MULTI CLIENT MULTI SERVER PROGRAM FOR IDENTIFYING GEO-LOCATION THROUGH IP

Sharan SK	CED18I049
Praveen VS	CED18I054
Yoga Sri Varshan V	CED18I058
John Zakkam	CED18I059
Logeshwaran T	COE18B033
Pavendhan N	COE18B041

INTRODUCTION

This project focuses on finding GEO-LOCATION of the client willing to find his geometric parameters through the IP.

LOGIC/APPROACH

- This program showcases the potential of a client-server program.In Order to make the program user-friendly and to be accessible across all workstations ,the server has been hosted in a virtual machine(Microsoft Azure)
- In this alternate version ,we have modified the server functionalities.We have introduced a new program that acts as an interface between the server program running on a virtual machine and the client on local network.This program is called router(as it performs similar in fashion to a router performing NAT)
- We have created four virtual machines running on different public IPs.Router Program has all these four IPs.Once the client opens his program and presses the button ,client automatically connect to the router program ,it is multithreaded in a similar way to that of previous server program.Now this program checks whether a public IP is free (or left unallocated) using a semaphore variable.
- Now the router maps the free ip to the client's private ip and connects to free ip of the virtual machine functioning in a similar way a router masks the private ip.Now the server running on virtual machine ,recognizes the public ip and processes the required data and sends back to the router.Since a client-server communication is established for every thread,we send the processed data from server function present in router back to the client program.
- Semaphore variable "Sem" keeps track of available free public IPs to make requests.
- Json File keeps track of all the private -public ip mappings for every running instance of the server.Furthermore details about this can be found explained in the output section below.

THEORY:

NETWORK ADDRESS TRANSLATOR:

Network Address Translation (NAT) is the process where a network device, usually a firewall, assigns a public address to a computer (or group of computers) inside a private network. The main use of NAT is to limit the number of public IP addresses an organization or company must use, for both economy and security purposes.

The most common form of network translation involves a large private network using addresses in a private range (10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, or 192.168.0 0 to 192.168.255.255). The private addressing scheme works well for computers that only have to access resources inside the network, like workstations needing access to file servers and printers. Routers inside the private network can route traffic between private addresses with no trouble. However, to access resources outside the network, like the Internet, these computers have to have a public address in order for responses to their requests to return to them. This is where NAT comes into play.

ADVANTAGES OF NAT:

The main advantage of NAT (Network Address Translation) is that it can prevent the depletion of IPv4 addresses.

• NAT (Network Address Translation) can provide an additional layer of security by making the original source and destination addresses hidden.

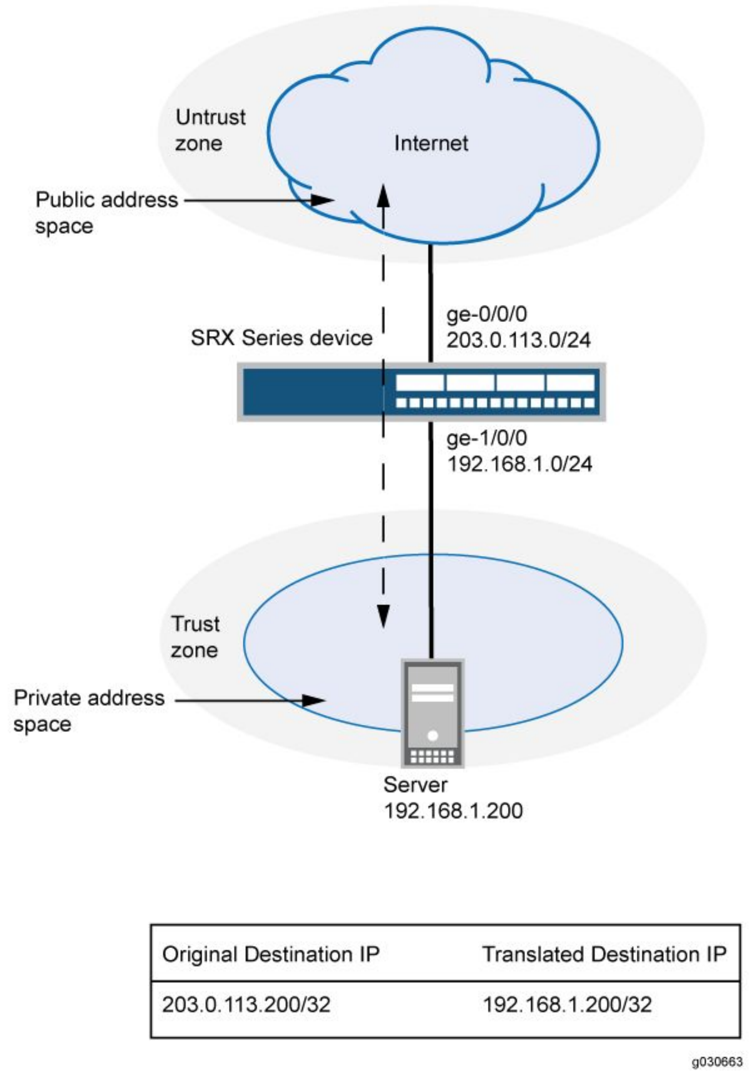
• NAT (Network Address Translation) provides increased flexibility when connecting to the public Internet.

• NAT (Network Address Translation) allows to use your own private IPv4 addressing system and prevent the internal address changes if

The main benefit of Static NAT is that Static NAT allows a computer from a remote network to initiate a connection to a Server inside the network, configured with a Private IPv4 Address.

STATIC NAT:

Static NAT defines a one-to-one mapping from one IP subnet to another IP subnet. The mapping includes destination IP address translation in one direction and source IP address translation in the reverse direction. From the NAT device, the original destination address is the virtual host IP address while the mapped-to address is the real host IP address.



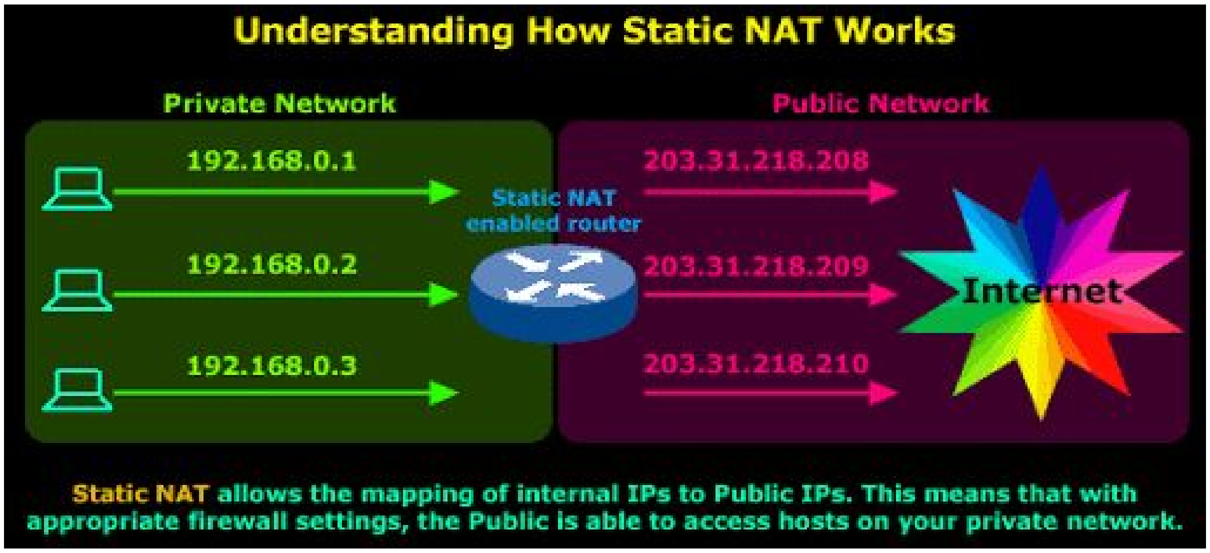
Original Destination IP	Translated Destination IP
203.0.113.200/32	192.168.1.200/32

Static NAT allows connections to be originated from either side of the network, but translation is limited to one-to-one or between blocks of addresses of the same size. For each private address, a public address must be allocated. No address pools are necessary.

In Figure given below, devices in the untrust zone access a server in the trust zone by way of public address 203.0.113.200/32. For packets that enter the Networks security device from the untrust zone with the destination IP address 203.0.113.200/32, the destination IP address is translated to the private address 192.168.1.200/32. For a new session originating from the server, the source IP address in the outgoing packet is translated to the public address 203.0.113.200/32.

This example describes the following configurations:

- Static NAT rule set rs1 with rule r1 to match packets from the untrust zone with the destination address 203.0.113.200/32. For matching packets, the destination IP address is translated to the private address 192.168.1.200/32.
- Proxy ARP for the address 203.0.113.200 on interface ge-0/0/0.0. This allows the Networks security device to respond to ARP requests received on the interface for that address.
- Security policies to permit traffic to and from the 192.168.1.200 server.


Understanding How Static NAT Works
Private Network
Public Network
192.168.0.1
192.168.0.2
192.168.0.3
Static NAT enabled router
203.31.218.208
203.31.218.209
203.31.218.210
Internet
Static NAT allows the mapping of internal IPs to Public IPs. This means that with appropriate firewall settings, the Public is able to access hosts on your private network.

ADVANTAGES:

The main benefit of Static NAT is that Static NAT allows a computer from a remote network to initiate a connection to a Server inside the network, configured with a Private IPv4 Address.

Static NAT allows a Server from inside the network (configured with a Private IPv4 Address), such as a Web Server or Mail Server, be reachable over the Internet.

FEATURES OF THE PROGRAM:

Additional OS features have been implemented to prevent critical section problem and deadlock from happening.Some of the features are:

MULTITHREADED:

Program has been Multithreaded in such a way that when a client connects to the server ,a new thread is created for that particular client.Hence every (client)thread runs simultaneously.

THREAD LOCK/SEMAPHORE:

IP-TABLE consists of all the ip addresses(private ip) of the client.This is a critical section problem as the ip addresses has to be updated every time client connects and there are chances for corruption of data .Hence every time a thread is created ,thread_lock will be acquired and will be released when ip address are updated.Semaphore variable is used to keep track of available IPs of the virtual machine to make requests.

SIGNAL HANDLER:

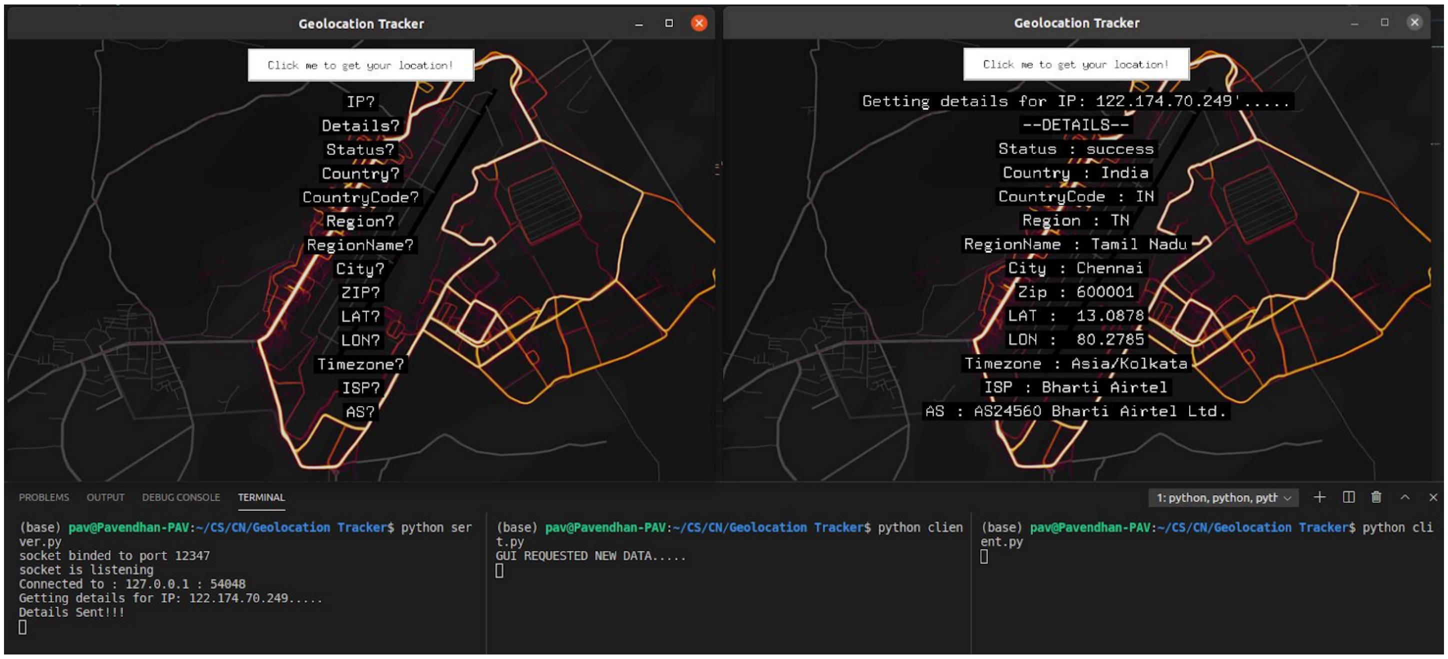
Server Program stops only when CTRL^C is pressed.Specific functions have been implemented such that when SIGINT signal is encountered by the program ,it stores all the ip addresses captured while running the program into a json file to keep track as a log file and the program is terminated.

ROUTER-VIRTUAL MACHINE-SSH:

The Program has been made into three modules to mimic the actual functionalities of how a client communicates with a server.There are Four Separate Virtual Machines Created each of them with their own separate public IPs .So once ssh connections has been made and when four server programs starts to run,it can be accessed by any workstation.Router program allocates client private ip to a public ip(that of virtual machine) and uses them whenever they are free.Further working details have been included in “Logic” section.

OUTPUT

CLIENT - ROUTER - SERVER & THE GUI APPLICATION



```
IP?
Details?
Status?
Country?
CountryCode?
Region?
RegionName?
City?
ZIP?
LAT?
LON?
Timezone?
ISP?
AS?
```

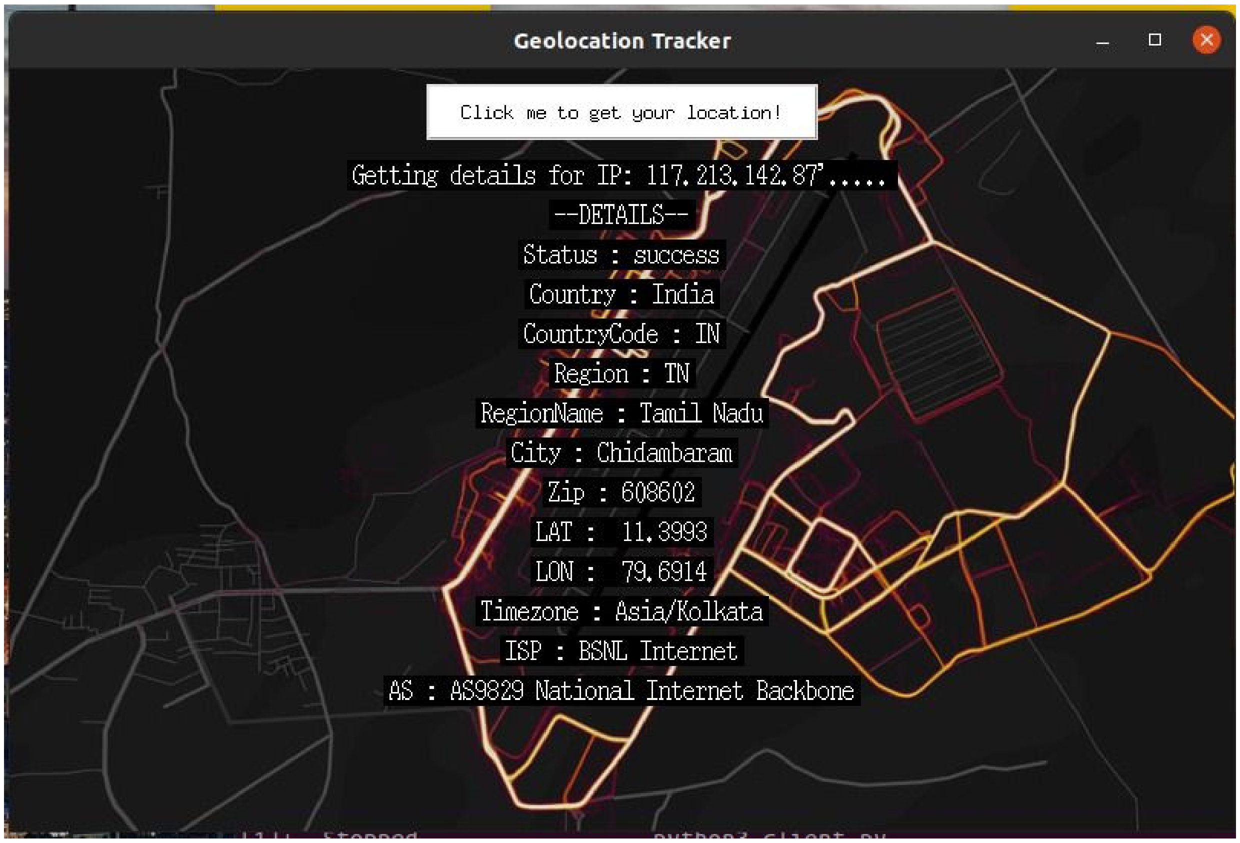
```
Getting details for IP: 122.174.70.249'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : TN
RegionName : Tamil Nadu
City : Chennai
Zip : 600001
LAT :  13.0878
LON :  80.2785
Timezone : Asia/Kolkata
ISP : Bharti Airtel
AS : AS24560 Bharti Airtel Ltd.
```

```
(base) pav@Pavendhan-PAV:~/CS/CN/Geolocation Tracker$ python ser
ver.py
socket binded to port 12347
socket is listening
Connected to : 127.0.0.1 : 54048
Getting details for IP: 122.174.70.249.....
Details Sent!!!
```

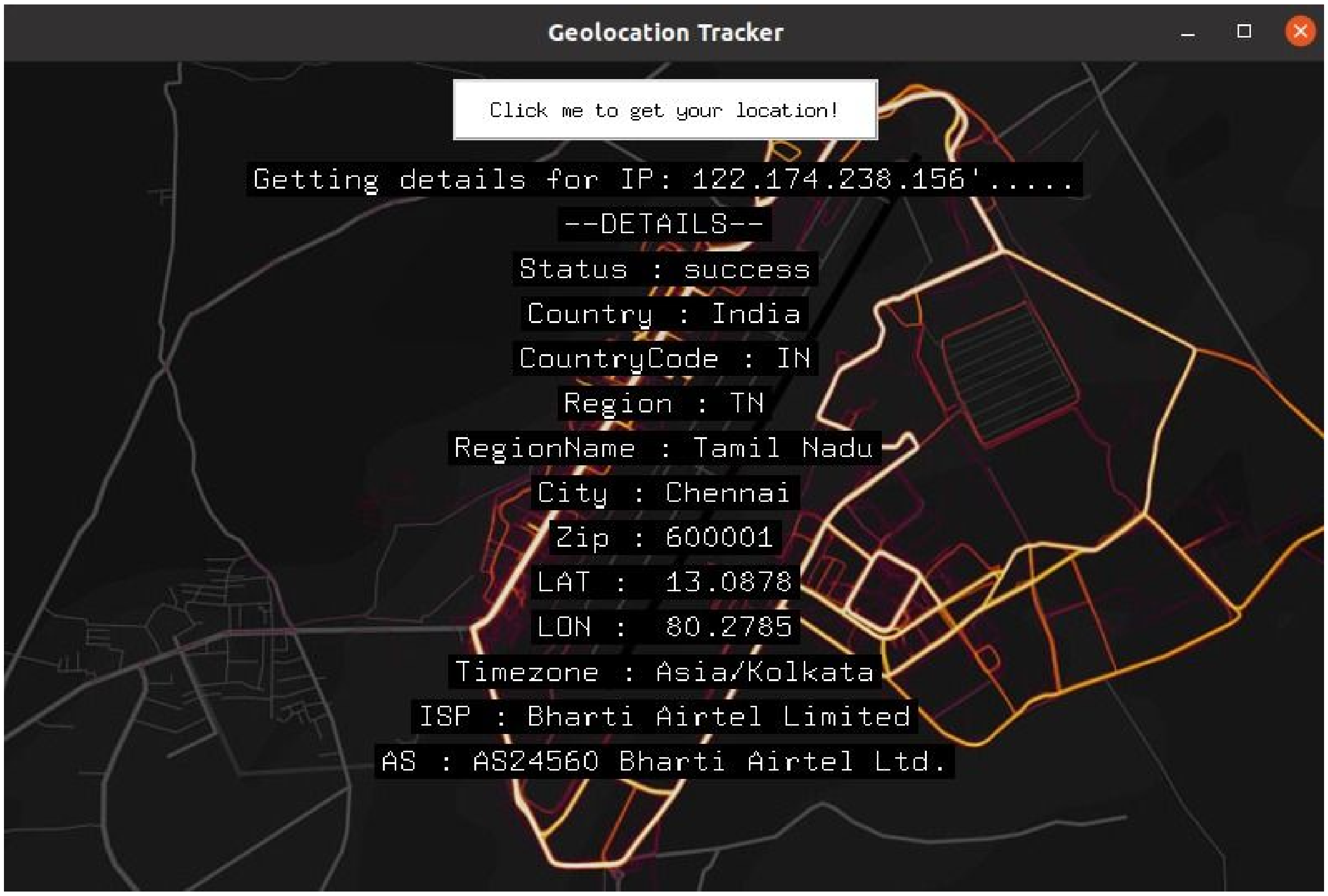
```
(base) pav@Pavendhan-PAV:~/CS/CN/Geolocation Tracker$ python clien
t.py
GUI REQUESTED NEW DATA.....
```

```
(base) pav@Pavendhan-PAV:~/CS/CN/Geolocation Tracker$ python cli
ent.py
```

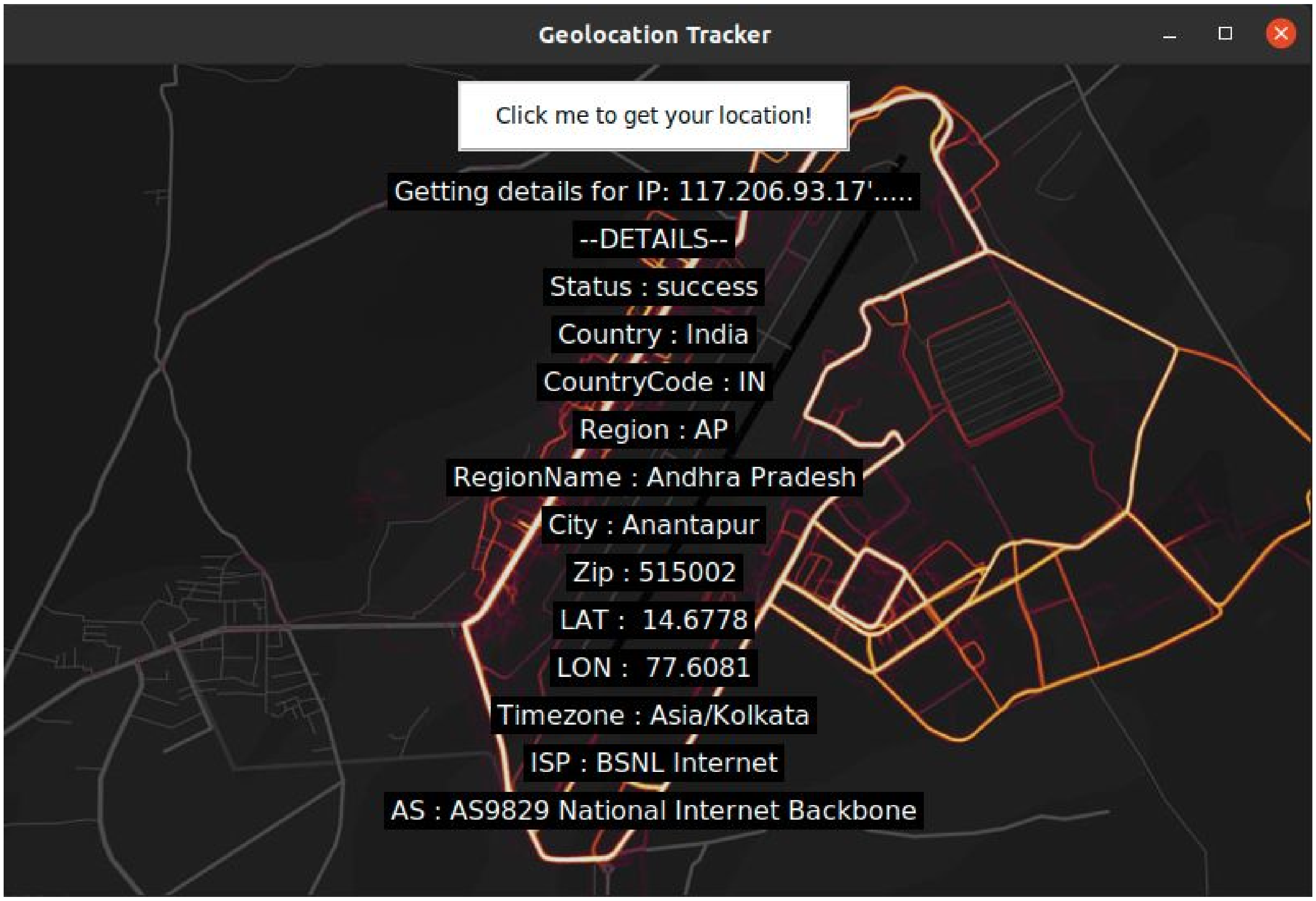
SOME OF THE CLIENT-SIDE OUTPUT (THOSE PRESENT IN THE IPTABLE):



```
Getting details for IP: 117.213.142.87'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : TN
RegionName : Tamil Nadu
City : Chidambaram
Zip : 608602
LAT :  11.3993
LON :  79.6914
Timezone : Asia/Kolkata
ISP : BSNL Internet
AS : AS9829 National Internet Backbone
```

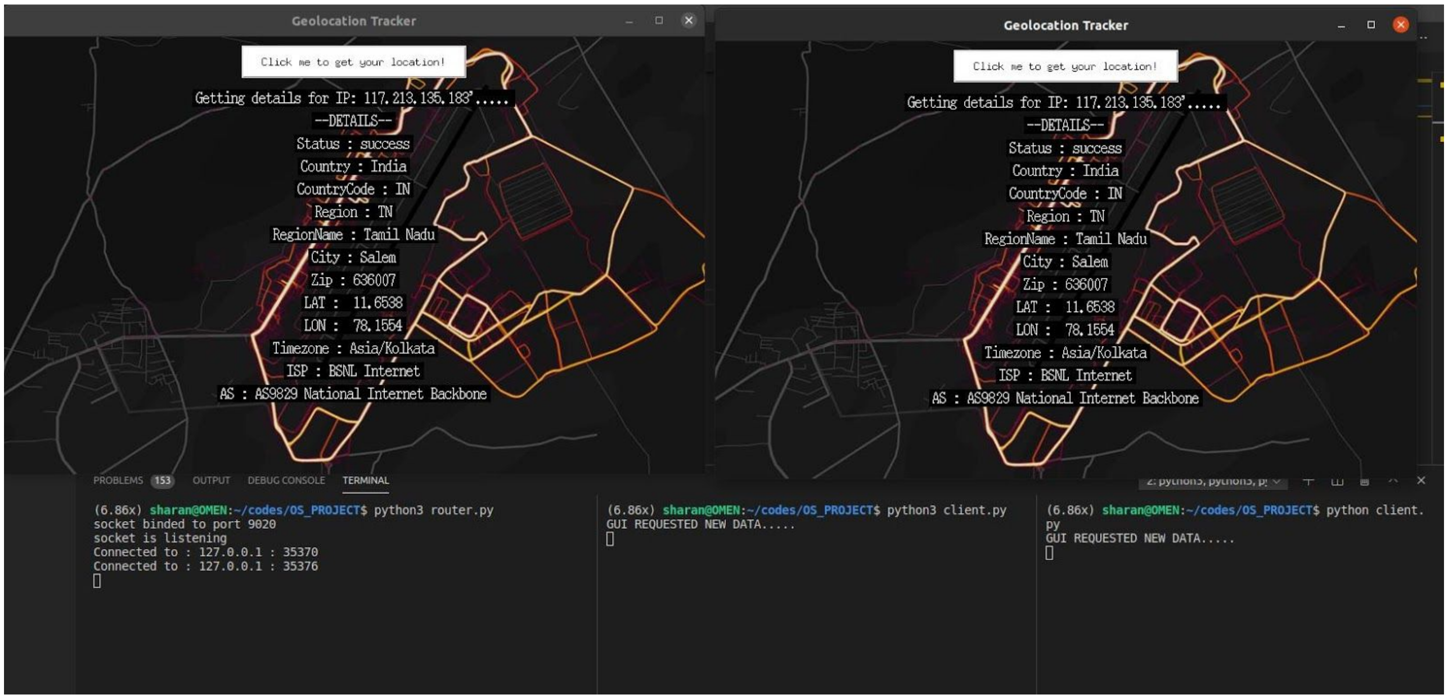


```
Getting details for IP: 122.174.238.156'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : TN
RegionName : Tamil Nadu
City : Chennai
Zip : 600001
LAT :  13.0878
LON :  80.2785
Timezone : Asia/Kolkata
ISP : Bharti Airtel Limited
AS : AS24560 Bharti Airtel Ltd.
```



```
Getting details for IP: 117.206.93.17'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : AP
RegionName : Andhra Pradesh
City : Anantapur
Zip : 515002
LAT :  14.6778
LON :  77.6081
Timezone : Asia/Kolkata
ISP : BSNL Internet
AS : AS9829 National Internet Backbone
```

Router Images:



```
Getting details for IP: 117.213.135.183'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : TN
RegionName : Tamil Nadu
City : Salem
Zip : 636007
LAT :  11.6538
LON :  78.1554
Timezone : Asia/Kolkata
ISP : BSNL Internet
AS : AS9829 National Internet Backbone
```

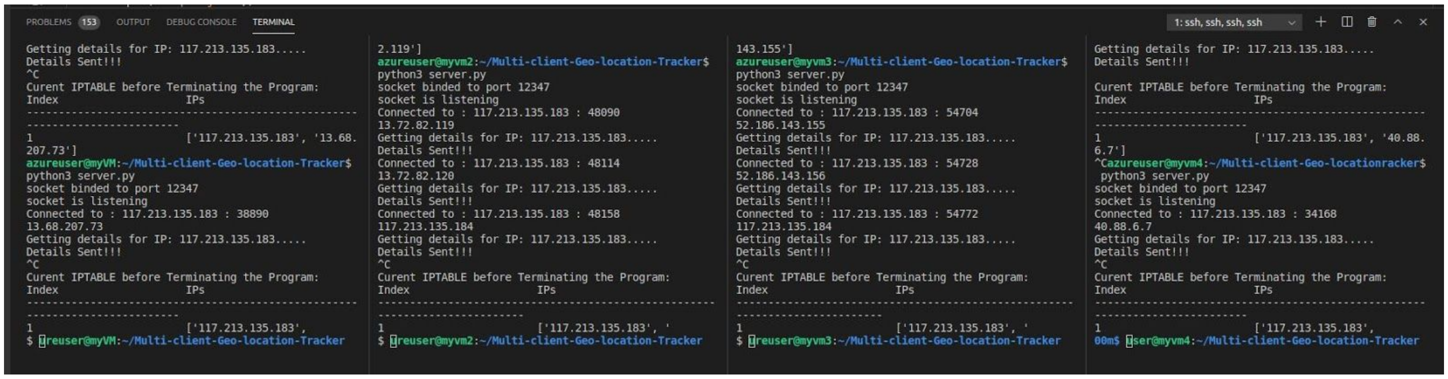
```
Getting details for IP: 117.213.135.183'.....
--DETAILS--
Status : success
Country : India
CountryCode : IN
Region : TN
RegionName : Tamil Nadu
City : Salem
Zip : 636007
LAT :  11.6538
LON :  78.1554
Timezone : Asia/Kolkata
ISP : BSNL Internet
AS : AS9829 National Internet Backbone
```

```
(6.86x) sharan@OMEN:~/codes/OS_PROJECT$ python3 router.py
socket binded to port 9020
socket is listening
Connected to : 127.0.0.1 : 35370
Connected to : 127.0.0.1 : 35376
```

```
(6.86x) sharan@OMEN:~/codes/OS_PROJECT$ python3 client.py
GUI REQUESTED NEW DATA.....
```

```
(6.86x) sharan@OMEN:~/codes/OS_PROJECT$ python client.
py
GUI REQUESTED NEW DATA.....
```

https://drive.google.com/file/d/1vrsh_m9qr7MvHoovORLcgRM-HP32QcoN/view?usp=sharin
g



```
Getting details for IP: 117.213.135.183.....
Details Sent!!!
^C
Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183', '13.68.
207.73']
azureuser@myVM:~/Multi-client-Geo-location-Tracker$
python3 server.py
socket binded to port 12347
socket is listening
Connected to : 117.213.135.183 : 38890
13.68.207.73
Getting details for IP: 117.213.135.183.....
Details Sent!!!
^C
Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183',
$ ureuser@myVM:~/Multi-client-Geo-location-Tracker
```

```
2.119']
azureuser@myvm2:~/Multi-client-Geo-location-Tracker$
python3 server.py
socket binded to port 12347
socket is listening
Connected to : 117.213.135.183 : 48090
13.72.82.119
Getting details for IP: 117.213.135.183.....
Details Sent!!!
Connected to : 117.213.135.183 : 48114
13.72.82.120
Getting details for IP: 117.213.135.183.....
Details Sent!!!
Connected to : 117.213.135.183 : 48158
117.213.135.184
Getting details for IP: 117.213.135.183.....
Details Sent!!!
^C
Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183', '
$ ureuser@myvm2:~/Multi-client-Geo-location-Tracker
```

```
143.155']
azureuser@myvm3:~/Multi-client-Geo-location-Tracker$
python3 server.py
socket binded to port 12347
socket is listening
Connected to : 117.213.135.183 : 54704
52.186.143.155
Getting details for IP: 117.213.135.183.....
Details Sent!!!
Connected to : 117.213.135.183 : 54728
52.186.143.156
Getting details for IP: 117.213.135.183.....
Details Sent!!!
Connected to : 117.213.135.183 : 54772
117.213.135.184
Getting details for IP: 117.213.135.183.....
Details Sent!!!
^C
Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183', '
$ ureuser@myvm3:~/Multi-client-Geo-location-Tracker
```

```
Getting details for IP: 117.213.135.183.....
Details Sent!!!

Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183', '40.88.
6.7']
^Cazureuser@myvm4:~/Multi-client-Geo-locationracker$
 python3 server.py
socket binded to port 12347
socket is listening
Connected to : 117.213.135.183 : 34168
40.88.6.7
Getting details for IP: 117.213.135.183.....
Details Sent!!!
^C
Curent IPTABLE before Terminating the Program:
Index                         IPs
------------------------------------------------------------
-----------------------
1                             ['117.213.135.183',
00m$ user@myvm4:~/Multi-client-Geo-location-Tracker
```

https://drive.google.com/file/d/1QojVmd6vUTcD3BkpMrNFbb7ELmcLgY6X/view?usp=sharin
g

IPTABLE (JSON FILE)

Note -

The screenshot below shows one of the json files after the server had been stopped.Once the server stops ,the json file will look similar to this.For instance, IP-Login-1indicates the first running of the server.For the first time only one client got connected to the server ,where 127.0.0.1 is the client's private ip ,46516 is it's port number and '115.97.93.14.' is the mapped public ip which will be sent to the api to get the details.Similar for the second running of the program and third running of the program IP-Login-2 and IP-Login-3 are the stored iptable respectively.So everytime the program runs,the IP-Login will get updated.

```
{"count": 4, "IP-Login-1": [{"Index": 1, "IP No.": [["127.0.0.1", 46516], ["115.97.93.214", 46516]]}], "IP-Login-2": [{"Index": 1, "IP No.": [["127.0.0.1", 47462], ["115.97.93.214", 47462]]}], "IP-Login-3": [], "IP-Login-4": [{"Index": 1, "IP No.": [["127.0.0.1", 41800], ["122.174.70.249", 41800]]}, {"Index": 2, "IP No.": [["127.0.0.1", 41804], ["122.174.70.250", 41804]]}, {"Index": 3, "IP No.": [["127.0.0.1", 41808], ["122.174.70.251", 41808]]}, {"Index": 4, "IP No.": [["127.0.0.1", 41820], ["122.174.70.252", 41820]]}, {"Index": 5, "IP No.": [["127.0.0.1", 41836], ["122.174.70.253", 41836]]}]}
```

LANGUAGES USED

1. Python

CONCEPTS USED

1. Static Network Address Translation - Public & private IPs
2. Ssh Connection
3. IP Table
4. Client-Server Communication(TCP)
5. Multithreading
6. Critical Section Problem
7. Deadlock Prevention
8. Signal Handling

CODE

SERVER

```
# import socket programming library

import socket


# import thread module

from _thread import *

import threading

import sys

import json

import requests

import csv

import socket

from signal import signal, SIGINT

from sys import exit
```

```
ip=requests.get('https://api.ipify.org').text #public ip

iptable=[]


print_lock = threading.Lock()


def handler(signal_received,frame):

    data = {}

    f = open('sample.json',)

    d = json.load(f)

    f.close()

    d['count']=d['count']+1

    # temp = "IP-Login-"+str(d['count'])

    num = d['count']

    temp = 'IP-Login-' + str(num)

    d[temp]=[]

    print("\nCurent IPTABLE before Terminating the Program:")

    ind=1

    print("Index\t\t\t","IPs")


print("-----------------------------------------------------------------------------")

    if not iptable:

        print("IPTABLE IS EMPTY!!!")

    else:

        for x in iptable:

            d[temp].append({

                'Index':ind,
                'IP No.':x

                })

            print(str(ind),"\t\t\t",x)

            ind+=1

    with open('sample.json', 'w') as outfile:

        json.dump(d, outfile)

    sys.exit()



 def trigger_api(ip):

  querystring = {"ip": ip}

  url =
 "https://geo.ipify.org/api/v1?apiKey=at_LHZrfcs9aoOIVLAmHfZdyGOj9hzKW&ipAddress="+ip


  response =  requests.request('GET', 'http://ip-api.com/json/'+ip)


  if(200 == response.status_code):

    return json.loads(response.text)

  else:

    return None


 def main(ip):


    print("Getting details for IP: ",ip,".....")

    print("Details Sent!!!")

    api_response = trigger_api(ip)
    return api_response


 def splitip():

    global ip

    ip1=ip.split(".")

    temp=int(ip1[3])+1

    if(temp>255):

        temp-=254

    ip1[3]=str(temp)

    ip="."

    ip=ip.join(ip1)



 # thread function

 def threaded(c,addr):

    flag=0

    global ip,iptable

    for i in range(len(iptable)):

        if(iptable[i][0][1]==addr[1] and iptable[i][0][0]==addr[0]):

            flag=1

            ip=iptable[i][1][0]

            break

    if(flag==0): iptable.append([addr,(ip,addr[1])])

    print_lock.release()

    info=main(addr)

    c.send(str(info).encode())
```

```
    c.close()

    splitip()







def Main():

    host = ""


    # reverse a port on your computer

    # in our case it is 12345 but it

    # can be anything

    port = 12347

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.bind((host, port))

    print("socket binded to port", port)


    # put the socket into listening mode

    s.listen(5)

    print("socket is listening")


    # a forever loop until client wants to exit

    while True:


        # establish connection with client
```

```
       c, addr = s.accept()


       # lock acquired by client

       print_lock.acquire()

       print('Connected to :', addr[0], ':', addr[1])


       # Start a new thread and return its identifier

       start_new_thread(threaded, (c,addr))

   s.close()




if __name__ == '__main__':

   signal(SIGINT,handler)

   Main()
```

CLIENT

```
import sys

import json

import requests

import csv

import socket

import ast

from tkinter import *
```

```
def trigger_api(ip):

 querystring = {"ip": ip}

 url =
"https://geo.ipify.org/api/v1?apiKey=at_LHZrfcs9aoOIVLAmHfZdyGOj9hzKW&ipAddress="+ip


 response =  requests.request('GET', 'http://ip-api.com/json/'+ip)


 if(200 == response.status_code):

   return json.loads(response.text)

 else:

   return None




root = Tk()

root.title('Geolocation Tracker')

root.geometry("800x500")


def connectip():

 s = socket.socket()

 port=9020

 s.connect(('127.0.0.1', port))

 api_response=s.recv(1024).decode()

 s.close()

 return api_response


def convert(api_response):
   temp=api_response.split(",")

   temp=[x.strip("'") for x in temp]

   temp=[x.strip("{") for x in temp]

   temp=[x.split(":") for x in temp]


   temp[len(temp)-1][1]=temp[len(temp)-1][1].split(".")

   temp[len(temp)-1][1]=[x.strip("\"} ") for x in temp[len(temp)-1][1]]


temp[len(temp)-1][1]=[temp[len(temp)-1][1][0],temp[len(temp)-1][1][1],temp[len(temp)-
1][1][2],temp[len(temp)-1][1][3]]

   temprem=temp[len(temp)-1][1][3]

   temprem=temp[len(temp)-1][1][3].split("\"")

   temp[len(temp)-1][1][3]=temprem[0]


   iptemp="."

   iptemp=iptemp.join(temp[len(temp)-1][1])

   data={

     'status':temp[0][1][2:],

     'country':temp[1][1][2:],

     'countryCode':temp[2][1][2:],

     'region':temp[3][1][2:],

     'regionName':temp[4][1][2:],

     'city':temp[5][1][2:],

     'zip':temp[6][1][2:],

     'lat':temp[7][1].lstrip("'"),

     'lon':temp[8][1].lstrip("'"),

     'timezone':temp[9][1][2:],
     'isp':temp[10][1][2:],

     'as':temp[12][1][2:],

     'ip':iptemp[1:],

   }

   return data





def location():

 print("GUI REQUESTED NEW DATA.....")

 api_response=connectip()

 api_response=convert(api_response)

 # print("Getting details for IP: " + ip+".....")

 l1.config(text="Getting details for IP: " +api_response['ip']+".....")

 # print("Details:")

 l2.config(text="--DETAILS--")


 l3.config(text='Status : ' + api_response['status'])

 l4.config(text='Country : ' + api_response['country'])

 l5.config(text='CountryCode : ' + api_response['countryCode'])

 l6.config(text='Region : ' + api_response['region'])

 l7.config(text='RegionName : ' + api_response['regionName'])

 l8.config(text='City : ' + api_response['city'])

 l9.config(text='Zip : ' + api_response['zip'])

 l10.config(text='LAT : ' + str(api_response['lat']))

 l11.config(text='LON : ' + str(api_response['lon']))
 l12.config(text='Timezone : ' + api_response['timezone'])

 l13.config(text='ISP : ' + api_response['isp'])

 l15.config(text='AS : ' + api_response['as'])






bg = PhotoImage(file="bg.png")


label=Label(root,image=bg)

label.place(x=0,y=0,relwidth=1,relheight=1)


button = Button(root, text="Click me to get your location!", padx=20, pady=10,
command=location, bg='white')

# button.config(font=("Robotto", 10))

button.pack(pady=10)


l1 = Label(root,text="IP?",fg="white",bg="black",padx='3')

l1.pack(pady=3)

l1.config(font=("Robotto", 12))

l2 = Label(root,text="Details?",fg="white",bg="black",padx='3')

l2.pack(pady=3)

l2.config(font=("Robotto", 12))

l3 = Label(root,text="Status?",fg="white",bg="black",padx='3')

l3.pack(pady=3)

l3.config(font=("Robotto", 12))

l4 = Label(root,text="Country?",fg="white",bg="black",padx='3')
```

```
l4.pack(pady=3)

l4.config(font=("Robotto", 12))

l5 = Label(root,text="CountryCode?",fg="white",bg="black",padx='3')

l5.pack(pady=3)

l5.config(font=("Robotto", 12))

l6 = Label(root,text="Region?",fg="white",bg="black",padx='3')

l6.pack(pady=3)

l6.config(font=("Robotto", 12))

l7 = Label(root,text="RegionName?",fg="white",bg="black",padx='3')

l7.pack(pady=3)

l7.config(font=("Robotto", 12))

l8 = Label(root,text="City?",fg="white",bg="black",padx='3')

l8.pack(pady=3)

l8.config(font=("Robotto", 12))

l9 = Label(root,text="ZIP?",fg="white",bg="black",padx='3')

l9.pack(pady=3)

l9.config(font=("Robotto", 12))

l10 = Label(root,text="LAT?",fg="white",bg="black",padx='3')

l10.pack(pady=3)

l10.config(font=("Robotto", 12))

l11 = Label(root,text="LON?",fg="white",bg="black",padx='3')

l11.pack(pady=3)

l11.config(font=("Robotto", 12))

l12 = Label(root,text="Timezone?",fg="white",bg="black",padx='3')

l12.pack(pady=3)

l12.config(font=("Robotto", 12))
```

```
l13 = Label(root,text="ISP?",fg="white",bg="black",padx='3')

l13.pack(pady=3)

l13.config(font=("Robotto", 12))

l15 = Label(root,text="AS?",fg="white",bg="black",padx='3')

l15.pack(pady=3)

l15.config(font=("Robotto", 12))




root.mainloop()




if __name__ == "__main__":

 # location()

 print("GUI CLOSED")
```

ROUTER

```
# import socket programming library

import socket



# import thread module

from _thread import *

from threading import *
```

```
import threading

import sys

import time

import json

import requests

import csv

import socket

from signal import signal, SIGINT

from sys import exit



sem=Semaphore(4)

ip=requests.get('https://api.ipify.org').text #public ip

iptable=[]

ips=['13.72.82.119','52.186.143.155','13.68.207.73','40.88.6.7']

avail=[1,1,1,1]



print_lock = threading.Lock()
```

```
def handler(signal_received,frame):

    data = {}

    f = open('sample.json',)

    d = json.load(f)

    f.close()

    d['count']=d['count']+1

    # temp = "IP-Login-"+str(d['count'])

    num = d['count']

    temp = 'IP-Login-' + str(num)

    d[temp]=[]

    print("\nCurent IPTABLE before Terminating the Program:")

    ind=1

    print("Index\t\t\t","IPs")


print("------------------------------------------------------------------------------")

    if not iptable:

        print("IPTABLE IS EMPTY!!!")
```

```
   else:

       for x in iptable:

           d[temp].append({

               'Index':ind,

               'IP No.':x

               })

           print(str(ind),"\t\t\t",x)

           ind+=1

   with open('sample.json', 'w') as outfile:

       json.dump(d, outfile)

   sys.exit()









def check():

   index=-1
```

```
    global avail

    for i in range(len(avail)):

        if (avail[i] == 1):

            avail[i]=0

            return i

    return index



def serverconnect(ipt):

    s = socket.socket()

    port=12347

    s.connect((ipt, port))

    ip=requests.get('https://api.ipify.org').text #public ip

    s.send(str(ip).encode())

    api_response=s.recv(1024).decode()

    s.close()

    return api_response
```

```
# thread function

def threaded(c,addr):



   global iptable,ips



   sem.acquire()

   index=check()

   iptable.append([addr,ips[index]])

   info=serverconnect(ips[index])

   c.send(str(info).encode())

   time.sleep(5)

   avail[index]=1

   sem.release()

   c.close()




def Main():
```

```
host = ""



# reverse a port on your computer

# in our case it is 12345 but it

# can be anything

port = 9020

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind((host, port))

print("socket binded to port", port)



# put the socket into listening mode

s.listen(5)

print("socket is listening")



# a forever loop until client wants to exit

while True:
        # establish connection with client

        c, addr = s.accept()



        # lock acquired by client



        print('Connected to :', addr[0], ':', addr[1])



        # Start a new thread and return its identifier

        start_new_thread(threaded, (c,addr))

    s.close()





if __name__ == '__main__':

    signal(SIGINT,handler)

    Main()
```

REFERENCES

1. https://www.whatismyip.com/
2. https://www.geeksforgeeks.org/network-address-translation-nat/