COMPUTER COMMUNICATION AND NETWORKING

LAB 2 NETWORK SOCKET PROGRAMMING

Maaz Malik

2022600031

AIML

Aim: Network Socket Programming

Objective: the objective of this experiment is to make students acquainted with socket

programming. And make them accustomed with applications executing on top of these sockets.

Outcomes:After completion of the lab students will be able to

[1] Proficiently complete the Socket programming.

[2] Establish a process to process communication.

[3] Understand the Client-Server paradigm.

Theory: A Socket serves as a connection end point for process-to-process delivery. It consists of

Port Number and IP Address. The detailed working of the socket is mentioned in the diagram

Below.

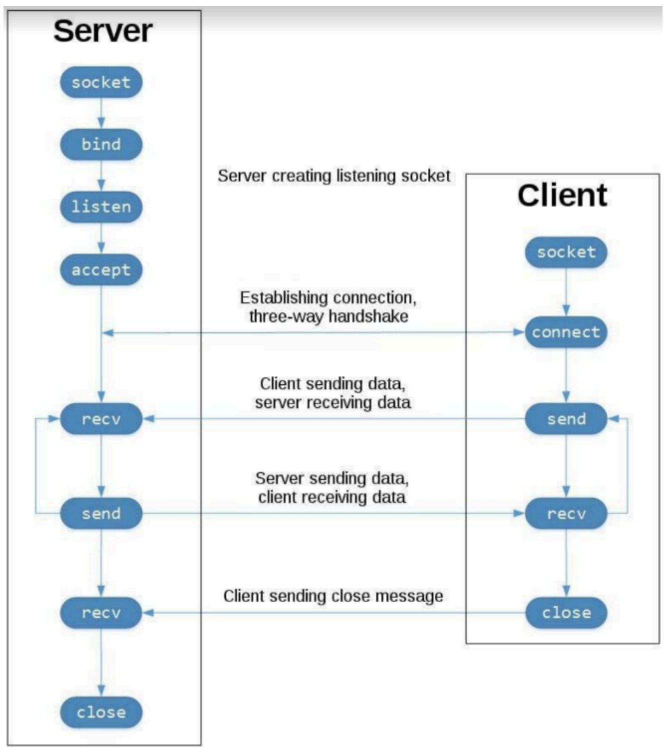

Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

Fig.1. Working of Socket on Server and Client Side

Requirements:

1. Unix based system on machine.

2. Python/Java/C language compiler/interpreter should be intstalled.

Steps to run a socket program:

1. Open a terminal and run the Server program first. **

2. Open Separate second terminal window and run a client program. Please make sure the

server is up before running client code.

3. If you want to run multiple clients, open multiple terminal windows and run your client

program on One client per terminal basis.

Problem Statement

All four parts need to be submitted in the final uploaded document. Once done with lab

assignment, start with home assignment. Students can use any programming language out of

Python, Java or C. But once selected all 4 parts need to be implemented in same language.

Lab Assignment (To be Completed in College lab hours)

Part-1: Implement the following rudimentary string processing application using

connection-oriented client-server programming. Some guidelines for the implementation are as

follows. The client will send a textual paragraph terminated by '\n' to the server (assume that in

the paragraph, '.' appears only at the end of sentences and nowhere else). The server will

compute the number of characters, number of words, and number of sentences in the paragraph, and send these numbers back to the client. The client will print these numbers on the screen.

Part-2: Make it concurrent so that it can serve multiple clients at a time. (Multiple clients on

multiple terminals and single server terminals)

Home Assignment (Can be completed till the submission due date)

Part-3: Write client-server application for chat server. The two clients connected to the same

server should be able to communicate with each other. Communication should be interactive and

go on till one of them terminates.

Part-4: Upgrade the code in part-3 chat server to act as both group and direct chats.

References:

Computer Networking: Top-Down Approach, by Kurose and Ross, Page 159: UDP Sockets and

Page 164: TCP Socket

Python: Socket Programming in Python (Guide) – Real Python

Java Programming: Java Socket Programming (Java Networking Tutorial) - javatpoint

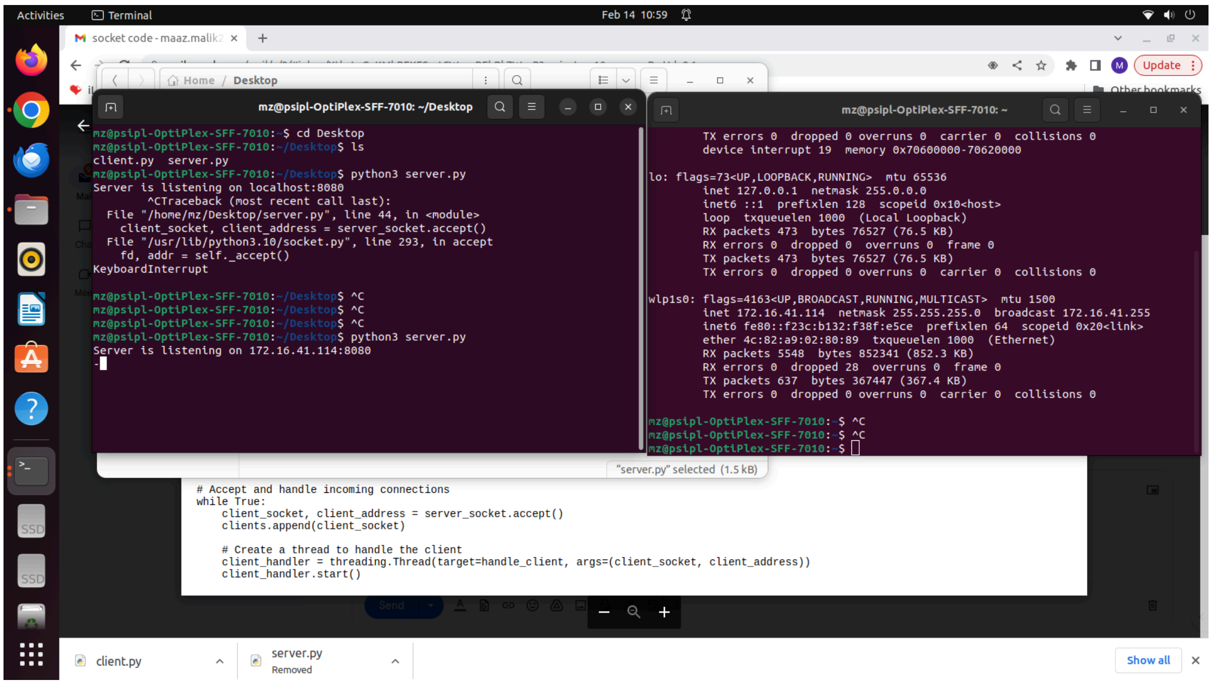
C Programming: Socket Programming in C/C++ - GeeksforGeeks

PROGRAM

```
#server.py
import socket
import threading

# Server configuration
HOST = 'localhost'
PORT = 8080

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to a specific address and port
server_socket.bind((HOST, PORT))

# Listen for incoming connections
server_socket.listen()

print(f"Server is listening on {HOST}:{PORT}")

# List to store connected clients
clients = []

def handle_client(client_socket, client_address):
    # Send a welcome message to the client
    client_socket.send("Welcome to the chat server!".encode('utf-8'))

    try:
        while True:
            # Receive messages from the client
            message = client_socket.recv(1024).decode('utf-8')

            # Broadcast the message to all clients
            for other_client in clients:
                if other_client != client_socket:
                    other_client.send(f"{client_address}:
{message}".encode('utf-8'))

    except (socket.error, BrokenPipeError):
        # Handle client disconnection
        print(f"Connection with {client_address} closed.")
        clients.remove(client_socket)
        client_socket.close()

# Accept and handle incoming connections
while True:
    client_socket, client_address = server_socket.accept()
    clients.append(client_socket)

    # Create a thread to handle the client
    client_handler = threading.Thread(target=handle_client,
args=(client_socket, client_address))
    client_handler.start()
```

```
#client.py
import socket
import threading

# Client configuration
HOST = 'localhost'
PORT = 8080

# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect((HOST, PORT))

def receive_messages():
    try:
        while True:
            # Receive messages from the server
            message = client_socket.recv(1024).decode('utf-8')
            print(message)
    except (socket.error, BrokenPipeError):
        # Handle server disconnection
        print("Disconnected from the server.")
        client_socket.close()

# Create a thread to receive messages
receive_thread = threading.Thread(target=receive_messages)
receive_thread.start()

# Send messages to the server
try:
    while True:
        user_input = input()
        client_socket.send(user_input.encode('utf-8'))
except KeyboardInterrupt:
    # Handle keyboard interrupt (Ctrl+C)
    print("Closing the chat.")
    client_socket.close()
```

OUTPUT

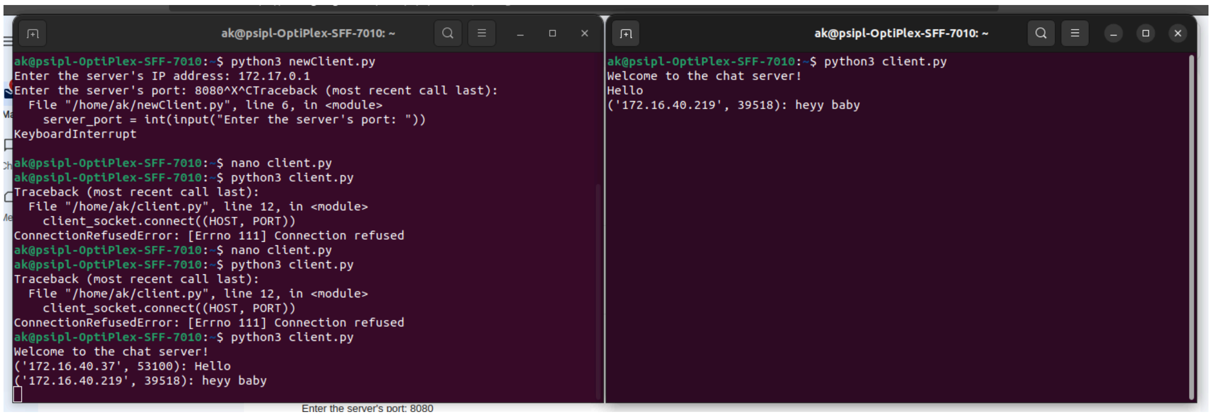


```
mz@psipl-OptiPlex-SFF-7010:~$ cd Desktop
mz@psipl-OptiPlex-SFF-7010:~/Desktop$ ls
client.py  server.py
mz@psipl-OptiPlex-SFF-7010:~/Desktop$ python3 server.py
Server is listening on localhost:8080
        ^CTraceback (most recent call last):
  File "/home/mz/Desktop/server.py", line 44, in <module>
    client_socket, client_address = server_socket.accept()
  File "/usr/lib/python3.10/socket.py", line 293, in accept
    fd, addr = self._accept()
KeyboardInterrupt

mz@psipl-OptiPlex-SFF-7010:~/Desktop$ ^C
mz@psipl-OptiPlex-SFF-7010:~/Desktop$ ^C
mz@psipl-OptiPlex-SFF-7010:~/Desktop$ ^C
mz@psipl-OptiPlex-SFF-7010:~/Desktop$ python3 server.py
Server is listening on 172.16.41.114:8080
-
```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  memory 0x70600000-70620000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 473  bytes 76527 (76.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 473  bytes 76527 (76.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.41.114  netmask 255.255.255.0  broadcast 172.16.41.255
        inet6 fe80::f23c:b132:f38f:e5ce  prefixlen 64  scopeid 0x20<link>
        ether 4c:82:a9:02:80:89  txqueuelen 1000  (Ethernet)
        RX packets 5548  bytes 852341 (852.3 KB)
        RX errors 0  dropped 28  overruns 0  frame 0
        TX packets 637  bytes 367447 (367.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mz@psipl-OptiPlex-SFF-7010:~$ ^C
mz@psipl-OptiPlex-SFF-7010:~$ ^C
mz@psipl-OptiPlex-SFF-7010:~$ 
```

```
# Accept and handle incoming connections
while True:
    client_socket, client_address = server_socket.accept()
    clients.append(client_socket)

    # Create a thread to handle the client
    client_handler = threading.Thread(target=handle_client, args=(client_socket, client_address))
    client_handler.start()
```

Terminal
Feb 14 10:59
socket code - maaz.malik2
Home / Desktop
mz@psipl-OptiPlex-SFF-7010: ~/Desktop
mz@psipl-OptiPlex-SFF-7010: ~
"server.py" selected (1.5 kB)
Update
Other bookmarks
client.py
server.py
Removed
Show all



```
ak@psipl-OptiPlex-SFF-7010:~$ python3 newClient.py
Enter the server's IP address: 172.17.0.1
Enter the server's port: 8080^X^CTraceback (most recent call last):
  File "/home/ak/newClient.py", line 6, in <module>
    server_port = int(input("Enter the server's port: "))
KeyboardInterrupt

ak@psipl-OptiPlex-SFF-7010:~$ nano client.py
ak@psipl-OptiPlex-SFF-7010:~$ python3 client.py
Traceback (most recent call last):
  File "/home/ak/client.py", line 12, in <module>
    client_socket.connect((HOST, PORT))
ConnectionRefusedError: [Errno 111] Connection refused
ak@psipl-OptiPlex-SFF-7010:~$ nano client.py
ak@psipl-OptiPlex-SFF-7010:~$ python3 client.py
Traceback (most recent call last):
  File "/home/ak/client.py", line 12, in <module>
    client_socket.connect((HOST, PORT))
ConnectionRefusedError: [Errno 111] Connection refused
ak@psipl-OptiPlex-SFF-7010:~$ python3 client.py
Welcome to the chat server!
('172.16.40.37', 53100): Hello
('172.16.40.219', 39518): heyy baby
```

```
ak@psipl-OptiPlex-SFF-7010:~$ python3 client.py
Welcome to the chat server!
Hello
('172.16.40.219', 39518): heyy baby
```

ak@psipl-OptiPlex-SFF-7010: ~
ak@psipl-OptiPlex-SFF-7010: ~
Enter the server's port: 8080

PTO

CONCLUSION

In conclusion, this experiment aimed to familiarize students with socket programming, emphasizing client-server interactions. The tasks included implementing basic string processing and extending it to handle multiple clients concurrently. Additionally, a home assignment involved creating a chat server with interactive communication capabilities. This experiment provided a practical understanding of network socket programming concepts, laying a foundation for further exploration in the field.

REFERENCES

Computer Networking: Top-Down Approach, by Kurose and Ross, Page 159: UDP Sockets and Page 164: TCP Socket.
Python: Socket Programming in Python (Guide) – Real Python.
Java Programming: Java Socket Programming (Java Networking Tutorial) - javatpoint.
C Programming: Socket Programming in C/C++ - GeeksforGeeks.