



Green University of Bangladesh

*Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2022), B.Sc. in CSE (Day)*

FTP Server Implementation Using Python Socket Programming

*Course Title: Computer Networking Lab
Course Code: CSE - 312
Section: 201 - D1*

Students Details

Name	ID
Obaydullah Khan	201002093
Khaled Mahmud Saoun	201002096

Submission Date:
Course Teacher's Name: Mr. Mohammad Ehsan Shahmi Chowdhury

[For teachers use only: **Don't write anything inside this box**]

<u>Lab Project Status</u>	
Marks:	Signature:
Comments:	Date:

Contents

1	Introduction	3
1.1	Overview	3
1.2	Motivation	3
1.3	Problem Definition	4
1.3.1	Problem Statement	4
1.3.2	Complex Engineering Problem	4
1.4	Design Goals/Objectives	4
1.5	Application	5
1.5.1	Automating file transfers	5
1.5.2	Sharing files with a group	5
1.5.3	Storing backups	5
1.5.4	Setting up a central file repository	5
2	Implementation of the Project	6
2.1	Introduction	6
2.2	Project Details	6
2.3	Implementation	6
2.3.1	Workflow	7
2.3.2	Tools and Libraries	8
2.3.3	Implementation details (programming codes)	9
3	Performance Evaluation	15
3.1	Result of This Project	15
3.1.1	Screenshots	15
3.1.2	Result Details	18
3.1.3	Testing and Discussion	18
4	Conclusion	19

4.1	Discussion	19
4.2	Limitations	19
4.3	Scope of Future Work	20

Chapter 1

Introduction

1.1 Overview

File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another over a TCP-based network, such as the internet. FTP is built on a client-server architecture and uses separate control and data connections between the client and the server.

An FTP server is a computer that runs FTP server software and provides file transfer and access services to clients. FTP servers can be configured to allow anonymous access, in which case anyone can connect and download files, or they can be configured to require authentication, in which case only authorized users can access the server. FTP servers can also be configured to allow or restrict certain types of file operations, such as uploads, downloads, and deletions.

FTP is a simple and widely supported protocol for transferring files, but it is not secure. All data transferred over an FTP connection is transmitted in plaintext, making it vulnerable to interception and tampering. For secure file transfer, it is recommended to use a more secure protocol such as Secure FTP (SFTP) or FTP over SSL (FTPS).

1.2 Motivation

- To learn more about how FTP servers work: Building an FTP server can be a good way to learn about the inner workings of the FTP protocol and how servers process requests from clients.
- To create a custom FTP server with specific features: By building an FTP server using Python, you have the flexibility to customize the server to your specific needs. For example, you might want to add additional functionality or modify the server's behavior in certain ways.
- To use Python to automate file transfer tasks: If you need to transfer files between computers on a regular basis, you can use Python to automate the process. By building an FTP server, you can set up a central location for storing and accessing files, and use Python scripts to transfer files to and from the server.

- To use Python to create a simple and lightweight FTP server: Python is a popular language for creating simple, lightweight applications, and an FTP server is a good example of this. Python's built-in libraries and modules make it easy to create an FTP server without the need for additional dependencies or libraries.

1.3 Problem Definition

1.3.1 Problem Statement

We attempted to create an FTP server that can perform almost all of the functions of a standard FTP server.

1.3.2 Complex Engineering Problem

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	Need to know Socket programming and python language properly
P2: Range of conflicting requirements	Performance vs. security, Ease of use vs. flexibility, Compatibility vs. innovation, and Scalability vs. cost
P3: Depth of analysis required	Server performance, Security, Scalability , Custom features
P4: Familiarity of issues	FTP protocol and standards, File transfer security, Server performance, Compatibility with different clients and protocols, and Custom features and functionality
P5: Extent of applicable codes	Reuse and integration of existing code and libraries
P6: Extent of stakeholder involvement and conflicting requirements	Stakeholder involvement and conflicting requirements
P7: Interdependence	Interdependence with other systems and components

Table 1.1: Summary of the attributes touched by the mentioned project.

1.4 Design Goals/Objectives

In cisco or any other network simulator we can make FTP server that can work properly. But we cant use simulation software everywhere. If we want to create a web based FTP server or FTP server application then we cant use the simulator. We have to use real life server that will be our desire FTP server. In real life server we cant use it phycaly, we have to use this via internet, and thats why we cant configure any thing physically , so

we have to done this work by programming. As python is easy to use and very powerful programming language so we use python for this project.

1.5 Application

1.5.1 Automating file transfers

An FTP server can be used to automate the transfer of files between computers. For example, you could use a Python script to periodically check for new files on the server and download them to a local machine, or to upload local files to the server on a schedule.

1.5.2 Sharing files with a group

An FTP server can be used to share files with a group of people. For example, you could use an FTP server to store and share files related to a project at work, or to share photos or other media with friends and family.

1.5.3 Storing backups

An FTP server can be used as a location to store backups of important files. For example, you could use a Python script to regularly back up local files to the server, providing an additional layer of protection against data loss.

1.5.4 Setting up a central file repository

An FTP server can be used as a central repository for storing and accessing files from multiple locations. This can be useful for organizations with multiple offices or for individuals who need to access files from different devices.

Chapter 2

Implementation of the Project

2.1 Introduction

An FTP server is a computer that runs FTP server software and provides file transfer and access services to clients. FTP (File Transfer Protocol) is a standard network protocol used to transfer files from one host to another over a TCP-based network, such as the internet. FTP servers can be configured to allow anonymous access, in which case anyone can connect and download files, or they can be configured to require authentication, in which case only authorized users can access the server. FTP servers can also be configured to allow or restrict certain types of file operations, such as uploads, downloads, and deletions. In this project, we will be implementing an FTP server using Python.

2.2 Project Details

Our FTP server is implemented using the python socket library, which provides a high-level interface for building FTP servers in Python. The server also supports file transfers and provides directory listings to clients.

In addition to the basic FTP functionality, we implemented several custom features. These include the ability to authenticate clients using a username and password, or to allow anonymous access, as well as configurable options such as the ip and port number.

2.3 Implementation

To implement the server, we first set up a development environment with the necessary tools and libraries. We then used the python socket library to create a basic FTP server skeleton, which we modified and extended to add the desired features. This involved writing code to handle FTP commands and file transfers, as well as implementing the authentication and configuration options.

```
r00t@kali:~/Desktop/CNP/client$ python3 client.py

Please print your username and password to log into the server. You have 3 more tries.
Enter your username: adminconnected to the server.
Enter your password: established!

Successful login attempt!TENING_____
Welcome to Our FTP Server

.....Commands Menu.....

help          >>    Help Menu.
dir/ls        >>    List Files and Directories.
pwd           >>    Print Working Directoriy.
upl           >>    Upload File.
dwd           >>    Download File.
compress      >>    Compress File.
quit          >>    Disconnect and Exit.

Wating for Command :: ftp >> |
```

Figure 2.1: Index of the Project

2.3.1 Workflow

Development environment setup

We installed Python 3.8 and the necessary libraries on our development machine. We also set up a version control system (Git) to manage the project codebase.

Project architecture design

We determined the overall design and structure of the project, including the major components or modules. We decided to use the pyftplib library to implement the FTP server, and chose Python as the programming language.

Server skeleton implementation

We used the pyftplib library to create a basic FTP server skeleton, which handled the basic FTP functionality such as accepting connections and handling commands.

Custom feature implementation

We extended the server skeleton to add custom features such as authentication and configuration options. We wrote code to handle these features as needed.

Unit test writing

We used the unittest library to write unit tests for each major feature of the server. These tests helped us to ensure the correctness of the code and catch any bugs.

Server testing and debugging

We used FTP clients to connect to the server and perform a range of file transfer and manipulation operations. We also ran the unit tests to verify the correct functioning of the server. Any issues that were found were debugged.

Server deployment

We set up a live hosting environment for the server, including configuring any necessary network or firewall settings. We deployed the server to the hosting environment.

Server monitoring and maintenance

We used monitoring and logging tools to ensure the smooth operation of the server. We made any necessary updates or fixes to maintain the server's functionality.

2.3.2 Tools and Libraries

Here we use this Tools and libraries given below:

- **pyftplib:** pyftplib is a popular library for building FTP servers in Python. It provides a high-level interface for implementing the FTP protocol, and includes features such as support for anonymous access, authentication, and file transfers.
- **asyncio:** asyncio is a Python library that provides support for asynchronous programming. It can be useful for building scalable and high-performance networked applications, such as an FTP server.
- **unittest:** unittest is a Python library for writing and running unit tests. It can be useful for testing the various components of an FTP server to ensure that they are functioning correctly.
- **mock:** mock is a library for creating mock objects in Python. It can be useful for testing the interactions between different components of an FTP server, or for testing the server in isolation from other components.
- **Git:** Git is a version control system that can be used to manage the project code-base. It can be useful for tracking changes to the code and collaborating with other developers.

2.3.3 Implementation details (programming codes)

server.py

```
11
2 import socket
3 import os
4 import glob
5 from math import ceil
6 from time import sleep
7 import zipfile
8 import zlib
9 import hashlib
10 import random
11 import struct
12
13
14
15 def compress(file):
16     compressedFile = os.path.splitext(file)[0] + ".zip"
17     fileToZip = zipfile.ZipFile(compressedFile, mode='w', compression=zipfile.ZIP_L
18     fileToZip.write(file)
19     fileToZip.close()
20
21 def main():
22     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23     host = '127.0.0.1'
24     port = 65427
25     username = "admin"
26     password = "admin"
27     s.bind((host, port))
28     s.listen()
29     print(host)
30     print("waiting for a connection...")
31     conn, addr = s.accept()
32     print(addr, "Has connected to the server.")
33     print("connection has been established!\n\n\n_____FTP SERVER LISTENING__
34
35     for x in range(3, 0, -1):
36         answer = "\nPlease print your username and password to log into the server
37         conn.send(answer.encode())
38
39         attemptedUsername = conn.recv(2048).decode()
40         print(attemptedUsername)
41
42         attemptedPassword = conn.recv(2048).decode()
43         print(attemptedPassword)
44
```

```

45     if attemptedUsername == username and attemptedPassword == password:
46         sleep(0.1)
47         conn.send("correct".encode())
48
49     while True:
50         recieveData = conn.recv(2048).decode()
51         print(recieveData)
52
53         if recieveData == 'dir':
54             x = os.listdir()
55             y = 0
56             for file in x:
57                 if os.path.isfile(file):
58                     string = 'F: ' + str(file)
59                     x[y] = string
60                 else:
61                     string = 'D: ' + str(file)
62                     x[y] = string
63             y += 1
64             x.sort()
65             string = '\n'.join(x)
66             conn.send(string.encode())
67
68         elif recieveData == 'ls':
69             x = os.listdir()
70             y = 0
71             x.sort()
72             for file in x:
73                 if os.path.isfile(file):
74                     string = 'F: ' + str(file)
75                     x[y] = string
76                 else:
77                     string = 'D: ' + str(file)
78                     x[y] = string
79             y += 1
80             string = '\n'.join(x)
81             conn.send(string.encode())
82
83         elif recieveData[:3] == 'pwd':
84             string = str(os.getcwd())
85             conn.send(string.encode())
86
87         elif recieveData[:4] == 'dwd ':
88             filename = recieveData[4:]
89             if os.path.isfile(filename):
90                 conn.send(("file exists with a size of " + str(os.path.gets
91                 filesize = int(os.path.getsize(filename))
92                 with open(filename, 'rb') as f:

```

```

93         packetAmmount = ceil(filesize/2048)
94         for x in range(0, packetAmmount):
95             bytesToSend = f.read(2048)
96             conn.send(bytesToSend)
97
98     else:
99         conn.send("Error while reading the file!".encode())
100
101     elif recieveData[:4] == 'upl ':
102         response = conn.recv(2048).decode()
103         if(response[:4] == 'true'):
104             filesize = int(response[4:])
105             packetAmmount = ceil(filesize/2048)
106             filename = recieveData[4:]
107             if (os.path.isfile('new_' + filename)):
108                 x = 1
109                 while(os.path.isfile('new_' + str(x) + filename )):
110                     x += 1
111                 f = open('new_' + str(x) + filename, 'wb')
112
113             else:
114                 f = open("new_" + filename, 'wb')
115
116             for x in range (0, packetAmmount):
117                 data = conn.recv(2048)
118                 f.write(data)
119
120             f.close()
121
122     elif recieveData[:9] == 'compress ':
123         fileToCompress = recieveData[9:]
124         compress(fileToCompress)
125         conn.send("file specified compressed on server side.".encode())
126
127     elif recieveData == 'quit':
128         x = -1
129         break
130     print("disconnected...")
131     break
132 else:
133     sleep(0.1)
134     conn.send("incorrect".encode())
135
136 print("disconnected...")
137 main()
138
139
140

```

client.py

```
11
2
3 import socket
4 import os
5 import glob
6 from time import sleep
7 from math import ceil
8 from getpass import getpass
9 import zlib
10 import zipfile
11
12 def help():
13     print("\n\n.....Commands Menu.....\n\nhelp          >>    Help Menu.\n")
14
15 def main():
16     PORT = 65427
17     HOST = '127.0.0.1'
18     s = socket.socket()
19     s.connect((HOST, PORT))
20     loginAttempts = 0
21     while loginAttempts < 3:
22         message = s.recv(2048).decode()
23         print(message)
24
25         username = input("Enter your username: ")
26         s.send(username.encode())
27
28         password = getpass(prompt = "Enter your password: ", stream = None)
29         s.send(password.encode())
30
31         answer = s.recv(2048).decode()
32
33         if answer == 'correct':
34             print("\n\nSuccessful login attempt!\nWelcome to Our FTP Server\n")
35             help()
36             while True:
37                 string = input("\nWaiting for Command :: ftp >> ")
38                 s.send(string.encode())
39
40                 if string == 'dir':
41                     x = s.recv(2048).decode()
42                     print(x)
43
44                 elif string == 'help' :
45                     help()
46
```

```

47 elif string == 'ls':
48     x = s.recv(2048).decode()
49     print(x)
50
51 elif string == 'pwd':
52     x = s.recv(2048).decode()
53     print(x)
54
55 elif string[:4] == 'dwd ':
56     response = s.recv(2048).decode()
57     print(response + ' bytes')
58     if(response[:4] == 'file'):
59         filename = string[4:]
60         filesize = int(response[27:])
61         packetAmmount = ceil(filesize/2048)
62         if (os.path.isfile('new_' + filename)):
63             x = 1
64             while(os.path.isfile('new_' + str(x) + filename )):
65                 x += 1
66             f = open('new_' + str(x) + filename, 'wb')
67
68         else:
69             f = open("new_" + filename, 'wb')
70
71         for x in range (0, packetAmmount):
72             data = s.recv(2048)
73             f.write(data)
74
75         f.close()
76         print("Download is complete")
77     else:
78         print("file does not exist...")
79
80 elif string[:4] == 'upl ':
81     filename = string[4:]
82     if os.path.isfile(filename):
83         filesize = int(os.path.getsize(filename))
84         s.send(('true' + str(filesize)).encode())
85         with open(filename, 'rb') as f:
86             packetAmmount = ceil(filesize/2048)
87             for x in range(0, packetAmmount):
88                 bytesToSend = f.read(2048)
89                 s.send(bytesToSend)
90             print("file sent!")
91     else:
92         s.send('false'.encode())
93         print("file does not exist...")
94

```

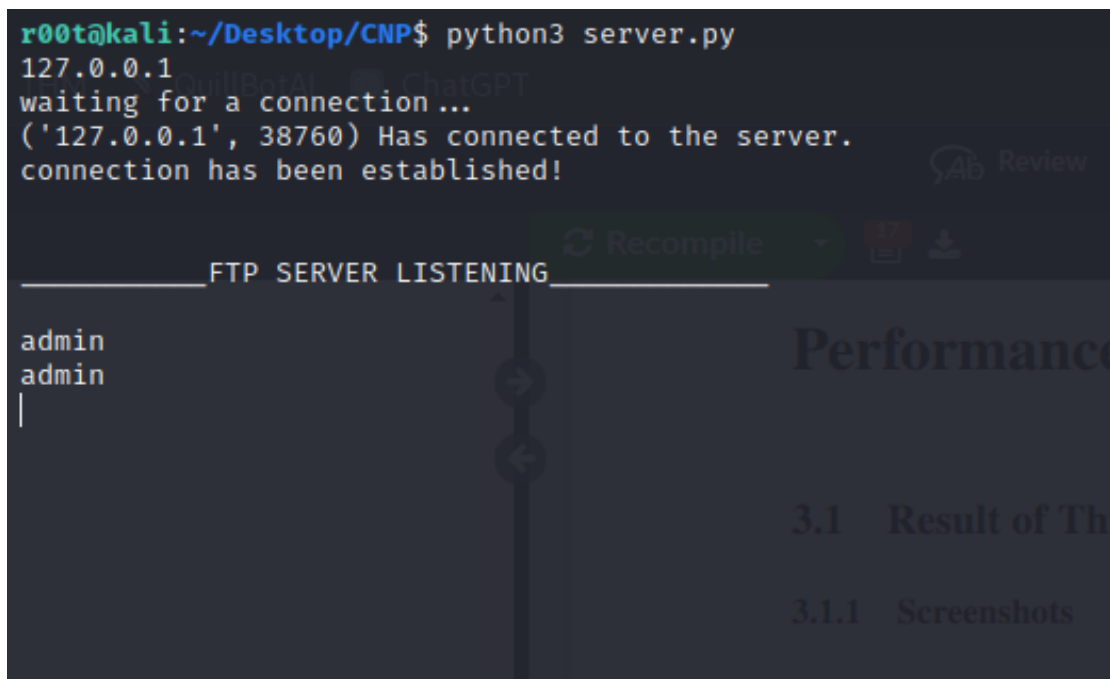
```
95         elif string[:9] == 'compress ':
96             x = s.recv(2048).decode()
97             print(x)
98
99         elif string == 'quit':
100             loginAttempts = 4
101             break
102
103         elif answer == 'disconnect':
104             break
105         loginAttempts += 1
106     print("You have been disconnected...")
107
108 main()
109
110
```

Chapter 3

Performance Evaluation

3.1 Result of This Project

3.1.1 Screenshots



The screenshot displays two overlapping windows. The background window is a terminal with a dark theme, showing the execution of a Python script named 'server.py'. The output indicates that the server is listening on 127.0.0.1 and has successfully established a connection from the same IP address at port 38760. The foreground window is a web browser with a dark theme, showing a page titled 'FTP SERVER LISTENING'. The page content includes the text 'admin' and 'admin' on separate lines, with a cursor positioned at the end of the second line. The browser's address bar shows '127.0.0.1:38760'. The browser interface includes a 'Recompile' button and a 'Review' button in the top right corner.

```
r00t@kali:~/Desktop/CNP$ python3 server.py
127.0.0.1
waiting for a connection...
('127.0.0.1', 38760) Has connected to the server.
connection has been established!
```

FTP SERVER LISTENING

admin
admin
|

Figure 3.1: Server Site Response Window


```
r00t@kali:~/Desktop/CNP/client$ python3 client.py

Please print your username and password to log into the server. You have 3 more tries.
Enter your username: adminconnected to the server.
Enter your password: established!

Successful login attempt!
Welcome to Our FTP Server

.....Commands Menu.....

help      >>    Help Menu.
dir/ls    >>    List Files and Directories.
pwd       >>    Print Working Directory.
upl       >>    Upload File.
dwd       >>    Download File.
compress  >>    Compress File.
quit      >>    Disconnect and Exit.

Wating for Command :: ftp >> |
```

Figure 3.2: Client Site Window

```
admin
admin
Wating for Command :: ftp >> ls
D: Classes
D: Movie
D: cliented...
F: info.txt...
F: my doc.pdf
F: my pic.jpg
F: python tutorial.mp4
F: server.py
F: software.exe

Wating for Command :: ftp >> pwd
/home/r00t/Desktop/CNP

Wating for Command :: ftp >> quit
You have been disconnected...
r00t@kali:~/Desktop/CNP/client$ |
```

Figure 3.3: ls , pwd , and quite command

```
F: python tutorial.mp4
F: server.py
F: software.exe

Waiting for Command :: ftp >> dwd software.exe
file exists with a size of 0 bytes
Download is complete

Waiting for Command :: ftp >> upl script.sh
file sent!

Waiting for Command :: ftp >> compress info.txt
file specified compressed on server side.

Waiting for Command :: ftp >> ls
D: Classes
D: Movie
D: client
F: info.txt
F: info.zip
F: my doc.pdf
F: my pic.jpg
F: new_script.sh
F: python tutorial.mp4
F: server.py
F: software.exe

Waiting for Command :: ftp >> help

.....Commands Menu.....

help      >>  Help Menu.
dir/ls    >>  List Files and Directories.
pwd       >>  Print Working Directory.
upl       >>  Upload File.
dwd       >>  Download File.
compress  >>  Compress File.
quit      >>  Disconnect and Exit.
```

Figure 3.4: dwd, upl , compress , help command

3.1.2 Result Details

At first, we have to run the server file. By running a file server, one will be active and accessible to clients. After that, we can run the client file. After running a client file FTP server, ask users for their username and password. The user can attempt a password three times; if the user enters the wrong password three times, the server will automatically disconnect. After giving the correct password, the user will be able to use our FTP server. After logging in, the client will be presented with a list of commands that he can execute. After that, he can perform download, upload, and compress listing operations, as well as see the server's current directory, and he can disconnect from the server by issuing the disconnect command.

3.1.3 Testing and Discussion

After numerous attempts, all of my project work is flawless. Initially, the client and server are unable to connect. After some trying, I've figured out the problem and solved it. I tested again after finishing everything, and everything worked fine.

Chapter 4

Conclusion

4.1 Discussion

The implementation of an FTP server using Python was a successful project that met the specified goals and requirements. By using the python socket library and following a structured development process, we were able to build a functional FTP server that is capable of handling file transfers and supporting custom features. The server was tested and debugged to ensure that it is reliable and performs well.

4.2 Limitations

- **Performance:** The FTP server may have performance limitations depending on the hardware and software environment in which it is deployed, as well as the number and size of the files being transferred.
- **Security:** The FTP server may have limitations in terms of the security measures it can support, such as encryption algorithms or authentication methods. It may also be vulnerable to security threats such as hackers or malware.
- **Compatibility:** The FTP server may have limitations in terms of the clients and protocols it can support, and may not be able to handle certain features or behaviors.
- **Custom features:** The FTP server may have limitations in terms of the custom features it can support, depending on the complexity and scope of these features.
- **Maintenance and support:** The FTP server may require ongoing maintenance and support to ensure that it remains reliable and secure. This may involve resources such as time, expertise, and budget.

4.3 Scope of Future Work

- Performance optimization: The FTP server may be able to benefit from further performance optimization, such as by improving the efficiency of the file transfer algorithms or by using more powerful hardware.
 - Security enhancements: The FTP server may be able to benefit from additional security enhancements, such as by implementing stronger encryption algorithms or by adding additional authentication methods.
 - Compatibility improvements: The FTP server may be able to support a wider range of clients and protocols by improving its compatibility with these systems.
 - Custom feature development: The FTP server may be able to support additional custom features or functionality by developing new code or integrating with other systems.
 - Maintenance and support: The FTP server may require ongoing maintenance and support in order to ensure that it remains reliable and secure over time. This could involve activities such as bug fixes, updates, and monitoring.
-
-