NETWORKING LAB

Gokul P ro

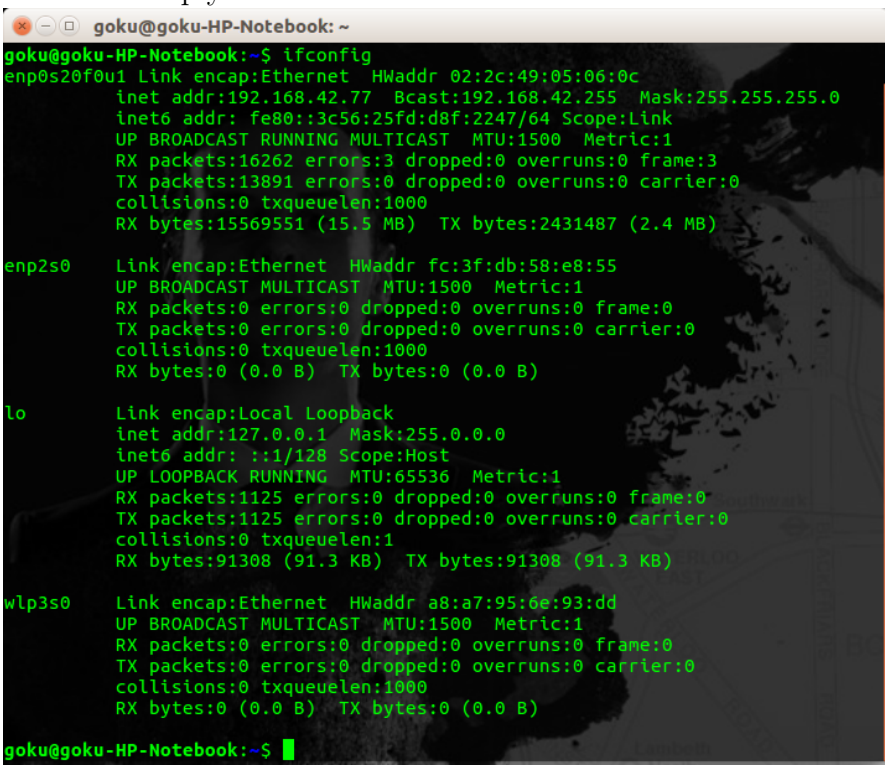
March 13, 2018

Contents

1 EXPERIMENT 1

1.1 ifconfig

ifconfig (interface configurator) command is use to initialize an interface, assign IP Address to interface and enable or disable interface on demand. With this command you can view IP Address and Hardware / MAC address assign to interface and also MTU (Maximum transmission unit) size.
Use ifconfig as either:
ifconfig
This will simply list all information on all network devices currently up.



ifconfig eth0 down
This will take eth0 (assuming the device exists) down, it won't be able to receive or send anything until you put the device back "up" again.

1.1.1 ifup

Use ifup device-name to bring an interface up by following a script (which will contain your default networking settings). Simply type ifup and you will get help on using the script.
For example typing:

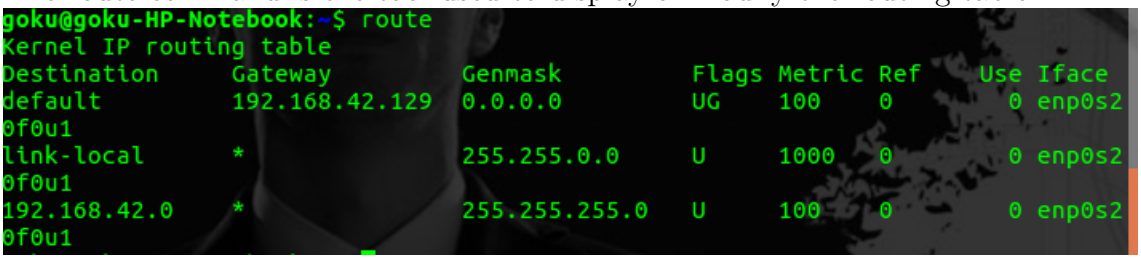
ifup eth0
Will bring eth0 up if it is currently down.

1.1.2 ifdown

Use ifdown device-name to bring an interface down using a script (which will contain your default network settings). Simply type ifdown and you will get help on using the script.
For example typing:
ifdown eth0
Will bring eth0 down if it is currently up.

1.2 route

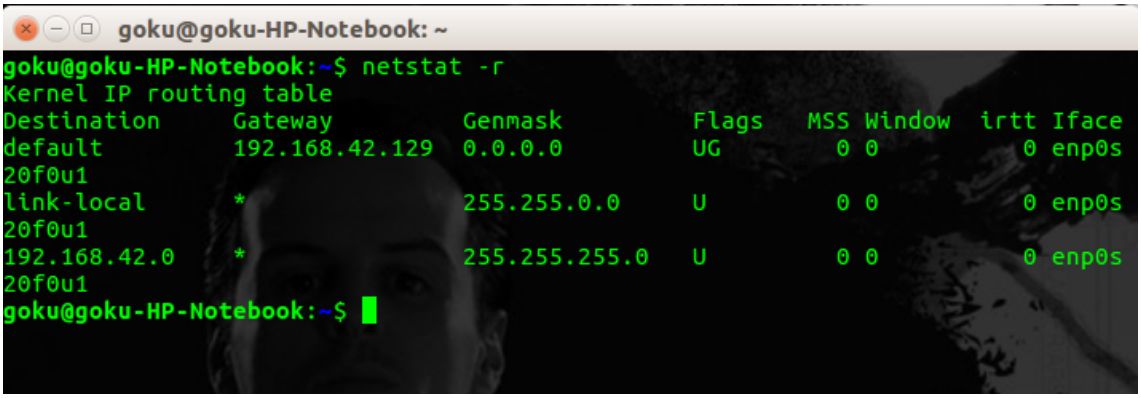
The route command is the tool used to display or modify the routing table.



To add a new route:
route add -net 10.10.10.0/24 gw 192.168.0.1
To delete a route:
route del -net 10.10.10.0/24 gw 192.168.0.1
To add a default gateway:
route add default gw 192.168.0.1

1.3 netstat

Displays contents of /proc/net files. It works with the Linux Network Subsystem, it will tell you what the status of ports are ie. open, closed, waiting, masquerade connections. It will also display various other things. To displays routing table information use option as -r.

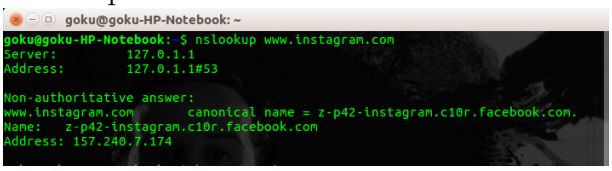


1.4 tcpdump

This is a sniffer, a program that captures packets off a network interface and interprets them for you. It understands all basic internet protocols, and can be used to save entire packets for later inspection.

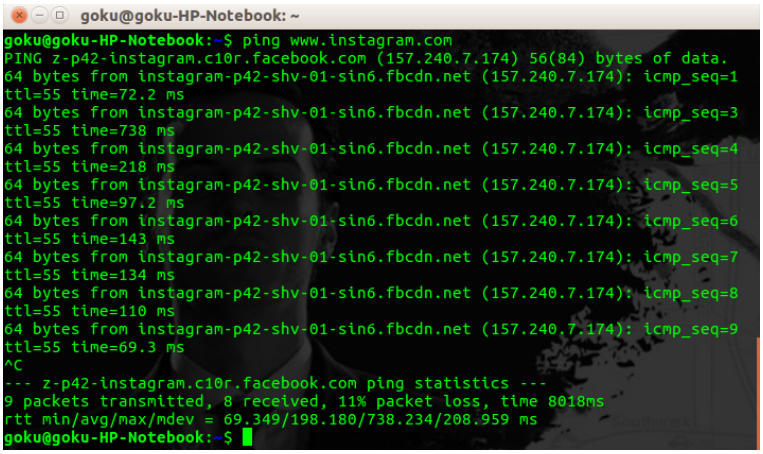
1.5 nslookup

nslookup command also use to find out DNS related query.



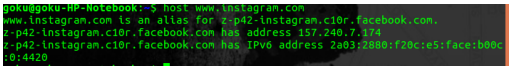
1.6 ping

PING (Packet INternet Groper) command is the best way to test connectivity between two nodes. Whether it is Local Area Network (LAN) or Wide Area Network (WAN). Ping use ICMP (Internet Control Message Protocol) to communicate to other devices The ping command (named after the sound of an active sonar system) sends echo requests to the host you specify on the command line, and lists the responses received their round trip time.



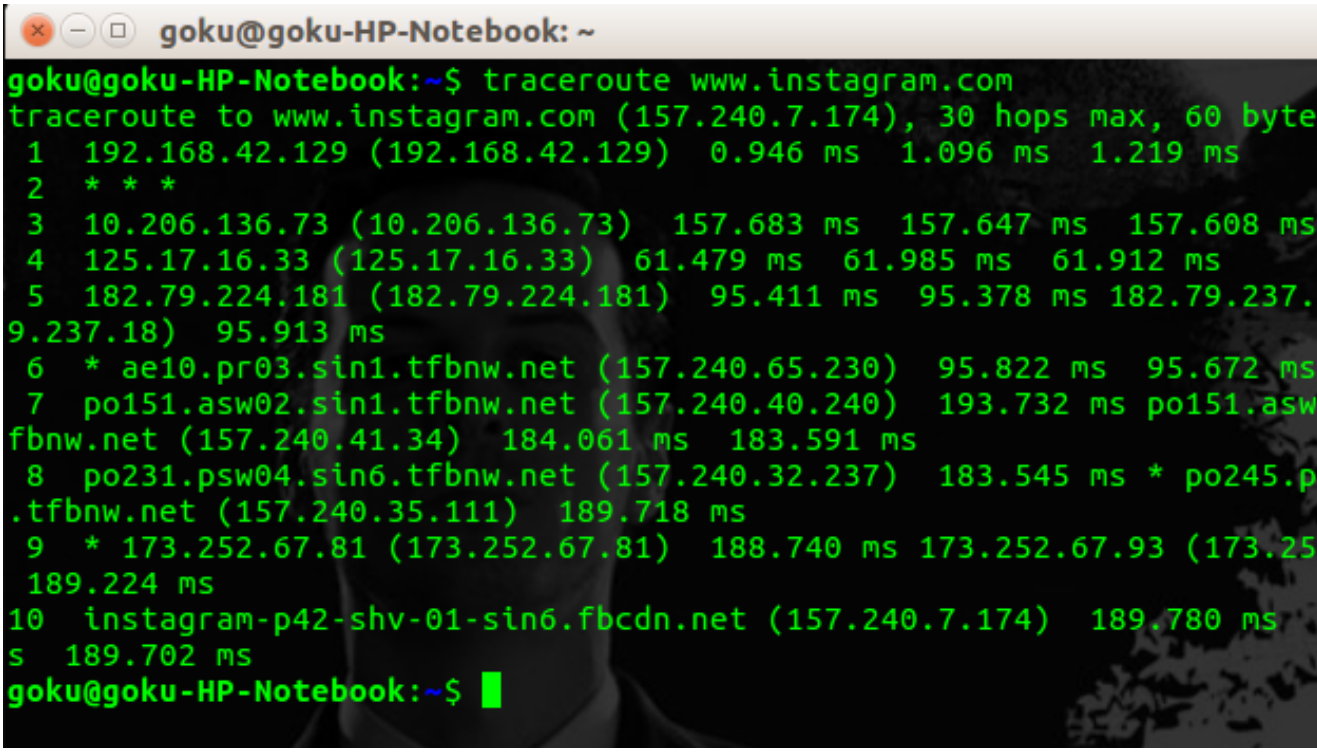
1.7 host

host command to find name to IP or IP to name in IPv4 or IPv6 and also query DNS records.



1.8 traceroute

Traceroute will show the route of a packet. It attempts to list the series of hosts through which your packets travel on their way to a given destination.



1.9 dig

The "domain information groper" tool. More advanced then host. If you give a hostname as an argument to output information about that host, including it's IP address, hostname and various other information.
For example, to look up information about "www.instagram.com" type:
dig www.instagram.com

```
goku@goku-HP-Notebook:~$ dig www.instagram.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.instagram.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32766
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;www.instagram.com.             IN      A

;; ANSWER SECTION:
www.instagram.com.      2550    IN      CNAME   z-p42-instagram.c10r.facebook.
com.
z-p42-instagram.c10r.facebook.com. 12 IN A      157.240.7.174

;; Query time: 99 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Thu Jan 25 21:53:29 IST 2018
;; MSG SIZE  rcvd: 106

goku@goku-HP-Notebook:~$
```

1.10 arp

ARP (Address Resolution Protocol) is useful to view / add the contents of the kernel's ARP tables. To see default table use the command as.

```
goku@goku-HP-Notebook: ~
goku@goku-HP-Notebook:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Ifa
ce
192.168.42.129           ether   f2:34:f3:0d:87:d4   C                     enp
0s20f0u1
```

2 EXPERIMENT 7

2.1 Client-Server communication using Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

Server program:

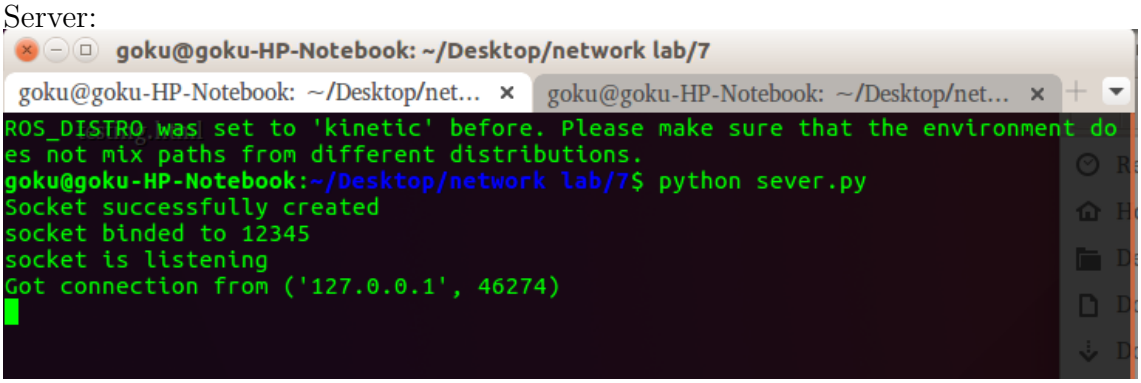
```
# first of all import the socket library
import socket

# next create a socket object
s = socket.socket()
print "Socket successfully created"

# reserve a port on your computer in our
# case it is 12345 but it can be anything
port = 12345

# Next bind to the port
# we have not typed any ip in the ip field
# instead we have inputted an empty string
# this makes the server listen to requests
# coming from other computers on the network
s.bind(('', port))
print "socket binded to %s" %(port)

# put the socket into listening mode
s.listen(5)
print "socket is listening"

# a forever loop until we interrupt it or
# an error occurs
while True:

   # Establish connection with client.
```

```
c, addr = s.accept()
print 'Got connection from', addr

# send a thank you message to the client.
c.send('Thank you for connecting')

# Close the connection with the client
c.close()
```

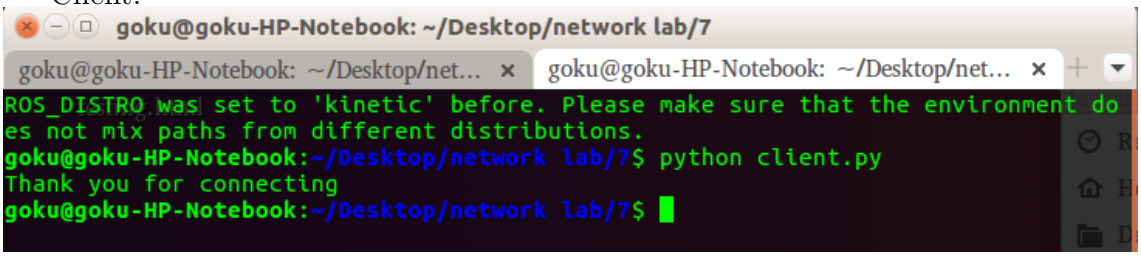
Client program:

```
# Import socket module
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 12345

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server
print s.recv(1024)
# close the connection
s.close()
```

Output :
Server:



Client:



3 EXPERIMENT 8

3.1 Implement Client-Server communication using Socket Programming and UDP as transport layer protocol

In UDP, every time you send a datagram, you have to send the local descriptor and the socket address of the receiving socket along with it. Since TCP is a connection-oriented protocol, on the other hand, a connection must be established before communications between the pair of sockets start. So there is a connection setup time in TCP. In UDP, there is a size limit of 64 kilobytes on datagrams you can send to a specified location, while in TCP there is no limit. Once a connection is established, the pair of sockets behaves like streams: All available data are read immediately in the same order in which they are received. UDP is an unreliable protocol – there is no guarantee that the datagrams you have sent will be received in the same order by the receiving socket. On the other hand, TCP is a reliable protocol; it is guaranteed that the packets you send will be received in the order in which they were sent.

Client program:

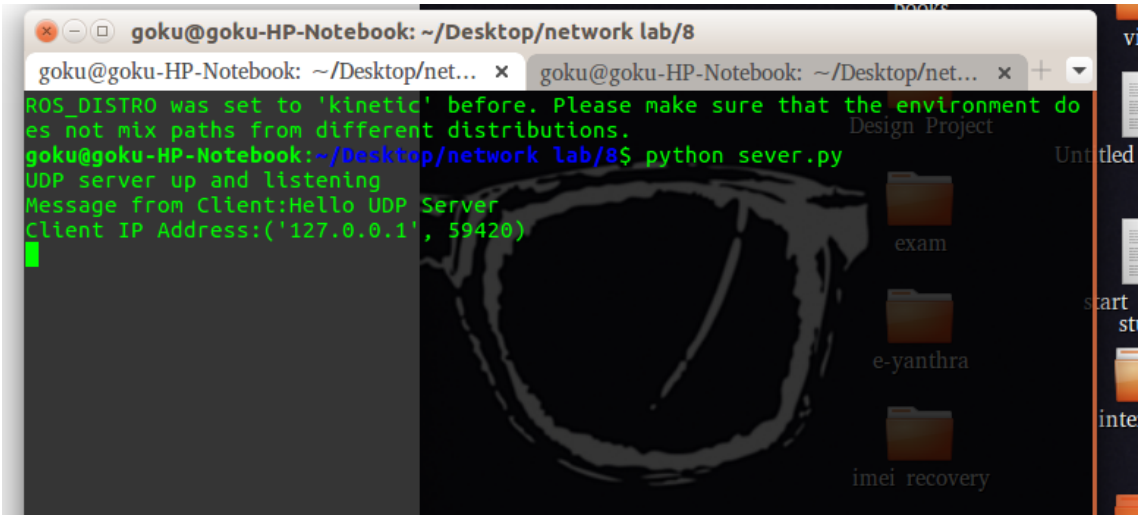
```
import socket


msgFromClient       = "Hello UDP Server"
bytesToSend         = str.encode(msgFromClient)
serverAddressPort   = ("127.0.0.1", 20001)
bufferSize          = 1024


# Create a UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCI


# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend, serverAddressPort)


msgFromServer = UDPClientSocket.recvfrom(bufferSize)


msg = "Message from Server {}".format(msgFromServer[0])
print(msg)
```

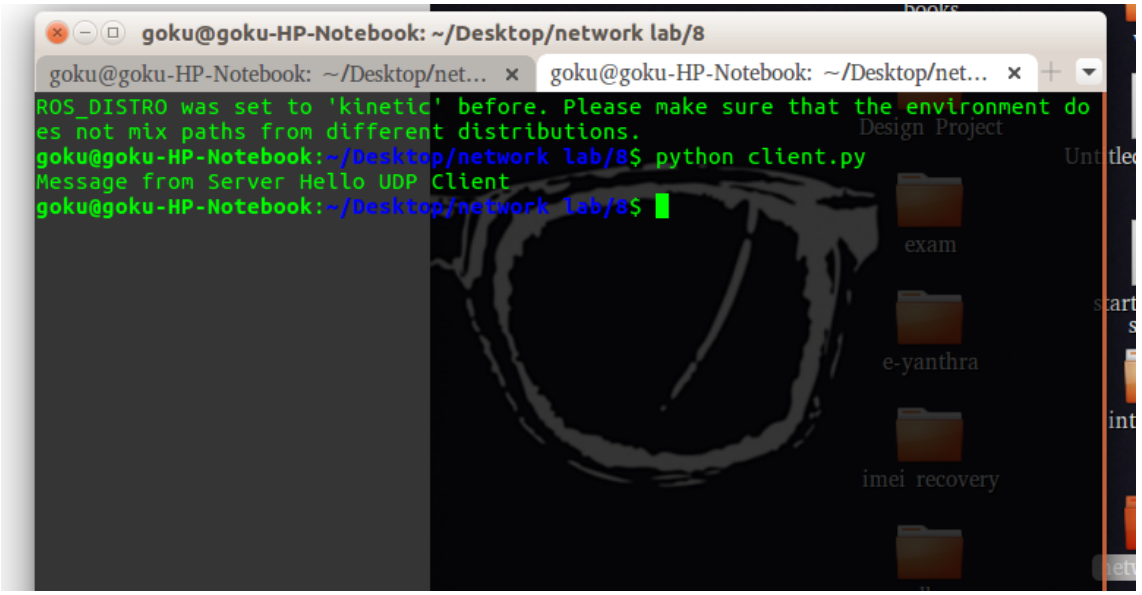
Server program:

```
import socket


localIP     = "127.0.0.1"

localPort   = 20001

bufferSize  = 1024


msgFromServer       = "Hello UDP Client"

bytesToSend         = str.encode(msgFromServer)


# Create a datagram socket

UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCI


# Bind to address and ip

UDPServerSocket.bind((localIP, localPort))


print("UDP server up and listening")


# Listen for incoming datagrams

while(True):
```

```
bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

message = bytesAddressPair[0]

address = bytesAddressPair[1]

clientMsg = "Message from Client:{}".format(message)
clientIP  = "Client IP Address:{}".format(address)

print(clientMsg)
print(clientIP)


# Sending a reply to client

UDPServerSocket.sendto(bytesToSend, address)
```

Output:

Server:8.png 

Client:8.png 

4 EXPERIMENT 9

4.1 Implement a multi user chat server using TCP as transport layer protocol

Server
This server can be set up on a local area network by choosing any on computer to be a server node, and using that computer's private IP address as the server IP address. For example, if a local area network has a set of private IP addresses assigned ranging from 192.168.1.2 to 192.168.1.100, then any computer from these 99 nodes can act as a server, and the remaining nodes may connect to the server node by using the server's private IP address. Care must be taken to choose a port that is currently not in usage. For example, port 22 is default for ssh, and port 80 is default for HTTP protocols. So these two ports preferably, shouldnt be used or reconfigured to make them free for usage
Server:

```
import socket
from thread import *
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
port=1234
#made a list for clients
```

```
list_of_clients=[]
#binding to the port
s.bind(('',port))
def client_thread(conn, addr):
        #recieves name from the client
        name=conn.recv(1024).decode()
        print name+" joined chat room"
        welcome="welcome to this chat room"
        data=''
        #server sends the welcome note to the client
        conn.send(welcome.encode())
        while True and data!="bye":
                try:
                        #server recieves
                        data=conn.recv(1024).decode()
                        if not data:
                                break;
                        #prints on server terminal
                        message=str(name)+"->"+str(data)
                        print message
                        #broadcasts the message
                        broadcast(conn,message)
                except:
                        continue
        #server sends bye to the client
        conn.send(data.encode())
        conn.close()
        list_of_clients.remove(conn)
def broadcast(conn,data):
        #iterates through all connections in list of clients
        #skips the connection which sended the message
        for connection in list_of_clients:
                if connection!=conn:
                        #print "i reached broadcast"
                        try:
                                connection.send(data.encode())
                        except:
                                #if connection fails closes connection
                                connection.close()
                                list_of_clients.remove(connection)
                else:
```

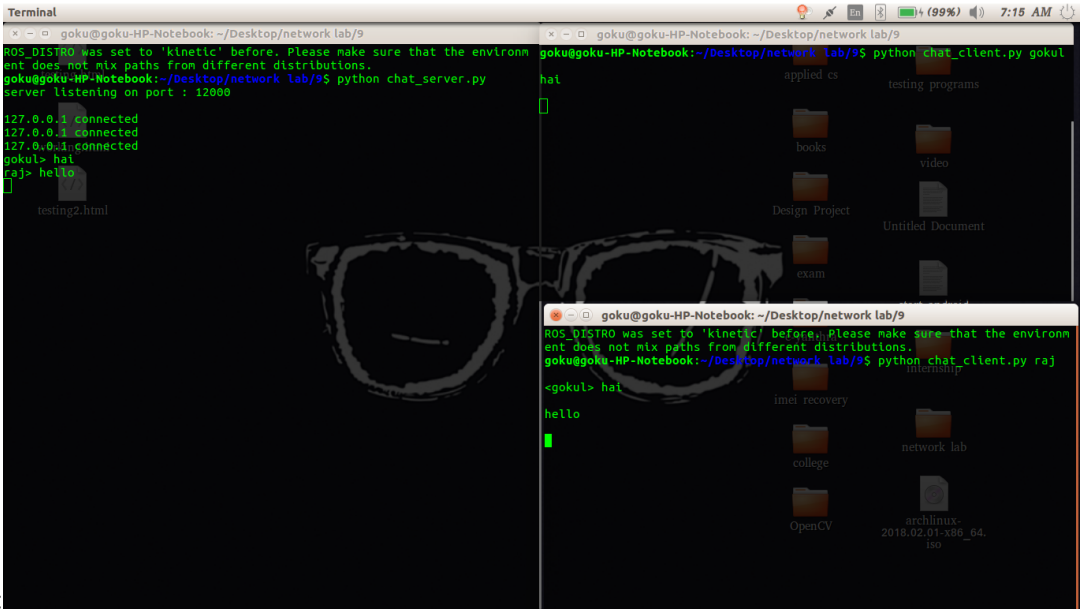
```
                        continue
#main starts here
s.listen(10)
while True:
        #server accepts a connection
        conn,addr=s.accept()
        #appends to the list
        list_of_clients.append(conn)
        #starting a new thread with ths connection
        start_new_thread(client_thread,(conn,addr))
s.close()
```

Client:
The client side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. If the input is from the user, it sends the message that the users enters to the server for it to be broadcasted to other users.

```
import socket
import thread
import sys
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
port=1234
hostname='127.0.0.1'
def listening(server):
        data=''
        while True and data!='bye':
                #print "i reached listening"
                data=server.recv(1024).decode()
                print data
        quit()
s.connect((hostname,port))
name=raw_input('name:')
s.send(name.encode())
data=s.recv(1024).decode()
print "server:"+data
thread. start_new_thread(listening,(s,))
while True:
        data=raw_input()
```

```
            s.send(data.encode())
s.close()
quit()
```

Output :

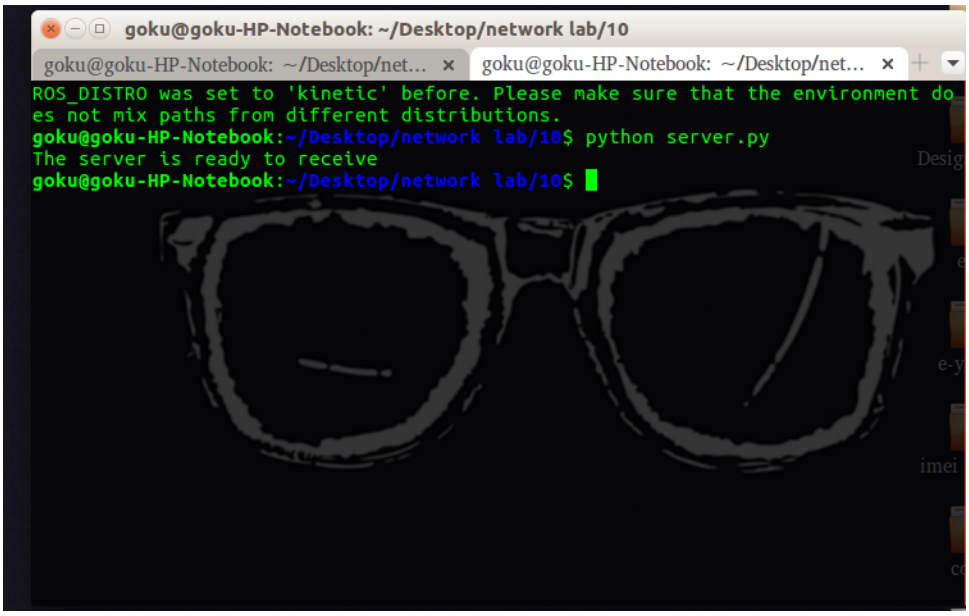


5 EXPERIMENT 10

5.1 Implement Concurrent Time Server application using UDP to execute the program at remoteserver. Client sends a time request to the server, server sends its system time back to the client. Client displays the result

Concurrent servers are designed using three basic mechanisms.Process-based using fork Event-based using I/O Multiplexing Spawn one server process to handle each client connection Spawn one server process to handle each client connection Kernel automatically interleaves Each server process has its own private address space One process, one thread, but programmer manually interleaves multiple connections Relies on lower-level system abstractions Create one server thread to handle each client connection Kernel automatically interleaves multiple server threads All threads share the same address space Server:

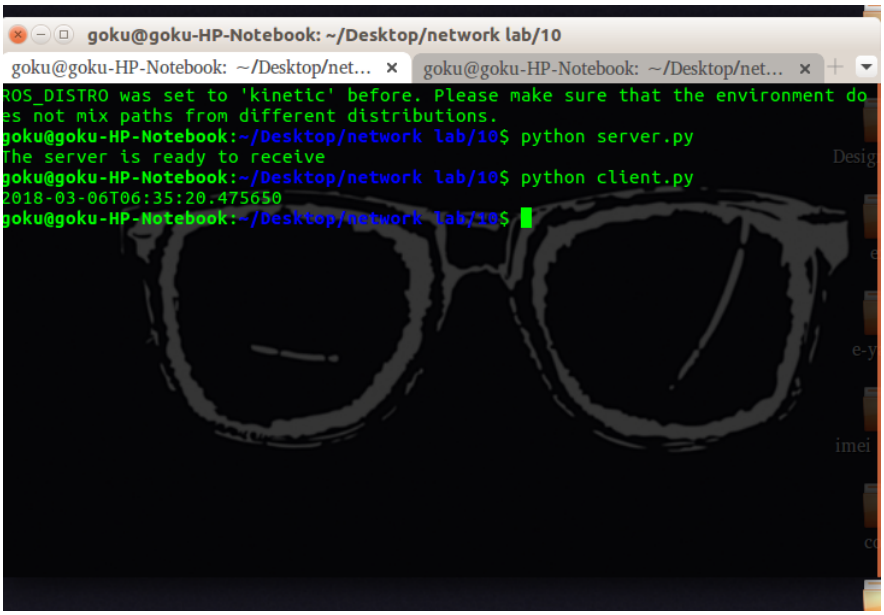
```
import socket
import datetime
from thread import *
import time
port=1234
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind(('',port))
def sendtime(sock, addr):
        print "u reached send time"
        time.sleep(8)
        tim=datetime.datetime.now()
        sock.sendto(str(tim),addr)

        return
while True:
        print "Server is ready \n"
        message,addr=s.recvfrom(1024)
        start_new_thread(sendtime,(s,addr))
```



Client :

```
import socket
port=1234
hostname='127.0.0.1'
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

```
print "sending time request\n"
message="give time"
s.sendto(message,(hostname,port))
message,addr=s.recvfrom(1024)
print "time is :"+str(message)\newline
```



6 EXPERIMENT 13

6.1 Implement Simple Mail Transfer Protocol

The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

SMTP stands for Simple Mail Transfer Protocol.

The smtplib modules is useful for communicating with mail servers to send mail.

Sending mail is done with Python's smtplib using an SMTP server.

Actual usage varies depending on complexity of the email and settings of the email server, the instructions here are based on sending email through Gmail.

Mail sending program:

```
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
import smtplib

fromaddr = "g.gokuldas68@gmail.com"
toaddr = "gokulpulikkal@cet.ac.in"
msg = MIMEMultipart()
msg['From'] = fromaddr
msg['To'] = toaddr
msg['Subject'] = "Python email"

body = "Python test mail"
msg.attach(MIMEText(body, 'plain'))

server = smtplib.SMTP('smtp.gmail.com', 587)
server.ehlo()
server.starttls()
server.ehlo()
server.login("g.gokuldas68@gmail.com", "password")
text = msg.as_string()
server.sendmail(fromaddr, toaddr, text)
```

7 EXPERIMENT 14

7.1 Develop concurrent file server

Aim : Develop concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with file or the message.

steps for creating server :
1.create socket and bind it to the port with socket bind function
2.accept connections from clients with socket accept function
3.if got a connection then start a new thread of function 'client thread' with startnewthread function in built in function thread
4.in 'client thread' function first recieve the name of file and check the existence of file with os.path.isfile() function.
5.if file exist then read file content and send it to the client . if file not exist then send that message to the client

steps for creating client :
1.after getting connection from server send the file name
2.if positive acknowledge is got from the server then recieve data from server and write to the file in client directory
3.if negative acknowledge is got from server then print file does not exist and terminate

server program:

```
import socket
import os.path
from thread import *
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
port=1234
s.bind(('',port))
s.listen(2)
def clientthread(conn, s):
        data=conn.recv(1024)                    #receives file name
        if os.path.isfile(str(data)):   #cjecks avaialability of file
                d="s"
                conn.send(d)                            #sends to clie
                f=open(str(data),'r')           #opening file in read
                data=f.read(1024)
                while data:
                        conn.send(data)                 #sends read dt
                        data=f.read(1024)
                print "file send !"                     #shows full fi
                f.close()                                       #close
        else:
                print "no such file exist"      #only when file not ex
                d="n"
                conn.send(d)
        conn.close()
        s.close()
while True:
        print "server is ready"
        #receives connection from clients
        conn, addr=s.accept()
        start_new_thread(clientthread,(conn,s))
```

Client program :

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
port=1234
host='127.0.0.1'
s.connect((host,port))
data=raw_input('enter file name :')
s.send(data)                    #sending file name
d=s.recv(1024)                  #receives acknowledgment
if str(d)!="n":
        f=open(data,'w')        #opens same file in write mode
        data=s.recv(1024)       #
        while data:
                f.write(data)
                data=s.recv(1024)
        f.close()
        print "file recieved"
else:
        print "no such file in server"
```

8 EXPERIMENT 19

8.1 Install network simulator NS-2

NS2 is an open-source simulation tool that runs on Linux. It is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks.
Step 1:
Copy downloaded ns2 file to /opt folder by following command
cp /home/username/Downloads/ns-allinone-2.35.tar.gz /opt/
Step 2:
Install prerequisites
Type following commands on terminal
1.sudo apt-get update
2.sudo apt-get dist-upgrade
3.sudo apt-get update
4.sudo apt-get gcc
5.sudo apt-get install build-essential autoconf automake
6.sudo apt-get install tcl8.5-dev tk8.5-dev
7.sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev
Step 3:
Extract ns2
Type following commands on terminal
1.tar -zxvf ns-allinone-2.35.tar.gz
2.cd ns-allinone-2.35
3../install
Step 4:
Open bashrc file to Set the Environment Variables
Type following commands on terminal
1.sudo gedit /.bashrc

Copy the following lines at the end of the file.

```
# LD_LIBRARY_PATH
OTCL_LIB=/opt/ns-allinone-2.35/otcl-1.14/
NS2_LIB=/opt/ns-allinone-2.35/lib/
USR_Local_LIB=/usr/local/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$US
```

```
# TCL_LIBRARY
TCL_LIB=/opt/ns-allinone-2.35/tcl8.5.10/library/
USR_LIB=/usr/lib/
export TCL_LIBRARY=$TCL_LIBRARY:$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/opt/ns-allinone-2.35/xgraph-12.2/:/opt/ns-allinone-2.3
NS=/opt/ns-allinone-2.35/ns-2.35/
NAM=/opt/ns-allinone-2.35/nam-1.15/
export PATH=$PATH:$XGRAPH:$NS:$NAM
#    -
```

Type following commands on terminal
source /.bashrc