

Blocking socket server

To illustrate the difference with a blocking socket approach, we'll create a simple blocking TCP server and a corresponding client. This server will handle one connection at a time in a blocking manner, meaning it will wait (or block) on I/O operations like accepting new connections or receiving data.

```
import socket

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 65432       # Port to listen on (non-privileged ports are > 1023)

# Create a socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((HOST, PORT))
    server_socket.listen()
    print(f"Server is listening on {HOST}:{PORT}")

    while True:
        # Accept a new connection
        conn, addr = server_socket.accept()
        with conn:
            print(f"Connected by {addr}")
            while True:
                data = conn.recv(1024)
                if not data:
                    break # No more data from client, close connection
                print(f"Received {data.decode()} from {addr}")
                response = "This is a response from the server.".encode()
                conn.sendall(response)
```

Blocking client

```
import socket

HOST = '127.0.0.1' # The server's hostname or IP address
PORT = 65432       # The port used by the server

# Create a socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    print("Connected to the server")

    # Send data
    message = 'Hello, server'.encode()
    s.sendall(message)
```

```
print("Message sent to the server")

# Wait for a response
data = s.recv(1024)
print("Received response from the server")

print(f"Received: {data.decode()}")
```

Non-blocking server

Creating a non-blocking TCP socket server in Python involves setting up a socket to listen for connections without blocking the main execution thread of the program. Below is a simple example of a non-blocking TCP server that accepts multiple client connections and handles them asynchronously. This server uses the select method, which is a way to check for I/O readiness on sockets, making it possible to manage multiple connections without blocking on any single one.

```
import socket
import select

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 65432       # Port to listen on (non-privileged ports are > 1023)

# Create a socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# Bind the socket to the address and port
server_socket.bind((HOST, PORT))

# Listen for incoming connections
server_socket.listen()

print(f"Listening on {HOST}:{PORT}")

# Set the server socket to non-blocking mode
server_socket.setblocking(0)

# Keep track of input sockets
inputs = [server_socket]
outputs = []

while inputs:
    # Wait for at least one of the sockets to be ready for processing
    readable, writable, exceptional = select.select(inputs, outputs, inputs)

    for s in readable:
```

```
    if s is server_socket:
        # Accept new connection
        connection, client_address = s.accept()
        print(f"New connection from {client_address}")
        connection.setblocking(0)
        inputs.append(connection)
    else:
        data = s.recv(1024)
        if data:
            # A readable client socket has data
            print(f"Received {data} from {s.getpeername()}")
            # Add output channel for response
            if s not in outputs:
                outputs.append(s)
        else:
            # Interpret empty result as closed connection
            print(f"Closing {client_address}")
            if s in outputs:
                outputs.remove(s)
            inputs.remove(s)
            s.close()

for s in writable:
    response = b'This is a response from the server.'
    s.send(response)
    # Once response has been sent, we don't need to write anymore
    outputs.remove(s)

for s in exceptional:
    print(f"Handling exceptional condition for {s.getpeername()}")
    # Stop listening for input on the connection
    inputs.remove(s)
    if s in outputs:
        outputs.remove(s)
    s.close()
```

Non-blocking client

To test the non-blocking TCP server, you can create a simple client that connects to the server, sends a message, and then waits to receive a response. Below is an example of a basic TCP client in Python that interacts with our non-blocking server.

```
import socket

HOST = '127.0.0.1' # The server's hostname or IP address
PORT = 65432       # The port used by the server

# Create a socket
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  
    # Connect to the server  
    s.connect((HOST, PORT))  
    print("Connected to server")  
  
    # Send data  
    message = 'Hello, server'.encode()  
    s.sendall(message)  
    print("Message sent to server")  
  
    # Wait for a response  
    data = s.recv(1024)  
    print("Received response from server")  
  
print(f"Received: {data.decode()}")
```

From:
<https://edu.iit.uni-miskolc.hu/> - Institute of Information Science - University of Miskolc

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:iss_t:python_example_for_blocking_and_non-blocking_socket

Last update: 2024/02/25 15:55

