```python
# ------------------------------------------------
# Server
# ------------------------------------------------


import queue
import socket
import threading
import time

host = "127.0.0.1"  # Set server ip
port = 55556  # Set server port

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Select Internet and TCP protocol
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  # Make server address reusable
server.bind((host, port))  # Set host ip and port
server.listen()  # Listen for incoming connections

clients = []  # List of active clients
nicknames = []  # List of nicknames for active clients

broadcast_queue = queue.Queue()  # Queue for collecting thread messages and sending them in order to clients


def cli():  # A simple command line function for listing users and kikcking the form server
    while True:

        cli_in = input(">>")

        if cli_in == "-help":
            print(f'Valid CLI commands:\n'
                  f'<list> List all activ users\n'
                  f'<kick> Remove user from server\n'
                  f'<-help> or <man> Display help information')

        elif cli_in == "list":
            print("List of connected clients: ")
            for nickname in nicknames:
                print(nickname)

        elif cli_in == "kick":
            kick = input("Enter name of client to kick:")
            for n in nicknames:
                if n == kick:
                    index = nicknames.index(kick)
                    client = clients[index]
                    client.send('KICK'.encode('utf8'))
        else:
            print(f'{cli_in} not a valid input command ')


def broadcast_q():  # Function for sending a message from one client to other clients

    while True:
        message = broadcast_queue.get()

        try:
            # Gets first string in message and finds name tag

            name_tag = message.decode('utf8').split()[0].replace(":", "")

            sender_index = nicknames.index(name_tag)  # Finds index of sender

```

```
64              # Makes sender_list that sends to every one except sender
65
66              send_list = [element for i, element in enumerate(clients) if i not in {
    sender_index}]
67
68              for client in send_list:  # Sends message to clients in list
69                  client.send(message)
70                  time.sleep(0.001)
71          except:
72              print("User disconnected ")
73              # Sending disconnect message to everyone
74
75              for client in clients:  # Sends message to clients in list
76                  client.send(message)
77                  time.sleep(0.001)
78
79
80  def handel(client):  # Function for handling clients if client not available remove
    client from server
81
82      while True:
83          message = client.recv(1024)
84          if message:  # if message is not zero byte and not kicked
85              broadcast_queue.put(message)
86          else:  # when client disconnects zero byte stream is send / Disconnect
    client and end stop thread
87              index = clients.index(client)
88              clients.remove(client)
89              client.close()
90              nickname = nicknames[index]
91              broadcast_queue.put(f'{nickname} left the chat '.encode('utf8'))
92              nicknames.remove(nickname)
93              print(f'client removed {nickname}')
94              break
95
96
97  def set_keepalive(sock, after_idle_sec=1, interval_sec=3, max_fails=5):
98      """Set TCP keepalive on an open socket.
99
100     It activates after 1 second (after_idle_sec) of idleness,
101     then sends a keepalive ping once every 3 seconds (interval_sec),
102     and closes the connection after 5 failed ping (max_fails), or 15 seconds
103
104     https://www.programcreek.com/python/example/4925/socket.SO_KEEPALIVE example 17
105     """
106     if hasattr(socket, "SO_KEEPALIVE"):
107         sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
108     if hasattr(socket, "TCP_KEEPIDLE"):
109         sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPIDLE, after_idle_sec)
110     if hasattr(socket, "TCP_KEEPINTVL"):
111         sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPINTVL, interval_sec)
112     if hasattr(socket, "TCP_KEEPCNT"):
113         sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPCNT, max_fails)
114
115
116 def receive():
117     while True:
118         client, adress = server.accept()  # Looking for connection
119         print(f'Conected with {str(adress)}')  # Server side system message
120         set_keepalive(client)  # Keep TCP connection alive to
121         client.send('NICK'.encode('utf8'))  # Asking for nickname from client
122         nickname = client.recv(1024).decode('utf8')  # Receive nickname and store
    nickname and client in lists
123         if nickname in nicknames: # Error message if Nick is in use
124             print(f"{nickname} already in use")
125             client.send('NICK_INVALID'.encode('utf8'))
126         else:
```

```
127             client.send('NICK_OK'.encode('utf8')) # Nick is ok and registered
128             nicknames.append(nickname)
129             clients.append(client)
130             print(f'Nickname of connected client is {nickname}')  # Server side
    system message
131             broadcast_queue.put(f'{nickname} just connected'.encode('utf8'))  #
    Broadcast new user conection
132             client.send(
133                 'Connection successful, welcome to ChatyChaty !'.encode('utf8'))  #
    Tell user client that they are
134             # connected to server
135
136             # Threading to be enabled to handel multiple clients
137             thread = threading.Thread(target=handel, args=(client,))
138             thread.start()
139
140             print(f'Thread count:{threading.active_count()}')
141
142
143 def main():
144     # Starting threads for receive(), broadcast_q () and cli()
145     print("Server started")
146     thread_receive = threading.Thread(target=receive)
147     thread_receive.start()
148     thread_broadcast_q = threading.Thread(target=broadcast_q)
149     thread_broadcast_q.start()
150     thread_cli = threading.Thread(target=cli)
151     thread_cli.start()
152
153
154 if __name__ == "__main__":
155     main()
156
```