Algorithms and Operating System Final Project
Video Chat Application

Problem statement
Create a video chat app with multithreading

Introduction
The demand for video chat has increased a lot during the pandemic. From online classes to business meetings, everything is carried out online. So privacy and data security became a severe issue. In this project, we propose a video chat application that works over a local network without concerns about data theft.

Workflow

Week One - **Planning and the designing**

During the first week, we decided on our plan of action and tried out small experiments with the socket library in python.

Week Two - **Built one to one video chat**

Once we got an idea of how the socket library works, we moved to sending text messages from one client to another. Towards the end of the week, we upgraded this and made two-way video communication possible.

Week Three - **Multiple user video chat application with audio**

Now multiple users can connect to the host (server) and video chat with each other. Audio communication has also been implemented.

Experiments

- **Basic two-way chat (Without server)**

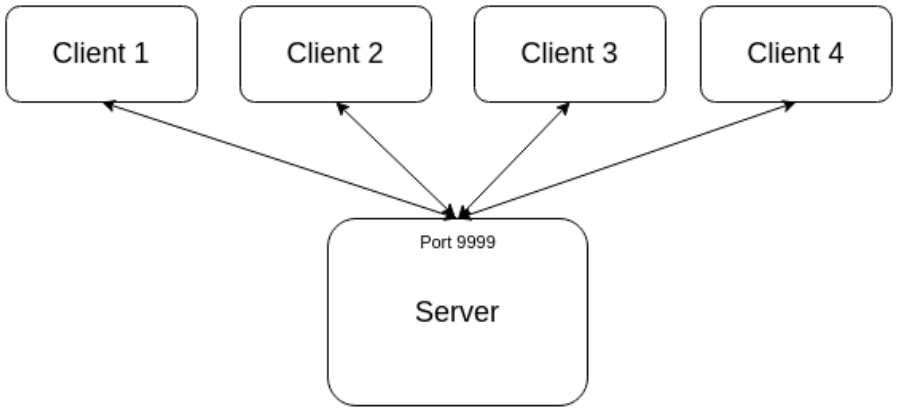
- Used socket library to create socket and bind to a particular pre-defined port (port 9999 by default) from both client sides
- For this project, we will be using TCP/IPv4 based connection as it is less susceptible to data loss
- It sends a connection request, once each client receives the IP and port info of the other client (given through terminal UI)

- The step above creates two separate data paths for sending and receiving

- There will be two separate threads for sending and receiving data. Once the connection is established these two threads are started
- The infinite loop inside the send function keeps on waiting for the client to enter a message. On receiving an input it is encoded and sent using the socket.send function
- The receive function is also waiting indefinitely inside a while loop. As soon as it receives data it is decoded and printed on the terminal

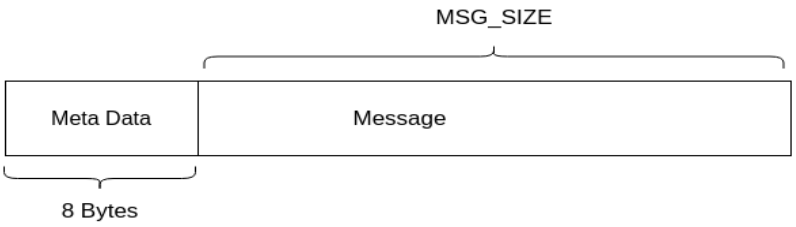
- **Server-based simple chat application (multiple users)**

- Even though the basic idea of sending and receiving messages is the same, we need a server to make multiple users connectivity possible
- The server is always running and waiting for the clients to join. The maximum number of clients allowed to connect at the same time is 5 (This is defined using the socket.listen method)
- We keep a list called ‘threads’ to keep track of all the threads running and clients associated with it. Another variable ‘ThreadCount’ is kept for keeping the count of threads running.
- Each time a client connects to the server a function is called on a new thread to handle both receiving and sending (Receive and send functions are not separate since we only have one channel to a particular client)

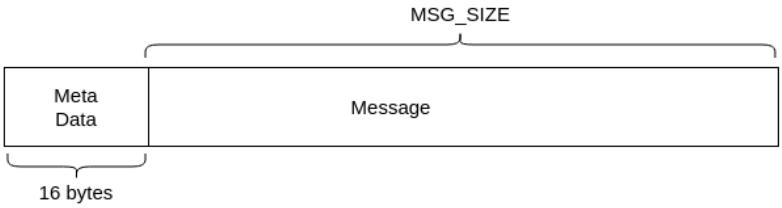


- Each time the server receives a message from one of the clients, it attaches the client number associated with it at the beginning of the message and sends it to all the clients connected at the moment (by looping through the ‘threads’ array).
- On the client-side, we have two threads for receiving and sending a message (just like the one-to-one scenario).

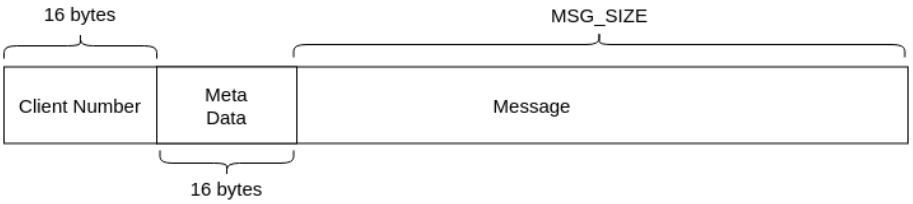
- **One to one video chat (without server)**
  - This was made as an extension to the one-to-one chat application mentioned above.
  - OpenCV library is used to read the live feed from the camera frame by frame
  - As soon as the connection (two-way) between both the clients is established, the while loop inside the send function keeps on reading frames. Each frame is serialized using the pickle library.
  - Since the receiving end has no idea about the size of the image it is about to receive, we attach that information in front of the message as a number stored in 8 bytes (using struct library).
  - In the receiving end, we read the first 8 bytes and unpack it to find out the size of the image sent



- **Server-based video chat application (Multiple users)**
  - Just like the multi-user chat application, we have a single function to handle the receiving and sending in the server.
  - This time we used the cv2.imencode method for encoding the image in jpg format. And then it’s converted to a string and then binary encoded using str.encode method. The size of this image is attached in the front of the message in a 16 bytes space.



- A separate receive function is written to receive a specified no of bytes of data. This function is used to read the metadata and image data separately.
- The image received at the server is sent back to all the clients. But an additional data about the user is added in the front.

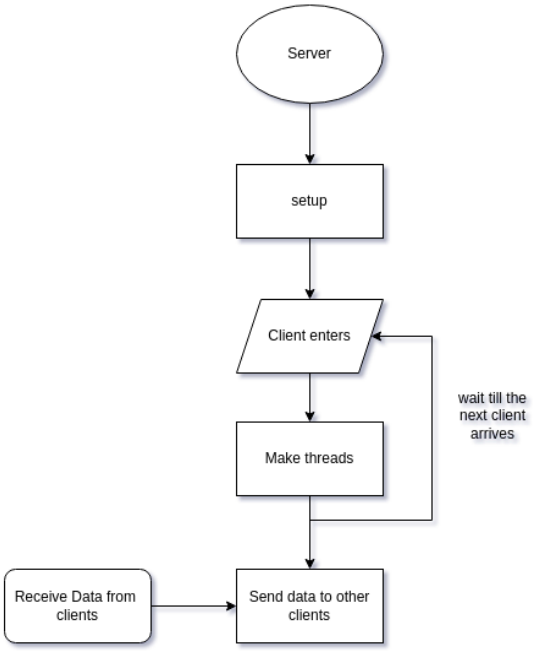


- **Video chat application for multiple users with audio (Final version)**

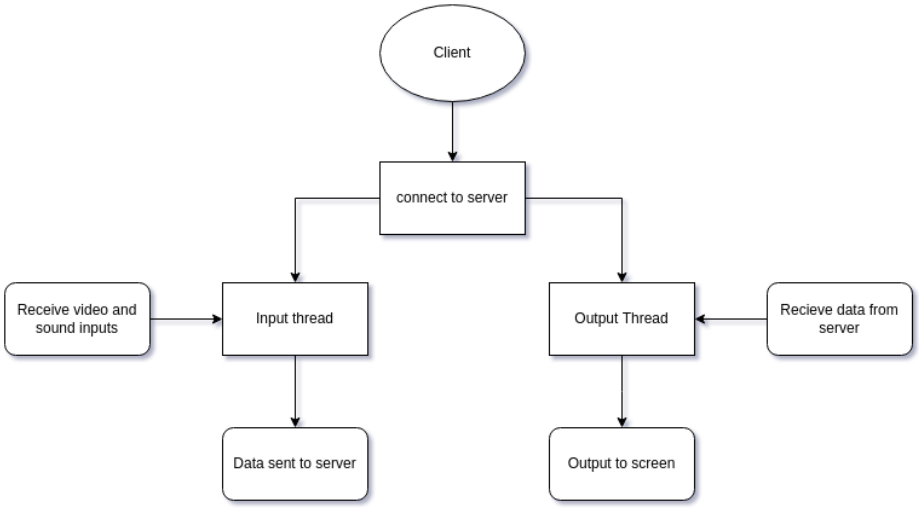
- On top of the last version, we added audio streaming functionality as well
- Used pyShine library for recording and playing audio
- Send and receive function for audio is similar to that of the video, but on a different port (8003 by default)

Overall Flow Chart

For server :



For client :



Problems faced

- Even though this program is written to support multiple users, the CPU usage of the machine where the server is hosted will increase as more clients connect.
- To stream audio, we tried using the pyAudio library but had to drop the plan midway because the library was buggy. Even the alternative that we used (pyShine) is a bit laggy in terms of audio reception.