

NATIONAL INSTITUTE OF TECHNOLOGY, DELHI



Network Programming using Socket API

**Assignment By : Moulik Sharma, Muhammed
Hisham, Rohit Kumar**

Roll Number : 211210039, 211210040, 211210053

Subject Code : CSB 351

Date Submitted : 21 Apr 2024

Submitted To : Dr Preeti Mehta

Department of Computer Science and Engineering

Problem Statement: To design and implement a simplified File Transfer Protocol (FTP) using the Socket API in Python. The goal is to create a basic FTP system that allows users to upload and download files to and from a central server. Focus on the functionality of transferring files securely and efficiently.

Use Cases:

1) Connection:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\dell\Desktop\Projects\FTP_Server\Server> python server.py
Welcome to the FTP server.
To get started, connect a client.
Connected to by address: ('10.10.49.91', 54419)
Connected to by address: ('10.10.48.227', 49378)
Authentication failed for user: Hisham
█

PS C:\Users\dell\Desktop\Projects\FTP_Server> cd Client
PS C:\Users\dell\Desktop\Projects\FTP_Server\Client> python client.py
Enter server IP address: 10.10.48.227

Welcome to the FTP client.

Call one of the following functions:
CONN      : Connect to server
UPLD file_path : Upload file
LIST      : List files
DWLD file_path : Download file
DELF file_path : Delete file
RDLOG     : Read log
CLRLOG    : Clear Log
QUIT      : Exit

Enter a command: conn
Sending server request...
Connection successful

Choose an action.
LOGIN or SIGNUP: █
```

2) Signup:

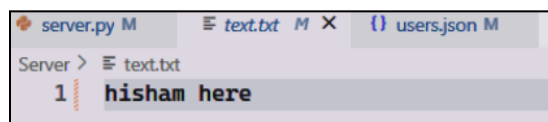
```
Enter a command: conn
Sending server request...
Connection successful

Choose an action.
LOGIN or SIGNUP: SIGNUP
Enter your username: Trex_007
Enter your password: admin123█
```

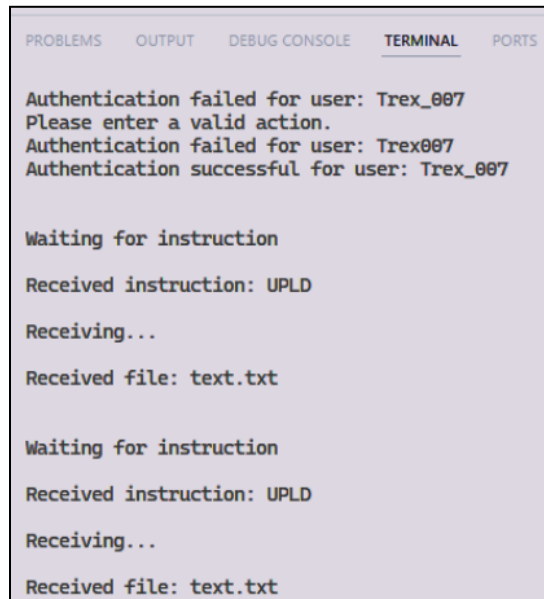
3) Login:

```
Choose an action.  
LOGIN or SIGNUP: LOGIN  
Enter your username: Trex_007  
Enter your password: admin123  
Authentication successful  
  
Enter a command: 
```

4) Upload (Server-Side):



The screenshot shows a code editor with three tabs: 'server.py', 'text.txt', and 'users.json'. The 'text.txt' tab is active, showing the content 'hisham here' on line 1.



The screenshot shows a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
Authentication failed for user: Trex_007  
Please enter a valid action.  
Authentication failed for user: Trex007  
Authentication successful for user: Trex_007  
  
Waiting for instruction  
Received instruction: UPLD  
Receiving...  
Received file: text.txt  
  
Waiting for instruction  
Received instruction: UPLD  
Receiving...  
Received file: text.txt
```

5) Upload (Client-Side):

```
Enter a command: upld text.txt  
  
Uploading file: text.txt...  
  
Sending...  
  
Sent file: text.txt  
Time elapsed: 0.17510724067687988s  
File size: 11b
```

6) List:

```
Enter a command: list
Requesting files...

auth.py - 889b
log.py - 912b
log.txt - 511b
server.py - 8432b
text.txt - 11b
users.json - 167b
__pycache__ - 4096b
Total directory size: 15018b
```

7) RDLOG:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Waiting for instruction
Exception in thread Thread-1 (handle_client):
Traceback (most recent call last):
  File "C:\Python311\Lib\threading.py", line 1038, in _bootstrap_inner
    self.run()
  File "C:\Python311\Lib\threading.py", line 975, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Users\dell\Desktop\Projects\FTP_Server\Server\server.py", line 182, in handle_client
    data = conn.recv(BUFFER_SIZE).decode()
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ConnectionResetError: [WinError 10054] An existing connection was forcibly closed by the remote host

Connected to by address: ('10.10.48.227', 49551)
Authentication successful for user: Trex_007

Waiting for instruction
Received instruction: RDLOG

Waiting for instruction
█
```

```
Call one of the following functions:
CONN      : Connect to server
UPLD file_path : Upload file
LIST      : List files
DWLD file_path : Download file
DELF file_path : Delete file
RDLOG     : Read log
CLRLOG    : Clear Log
QUIT      : Exit

Enter a command: conn
Sending server request...
Connection successful

Choose an action.
LOGIN or SIGNUP: LOGIN
Enter your username: Trex_007
Enter your password: admin123
Authentication successful

Enter a command: RDLOG
2024-04-18 12:31:03.694276 - UPLOAD: Uploaded file 'text.txt' by ''
2024-04-18 12:31:36.146861 - DOWNLOAD: Downloaded file 'text.txt' by ''
2024-04-18 12:31:58.876793 - DELETE: File 'text.txt' was deleted 'by ''
2024-04-18 13:18:46.849142 - DOWNLOAD: Downloaded file 'log.txt' by ''
2024-04-18 13:19:09.138113 - UPLOAD: Uploaded file 'text.txt' by ''
2024-04-21 17:57:28.090864 - UPLOAD: Uploaded file 'text.txt' by 'new_user'
2024-04-21 17:57:46.808812 - UPLOAD: Uploaded file 'text.txt' by 'new_user'
```

8) CLRLOG:

```
Enter a command: CLRLOG

Enter a command: RDLOG
```

Libraries Used:

- **Socket Library:** We used the socket library to establish network connections between the FTP server and clients. With this library, we were able to create sockets for communication, allowing clients to connect to the server and transfer files securely.
- **os Library:** Leveraging the os library, we managed file operations within the FTP server. This library provided us with functions to handle file uploads, downloads, and directory listings. We could interact with the file system, ensuring smooth file management processes.
- **struct Library:** The struct library played a crucial role in encoding and decoding data structures for network communication. By using this library, we could pack and unpack binary data, facilitating the transmission of file details such as file names and sizes between the server and clients.
- **json Library:** With the json library, we stored and retrieved user credentials securely in a JSON file. This library allowed us to parse JSON data, ensuring that user authentication information was managed efficiently and safely.
- **datetime Library:** Utilizing the datetime library, we generated timestamps for logging events within the FTP server. This library enabled us to record the exact date and time of user actions, providing valuable insights into system activities.
- **time Library:** We employed the time library to measure the elapsed time during file transfers. By utilizing functions from this library, we could calculate the duration of file uploads and downloads, optimizing performance and user experience.
- **threading Library:** Leveraging the threading library, we implemented multithreading support in the FTP server. This library allowed us to handle multiple client connections concurrently, enhancing the scalability and responsiveness of the server.

Implementation Steps:

1) Setting Up Server-Client Architecture:

Server Configuration:

- We started off by creating a Python script named server.py.
- Using Python's socket library, we established a server socket to listen for incoming connections.
- Defined constants for the server IP address, port number, and buffer size to facilitate communication.
- With threading, we managed to handle multiple client connections simultaneously, ensuring efficient server operation.

Client Configuration:

- In parallel, we developed a client script named client.py.
- Leveraging the socket library, we enabled clients to connect to the server and perform file transfer operations.
- We set up constants for the server IP address, port number, and buffer size to ensure seamless communication.
- We also implemented functions in the client script to facilitate connection establishment and command transmission to the server.

2) Implementing User Authentication:

- We created an authentication module named auth.py to handle user authentication.
- Utilizing Python's json library, we securely stored user credentials in a JSON file named users.json.
- We implemented functions within the module to authenticate users, add new users, and delete existing users.
- Before storage, we ensured that the passwords were hashed to enhance security.

3) Handling File Transfer Operations:

Uploading and Downloading Files:

- We developed functions in the server script (server.py) to manage file upload (upld) and download (dwld) operations.
- We utilized struct for encoding and decoding data structures during file transfer.
- We also ensured that files were transmitted in chunks, guaranteeing efficient and reliable transfer between the server and clients.

Deleting Files:

- We implemented a function (delf) in the server script to handle file deletion requests from clients.
- We also validated file existence and performed deletion operations after authentication and permission checks.

4) Implementing Directory Listing:

- Within the server script (server.py), we created a function (list_files) to list the contents of the server's file directory.
- We also sent the list of available files, along with their sizes, to clients, allowing them to view and select files for download or deletion.

5) Logging and Auditing:

- We developed a logging module named log.py to record important events such as user logins, file transfers, and deletions.
- We also utilized Python's datetime library to generate timestamps for logging entries, aiding in tracking and auditing system activities.

6) Error Handling:

- Throughout the development process, we implemented robust error handling mechanisms to address authentication failures, invalid commands, and file transfer interruptions.

- We also ensured that informative error messages were provided to users, guiding them through troubleshooting procedures.

7) Testing:

- We conducted comprehensive testing to verify the reliability and security of the FTP application.
- We also designed test cases covering various scenarios, including successful file transfers, authentication processes, error handling, and logging functionalities, ensuring the robustness of the system.

Conclusion: The implemented FTP system offers a secure and efficient file transfer solution, suitable for small-scale file management tasks. Future enhancements could include additional features such as permission management, enhanced logging capabilities, and improved user interfaces to further enhance usability and functionality.

Future Scope: Although not explicitly implemented yet, we considered the development of a user interface (CLI or GUI) to enhance the user experience. We planned features such as connecting to the server, managing file transfers, sessions, and viewing logs to provide users with a more intuitive interaction environment.

Contribution:

- **Moulik Sharma:** Documentation & Error Handling
- **Muhammed Hisham:** Server Implementation & Authentication
- **Rohit Kumar:** Client Implementation & Logging

THANK
YOU!