Multi-Threading Example: A Simple Multi-Threaded TCP server

Here, we want to change the simple server program below to a multi-threaded server:

The server gets a message from the client, transforms it to uppercase and sends it back to the client (the code is from the textbook and discussed in the lectures already). Here is the server code:

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET,SOCK_STREAM)

serverSocket.bind(('',serverPort))

serverSocket.listen(1)

print('The server is ready to receive')

while True:

    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()

    capitalizedSentence = sentence.upper()

    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()
```

In your python file, import threading module to be able to work with threads:

```
import threading
```

After creating a server socket, binding it and listening to the incoming connections, accept method is called in an infinite while loop. After accept gets an incoming connection and returns, create a thread using threading.Thread constructor. This thread will execute the handle_client function which we are going to define later, and pass connectionSocket and addr to it as arguments (addr tuple holds an IP address and a port of the connected client). After the thread is created, we call start on it to begin its execution.

Note that accept is a blocking code and as soon as the client connects, accept method returns, and we continue the execution: spawn a thread, which will handle said client and go to the next iteration where we will again halt at the accept call waiting for another client to connect.

Here is code that we have already got:

```
serverPort = 12000

serverSocket = socket(AF_INET,SOCK_STREAM)

serverSocket.bind(('',serverPort))

serverSocket.listen(1)

print('The server is ready to receive')

while True:

    connectionSocket, addr = serverSocket.accept()

    print(f"Accepted connection from {addr[0]}:{addr[1]}")
    # start a new thread to handle the client
    thread = threading.Thread(target=handle_client, args=( connectionSocket, addr,))
    thread.start()
```

Creating client-handling function to run in a separate thread

Now, we need to define another function called handle_client. This function will be the one executing in a separate thread for every client's connection. It receives the client's socket object and the addr tuple as arguments.

Inside this function, we do the same as we did in a single threaded case for handling request of each client:

```
def handle_client(connectionSocket, addr):
    sentence = connectionSocket.recv(1024).decode()

    capitalizedSentence = sentence.upper()

    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()
```

Complete multithreaded server code example

```
import threading

from socket import *


def handle_client(connectionSocket, addr):
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()

    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()


serverPort = 12000

serverSocket = socket(AF_INET,SOCK_STREAM)

serverSocket.bind(('',serverPort))

serverSocket.listen(1)

print('The server is ready to receive')

while True:

    connectionSocket, addr = serverSocket.accept()

    print(f"Accepted connection from {addr[0]}:{addr[1]}")
    # start a new thread to handle the client
    thread = threading.Thread(target=handle_client, args=( connectionSocket, addr,))
    thread.start()
```

Testing the multithreading example

If you want to test the multi-client implementation, open several terminal windows for clients and one for the server. First start the server. After that start a couple clients. In the server terminal windows you will see how new clients get connected to the server. You can now proceed with sending messages from different clients by entering text into the respective terminals and all of them will be handled on the server side.

Practice

Try to change your lab 1 chatting server program to a multi-threaded server.

Note: You don’t need to change client code.