CLI Based Chat Tool

Project Description

Building a command-line chat tool using Python. The tool will support multiple chat rooms and can be used to communicate with multiple users on a network. This project will be useful for anyone who wants to learn and implement socket programming as well as for anyone who wants to understand and build applications using client-server architecture.

Author

Bhargava N Reddy

Collaborator(s)

Ayush Kumar Shaw , Kiran Suresh

Project Language(s)

Python

Difficulty

Intermediate

Duration

35 hours

Prerequisite(s)

Python

Skills to be learned

Python, Socket Programming, Multithreading

Overview

Objective

Building a chat tool with a simple command-line interface which supports multiple chat rooms.

Project Context

Most often than not, we run heavy GUI-based applications for the simplest of tasks. One such simple task is instant messaging or chatting. Chat tools enable users to start chatting with other users in real-time. It also enables users to transmit text messages, images, videos, and hyperlinks.

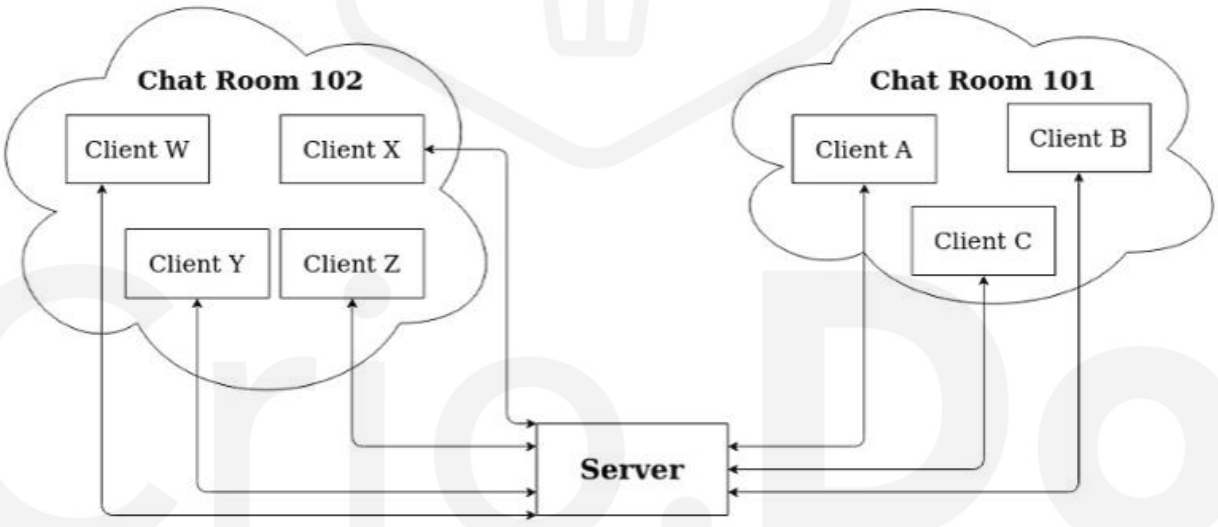
In this project, we aim to build a simple command-line chat tool which is easy to use and also has a very minimal interface.

Project Stages

- Understanding how to use `socket` and `threading` modules in Python.
- Implementing a simple client-server chat tool in Python.
- Incorporating multiple chat rooms functionality into the chat tool.

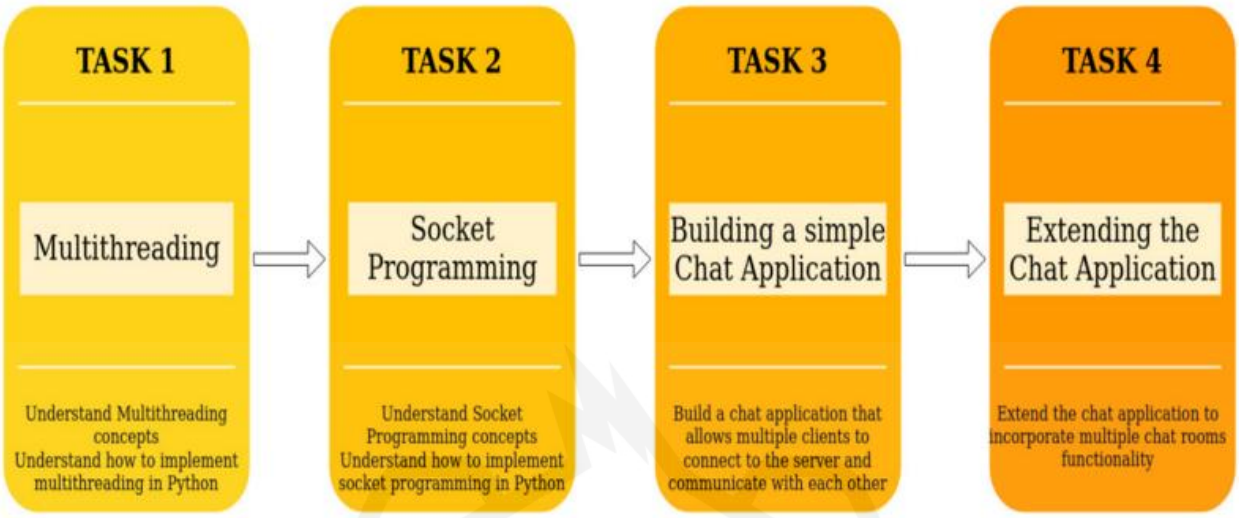
The diagram below shows the basic architecture of the chat tool. The server should be connected to all the clients directly but should be able to categorize them into their respective chat rooms and broadcast messages only to the sender's chat room.

For example: If Client A sends a message, the message should only be broadcasted to clients B and C and not any other clients.



High-level Approach

The diagram below shows a high level approach used to build and develop the chat tool.



The desired end result of the project is like this:
https://www.youtube.com/embed/HnnBJsr6fBs

Primary goals

- Understand socket programming how it can be used to build an application.
- Understand multithreading and how it helps a program to maximum utilize the CPU time.
- Build a fully functioning chat tool that allows multiple users to communicate with each other.
- Incorporate multiple chat rooms(or groups) functionality into the chat tool.

Applications

- Chat tools enable instant messaging and can be used in a number of scenarios.
- Command-line tools are often preferred over GUI applications by software developers.

Task 1

Multithreading

A thread is a light-weight smallest part of a process that can run concurrently with the other parts(other threads) of the same process. Threads are independent because they all have separate paths of execution. All threads of a process share the common memory. The process of executing multiple threads simultaneously is known as multithreading. (definition source: beginnersbook.com)

In this milestone, you need to understand the concepts of multithreading and learn how to implement multithreading in Python.

Requirements

- Understand the concept of multithreading.
- Familiarize yourself with basics of the `threading` module in Python.
- Implement a simple multithreaded program in Python.
  - Create a simple function in Python.
  - Call the function with different arguments in different threads.

References

- Multithreading
- Multithreading in Python

Expected Outcome

You should be able to understand how to implement multithreading in Python using the `threading` library.

Task 2

Socket Programming

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes. (definition source: tutorialspoint.com)

In this milestone, you have to understand socket programming and have to implement a simple client-server program and establish a connection between the client and server.

Requirements

- Understand the concept of socket programming.
- Familiarize yourself with basics of `socket` module in Python.
- Implement a simple client-server program in Python using the `socket` module.
  - Start a chat server and accept connections from clients.
  - Connect to the server from a client (using a different terminal on the same machine or a different machine on the same local network).

- Once the connection is successful, send a message from client to server or vice-versa and close the connection.

References

- Understanding Socket Connections in Computer Networking
- Socket Programming in Python
- Socket Programming in Python (Video)

Expected Outcome

You should be able to implement a simple client-server program in Python using the `socket` module.

Task 3

Building a chat application

In this milestone you'll be developing a simple chat application. Extend the previously built client-server program to allow server to accept connections from multiple clients. Also, allow the server the to accept messages from all the client and broadcast the messages to all the clients.

Requirements

- Extend the program developed as part of previous task to allow connections from multiple clients to the server.
- Come up with a way to identify unique clients on the server.
- Allow the clients to send messages to the server.
- Broadcast the messages from the sender to all other clients through the server.

References

- Socket Programming with Multithreading - 1
- Socket Programming with Multithreading - 2

Note

- The clients shouldn't communicate with each other directly. The clients should only be connected to the server and therefore they should only send and receive messages to and from the server.

Expected Outcome

You should be able to achieve the following:

- Chat tool implemented in Python.

- Multiple clients should be able to communicate with each other through the chat server.

Hint 1

Using threads on both client-side and server-side programs.

Hint Description

- Use threading on the server-side program to create threads to handle incoming clients and for their operations.
- Use threading on the client-side program to create threads to perform client operations (sending and receiving messages).

Task 4

Extending the chat tool

Extend the functionality of the chat tool so it supports multiple chat rooms. The server should be able to manage multiple users in multiple chat rooms at a given time.

Requirements

- Extend the chat tool to support multiple chat rooms (i.e., the clients should be able to join or create a chat room)
- When the client connects to the server, the client should be presented with a few options
  - List all existing chat rooms - This allows the client to list all the existing chat rooms.
  - Create a new chat room - This allows the client to create a new chat room.
  - Join an existing chat room - This allows the client to join an existing chat room.
- A client should only be able to join only one chat room.
- After joining a chat room, all messages sent by the client should only be broadcasted to other clients in the same room.

Tip

- Use unique identifiers to identify the chat room (for example: Room ID or Room Number).

Note

- Remember to cover boundary cases such as
  - Clients creating a new chat room with an already existing Room ID or Room Number
  - Clients trying to join an invalid room

Bring it On!

- Limit the number of clients that can join a chat room.
- Add functionality to create private chat rooms.
  - These chat rooms shouldn't be listed when the user chooses to list all the existing chat rooms.

Expected Outcome

You should be able to achieve the following:

- Chat tool with multiple room functionality.
- The clients should be able to do the following
  - List all the existing chat rooms.
  - Create a new chat room.
  - Join an existing chat room.
- Messages in sent by a client in a chat room should be only broadcasted to same chat room.
- Multiple chat rooms should be able to function at the same time.
  - Each chat room should allow communication between multiple clients.

Task 5

Spice it up!

As the basic implementation is all done, for all the curious cats out there, these are some of the line items which can be implemented to spice up the existing functionality.

[**NOTE**: This is not a mandatory milestone.]

Requirements

- Saving the chat history on disk for later use.
- Store the chat history of each chat separately.
- The chat history can be saved in multiple ways
  - Custom objects(`dict` or objects of a class) can be created with a specified format and saved using Python's `pickle` module.
  - It can be saved as JSON-like objects using NoSQL databases such as TinyDB or MongoDB.
  - It can also be saved in a table-like structure using relational databases such as SQLite or MySQL.

References

- Pickle Module in Python - Documentation
- TinyDB - Documentation

- MongoDB Getting started guide
- SQLite - Documentations
- MySQL with Python - Getting started guide

Tip

- Use a well-defined structure (containing sender name, message, etc.) to store each message in the chat history.

Expected Outcome

The chat history for all the chat rooms should be stored in files or databases.

Task 6

Spice it up X2!

As the basic implementation is all done, for all the curious cats out there, these are some of the line items which can be implemented to spice up the existing functionality.

[**NOTE**: This is not a mandatory milestone.][**P. S.** : This milestone is independent of milestone 5. Meaning, you can skip milestone 5 and implement this one if you want to.]

Requirements

- Setup Port forwarding on your router and modify the server program accordingly.
  - Forward the incoming requests to router from Port X to Port Y of the local machine.
- Make sure all the chat tool functionality is working as expected over the internet as well.

References

- Port Forwarding
- Socket Connection over Internet - 1
- Socket Connection over Internet - 2

Expected Outcome

- Clients on a different network should be able to connect to chat server using the server's public IP.
- The chat tool should function as expected over the internet.