Comp3331 Assignment Report

Language used: Python

Submitted: Client.py and Server.py in one directory

Organization of code: Client.py, Server.py

Overall Program Design

Server

Data Structures
Online Clients: This dictionary maps each client's address to a tuple containing the username, the last heartbeat timestamp, and the TCP listening port.

```
onlineClients = {}  # Example: {clientAddress} : ({username}, {heartbeat}, {TCPlisteningPort})
```

User Files: This dictionary maps each username to a list of files that the user has published.

```
userFiles = {}  # Example: {username} : {[listOfFiles]}
```

Threads

The server uses two threads to manage its operations:

- **Thread 1**: Constantly monitors online users, removing clients that have not sent a heartbeat in the last 3 seconds. This ensures that the server maintains an accurate list of active users.
- **Thread 2**: Listens for incoming UDP messages from clients and executes the appropriate commands, such as login, peer list, publish files, list published files, unpublish files, search for files, and retrieve files.

Helper Functions

- **alreadyOnline(username, onlineClients)**: Checks if a user is already logged in to prevent duplicate logins.

- **process_login(userPassConnection, clientAddress, onlineClients)**: Validates user credentials against a stored file as well as uses alreadyOnline to check if the user is already online to handle login requests
- **monitorOnlineUsers()**: Regularly checks for inactive clients and removes them from the `onlineClients` dictionary.
- **login(message, clientAddress, onlineClients, userFiles)**: Processes login requests, adding valid users to the online clients.
- **lap(onlineClients, clientAddress, serverSocket)**: Responds to requests for the list of active users in the network.
- **heartbeat(onlineClients, clientAddress)**: Updates the last heartbeat timestamp for active users.
- **pub(userFiles, file, user, serverSocket, clientAddress)**: Allows users to publish files to the server.
- **lpf(userFiles, user, clientAddress, serverSocket)**: Returns a list of files published by a user.
- **unp(userFiles, clientAddress, serverSocket, user, file)**: Removes a file from a user's published list.
- **sch(substring, userFiles, clientAddress, serverSocket, currUser):** Searches for files across all users based on a substring.
- **get(targetFile, userFiles, serverSocket, clientAddress, onlineClients)**: Provides the IP address of a peer who has the requested file.

Client

Threads

The server uses three threads to manage its operations:

- **Thread 1**: sends a heartbeat to the server every 2 seconds
- **Thread 2**: Listens for incoming TCP connections from the listening port and then creates TCP connections.
- **Thread 3:** sends UDP messages to the server based on client commands (input)

Helper Functions

- **newTCPconnections(clientTCPListeningSocket)**: listens for new TCP connections and then creates a new connections to receive information and write it into a file from that new connection
- **heartbeat():** sends a heartbeat to the client every 2 seconds
- **requestFile()**: request a file from the client with the target file and receives the bytes of data and writes it into their own file
- **primary():** Loop for commands by user and sending the messages to the server

Before entering the threads there is an unverified loop that will occur unless the user successfully logs in.

Application Layer Protocol Message Format

The client communicates with the server using a simple text-based protocol with the following message formats:

- **Login**: `"Login:" + userName + " " + userPassword + " " + str(listeningSocketAddress)`
- **Heartbeat**: `"heartbeat"`
- **List Active Peers**: `"lap"`
- **Publish File**: `"pub" + filename`
- **List Published Files**: `"lpf"`
- **Unpublish File**: `"unp" + filename`
- **Search Files**: `"sch" + substring`
- **Get File Information**: `"get" + filename`

The server simply responds with the request i.e will just send a string containing the list of active peers however for successful get requests a message formatted like this will be sent

- **Successful get**: "ClientTCPListeningAddress:" + destinationTCPlisteningAddress + "/" + targetFile

Known Limitations

- The server does not handle UDP packet loss, meaning clients may not receive responses for certain requests if packets are dropped.
- Client can only receive max 1024 bytes from the server in a single call from a function
- The current implementation lacks security measures, such as encryption or authentication beyond simple username/password checks.
- 10 maximum number of queued connections that a client will allow for incoming client connections
- The host for the client and server hard coded as the local host therefore all transfers can be only be done locally from different ports