CoolplayDev

TCP Server: How to create a TCP server in Python

A TCP server is a very important component of communication between computers. Learn how to create a basic TCP server on your own with Python!

OLIVER ZINK / MARCH 10, 2022 / NETWORKING, PROGRAMMING

TCP servers are one of the most important components of communication between computers. In this article, you will learn how to create your own TCP server, that is able to accept multiple clients at once, with Python. You can upgrade this code for your needs and import it into any project of yours.

Introduction

Luckily, creating a **basic** TCP server in Python is very easy and only takes a few lines of code. However, you should know about the basics of programming and networking to really understand what's going on.

Before we get to the coding part, I want to briefly explain some key terms:

- TCP stands for **Transmission Control Protocol** and is part of the TCP/IP suite. The difference between TCP and UDP (**User Datagram Protocol**) is that TCP makes sure that there is a connection between devices and all packages are received by the receiver. UDP, however, just sends the packages no matter whether they are even received by anyone.

- Threading, in short, is a method to run multiple code sequences at the same time by distributing them to different parts of the processor. This allows the server to handle multiple clients at the same time.
- A socket is an interface for the exchange of data on the same device or between different devices. It is provided and managed by the operating system.

Code explanation

First, we need to include two modules – the socket module and the threading module. The socket module allows us to create a network socket for TCP communication. The threading module allows the server to handle multiple clients at the same time.

```
import socket
import threading
```

After we have done that, we need to set some variables for storing the server's IP address and the port that it should listen on. Additionally, we create a socket for the server and bind it to our IP and port.

```
bind_ip = "192.168.178.73" # Replace this with your own IP address
bind_port = 27700 # Feel free to change this port
# create and bind a new socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((bind_ip, bind_port))
server.listen(5)
print("Server is listening on %s:%d" % (bind_ip, bind_port))
```

The next step is to define a function to handle the connected clients. This function will encode and send "ready" to the client, wait until it receives a message from the client, decode it, and then close the connection again.

```
def clientHandler(client_socket):
    # send a message to the client
    client_socket.send("ready".encode())
    # receive and display a message from the client
    request = client_socket.recv(1024)
    print("Received \"" + request.decode() + "\" from client")
    # close the connection again
    client_socket.close()
    print("Connection closed")
```

The number **1024 in client_socket.recv(1024)** specifies the maximum amount of bytes the client receives from the server. That means if the server sends a string that is bigger than the specified amount of bytes, the client will not receive the complete string.

The last part is a simple endless loop that listens for new connections and starts new threads to handle them. This loop will run until the program gets closed.

```
while True:
    # wait for client to connect
    client, addr = server.accept()
    print("Client connected " + str(addr))
    # create and start a thread to handle the client
    client_handler = threading.Thread(target = clientHandler, args=(client,))
    client_handler.start()
```

At the moment, this server isn't very useful, but you can upgrade it however you want. Here are the most important lines for upgrading the server:

- **client_socket.send("any string".encode())**
  - This simple line of code sends a message to the client
- **request = client_socket.recv(1024)**
  - This line of code waits for the client to send a message to the server. You can later access the message with

request.decode()

- **client_socket.close()**
  - Closes the connection to a client
- **client, addr = server.accept()**
  - Waits for a client to connect. You can access the actual client with the **client** variable and its IP address and the port it connected on with the **addr** variable. To convert the addr variable to a string, use **str(addr)**

Here is the complete code:

```
import socket
import threading

bind_ip = "192.168.178.73" # Replace this with your own IP address
bind_port = 27700 # Feel free to change this port
# create and bind a new socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((bind_ip, bind_port))
server.listen(5)
print("Server is listening on %s:%d" % (bind_ip, bind_port))

def clientHandler(client_socket):
    # send a message to the client
    client_socket.send("ready".encode())
    # receive and display a message from the client
    request = client_socket.recv(1024)
    print("Received \"" + request.decode() + "\" from client")
    # close the connection again
    client_socket.close()
    print("Connection closed")

while True:
    # wait for client to connect
    client, addr = server.accept()
    print("Client connected " + str(addr))
    # create and start a thread to handle the client
    client_handler = threading.Thread(target = clientHandler, args=(client,))
    client_handler.start()
```

Summary

That's it! You now got your own TCP server that you can upgrade and use in any project you want. If you now want to test your server with a TCP client that connects to it, you can read this article on how to create a TCP client.

Thanks for reading!

Share this article

Interesting Reads

A Beginner's Guide to Web Scraping with Python [2023]

August 25, 2023

Python to Executable: Easily Convert a Python File to an Executable

April 23, 2023

SSH Server on Windows: How to set up OpenSSH on Windows

April 9, 2023

Navigation

- Home
- Blog
- Contact

Categories

- Programming
- Computer
- Networking

Tools

- Bin Dec Hex Converter
- LZW Encoder

Important Links

- Privacy Policy
- Terms & Conditions
- Imprint