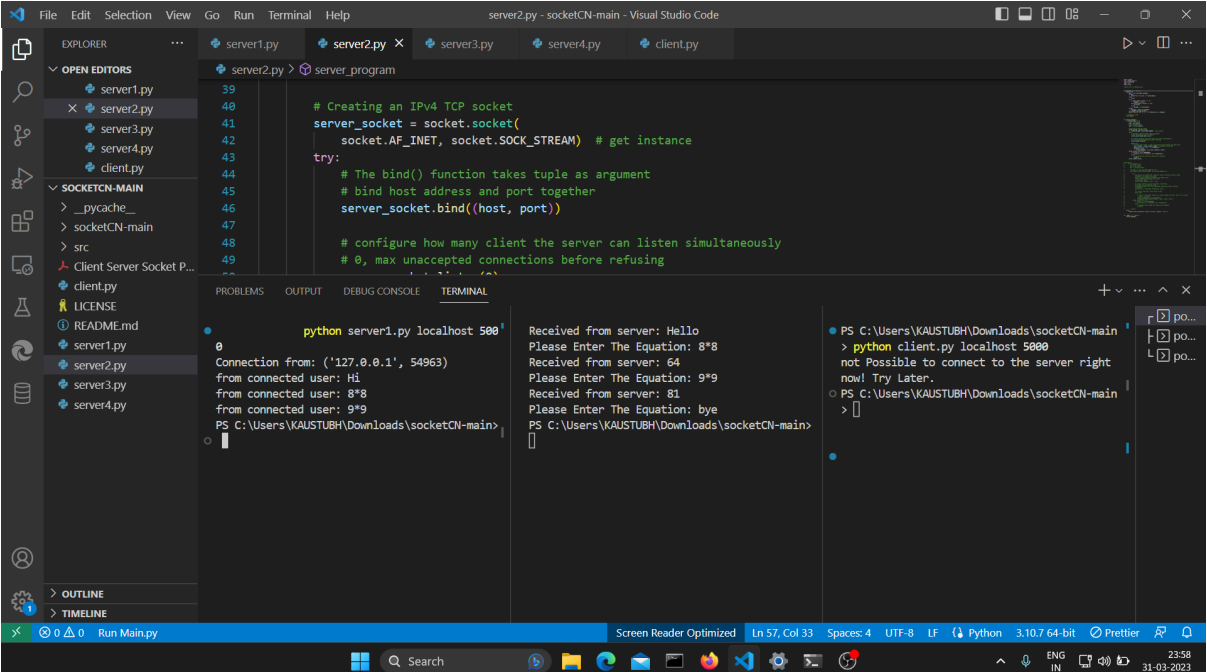


Client Server Socket Programming

(One Client Program and four Server Program)

1. **server1.py** : Your server program "server1.py " will be a single process server that can handle only one client at a time. If a second client tries to chat with the server while some other client's session is already in progress, the second client's socket operations should see an error. After the first client closes the connection, the server should then accept connections from the other client.



The screenshot displays the Visual Studio Code interface with a project named 'socketCN-main'. The Explorer sidebar on the left shows the file structure, including 'server1.py', 'server2.py', 'server3.py', 'server4.py', and 'client.py'. The main editor window is open to 'server2.py', which contains the following Python code:

```
39
40
41 # Creating an IPv4 TCP socket
42 server_socket = socket.socket(
43     socket.AF_INET, socket.SOCK_STREAM) # get instance
44
45 try:
46     # The bind() function takes tuple as argument
47     # bind host address and port together
48     server_socket.bind((host, port))
49
50     # configure how many client the server can listen simultaneously
51     # 0, max unaccepted connections before refusing
```

The bottom panel shows the 'TERMINAL' output, which includes the command 'python server1.py localhost 5000' and the resulting network interaction:

```
Connection from: ('127.0.0.1', 54963)
from connected user: Hi
from connected user: 8*8
from connected user: 9*9
PS C:\Users\KAUSTUBH\Downloads\socketCN-main>
```

On the right side of the terminal, a PowerShell prompt shows an attempt to connect to the server:

```
PS C:\Users\KAUSTUBH\Downloads\socketCN-main> python client.py localhost 5000
not Possible to connect to the server right now! Try Later.
PS C:\Users\KAUSTUBH\Downloads\socketCN-main>
```

2. **server2.py** : Your server program "server2.py " will be a multi-threaded server that will create a new thread for every new client request it receives. Multiple clients should be able to simultaneously chat with the server.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows a project named 'socketCN-main' with files: `_pycache_`, `socketCN-main`, `src`, `Client Server Socket P...`, `client.py`, `LICENSE`, `README.md`, `server1.py`, `server2.py`, `server3.py`, and `server4.py`.
- Editor:** The `client.py` file is open, showing lines 30-33:


```
30
31 # again take input
32 message = input("Please Enter The Equation: ")
33
```
- Terminal:** The terminal shows the execution of `python server2.py localhost 5000` and `python client.py localhost 5000`. The output shows a series of messages and responses:


```
PS C:\Users\KAUSTUBH\Downloads\socketCN-main> python server2.py localhost 5000
Connection from ('127.0.0.1', 54819)
Data received: Hi
Response sent to ('127.0.0.1', 54819) = hello
Connection from ('127.0.0.1', 54820)
Data received: Hi
Response sent to ('127.0.0.1', 54820) = hello
Data received: 8*8
Response sent to ('127.0.0.1', 54820) = 64
Data received: 8+8
Response sent to ('127.0.0.1', 54819) = 16
Data received: (9*9+9)
Response sent to ('127.0.0.1', 54819) = 90
[]
```

- server3.py** : Your server program "server3.py " will be a single process server that uses the "*select*" method to handle multiple clients concurrently.

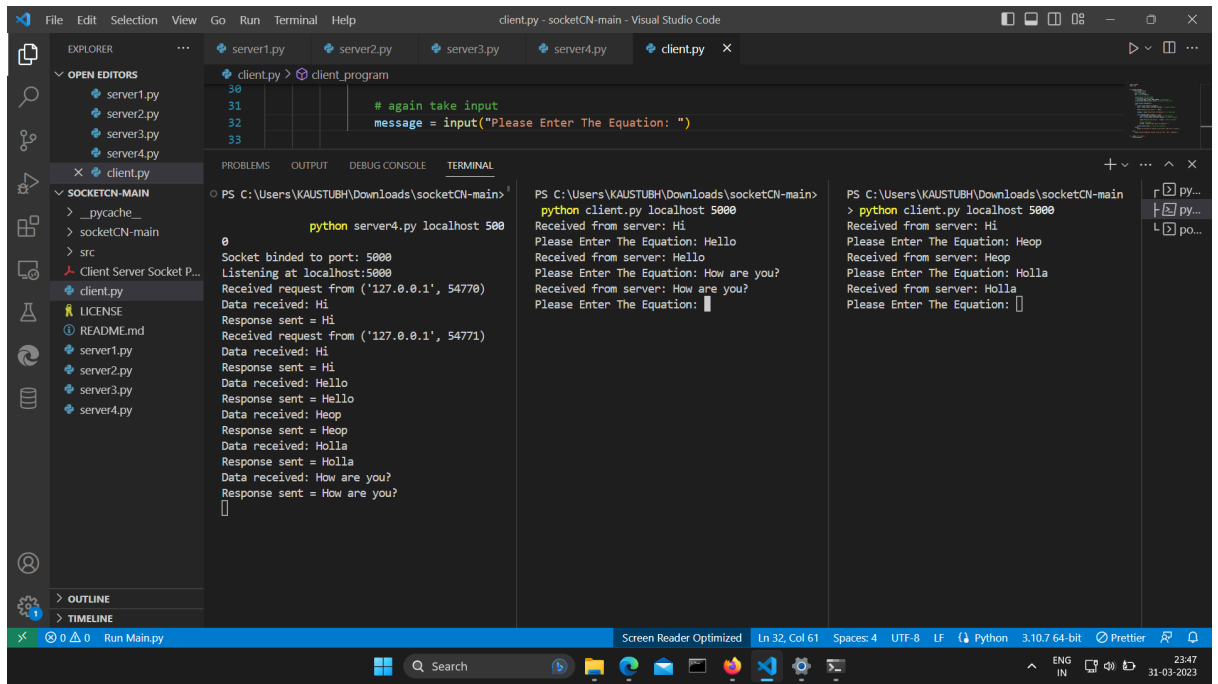
The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Same as the previous screenshot, with `server3.py` highlighted in the Explorer.
- Editor:** The `server3.py` file is open, showing lines 2-5:


```
2 import errno
3 import sys
4 import select
5 import threading
```
- Terminal:** The terminal shows the execution of `python server3.py localhost 5000` and `python client.py localhost 5000`. The output shows the server listening on port 5000 and handling multiple clients:


```
PS C:\Users\KAUSTUBH\Downloads\socketCN-main> python server3.py localhost 5000
Socket binded to port: 5000
Listening at localhost:5000
Received request from ('127.0.0.1', 54699)
Data received: Hi
Response sent to hello
Received request from ('127.0.0.1', 54700)
Data received: Hi
Response sent to hello
Data received: 8+8
Response sent to 16
Data received: 8*8
Response sent to 64
[]
```

- server4.py** : Your server program "server4.py" will be an echo server (that replies the same message to the client that was received from the same client); it will be a single process server that uses the "*select*" method to handle multiple clients concurrently.



Instructions to run the codes:

1. `python serverx.py localhost <portnumber>`
2. `python client.py localhost <portnumber>`

x = 1,2,3,4

Additional or special features that I have implemented/incorporated:

1. I used the fashion that client first says 'Hi' to the server then server says 'Hello' to client (except in server4.py (echo server) where the server also says 'Hi' to the client) then the actual code starts.
2. Various try except statements are used to get to know if some exceptions occur as well and safely run the code.
3. Various explanation with brief comments is given.
4. When the client writes 'bye' the connection is closed.