OS MINI PROJECT

RA2211003010666 - Rohan S

RA2211003010402 - Pankaj Krishna

RA2211003010379

P S Krishna Vardhan

Problem Statement

Design and implement a simple proxy server program that facilitates network communication between clients and remote servers. The proxy server should serve as an intermediary, forwarding requests and responses while allowing for basic functionality such as caching and filtering. The primary objective of this project is to create a lightweight and functional proxy server that can be used for various purposes, such as improving network performance or enforcing content filtering policies.

Proxy Server Functionality:

The proxy server should be able to accept incoming requests from client applications and forward them to the target server, receiving the server's responses in return.
It should support both HTTP and HTTPS requests, handling HTTP and HTTPS traffic accordingly.
The server should properly maintain the connection with both the client and the remote server.

Caching:

Implement a caching mechanism to store and manage recently requested web content. The proxy server should serve cached content to clients when applicable, reducing latency and network load.

Request and Response Handling:

The proxy should be capable of modifying or filtering requests and responses based on predefined rules. For example, it could block specific websites, modify headers, or log traffic.
Logging and tracking of requests and responses should be available for monitoring and analysis.

PURPOSE OF A PROXY SERVER

The purpose of a proxy server can vary depending on the specific use case and requirements, but its primary functions and purposes include:

Enhancing Security:

Proxy servers can act as an intermediary between clients and remote servers, adding a layer of security. They can hide the client's IP address, making it more challenging for malicious entities to identify the client's location or initiate direct attacks.

Content Filtering and Access Control:

Organizations often use proxy servers to enforce content filtering policies. They can restrict access to specific websites or content categories, ensuring compliance with acceptable usage policies and protecting users from malicious or inappropriate content.

Caching and Acceleration:

Proxy servers can cache frequently requested web content. When a client requests cached content, the proxy can deliver it quickly, reducing the load on the internet connection and improving overall performance.

Load Balancing:

Proxy servers can distribute client requests to multiple backend servers, ensuring even load distribution. This helps optimize server performance and prevents server overload, particularly in high-traffic environments.

Anonymity and Privacy:

Users can use proxy servers to hide their IP addresses, enhancing online privacy and anonymity. This is particularly valuable in regions with strict internet censorship or for users who want to keep their online activities private.

Bypassing Geo-Restrictions:

Proxy servers located in different regions or countries can help users bypass geo-restrictions, allowing them to access content that might be limited to specific geographic areas.

Bandwidth Savings:

Caching proxy servers can save bandwidth by serving cached content to clients, reducing the need to download the same data repeatedly from the internet.

Monitoring and Logging:

Proxy servers can log network traffic, providing insights into user activities and usage patterns. This is useful for network administrators to monitor and analyze traffic.

Network Performance and Optimization:

Proxy servers can optimize network traffic by reducing latency, compressing data, and accelerating content delivery. This results in a faster and more efficient internet experience for clients.

Content Transformation:

Proxies can modify the content of web pages, such as resizing images or altering HTML, to improve performance or adapt content for specific devices.

Access to Restricted Resources:

Users can access resources on the internet that may be otherwise restricted or blocked in their location due to censorship or regional restrictions.

Testing and Development:

Developers and testers can use proxy servers to simulate various network conditions, test applications in different geographical locations, and troubleshoot network-related issues.

The specific purpose of a proxy server can vary based on the needs of the individual or organization using it. It is a versatile tool that can enhance security, privacy, network performance, and control over internet traffic.

CODE

Open *server.py ~/ Save

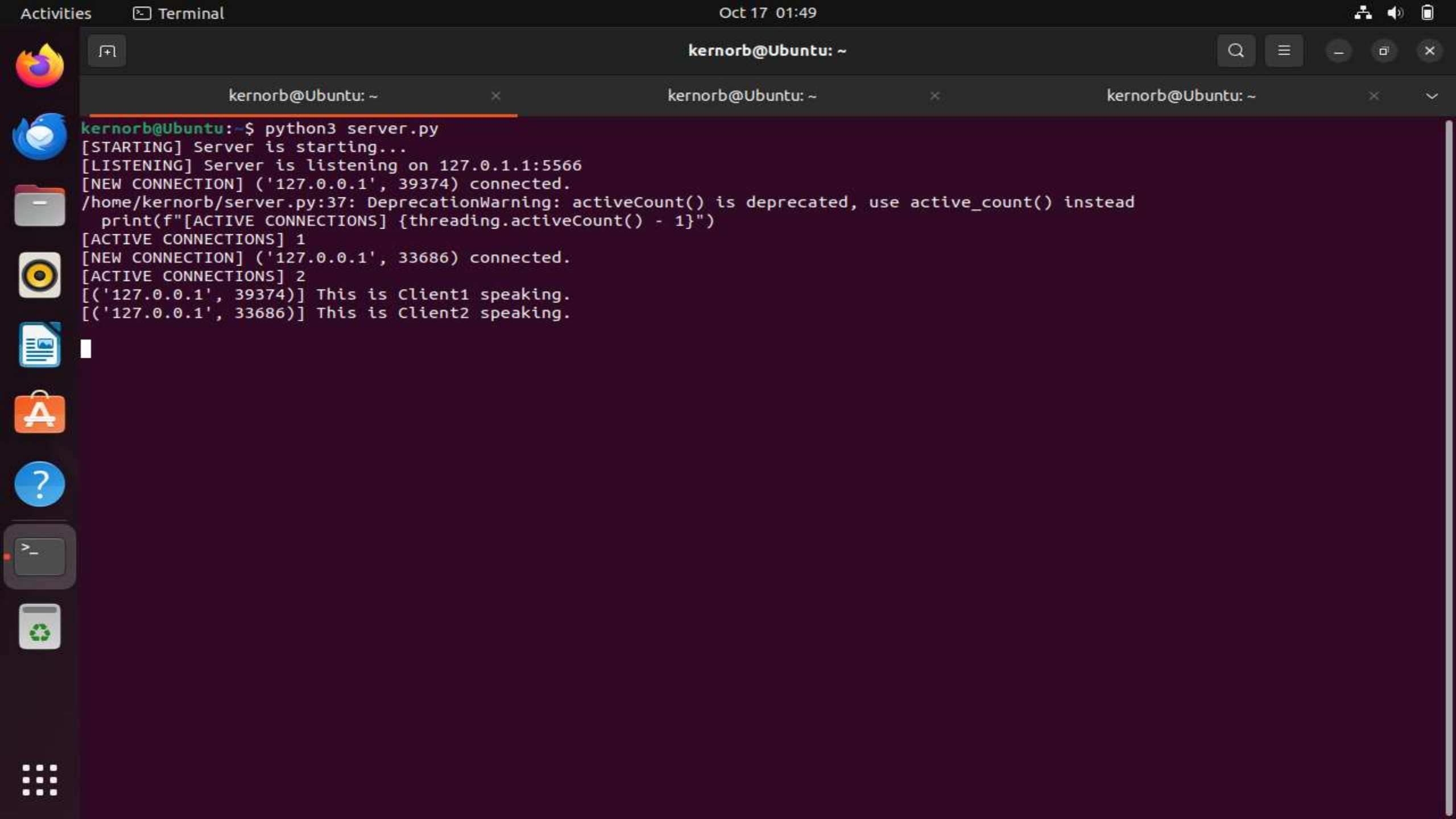
```python
import socket
import threading
IP = socket.gethostbyname(socket.gethostname())
PORT = 5566
ADDR= (IP,PORT)
SIZE=1024
FORMAT = "utf-8"
DISCONNECT_MSG = "!DISCONNECT"

def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True
    while connected:
        msg = conn.recv(SIZE).decode(FORMAT)
        if msg == DISCONNECT_MSG:
            connected = False

        print(f"[{addr}] {msg}")
        msg=f"Msg received: {msg}"
        conn.send(msg.encode(FORMAT))

    conn.close()

def main():
    print("[STARTING] Server is starting...")
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    print(f"[LISTENING] Server is listening on {IP}:{PORT}")

    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

if __name__ == "__main__":
    main()
```

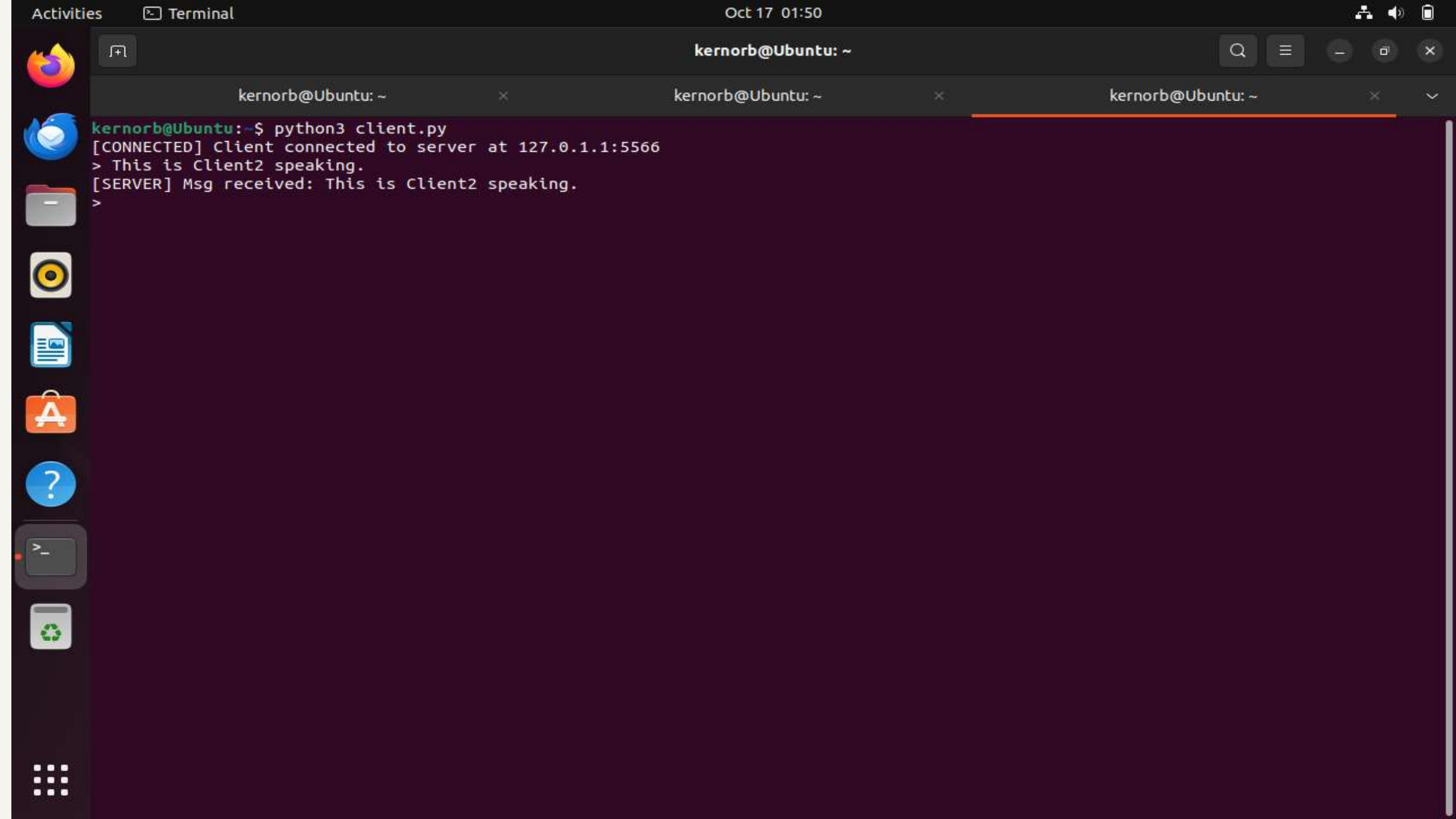
Open client.py ~/ Save

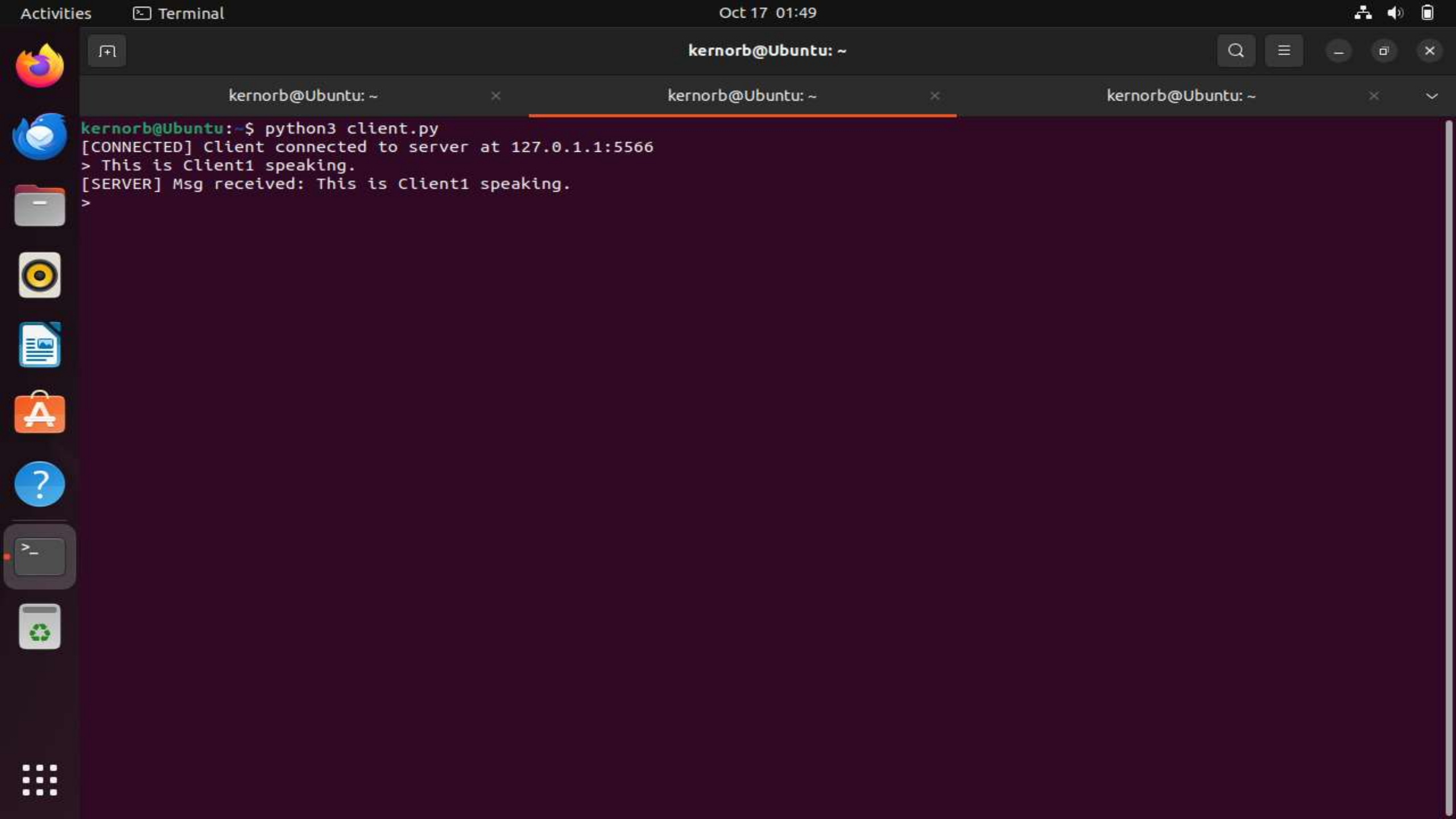
*server.py client.py

```python
import socket

IP = socket.gethostbyname(socket.gethostname())
PORT = 5566
ADDR = (IP, PORT)
SIZE = 1024
FORMAT = "utf-8"
DISCONNECT_MSG = "!DISCONNECT"

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Client connected to server at {IP}:{PORT}")

    connected = True
    while connected:
        msg = input("> ")

        client.send(msg.encode(FORMAT))

        if msg == DISCONNECT_MSG:
            connected = False
        else:
            msg = client.recv(SIZE).decode(FORMAT)
            print(f"[SERVER] {msg}")

if __name__ == "__main__":
    main()
```

OUTPUT


Activities
Terminal
Oct 17 01:49
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu:~$ python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on 127.0.1.1:5566
[NEW CONNECTION] ('127.0.0.1', 39374) connected.
/home/kernorb/server.py:37: DeprecationWarning: activeCount() is deprecated, use active_count() instead
print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")
[ACTIVE CONNECTIONS] 1
[NEW CONNECTION] ('127.0.0.1', 33686) connected.
[ACTIVE CONNECTIONS] 2
[('127.0.0.1', 39374)] This is Client1 speaking.
[('127.0.0.1', 33686)] This is Client2 speaking.


Activities
Terminal
Oct 17 01:50
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu:~$ python3 client.py
[CONNECTED] Client connected to server at 127.0.1.1:5566
> This is Client2 speaking.
[SERVER] Msg received: This is Client2 speaking.
>


Activities
Terminal
Oct 17 01:49
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu: ~
kernorb@Ubuntu:~$ python3 client.py
[CONNECTED] Client connected to server at 127.0.1.1:5566
> This is Client1 speaking.
[SERVER] Msg received: This is Client1 speaking.
>

THANK YOU