CS3530: Computer Networks - I
Professor : Kotaro Kataoka
Department of Computer Science
Indian Institute of Technology Hyderabad

MULTI MEDIA GROUP CHAT

Sri Hari Malla - CS19BTECH11039

Veeramalli Sathwik - CS19BTECH11022

Nalari Sushma - CS19BTECH11006

Sri Harsha Vardhan Reddy - CS19BTECH11023

Pochetti Rashmitha - CS19BTECH11019 K Sri Teja - CS19BTECH11002

December 10, 2021

Contents

1 THE SOFTWARE OVERVIEW:

1.1 Socket:

A socket is one endpoint of a two-way communication link between two programs running on the network. So we use socket properties to achieve connection between server and client.

1.2 Server:

Server is now one endpoint of communication which has a socket. So the communication doesn't go beyond it. It receives the message and respond accordingly. Server creates a socket and attaches it to Ip address and port number to establish a server side connection. Listen method on server starts waiting for client.

1.3 Client:

Client is now the other endpoint of communication. Many clients can connect to server thus establishing one to many connection from server to clients. A client needs to connect to server using the IP and port number from which server is connected to the network. Look at system design to find the flow chart corresponding to event sequence.

1.4 Protocol:TCP

We used TCP to enable client and server to exchange messages. We used TCP because it guarantees proper file transfer by establishing a proper connection between client and server unlike UDP. As our project is Multi-media group chat, we don't want to loose data over the network when reaching from client to server or server to client. So we used TCP.

1.5 Baseline Implementation:

The baseline implementation contains server-client connection. It also allows a normal text chat within a group. A server is present and can accept any number of connections from clients and broadcasts messages received from one client to all other clients.

1.6 Enhanced Features:

To the existing implementation, the below features are added.

- Threading is used in the code to efficiently establish a proper connection between client and server and to communicate properly.
- File transfer(all formats such as `.txt,.pdf,.zip`) support is added properly.

- Basic Unix Command support (such as `ls, date, echo, cd`) is supported in the project.
- Bank-Authentication support is added. Server creates a qr code and sends it to client. To mimic real life world, the qr code will be deleted after 5 seconds from when it received.
- Some other features are added like to get people who are currently online and are present in the server and to properly exit from the connection.
- The features can be accessed using the protocols given below.

1.7 send/recv:

The commands send/recv are used on the socket to properly send and receive the bytes. No other wrapper/api is used in the program to send and receive the messages.

1.8 Rules/protocols:

Now that connections are established, now we implement certain protocols to establish a proper response coming out from a server after a particular request from client.

- `!req <filename>` is one such a protocol, in which client requests the file with name `<filename>` from server.
- `!send <filename>` is another protocol that sends an existing file from client with name `<filename>` to server.
- `!cmd <command>` is a protocol that client asks the server to run the particular command in server and send out the response.
- The message `!get QRcode` from client parses in the server and generates a unique QRcode and sends it to the client. However we will delete the qrcode within 5 seconds.
- The message `!online` gives the names of clients who are currently online.
- The messages which don't contain the above protocols are processed as normal messages and they are broadcasted.

1.9 File transfer:

General send and receive will throw errors or might encounter packet loss for large files and with different formats. To transfer files properly from server to client and viceversa, the files are read in byte format of small chunks and will be sent/received accordingly. The basic `send,recv` functions are used to achiev this feature.

1.10 Unix Command Support:

The unix command support is added to the baseline implementation by using python subprocess module. The basic commands like `ls,date,echo` works and the commands which contains `sudo,rm` are restricted.

1.11 Libraries:

We almost used basic networking libraries of Python to complete the project. But with adding features, there existed a need to import two unorthodox libraries namely `tqdm` and `pyqrcode`.

1.12 Difficulties faced/Error Handling:

We faced some difficulties through the project phase like the problems in file transfer and in executing commands. However we used various online references to clear our doubt and to find a solution. All the references are at another section in this report. We used `try,except` to handle errors like file not present or similar type of errors.

2 System Design:

2.1 Base Line Implementation:

- It contains normal chat features where a server is present and group of clients connect to it. For this the server must first create a socket and establish an end point and must bind with desired IP and port number. Then the listen method on server enables the server and is now ready to listen from the client.
- Now client need to connect with server to establish a two way communication. The client should create a socket and then connect to the socket created by server using connect method on the client created.
- `send,recv` are used on the server and client and to transfer messages as stated in the assignment statement.

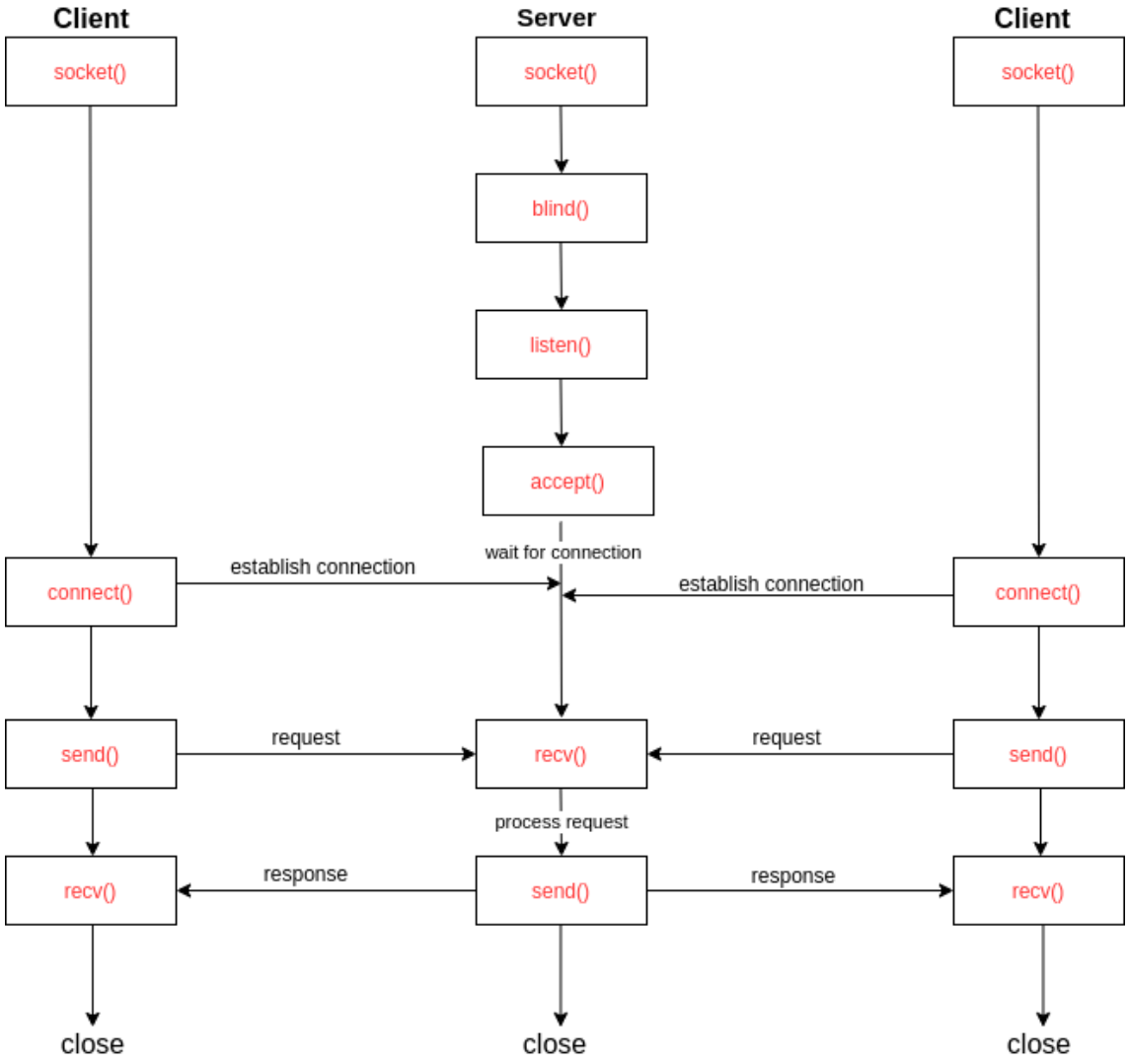

Figure 1: Control Flow

2.2 Threading:

When multiple clients are present in the network, there might exist a situation where multiple clients messaging, requesting or sending a file at the same time. This causes a havoc in the normal system. To handle this we used threading.

2.2.1 Server:

A server must connect to many clients. So we used threading on a server to handle multiple clients efficiently. Each thread is created when a new connection is established and it each thread takes care of each client which server is dealing with.

2.2.2 Client:

The client on the other hand must accept the response from the server and should be able to send message to the server properly at the same time. With Procedural functions, we will not be able to do this. For this we used threads such that each client program runs a separate thread to deal with writing to server and receiving from server.

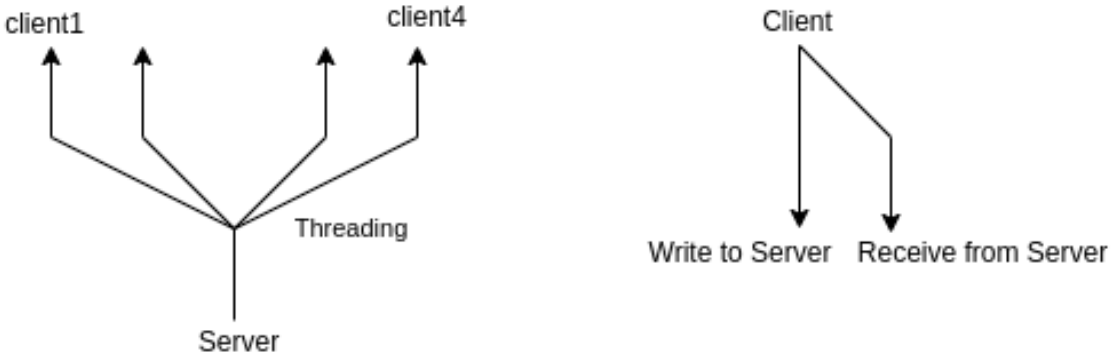

Figure 2: Threading

2.3 Enhanced Features:

All the features are the addition to baseline implementation. No complex library or attributes are used, used the basic components to make the features.

2.3.1 File transferring:

To transfer files we divided the total file into smaller chunks and were sent one by one. We used the standard `send,recv` to achieve this too.

2.3.2 Unix command support:

To add this support, subprocess module in python is used.

3 References:

- **send,recv:** send,recv in socket programming python
- **Threading:** Threading in Python
- **Shell Commands in python:** Running Unix commands from python
- **pyqrcode:** pyqrcode module in python

- **tqdm:** Progress bars in python
- **pkg_resources:** pkg_resources module in python

4 STARTING SERVER AND CLIENT

4.1 server

Using the command **"python3 server.py"**, the server starts running with IP address **192.168.0.18** and **PORT 12002**. It displays "Starting the server" and "Server listening at 192.168.0.18" which shows that the server is open and clients can connect to the server.

```
sri@golden:~/Documents/acads/sem5/cn1/final_files$ python3 server.py
**Starting** Starting the server
Server listening at 192.168.0.18
```

Figure 3: starting the server

If a client tries to connect to the server, "New connection request incoming!", the name of the client along with their (IP Address, PORT number) is displayed. The server maintains the count of the active client connections.

```
sri@golden:~/Documents/acads/sem5/cn1/final_files$ python3 server.py
**Starting** Starting the server
Server listening at 192.168.0.18
New connection request incoming!
**Connecting** Client connecting with name : sathwik
[New connection] Address with ('192.168.0.10', 60224) connected!
Active connections at 2021-12-10 17:45:46.844921 :  1
New connection request incoming!
**Connecting** Client connecting with name : sushma
[New connection] Address with ('192.168.0.17', 60376) connected!
Active connections at 2021-12-10 17:46:04.968912 :  2
```

Figure 4: server dispalys about number of clients and information

4.2 client

Using the command **"python3 client.py"**, the client is prompted to enter the HOST and PORT, along with their name. On choosing the server's HOST and PORT, the client gets connected to the server.

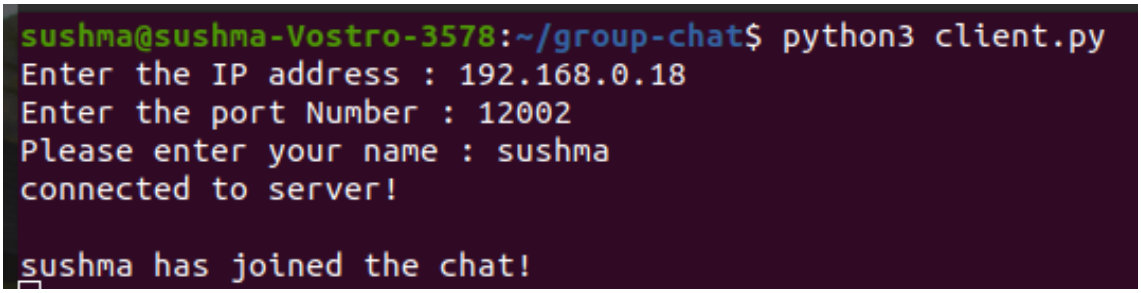

Figure 5: client joins the server

5 ONLINE GROUP CHAT

5.1 Server

As the clients keep joining and connecting to the server, the server keeps updating the count of active connections. Each time a client joins or exits from the group chat, "client_name" connecting and leaving is displayed.

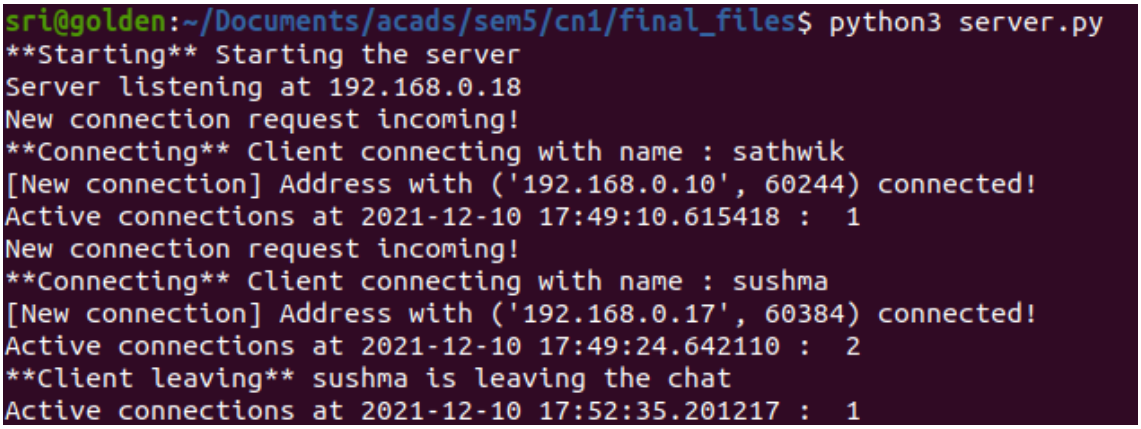

Figure 6: count of clients

5.2 Client

In the group chat, the client can chat with other clients by just texting the message and clicking enter. The chat supports the written text along with emojis.

A client can leave the chat by choosing the command **"!exit".**

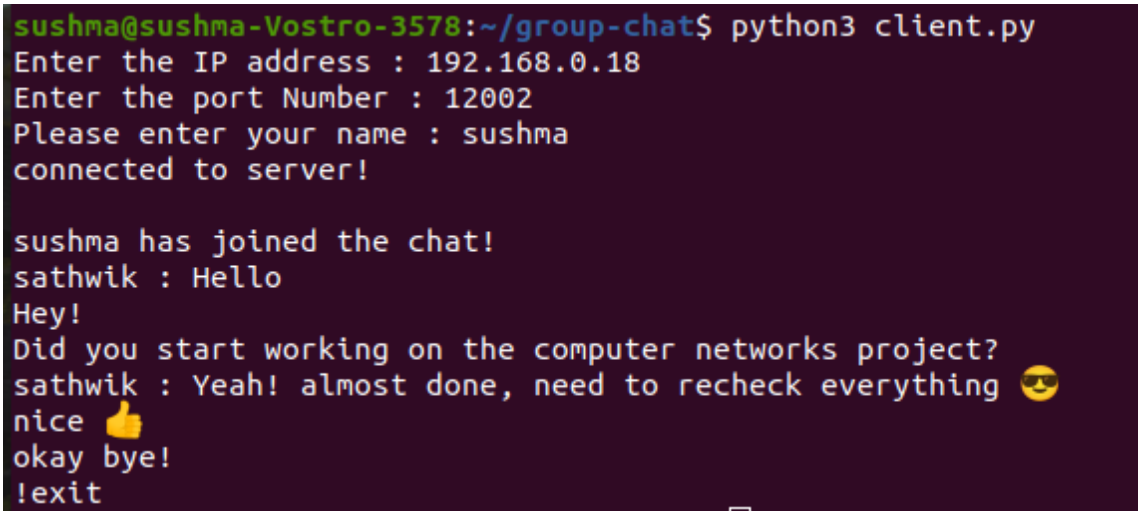

Figure 7: chat box

6 File Transfer

- **!send "filename"** command is used to send a file from the client to the server.
- **!req "filename"** command is used to request a file from the server to the client.
- It supports all file extensions like .txt, .pdf, .mp3, .mp4, .jpeg, .png, .py....

6.1 Server

When a client tries to send a file to the server, "File incoming" and "File received from client" are printed. The received file is saved as **"FILENAME_received_to_server"** in the server's directory.

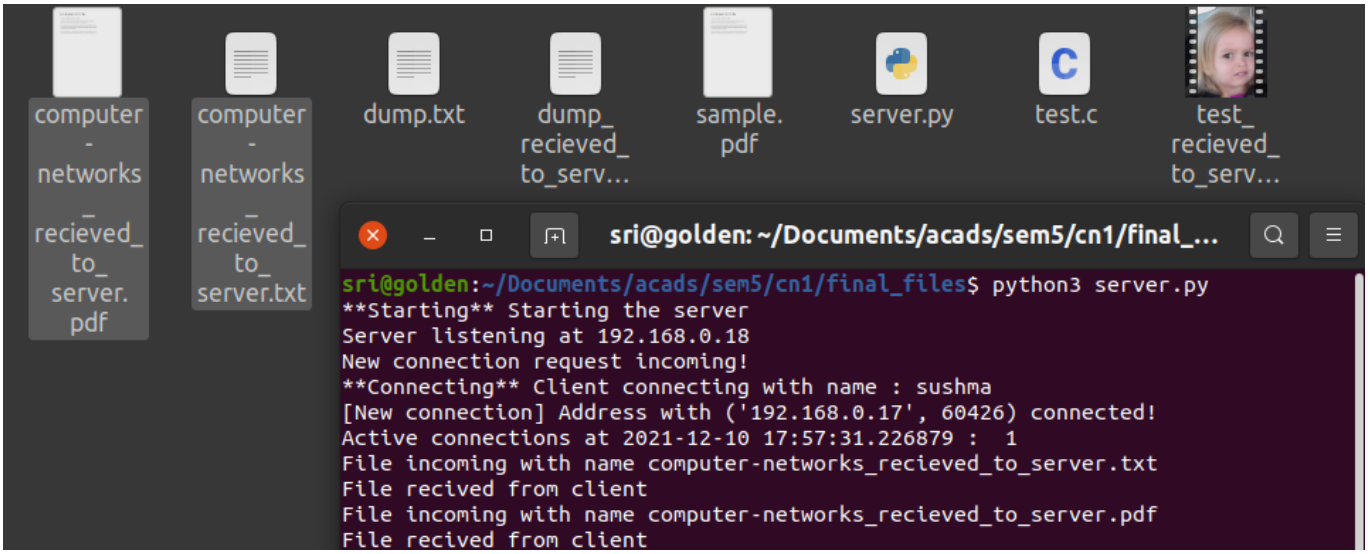

Figure 8: files received by server

6.2 Client

A file can be sent from the client to the server using the command **"!send filename".** The bytes transferred is kept updated in the progress bar until it reaches the total file_size(showing the complete file transfer).

Figure 9: files sent by client to server

7 Request File

7.1 Server

When a client tries to request a file from the server, the server gets the message "Client X requested file with name: Y". The bytes transferred is kept updated in the progress bar until it reaches the total file_size(showing the complete file transfer).

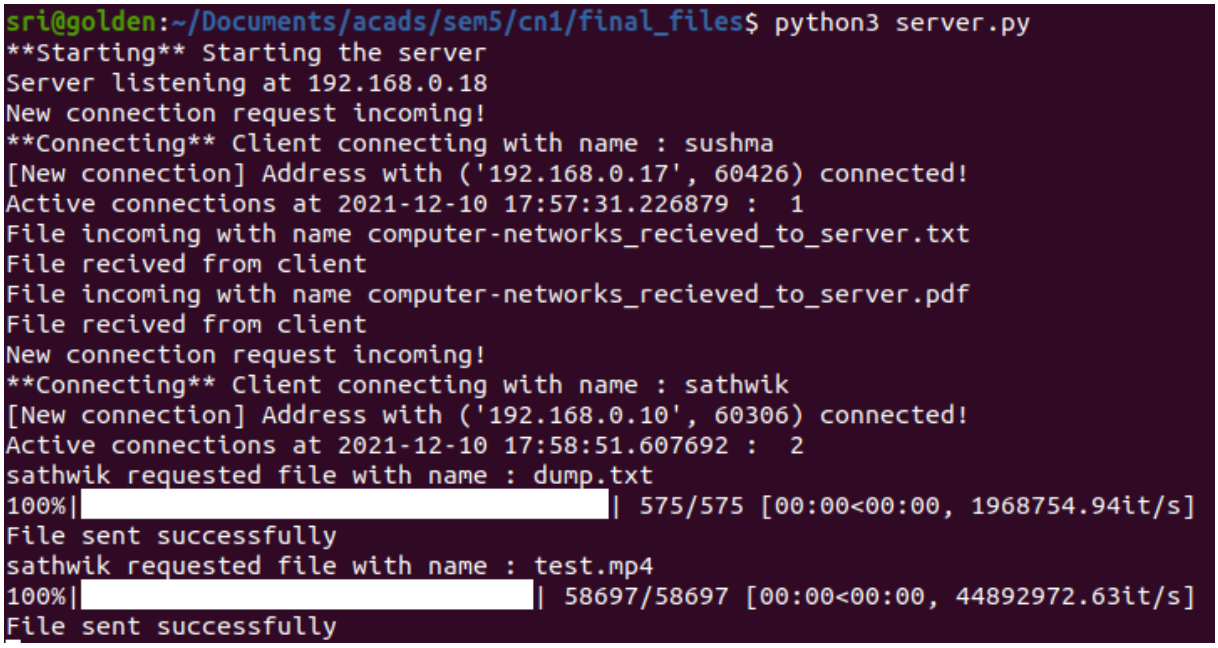

Figure 10: files sent by server to client

7.2 Client

To request a file from the server, one must know the filename. This can be obtained by using the linus command **'ls'**(Linux command support is explained in the next sections).

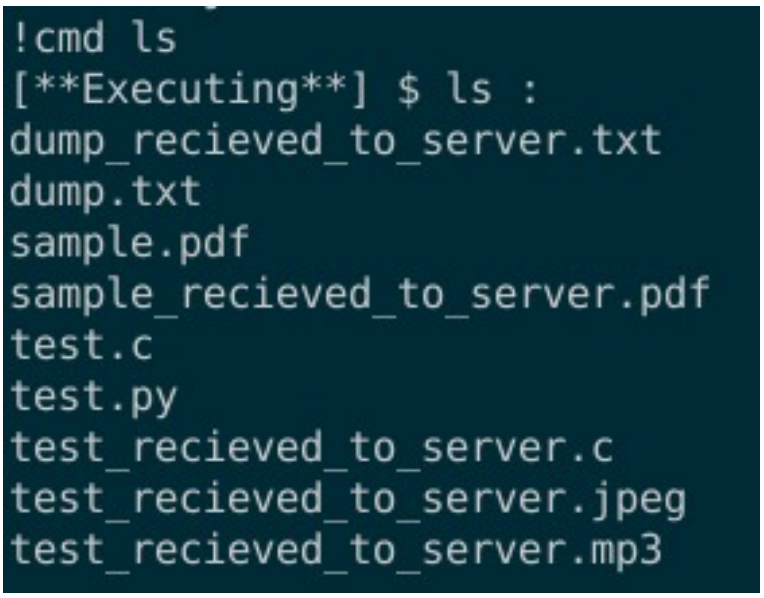

Figure 11: listing the files present in server

A selected file can be received from the server to the client using the command **"!req filename".** When a client tries to request a file from the server, the message "File received successfully" is displayed. The received file is saved as **"FILENAME_received"** in the client's directory.

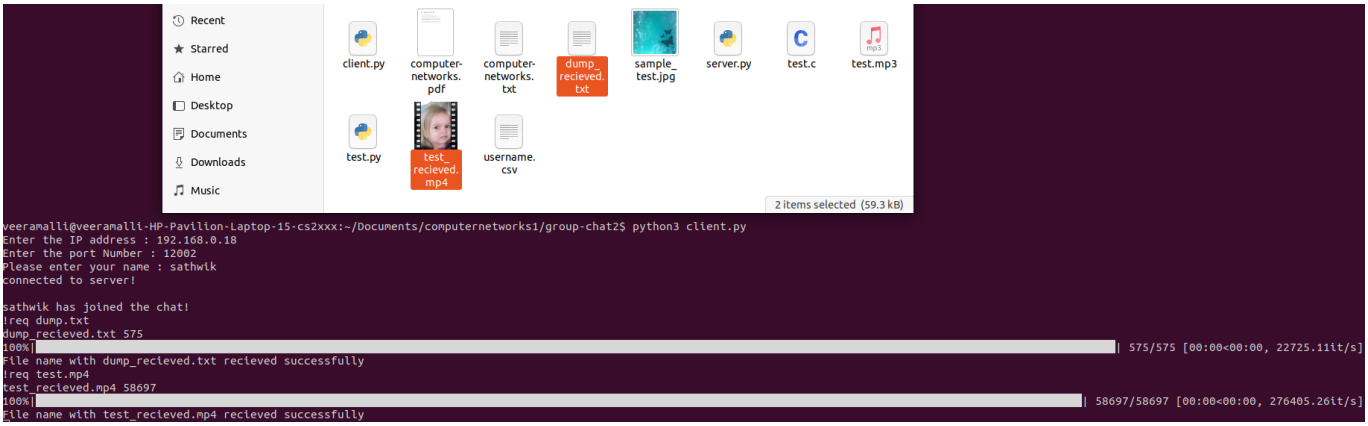

Figure 12: client receives files from server

8 ACTIVE CLIENTS INFORMATION

8.1 Server

On receiving the request from the client to display the names of the clients currently active, the server sends the list of names to the client and prints the message **"Command !online executed."**

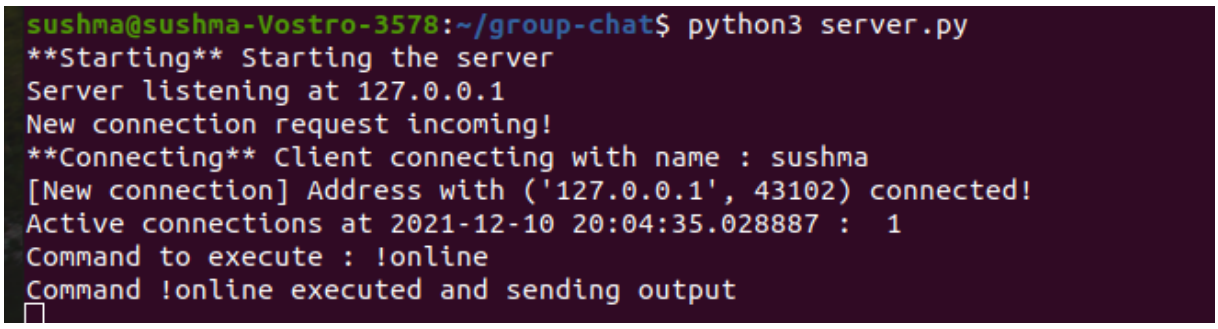

Figure 13: Server execution on request

8.2 Client

A client can request the list of active connections in the group using the command **"!online".** The client_names get printed in separate lines.

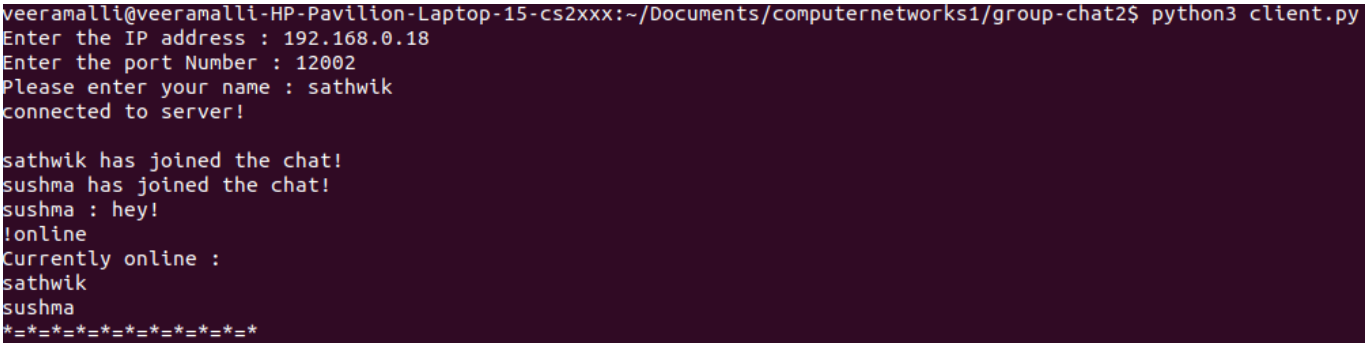

Figure 14: Server execution on request

9 Bank Authentication

9.1 Server

When a client requests to generate a QR code to scan, the server generates a unique QR code and sends it to the client. The QR Code is sent as an image with the filename "bank_request_received". The process ends with the message "QR code received successfully."

Figure 15: client QR request sent

9.2 Client

A client can use this network to get a QR code from a bank server to enable transactions using the command `!get QR`. The received popped-up QR code gets stored in the client's directory for 5 seconds.

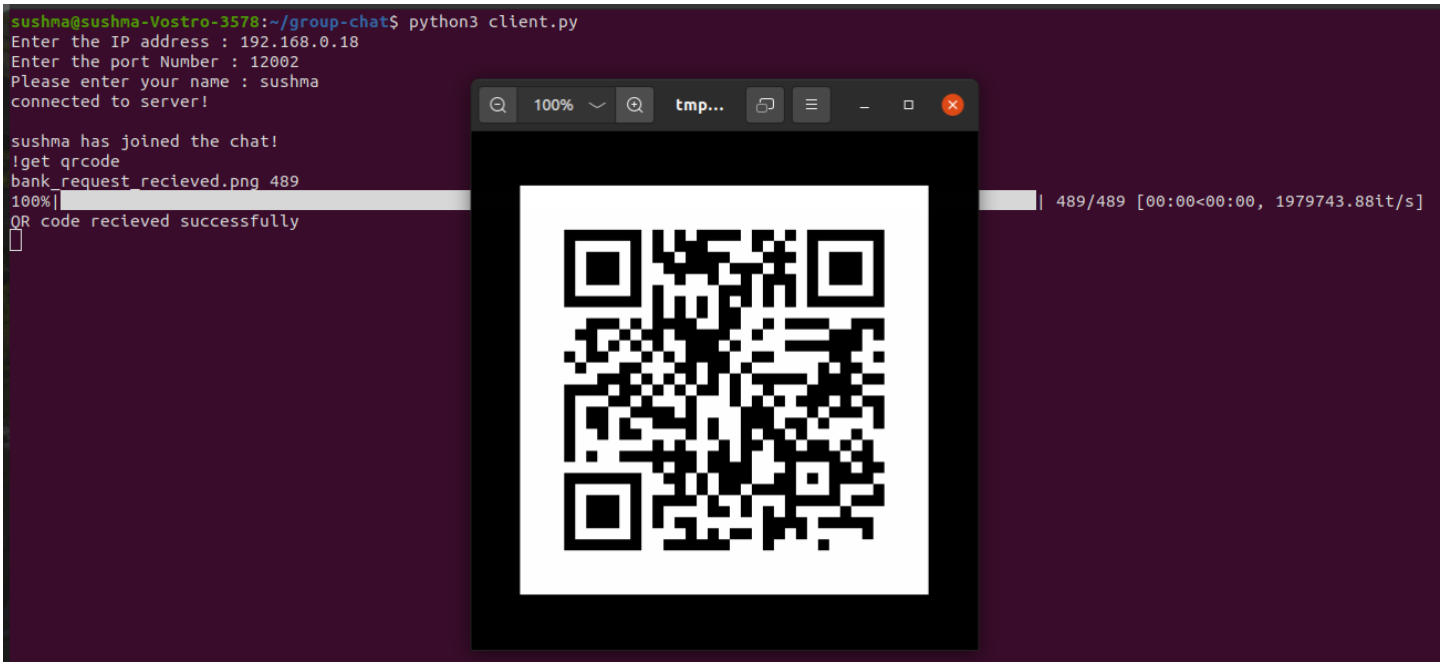

Figure 16: Unique Generated QR poped out

It then gets deleted automatically to simulate real life.

Figure 17: QR code gets deleted

10 Basic UNIX Command

10.1 Server

Command to execute: UNIX_COMMAND" is displayed by the server when a client requests to execute a UNIX_COMMAND. "Command executed and sending "

Example Commands:

- ls - allows to view a list of the files and folders in the server's directory

- date - displays the system date and time
- echo - displays line of text/string passed as an argument
- cat - allows to view contents of a file

```
sri@golden:~/Documents/acads/sem5/cn1/final_files$ python3 server.py
**Starting** Starting the server
Server listening at 192.168.0.18
New connection request incoming!
**Connecting** Client connecting with name : sushma
[New connection] Address with ('192.168.0.17', 60470) connected!
Active connections at 2021-12-10 18:11:09.416045 :  1
Command to execute :  date
Command executed and sending output
Command to execute :  echo computer_networks
Command executed and sending output
Command to execute :  cat dump.txt
Command executed and sending output
```

Figure 18: Requested Unix Commands got Executed

The commands rm, rf and sudo are restricted and "No privilages to run the command" is displayed.

```
sri@golden:~/Documents/acads/sem5/cn1/final_files$ python3 server.py
**Starting** Starting the server
Server listening at 192.168.0.18
New connection request incoming!
**Connecting** Client connecting with name : sushma
[New connection] Address with ('192.168.0.17', 60476) connected!
Active connections at 2021-12-10 18:13:52.143231 :  1
No previliges to run the command:rm dump.txt
No previliges to run the command:sudo
```

Figure 19: Restricted Unix Commands

10.2 Client

To execute the required Linux commands, a client can use the command `!cmd UNIX_COMMAND`. The respective output gets displayed to the client.

Figure 20: Unix Command Outputs

As mentioned above, the restricted commands are not accessible to the client and shows a warning.

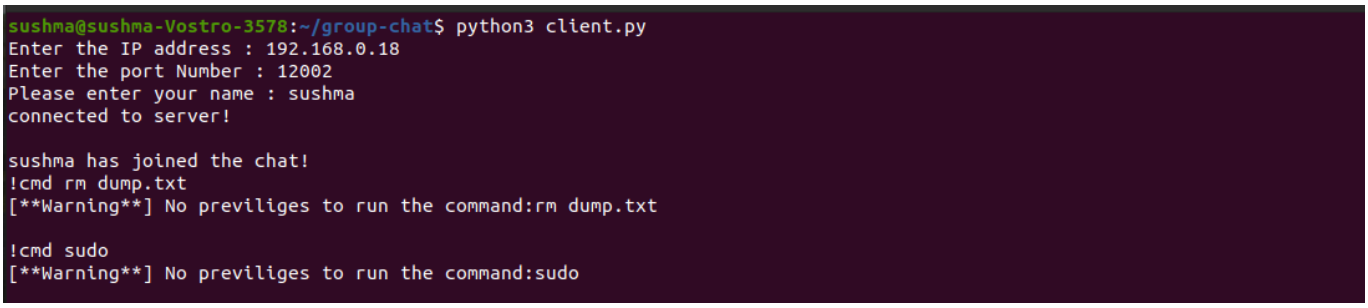

Figure 21: Restricted Unix Command outputs