Tecnológico Nacional de México campus Colima

Maestría en Sistemas Computacionales

Servidor Socket Multihilo TCP y Cliente TCP en Python

Ejercicio Práctico:Servidor Socket Multihilo TCP

Docente:

D. en C. Patricia Figueroa Millan

Autor:

Ing. Gilberto Rene Martinez Gutierrez G2146013

Villa de Álvarez, Colima, México

05 de noviembre de 2022

Objetivo

Desarrollar un ejercicio sobre los usos e implementacion de un servidor multihilo a traves del uso de tecnologia TCP

Metodologia

Para la resolucion del problema se realizo un analisis previo sobre el funcionamiento de un "Servidor Socket Multihilo TCP" donde se recurrio al portal de recursos python , el contenido de este material facilito el entendimiento , ademas de permitir analizar unos ejemplos de su funcionamiento

Servidor TCP Multi-hilo

Servidor TCP multihilo o multithreaded para lograr la conexión simultánea de varios clientes. El código hace uso de los módulos estándar threading y socket, escucha peticiones en el puerto 6030 y, ante la llegada de datos, responde con la misma información (Echo server).

Ejemplo

In [ ]:
```
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = ('localhost', 10000)
print('starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(16)
            print('received {!r}'.format(data))
            if data:
                print('sending data back to the client')
```

```
                connection.sendall(data)
            else:
                print('no data from', client_address)
                break

    finally:
        # Clean up the connection
        connection.close()
```

cliente TCP

Puede utilizarse el siguiente cliente que conecta al servidor y permite demostrar la eficacia del código.

In [ ]:
```
import socket
import sys


def get_constants(prefix):
    """Create a dictionary mapping socket module
    constants to their names.
    """
    return {
        getattr(socket, n): n
        for n in dir(socket)
        if n.startswith(prefix)
    }


families = get_constants('AF_')
types = get_constants('SOCK_')
protocols = get_constants('IPPROTO_')

# Create a TCP/IP socket
sock = socket.create_connection(('localhost', 10000))

print('Family  :', families[sock.family])
print('Type    :', types[sock.type])
print('Protocol:', protocols[sock.proto])
print()

try:

    # Send data
    message = b'This is the message.  It will be repeated.'
    print('sending {!r}'.format(message))
    sock.sendall(message)

    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print('received {!r}'.format(data))
```

```
finally:
    print('closing socket')
    sock.close()
```

Servidor y cliente

Los conectores se pueden configurar para que actúen como un servidor y escuchen mensajes entrantes, o se conecten a otras aplicaciones como un cliente. Después de que ambos extremos de un conector TCP/IP estén conectados, la comunicación es bidireccional.

In [ ]:

```
import threading
import socket
import sys

def server():

    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Bind the socket to the port
    server_address = ('localhost', 10000)
    print('starting up on {} port {}'.format(*server_address))
    sock.bind(server_address)

    # Listen for incoming connections
    sock.listen(1)

    while True:
        # Wait for a connection
        print('waiting for a connection')
        connection, client_address = sock.accept()
        try:
            print('connection from', client_address)

            # Receive the data in small chunks and retransmit it
            while True:
                data = connection.recv(16)
                print('received {!r}'.format(data))
                if data:
                    print('sending data back to the client')
                    connection.sendall(data)
                else:
                    print('no data from', client_address)
                    break

        finally:
            # Clean up the connection
            connection.close()


def get_constants(prefix):
    """Create a dictionary mapping socket module
    constants to their names.
    """
    return {
```

```
        getattr(socket, n): n
        for n in dir(socket)
        if n.startswith(prefix)
    }


def cliente():

    families = get_constants('AF_')
    types = get_constants('SOCK_')
    protocols = get_constants('IPPROTO_')

    # Create a TCP/IP socket
    sock = socket.create_connection(('localhost', 10000))

    print('Family  :', families[sock.family])
    print('Type    :', types[sock.type])
    print('Protocol:', protocols[sock.proto])
    print()

    try:

        # Send data
        message = b'---------Inicia Conexion------------'
        print(message)
        sock.sendall(message)

        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print('received {!r}'.format(data))

    finally:
        print('closing socket')
        sock.close()


if __name__ =="__main__":
    # creating thread
    t1 = threading.Thread(target=server)
    t2 = threading.Thread(target=cliente)
    t3 = threading.Thread(target=cliente)

    # starting thread 1
    t1.start()
    # starting thread 2
    t2.start()
    t3.start()

    # wait until thread 1 is completely executed
    t1.join()
    # wait until thread 2 is completely executed
    t2.join()
    t3.join()

    # both threads completely executed
    print("Done!")
```

Link del repositorio github

https://github.com/Gilberto-profesional/ProgramingPOO/tree/main/POOprograming/Servidor%20Socket%20Multihilo%20TCP%5