UDP Server Script

The UDP server script listens for datagrams from clients on a specified IP and port using a `SOCK_DGRAM` (UDP) socket. When a new client sends a message, their address is registered and treated as their username. The server then broadcasts that client's messages to all other connected clients without maintaining a persistent connection, since UDP is connectionless. All messages are received using `recvfrom()`, decoded, and then relayed using `sendto()`. The server handles graceful shutdown on a keyboard interrupt (`Ctrl+C`) by broadcasting a shutdown message and closing the socket.

Here is a detailed explanation for the UDP server script of the multi-client chat application.

File Header and Imports

```
#!/usr/bin/python3
"""UDP Server Script"""
```

- **Shebang**: Indicates that the script should be executed using Python 3.
- **Docstring**: Describes the purpose of the script, which is to implement a UDP server.

```
import signal
import socket
import sys
```

Imports:

- **`signal` :** Allows handling of asynchronous events, such as interrupts (e.g., Ctrl+C), enabling graceful shutdown of the server.
- **`socket` :** Provides access to the BSD socket interface for network communication, specifically for creating and managing sockets.
- **`sys` :** Provides access to system-specific parameters and functions, such as exiting the program.

Server Configuration Constants

```
SERVER_IP = "127.0.0.1"
SERVER_PORT = 65432
BUFFER_SIZE = 2048
```

Constants:

- **`SERVER_IP` :** The IP address on which the server will listen for incoming UDP packets (localhost in this case)
- **`SERVER_PORT` :** The port number on which the server will accept incoming UDP packets.
- **`BUFFER_SIZE` :** The maximum size of the incoming message buffer (2048 bytes), which defines how much data can be received in one go.

Client Management Dictionary and Socket Initialization

```
clients = {}  # addr -> username
server_socket = None
```

Clients Dictionary:

- **`clients` :** A dictionary that maps client addresses (IP and port) to their usernames.
- This helps in identifying clients and broadcasting messages to them.

Server Socket:

- **`server_socket` :** A variable to hold the server socket object, initialized to **`None`** .

Message Handling Function

```
def handle_new_message(data, addr):
    """Handles new messages"""
```

- **Function Definition:** Defines a function to handle incoming messages from clients. It takes two parameters: **`data`** (the message received) and **`addr`** (the address of the client).

```
message = data.decode("utf-8")
```

- **Message Decoding**: Decodes the incoming byte data into a UTF-8 string for processing. This is necessary because data received over sockets is in bytes.

```
if addr not in clients:
    clients[addr] = message
    broadcast_message(f"[SERVER] {message} has joined the chat.")
```

- **New Client Handling:** Checks if the client's address is not already in the `clients` dictionary.
- If it's a new client, it adds the address and username to the dictionary and broadcasts a message indicating that the client has joined the chat.

```
else:
    full_message = f"{clients[addr]} ~ {message}"
    print(full_message)
    broadcast_message(full_message)
```

- **Existing Client Handling**:
  - If the client already exists in the dictionary, it constructs a message that includes the client's username and the message they sent.
  - Prints the full message to the server console and broadcasts it to all clients.

Broadcast Function

```
def broadcast_message(message: str):
    """Broadcastes messages to all clients"""
```

- **Function Definition**: Defines a function to send messages to all connected clients. It takes a single parameter, `message` , which is the message to be sent.

```
for client_addr in clients:
```

- **Loop Through Clients:** Iterates over all client addresses stored in the `clients` dictionary.

```
try:
    server_socket.sendto(message.encode("utf-8"), client_addr)  # pyright: ignore
```

- **Sending Messages**:
  - Attempts to send the encoded message to each client using **`sendto() ,`** which is specific to UDP sockets.
  - **`sendto()`** takes the message (encoded as bytes) and the client's address as arguments.

```
except Exception as e:
    print(f"[SERVER] Failed to send to {client_addr}: {e}")
```

- **Error Handling**: Catches any exceptions that occur during message sending and prints an error message to the server console. This is important for debugging and ensuring the server remains operational.

Server Shutdown Function

```
def shutdown_server():
    """Handles server shutdown and disconnects all clients"""
```

- **Function Definition**: Defines a function to handle server shutdown procedures.

```
print("\n[SERVER] Shutting down...")
```

- **Shutdown Message**: Prints a message indicating that the server is shutting down.

```
broadcast_message("[SERVER] Server is shutting down.")
```

- **Broadcast Shutdown Message**: Sends a message to all clients informing them that the server is shutting down.

```
if server_socket:
    server_socket.close()
```

- **Socket Closure:** Checks if the `server_socket` is initialized and closes it to free up the port and resources.

```
sys.exit(0)
```

- **Exit Program**: Exits the program with a status code of 0, indicating successful termination.

Main Function

```
def main():
    """Server main function"""
```

- **Function Definition**: Defines the main function that will run the server.

```
global server_socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

- **Global Declaration:** Declares `server_socket` as a global variable to be used within the function.
- **Socket Creation:** Creates a UDP socket using IPv4, similar to the client setup.

```
server_socket.bind((SERVER_IP, SERVER_PORT))
```

- **Binding**: Binds the socket to the specified IP address and port, allowing it to listen for incoming messages on that address.

```
print(f"[SERVER] Running on {SERVER_IP}:{SERVER_PORT}")
```

- **Server Start Message**: Prints a message indicating that the server is running and listening for connections.

```
while True:
    try:
        data, addr = server_socket.recvfrom(BUFFER_SIZE)
```

- **Infinite Loop:** Enters an infinite loop to continuously listen for incoming messages.
- **Receiving Data:** Uses `recvfrom()` to receive data from clients, which returns the data and the address of the sender.

```
handle_new_message(data, addr)
```

- **Message Handling:** Calls the **`handle_new_message()`** function to process the received data and address.

```
except Exception as e:
    print(f"[SERVER] Error: {e}")
```

- **Error Handling**: Catches any exceptions that occur during message reception and prints an error message to the server console.

Signal Handling

```
# Register signal handler
signal.signal(signal.SIGINT, lambda sig, shutdown_server())server())
```

- **Signal Registration:** Registers a signal handler for the SIGINT signal (triggered by Ctrl+C). When this signal is received, it calls the **`shutdown_server()`** function to gracefully shut down the server.

Entry Point

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

- **Main Entry Point**: Checks if the script is being run directly (not imported as a module) and calls the **`main()`** function to start the server.