Digital Photography

SPRING 2023

Computer Network Programming

Final Project

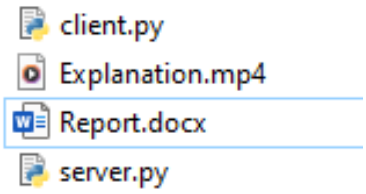
Akif Tunç - 190315078

Submission Date: 15 June 2023

General Information:

The objective of this homework is to establish multiple communications between two servers and multiple clients. The project involves creating a multi-server, multi-client music streaming system. There are two servers responsible for storing MP3 music files, and multiple clients want to access and listen to songs in MP3 format. The availability of songs on the servers is not guaranteed. To ensure efficient usage of resources, clients need to send a message to the servers to check the existence of a specific song. If either server responds positively, the client can request to download or stream the song from that server. If both servers have the requested song, the client can choose its preferred server. However, if neither server has the file, the client must cancel the request. The system relies on TCP protocol for communication and maintains a globally available song list that is consistent across servers and clients.

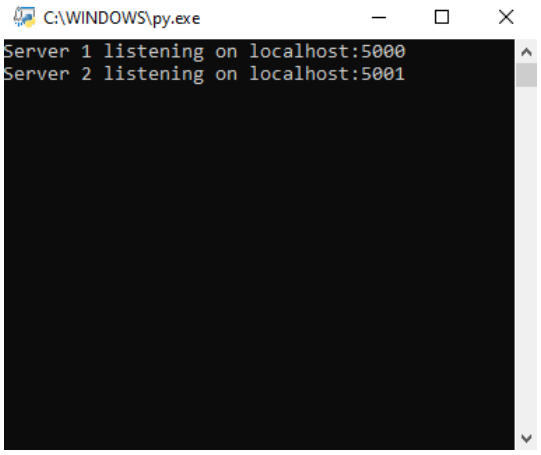
How project works?

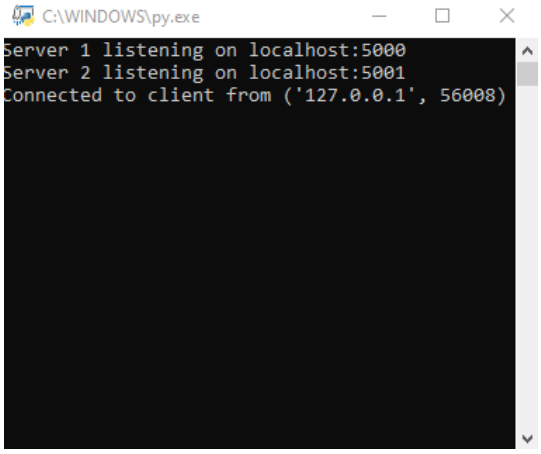


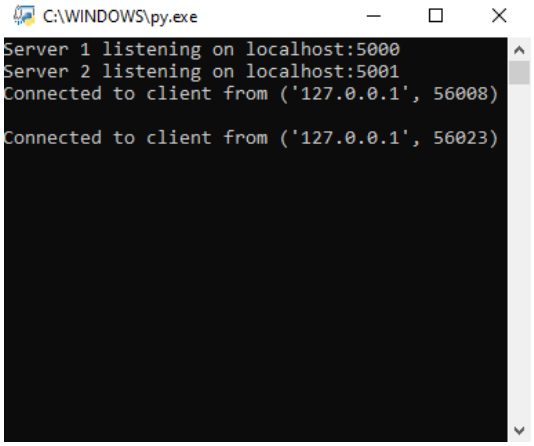
Firstly, we need to open the server.py file and activate the servers. If we open the client.py file while the servers are not active, the client will close itself because it cannot find an active server to send requests to. After opening the server.py file, we can open as many clients as we want. Each of these clients can establish a connection with the server.

Step By Step Outputs:

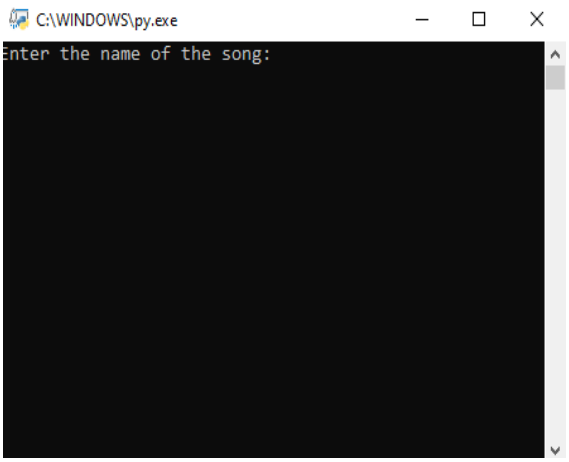
Server.py

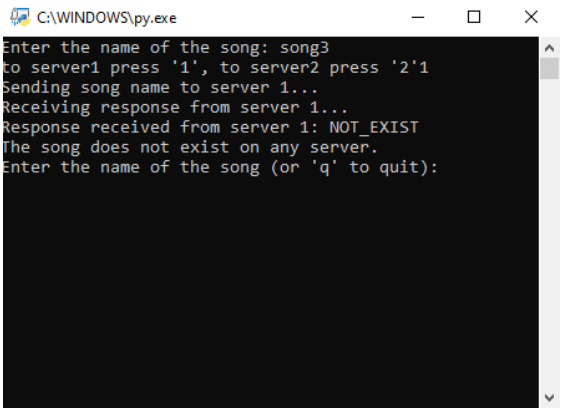


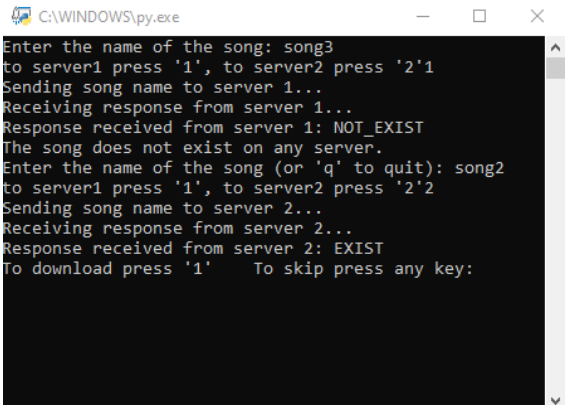


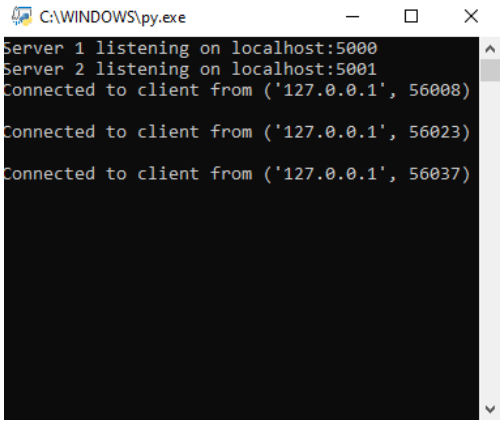


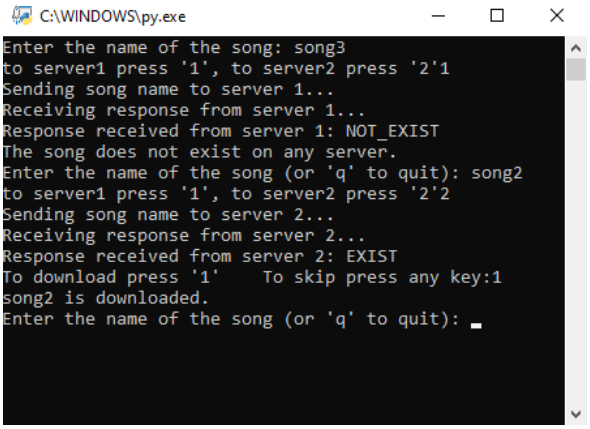
Client.py











We can open as many clients as we want and establish connections with the server using these clients.

Code Explanation:

Server.py:

This code sets up the server configuration by defining the host address (localhost) and two port numbers (5000 and 5001). It also creates a shared song list containing the names of available songs (song1, song2, and song3).

```python
import socket
import threading

# Server configuration
HOST = 'localhost'
PORT1 = 5000  # Server 1 port
PORT2 = 5001  # Server 2 port

# Shared song list
song_list = ['song1', 'song2', 'song3']
```

```python
def server1_handler(conn):
    available_list = song_list[0:2]
    data = conn.recv(1024).decode()
    if data in available_list:
        conn.send(b'EXIST')
    else:
        conn.send(b'NOT_EXIST')
```

```python
def server2_handler(conn):
    available_list = song_list[1:3]
    data = conn.recv(1024).decode()
    if data in available_list:
        conn.send(b'EXIST')
    else:
        conn.send(b'NOT_EXIST')
```

The server1_handler function receives a song name from a client through the conn connection. It checks if the song exists on Server 1 by comparing the received song name with the first two elements of the song_list. If the song exists, it sends the response 'EXIST' back to the client through the conn connection. Otherwise, it sends the response 'NOT_EXIST'. All of them are same as server2_handler.

```python
def start_servers():
    server1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server1.bind((HOST, PORT1))
    server2.bind((HOST, PORT2))
    server1.listen()
    server2.listen()
    print(f"Server 1 listening on {HOST}:{PORT1}")
    print(f"Server 2 listening on {HOST}:{PORT2}")

    while True:
        conn1, addr1 = server1.accept()
        print(f"Connected to client from {addr1}")
        threading.Thread(target=server1_handler, args=(conn1,), daemon=True).start()

        conn2, addr2 = server2.accept()
        print(f"Connected to client from {addr2}")
        threading.Thread(target=server2_handler, args=(conn2,), daemon=True).start()

        conn1.settimeout(1)  # Set a timeout for conn1
        conn2.settimeout(1)  # Set a timeout for conn2
```

The start_servers function creates two server sockets, server1 and server2, binds them to the specified host address and port numbers, and starts listening for incoming client connections. Inside an infinite loop, it accepts client connections on both servers, prints the client address, and starts a new thread to handle each connection using the respective handler functions (server1_handler and server2_handler). It sets a timeout of 1 second for each connection. The servers continue running indefinitely, accepting new connections and handling them in separate threads.

At least I call start_servers() function.

```
start_servers()
```

Client.py:

The code snippet defines the server configurations. It sets the host address and port numbers for two servers, Server 1 and Server 2, using concise variable assignments. The SERVER1_HOST and SERVER2_HOST variables are set to 'localhost', representing the local machine. The SERVER1_PORT and SERVER2_PORT variables are set to 5000 and 5001, respectively, specifying the port numbers for the servers.

```
import socket

# Server configuration
SERVER1_HOST = 'localhost'
SERVER1_PORT = 5000
SERVER2_HOST = 'localhost'
SERVER2_PORT = 5001
```

```
def check_song_existence(song_name):
    check_server = input("to server1 press '1', to server2 press '2'")
    # Server 1
    if check_server == "1":
        server1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server1.connect((SERVER1_HOST, SERVER1_PORT))
        print("Sending song name to server 1...")
        server1.send(song_name.encode())
        print("Receiving response from server 1...")
        response1 = server1.recv(1024).decode()
        print("Response received from server 1:", response1)
        server1.close()
        return response1

    # Server 2
    elif check_server == "2":
        server2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server2.connect((SERVER2_HOST, SERVER2_PORT))
        print("Sending song name to server 2...")
        server2.send(song_name.encode())
        print("Receiving response from server 2...")
        response2 = server2.recv(1024).decode()
        print("Response received from server 2:", response2)
        server2.close()
        return response2

    else:
        print("wrong Key!")
        return 0
```

The check_song_existence function takes a song_name parameter and checks the existence of the song on the specified server. It prompts the user to input a server choice, either '1' for Server 1 or '2' for Server 2.

If the user inputs '1', it creates a new socket object, server1, and connects it to Server 1 using the specified host and port (SERVER1_HOST and SERVER1_PORT). It prints a message indicating that it is sending the song name to Server 1, then sends the encoded song_name to Server 1 using the send method. It prints a message indicating that it is receiving a response from Server 1, then receives the response from Server 1 using the recv method. It decodes the received data from bytes to a string. It prints the received response from Server 1 and closes the connection to Server 1 using the close method. Finally, it returns the received response.

If the user inputs '2', it follows the same process as steps 2-5, but for Server 2 using the specified host and port (SERVER2_HOST and SERVER2_PORT). If the user inputs neither '1' nor '2', it prints a message indicating an incorrect input and returns 0.

In summary, the check_song_existence function prompts the user to choose a server, creates a socket connection to the chosen server, sends the song name to the server, receives and decodes the response, and returns the response. If an incorrect server choice is made, it returns 0.

```
def download_song(server_host, server_port, song_name):
    download_input = input("To download press '1' \t To skip press any key:")
    if download_input == "1":
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.connect((server_host, server_port))
        server.send(song_name.encode())
        response = server.recv(1024)
        print(f"{song_name} is downloaded.")
        server.close()
    else:
        print("Download Skipped.")
```

The download_song function takes three parameters: server_host (the host address of the server), server_port (the port number of the server), and song_name (the name of the song to download). It prompts the user to input a choice: press '1' to download the song or any other key to skip the download.

If the user inputs '1', it creates a new socket object, server, and establishes a connection to the specified server using the connect method. It sends the encoded song_name to the server using the send method. It receives the response from the server, which represents the downloaded song data, using the recv method. The response is stored in the response variable. It prints a message indicating that the song is downloaded. It closes the connection to the server using the close method.

If the user input is anything other than '1', it prints a message indicating that the download is skipped.

In summary, the download_song function prompts the user for a download choice, establishes a socket connection to the server, sends the song name, receives the song data if requested, and prints a download confirmation message. If the download is skipped, it prints a corresponding message.

```
song_name = input("Enter the name of the song: ")

# Check song existence on both servers
response1 = check_song_existence(song_name)

if response1 == 'EXIST':
    # Server 1 has the song
    download_song(SERVER1_HOST, SERVER1_PORT, song_name)
else:
    print("The song does not exist on any server.")
```

It prompts the user to enter the name of the song. It checks the existence of the song on both servers using the check_song_existence function and assigns the response to the response1 variable.

If the response indicates that the song exists (response1 is 'EXIST'), it calls the download_song function, passing the server host and port for Server 1 (SERVER1_HOST and SERVER1_PORT), along with the song name.

If the response indicates that the song does not exist on any server, it prints a message stating that the song does not exist.

```
while True:
    song_name = input("Enter the name of the song (or 'q' to quit): ")
    if song_name == 'q':
        break

    # Check song existence on both servers
    response1 = check_song_existence(song_name)

    if response1 == 'EXIST':
        # Server 1 has the song
        download_song(SERVER1_HOST, SERVER1_PORT, song_name)
    else:
        print("The song does not exist on any server.")
```

It creates an infinite loop that continues until the user enters 'q' to quit. Inside the loop, it prompts the user to enter the name of a song or 'q' to quit.

If the user enters 'q', the loop breaks and the program terminates. Otherwise, it checks the existence of the entered song on both servers using the check_song_existence function and assigns the response to the response1 variable.

If the response indicates that the song exists (response1 is 'EXIST'), it calls the download_song function, passing the server host and port for Server 1 (SERVER1_HOST and SERVER1_PORT), along with the song name.

If the response indicates that the song does not exist on any server, it prints a message stating that the song does not exist.