server.py

```python

import socket
import threading


TOKEN = "TOKEN"
PORT = 8080
BUFFER_SIZE = 1024




class TokenRingServer:
    def __init__(self):
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.clients = []
        self.client_threads = []
        self.running = False


    def start(self):
        self.server_socket.bind(("localhost", PORT))
        self.server_socket.listen()
        self.running = True
        print("Server started. Listening for connections...")


        try:
            while self.running:
                ## Accept new connections
                client_socket, client_address = self.server_socket.accept()
                print(f"New client connected: {client_address}")
                self.clients.append(client_socket)


                ## If this is the first client, send the token
                if len(self.clients) == 1:
                    # Send the token to the first client
                    client_socket.send(TOKEN.encode())


                ## Start a new thread to handle the client
                thread = threading.Thread(
                    target=self.handle_client, args=(client_socket,)
                )
                thread.start()


                self.client_threads.append(thread)


        except KeyboardInterrupt:
            self.stop()


    def handle_client(self, client_socket):
        while self.running:
            ## Receive data from the client
            data = client_socket.recv(BUFFER_SIZE).decode()


            ## select the next client to send the token to
            next_client = self.clients[
                (self.clients.index(client_socket) + 1) % len(self.clients)
            ]


            ## If the client sends CLOSE, remove it from the list of clients and close the connection
            if data == "CLOSE":
                print(f"Client disconnected: {client_socket.getpeername()}")
                self.clients.remove(client_socket)
                client_socket.close()
                data = TOKEN
                break


            ## If the client sends TOKEN, send it to the next client
            if data == TOKEN:
                print("Received token")
                if len(self.clients) >= 1:
                    if self.running:
                        print("Sending token to next client")
                        next_client.send(TOKEN.encode())


                    else:
                        print("Server stopped. Not sending token to next client")
                        break


    def stop(self):
        self.running = False


        print("Closing server..")


        ## Send close signal to all clients
        for client in self.clients:
            print(f"Sending close signal to {client.getpeername()}")
            client.send("CLOSE".encode())
            client.close()


        ## Wait for all threads to finish
        for thread in self.client_threads:
            thread.join()


        self.server_socket.close()




if __name__ == "__main__":
   server = TokenRingServer()
   server.start()



```

client.py

```python
import socket


SERVER_ADDRESS = ("localhost", 8080)
BUFFER_SIZE = 1024




class TokenRingClient:
    def __init__(self):
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


    def connect(self):
        self.client_socket.connect(SERVER_ADDRESS)
        print("Connected to server")


    def start(self):
        try:
            while True:
                data = self.client_socket.recv(BUFFER_SIZE).decode()
                if data == "TOKEN":
                    print("Token received. Accessing resource.")
                    # Perform operations on the resource


                    # Simulating work on the resource
                    print("Working on the resource...")
                    # Simulating work by sleeping for 5 seconds
                    import time


                    time.sleep(5)


                    print("Resource access complete. Releasing token.")
                    self.client_socket.send("TOKEN".encode())


                if data == "CLOSE":
                    print("Closing client..")
                    self.stop()
                    break


        except KeyboardInterrupt:
            print("Closing client..")
            self.client_socket.send("CLOSE".encode())
            self.stop()


    def stop(self):
        self.client_socket.close()




if __name__ == "__main__":
    client = TokenRingClient()
    client.connect()
    client.start()



```