Report z5494973 : BitTrickle Peer-to-Peer File Sharing System with a centralized indexing server.

1. Language and Code Organization

- **Language:** Python 3
- **File Structure:**
  - server.py: Implements the central server for indexing, authentication, and file management in the network.
  - client.py: Implements the client functionality, allowing users to interact with the server and peer nodes.
- **Execution Instructions:**
  - **Server**: python3 server.py <server_port>
  - **Client**: python3 client.py <server_port>
- **Dependencies:** Both files rely on Python standard libraries (socket, threading, argparse, os, and time).

2. Program Design Overview

Server (server.py):

- Server.py only has the main thread – **one class** which is the UDP_server class:
  - Manages user authentication.
  - Responds to 8 of the client requests like file publication, searching for files, and locating peers.
  - Instead of keeping a separate thread to check the heartbeat every second, the code instead checks the heartbeat/ if a client is active, before a processing a client's request. Therefore if a client is inactive, we would have removed the client from the 'active client data structure', before processing the client's request.
- Server.py keeps track of which users have published files and which files are active in the network (see **3. Data Structure Design below)**

Client (client.py):

- This code manages TCP and UDP connections, with UDP used for server communication and TCP for peer-to-peer file transfer. It interacts with the server to join the peer-to-peer network.
- At any point in time a minimum of three threads is active on client.py, client.py is divided into **three classes:**
  - Main thread (UDP_Client class) – handles authentication to server and process the 7 commands- publishing files, searches for available files, and retrieves files directly from other peers (get, lap, lpf, pub, sch, unp, xit).
  - Heartbeat thread class – sends a heartbeat to the server every 2 second to let server knows that the client is active
  - TCP_Server thread class – which actively listen for incoming request from other peers and spawns a thread to handle each client. For this a pool thread is

implemented – where there is 5 threads open to handle client's incoming request which are queued into the pool.

- Since client.py is a multithreaded program, a graceful_shutdown helper function is implemented to ensure that any point of the program, a graceful shutdown is implemented, all resources are releases appropriated and no thread are left unfinished.

Component Interaction between server.py and client.py

1. **Authentication**: Clients authenticate with the server using a username and password, which is verified against a stored list in credentials.txt.
2. **File Sharing and Searching**: Authenticated clients can publish or unpublish files on the server and search for available files by name or substring.
3. **File Transfer**: After a search, if a file is found with an active peer, server sends the TCP address for the client, so the client can establishes a TCP connection with the peer to download the file directly.
4. **An example interaction would be something like this:**
   - "Client A" informs the server that they wish to make "X.mp3" available to the network.
   - The server responds to "A" indicating that the file has been successfully indexed.
   - "Client B" queries the server where they might find "X.mp3".
   - The server responds to "B" indicating that "A" has a copy of "X.mp3".
   - "B" establishes a TCP connection with "A" and requests "X.mp3".
   - "A" reads "X.mp3" from disk and sends it over the established TCP connection, which "B" receives and writes to disk.

3. Data Structure Design

3 data structure is implemented on the server-sides:

- **Credentials Dictionary (self.credentials)**: Stores username -> password mappings for authentication purposes.

```
self.credentials[username] = password
```

- **Active Clients Dictionary (self.active_clients)**: Tracks each active client's UDP address, TCP address, and last heartbeat timestamp, using a dictionary format:

```
self.active_clients[username] = {
        "udp_address" : client_address
        "last_heartbeat": timestamp
        "tcp_address": client tcp address
}
```

- **Published Files Dictionary (self.published_files)**: Maintains mappings of filename -> set of usernames, tracking which users have published which files. This helps the server quickly identify peers sharing a specific file.

```
self.published_files[username] = set(user1, user2, user3,….)
```

The client don't maintain state, so client.py don't have any data structure.

4. Application Layer Protocol Message Format(s)

Client Requests: Clients communicate with the server using simple text-based commands over UDP. Each request from client has their **username** as part of the protocol, this identified the client uniquely, as their ip address and port number could possibly change during relogin/ them using another computer/interface.

Commands:

- login <username> <password> <tcp_port>: Authenticates the client.
- heartbeat <username>: Keeps the client connection alive.
- lap <username>: list all the active peers in Bittrickle.
- pub <username> <filename>: Publishes a file for sharing.
- unp <username> <filename>: Unpublishes a previously shared file.
- lpf <username>: list of all published files from the current peer
- sch <username> <substring>: Searches for files containing a substring in their name.
- get <username> <filename>: Requests the location of a peer sharing the file.

Server Responses:

- Success or error messages, formatted as OK or Error messages, are returned to the client in response to each command.
- For file requests, the server sends a 200 <peer_ip> <tcp_port> response to indicate where the file can be downloaded.

5. Known Limitations

- **Concurrency**: The program relies on threads for handling multiple client connections, which may limit performance under high concurrent usage. For this assignment, only 5 thread workers is used to concurrently accept client request on the client.py. Under the marking condition there will be at most 5 clients concurrently.
- **File Existence Validation**: The server does not verify the existence of files before they are published, this assumes the client has the file in their current working directory, a check is done to ensure this
- **No Advanced Error Handling**: Error handling is basic; network disconnections or message parsing errors could result in unhandled exceptions.
- **Peer-to-Peer File Transfer Stability**: The client-to-client TCP connection may be reset if there is an interruption during file transfer, and no retries protocol when this happens are implemented.
- **Assumption**: all assumption from the project specification are also taken into consideration and taken as a black-box

6. Code Sources

- **Original Code**: Majority of the code was custom written for this assignment, following the project specifications.
- **External Sources**: Some basic socket programming patterns and thread management were inspired by Python's official documentation and examples on socket and threading.
- **Multihreading Codes:** took heavy inspiration from Webcms Multithreaded starter code and Tutorial programming from lab 7 to help write the multithreaded program
- **Threadpool:** I took codes from the Python official documentation, https://docs.python.org/3/library/concurrent.futures.html and took inspiration from https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_pool_of_threads.htm