

CSC 645-02 – Computer Network

Spring 2024

Project #2

Network Programming

with UDP and TCP in Python

Nhan Nguyen

923100929

[GitHub repository](#)

1. Introduction

The purpose of the application is to demonstrate basic network programming concepts, focusing on **UDP/TCP** for client-server communication where the client sends a sentence to the server, and the server returns the sentence with the order of the words reversed.

Python is selected due to its extensive standard library support for networking, ease of use, readability, and widespread popularity in educational contexts. Python's socket module provides comprehensive functionalities for socket programming, suitable for both learning and practical implementations.

Utilizing simple **client-server architecture**, the server listens for messages from any client, processes them, and sends a response. The client sends messages to the server and displays the responses it receives.

A utility function **reverse_sentence** was implemented to reverse the order of words in a sentence, showcasing basic string manipulation alongside network programming.

2. UDP Socket Programming

UDP is connectionless, allows quick data transmission without the overhead of establishing and maintaining a connection, making it suitable for applications where speed is more important than reliability.

Here are the codes:

- **udpclient.py**

```
from socket import *

# serverName = '10.143.217.160'
# serverName = '10.143.15.196' # school IP address
serverName = '192.168.56.1' # home IP address

serverPort = 12000

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = None

while message != '!q': # Loop until the user types '!q'
    message = input('Input a sentence you want to reverse: ') #
    Get user input
    if message == '!q':
        break # Exit the loop before sending '!q' to the server
    clientSocket.sendto(message.encode(), (serverName,
serverPort))

    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
    print("Message was received and reversed to: " +
modifiedMessage.decode()) # Print the reversed message received
from the server

clientSocket.close()
```

- **udpserver.py**

```
from socket import *

def reverse_sentence(sentence):
    words = sentence.split()
    reversed_sentence = ' '.join(reversed(words))
    return reversed_sentence

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print("The server is ready to receive!")

while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    message = message.decode() # Decode received message
    modifiedMessage = reverse_sentence(message) # Reverse
sentence
    print('Received "' + message + '" and reversed to: "' +
modifiedMessage + '"')
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

- Output

```

C:\Users\MinhNhan\Documents\python_env\Scripts\python.exe "G:/My Drive/0_SFSU
Study/02_CSC 645 - Computer Networks/project#2/socket-programming/udpserver.py"
The server is ready to receive!
Received "a b c d 1" and reversed to: "1 d c b a"
Received "1 2 3 4 5" and reversed to: "5 4 3 2 1"

```

```

C:\Users\MinhNhan\Documents\python_env\Scripts\python.exe "G:/My Drive/0_SFSU
Study/02_CSC 645 - Computer Networks/project#2/socket-programming/udpclient.py"
Input a sentence you want to reverse: a b c d 1
Message was received and reversed to: 1 d c b a
Input a sentence you want to reverse: 1 2 3 4 5
Message was received and reversed to: 5 4 3 2 1
Input a sentence you want to reverse:

```

3. TCP Socket Programming

TCP is reliable, ensures that all sent packets arrive in order without errors. TCP's connection-oriented nature makes it suitable for applications where data integrity and order are critical, such as text processing.

Here are the codes:

- tcpclient.py

```

from socket import *

# serverName = '10.143.15.196' # school IP address
serverName = '192.168.56.1' # home IP address

```

```

serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

sentence = input('Input a sentence you want to reverse: ') #
Get user input
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)

print("Message was received and reversed to: " +
modifiedSentence.decode()) # Print the reversed message
received from the server
clientSocket.close()

```

- tcpserver.py

```

from socket import *

def reverse_sentence(sentence):
    words = sentence.split()
    reversed_sentence = ' '.join(reversed(words))
    return reversed_sentence

serverPort = 12000

```

```

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen(1)

print('The server is ready to receive!')

while True:

    connectionSocket, addr = serverSocket.accept()

    message = connectionSocket.recv(1024).decode()

    modifiedMessage = reverse_sentence(message)

    print('Received "' + message + '" and reversed to: "' +
modifiedMessage + '"')

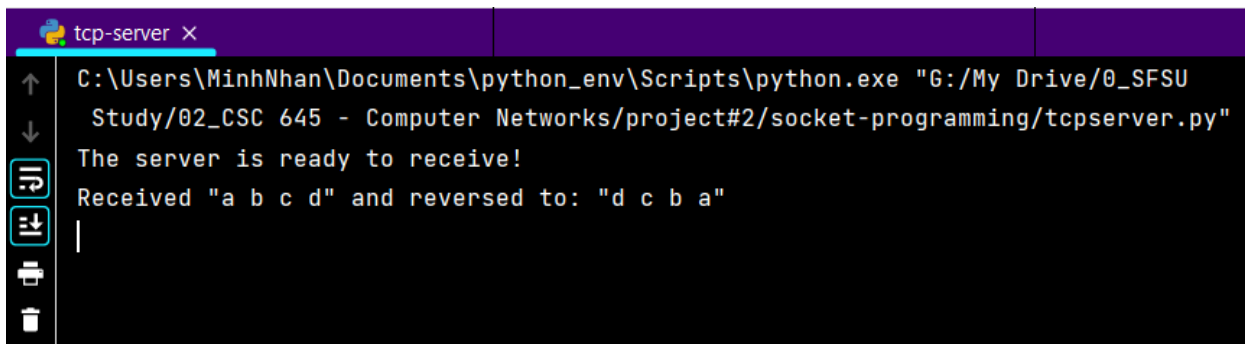
    connectionSocket.send(modifiedMessage.encode())

    connectionSocket.close()

```

Here, **serverSocket.listen(1)** is used to allow the server to manage one connection at a time for simplicity. Also, incorporated **connectionSocket.close()** within the loop to ensure that each connection is cleanly closed after handling.

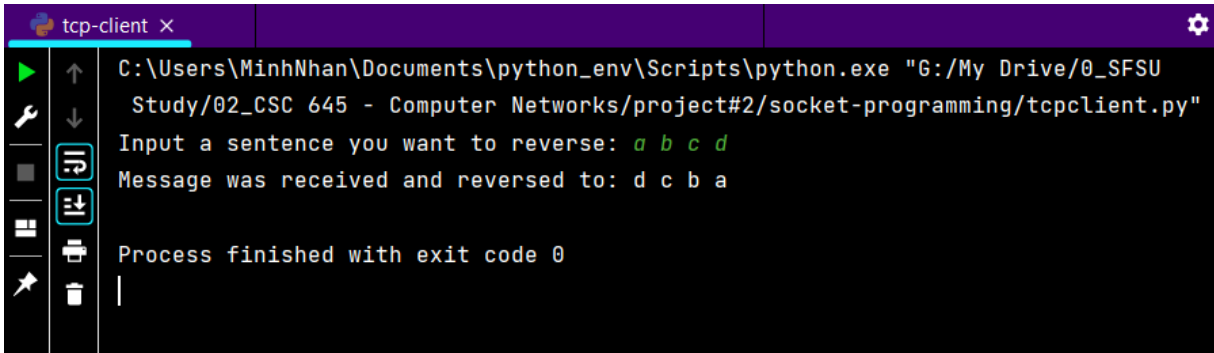
- **Output:**



```

tcp-server X
C:\Users\MinhNhan\Documents\python_env\Scripts\python.exe "G:/My Drive/0_SFSU
Study/02_CSC 645 - Computer Networks/project#2/socket-programming/tcpserver.py"
The server is ready to receive!
Received "a b c d" and reversed to: "d c b a"
|

```



```
tcp-client X
C:\Users\MinhNhan\Documents\python_env\Scripts\python.exe "G:/My Drive/0_SFSU
Study/02_CSC 645 - Computer Networks/project#2/socket-programming/tcpclient.py"
Input a sentence you want to reverse: a b c d
Message was received and reversed to: d c b a
Process finished with exit code 0
```

4. Challenges encountered

Challenge: I went to this error: *"TypeError: sequence item 0: expected str instance, bytes found"*. This is due to incorrect handling of data types during transmission. Data sent through sockets needs to be in bytes, whereas the application logic works with strings.

Solution:

Decoding the Message: After receiving the message with `recvfrom()`, decode the bytes into a string using `.decode()`. This converts the byte data into a format that can be processed by `reverse_sentence` function.

Encoding the Response: Before sending the response back with `sendto()`, encode the reversed string back into bytes using `.encode()`. This is necessary because network communication via sockets is done with bytes, not strings.