

Wasfi Momen

9/22/18

1. TCP Server (Multithreaded)

Multithreaded Feature

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/  
MasterFiles/Notes/CS-6220/HW2 (cs)  
$ python tcpserver.py  
SERVER READY @ ('127.0.0.1', 65434)  
Connected by: ('127.0.0.1', 52590)  
Connected by: ('127.0.0.1', 52591)
```

```
Sentence is: Hello, I am client 1.  
Number of strings 1
```

```
Sentence is: Hello, I am client 2.  
Number of strings 2
```

```
█
```

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/  
MasterFiles/Notes/CS-6220/HW2 (cs)  
$ python tcpclient.py
```

```
Input a lowercase sentence...  
Hello, I am client 1.
```

```
From Server: HELLO, I AM CLIENT 1.  
Awaiting next string...
```

```
Input a lowercase sentence...  
█
```

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/  
MasterFiles/Notes/CS-6220/HW2 (cs)  
$ python tcpclient.py
```

```
Input a lowercase sentence...  
Hello, I am client 2.
```

```
From Server: HELLO, I AM CLIENT 2.  
Awaiting next string...
```

```
Input a lowercase sentence...  
█
```

Client shutdown feature

(Note the CLIENT HAS REQUESTED SHUTDOWN message. Server will continue to run if more than two clients are connected. Exception is reported but does not halt execution.)

1 client scenario

```
MasterFiles/Notes/CS-6220/HW2 (cs)
$ python tcpserver.py
SERVER READY @ ('127.0.0.1', 65434)
Connected by: ('127.0.0.1', 52598)

CLIENT HAS REQUESTED SHUTDOWN
Traceback (most recent call last):
  File "tcpserver.py", line 168, in <module>
    TCPServer(host, port).start_client_listen
()
  File "tcpserver.py", line 140, in start_client_listen
    conn, addr = self.socket.accept()
  File "C:\Users\Wasfi\AppData\Local\Programs\Python\Python36\lib\socket.py", line 205, in accept
    fd, addr = self._accept()
OSError: [WinError 10038] An operation was attempted on something that is not a socket

Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
$

Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
$ python tcpclient.py

    Input a lowercase sentence...

Interrupted
```

2 client scenario (server continues to function)

```
ent_listen
    conn, addr = self.socket.accept()
    File "C:\Users\Wasfi\AppData\Local\Programs\Python\Python36\lib\socket.py", line 205, in accept
        fd, addr = self._accept()
OSError: [WinError 10038] An operation was attempted on something that is not a socket

Sentence is: hello
Number of strings 1

Sentence is: asdmlsad
Number of strings 2
[]

Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
$ python tcpclient.py

    Input a lowercase sentence...

Interrupted

Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
$

    Input a lowercase sentence...
hello

From Server: HELLO
Awaiting next string...

    Input a lowercase sentence...
asdmlsad

From Server: ASDMLSAD
Awaiting next string...

    Input a lowercase sentence...
```

Strings Limit Reached Feature

```
Number of strings 9

Sentence is: sad
Number of strings 10
STRINGS LIMIT REACHED, SHUTTING DOWN AND CLOSING SOCKET
Traceback (most recent call last):
  File "tcpserver.py", line 169, in <module>
    TCPServer(host, port).start_client_listen()
  File "tcpserver.py", line 141, in start_client_listen
    conn, addr = self.socket.accept()
  File "C:\Users\Wasfi\AppData\Local\Programs\Python\Python36\lib\socket.py", line 205, in accept
    fd, addr = self._accept()
OSError: [WinError 10038] An operation was attempted on something that is not a socket
```

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
```

```
$
```

```
Input a lowercase sentence...
```

```
dasd
```

```
From Server: DASD
```

```
Awaiting next string...
```

```
Input a lowercase sentence...
```

```
sad
```

```
From Server: SAD
```

```
Awaiting next string...
```

```
Input a lowercase sentence...
```

```
asd
```

```
From Server: STRINGS LIMIT REACHED, CLOSING CONNECTION
```

```
CLIENT CONNECTION CLOSED.
```

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/MasterFiles/Notes/CS-6220/HW2 (cs)
```

```
$ a
```

2. UDP Server

Check Balance Feature

```
TERMINAL  ...  1: python, pyth ▾  +  ≡  🗑️

Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/
MasterFiles/Notes/CS-6220/HW2 (cs)
$ python udpserver.py
SERVER READY @ ('127.0.0.1', 65434)
Connected by: ('127.0.0.1', 62151)
█

(1) Check Balance
(2) Withdraw
(3) Deposit

Input command by sending a number...
1
From Server:  0 dollar(s) are in your balance
.

(1) Check Balance
(2) Withdraw
(3) Deposit

Input command by sending a number...
█
```

Withdraw Feature

```
Wasfi@DESKTOP-EB3J9PF MINGW64 /d/ThingsOfAll/  
MasterFiles/Notes/CS-6220/HW2 (cs)  
$ python udpserver.py  
SERVER READY @ ('127.0.0.1', 65434)  
Connected by: ('127.0.0.1', 62151)  
█
```

(3) Deposit

Input command by sending a number...

2

How much do you want to withdraw?

23

From Server:

Please give the username and pin number.

Please give the username.

charles

Please give the pin number.

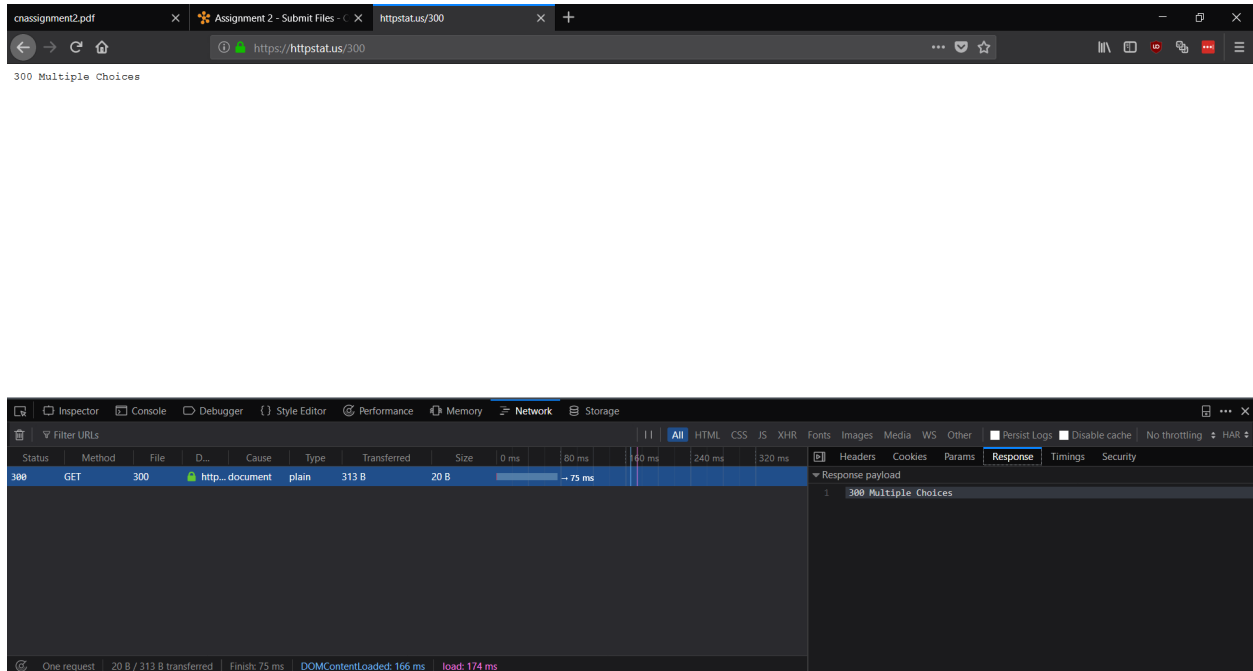
02323

From Server: Withdrew 23 dollar(s). Balance is now -23 dollar(s).

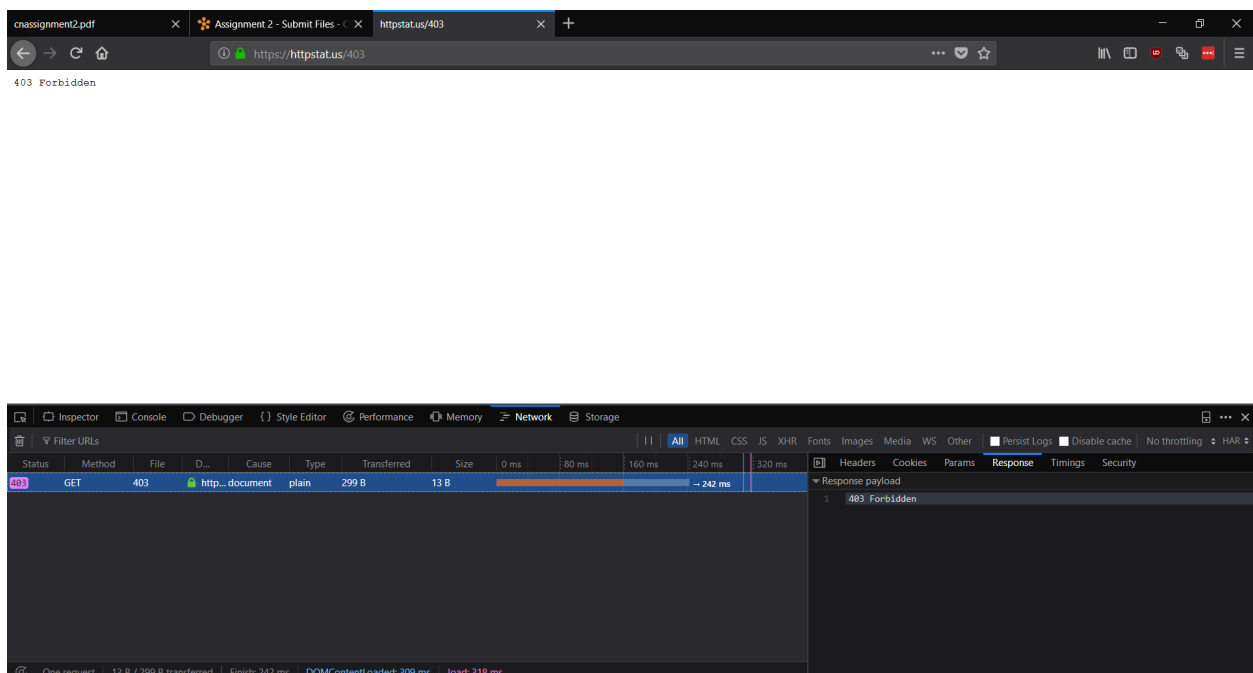
3. HTTP Responses

Went to <httpstat.us> and tried a couple of responses.

300 multiple choices



403 Forbidden



Also tried a GSU domain lklarich.gsucreate.org, but all it returned were 200s.

The screenshot shows a web browser window with the URL `lklarich.gsucreate.org`. The page displays the 'DREAM MACHINE' logo and navigation links: 'COLLEGE RESOURCES', 'COMMUNITY RESOURCES', and 'CONTACT US'. A banner image with the text 'Opening doors to the dreams of homeless youth!' is visible. The browser's developer tools are open, showing the Network tab. The following table represents the data from the Network tab:

Status	Method	File	D...	Cause	Type	Transferred	Size	0 ms	640 ms	128 s	192 s	256 s	320 s	34	Headers	Cookies	Params	Response	Timings
200	GET	dashicon...	lklar...	stylesheet	css	cached	45.27 KB												
200	GET	style.css7...	lklar...	stylesheet	css	0 GB	42.50 KB												
200	GET	cooperhe...	lklar...	stylesheet	css	0 GB	1.35 KB												
200	GET	generico...	lklar...	stylesheet	css	27.84 KB	27.60 KB												
200	GET	social-ic...	lklar...	stylesheet	css	1.63 KB	1.37 KB												
200	GET	jetpack.c...	lklar...	stylesheet	css	66.56 KB	66.32 KB												
200	GET	/custom...	lklar...	stylesheet	css	0 GB	3.03 KB												
200	GET	web4.jpg	lklar...	img	jpeg	49.32 KB	49.08 KB												
200	GET	wpgroho...	lklar...	script	js	1.27 KB	0.99 KB												
200	GET	navigatio...	lklar...	script	js	3.38 KB	3.13 KB												

The 'Response' tab for the 'jetpack.css' file shows the following CSS code:

```
1 /*!
2  * Do not modify this file directly. It is concatenated from individual modules.
3  */
4  .jp-carousel-wrap *{line-height:inherit}.jp-carousel-overlay{background:#000}
5  .contact-form .clear-form{clear:both}.contact-form input[type=submit],.contact-
6  .infinite-loader{color:#000;display:block;height:20px;text-indent:-999px}
7  #font-face{font-family:Noticons;src:url(https://wordpress.com/1/notifications/Notic
8  #jp-relatedposts{display:none;padding-top:1em;margin:1em 0;position:relative;
9  #jp-post-flair{padding-top:5em}#content div.sharedaddy,#main div.sharedaddy,d
10 .slideshow-window{background-color:#222;border:10px solid #222;border-radius:1
11 body.presentation-wrapper-fullscreen-parent,html.presentation-wrapper-fullscre
12 div.jetpack-quiz{border:1px solid #dee3;background-color:#f3f3f3;padding:1em
13 #subscribe-email input{width:95%}.comment_subscribe_form .subscribe-label{
14 .jetpack-video-wrapper{margin-bottom:1.6em}.jetpack-video-wrapper>video,.j
15 .jetpack-social-navigation ul{display:block;margin:0 0 1.5em;padding:0}.jetpac
16 tiled-gallery{clear:both;margin:0 0 20px;overflow:hidden}.tiled-gallery img{
```

CODE

```
"""
udpserver.py

A udp server made in python.
Some variable and function names from socketserver.py in python 3.7 release.
Made by Wasfi Momen.

"""

"""
REQUIRMENTS
    - ~~Maintains file with: name, pin, and balance~~
    - ~~Prompts client for auth using name and pin~~
    - ~~Allows client to deposit and withdraw AFTER auth~~
    - ~~Messages client confirmation of record update~~
    - ~~Allows client to ask for and receive current balance~~
    ALL COMPLETE
"""

import socket
import sys

class UDPServer:

    address_family = socket.AF_INET # only IPv4 connections
    socket_type = socket.SOCK_DGRAM # constant to use UDP socket type

    # to be put into a file
    balance = 0
    username = "charles"
    pin = "02323"

    client_details = [] # list to hold client tuples

    # constructor from the socket.py module
    def __init__(self, server_address, server_port):
        self.server_address = server_address
        self.server_port = server_port
        self.socket = socket.socket(self.address_family, self.socket_type)
```



```

    try:
        self.write_to_file() # initialize and write to file
        self.bind()
    except:
        self.socket.close()
        raise Exception('Error in setting up connection to host port.')

def add_client_details(self, addr):
    """
    Add the tuples of our connected clients to a list.
    This is used namely to connect back to clients for an auth check.
    See auth_challenge() for more info.
    """
    if addr in self.client_details:
        return
    self.client_details.append(addr)
    print("Connected by: ", addr)

def bind(self):
    """Binds the socket to any available port"""
    try:
        # a tuple of the address and port is used for the bind function
        self.socket.bind((self.server_address, self.server_port))
    except:
        self.socket.close()
        raise Exception(
            'Error in binding socket to address and port specified.')

def check_auth(self, username, pin):
    # forgot list comprehension, so we separate the first chars here
    username = username[1:]
    pin = pin[1:]

    if (username == self.username and pin == self.pin):
        return True
    else:
        return False

def auth_challenge(self):
    """
    Challenges the client to provide credentials.
    Credentials are returned as a tuple that we can pass to
    check_auth individually.

    Returns true if credentials are correct and false otherwise.

```

```

"""

message = "\n\tPlease give the username and pin number.\n"

# send a message back to the client asking for auth info
self.socket.sendto(message.encode(), (self.client_details[0]))

# initiate a new frame to get UDP data. (Probably can just do
self.socket.recvfrom(2048))
conn, addr = self.socket.recvfrom(2048)
credentials = conn.decode()
credentials = credentials.split()
credentials = [
    word for line in credentials for word in line.split()] # separate
the username and password strings

if (self.check_auth(credentials[0], credentials[1]) == True):
    return True
else:
    return False

def check_balance(self):
    return str(self.balance) + " dollar(s) are in your balance."

def deposit(self, amount):
    amount = amount[0:]
    self.balance = self.balance + int(amount)
    self.write_to_file()

def withdraw(self, amount):
    amount = amount[0:]
    self.balance = self.balance - int(amount)
    self.write_to_file()

def process_command(self, message):
    """
    Takes in a message of 3 formats -- either b, w{int number}, or d{int
number}
    The first character in the message string represent the check_balance(),
deposit(),
    and withdraw() functions, respectively.

    For withdraw and deposit, the auth_challenge() validates the user
permission to
    execute the functions.

```

```

        Returns strings to send to clients, either a confirmation or invalid
message, to
        be then sent back to the client.

        """
        if (len(message) >= 1):
            if (message[0] == "b"):
                return self.check_balance()
            elif (message[0] == "w"):
                if (self.auth_challenge() != False):
                    self.withdraw(message[1:])
                    return "Withdrew " + message[1:] + " dollar(s). Balance is
now " + str(self.balance) + " dollar(s)."
                else:
                    return "Invalid credentials. Please try the withdraw command
again."
            elif (message[0] == "d"):
                if (self.auth_challenge() != False):
                    self.deposit(message[1:])
                    return "Deposited " + message[1:] + " dollar(s). Balance is
now " + str(self.balance) + " dollar(s)."
                else:
                    return "Invalid credentials. Please try the deposit command
again."
            else:
                return "Improper format, please send command again."

    def print_details(self):
        """Print out the server details"""
        print('SERVER READY @ ', self.socket.getsockname())

    def write_to_file(self):
        """
        Writes to the user.txt file. This function is called
        once every time a UDPServer is instantiated.

        """
        string = str(self.balance) + "\n" + \
            self.username + "\n" + self.pin + "\n"
        file = open("user.txt", "w")

        try:
            file.write(string)
            file.close()

```

```
except Exception as e:
    file.close() # just in case
    sys.exit(1) # to force stop execution
    print(e)

host = "127.0.0.1"
# port = 0 # the OS should choose an open port for us
port = 65434

def main():
    try:
        server_sock = UDPServer(host, port)
        server_sock.socket.setblocking(1)
        server_sock.print_details()
        while True:
            conn, addr = server_sock.socket.recvfrom(2048)
            server_sock.add_client_details(addr)
            message = conn.decode()
            message = server_sock.process_command(message)
            server_sock.socket.sendto(message.encode(), addr)

        server_sock.socket.close()
    except KeyboardInterrupt:
        print("\nExited by Ctrl+C.")
        server_sock.socket.close()
        sys.exit(1)

main()
```

```

"""
udpclient.py

A udp client made in python.
Some variable and function names from socketserver.py in python 3.7 release.
Made by Wasfi Momen.

"""

import socket
import sys

class UDPClient:

    address_family = socket.AF_INET # only IPv4 connections
    socket_type = socket.SOCK_DGRAM # constant to use TCP socket type

    # constructor from the socket.py module
    def __init__(self, server_address, server_port):
        self.server_address = server_address
        self.server_port = server_port
        self.socket = socket.socket(self.address_family, self.socket_type)

    def connect(self):
        """Attempts to connect to the server socket, throws exception and closes
        socket if fails."""
        try:
            self.socket.connect((self.server_address, self.server_port))
        except:
            self.socket.close()
            raise Exception('Failed to connect to server socket.')

    def process_command(self, number):
        """Formats a message to be received and interpreted server-side."""
        amount = 0

        if (number == "1"):
            return "b"
        elif (number == "2"):
            amount = input("\n\tHow much do you want to withdraw?\n")
            return ("w" + amount)
        elif (number == "3"):
            amount = input("\n\tHow much do you want to deposit?\n")

```

```

        return ("d" + amount)
    else:
        return "\nIllegal Command."

host = '127.0.0.1'
# port = 0 # the OS should choose an open port for us
port = 65434

def main():
    try:
        client_sock = UDPClient(host, port)
        client_sock.connect()
        while True: # from python 3.7 docs examples
            print("\n")
            print("\t(1) Check Balance\n")
            print("\t(2) Withdraw\n")
            print("\t(3) Deposit\n")
            command = input("\n\tInput command by sending a number...\n")

            # get a message to send to the server
            message = client_sock.process_command(command)

            if (message != "\nIllegal Command."):
                # send message to server.
                client_sock.socket.sendto(
                    message.encode(), (host, port))

            conn, addr = client_sock.socket.recvfrom(2048)
            message_from_server = conn.decode()
            print("\nFrom Server: ", message_from_server)

            if (message_from_server == "\n\tPlease give the username and pin
number.\n"):
                username = input("\n\tPlease give the username.\n")
                pin = input("\n\tPlease give the pin number.\n")
                formatted = "u" + username + " " + "p" + pin
                client_sock.socket.sendto(formatted.encode(), (host, port))
                # we make another conn, addr here to get another frame of the UDP
                packet.
                # dunno how to fix except keeping track of packet data sent and
                received.

                conn, addr = client_sock.socket.recvfrom(2048)
                message_from_server = conn.decode()

```

```
        print("\nFrom Server: ", message_from_server)

    client_sock.socket.close()
except KeyboardInterrupt:
    print("Interrupted")
    client_sock.socket.close()
    sys.exit(1)

main()
```

```

"""

tcpclient.py

A tcp client made in python.
Some variable and function names from socketserver.py in python 3.7 release.
Made by Wasfi Momen.

"""

import socket
import sys

class TCPClient:

    """TCPClient class that only binds the
    socket using the specified ports in the
    constructor.

    All other socket operations should
    be used by the .socket member access.

    """

    address_family = socket.AF_INET # only IPv4 connections
    socket_type = socket.SOCK_STREAM # constant to use TCP socket type

    # constructor from the socket.py module
    def __init__(self, server_address, server_port):
        self.server_address = server_address
        self.server_port = server_port
        self.socket = socket.socket(self.address_family, self.socket_type)

    def connect(self):
        """Attempts to connect to the server socket, throws exception and closes
        socket if fails."""
        try:
            self.socket.connect((self.server_address, self.server_port))
        except:
            self.socket.close()
            raise Exception('Failed to connect to server socket.')

host = '127.0.0.1'

```



```
# port = 0 # the OS should choose an open port for us
port = 65434

def main():
    try:
        client_sock = TCPClient(host, port)
        client_sock.connect()
        while True: # from python 3.7 docs examples
            sentence = input("\n\tInput a lowercase sentence...\n")
            client_sock.socket.sendall(sentence.encode())
            data = client_sock.socket.recv(1024)
            print("\nFrom Server: ", data.decode())
            # server has terminated connection, clean up and close socket
            if (data == "STRINGS LIMIT REACHED, CLOSING CONNECTION".encode()):
                print("\n\tCLIENT CONNECTION CLOSED.")
                client_sock.socket.shutdown(2)
                break
        client_sock.socket.close()
    except KeyboardInterrupt:
        print("Interrupted")
        client_sock.socket.sendall("CLIENT REQUESTS SHUTDOWN".encode())
        client_sock.socket.shutdown(2)
        client_sock.socket.close()
        sys.exit(1)

main()
```

```

"""

tcpserver.py

A tcp server made in python.
Some variable and function names from socketserver.py in python 3.7 release.
Made by Wasfi Momen.

"""

"""
REQUIRMENTS
    - ~~Capitalizes strings~~
    - ~~Max 10 strings~~
    - ~~Client ask for termination~~ **"Appropriate message" to terminate is
Ctrl+C**
    - ~~Limit reached, send message to client~~
    - ~~Multithreaded~~
    ALL COMPLETE

"""

"""
ISSUES
    - Non-graceful shutdown of more than two clients. Exception will be raised
    but server will continue to function till the last client disconnects.
    - KeyboardInterrupt not working correctly.
"""

import socket
import sys
import threading

class TCPServer:

    """
    TCPServer class contains the necessary functions
    for the application by keeping track of the strings
    and only binding the socket to the specified ports in
    the constructor.

    All other socket operations should
    be used by the .socket member access.
    """

```

```

address_family = socket.AF_INET # only IPv4 connections
socket_type = socket.SOCK_STREAM # constant to use TCP socket type

# request_queue_size = 1 # we only take one connection

num_strings = 0 # number of strings received
client_details = [] # list of clients connected

# constructor from the socket.py module
def __init__(self, server_address, server_port):
    self.server_address = server_address
    self.server_port = server_port
    self.socket = socket.socket(self.address_family, self.socket_type)

    try:
        self.bind()
    except:
        self.socket.close()
        raise Exception('Error in setting up connection to host port.')

def add_client_details(self, addr):
    """
    Add the tuples of our connected clients to a list.
    This is not currently used, but would be great to
    gracefully except during shutdowns of client sockets.
    """
    if addr in self.client_details:
        return
    self.client_details.append(addr)
    print("Connected by: ", addr)

def capitalize_string(self, client_string):
    """Capitalizes string that is received from the client."""
    self.receive_string()
    return client_string.upper()

def bind(self):
    """Binds the socket to a port"""
    try:
        # option to reuse addresses for sockets
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # a tuple of the address and port is used for the bind function
        self.socket.bind((self.server_address, self.server_port))
    except:

```

```

        self.socket.close()
        raise Exception(
            'Error in binding socket to address and port specified.')

def process_clients(self, connection, address):
    """
    Main logic of the program. Takes care of the shutdown of
    client sockets and termination of server when strings limit
    is reached.
    """
    while True:
        sentence = connection.recv(1024).decode()

        # break out of while loop if shutdown request is given
        if (sentence == "CLIENT REQUESTS SHUTDOWN"):
            print("\nCLIENT HAS REQUESTED SHUTDOWN")
            # 2 means SHUT_RDWR or disable read and write, Windows only takes
            numbers
            connection.shutdown(2)
            connection.close()
            break

        # break out of while loop if number of strings exceeds 10
        if (self.get_num_of_strings() >= 10):
            print(
                "STRINGS LIMIT REACHED, SHUTTING DOWN AND CLOSING SOCKET")
            connection.sendall(
                "STRINGS LIMIT REACHED, CLOSING CONNECTION".encode())
            connection.shutdown(2)
            connection.close()
            break

        print("\nSentence is: ", sentence)
        connection.sendall(
            self.capitalize_string(sentence).encode() + "\nAwaiting next
string...".encode())
        print("Number of strings ",
            self.get_num_of_strings())
        self.socket.close()

def start_client_listen(self):
    """
    Starts the multithreaded listen for the
    clients. Also adds clients to a list of
    connected clients.

```

```

Every client added will go through
here to be processed by process_clients()
"""
try:
    self.socket.listen(4) # listen to 4 clients max
    self.print_details()
    while True:
        conn, addr = self.socket.accept()
        self.add_client_details(addr)
        threading.Thread(target=self.process_clients,
                          args=(conn, addr)).start()

    self.socket.close()
except KeyboardInterrupt:
    print("\nExited by Ctrl+C.")
    self.socket.close()
    sys.exit(1)

def print_details(self):
    """Print out the server details"""
    print('SERVER READY @ ', self.socket.getsockname())

def received_string(self):
    """Increments the number of string received by the server."""
    self.num_strings += 1

def get_num_of_strings(self):
    """Returns the number of strings the server has received."""
    return self.num_strings

host = "127.0.0.1"
# port = 0 # the OS should choose an open port for us
port = 65434

TCPServer(host, port).start_client_listen()

```