Project Title:

File Transfer System Using Socket Programming

Submitted by:

Ebaa Haq	**2021-CE-22**
Maham Nadeem	**2021-CE-10**
Faiza Riaz	**2021-CE-20**
Sana Israr	**2021-CE-55**

Submitted to:

Ma'am Darakhshan Abdul Ghaffar

Course:

CMPE-333L Computer Networks

Semester:

Spring 2024

Date of Submission:

April 7, 2024

Department of Computer Engineering

University of Engineering and Technology, Lahore

Table of Contents

Abstract 3

1. Introduction 3

2. Problem statement 3

3. Scope 3

4. Explanation 4

5. Flowchart 4

6. Working 4

7. Code 5

8. Output 10

9. Applications in Computer Network 14

References 15

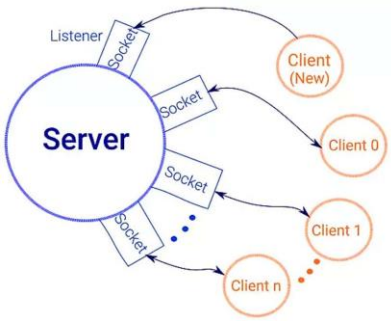
Abstract

This project proposes the development of a File Transfer System using Socket Programming to enable efficient exchange of files between a server and client. The system employs robust client-server architecture, implementing socket programming for seamless communication. The objectives include designing reliable file transfer mechanisms, ensuring data integrity, optimizing network bandwidth utilization, and providing user-friendly interfaces. Expected outcomes include a fully functional system with enhanced network efficiency, improved reliability, and versatile applications in enterprise file sharing, media streaming, backup solutions, content distribution, and remote access.

1. Introduction

In today's digital era, the need for seamless file transfer mechanisms is paramount. Traditional file transfer methods may lack efficiency and security, especially in network environments. To address these challenges, our project focuses on leveraging socket programming techniques to develop a robust system for transferring files between a server and client.

Socket programming allows for the establishment of communication channels between devices over a network. By harnessing this technology, we aim to create a file transfer system that ensures reliable and efficient file exchange. The system will enable users to seamlessly transfer files of varying sizes and types, enhancing collaboration and productivity in networked environments.



2. Problem statement

Current methods of transferring files often have problems like slow speeds, security risks, and trouble with different file types. These issues can slow down work and put data at risk. Our project is all about fixing these problems by making a special File Transfer System using Socket Programming.

We're using socket programming to make this system work better than the usual methods. With our system, transferring files is smooth and easy. It will help people and companies work together better by making it simpler to share files and get things done faster.

3. Scope

The objective of this project is to create an audio streaming application versatile enough to cater to a range of needs, including distance learning, training sessions, corporate events, academic

conferences, and entertainment purposes. Users will have the ability to fine-tune quality settings to suit their requirements, ensuring an optimal listening experience. The application will facilitate seamless real-time streaming of music, lectures, presentations, and live performances. Its broad scope allows for adaptation to meet the unique demands of various industries and sectors, offering a dependable, affordable, and user-friendly solution for live audio streaming needs.

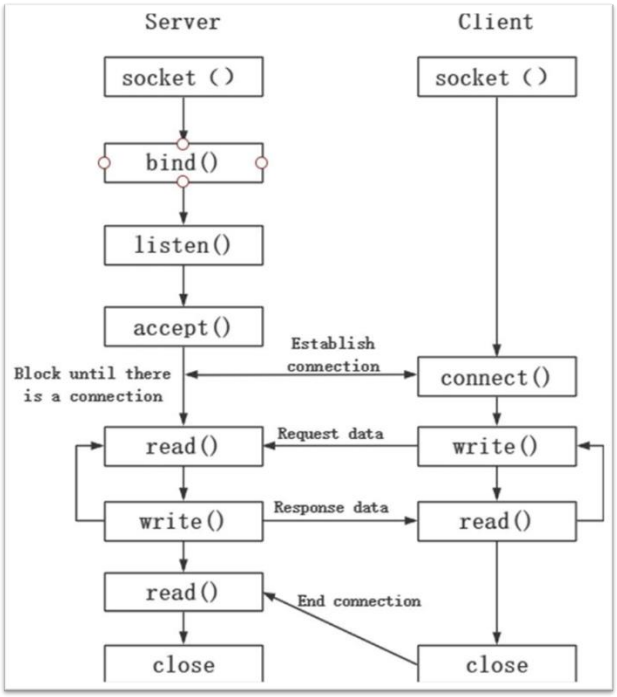
4. Explanation

Our project involves creating an audio streaming application using Python's socket programming. This means we'll establish a server-client architecture where the server hosts send the audio stream and clients connect to receive the audio stream. The server will break down the audio data into smaller packets and transmit them through socket connections to the clients, who will reassemble them for playback. That audio will also convert into text using speech to txt recognition then save it in a .txt file

While socket programming enables efficient real-time communication, it also poses challenges. High-quality audio streaming demands significant bandwidth and network latency can affect the speed and reliability of data transmission. Additionally, ensuring data security during transmission is crucial to prevent unauthorized access or tampering.

5. Flowchart

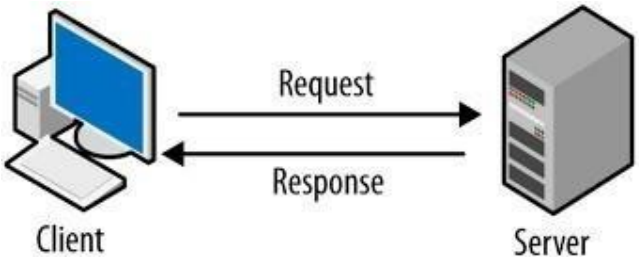
Here is a flowchart of transferring data using Socket Programming



6. Working

The working methodology involves several key steps:

1. **Establishing a Connection**: Develop server and client components to establish a connection using socket programming.



2. **Implementing File Request Handling**: Enable clients to request specific files from the server.
3. **Developing File Transfer Mechanisms:** Implement mechanisms for seamless and efficient file transfer between server and client.
4. **Ensuring Error Handling**: Incorporate robust error handling mechanisms to ensure data integrity and reliability during file transfer.
5. **Creating User Interface**: Develop a user-friendly interface to enhance user experience and facilitate file transfer operations.

Through these steps, we aim to create a functional and user-friendly File Transfer System that meets the needs of users in networked environments.

7. Code

Here is the server code:

```
# Complex Engineering Activity, File Transfer System using SOCKET programming

#    Group Members
# Ebaa Haq (2021-CE-22)
# Maham Nadeem (2021-CE-10)
# Faiza Riaz (2021_CE-20)
# Sana Israr (2021-CE-55)

# This is Server code to Send audio frames over TCP

import socket
import threading
import tkinter as tk
import wave
import pyaudio
import pickle
import struct
import tkinter.messagebox as messagebox
import os

class ServerGUI:
    def __init__(self):
        self.host_name = socket.gethostname()
        self.host_ip = '192.168.10.7'  # Set to appropriate IP address
```

```
        self.port = 9611
        self.server_socket = None
        self.client_sockets = []
        self.lock = threading.Lock()  # Adding a lock for thread-safe
operations on the client_sockets list

        self.root = tk.Tk()
        self.root.title("Server")
        self.status_label = tk.Label(self.root, text="Not Connected")
        self.status_label.pack()

        self.client_count_label = tk.Label(self.root, text="Clients
Connected: 0")
        self.client_count_label.pack()

        self.connect_button = tk.Button(self.root, text="Start Server",
command=self.start_server)
        self.connect_button.pack()

        self.send_audio_button = tk.Button(self.root, text="Send Audio File",
command=self.send_audio_file, state=tk.DISABLED)
        self.send_audio_button.pack()

        self.disconnect_button = tk.Button(self.root, text="Disconnect All",
command=self.disconnect_all, state=tk.DISABLED)
        self.disconnect_button.pack()

    def start_server(self):
        self.server_socket = socket.socket()
        self.server_socket.bind((self.host_ip, self.port))
        self.server_socket.listen(5)
        self.status_label.config(text="Server Listening at " + self.host_ip)
        self.connect_button.config(state=tk.DISABLED)
        self.send_audio_button.config(state=tk.NORMAL)
        self.disconnect_button.config(state=tk.NORMAL)
        threading.Thread(target=self.accept_connections, daemon=True).start()

    def accept_connections(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            with self.lock:
                self.client_sockets.append(client_socket)
                self.update_client_count()  # Update GUI every time a new
client connects
            threading.Thread(target=self.handle_client,
args=(client_socket,), daemon=True).start()
            messagebox.showinfo("Client Connected", f"Client connected from
{addr}")

    def handle_client(self, client_socket):
        try:
            while True:
                data = client_socket.recv(1024)
                if not data:
                    break  # Client has disconnected
        except Exception as e:
            print(f"Error with client {client_socket}: {e}")
```

```
        finally:
            with self.lock:
                if client_socket in self.client_sockets:
                    self.client_sockets.remove(client_socket)
                    self.update_client_count()  # Update GUI when a client
disconnects
            client_socket.close()

    def update_client_count(self):
        # Ensure this method is called within a locked context to maintain
thread safety
        self.client_count_label.config(text=f"Clients Connected:
{len(self.client_sockets)}")

    def send_audio_file(self):
        file_name = "Ebaa1.wav"
        if os.path.exists(file_name):
            with open(file_name, 'rb') as f:
                audio_data = f.read()
                data_packet = pickle.dumps(audio_data)
                message = struct.pack("Q", len(data_packet)) + data_packet
                with self.lock:
                    for client_socket in self.client_sockets:
                        client_socket.sendall(message)
            messagebox.showinfo("File Sent", f"The file '{file_name}' has
been sent to clients.")
        else:
            messagebox.showerror("File Not Found", f"The file '{file_name}'
does not exist.")

    def disconnect_all(self):
        with self.lock:
            while self.client_sockets:
                client = self.client_sockets.pop()
                client.close()
        self.server_socket.close()
        self.update_client_count()
        self.status_label.config(text="All clients disconnected")
        self.connect_button.config(state=tk.NORMAL)
        self.send_audio_button.config(state=tk.DISABLED)
        self.disconnect_button.config(state=tk.DISABLED)
        messagebox.showinfo("Disconnection", "All clients disconnected and
server stopped.")

if __name__ == "__main__":
    server_gui = ServerGUI()
    server_gui.root.mainloop()
```

Here is the client code:

```
# Complex Engineering Activity, File Transfer System using SOCKET programming

#    Group Members
# Ebaa Haq (2021-CE-22)
# Maham Nadeem (2021-CE-10)
```

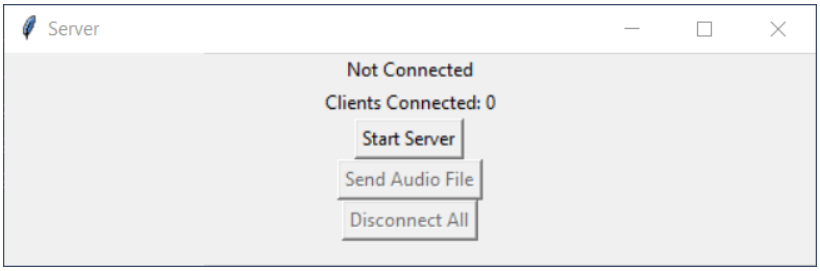
```
# Faiza Riaz (2021_CE-20)
# Sana Israr (2021-CE-55)

# This is client code to receive audio frames over TCP


import socket
import threading
import tkinter as tk
import wave
import pyaudio
import pickle
import struct
import os
import tkinter.messagebox as messagebox
import speech_recognition as sr


class ClientGUI:
    def __init__(self):
        self.host_name = socket.gethostname()
        self.host_ip = '192.168.10.7'  # Server IP address
        self.port = 9611
        self.client_socket = None

        self.root = tk.Tk()
        self.root.title("Client")
        self.status_label = tk.Label(self.root, text="Not Connected")
        self.status_label.pack()

        self.connect_button = tk.Button(self.root, text="Connect",
command=self.connect_to_server)
        self.connect_button.pack()

        self.stream_button = tk.Button(self.root, text="Start Streaming",
state=tk.DISABLED,
                                       command=self.start_streaming)
        self.stream_button.pack()

        self.disconnect_button = tk.Button(self.root, text="Disconnect",
state=tk.DISABLED, command=self.disconnect)
        self.disconnect_button.pack()

    def connect_to_server(self):
        self.client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        socket_address = (self.host_ip, self.port)
        self.client_socket.connect(socket_address)
        self.status_label.config(text="Connected to Server")
        self.connect_button.config(state=tk.DISABLED)
        self.stream_button.config(state=tk.NORMAL)
        self.disconnect_button.config(state=tk.NORMAL)
        threading.Thread(target=self.receive_audio_file).start()
        messagebox.showinfo("Connection", "Connected to Server")

    def receive_audio_file(self):
        try:
```

```
            payload_size = struct.calcsize("Q")
            data = b""
            while len(data) < payload_size:
                packet = self.client_socket.recv(4 * 1024)  # 4K
                if not packet:
                    return  # No more data
                data += packet

            packed_msg_size = data[:payload_size]
            data = data[payload_size:]
            msg_size = struct.unpack("Q", packed_msg_size)[0]

            while len(data) < msg_size:
                data += self.client_socket.recv(4 * 1024)

            audio_data = pickle.loads(data[:msg_size])
            with open("received_audio.wav", "wb") as f:
                f.write(audio_data)
            # File is received, enable streaming button
            self.stream_button.config(state=tk.NORMAL)

            # Convert audio to text
            recognizer = sr.Recognizer()
            with sr.AudioFile("received_audio.wav") as source:
                audio_data = recognizer.record(source)
                text = recognizer.recognize_google(audio_data)

            # Save text to a text file
            with open("received_file.txt", "w") as text_file:
                text_file.write(text)
            messagebox.showinfo("Conversion", "Audio file converted to text
and stored as received_file.txt")

        except Exception as e:
            messagebox.showerror("Error", f"Error receiving audio: {e}")

    def start_streaming(self):
        if os.path.exists("received_audio.wav"):
            threading.Thread(target=self.audio_stream).start()
            messagebox.showinfo("Streaming", "Received file is streaming")
        else:
            messagebox.showerror("File Not Found", "The received audio file
does not exist.")

    def audio_stream(self):
        try:
            CHUNK = 1024
            wf = wave.open("received_audio.wav", 'rb')
            p = pyaudio.PyAudio()
            stream =
p.open(format=p.get_format_from_width(wf.getsampwidth()),
                           channels=wf.getnchannels(),
                           rate=wf.getframerate(),
                           output=True,
                           frames_per_buffer=CHUNK)
            data = wf.readframes(CHUNK)
            while data:
```

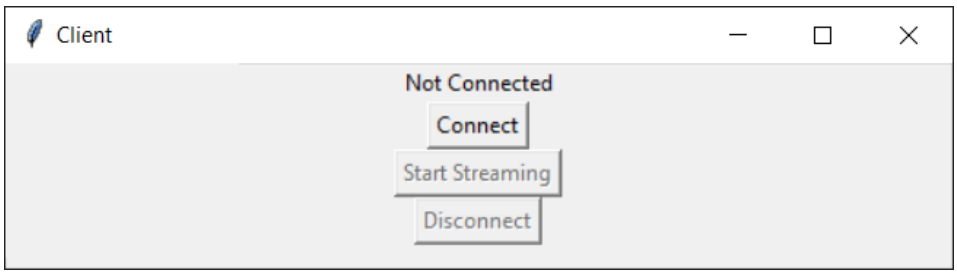
```
                stream.write(data)
                data = wf.readframes(CHUNK)
            stream.stop_stream()
            stream.close()
            p.terminate()
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {e}")

    def disconnect(self):
        if self.client_socket:
            self.client_socket.close()
            messagebox.showinfo("Disconnection", "Disconnected from Server")
        self.status_label.config(text="Disconnected")
        self.connect_button.config(state=tk.NORMAL)
        self.stream_button.config(state=tk.DISABLED)
        self.disconnect_button.config(state=tk.DISABLED)


if __name__ == "__main__":
    client_gui = ClientGUI()
    client_gui.root.mainloop()
```

8. Output

Before establishing the connection between the client and server, the server and client windows are:

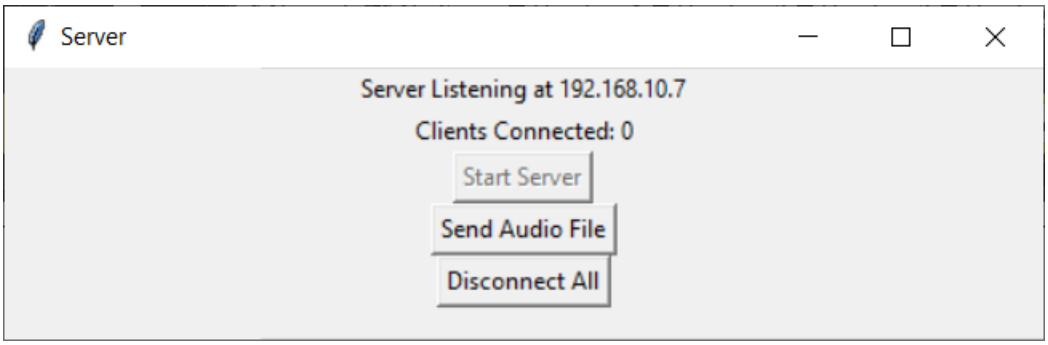


Server Window

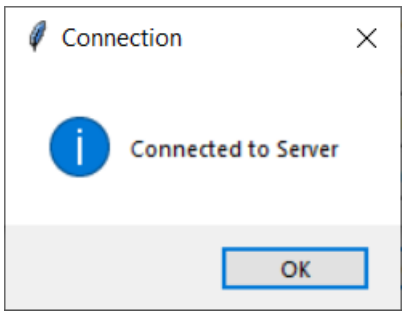


Client Window

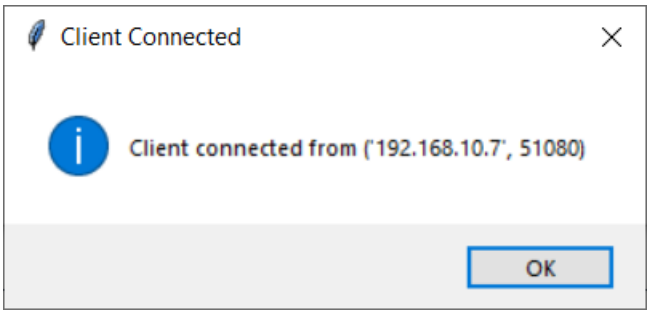
So, when Server click the **Start Server** button then server window will be as:



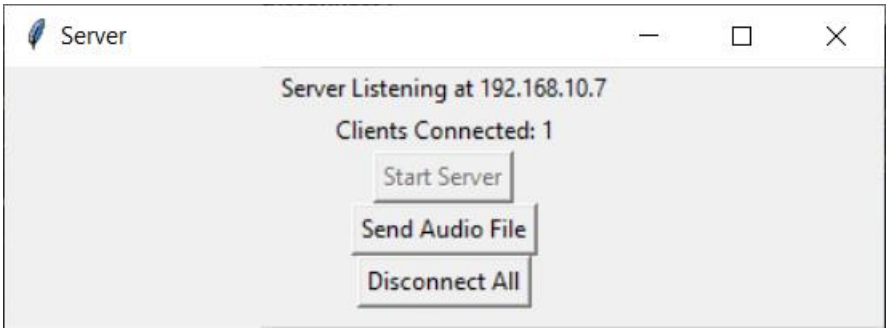
And when the Client click the **Connect** button then a **Connection** dialog box shows to client that is



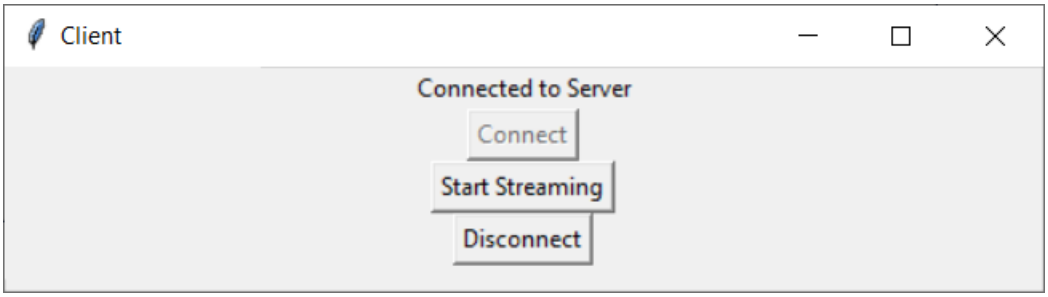
And a **Client Connected** dialog box shows to Server that is



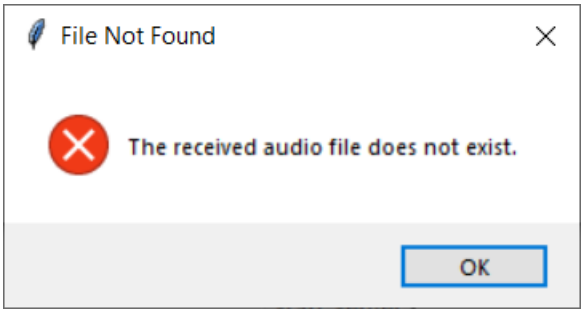
Then server window will be as



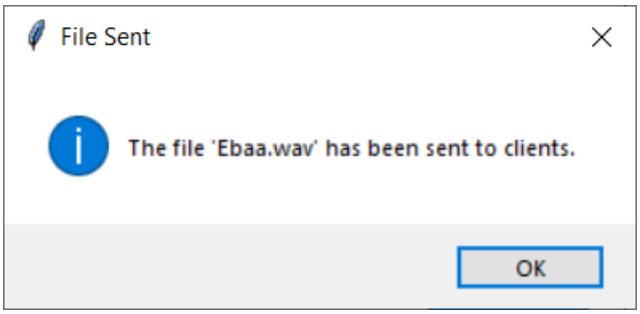
And the Client window will be as



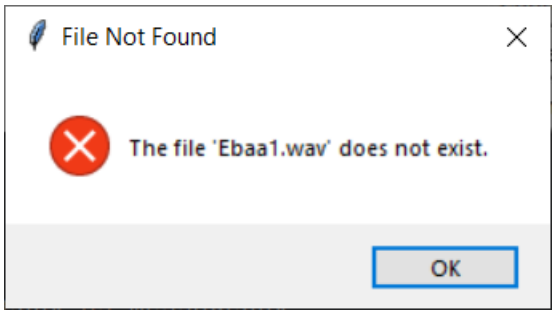
When Client clicks to the start streaming button without before the server sends the audio file then **File Not Found** dialog box appear that is



when the Server clicks the **Send Audio File** then the file sent to client and show **File Sent** dialog box that is

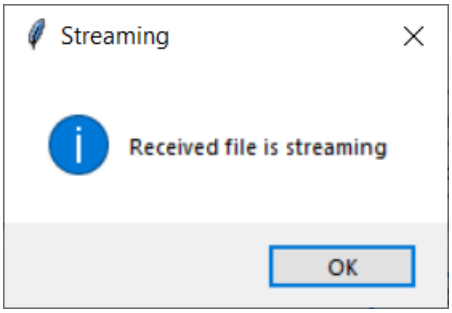


when the Server clicks the **Send Audio File** but the filename is incorrect or file does not exists then **File Not Found** dialog box will appear that is

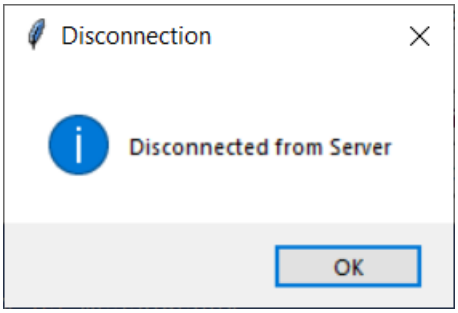


When that audio file receives at client side it will save as **received_audio.wav** then that audio file will convert into text and then save as **received_file.txt** and a **Conversion** dialog box will appear that is

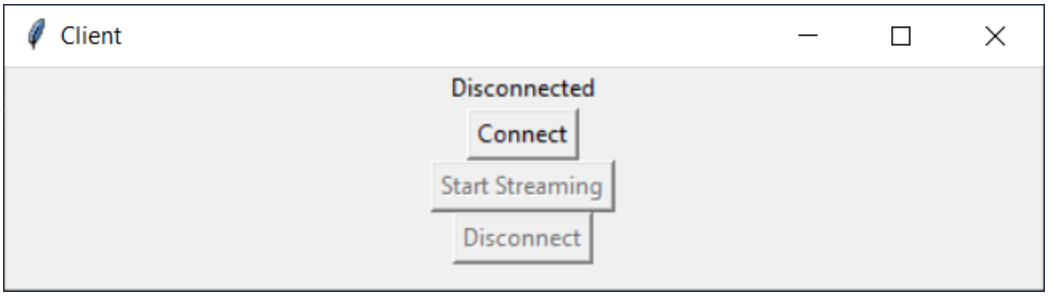
So, after receiving file when client clicks the **Start Streaming** button then **Streaming** dialog box will appear and audio file will be streamed.



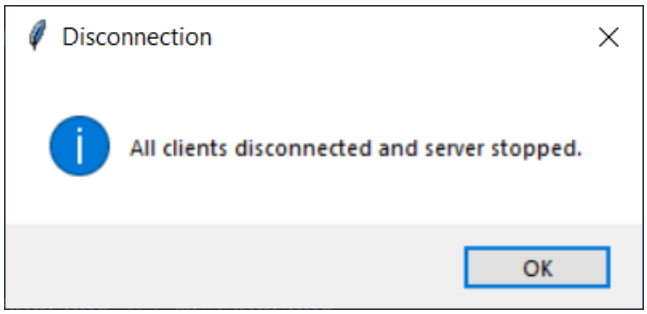
When the client clicks the Disconnect button then **Disconnection** dialog box will appear that is



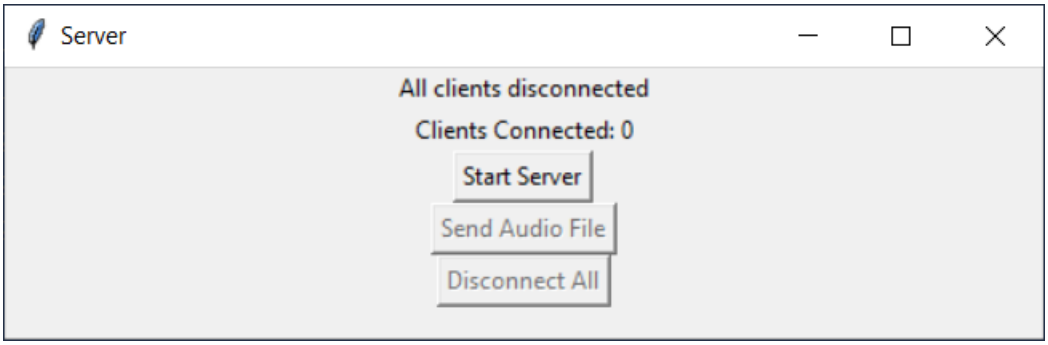
And a Client Window will be as



When the Server clicks the Disconnect button then **Disconnection** dialog box will appear that is



And the Server Window will be as:



9. Applications in Computer Network

The developed File Transfer System holds significant applications in computer networks, including:

- **Efficient File Sharing**: Facilitating efficient sharing of files among users in networked environments.

- **Remote File Access**: Enabling users to remotely access files stored on a server from any location.

- **Collaborative Work Environments**: Supporting collaborative work environments by facilitating the exchange of files among team members.

- **Data Backup and Recovery**: Offering a reliable mechanism for backing up critical data and restoring it in case of system failures or data loss incidents.

- **Remote System Administration**: Allowing system administrators to remotely manage and administer networked systems, including software updates, configuration changes, and troubleshooting tasks.

References

[1] https://codedamn.com/news/java/transferring-files-from-client-to-server-socket-in-java

[2] www.geeksforgeeks.org/file-transfer-using-tcp-socket-in-python/