0.0.24	FTP	117	Response: 250 CWD command successful. "/download" is current directory.
160	38.986898	10.0.0.24	199.71.215.197	FTP	59	Request: PWD
163	39.309967	199.71.215.197	10.0.0.24	FTP	93	Response: 257 "/download" is current directory.
164	39.310466	10.0.0.24	199.71.215.197	FTP	60	Request: PASV
167	39.631904	199.71.215.197	10.0.0.24	FTP	103	Response: 227 Entering Passive Mode (199,71,215,197,4,17)
172	39.955442	10.0.0.24	199.71.215.197	FTP	60	Request: LIST
178	40.286047	199.71.215.197	10.0.0.24	FTP	100	Response: 150 Opening data channel for directory list.
183	40.648177	199.71.215.197	10.0.0.24	FTP	80	Response: 226 Transfer successful.
210	51.288123	10.0.0.24	199.71.215.197	FTP	62	Request: TYPE A
213	51.611668	199.71.215.197	10.0.0.24	FTP	73	Response: 200 Type set to A
238	57.762225	10.0.0.24	199.71.215.197	FTP	60	Request: PASV
239	58.083896	199.71.215.197	10.0.0.24	FTP	103	Response: 227 Entering Passive Mode (199,71,215,197,4,18)
246	58.410247	10.0.0.24	199.71.215.197	FTP	74	Request: RETR manual_en.pdf
247	58.733376	199.71.215.197	10.0.0.24	FTP	140	Response: 150 Data connection accepted; transfer starting for manual_en.pdf (10187202 Bytes).
10008	70.953091	199.71.215.197	10.0.0.24	FTP	136	Response: 226 File sent successful. Transferred:10187202Bytes;Average speed is:127.544KB/s
10516	86.934805	10.0.0.24	199.71.215.197	FTP	60	Request: CDUP
10517	87.256622	199.71.215.197	10.0.0.24	FTP	110	Response: 250 CDUP command successful. "/" is current directory.
10555	95.714808	10.0.0.24	199.71.215.197	FTP	66	Request: CWD upload
10556	96.040424	199.71.215.197	10.0.0.24	FTP	115	Response: 250 CWD command successful. "/upload" is current directory.
10572	104.163464	10.0.0.24	199.71.215.197	FTP	62	Request: TYPE A
10573	104.484987	199.71.215.197	10.0.0.24	FTP	73	Response: 200 Type set to A
10579	107.645334	10.0.0.24	199.71.215.197	FTP	60	Request: PASV
10580	107.967529	199.71.215.197	10.0.0.24	FTP	103	Response: 227 Entering Passive Mode (199,71,215,197,4,19)
10587	108.287726	10.0.0.24	199.71.215.197	FTP	69	Request: STOR user.txt
10589	108.609560	199.71.215.197	10.0.0.24	FTP	90	Response: 550-This file is banned on server!
10597	108.978437	199.71.215.197	10.0.0.24	FTP	87	Response: 550 Cannot STOR. No permission.

Figure 7 : FileZilla log

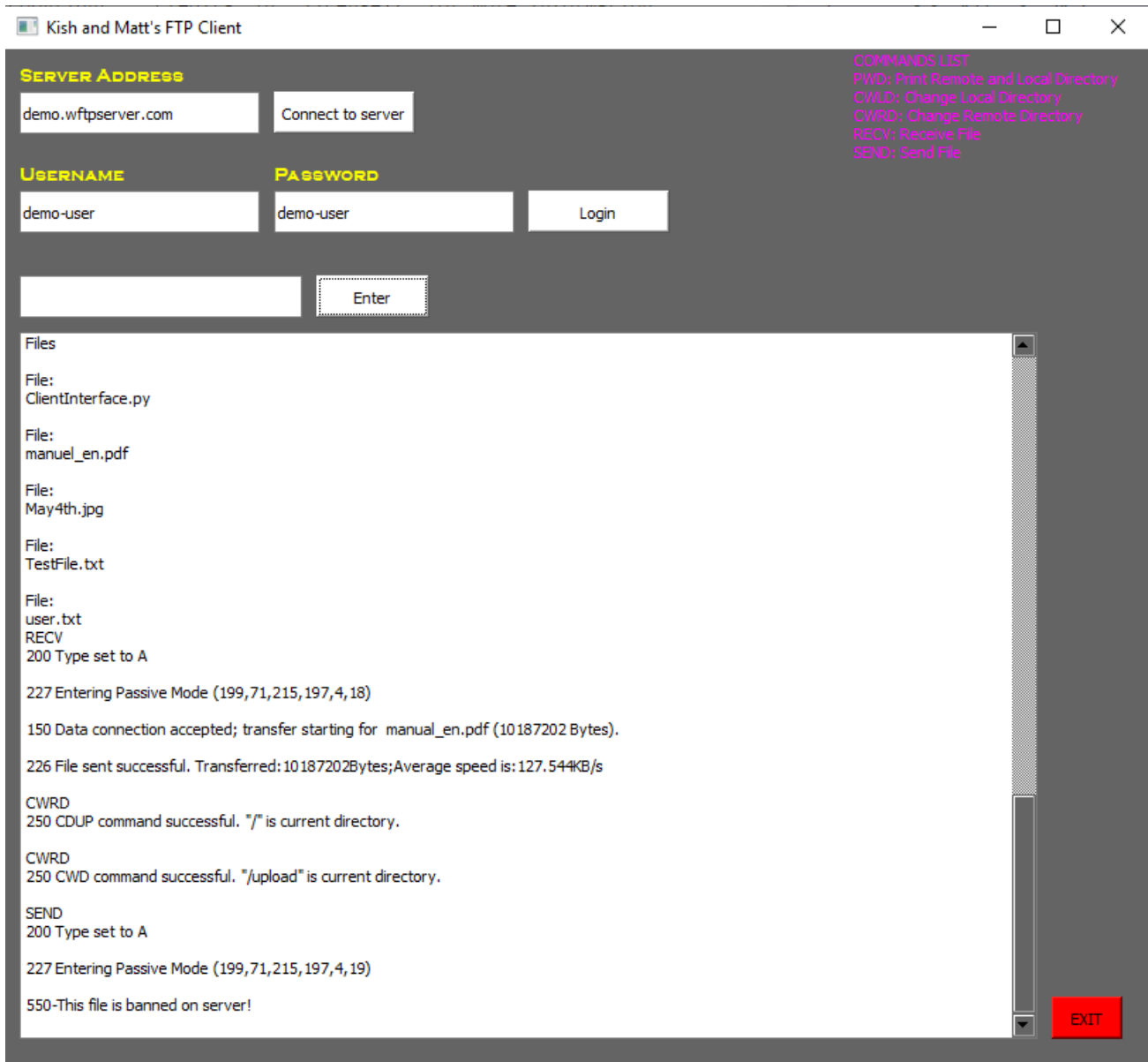


Figure 8 : GUI output after downloading a file from the server (demo.wftpserver.com)