Sockets Spoon Feeding: The Practical Way To Get Into Sockets

Python Mauritius UserGroup (pymug)

More info: mscc.mu/python-mauritius-usergroup-pymug/

pymug.com

Why	Where
codes	github.com/pymug
share events	twitter.com/pymugdotcom
ping professionals	linkedin.com/company/pymug
all info	pymug.com
tell friends by like	facebook.com/pymug

Abdur-Rahmaan Janhangeer
(Just a Python programmer)

twitter: @osdotsystem
github: github.com/abdur-rahmaanj

compileralchemy.com

Sockets Spoon Feeding: The Practical Way To Get Into Sockets

Spoon 1

IRC: A Boon To Play With, As Much As You Like

It's very simple

```
BOT_IRC_SERVER ="chat.freenode.net"
BOT_IRC_CHANNEL = "##bottestingmu"
#BOT_IRC_CHANNEL = "#python"
BOT_IRC_PORT = 6667
BOT_NICKNAME = "appinventormuBot"


import socket
irc = socket.socket()


irc.connect((BOT_IRC_SERVER, BOT_IRC_PORT))
irc.recv(4096)
```

Sending

```
irc.send(bytes('NICK ' + BOT_NICKNAME + '\r\n','utf8'  )  )
pingChecker(irc.recv(4096))
irc.send(bytes('USER appinventormuBot appinventormuBot appinventormuBot : appinventormuBot IRC\r\n','utf8'  )  )
pingChecker(irc.recv(4096))
irc.send(bytes('msg NickServ identify ' + BOT_PASSWORD + " \r\n"  ,'utf8')  )
pingChecker(irc.recv(4096))
irc.send(bytes('NICKSERV  identify ' + BOT_NICKNAME+' '+BOT_PASSWORD+ '\r\n','utf8'  )  )
pingChecker(irc.recv(4096))
time.sleep(3)
irc.send(bytes('JOIN ' + BOT_IRC_CHANNEL + '\r\n','utf8'  )  )
```

Keeping Alive

```
while 1:
        pass
        line = irc.recv(4096)
        print(line)
        pingChecker(line)
        if line.find(bytes('PRIVMSG' ,'utf8')) != -1 or line.find(bytes('NOTICE'  ,'utf8')) != -1 :
                messagechecker(line)
                target.write(str(line))
                target.flush()
```

Ping checker view

```
def pingChecker(pingLine):
    if pingLine.find(bytes('PING'  ,'utf8')) != -1:
        pingLine = pingLine.rstrip().split()
        if pingLine[0] == bytes("PING"  ,'utf8'):
            irc.send(bytes("PONG "  ,'utf8') + pingLine[1] + bytes("\r\n"  ,'utf8')  )
```

💡 Fact

> The default port for web communication is port 80
>
> Typing www.google.com is actually connecting to some IP with port 80

💡 Fact

> When a recv returns 0 bytes, it means the other side has closed (or is in the process of closing) the connection. You will not receive any more data on this connection. Ever. You may be able to send data successfully [1]

💡 Fact

> There is no EOT on a socket [1]

Spoon 2: The echo server

server

```
import socket

s = socket.socket()
print("Socket successfully created")
port = 12345
s.bind(("", port))
print("socket binded to %s" % (port))
s.listen(5)
print("socket is listening")
while True:
    c, addr = s.accept()
    print("Got connection from", addr, file=open('file.txt', 'a'))
    c.send(b"Thank you for connecting")
    c.close()
```

client

```
import socket
s = socket.socket()
port = 12345
s.connect(('127.0.0.1', port))
print(s.recv(1024) )
s.close()
```

Brief:

server -> waiter

listens to cusomer requests

say thank you and fetch whats needed

Common web servers

- web
- file
- email

different web servers means different protocols.

IRC is one protocol

💡 Fact

> The client application (your browser, for example) uses “client” sockets exclusively; the web server it’s talking to uses both “server” sockets and “client” sockets. [1]

🍭 Tip

You don't always need a client-server case to use the socket module

```
>>> import socket
>>> socket.gethostbyname('www.google.com')
'216.58.223.68'
```

🍭 Tip

Use `socket.shutdown()` before `socket.close()`

> close() releases the resource associated with a connection but does not necessarily close the connection immediately. If you want to close the connection in a timely fashion, call shutdown() before close() [2]

?

Many times you will see:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
>>> socket.AF_INET.value
2
>>> socket.SOCK_STREAM.value
1
```

> This address family provides interprocess communication between processes that run on the same system or on different systems.
>
> Addresses for AF_INET sockets are IP addresses and port numbers. You can specify an IP address for an AF_INET socket either as an IP address (such as 130.99.128.1) or in its 32–bit form (X'82638001'). [3]

Use AF_INET6 for IPV6, AF_UNSPEC for handling both [2]

SOCK_STREAM: TCP connection

SOCK_DGRAM: UDP connection [4]

Why with?

With is called context manager in Python and adds shutdown and closing codes automatically for you. It is used for pre and post processing.

help:

```
socket(family=-1, type=-1, proto=-1, fileno=None)
```

so socket can accept 3 params

Spoon 3: Terminal chat app

https://github.com/TiagoValdrich/python-socket-chat

super implementation

server

```
def server() -> None:
    '''
        Main process that receive client's connections and start a new thread
        to handle their messages
    '''

    LISTENING_PORT = 12000

    try:
        socket_instance = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        socket_instance.bind(('', LISTENING_PORT))
        socket_instance.listen(4)

        print('Server running!')

        while True:
            socket_connection, address = socket_instance.accept()
            connections.append(socket_connection)
            threading.Thread(target=handle_user_connection, args=[socket_connection, address]).start()

    except Exception as e:
        print(f'An error has occurred when instancing socket: {e}')
    finally:
        # In case of any problem we clean all connections and close the server connection
        if len(connections) > 0:
            for conn in connections:
                remove_connection(conn)

        socket_instance.close()
```

```
def remove_connection(conn: socket.socket) -> None:
    '''
        Remove specified connection from connections list
    '''

    if conn in connections:
        conn.close()
        connections.remove(conn)
```

```python
def broadcast(message: str, connection: socket.socket) -> None:
    '''
        Broadcast message to all users connected to the server
    '''
    for client_conn in connections:
        # Check if isn't the connection of who's send
        if client_conn != connection:
            try:
                # Sending message to client connection
                client_conn.send(message.encode())

                # if it fails, there is a chance of socket has died
                except Exception as e:
                    print('Error broadcasting message: {e}')
                    remove_connection(client_conn)
```

```python
def handle_user_connection(connection: socket.socket, address: str) -> None:
    '''
        Get user connection in order to keep receiving their messages and
        sent to others users/connections.
    '''
    while True:
        try:
            # Get client message
            msg = connection.recv(1024)

            # If no message is received, there is a chance that connection has ended
            # so in this case, we need to close connection and remove it from connections list.
            if msg:
                # Log message sent by user
                print(f'{address[0]}:{address[1]} - {msg.decode()}')

                # Build message format and broadcast to users connected on server
                msg_to_send = f'From {address[0]}:{address[1]} - {msg.decode()}'
                broadcast(msg_to_send, connection)

            # Close connection if no message was sent
            else:
                remove_connection(connection)
                break

        except Exception as e:
            print(f'Error to handle user connection: {e}')
            remove_connection(connection)
            break
```

Client

```
def client() -> None:
    '''
        Main process that start client connection to the server
        and handle it's input messages
    '''

    SERVER_ADDRESS = '127.0.0.1'
    SERVER_PORT = 12000

    try:
        # Instantiate socket and start connection with server
        socket_instance = socket.socket()
        socket_instance.connect((SERVER_ADDRESS, SERVER_PORT))
        # Create a thread in order to handle messages sent by server
        threading.Thread(target=handle_messages, args=[socket_instance]).start()
        print('Connected to chat!')

        # Read user's input until it quit from chat and close connection
        while True:
            msg = input('> ')
            if msg == 'quit':
                break
            socket_instance.send(msg.encode()) # utf8
        socket_instance.close()
    except Exception as e:
        print(f'Error connecting to server socket {e}')
        socket_instance.close()


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

```
import socket, threading

def handle_messages(connection: socket.socket):
    '''
        Receive messages sent by the server and display them to user
    '''

    while True:
        try:
            msg = connection.recv(1024)

            # If there is no message, there is a chance that connection has closed
            # so the connection will be closed and an error will be displayed.
            # If not, it will try to decode message in order to show to user.
            if msg:
                print(msg.decode())
            else:
                connection.close()
                break

        except Exception as e:
            print(f'Error handling message from server: {e}')
            connection.close()
            break
```

Now you can build a webserver [5]

Spoon 4: Web Server

The start of which is

```
# Python3.7+
import socket

HOST, PORT = '', 8888

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_socket.bind((HOST, PORT))
listen_socket.listen(1)
print(f'Serving HTTP on port {PORT} ...')
while True:
    client_connection, client_address = listen_socket.accept()
    request_data = client_connection.recv(1024)
    print(request_data.decode('utf-8'))

    http_response = b"""\
HTTP/1.1 200 OK

Hello, World!
"""
    client_connection.sendall(http_response)
    client_connection.close()
```

What your browser sends?

```
GET / HTTP/1.1
Host: 127.0.0.1:8888
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
90.0.4430.72 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-GPC: 1
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: session=.
eJwlzjtuAzEMBNC7qE5BUhQl-TIL8QcbARJg166C3D0yUg1mmjc_5cgzrnu5Pc9XfJTj4eVWJHRIZ3YTZB7h-M
7aGaThkpU-EEShj7T0JMWpKsF9prWqxNhVydZctCDIxHOO1CDnIRoTsHuiEcy2lTprZlibPnrNqiFlH3ldcf6_
wV3tOvN4fn_
G1x4AKhPnm5nBtdEgoS5bqI51gm1BoFv5_QPd5D5v.YGv4Yw.PvfkZjmFkD_57uFld1-8CaYaNio
```

🍭 Tip

> For making requests, just use the requests library

Terms:

- IPV4
- IPV6

Look into

- selectors
- async server approach

A most interesting piece of socket programming:

Phone app sends UDP, Machine Learning used.

https://github.com/YashIndane/Call-of-Duty-

Credits: see credits.txt in repo

Refs:

[1] https://docs.python.org/3/howto/sockets.html
[2] https://docs.python.org/3/library/socket.html
[3] https://www.ibm.com/docs/en/i/7.4?topic=family-af-inet-address
[4] https://www.ibm.com/docs/en/aix/7.2?topic=protocols-socket-types
[5] https://ruslanspivak.com/lsbaws-part1/