



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2 : Introduction to Client-Server

Communication using Socket Programming — Simulating
an ATM Machine Operation

Submitted By:

Name: Md Shamsur Rahman Sami

Roll No : 57

Name: Md Rakib Hossain

Roll No : 55

Submitted On :

January 26, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Contents

1	Introduction	2
1.1	Objectives	2
2	Theory	3
2.1	Server	3
2.2	Client	3
3	Methodology	3
3.1	Creating a Socket	3
3.2	Connecting to a Server	3
3.3	Sending and Receiving Data	4
3.4	Closing the Connection	4
3.5	Idempotent Operation	4
3.6	Exactly-once Semantics	4
4	Experimental Result	4
4.1	Establish a TCP connection in between a server process, running on host A and a client process, running on host B and then perform some operation by the server process requested by the client and send responses from the server.	4
4.2	Send an integer and operation name (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not.	7
4.3	Using the above connection, design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.	10
4.4	Error Handelling	15
5	Experience	19

1 Introduction

1.1 Objectives

Creating TCP Connections using Socket Programming

- Socket programming can be used to establish a TCP connection between a client process running on host B and a server process running on host A.
 - For a line of text, the client process can send a request including a tiny letter to the server process, which will then transform all small letters to capital letters and return the modified text back to the client process.
 - Finding out if a number is palindrome or not: The client process can send a request including a number to the server process, which will determine if the number is palindrome or not and reply to the client process with the outcome.
- It is possible to implement a non-idempotent operation utilizing exactly-once semantics to handle failures of request and response messages as well as failures of process execution.
 - It is possible to create an application-level protocol that enables the verification of a user's card and password, the checking of their account balance, and the making of account withdrawals. Messages like:
 - * Request for verification of card , password
 - * Request for user account balance
 - * Request for user withdrawal
 - * Confirmation of user withdrawal
 - * Error message if there is not enough money in the account of user
 - To handle errors related to both request and response messages, the protocol can include a mechanism for resending messages in case of failures and an acknowledgment system to confirm.

2 Theory

Socket programming is a method of inter-process communication (IPC) that allows processes to communicate with each other across a network using sockets. A socket is an endpoint for sending or receiving data across a computer network. It is a combination of an IP address and a port number. In socket programming, a client creates a socket and connects to a server using the server's IP address and port number. The server then creates a socket and binds it to a specific IP address and port number. Once the connection is established, the client and server can send and receive data through the socket.

2.1 Server

When we started server side program , it will wait for client requests. If it gets any request then it will establish a connection.

2.2 Client

From client side , We will enter IP address of our server and the port number. Then the client connection request will be sent from our client to the server.

3 Methodology

Socket programming allows the creation of TCP (Transmission Control Protocol) connections through a series of phases that include socket creation, server connection, and data sending and receiving. It gets an HTTP query indicating the file the client requested after establishing the connection. After that, it will send the client the bytes it read from that file.

3.1 Creating a Socket

First we create a socket using the `socket()` function. This function takes three parameters: the address family, the type of socket, and the protocol.

3.2 Connecting to a Server

After creating the socket is created, the next step is to connect to a server using the `connect()` function. This function takes three parameters: the socket descriptor, the server's IP address, and the server's port number.

3.3 Sending and Receiving Data

Once the connection is established, the client and server can send and receive data through the socket.

3.4 Closing the Connection

When data exchange is complete, the connection can be closed using the `close()` function. This function takes one argument: the socket descriptor.

3.5 Idempotent Operation

In computing, an idempotent operation is one that has no additional effect if it is called more than once with the same input parameters. For example, removing an item from a set can be considered an idempotent operation on the set.

3.6 Exactly-once Semantics

As its name suggests, exactly-once semantics means that each message is delivered precisely once. The message can neither be lost nor delivered twice (or more times). Exactly-once is by far the most dependable message delivery guarantee.

4 Experimental Result

4.1 Establish a TCP connection in between a server process, running on host A and a client process, running on host B and then perform some operation by the server process requested by the client and send responses from the server.

- **Small Letter to Capital Conversion for a Line of Text:**

Here first we create a connection with client and server and then send a small letter text from client to the server, server accepts the client request and converts it into big letter and sends the response to the client.

```
1      SERVER SIDE CODE
2
3  import socket
```

```

4
5 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 s.bind((socket.gethostname(), 1234))
7 s.listen(5)
8 print(socket.gethostname())
9
10 while True:
11     try:
12         clientsocket, address = s.accept()
13         print(f"Connection from {address} has been established")
14         # clientsocket.send(bytes("Welcome ", "utf-8"))
15         msgg= clientsocket.recv(1024)
16         sami=msgg.decode("utf-7")
17         samiii =sami.lower()
18         print("I am receiving the message form client= " +sami)
19         clientsocket.send(bytes(samiii, "utf-8"))
20
21         print(sami)
22
23     except Exception as e:
24         print(f"Error: {e}")
25
26
27     clientsocket.close()
28
29
30 CLIENT SIDE CODE
31
32 import socket
33
34 #socket first
35 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
36 s.connect(("192.168.0.6", 5000))
37 message = "HELLO SERVER"
38 s.send(message.encode('utf-7'))
39
40
41 msg= s.recv(1024)
42 print("Sending data from client :"+ message)
43
44 print("After receiving Data Fron Server : "+msg.decode("utf-8"))
45 s.close()

```

```

1  import socket
2
3  #socket first
4  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
5  s.connect(("192.168.8.10", 5000))
6  message = "HELLO Server!"
7  s.send(message.encode('utf-8'))
8
9  msg = s.recv(1024)
10 print("receiving data from client : "+ message)
11
12 #client side
13 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
14 s.connect(("192.168.8.10", 5000))
15 message = "HELLO Server!"
16 s.send(message.encode('utf-8'))
17

```

Figure 1: Client side code

```

1  import socket
2
3  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4  s.bind((socket.gethostname(), 1234))
5  s.listen(5)
6  print(socket.gethostname())
7
8  while True:
9      # 1st
10     clientsocket, address = s.accept()
11     print(f"Connection from {address} has been established")
12     # clientsocket.send(bytes("Welcome - ", "utf-8"))
13     msg = clientsocket.recv(1024)
14     smsg=msg.decode('utf-8')
15     smsg=smsg.lower()
16     print(f"receiving the message from client- " + smsg)
17     clientsocket.send(bytes(smsg, "utf-8"))
18
19     #check palindrom(smsg)
20     #clientsocket.send(bytes(check_palindrom(smsg), "utf-8"))
21     smsg=smsg
22     # print(smsg)
23     print(smsg)
24
25 except Exception as e:
26     print("Error : "+e)
27
28 clientsocket.close()
29
30
31
32

```

Figure 2: Server Side Code

```
C:\Windows\System32\cmd.exe
C:\Python\Python\NetworkLab\Lab 2\Problem A\capital letter to small letter.py -> client.py
Sending data from client: HELLO SERVER
After receiving data from server: Hello server
C:\Python\Python\NetworkLab\Lab 2\Problem A\capital letter to small letter.py
```

Figure 3: Client Side result

```
C:\Windows\System32\cmd.exe python -> server.py
Microsoft Windows [Version 10.0.19041.50]
(c) Microsoft Corporation. All rights reserved.

C:\Python\Python\NetworkLab\Lab 2\Problem A\capital letter to small letter.py -> server.py
and
Connection from ("192.168.8.1", 12297) has been established
I am receiving the message from client: HELLO SERVER
HELLO SERVER
```

Figure 4: Server Side result

4.2 Send an integer and operation name (either ‘prime’ or ‘palindrome’) to the server and check whether it’s a prime (or palindrome) or not.

```
1 SERVER SIDE CODE
2 import socket
3
4 def is_prime(n):
5     if n <= 1:
```



```

6         return False
7     elif n <= 3:
8         return True
9     elif n % 2 == 0 or n % 3 == 0:
10        return False
11    i = 5
12    while i * i <= n:
13        if n % i == 0 or n % (i + 2) == 0:
14            return False
15        i += 6
16    return True
17
18    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19    s.bind((socket.gethostname(), 1234))
20    s.listen(5)
21
22    while True:
23        connection, address = s.accept()
24        print(f"Connection from {address} has been established")
25        try:
26            data = connection.recv(1024)
27            data = int(data.decode())
28            success = is_prime(data)
29            if success:
30                connection.sendall("It is a prime number".encode())
31            else:
32                connection.sendall("It is not a prime number".encode())
33        finally:
34            connection.close()
35
36
37    CLIENT SIDE CODE
38    import socket
39
40    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
41
42    s.connect(("Sami", 1234))
43
44    try:
45        request = int(input("Enter a number: "))

```

```

10
11     s.sendall(str(request).encode())
12     response = s.recv(1024).decode()
13     print("The number is ",request," "+response)
14
15 finally:
16     s.close()

```

•

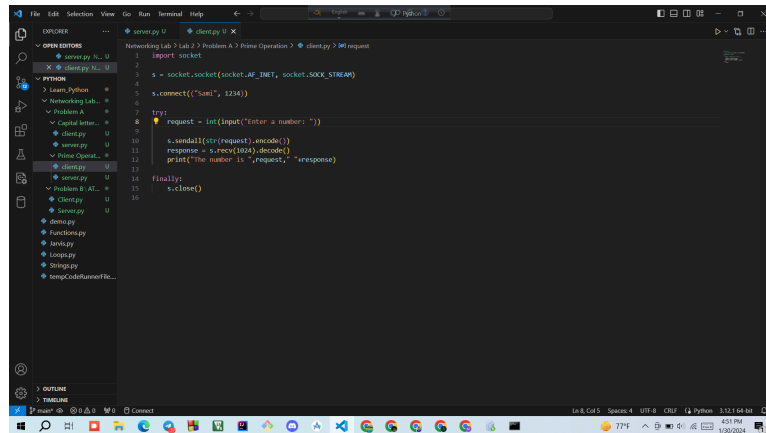


Figure 5: Client side code

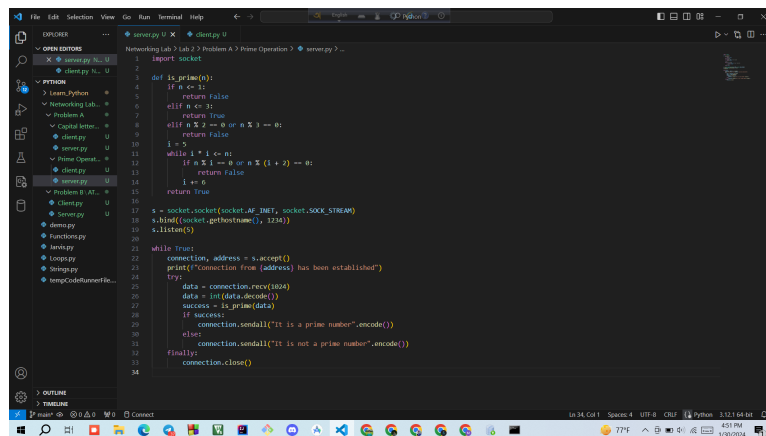


Figure 6: Server Side Code

```

C:\Windows\System32\cmd.exe
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
It is a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
It is not a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
The number is: 8 It is not a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
Enter a number: 1
The number is: 1 It is not a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
Enter a number: 12
The number is: 12 It is not a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
Enter a number: 17
The number is: 17 It is a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
Enter a number: 5
The number is: 5 It is a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o client.py
Enter a number: 2
The number is: 2 It is a prime number
Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operation

```

Figure 7: Client Side result

```

C:\Windows\System32\cmd.exe python -o server.py
Microsoft Windows [Version 10.0.19041.1]
(c) Microsoft Corporation. All rights reserved.

Python\PythonNetworkLib\LabLib\2\Primaline AP\Prime Operationpython -o server.py
connection from ("192.168.8.1", 12048) has been established
connection from ("192.168.8.1", 12088) has been established
connection from ("192.168.8.1", 12091) has been established
connection from ("192.168.8.1", 12019) has been established
connection from ("192.168.8.1", 12026) has been established
connection from ("192.168.8.1", 12021) has been established
connection from ("192.168.8.1", 12025) has been established

```

Figure 8: Server Side result

4.3 Using the above connection, design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.

- Enhance the above protocol so that it can handle errors related to both request and response messages to and from the server.

```

1     SERVER SIDE CODE
2     import socket
3
4
5     accounts = {
6         "123456789": {"password": "1234", "balance": 1000}
7     }
8
9     def process_withdrawal(card_number, password, amount):
10         if card_number in accounts and accounts[card_number]["password"] == password:
11             if accounts[card_number]["balance"] >= amount:
12                 accounts[card_number]["balance"] -= amount
13                 return True, accounts[card_number]["balance"]
14             else:
15                 return False, "Insufficient funds"
16         else:
17             return False, "Invalid card number or password"
18
19
20
21
22     s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
23     s.bind((socket.gethostname(),1234))
24     s.listen(5)
25
26
27     while True:
28         connection,address = s.accept()
29         print(f"Connection from {address} has been established")
30
31         try:
32
33             # Receive the data from the ATM
34             data = connection.recv(1024)
35             data = data.decode().split(',')
36
37             if len(data) == 4:
38                 # Process the received data
39                 card_number, password, operation, amount = data
40

```

```

41         if operation == "WITHDRAW":
42             success, response = process_withdrawal(card_number, password, in
43             if success:
44                 # Send success response to ATM
45                 connection.sendall(f"SUCCESS,{response}".encode())
46             else:
47                 # Send failure response to ATM
48                 connection.sendall(f"FAILURE,{response}".encode())
49         else:
50             connection.sendall("FAILURE,Invalid operation".encode())
51     else:
52         connection.sendall("FAILURE,Invalid request format".encode())
53
54     finally:
55         # Clean up the connection
56         connection.close()

1     Client Side Code
2     import socket
3
4     # Function to perform withdrawal
5     def withdraw(client_socket, card_number, password, amount):
6         request = f"{card_number},{password},WITHDRAW,{amount}"
7         client_socket.sendall(request.encode())
8         response = client_socket.recv(1024).decode()
9         return response.split(',')
10
11     # Create a TCP/IP socket
12     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13
14     client_socket.connect((socket.gethostname(), 1234))
15     print(socket.gethostname())
16
17     try:
18         # Example withdrawal request
19         card_number = "123456789"
20         password = "1234"
21         amount = 500
22
23         # Send withdrawal request to the server

```

```

24     response_code, message = withdraw(client_socket, card_number, password, amount)
25
26     # Process the response
27     if response_code == "SUCCESS":
28         print(f"Withdrawal successful. Updated balance: {message}")
29     else:
30         print(f"Withdrawal failed: {message}")
31
32 finally:
33     # Clean up the connection
34     client_socket.close()

```

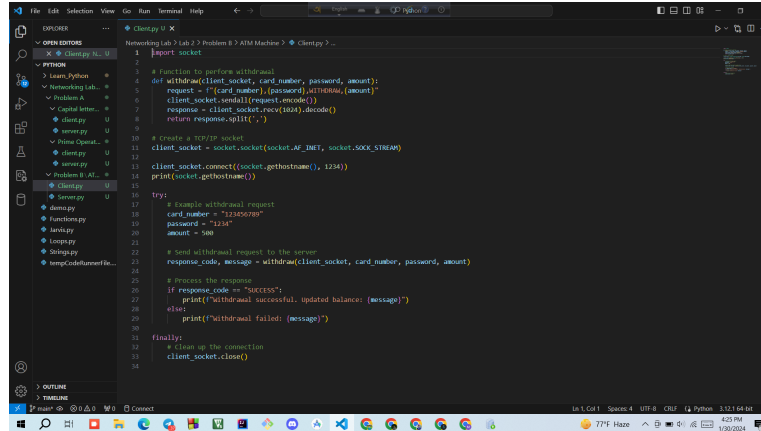


Figure 9: Client side code

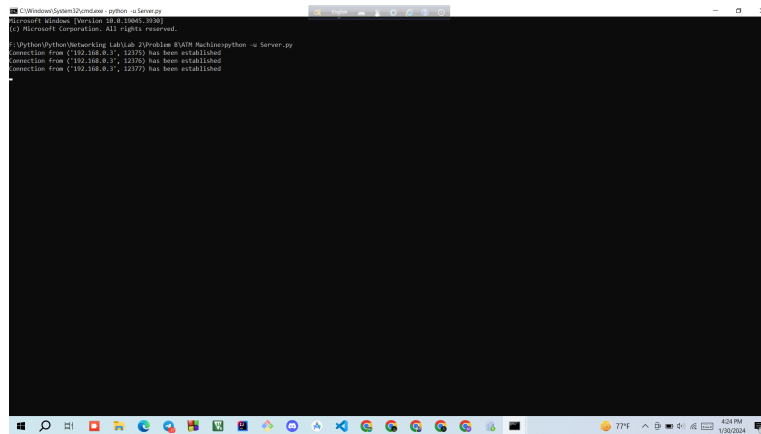


Figure 12: Server Side result

4.4 Error Handling

```

1      SERVER SIDE CODE
2
3  import socket
4
5  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6  s.bind((socket.gethostname(), 1234))
7  s.listen(5)
8  count=0
9  strr=""
10
11 while True:
12     try:
13         clientsocket, address = s.accept()
14         print(f"Connection from {address} has been established")
15
16         msgg= clientsocket.recv(1024)
17         x=(msgg.decode())
18
19
20
21         strr=strr+x
22         print(strr)
23         #if x>=3:

```



```

24         clientsocket.sendall(strrr.encode())
25
26         #else:
27         #     clientsocket.sendall("Error".encode())
28
29
30
31     except Exception as e:
32         print(f"Error: {e}")
33
34
35     clientsocket.close()
36
37
38

```

1 CLIENT SIDE CODE

```

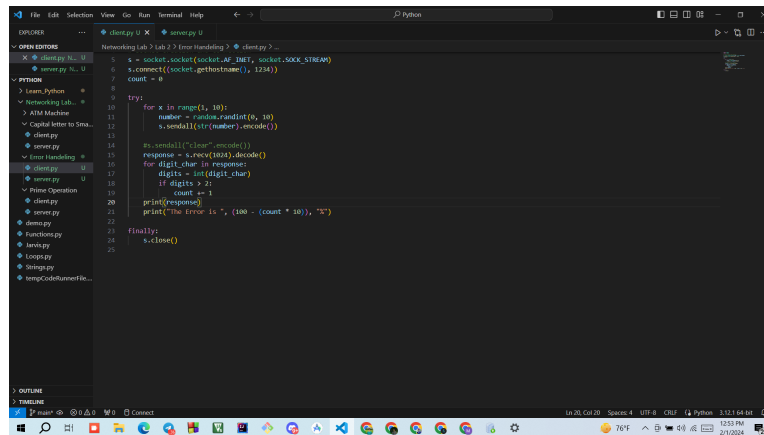
2 import socket
3 import random
4
5 # Socket creation and connection
6 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 s.connect((socket.gethostname(), 1234))
8 count = 0
9
10 try:
11     for x in range(1, 10):
12         number = random.randint(0, 10)
13         s.sendall(str(number).encode())
14
15         #s.sendall("clear".encode())
16         response = s.recv(1024).decode()
17         for digit_char in response:
18             digits = int(digit_char)
19             if digits > 2:
20                 count += 1
21         print(response)
22         print("The Error is ", (100 - (count * 10)), "%")
23
24 finally:

```

25

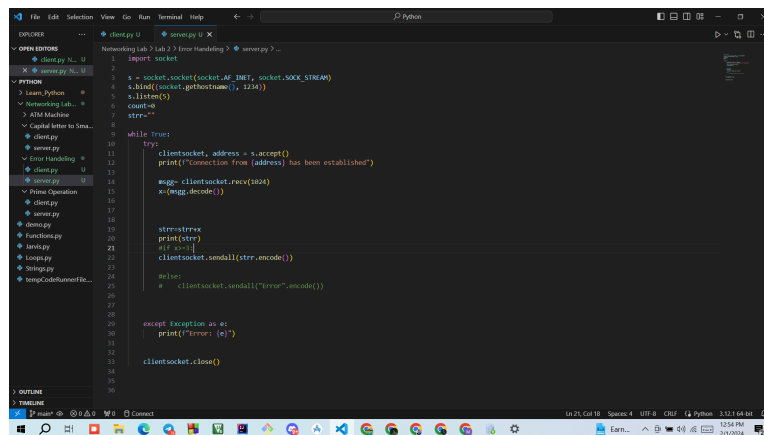
```
s.close()
```

26



```
1 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2 s.connect((socket.gethostname(), 1234))
3 count = 0
4
5 try:
6     for x in range(1, 10):
7         number = random.randint(0, 10)
8         s.sendall(str(number).encode())
9
10        response = s.recv(1024).decode()
11        for digit_char in response:
12            digits = int(digit_char)
13            if digits > 2:
14                count = 1
15            print(response)
16            print("The error is ", (100 - (count * 10)), "%")
17
18    finally:
19        s.close()
```

Figure 13: Client side code



```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 s.bind((socket.gethostname(), 1234))
5 s.listen(5)
6 count = 0
7 str = ""
8
9 while True:
10     try:
11         clientsocket, address = s.accept()
12         print("Connection from {address} has been established")
13         msg = clientsocket.recv(1024)
14         # msg.decode()
15
16         str += msg
17
18         print(str)
19         # if str == "1":
20             clientsocket.sendall(str.encode())
21         else:
22             clientsocket.sendall("Error".encode())
23
24     except Exception as e:
25         print("Error: {}".format(e))
26
27     clientsocket.close()
```

Figure 14: Server Side Code

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3306]
(c) Microsoft Corporation. All rights reserved.

C:\Python\Python\Networking Lab\Lab 2>python -u client.py
60101008
The error is: 50 5
C:\Python\Python\Networking Lab\Lab 2>python -u handling.py
```

Figure 15: Client Side result

```
C:\Windows\System32\cmd.exe python -u server.py
Microsoft Windows [Version 10.0.19045.3306]
(c) Microsoft Corporation. All rights reserved.

C:\Python\Python\Networking Lab\Lab 2>python -u server.py
Connection from ('192.168.0.10', 5900) has been established
60101008
```

Figure 16: Server Side result

5 Experience

1. We had to see some examples of how to use Server-Client package in Python.
2. We had to compile both of them on two different terminals or tabs.
3. We had to run the Server program first. Then run the Client program.
4. Then we had to type messages in the Client Window which will be received and shown by the Server Window simultaneously.
5. Then we had to type "Over" to end the program.

References

- [1] Socket Programming in Python: <https://www.geeksforgeeks.org/socket-programming-python/>
- [2] Socket Programming in Python: <https://realpython.com/python-sockets/>