Distributed Media Application

Group No : B28

Group Members:

Name	ID	Contribution
Parth Agarwal	202001098	Client Code
Jill Chaudhary	202001181	Server Code, Client Code
Harsh Shah	202001221	Client Code
Virat Chaudhari	202001240	Server Code
Zeel Bhanderi	202001412	Server Code

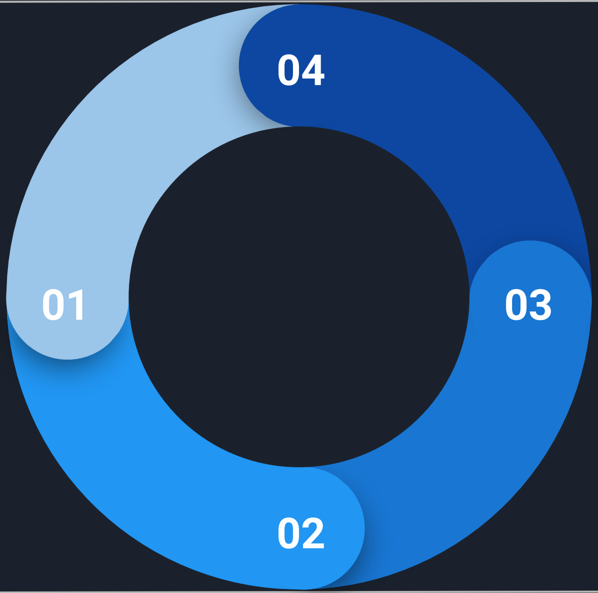
Problem Statement:

Implementing a ***client-server system*** for a multimedia application. ***One or more clients*** should connect with the server and request for an audio / video file (mp3/mp4). The server should ***receive and process message in an order,*** provide the file so the client can download.

Motivation Behind the Project:

- To provide a way for users to access multimedia files from a central location.
    - This would make it easier for users to download the files they need, and it would also free up space on their devices.
- To improve the performance of multimedia applications.
    - By having the server handle the processing of the files, the clients can focus on displaying the content.
- To make it easier to manage multimedia files.
    - The server can be used to store, organize, and update the files.

Technologies Used:



Python

Socket Programming

Multi - Threading

Queueing

Technologies Used:

- Python:

  Specific benefits of using Python for distributed media application development:

  1)Simplicity 2) Versatility 3) Library ecosystem

- Multi - Threading:

  The server code uses multi-threading to handle multiple client requests concurrently. Each client request is added to a queue, and a new thread is created to handle each request, allowing the server to handle multiple requests at the same time.

Technologies Used:

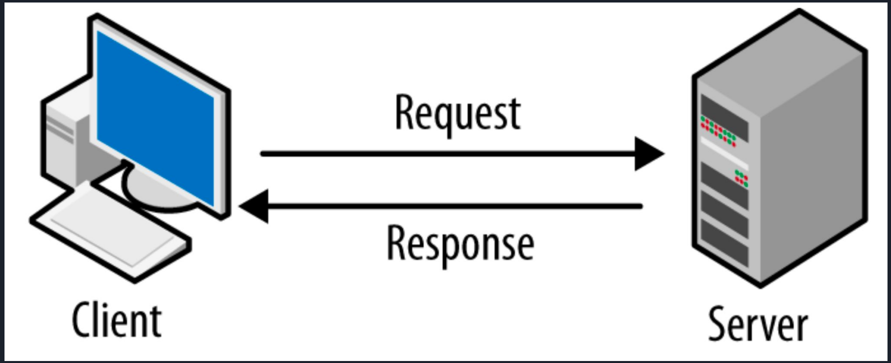
- ## Socket Programming:

  Our program demonstrates the use of Python's socket module to establish a connection between a client and server. This allows data to be transferred between the two over a network.

- ## Queueing:

  At the server-side, a queue is utilized to store incoming requests from clients. A thread is then initiated to process these requests in the order they were received from the queue.

System Architecture:



- **Client :** Represents individual users and runs a client-side application for sending and receiving messages.
- **Server :** Manages client interactions, maintains client connections, and provides files (audio, video, text)as an intermediary.

Communication between client and server

- The server application starts and waits for incoming client connections.
- One or more clients connect to the server and request a multimedia file.
- The server receives the request and places it in a queue.
- The server uses a separate thread or process to process requests from the queue in order, sending the requested files to the clients that made the requests.
- The clients receive the file(s) and download them to their local systems.
- The clients can then play the downloaded multimedia files using a suitable player application.

Key Takeaways:

1

How to handle multiple clients concurrently

2

How to establish connection using socket programming

3

Coordination and Collaborative Teamwork

Conclusion

- In conclusion, the distributed media application provides a reliable and efficient way for clients to access and download multimedia files from a centralized server.
- The use of Python programming language allowed us to create a scalable, flexible, and easy-to-use system that can handle multiple client requests simultaneously.
- By using techniques like multi-threading, socket programming, and queuing, we were able to build a robust system that can handle different types of multimedia files and ensure that the files are delivered in the correct order to the clients.

THANK YOU!