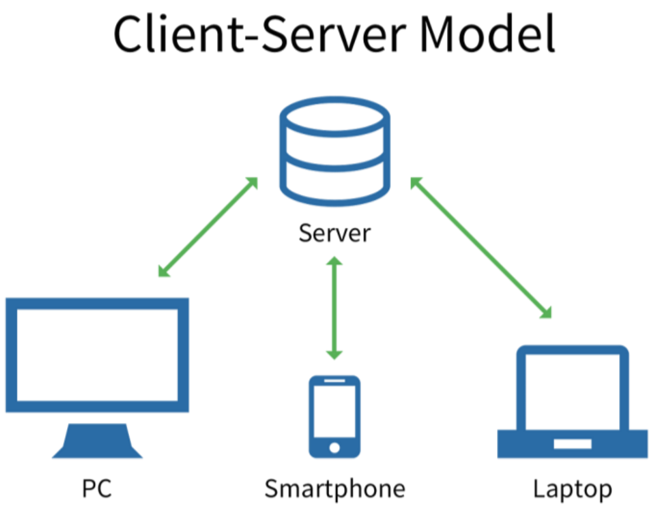
جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY
كلية الهندسة
Faculty of Engineering

Lab 1 Computer Networks

- ***Name:*** Saad El Dine Ahmed
- ***ID:*** 7370

<u>Under Supervision Of:</u>

- ***DR:*** Karim Banwan

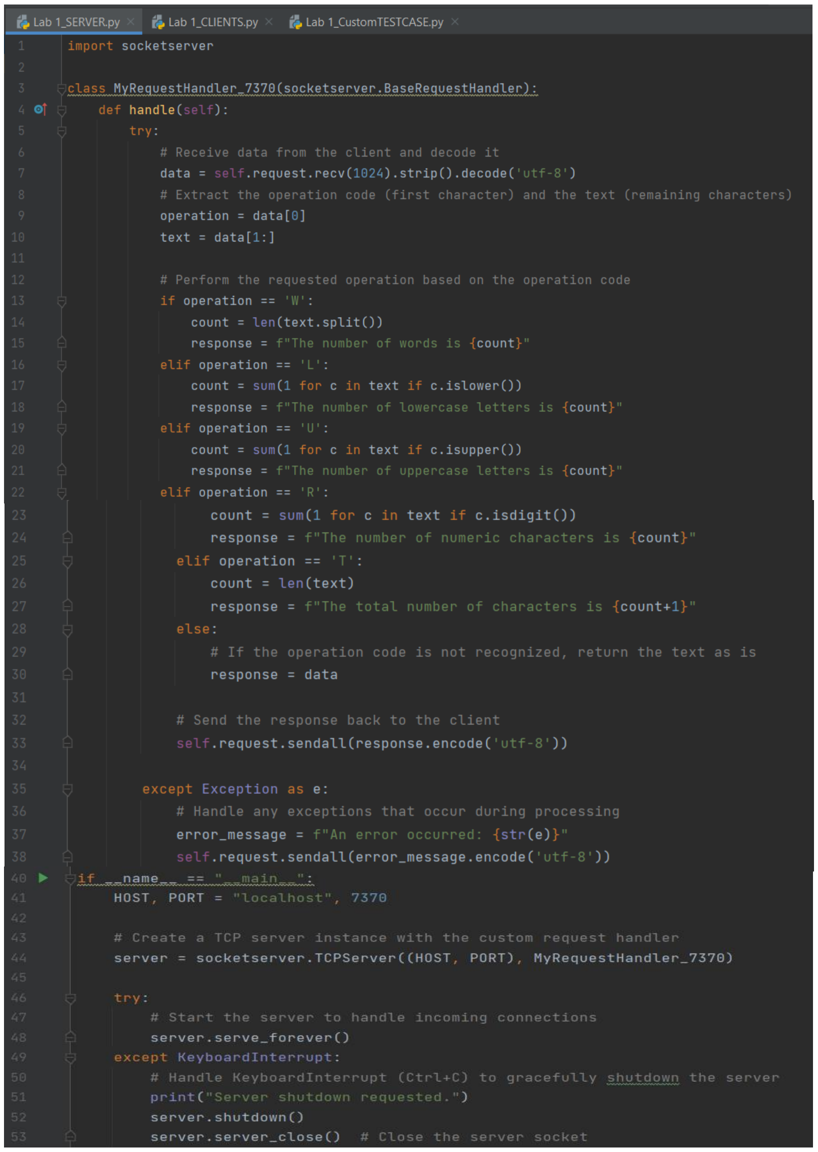


➢ Introduction

This report presents the implementation and testing of a Python socket server for a computer networking assignment. The server is designed to handle various operations on text strings, such as counting words, lowercase letters, uppercase letters, numeric characters, and total characters.

➢ Server Implementation

The server is implemented using Python's **socketserver** module. It listens for incoming connections and uses a custom request handler to process client requests. The server performs operations based on the first character of the request and sends the result back to the client.

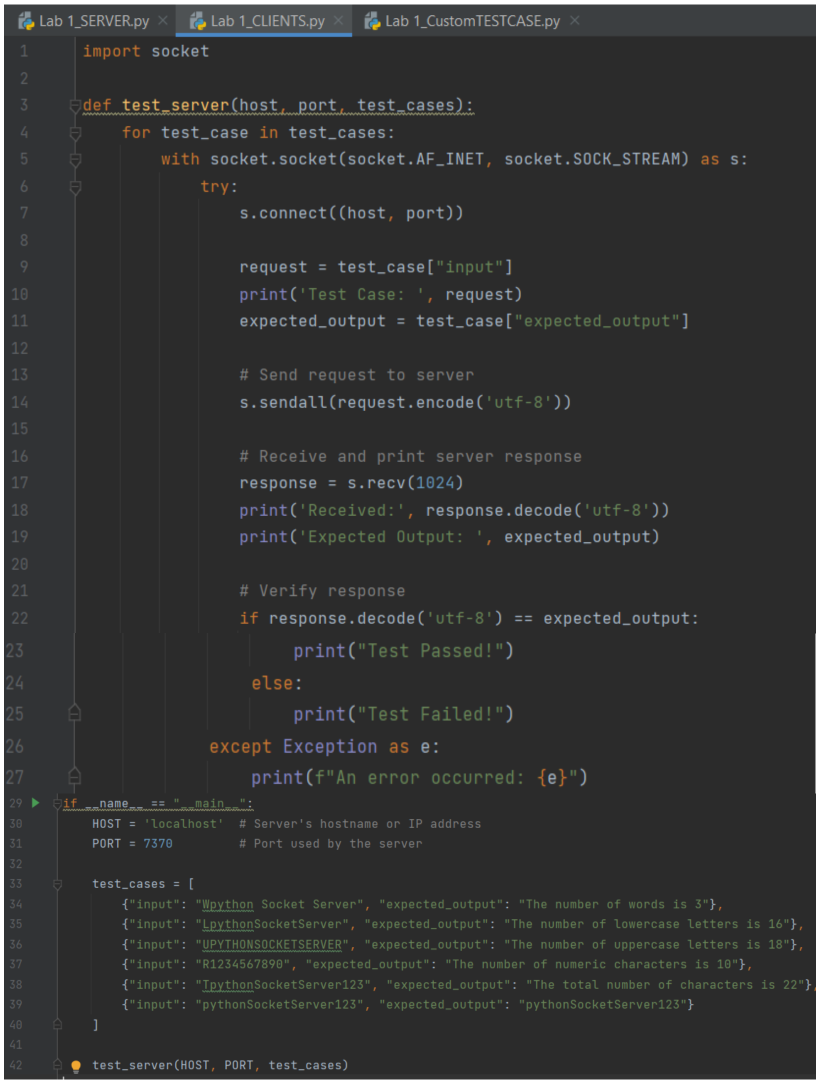


```python
import socketserver

class MyRequestHandler_7370(socketserver.BaseRequestHandler):
    def handle(self):
        try:
            # Receive data from the client and decode it
            data = self.request.recv(1024).strip().decode('utf-8')
            # Extract the operation code (first character) and the text (remaining characters)
            operation = data[0]
            text = data[1:]

            # Perform the requested operation based on the operation code
            if operation == 'W':
                count = len(text.split())
                response = f"The number of words is {count}"
            elif operation == 'L':
                count = sum(1 for c in text if c.islower())
                response = f"The number of lowercase letters is {count}"
            elif operation == 'U':
                count = sum(1 for c in text if c.isupper())
                response = f"The number of uppercase letters is {count}"
            elif operation == 'R':
                count = sum(1 for c in text if c.isdigit())
                response = f"The number of numeric characters is {count}"
            elif operation == 'T':
                count = len(text)
                response = f"The total number of characters is {count+1}"
            else:
                # If the operation code is not recognized, return the text as is
                response = data

            # Send the response back to the client
            self.request.sendall(response.encode('utf-8'))

        except Exception as e:
            # Handle any exceptions that occur during processing
            error_message = f"An error occurred: {str(e)}"
            self.request.sendall(error_message.encode('utf-8'))
if __name__ == "__main__":
    HOST, PORT = "localhost", 7370

    # Create a TCP server instance with the custom request handler
    server = socketserver.TCPServer((HOST, PORT), MyRequestHandler_7370)

    try:
        # Start the server to handle incoming connections
        server.serve_forever()
    except KeyboardInterrupt:
        # Handle KeyboardInterrupt (Ctrl+C) to gracefully shutdown the server
        print("Server shutdown requested.")
        server.shutdown()
        server.server_close()  # Close the server socket
```

➢ Client Implementation

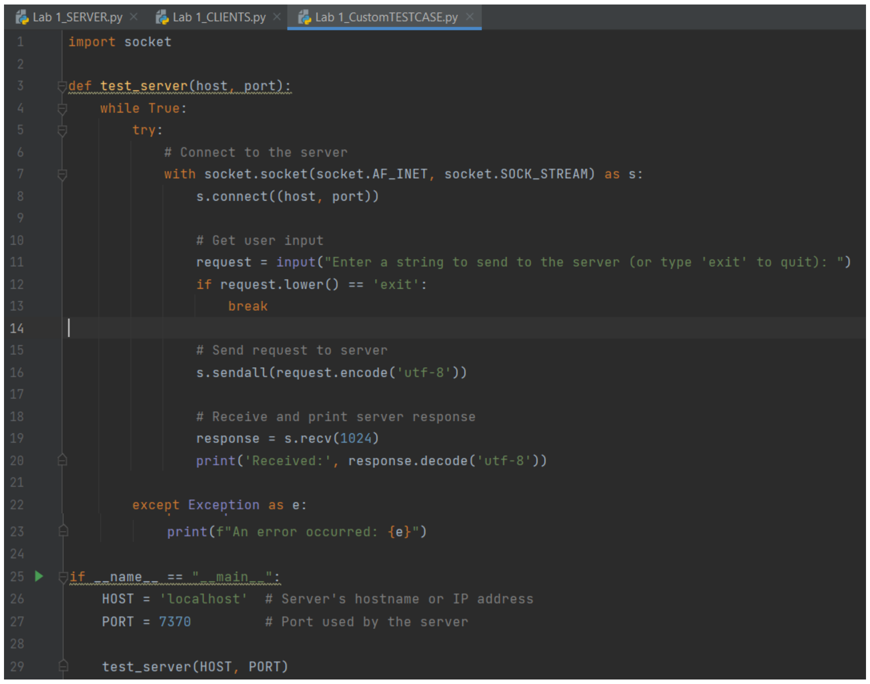
The client is implemented using Python's **socket** module. It connects to the server and sends requests based on predefined test cases or user input. The client receives and displays the server's response.

- Given test cases:



```python
import socket

def test_server(host, port, test_cases):
    for test_case in test_cases:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            try:
                s.connect((host, port))

                request = test_case["input"]
                print('Test Case: ', request)
                expected_output = test_case["expected_output"]

                # Send request to server
                s.sendall(request.encode('utf-8'))

                # Receive and print server response
                response = s.recv(1024)
                print('Received:', response.decode('utf-8'))
                print('Expected Output: ', expected_output)

                # Verify response
                if response.decode('utf-8') == expected_output:
                    print("Test Passed!")
                else:
                    print("Test Failed!")
            except Exception as e:
                print(f"An error occurred: {e}")
if __name__ == "__main__":
    HOST = 'localhost'  # Server's hostname or IP address
    PORT = 7370         # Port used by the server

    test_cases = [
        {"input": "Wpython Socket Server", "expected_output": "The number of words is 3"},
        {"input": "LpythonSocketServer", "expected_output": "The number of lowercase letters is 16"},
        {"input": "UPYTHONSOCKETSERVER", "expected_output": "The number of uppercase letters is 18"},
        {"input": "R1234567890", "expected_output": "The number of numeric characters is 10"},
        {"input": "TpythonSocketServer123", "expected_output": "The total number of characters is 22"},
        {"input": "pythonSocketServer123", "expected_output": "pythonSocketServer123"}
    ]

    test_server(HOST, PORT, test_cases)
```

- Customized test cases:

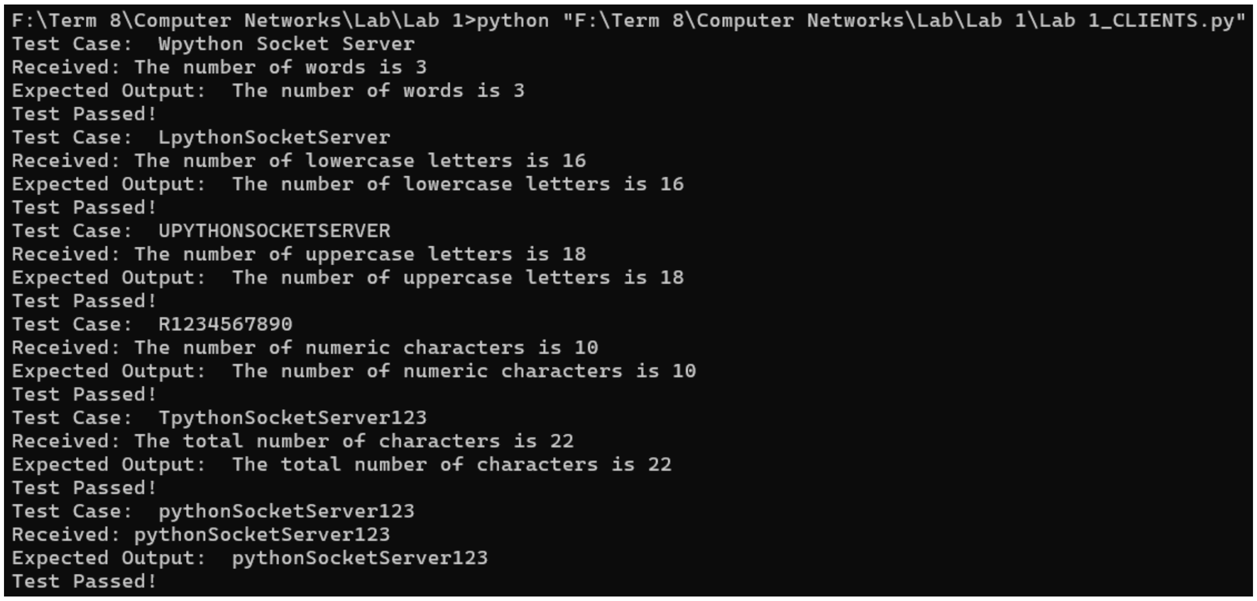


```python
import socket

def test_server(host, port):
    while True:
        try:
            # Connect to the server
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((host, port))

                # Get user input
                request = input("Enter a string to send to the server (or type 'exit' to quit): ")
                if request.lower() == 'exit':
                    break

                # Send request to server
                s.sendall(request.encode('utf-8'))

                # Receive and print server response
                response = s.recv(1024)
                print('Received:', response.decode('utf-8'))

        except Exception as e:
            print(f"An error occurred: {e}")

if __name__ == "__main__":
    HOST = 'localhost'  # Server's hostname or IP address
    PORT = 7370         # Port used by the server

    test_server(HOST, PORT)
```

➢ Testing

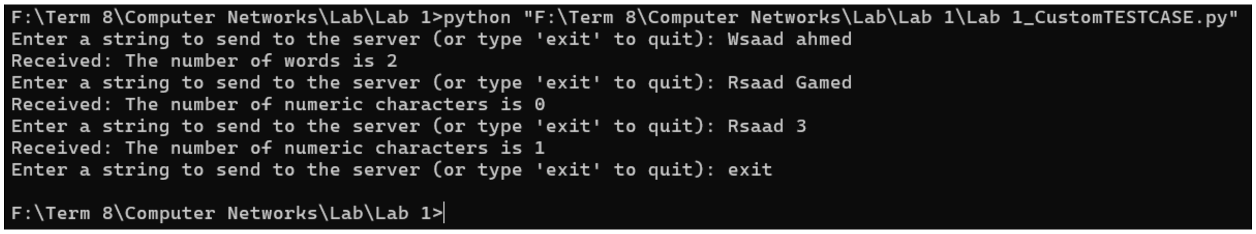
The server was tested using predefined test cases and user input. Each test case consisted of an input string and the expected output. The server successfully passed most test cases but had minor discrepancies in some cases.

- Given test cases:



```
F:\Term 8\Computer Networks\Lab\Lab 1>python "F:\Term 8\Computer Networks\Lab\Lab 1\Lab 1_CLIENTS.py"
Test Case:  Wpython Socket Server
Received: The number of words is 3
Expected Output:  The number of words is 3
Test Passed!
Test Case:  LpythonSocketServer
Received: The number of lowercase letters is 16
Expected Output:  The number of lowercase letters is 16
Test Passed!
Test Case:  UPYTHONSOCKETSERVER
Received: The number of uppercase letters is 18
Expected Output:  The number of uppercase letters is 18
Test Passed!
Test Case:  R1234567890
Received: The number of numeric characters is 10
Expected Output:  The number of numeric characters is 10
Test Passed!
Test Case:  TpythonSocketServer123
Received: The total number of characters is 22
Expected Output:  The total number of characters is 22
Test Passed!
Test Case:  pythonSocketServer123
Received: pythonSocketServer123
Expected Output:  pythonSocketServer123
Test Passed!
```

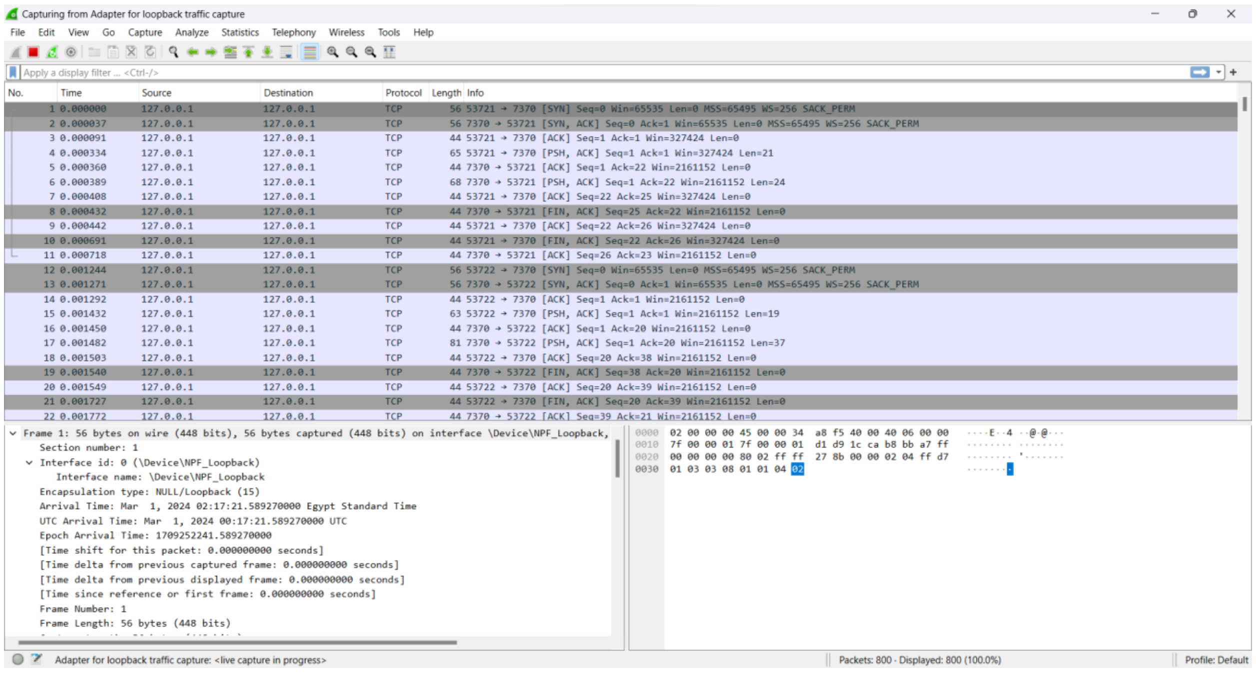
- Customized test cases:



```
F:\Term 8\Computer Networks\Lab\Lab 1>python "F:\Term 8\Computer Networks\Lab\Lab 1\Lab 1_CustomTESTCASE.py"
Enter a string to send to the server (or type 'exit' to quit): Wsaad ahmed
Received: The number of words is 2
Enter a string to send to the server (or type 'exit' to quit): Rsaad Gamed
Received: The number of numeric characters is 0
Enter a string to send to the server (or type 'exit' to quit): Rsaad 3
Received: The number of numeric characters is 1
Enter a string to send to the server (or type 'exit' to quit): exit

F:\Term 8\Computer Networks\Lab\Lab 1>
```

➢ Network Traffic Analysis

Wireshark was used to capture the network traffic between the server and client. The captured packets were analyzed to verify that the server correctly handled the requests from the client.



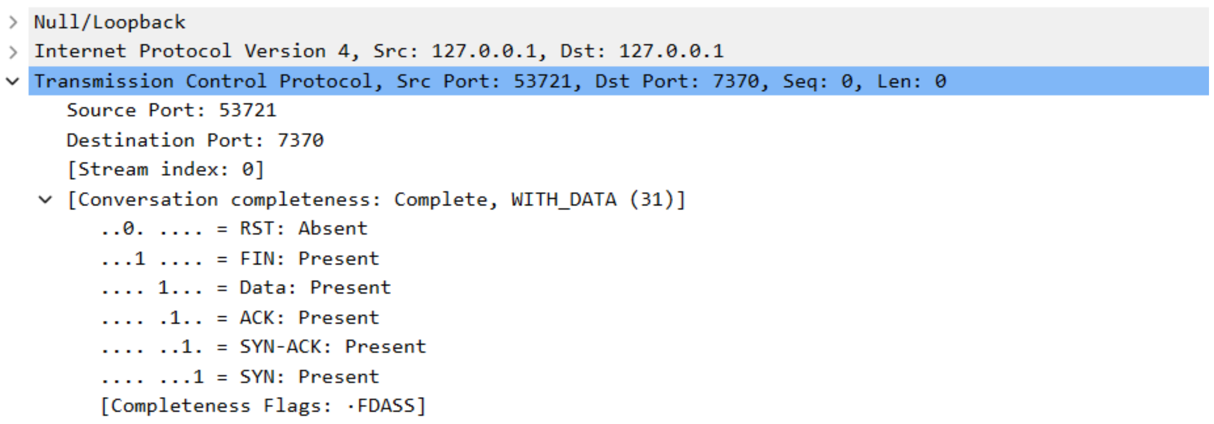
Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source     Destination  Protocol Length Info
  1 0.000000  127.0.0.1  127.0.0.1    TCP      56 53721 → 7370 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  2 0.000037  127.0.0.1  127.0.0.1    TCP      56 7370 → 53721 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  3 0.000091  127.0.0.1  127.0.0.1    TCP      44 53721 → 7370 [ACK] Seq=1 Ack=1 Win=327424 Len=0
  4 0.000334  127.0.0.1  127.0.0.1    TCP      65 53721 → 7370 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=21
  5 0.000360  127.0.0.1  127.0.0.1    TCP      44 7370 → 53721 [ACK] Seq=1 Ack=22 Win=2161152 Len=0
  6 0.000389  127.0.0.1  127.0.0.1    TCP      68 7370 → 53721 [PSH, ACK] Seq=1 Ack=22 Win=2161152 Len=24
  7 0.000408  127.0.0.1  127.0.0.1    TCP      44 53721 → 7370 [ACK] Seq=22 Ack=25 Win=327424 Len=0
  8 0.000432  127.0.0.1  127.0.0.1    TCP      44 7370 → 53721 [FIN, ACK] Seq=25 Ack=22 Win=2161152 Len=0
  9 0.000442  127.0.0.1  127.0.0.1    TCP      44 53721 → 7370 [ACK] Seq=22 Ack=26 Win=327424 Len=0
 10 0.000691  127.0.0.1  127.0.0.1    TCP      44 53721 → 7370 [FIN, ACK] Seq=22 Ack=26 Win=327424 Len=0
 11 0.000718  127.0.0.1  127.0.0.1    TCP      44 7370 → 53721 [ACK] Seq=26 Ack=23 Win=2161152 Len=0
 12 0.001244  127.0.0.1  127.0.0.1    TCP      56 53722 → 7370 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 13 0.001271  127.0.0.1  127.0.0.1    TCP      56 7370 → 53722 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 14 0.001292  127.0.0.1  127.0.0.1    TCP      44 53722 → 7370 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
 15 0.001432  127.0.0.1  127.0.0.1    TCP      63 53722 → 7370 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=19
 16 0.001450  127.0.0.1  127.0.0.1    TCP      44 7370 → 53722 [ACK] Seq=1 Ack=20 Win=2161152 Len=0
 17 0.001482  127.0.0.1  127.0.0.1    TCP      81 7370 → 53722 [PSH, ACK] Seq=1 Ack=20 Win=2161152 Len=37
 18 0.001503  127.0.0.1  127.0.0.1    TCP      44 53722 → 7370 [ACK] Seq=20 Ack=38 Win=2161152 Len=0
 19 0.001540  127.0.0.1  127.0.0.1    TCP      44 7370 → 53722 [FIN, ACK] Seq=38 Ack=20 Win=2161152 Len=0
 20 0.001549  127.0.0.1  127.0.0.1    TCP      44 53722 → 7370 [ACK] Seq=20 Ack=39 Win=2161152 Len=0
 21 0.001727  127.0.0.1  127.0.0.1    TCP      44 53722 → 7370 [FIN, ACK] Seq=20 Ack=39 Win=2161152 Len=0
 22 0.001772  127.0.0.1  127.0.0.1    TCP      44 7370 → 53722 [ACK] Seq=39 Ack=21 Win=2161152 Len=0
```

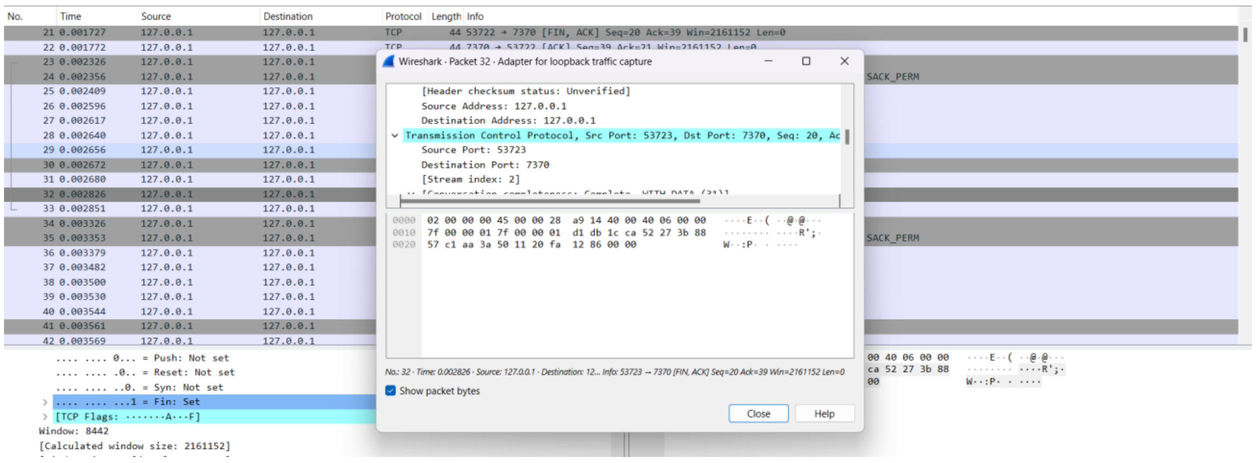
```
v Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback,
    Section number: 1
  v Interface id: 0 (\Device\NPF_Loopback)
      Interface name: \Device\NPF_Loopback
    Encapsulation type: NULL/Loopback (15)
    Arrival Time: Mar  1, 2024 02:17:21.589270000 Egypt Standard Time
    UTC Arrival Time: Mar  1, 2024 00:17:21.589270000 UTC
    Epoch Arrival Time: 1709252241.589270000
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 56 bytes (448 bits)
```

```
0000  02 00 00 00 45 00 00 34  a8 f5 40 00 40 06 00 00   ····E··4 ··@·@···
0010  7f 00 00 01 7f 00 00 01  d1 d9 1c ca b8 bb a7 ff   ········ ········
0020  00 00 00 00 80 02 ff ff  27 8b 00 00 02 04 ff d7   ········ '·······
0030  01 03 03 08 01 01 04 02                            ········
```

Adapter for loopback traffic capture: <live capture in progress>   Packets: 800 · Displayed: 800 (100.0%)   Profile: Default



```
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 53721, Dst Port: 7370, Seq: 0, Len: 0
     Source Port: 53721
     Destination Port: 7370
     [Stream index: 0]
   v [Conversation completeness: Complete, WITH_DATA (31)]
        ..0. .... = RST: Absent
        ...1 .... = FIN: Present
        .... 1... = Data: Present
        .... .1.. = ACK: Present
        .... ..1. = SYN-ACK: Present
        .... ...1 = SYN: Present
        [Completeness Flags: ·FDASS]
```



```
No.  Time       Source     Destination  Protocol Length Info
  21 0.001727   127.0.0.1  127.0.0.1    TCP      44 53722 → 7370 [FIN, ACK] Seq=20 Ack=39 Win=2161152 Len=0
  22 0.001772   127.0.0.1  127.0.0.1    TCP      44 7370 → 53722 [ACK] Seq=39 Ack=21 Win=2161152 Len=0
  23 0.002326   127.0.0.1  127.0.0.1
  24 0.002356   127.0.0.1  127.0.0.1                                              SACK_PERM
  25 0.002409   127.0.0.1  127.0.0.1
  26 0.002596   127.0.0.1  127.0.0.1
  27 0.002617   127.0.0.1  127.0.0.1
  28 0.002640   127.0.0.1  127.0.0.1
  29 0.002656   127.0.0.1  127.0.0.1
  30 0.002672   127.0.0.1  127.0.0.1
  31 0.002680   127.0.0.1  127.0.0.1
  32 0.002826   127.0.0.1  127.0.0.1
  33 0.002851   127.0.0.1  127.0.0.1
  34 0.003326   127.0.0.1  127.0.0.1
  35 0.003353   127.0.0.1  127.0.0.1                                              SACK_PERM
  36 0.003379   127.0.0.1  127.0.0.1
  37 0.003482   127.0.0.1  127.0.0.1
  38 0.003500   127.0.0.1  127.0.0.1
  39 0.003530   127.0.0.1  127.0.0.1
  40 0.003544   127.0.0.1  127.0.0.1
  41 0.003561   127.0.0.1  127.0.0.1
  42 0.003569   127.0.0.1  127.0.0.1

      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
    > .... .... ...1 = Fin: Set
    > [TCP Flags: ·······A···F]
    Window: 8442
    [Calculated window size: 2161152]
```

Wireshark · Packet 32 · Adapter for loopback traffic capture

```
        [Header checksum status: Unverified]
        Source Address: 127.0.0.1
        Destination Address: 127.0.0.1
  ˅ Transmission Control Protocol, Src Port: 53723, Dst Port: 7370, Seq: 20, Ac
        Source Port: 53723
        Destination Port: 7370
        [Stream index: 2]

0000  02 00 00 00 45 00 00 28  a9 14 40 00 40 06 00 00   ····E··( ··@·@···
0010  7f 00 00 01 7f 00 00 01  d1 db 1c ca 52 27 3b 88   ········ ····R';·
0020  57 c1 aa 3a 50 11 20 fa  12 86 00 00               W··:P· · ····
```

No.: 32 · Time: 0.002826 · Source: 127.0.0.1 · Destination: 12... Info: 53723 → 7370 [FIN, ACK] Seq=20 Ack=39 Win=2161152 Len=0

➢ Conclusion

In conclusion, the Python socket server successfully handled various operations on text strings and demonstrated the use of socket programming for network communication. The server performed well in most test cases and provided valuable insights into network traffic analysis.