Data Networks
Final Project: File Transfer Protocol (FTP)

Dr. Mohammad Reza Pakravan

Introduction

FTP, or File Transfer Protocol, is a standard network protocol used for transferring files between a client and server on a computer network. It operates on a client-server model (Fig. [1]), using separate control and data connections. FTP uses two ports, 20 and 21, for data and control information respectively and It can operate in active or passive mode. FTP is a well known protocol for download from and uploading to a server. Here's a simplified explanation on how it works:

1. **Connection Establishment**: In this step, the FTP client establishes a connection with the FTP server. This is done over TCP (Transmission Control Protocol), which is a reliable and ordered delivery of a stream of bytes from a program on one computer to another program on another computer. The client typically connects to the servers port 21, known as the control connection. This connection remains open for the duration of the FTP session, allowing the client and server to exchange commands and responses.

2. **Authentication**: Once the control connection is established, the client needs to authenticate itself to the server. This is typically done by sending a username and password to the server. The server checks these credentials against its access control list. If the credentials are valid, the server sends a positive response and the client is granted access to the server. If the credentials are not valid, the server sends a negative response and the client is denied access. This process is crucial for maintaining the security and integrity of the servers data.

3. **Command Sending**: After successful authentication, the client can send FTP commands to the server over the control connection. These commands are defined in the FTP protocol and tell the server what action the client wants to perform. For example, the LIST command asks the server to

send a list of files in the current directory, the GET command asks the server to send a specific file, and the PUT command asks the server to receive a file from the client. Each command sent by the client is responded to by the server with a reply code and a text message, which indicates the status of the requested command.

4. **Data Transfer**: For commands that involve transferring data, such as GET or PUT, a separate data connection is established. This connection, typically on port 20, is used to transfer the actual file data between the client and server. This separation of command and data connections allows the client to continue sending commands to the server over the control connection while data is being transferred over the data connection. Once the data transfer is complete, the data connection is closed, but the control connection remains open for further commands.

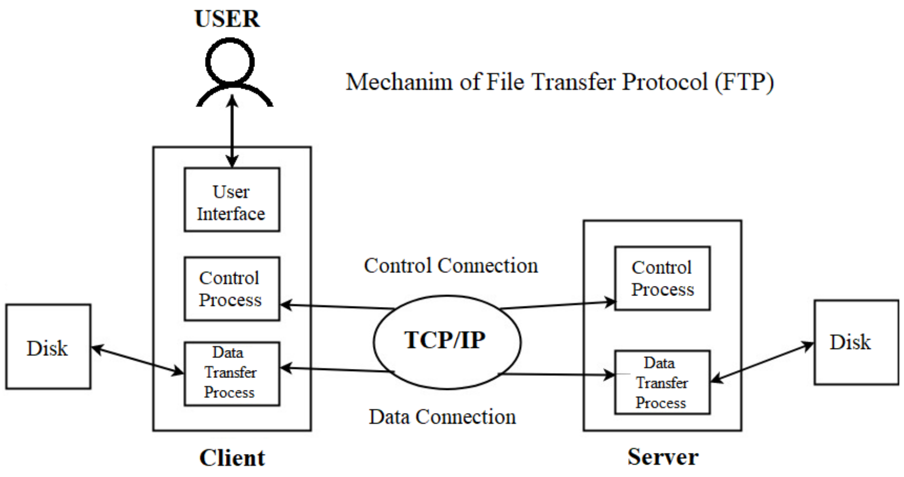
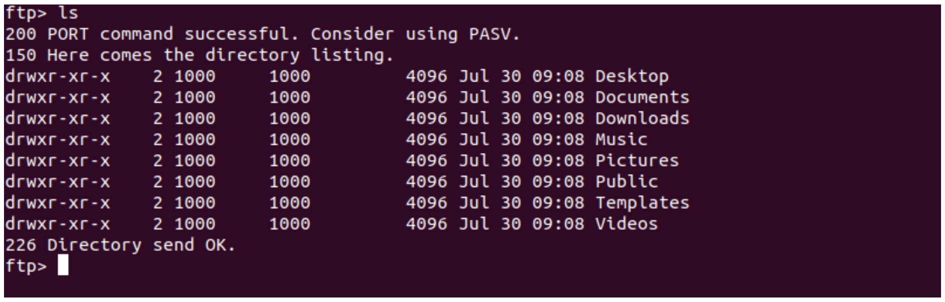


Figure 1: FTP diagram

Part 1: Client and Server Implementation (65 points)

The first step of the project is to implement a custom and simplified FTP server and client. The server needs to be set up to listen on its control port, which is usually 21, but for this project, it should be adjusted to 20021. When a client establishes a connection to this port, the server is expected to interpret its requests and respond in an appropriate format as outlined below.
Each client should be equipped with a command prompt that accepts the commands the user intends to transmit to the server. The server's response (just its description and content) should be appropriately displayed in the prompt. Refer to (Fig. [2]) as an example.
The format of the messages through the control channel is "JSON". A little search can clarify this data structure.
Note that this project is designed to be developed and tested on Linux-based operating systems (such as Ubuntu).
(Hint 1: Socket programming is the way to connect the server to the clients. Python has the socket library for this application.)
(Hint 2: As your server must support concurrent requests from multiple clients, you should use a multi-threaded approach where each thread handles a client.)



```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Desktop
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Documents
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Downloads
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Music
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Pictures
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Public
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Templates
drwxr-xr-x    2 1000     1000         4096 Jul 30 09:08 Videos
226 Directory send OK.
ftp> 
```

Figure 2: FTP client prompt. (Do not focus on the details. Just prompt appearance needs to be considered)

Implementing FTP commands

In this phase, we aim to execute a few of the FTP commands. The procedure for making requests from the client side is quite simple. However, on the server side, the client's requests need to be interpreted and handled. Factors such as whether the user has root access and whether the request can be accommodated will be a concern.
Each command that necessitates data transfer must establish a new data connection. There are two ways to do this, namely the PASSIVE and ACTIVE modes. For this project, we choose to use the PASSIVE mode. In this mode, the server starts the connection on a random port. After the initialization process is finished, the client obtains the servers data transfer port, and depending on the command, one of the nodes starts the data transmission. Its crucial to understand that the data link should not be persistent. In other words, the connection should automatically terminate once a file has been transmitted over it. The data transmission should not employ any data structures like JSON, as these are required for control commands.
During the data transfer process between the client and server, display the specifics of the transmitted file. These specifics include the files name and its size. Throughout all operations, on both ends (the server and client), any errors should be displayed clearly in the servers terminal or the clients prompt.

Authentication

Every user should establish a connection to the server via the control port. However, for certain privileged requests (such as PUT, MKD, RMD, and DELE), root access is required. To request root access, first, the user should call the command in the prompt:

```
    ftp> ath USR PASS
```

and then, the client should transmit the following JSON:

```
Client_String : {
    Cmd: "AUTH",
    User: "USR",
    Password: "PASS"
    }
```

Based on the server's verification, the client should receive <u>one</u> of these messages:

```
# The user has root access
Server_String : {
    StatusCode: 230,
    Description: "Successfully logged in. Proceed"
    }
# Otherwise
Server_String : {
    StatusCode: 430,
    Description: "Failure in granting root accessibility"
    }
```

Quit command

Using this command the client closes the control connection with the server.

```
    ftp> quit
```

and then, the client should transmit the following JSON:

```
Client_String : {
    Cmd: "QUIT"
    }
```

after this command, both nodes should close their connections.

List Command

This command acts as "ls -l" in the current server file directory. The user should enter the following command:

```
    ftp> ls
```

The client should send this message:

```
Client_String : {
    Cmd: "LIST"
    }
```

If the directory is not empty the server should open a new data connection on its random range and aware the client of this port.

```
# If the directory is not empty (case 1)
Server_String : {
    StatusCode: 150,
    Description: "PORT command successful",
    DataPort: port_number
    }
# Otherwise (case 2)
Server_String : {
    StatusCode: 210,
    Description: "Empty"
    }
```

When the first scenario (case 1) takes place, the server transmits the result of the ls -l command to the client over the data connection. In this case, the server sends a message via the control connection:

```
# It would be better to print its description after the "ls" output in the client's prompt
Server_String : {
    StatusCode: 226,
    Description: "Directory send OK"
    }
```

GET (RETR) Command

Using this command the client can download a file from the server.
Issuing command by the user:

```
    ftp> get fName
```

The messages between the client and server should be as follows:

```
# The client sends the get command
Client_String : {
    Cmd: "GET",
    FileName: "fName"
    }
```

The server needs to see if the file exists to send an appropriate response:

```
# The file is present (case 1) in the server directory
Server_String : {
    StatusCode: 150,
    Description: "OK to send data",
    DataPort: port_number
    }
# The file doesn't exist. (case 2)
Server_String : {
    StatusCode: 550,
    Description: "File doesn't exist"
    }
```

In the first scenario (case 1), a new data connection should be established and the server should start sending the file through this connection.

```
# This message should be sent as the server transmitted the requested file
Server_String : {
    StatusCode: 226,
    Description: "Transfer complete"
    }
```

PUT Command

An administrator or a user with root access is able to upload a file to the server using the PUT command. The server should verify whether this user has the necessary root permissions to carry out the requested command and accept a file from them.
Issuing command by the user :

```
    ftp> put fName
```

The messages between the client and server should be as follows:

```
# The client sends the PUT command
Client_String : {
    Cmd: "PUT",
    FileName: "fName"
    }
```

The server is required to verify whether the user has privileged (root) access. Depending on the verification result, it should then dispatch a corresponding message:

```
# The user has root access (case 1)
Server_String : {
    StatusCode: 150,
    Description: "OK to send data",
    DataPort: port_number
    }
# otherwise. (case 2)
Server_String : {
    StatusCode: 434,
    Description: "The client doesn't have the root access. File transfer aborted."
    }
```

In the first scenario (case 1), a new data connection should be established and the client starts sending the file through this connection.

```
# As the server received the file from the client (admin), this message should be delivered
    to the client
Server_String : {
    StatusCode: 226,
    Description: "Transfer complete"
    }
```

DELE Command

As an admin, a user can delete a file from the server. Just like PUT command the server is meant to verify the user accessibility.
Issuing command by the user:

```
    ftp> delete fName
```

The messages between the client and the server should be as follows:

```
# The client sends the DELE command
Client_String : {
    Cmd: "DELE",
    FileName: "fName"
    }
```

The server is required to verify whether the user has privileged (root) access. Depending on the verification result and existence of the file, it should then dispatch a corresponding message:

```
# The user has root access (case 1)
Server_String : {
    StatusCode: 200,
    Description: "Successfully deleted"
    }
# The user doesn't have root access. (case 2)
Server_String : {
    StatusCode: 434,
    Description: "The client doesn't have the root access."
    }
# The file doesn't exist. (case 3)
Server_String : {
    StatusCode: 550,
    Description: "File doesn't exist"
    }
```

Multiple PUT Command

A user with privileged (root) access can upload multiple files simultaneously using this command. You have the option to utilize a single data connection for transmitting multiple files, or you can use one data connection for each file you send.

Issuing command by the user:

```
    ftp> mput fName1,fName2,...,fNameM
```

The messages between the client and the server should be as follows:

```
# The client sends the MPUT command
Client_String : {
    Cmd: "MPUT",
    FileName_1: "fName1",
    FileName_2: "fName2",
    .
    .
    FileName_M: "fNameM"
    }
```

The server is required to verify whether the user has privileged (root) access. Depending on the verification result, it should then dispatch a corresponding message:

```
# The user has root access (case 1)
Server_String : {
    StatusCode: 150,
    Description: "OK to send data",
    DataPort: port_number
    }
# The user doesn't have root access. (case 2)
Server_String : {
    StatusCode: 434,
    Description: "The client doesn't have the root access."
    }
```

In the first scenario (case 1), a new data connection should be established and the client starts sending the files through this connection.

```
# As the server received the file X from the client (admin), this message should be
    delivered to the client
Server_String : {
    FileName :"fileNameX",
    StatusCode: 226,
    Description: "Transfer complete"
    }
```

Part 2: FTP Protocol Understanding (15 points)

Answer the following questions thoroughly:

1. Investigate other protocols that are used for file transfer and compare them with FTP.
2. What is the commonly used transport protocol for file transfers? Is it possible to use UDP as the transport layer protocol?
3. What are the drawbacks of FTP that have made it fairly obsolete on the modern web?
4. What are the disadvantages of using active mode FTP? How the passive mode can handle these problems?
5. In FTP, how is data transfer security guaranteed? Is there even a default security measure for FTP? Investigate SFTP, FTPS, and FTP over SSH protocols in terms of their security.

6. With Wireshark, you can observe network packets traversing any network interface. Use Wireshark to investigate packets while you are downloading a file from your FTP server. Note that you should run FTP on the loopback interface. What is the maximum size of a TCP packet containing data? Use the following command on Linux-based machines to create an arbitrary large file.

   ```
   # create a 10G file named file.txt
   fallocate -l 10G file.txt
   ```

   Include screenshots of packets captured by Wireshark in your report.

7. Did you know that Sharif University hosts an FTP server where you can download useful content like engineering programs, drivers, and so on? To visit it, first, you need to set up your Sharif VPN to access it when you are not on campus. Then, refer to this website and you can download tools you might need. While downloading a file from Sharif FTP, run Wireshark and observe the received packets. Are the captured packets similar to the previous part? If not, explain the difference.

Part 3: Setting Up a Local FTP Server on Ubuntu (20 points)

In this part, we will set up an FTP server using `vsftpd` on Ubuntu, configure users, and secure the connection.

Part A: Install and Configure FTP Server

1. Install the `vsftpd` package. After installing `vsftpd`, configure it to start the FTP server.
2. Create three dedicated FTP users and set up passwords for them.
3. Connect to the FTP server using an FTP client or command line, and take screenshots of your commands and results.

Part B: Configure Firewall

1. Verify if the firewall is active on your system.
2. Configure the firewall to allow FTP traffic.

- **Questions:**
  - Why is it important to configure firewall rules for FTP traffic?
  - What are the specific ports used for FTP, and how do you open them in the firewall?

Part C: Change Default Directory

1. Before making any changes, back up your `vsftpd` configuration files. How can you back up the configuration files?
   - **Question:** Why is it important to back up configuration files before editing them?
2. Modify the `vsftpd` configuration to change the default directory for FTP users. Create a new directory for the FTP server.
   - **Question:** Why is it beneficial to change the default directory for FTP users?
3. Create a new file in the new directory and check if it is accessible from the FTP client.

Part D: Securing FTP

1. **Encrypt FTP Traffic:** Configure the FTP server to use SSL/TLS encryption to secure the connection (provide code, but do not implement it).
   - **Questions:**
     - Why is it important to encrypt FTP traffic?
     - What are the benefits of using SSL/TLS encryption for FTP?
2. **Limit User Access:** Implement measures to limit user access to specific directories and functionalities to enhance security.
   - **Questions:**
     - Why do we need to secure FTP servers?
     - What are the methods to limit user access on an FTP server?

Part 4: Bandwidth and Transfer Rate Control (15 points Bonus)

In this part you will Implement bandwidth and transfer rate control in your FTP server using Python to ensure that file transfers do not exceed a specified rate. This will help in managing network resources more effectively and prevent the server from being overwhelmed by high-speed data transfers.

Question: Why is bandwidth control important in network applications?

Tasks

1. **Measurement of Transfer Rate:**
   - Track the amount of data being transferred and the time taken to calculate the current transfer rate.
   - Use this information to monitor the transfer speed in real-time.(print the progress percentage of transmission and transmission rate)
2. **Limiting Bandwidth:**
   - Introduce a maximum bandwidth limit (e.g., 100 KB/s).
   - If the transfer rate exceeds this limit, introduce a delay to throttle the speed of data transfer.
3. **Implementation Steps:**
   - Modify the server's file sending function to incorporate transfer rate monitoring and bandwidth throttling.
   - Ensure the client handles potentially slower data transfer rates smoothly.

Implementation Details

- **Server Side:**
  - Track the number of bytes sent and the elapsed time during file transfer.
  - Calculate the transfer rate and introduce delays if it exceeds the maximum bandwidth limit.
- **Client Side:**
  - The client will request a file and handle data reception as normal. No significant changes are required on the client side, but it should handle slower data reception gracefully.

Restrictions

Part 1. The use of high-level libraries for this task is not allowed. You are permitted to use libraries for socket programming and multi-threading.

Part 2. None

Part 3. None

Part 4. You are only allowed to use `socket`, `threading` and `time` libraries.

What should I do?

Part 1. Your task is to set up two directories for storing the client and server files. Initiate the server on the local host using port 20021 and attempt to establish connections with multiple clients. For testing purposes, your clients and server should be capable of transferring various types of files, such as images, text documents, audio files, and so on. This part does not need any report.

Part 2. Take screenshots of the Wireshark output and include it alongside the answers to the questions in the report file.

Part 3. Take screenshots of your code and the results, and answer questions in your report.

Part 4. Update codes of part 1 to satisfy the bandwidth limiting and explain briefly what you did in the report. Provide screenshots from the output containing transmission rate of data.

Compress all files and rename the compressed file to STUDENT_ID_PROJECT.zip. If you have any questions regarding the problem statement or understanding the concept, feel free to ask.

Good Luck !