



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 : Sudipto Das Sukanto

Roll No : 29

Name: Ahanaf Ahmed Shawn

Roll No : 47

Submitted On :

Feb 01, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

An inter-process communication (IPC) technology called socket programming makes it possible for processes to communicate with one another over a network. Sockets provide a standard interface for network communication in a variety of operating systems, such as Linux, macOS, and Windows. They can be used to create distributed, peer-to-peer, and client-server systems, among other kinds of networked applications.

In socket programming, a socket is created on one process and connected to the socket of another process. When two processes are connected, they can read from and write to the socket to exchange data. Just two of the transport protocols that can be utilized for communication are TCP and UDP.

1.1 Objectives

The following are the major goals of these lab experiments: • Learning the foundations of inter-process communication (IPC) and how it is used in networked applications.

- Acquiring knowledge of socket generation and usage across several programming languages, such as C, C++, Java, Python, etc.
- Becoming knowledgeable about the several transport protocols that socket programming uses, such as TCP and UDP.
- Mastering socket programming to construct basic client-server architectures.
- Understanding the concepts underlying ports and sockets and how to apply them to network communication
- Acquiring knowledge about socket usage for peer-to-peer and distributed systems.
- Being familiar with writing sockets, handling errors and exceptions, and using sockets to transfer data.
- Learning the best practices and guiding principles of socket programming, in addition to

2 Theory

Inter-process communication (IPC) is the fundamental concept of socket programming; it allows processes to communicate with one another through a system of connections. One kind of software endpoint that allows for network connection between apps is a socket. Sockets provide a standard interface for network communication across a range of operating systems, including Linux, macOS, and Windows. An IP address and a port, which is a host's logical endpoint for network communication, make up a socket. To uniquely identify a network endpoint, they all cooperate. Sockets can be used with a variety of transport protocols, including UDP and TCP.

Data cannot be sent over TCP unless a connection is established because it is a connection-oriented protocol. Since UDP is a connectionless protocol, data can be sent using it without first establishing a connection. In client-server systems, a client process sends requests to a server process, which responds to the requests. Web servers and email are two examples of networked applications that commonly use client-server architectures. Peer-to-peer networks use many processes that can act as clients and servers. Peer-to-peer networks are used in many decentralized applications including file sharing. Because it allows the programmer to handle any exceptions or errors that may occur while the program is being performed, error handling is an essential part of socket programming.

3 Methodology

3.1 Server

When we enable it on the server side, it will wait for any HTTP requests. It will create a connection if it receives any requests. Following connection establishment, it gets an HTTP query corresponding to the file that the client requested. After that, it will read the file's bytes and send them to the client.

3.2 Client

Any web browser is our client side in this case. We will type in our server's IP address and the port number. After that, our browser will send a server an HTTP request. Links to some files the server provided will be shown. We can download any of them to our computer by clicking on them.

4 Code

4.1 Server Code For Banking System

```
1
2 import socket
3
4 def isPalindrome(s):
5     return s == s[::-1]
6 s = socket.socket()
7 print ("Socket successfully created")
8
9 port = 2947
10
11
12 s.bind(('', port))
13 print ("socket binded to %s" %(port))
14
15 # put the socket into listening mode
16 s.listen(5)
17 print ("socket is listening")
18
19 # a forever loop until we interrupt it or
20 # an error occurs
21 c, addr = s.accept()
22 print ('Got connection from', addr )
23 ammount=5000
24
25 while True:
26     mass=c.recv(1024).decode()
27     print(mass)
28     if mass=='D':
29         c.send("give me a ammount".encode())
30         amm=c.recv(1024).decode()
31         yoo=int(amm)
32         ammount=ammount+yoo
33         jj=str(ammount)
34         c.send(jj.encode())
35
36     elif mass=='W':
37         c.send("give me a ammount".encode())
38         amm=c.recv(1024).decode()
```

```

39     yoo=int(amm)
40     ammount=ammount-yoo
41     jj=str(ammount)
42     c.send(jj.encode())

```

Upon receiving a request, the server prompts the client to specify whether they want to deposit or withdraw funds. The client responds with 'D' for deposit or 'W' for withdrawal. In the case of a deposit ('D'), the server prompts the client to provide an amount, updates the account balance accordingly, and sends back the updated balance to the client. Conversely, in the case of a withdrawal ('W'), the server follows a similar process, deducting the specified amount from the account balance and returning the updated balance to the client.

4.1.1 Output

```

PS C:\Users\ASUS\Desktop\pora> & c:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ASUS/Desktop/pora/clint.py
socket successfully created
socket binded to 2947
socket is listening
Got connection from ('10.42.0.123', 59315)
D

```

Figure 1: Server Output for banking system

4.2 Client Code For Banking System

```

1  import socket
2
3  # Create a socket object
4  s = socket.socket()
5
6  # Define the port on which you want to connect
7  port = 2947
8
9  # connect to the server on partner computer
10 s.connect(('10.42.0.207', port))
11

```

```

12 # receive data from the server and decoding to get the
    string.
13 while True:
14     message = input()
15
16     s.send(message.encode())
17     print (s.recv(1024).decode())
18     # reply = input()
19     # s.send(reply.encode())
20
21 s.close()

```

The provided Python script establishes a client-side socket connection to a server located at IP address '10.42.0.207' and port 2947. The client enters a perpetual loop where it prompts the user to input a message. The entered message is then encoded and sent to the server using the established socket connection. Subsequently, the client waits to receive a response from the server, decodes it, and prints the message to the console. This client-server interaction continues in a loop, allowing the user to input messages and receive corresponding responses from the server. The script effectively demonstrates a basic communication channel between a client and a server using socket programming in Python.

4.2.1 Output

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python client.py
D
give me a ammount
500
5500
W
give me a ammount
800
4700
█

```

Figure 2: Client Output for banking system

4.3 Server Code For Prime Number Check

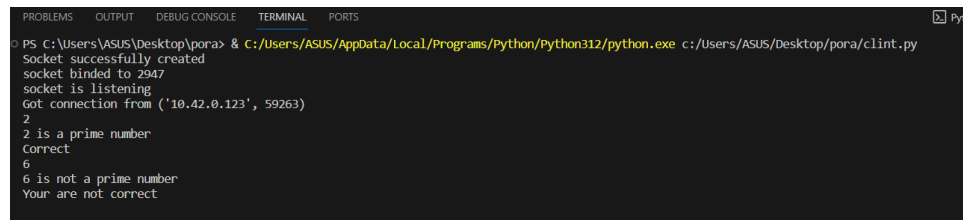
```
1
2 import socket
3
4 def isPalindrome(s):
5     return s == s[::-1]
6 s = socket.socket()
7 print ("Socket successfully created")
8
9 port = 2947
10
11
12 s.bind(('', port))
13 print ("socket binded to %s" %(port))
14
15 # put the socket into listening mode
16 s.listen(5)
17 print ("socket is listening")
18
19 # a forever loop until we interrupt it or
20 # an error occurs
21 c, addr = s.accept()
22 print ('Got connection from', addr )
23 ammount=5000
24
25 while True:
26     mass=c.recv(1024).decode()
27     print(mass)
28
29     num = int(mass)
30     if num > 1:
31         for i in range(2, int(num/2)+1):
32             if (num % i) == 0:
33                 print(num, "is not a prime number")
34                 c.send("no".encode())
35                 break
36         else:
37             print(num, "is a prime number")
38             c.send("yes".encode())
39
40     else:
```

```

41     print(num, "is not a prime number")
42     c.send("no".encode())
43
44     print(c.recv(1024).decode())

```

4.3.1 Output



```

PS C:\Users\ASUS\Desktop\pora> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ASUS/Desktop/pora/clint.py
socket successfully created
socket binded to 2947
socket is listening
Got connection from ('10.42.0.123', 59263)
2
2 is a prime number
correct
6
6 is not a prime number
Your are not correct

```

Figure 3: Server Output for Prime system

4.4 Client Code For Prime Number Check

```

1 import socket
2
3 # Create a socket object
4 s = socket.socket()
5
6 # Define the port on which you want to connect
7 port = 2947
8
9 # connect to the server on partner computer
10 s.connect(('10.42.0.207', port))
11
12 # receive data from the server and decoding to get the
    string.
13 while True:
14     message = input()
15

```



```

16     s.send(message.encode())
17     print (s.recv(1024).decode())
18     reply = input()
19     s.send(reply.encode())
20
21 s.close()

```

4.4.1 Output

```

PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python client.py
2
no
Correct
6
yes
Your are not correct

```

Figure 4: Client Output for prime number check

4.5 Server Code For palindrome Check

```

1
2
3 import socket
4
5 def isPalindrome(s):
6     return s == s[::-1]
7 s = socket.socket()
8 print ("Socket successfully created")
9
10 port = 2947
11
12
13 s.bind('', port)
14 print ("socket binded to %s" %(port))

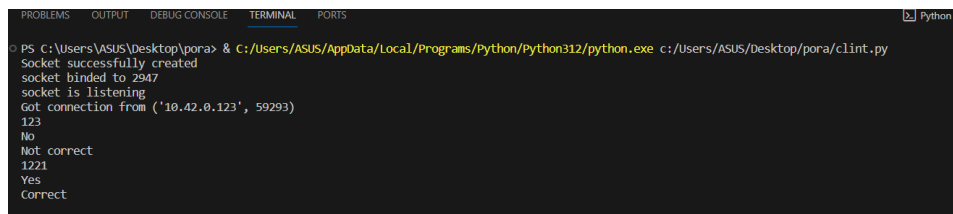
```

```

15
16 # put the socket into listening mode
17 s.listen(5)
18 print ("socket is listening")
19
20 # a forever loop until we interrupt it or
21 # an error occurs
22 c, addr = s.accept()
23 print ('Got connection from', addr )
24 ammount=5000
25
26 while True:
27     mass=c.recv(1024).decode()
28     print(mass)
29
30
31     ans = isPalindrome(mass)
32
33     if ans:
34         print("Yes")
35         c.send("yes".encode())
36
37     else:
38         print("No")
39         c.send("no".encode())
40     print(c.recv(1024).decode())

```

4.5.1 Output



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python
PS C:\Users\ASUS\Desktop\pora> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ASUS/Desktop/pora/clint.py
Socket successfully created
socket binded to 2947
socket is listening
Got connection from ('10.42.0.123', 59293)
123
No
Not correct
1221
Yes
Correct

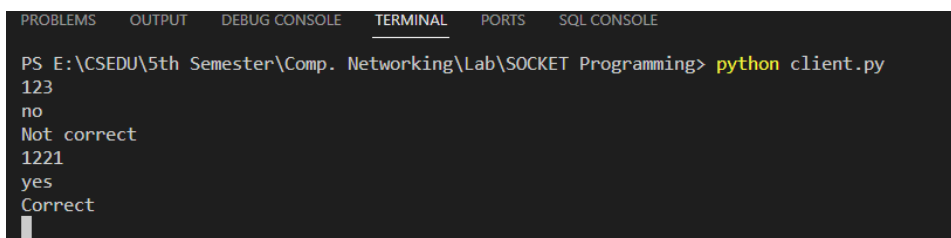
```

Figure 5: Server Output for Palindrome

4.6 Client Code For palindrome Check

```
1
2 import socket
3
4 # Create a socket object
5 s = socket.socket()
6
7 # Define the port on which you want to connect
8 port = 2947
9
10 # connect to the server on partner computer
11 s.connect(('10.42.0.207', port))
12
13 # receive data from the server and decoding to get the
    string.
14 while True:
15     message = input()
16
17     s.send(message.encode())
18     print (s.recv(1024).decode())
19     # reply = input()
20     # s.send(reply.encode())
21
22 s.close()
```

4.6.1 Output



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python client.py
123
no
Not correct
1221
yes
Correct
```

Figure 6: Client Code For palindrome Check

4.7 Server Code For Capital letter to Small letter conversion for a line of text

```
1
2 import socket
3
4 def isPalindrome(s):
5     return s == s[::-1]
6 s = socket.socket()
7 print ("Socket successfully created")
8
9 port = 2947
10
11
12 s.bind(('', port))
13 print ("socket binded to %s" %(port))
14
15 # put the socket into listening mode
16 s.listen(5)
17 print ("socket is listening")
18
19 # a forever loop until we interrupt it or
20 # an error occurs
21 c, addr = s.accept()
22 print ('Got connection from', addr )
23 ammount=5000
24
25 while True:
26     mass=c.recv(1024).decode()
27     print(mass)
28
29
30     small_line = mass.lower()
31     c.send(small_line.encode())
```

4.7.1 Output

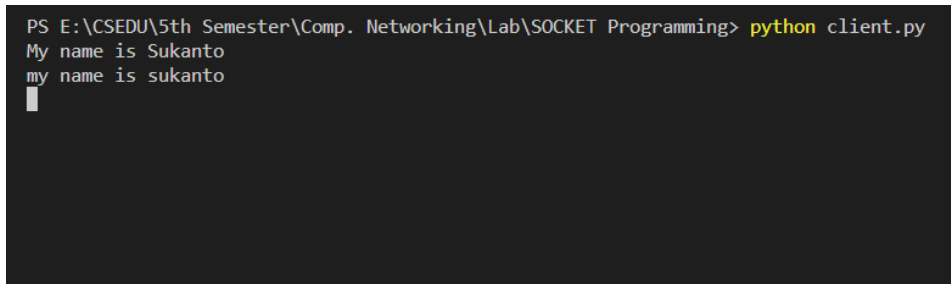
```
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python server.py
Socket successfully created
socket binded to 2947
socket is listening
Got connection from ('127.0.0.1', 51762)
My name is Sukanto
█
```

Figure 7: Server Output for Capital to Small letter

4.8 Client Code For a Capital letter to Small letter conversion for a line of text

```
1 import socket
2
3 # Create a socket object
4 s = socket.socket()
5
6 # Define the port on which you want to connect
7 port = 2947
8
9 # connect to the server on partner computer
10 s.connect(('10.42.0.207', port))
11
12 # receive data from the server and decoding to get the
   string.
13 while True:
14     message = input()
15
16     s.send(message.encode())
17     print (s.recv(1024).decode())
18     # reply = input()
19     # s.send(reply.encode())
20
21 s.close()
```

4.8.1 Output



```
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python client.py
My name is Sukanto
my name is sukanto
```

Figure 8: Client Output for capital to small letter

4.9 Handle errors in Server

In this client-server interaction, the client creates and transmits random numbers to the server. The server gets these numbers, determines whether they are less than 30, and, if so, marks them as mistakes. Additionally, the server computes the error percentage depending on the total amount of random numbers received. Simultaneously, the client runs a comparable calculation locally. This method assures that both the client and the server independently calculate and compare the error percentage. The communication process involves the client transmitting random numbers, the server analyzing them for errors, and both parties computing and cross-checking error percentages. This approach enables robust error identification and evaluation in a distributed computing environment.

```
1
2 import socket
3
4 def isPalindrome(s):
5     return s == s[::-1]
6 s = socket.socket()
7 print ("Socket successfully created")
8
```

```

9 port = 2947
10
11
12 s.bind('', port))
13 print ("socket binded to %s" %(port))
14
15 # put the socket into listening mode
16 s.listen(5)
17 print ("socket is listening")
18
19 # a forever loop until we interrupt it or
20 # an error occurs
21 c, addr = s.accept()
22 print ('Got connection from', addr )
23 error = 0
24 for i in range(100):
25     recevRand=c.recv(1024).decode()
26     recevRand = int(recevRand)
27     if(recevRand<=30) :
28         error+=1
29 error_str = str(error)
30 c.send(error_str.encode())
31 print("In server site error percentage : " + error_str +
      "%")

```

4.9.1 Output

```

PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> python server.py
Socket successfully created
socket binded to 2947
socket is listening
Got connection from ('127.0.0.1', 52968)
In server site error percentage : 31
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming> 

```

Figure 9: Server error handling

4.10 Handle errors in client

```
1 import socket
2 import random
3 import time
4
5 # Create a socket object
6 s = socket.socket()
7
8 # Define the port on which you want to connect
9 port = 2947
10
11 # connect to the server on partner computer
12 s.connect(('127.0.0.1', port))
13 client_error = 0
14 start_time = time.time()
15 for i in range(100):
16     random_number = random.randint(1, 100)
17     print(str(i) + "th number " + str(random_number))
18     if(random_number<=30):
19         client_error+=1
20     random_number_str = str(random_number)
21     s.send(random_number_str.encode())
22 end_time = time.time()
23
24 elapsed_time = end_time - start_time
25 server_error = s.recv(1024).decode()
26 client_error = str(client_error)
27 print("From server site error percentage : " +
28       server_error + "%")
29 print("In client site error percentage : " + client_error
30       + "%")
31
32 elapsed_time = str(elapsed_time)
33
34 print("Needed time for calculating error : " +
35       elapsed_time)
36
37 s.close()
```


4.10.1 Output

```
From server site error percentage : 31%  
In client site error percentage : 31%  
Needed time for calculating error : 0.04251670837402344  
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\SOCKET Programming>
```

Figure 10: Client error handling

5 Experience

1. We went through some tutorials on Socket Programming in Java.
2. We wrote a program that creates a socket and uses it to send a message to a server and receive a response.
3. Experimented with server and client interaction.
4. Wrote a program that creates a socket and uses it to send a message to a server and receive a response.
5. Experimented with different types of sockets, such as TCP and UDP.
6. Gained knowledge about the fundamentals of socket programming, including flow management, error handling, and security.
7. Practiced developing programs that implement typical network services, such as a straightforward client-server or a file transfer program, using sockets.

6 Graph of error handling (Intuition)

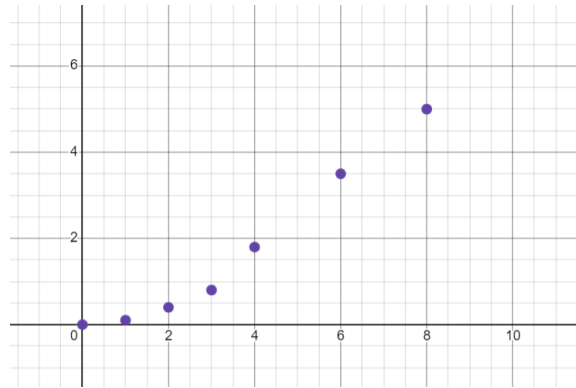


Figure 11: Error Handling graph

7 Conclusion

Try to implement every step but I am not clear about the error handling that the Teacher introduced in lab class. But I Try my best to implement every step which was spoke by the teacher. Also Graph is from my intuition.