

NAME: KEERTHISRI.D

DATE:18.08.2025

ROLL NO:241901047

EXERCISE-1

BASIC LINUX COMMANDS

1. *dir* - Equivalent of ls
2. *Synopsis:* dir [path] [[options]]
3. *Description:* List all files and directories.
4. *cd* - Change directory

Synopsis: cd [path]

Description: Changes the current working directory.

5. *cls* - Clear screen

Synopsis: cls

Description: Removes all text.

6. *echo* - Prints

Synopsis: echo [message]

Description: Displays messages, useful in batch scripts.

7. *dir -a* (like ls -a)

Synopsis: dir -a [path]

Description: Lists files and directories.

8. *mkdir* - make directory

Synopsis: mkdir [drive:] [path] foldername

Description: Create a new directory in a specified path.

9. *del* - delete file (remdir)

Synopsis: del file.txt

Description: Delete files or directories from the file system.

10. *tasklist* - display all currently running process

Synopsis: tasklist [options]

Description: Lists process name, PID (Process ID), session name/ID, and memory usage.

11. *find* - search for a specific string of text within files

Synopsis: find "string" [filename]

Description: Looks for the given "string" inside the specified file.

12. *systeminfo* - detailed system configuration information

Synopsis: systeminfo

Description: Often used for system audits and troubleshooting.

13. *typeperf* - display or log performance counter data

Synopsis: typeperf [counter...] [options]

Description: Can display output in the console or save it to a log file (CSV, TSV, binary).

14. *tracert* - Trace the path packets take to reach

Synopsis: tracert [options]

Description: Displays each router packets travel through until reaching the destination.

15. *ping* - network connectivity

Synopsis: ping [option]

Description: Displays packet loss, response time, and TTL (Time to Live).

16. *ipconfig* - displays and manages IP address

Synopsis: ipconfig [options]

Description: Shows current network adapter configuration.

EXERCISE-2

TCP CLIENT-SERVER USING SOCKET PROGRAMMING IN PYTHON

AIM:

To implement a simple client–server communication system using TCP sockets in Python, where the client sends a message to the server, and the server echoes (sends back) the same message to the client.

ALGORITHM:

SERVER:

- 1.Create a TCP socket and bind it to 'localhost' at port 55555.
- 2.Listen for incoming client connection requests (up to 3 queued).
- 3.Accept a client connection and receive a message from the client.
- 4.Print the client's message and send the same message back as a reply.
- 5.Ask the server operator if they want to continue; if not, close the server

CLIENT:

- 1.Create a TCP Socket
2. Connect the socket to the server at 'localhost' on port 55555.
3. Take input from the user as a message.
4. Send the encoded message to the server through the socket.
- 5.Receive and decode the server's response, then print it.

CODE:

SERVER:

```
import socket

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

print('Socket Created')

sockfd.bind(('localhost',55555))

sockfd.listen(3)

print('Waiting for connections')

while True:

    clientfd,addr=sockfd.accept()

    receivedMsg=clientfd.recv(1024).decode()

    print("Connected with ",addr)

    print("Message Received from Client: ",receivedMsg)

    clientfd.send(bytes(receivedMsg,'utf-8'))

    print("Message reply sent to Client!")

    print("Do you want to continue(type y or n):")

    choice=input()

    if choice=='n':

        break
```

CLIENT:

```
import socket

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

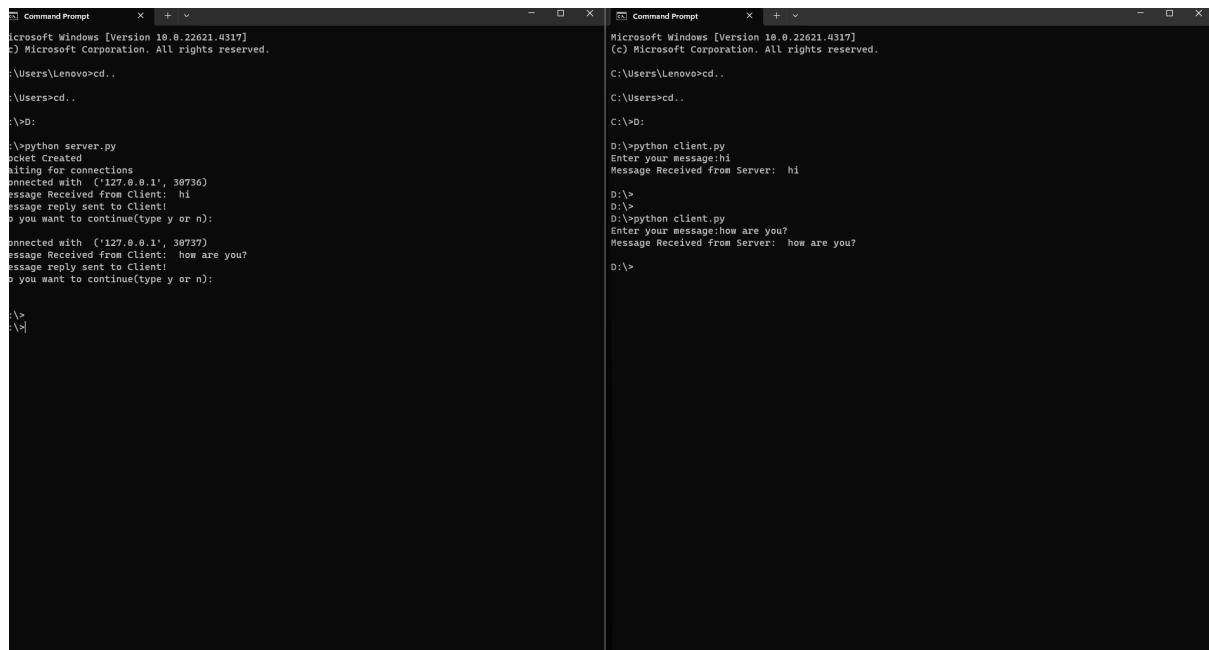
clientfd.connect(('localhost',55555))

name=input("Enter your message:")

clientfd.send(bytes(name,'utf-8'))

print("Message Received from Server: ",clientfd.recv(1024).decode())
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd..
C:\Users>cd..
C:\>D:
C:\>python server.py
Socket Created
Listening for connections
Connected with ('127.0.0.1', 30736)
Message Received from Client: hi
Message reply sent to Client!
Do you want to continue(type y or n):
Connected with ('127.0.0.1', 30737)
Message Received from Client: how are you?
Message reply sent to Client!
Do you want to continue(type y or n):
C:\>
C:\>

Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd..
C:\Users>cd..
C:\>D:
D:\>python client.py
Enter your message:hi
Message Received from Server: hi
D:\>
D:\>python client.py
Enter your message:how are you?
Message Received from Server: how are you?
D:\>
```

RESULT:

The server and client successfully establish a TCP connection. The message sent by the client is received and echoed back by the server correctly.

Roll No: 241901047

Name: Keerthisri D

Exercise-3

UDP CLIENT- SERVER COMMUNICATION USING SOCKET PROGRAMMING IN PYTHON

AIM:

Server:

1. Start the program.
- 2.Import the socket module.
- 3.Create a UDP socket using `socket.AF_INET` and `socket.SOCK_DGRAM`.
- 4.Bind the socket to an IP address and port.
- 5.Receive a message from the client.
- 6.Send a reply back to the client.
- 7.Close the socket.

Client:

1. Start the program.
- 2.Import the socket module.
3. Create a UDP socket using `socket.AF_INET` and `socket.SOCK_DGRAM`.
- 4.Send a message to the server using the server's IP and port. 5.Receive the server's reply.
- 6.Display the reply message.
- 7.Close the socket.

CODE:

SERVER:

```
import socket
sockfd = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) print("UDP Socket Created")
sockfd.bind(('localhost', 55555))
print("Waiting for messages on localhost:55555")
while True:
    data, addr = sockfd.recvfrom(1024)
    message = data.decode('utf-8')
    print(f"Received from {addr}: {message}")
    sockfd.sendto(data, addr)
    print(f"Echoed back to {addr}")
choice = input("Continue receiving? (y/n): ")
if choice.lower() == 'n':
    break
sockfd.close()
print("Server shutdown")
```

CLIENT:

```
import socket
clientfd = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) server_addr = ('localhost',
55555)
msg = input("Enter your message: ")
clientfd.sendto(msg.encode('utf-8'), server_addr)
data, _ = clientfd.recvfrom(1024)
```

```
print("Echo from server:",  
data.decode('utf-8')) clientfd.close()
```

OUTPUT

:

Server:

```
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Install the latest PowerShell for new features and improvements! https://aka.ms/PowerShell  
  
PS C:\Users\user> cd C:\Users\user\OneDrive\Documents  
PS C:\Users\user\OneDrive\Documents> python server.py  
Server is listening...  
Connected by ('127.0.0.1', 65189)  
Client says: Hello UDP Server  
PS C:\Users\user\OneDrive\Documents> |
```

Client:

```
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows  
  
PS C:\Users\user> cd C:\Users\user\OneDrive\Documents  
PS C:\Users\user\OneDrive\Documents> python client.py  
Enter message to send to server: Hello UDP Server  
Server says: Server received your message  
PS C:\Users\user\OneDrive\Documents> |
```

Result:

Thus, UDP server-client communication was successfully implemented using python.

NAME:KEERTHISRI.D

DATE:25.08.25

ROLL NO:241901047

EXERCISE-4

DEVELOP A CUSTOMIZED PING COMMAND TO TEST THE SERVER CONNECTIVITY

AIM:

To develop a customized ping command that tests the connectivity of a server by sending ICMP echo requests and receiving echo replies, measuring the response time and network availability status. This helps verify if the server is reachable and the network path is functional.

ALGORITHM:

- Set host ,port and numbers of pings.
- Create an empty list for RTT values.
- For each ping attempt:
 - 1.Start timer,connect to host,stop timer.
 - 2.Calculate RTT and store it,else print timeout
- After all attempts,display min,max and avg RTT.

CODE:

```
import socket
import time

host = "google.com"    # you can change this
port = 80               # HTTP port
count = 4               # number of pings

for i in range(count):
    try:
        s = socket.socket()
        start = time.time()
        s.connect((host, port))
        end = time.time()
```

```
s.close()
print(f"Reply from {host}: time={(end-start)*1000:.2f} ms")
except Exception:
    print("Request timed out")
```

Customized Ping Program to Measure Min, Max, and Average RTT

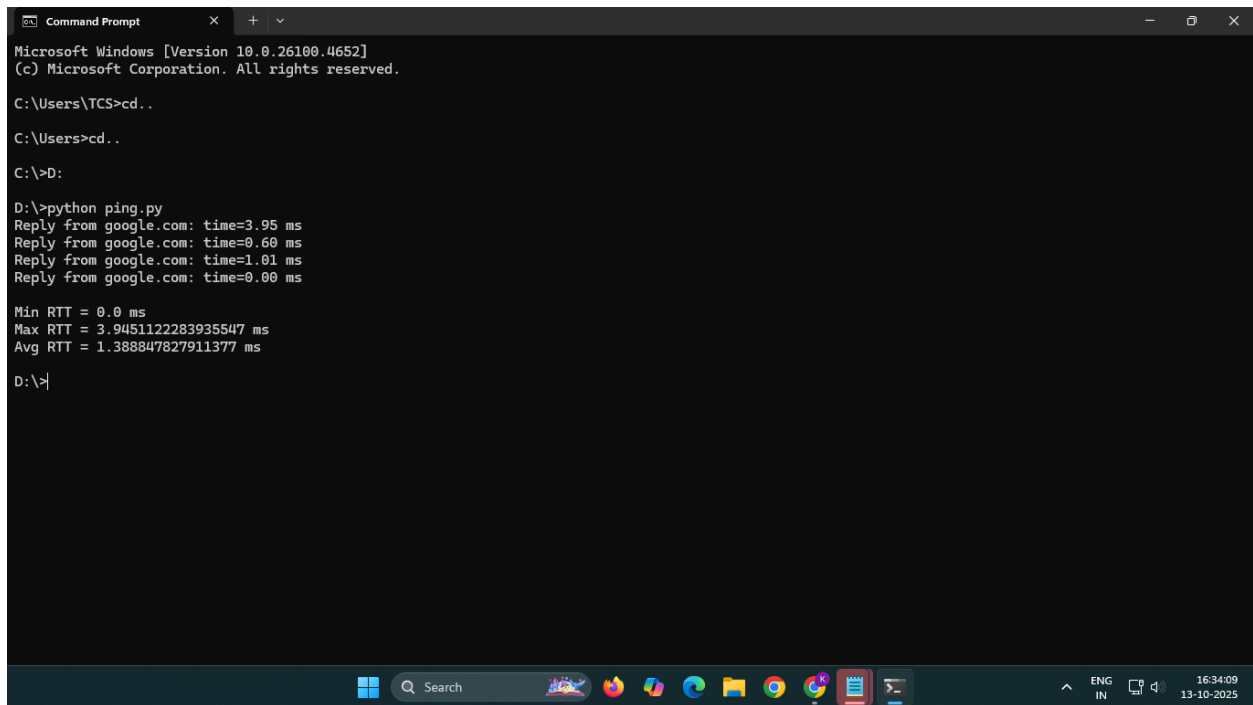
```
import socket, time

host = "google.com"
port = 80
count = 4
times = []

for i in range(count):
    try:
        s = socket.socket()
        start = time.time()
        s.connect((host, port))
        end = time.time()
        s.close()
        rtt = (end - start) * 1000
        times.append(rtt)
        print(f"Reply from {host}: time={rtt:.2f} ms")
    except:
        print("Request timed out")

if times:
    print("\nMin RTT =", min(times), "ms")
    print("Max RTT =", max(times), "ms")
    print("Avg RTT =", sum(times)/len(times), "ms")
```

OUTPUT:

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window content shows the following text:

```
Microsoft Windows [Version 10.0.26100.4652]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TCS>cd..

C:\Users>cd..

C:\>D:

D:\>python ping.py
Reply from google.com: time=3.95 ms
Reply from google.com: time=0.60 ms
Reply from google.com: time=1.01 ms
Reply from google.com: time=0.00 ms

Min RTT = 0.0 ms
Max RTT = 3.9451122283935547 ms
Avg RTT = 1.388847827911377 ms

D:\>|
```

The taskbar at the bottom shows the Windows Start button, a search bar, and several application icons including Firefox, Chrome, and File Explorer. The system tray on the right shows the language "ENG IN", a network icon, and the date and time "16:34:09 13-10-2025".

RESULT:

The program effectively meets the objective of testing server connectivity using multiple network diagnostics methods. It enhances traditional connectivity checks and is suitable for real-world troubleshooting and monitoring tasks.

NAME:KEERTHISRI.D

DATE:26.08.25

ROLL NO:241901047

EXERCISE-5

DEVELOP A SIMPLE CALCULATOR USING XMLRPC

AIM:

To develop a simple calculator application using XML-RPC (XML Remote Procedure Call) in Python that allows a client to perform basic arithmetic operations (addition, subtraction, multiplication, division) by calling functions on a remote server.

ALGORITHM:

SERVER:

- Create functions for add, subtract, multiply, and divide.
- Set up an XML-RPC server on a specified host and port.
- Register the arithmetic functions with the server.
- Keep the server running to listen for client requests.

CLIENT:

- Connect to the XML-RPC server using its URL.
- Request an arithmetic operation by calling the corresponding remote function with two numbers.
- Receive the result from the server.
- Display the result.

CODE:

SERVER:

```
from xmlrpc.server import SimpleXMLRPCServer
```

```
def add(x, y):
```

```
    return x + y
```

```
def subtract(x, y):
```

```
    return x - y

def multiply(x, y):

    return x * y

def divide(x, y):

    if y == 0:

        return "Error: Division by zero"

    return x / y

server = SimpleXMLRPCServer(("localhost", 8000))

print("Listening on port 8000...")


# Register functions

server.register_function(add, 'add')

server.register_function(subtract, 'subtract')

server.register_function(multiply, 'multiply')

server.register_function(divide, 'divide')


# Run the server's main loop

server.serve_forever()

CLIENT:

import xmlrpc.client


# Connect to the server
```

```
proxy = xmlrpc.client.ServerProxy("http://localhost:8000/")
```

Example usage

```
print("Add: 5 + 3 =", proxy.add(5, 3))
```

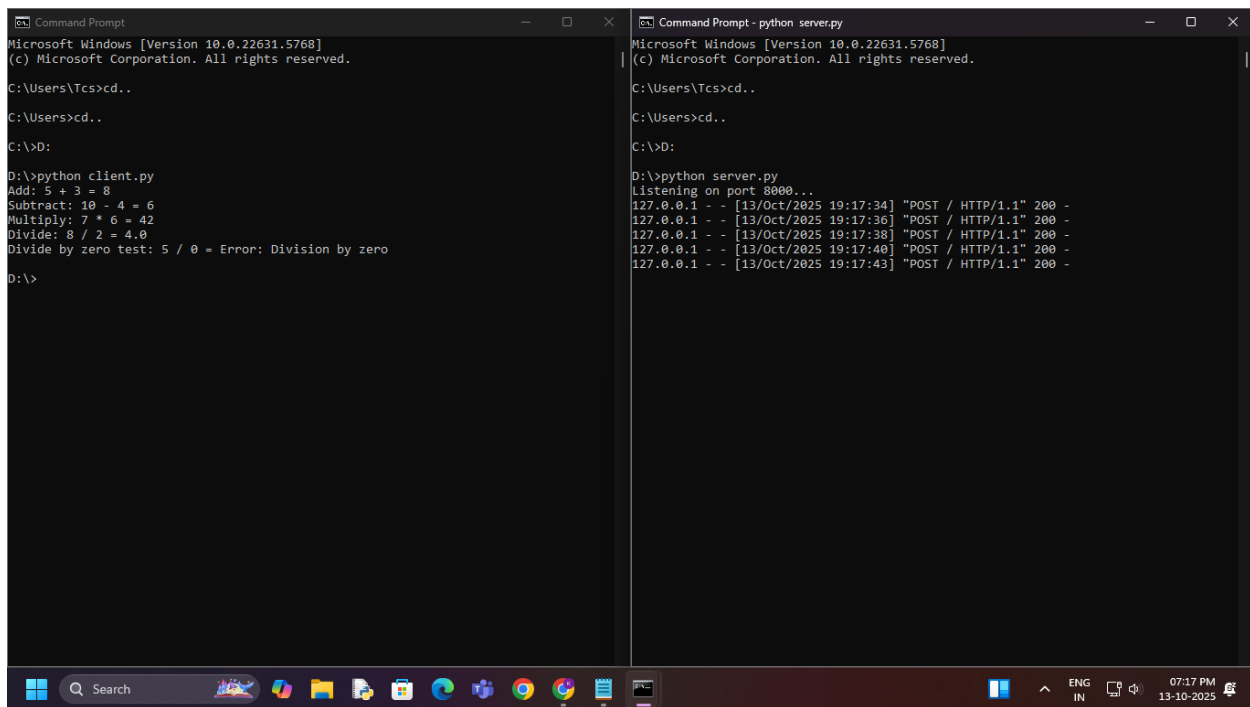
```
print("Subtract: 10 - 4 =", proxy.subtract(10, 4))
```

```
print("Multiply: 7 * 6 =", proxy.multiply(7, 6))
```

```
print("Divide: 8 / 2 =", proxy.divide(8, 2))
```

```
print("Divide by zero test: 5 / 0 =", proxy.divide(5, 0))
```

OUTPUT:



```
Command Prompt
Microsoft Windows [Version 10.0.22631.5768]
(c) Microsoft Corporation. All rights reserved.

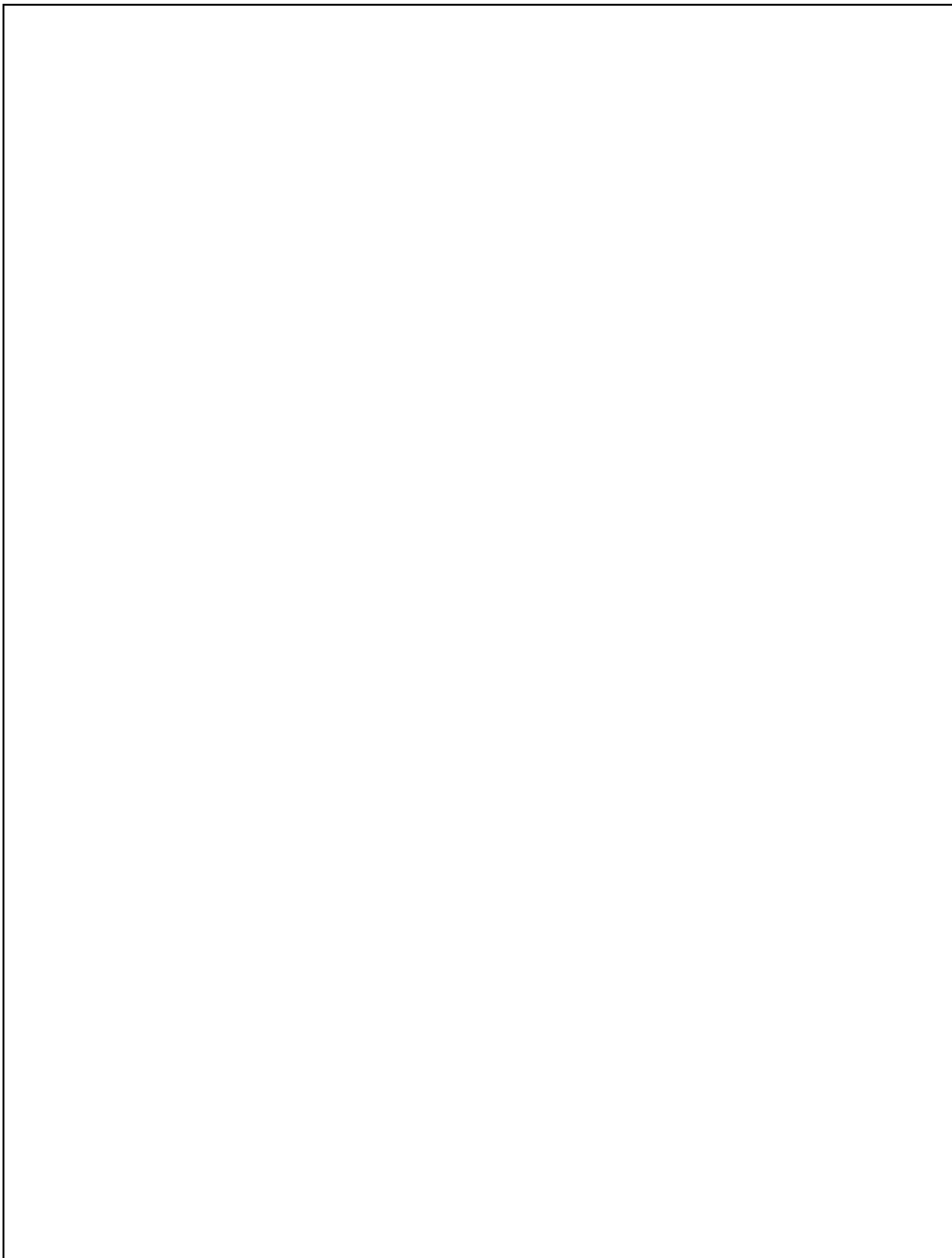
C:\Users\Tcs>cd..
C:\Users>cd..
C:\>D:
D:\>python client.py
Add: 5 + 3 = 8
Subtract: 10 - 4 = 6
Multiply: 7 * 6 = 42
Divide: 8 / 2 = 4.0
Divide by zero test: 5 / 0 = Error: Division by zero
D:\>
```

```
Command Prompt - python server.py
Microsoft Windows [Version 10.0.22631.5768]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Tcs>cd..
C:\Users>cd..
C:\>D:
D:\>python server.py
Listening on port 8000...
127.0.0.1 - - [13/Oct/2025 19:17:34] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [13/Oct/2025 19:17:36] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [13/Oct/2025 19:17:38] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [13/Oct/2025 19:17:40] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [13/Oct/2025 19:17:43] "POST / HTTP/1.1" 200 -
```

RESULT:

This simple XML-RPC calculator demonstrates how a client can remotely invoke basic arithmetic operations on a server, showcasing easy inter-process communication using Python's built-in XML-RPC modules.



EXERCISE-6

IMPLEMENT PACKET SNIFFING USING RAW SOCKETS IN PYTHON

AIM:

Build a simple Python raw-socket packet sniffer that captures Ethernet/IP packets, parses IPv4/TCP/UDP headers, and prints timestamped, human-readable summaries with short hexdumps

ALGORITHM:

- Create a raw socket to capture all packets on the network interface (requires elevated privileges).
- Continuously receive packets from the socket in a loop.
- Parse the packet headers: Ethernet (Linux), IPv4, then TCP or UDP if applicable.
- Extract and format key information such as source/destination IPs, ports, protocol, and packet length.
- Display a timestamped summary and a short hex dump of the packet payload; repeat until stopped.

CODE:

```
import socket
import struct
import binascii
import textwrap

def main():
    # Get host
    host = socket.gethostbyname(socket.gethostname())
    print('IP: {}'.format(host))
```

```

    conn = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_IP)
    conn.bind((host, 0))

    conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
    # Enable promiscuous mode
    conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    while True:
        # Recive data
        raw_data, addr = conn.recvfrom(65536)

        # Unpack data
        dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)

        print('\nEthernet Frame:')
        print("Destination MAC: {}".format(dest_mac))
        print("Source MAC: {}".format(src_mac))
        print("Protocol: {}".format(eth_proto))

# Unpack ethernet frame
def ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6s2s', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac),
get_protocol(proto), data[14:]

# Return formatted MAC address AA:BB:CC:DD:EE:FF
def get_mac_addr(bytes_addr):
    bytes_str = map('{:02x}'.format, bytes_addr)
    mac_address = ':'.join(bytes_str).upper()
    return mac_address

# Return formatted protocol ABCD
def get_protocol(bytes_proto):
    bytes_str = map('{:02x}'.format, bytes_proto)
    protocol = ''.join(bytes_str).upper()
    return protocol
main()

```

OUTPUT:

```
Administrator Command Prompt
Microsoft Windows [Version 10.0.26200.6725]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd..
C:\Windows>cd..
C:\>D:
D:\>python sniffing.py
IP: 172.16.36.65

Ethernet Frame:
Destination MAC: 45:00:00:39:02:3D
Source MAC: 00:00:00:11:00:00
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:00:39:02:3E
Source MAC: 00:00:00:11:00:00
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:01:3E:74:88
Source MAC: 40:00:40:11:24:C4
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:04:FE:8A:89
Source MAC: 40:00:40:11:25:53
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:04:FE:8A:EB
Source MAC: 00:00:00:11:00:00
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:04:FE:8A:EC
Source MAC: 40:00:00:11:00:00
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:00:44:00:00
Source MAC: 40:00:3C:11:40:8C
Protocol: 172C

Ethernet Frame:
Destination MAC: 45:00:04:FE:00:00
Source MAC: 40:00:3C:11:40:52
Protocol: 172C

Ethernet Frame:
Destination MAC: 45:00:04:FE:8A:ED
Source MAC: 00:00:00:11:00:00
Protocol: AC10

Ethernet Frame:
Destination MAC: 45:00:04:FE:00:00
Source MAC: 40:00:3C:11:40:52
Protocol: 172C

Ethernet Frame:
```

RESULT:

The implemented Python sniffer successfully captures live network packets, parses relevant header information, and prints readable summaries with payload hex dumps in real time.

NAME:KEERTHISRI.D

DATE:15.9.25

ROLL NO:241901047

EXERCISE-7

N-MAP TO DISCOVER LIVE HOSTS USING ARP REQUEST SCAN,ICMP SCAN AND TCP/UDP PING SCAN

INTRODUCTION:

In this module ,we focus on discovering live host on a network using Nmap.Finding which system are online is crucial before Scanning ports or Services to save time and avoid unecesary network traffic.

TASK 2:

SUBNETWORKS:

Room completed (100%)

Answer the questions below

Send a packet with the following:

Send Packet

From: computer1

To: computer1

Packet Type: arp_request

Data: computer6

Send Packet

- From computer1
- To computer1 (to indicate it is broadcast)
- Packet Type: "ARP Request"
- Data: computer6 (because we are asking for computer6 MAC address using ARP Request)

How many devices can see the ARP Request?

4

✓ Correct Answer

Hint

Did computer6 receive the ARP Request? (Y/N)

N

✓ Correct Answer

Send a packet with the following:

Send Packet

From: computer4

To: computer4

Packet Type: arp_request

Data: computer6

Send Packet

- From computer4
- To computer4 (to indicate it is broadcast)
- Packet Type: "ARP Request"
- Data: computer6 (because we are asking for computer6 MAC address using ARP Request)

How many devices can see the ARP Request?

4

✓ Correct Answer

Hint

Did computer6 reply to the ARP Request? (Y/N)

Y

✓ Correct Answer

Task 3 Enumerating Targets

Task 4 Discovering Live Hosts

TASK 3:

ENUMERATING TARGETS:

tryhackme.com/room/nmap01

Room completed (100%)

Task 2 Subnetworks

Task 3 Enumerating Targets

We mentioned the different techniques we can use for scanning in Task 1. Before we explain each in detail and put it into use against a live target, we need to specify the targets we want to scan. Generally speaking, you can provide a list, a range, or a subnet. Examples of target specification are:

- list: `MACHINE_IP scanner.nmap.org example.com` will scan 3 IP addresses.
- range: `10.11.12.15-20` will scan 6 IP addresses: `10.11.12.15`, `10.11.12.16`, ..., and `10.11.12.20`.
- subnet: `MACHINE_IP/30` will scan 4 IP addresses.

You can also provide a file as input for your list of targets, `nmap -iL list_of_hosts.txt`.

If you want to check the list of hosts that Nmap will scan, you can use `nmap -sL TARGETS`. This option will give you a detailed list of the hosts that Nmap will scan without scanning them; however, Nmap will attempt a reverse-DNS resolution on all the targets to obtain their names. Names might reveal various information to the pentester. (If you don't want Nmap to the DNS server, you can add `-n`.)

Launch the AttackBox using the Start AttackBox button, open the terminal when the AttackBox is ready, and use Nmap to answer the following.

Answer the questions below

What is the first IP address Nmap would scan if you provided `10.10.12.13/29` as your target?

✓ Correct Answer 🔍 Hint

How many IP addresses will Nmap scan if you provide the following range `10.10.0-255.101-125`?

✓ Correct Answer 🔍 Hint

Task 4 Discovering Live Hosts

Task 5 Nmap Host Discovery Using ARP

Light rain in the afternoon

ENG IN 11:01 15-10-2023

TASK 4:

DISCOVERING LIVE HOSTS:

tryhackme.com/room/nmap01

Room completed (100%)

Answer the questions below

Send a packet with the following:

- From computer1
- To computer3
- Packet Type: "Ping Request"

What is the type of packet that computer1 sent before the ping?

ARP Request ✓ Correct Answer

What is the type of packet that computer1 received before being able to send the ping?

ARP Response ✓ Correct Answer

How many computers responded to the ping request?

1 ✓ Correct Answer

Send a packet with the following:

- From computer2
- To computer5
- Packet Type: "Ping Request"

What is the name of the first device that responded to the first ARP Request?

router ✓ Correct Answer

What is the name of the first device that responded to the second ARP Request?

computer5 ✓ Correct Answer

Send another Ping Request. Did it require new ARP Requests? (Y/N)

tryhackme.com/room/nmap01

tryhackme.com/room/nmap01

Room completed (100%)

- To computer5
- Packet Type: "Ping Request"

What is the name of the first device that responded to the first ARP Request?

router ✓ Correct Answer

What is the name of the first device that responded to the second ARP Request?

computer5 ✓ Correct Answer

Send another Ping Request. Did it require new ARP Requests? (Y/N)

N ✓ Correct Answer

Task 5 ✓ Nmap Host Discovery Using ARP

Task 6 ✓ Nmap Host Discovery Using ICMP

Task 7 ✓ Nmap Host Discovery Using TCP and UDP

Task 8 ✓ Using Reverse-DNS Lookup

Task 9 ✓ Summary

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

Submit now

tryhackme.com/room/nmap01

TASK 5:

NMAP HOST DISCOVERING USING ARP:

Room completed (100%)

MAC Address	Type	Protocol	Request	Response
02:ba:eb:d6:18:2b	Broadcast	ARP	Who has 10.10.210.12? Tell 10.10.210.6	
02:ba:eb:d6:18:2b	Broadcast	ARP	Who has 10.10.210.13? Tell 10.10.210.6	
02:ba:eb:d6:18:2b	Broadcast	ARP	Who has 10.10.210.14? Tell 10.10.210.6	
02:ba:eb:d6:18:2b	Broadcast	ARP	Who has 10.10.210.15? Tell 10.10.210.6	
02:ba:eb:d6:18:2b	Broadcast	ARP	Who has 10.10.210.16? Tell 10.10.210.6	

Address Resolution Protocol: Protocol Packets: 1207 · Displayed: 512 (42.4%) Profile: Default

If you have closed the network simulator, click on the "Visit Site" button in Task 2 to display it again.

Answer the questions below

We will be sending broadcast ARP Requests packets with the following options:

- From computer1
- To computer1 (to indicate it is broadcast)
- Packet Type: "ARP Request"
- Data: try all the possible eight devices (other than computer1) in the network: computer2, computer3, computer4, computer5, computer6, switch1, switch2, and router.

How many devices are you able to discover using ARP requests?

3

✓ Correct Answer

Task 6 ✓ Nmap Host Discovery Using ICMP

Task 7 ✓ Nmap Host Discovery Using TCP and UDP

Task 8 ✓ Using Reverse-DNS Lookup

Task 9 ✓ Summary

How likely are you to recommend this room to others?

1 cm of rain
Friday

Search

ENG IN 11:04 15-10-2025

TASK 6:

NAMP HOST DISCOVERING USING ICMP:

tryhackme.com/room/nmap01

Room completed (100%)

10.11.35.214	10.10.68.2	ICMP	Address mask request id=0x683d, seq=0/0, ttl=
10.11.35.214	10.10.68.3	ICMP	Address mask request id=0x2711, seq=0/0, ttl=
10.11.35.214	10.10.68.4	ICMP	Address mask request id=0xfde3, seq=0/0, ttl=
10.11.35.214	10.10.68.5	ICMP	Address mask request id=0x2eb1, seq=0/0, ttl=
10.11.35.214	10.10.68.6	ICMP	Address mask request id=0x8380, seq=0/0, ttl=

nmap-PM-sn-openvpn.pcapng Packets: 1178 - Displayed: 512 (43.5%) Profile: Default

Answer the questions below

What is the option required to tell Nmap to use ICMP Timestamp to discover live hosts?

-PP ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Address Mask to discover live hosts?

-PM ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Echo to discover live hosts?

-PE ✓ Correct Answer

Task 7 Nmap Host Discovery Using TCP and UDP

Task 8 Using Reverse-DNS Lookup

Task 9 Summary

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

1 cm of rain Friday

TASK 7:

NAMP HOST DISCOVERING USING TCP/UDP :

tryhackme.com/room/nmap01

Room completed (100%)

generates. The syntax is quite similar: `-p` can be followed by a port number, list, or range. Consider the following examples:

- `masscan MACHINE_IP/24 -p443`
- `masscan MACHINE_IP/24 -p80,443`
- `masscan MACHINE_IP/24 -p22-25`
- `masscan MACHINE_IP/24 --top-ports 100`

Masscan is not installed on the AttackBox; however, it can be installed using `apt install masscan`.

Answer the questions below

Which TCP ping scan does not require a privileged account?

TCP SYN Ping ✓ Correct Answer

Which TCP ping scan requires a privileged account?

TCP ACK Ping ✓ Correct Answer

What option do you need to add to Nmap to run a TCP SYN ping scan on the telnet port?

-PS23 ✓ Correct Answer Hint

Task 8 Using Reverse-DNS Lookup

Task 9 Summary

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

Submit now

Very humid Now

TASK 8:

USING REVERSE-DNS LOOKUP:

Room completed (100%)

Task 5 Nmap Host Discovery Using ARP

Task 6 Nmap Host Discovery Using ICMP

Task 7 Nmap Host Discovery Using TCP and UDP

Task 8 Using Reverse-DNS Lookup

Nmap's default behaviour is to use reverse-DNS on live hosts. Because the hostnames can reveal a lot, this can be a helpful step. However, if you don't want to send such DNS queries, you use `-n` to skip this step.

By default, Nmap will look up online hosts; however, you can use the option `-R` to query the DNS server even for offline hosts. If you want to use a specific DNS server, you can add the `--dns-servers DNS_SERVER` option.

Answer the questions below

We want Nmap to issue a reverse DNS lookup for all the possible hosts on a subnet, hoping to get some insights from the names. What option should we add?

Correct Answer

Task 9 Summary

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

RESULT:

N-map detects live host using ARP ,ICMP,TCP,SYN/ACK and UDP pings DNS lookups run by default ,nut n,-R and –dns services control resolution.

NAME:KEERTHISRI.D

DATE:23.9.25

ROLL NO:241901047

EXERCISE-8

BUILDING ANONYMOUS FTP SCANNER USING FTPLIP MODULE

AIM:

Check if an FTP server allows anonymous login username “anonymous”.If successful display,the server’s welcome message and directory contents.

ALGORITHM:

1. Target Preparation & Safety Checks
2. Connection & Banner Fingerprinting
3. Anonymous Login Attempt
4. Non-Destructive Post-Login Probes
5. Record, Aggregate & Report Results

CODE:

```
import ftplib
def anonymousLogin(hostname):
    try:
        ftp = ftplib.FTP(hostname)
        response = ftp.login('anonymous', 'anonymous')
        print(response)
        if "230 Anonymous access granted" in response:
            print('\n[*] ' + str(hostname) + ' FTP Anonymous Login Succeeded.')
            print(ftp.getwelcome())
            ftp.dir()
    except Exception as e:
        print(str(e))
        print('\n[-] ' + str(hostname) + ' FTP Anonymous Login Failed.')
hostname = 'ftp.be.debian.org'
anonymousLogin(hostname)
```

OUTPUT:

```
Command Prompt
x + v

The server maintains software archive accessible via ftp, http, https and rsync.

ftp.be.debian.org is an alias for this host, but https will not work with that
alias. If you want to use https use mirror.as35701.net.

Contact: mirror-admin at as35701.net

230 Anonymous access granted, restrictions apply

[*] ftp.be.debian.org FTP Anonymous Login Succeeded.
220 ProFTPD Server (mirror.as35701.net) [::ffff:195.234.45.114]
drwxr-xr-x 9 ftp ftp 4096 Oct 15 02:43 debian
drwxr-xr-x 5 ftp ftp 105 Sep 7 06:58 debian-cd
drwxr-xr-x 7 ftp ftp 4096 Oct 14 17:27 debian-security
drwxr-xr-x 5 ftp ftp 4096 Oct 13 2006 ftp.irc.org
-rw-r--r-- 1 ftp ftp 717 Apr 15 2025 HEADER.html
drwxr-xr-x 5 ftp ftp 4096 Oct 15 03:52 mint
drwxr-xr-x 5 ftp ftp 40 May 16 11:09 mint-iso
lrwxrwxrwx 1 ftp ftp 33 Apr 29 2021 pub -> /var/www/html/www.kernel.org/pub/
drwxr-xr-x 7 ftp ftp 4096 Oct 15 04:47 ubuntu
drwxr-xr-x 39 ftp ftp 4096 Oct 15 05:26 ubuntu-cdimage
drwxr-xr-x 32 ftp ftp 4096 Oct 14 19:28 ubuntu-cloudimages
drwxr-xr-x 7 ftp ftp 4096 Oct 15 05:45 ubuntu-ports
drwxr-xr-x 16 ftp ftp 4096 Oct 15 01:19 ubuntu-releases
drwxr-xr-x 25 ftp ftp 383 Oct 14 23:09 video.fosdem.org
-rw-r--r-- 1 ftp ftp 390 Jul 9 2021 welcome.msg
drwxr-xr-x 4 ftp ftp 4096 Jun 14 2023 www.kernel.org

D:\>
```

RESULT:

SUCCESS: Host accepted anonymous login — an FTP session was established (may be read-only, limited, or a honeypot; login ≠ guaranteed file access).

FAILED: Connection/login failed or unreachable — covers rejected auth, no FTP service, TLS/SFTP-only, firewalls, timeouts, or transient network errors.

Roll No: 241901047

Name: Keerthisri D

Experiment: 9

DESIGN A SIMPLE TOPOLOGY AND CONFIGURE WITH ONE ROUTER, 2 SWITCHES AND PCS USING CISCO PACKET TRACER

AIM:

To design and configure a simple network topology with **one router, two switches, and multiple PCs** using **Cisco Packet Tracer** for basic communication.

ALGORITHM:

1. Start **Cisco Packet Tracer**.
2. Drag and place **1 Router, 2 Switches**, and **4 PCs**.
3. Connect the devices using **straight-through cables**.
4. Assign **IP addresses** to all PCs and router interfaces.
5. Configure the router interfaces with the given IP addresses and enable them.
6. Save the configuration.
7. **Ping** between PCs to check network connectivity.

CODE:

```
Router>enable
```

```
Router#configure terminal
```

Enter configuration commands, one per line. End with CNTL/Z.

```
Router(config)#interface gigabitEthernet0/0
```

```
Router(config-if)#ip address 192.168.1.1 255.255.255.0
```

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0,
changed state to up

exit

Router(config)#interface gigabitEthernet0/1

Router(config-if)#ip address 192.168.2.1 255.255.255.0

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,
changed state to up

exit

Router(config)#exit

Router#

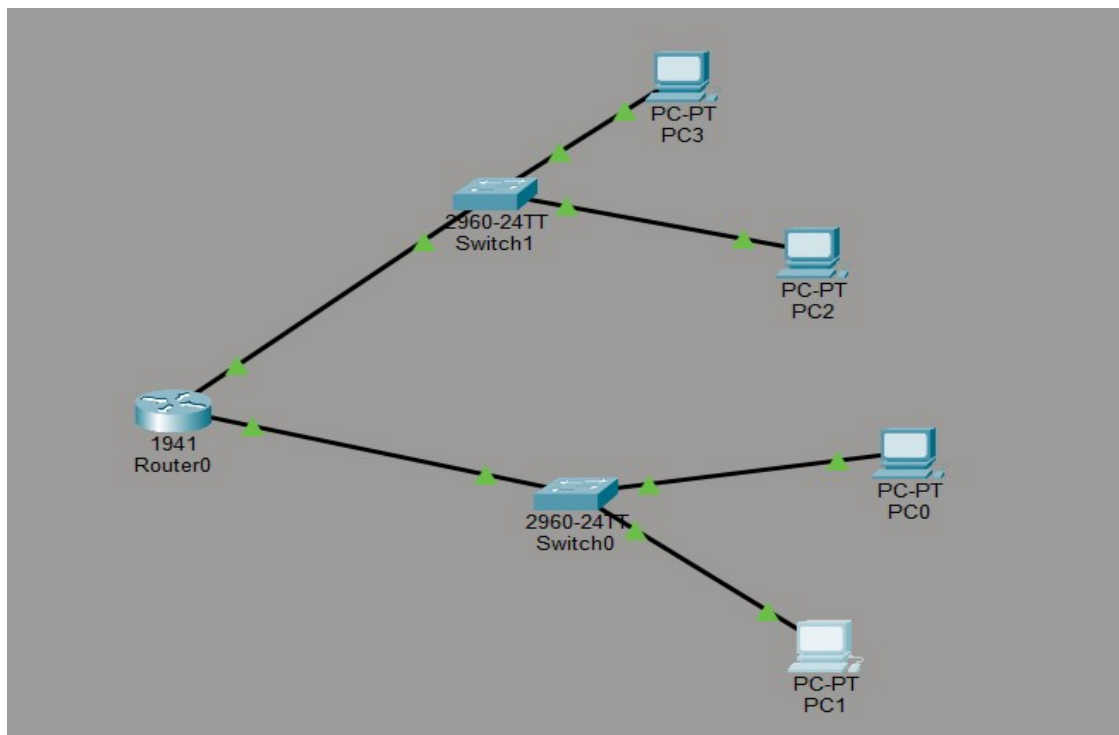
%SYS-5-CONFIG_I: Configured from console by console

write memory

Building configuration...

[OK]

Router#



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.10.12

Pinging 192.168.10.12 with 32 bytes of data:

Reply from 192.168.10.12: bytes=32 time=8ms TTL=128
Reply from 192.168.10.12: bytes=32 time=11ms TTL=128
Reply from 192.168.10.12: bytes=32 time=8ms TTL=128
Reply from 192.168.10.12: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 11ms, Average = 6ms

C:\>ping 192.168.20.21
```

RESULT:

Thus, the simple topology using router, switches and PCs are made using Cisco Packet Tracer.

EXERCISE 10

DESIGN A SIMPLE TOPOLOGY USING CISCO PACKET TRACER

AIM:

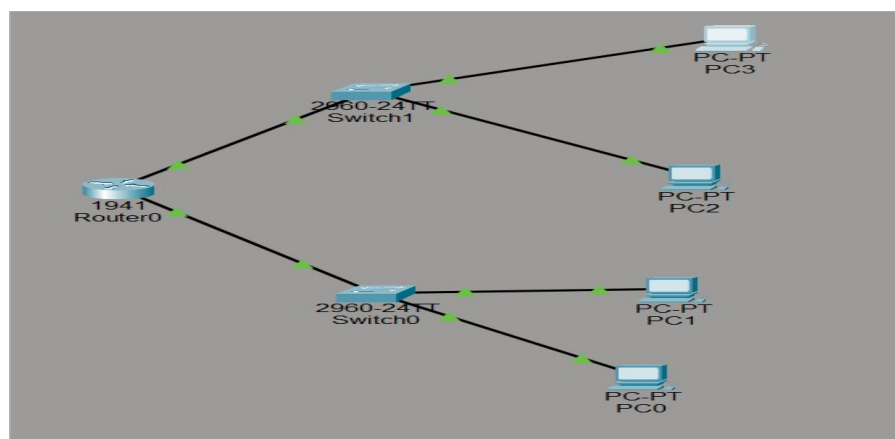
To design a simple network topology using Cisco Packet Tracer that includes one router, two switches, and multiple PCs, and to configure them for successful communication.

INTRODUCTION:

Cisco Packet Tracer is a powerful network simulation tool that allows users to design, configure, and test computer network topologies virtually. By using routers, switches, and PCs, students and professionals can build and practice real-world networking scenarios without physical hardware. In this experiment, a simple topology with one router, two switches, and multiple PCs is designed to demonstrate basic device setup and connectivity.

ALGORITHM:

1. Open Cisco Packet Tracer and create a new project workspace.
2. Add Devices:
 - Drag one router, two switches, and multiple PCs onto the workspace.



3. Connect Devices:
 - Use straight-through cables to connect each PC to a switch.
 - Connect each switch to a separate interface on the router.
4. Assign IP Addresses:

- Set appropriate IP addresses for each PC in their respective subnets.

The image displays three screenshots of PC configuration windows, labeled PC0, PC1, and PC2. Each window shows the 'Config' tab with 'FastEthernet0' selected as the interface. The configuration is divided into three main sections: IP Configuration, IPv6 Configuration, and 802.1X.

PC0 Configuration:

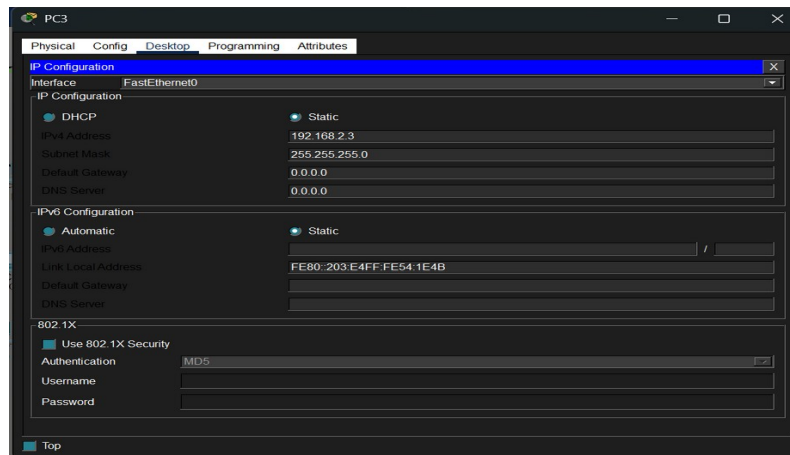
- IP Configuration:** DHCP is disabled, Static is selected. IPv4 Address: 192.168.1.2, Subnet Mask: 255.255.255.0, Default Gateway: 0.0.0.0, DNS Server: 0.0.0.0.
- IPv6 Configuration:** Automatic is disabled, Static is selected. IPv6 Address: (empty), Link Local Address: FE80::200:CFF:FEA4:C962, Default Gateway: (empty), DNS Server: (empty).
- 802.1X:** Use 802.1X Security is disabled. Authentication: MD5, Username: (empty), Password: (empty).

PC1 Configuration:

- IP Configuration:** DHCP is disabled, Static is selected. IPv4 Address: 192.168.1.3, Subnet Mask: 255.255.255.0, Default Gateway: 0.0.0.0, DNS Server: 0.0.0.0.
- IPv6 Configuration:** Automatic is disabled, Static is selected. IPv6 Address: (empty), Link Local Address: FE80::290:21FF:FE1B:C47, Default Gateway: (empty), DNS Server: (empty).
- 802.1X:** Use 802.1X Security is disabled. Authentication: MD5, Username: (empty), Password: (empty).

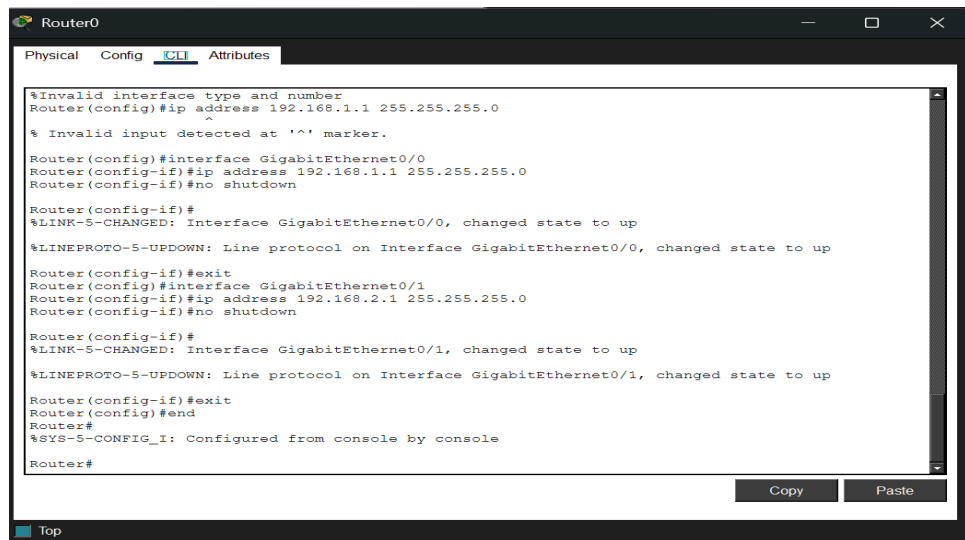
PC2 Configuration:

- IP Configuration:** DHCP is disabled, Static is selected. IPv4 Address: 192.168.2.2, Subnet Mask: 255.255.255.0, Default Gateway: 0.0.0.0, DNS Server: 0.0.0.0.
- IPv6 Configuration:** Automatic is disabled, Static is selected. IPv6 Address: (empty), Link Local Address: FE80::290:2BFF:FE98:54B, Default Gateway: (empty), DNS Server: (empty).
- 802.1X:** Use 802.1X Security is disabled. Authentication: MD5, Username: (empty), Password: (empty).



5. Configure Router Interfaces:

- Access the router CLI and assign IP addresses to each connected interface.

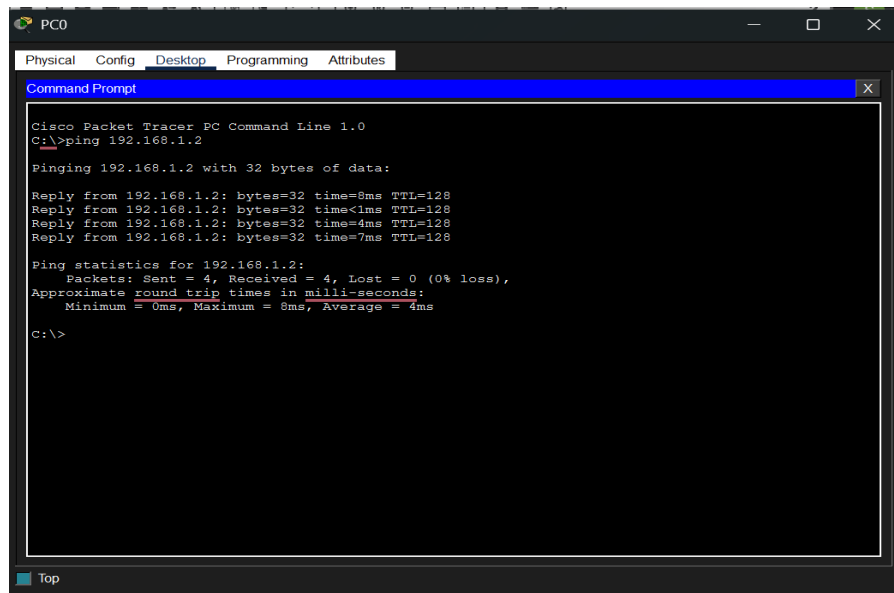


6. Set Default Gateway:

- Configure each PC's default gateway to match the router interface IP on its subnet.

7. Verify Connectivity:

- Use the 'ping' command on PCs to check network communication across the topology.



```
PC0
Physical Config Desktop Programming Attributes
Command Prompt
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time=8ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time=4ms TTL=128
Reply from 192.168.1.2: bytes=32 time=7ms TTL=128

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 8ms, Average = 4ms

C:\>
```

RESULT:

The designed topology, consisting of one router, two switches, and multiple PCs, was successfully configured. All devices were able to communicate with each other across the network, demonstrating correct network setup and connectivity verification using Cisco Packet Tracer.

EXERCISE 11

CUSTOMIZE SWITCH WITH NETWORK MODULES USING CISCO PACKET TRACER

AIM:

To customise Switch with Network Modules using Cisco Packet Tracer.

ALGORITHM:

1. Open Cisco Packet Tracer and place a switch in the workspace.



2. Click the switch to open its device window and go to the Physical tab.
3. Power off the switch by clicking the red power button.
4. Select a desired network module from the module list.
5. Drag and insert the module into an available slot on the switch chassis.
6. Power on the switch by clicking the power button again.
7. Verify the new module and interfaces are recognised in the Config or CLI tab.

```
Switch>enable
Switch#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)#hostname Switch1
Switch1(config)#banner motd $ Authorized Access Only $
Switch1(config)#line console 0
Switch1(config-line)#password cisco
Switch1(config-line)#login
Switch1(config-line)#exit
Switch1(config)#enable secret class
Switch1(config)#end
Switch1#
%SYS-5-CONFIG_I: Configured from console by console
write memory
Building configuration...
[OK]
Switch1#
```

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.10.2

Pinging 192.168.10.2 with 32 bytes of data:

Reply from 192.168.10.2: bytes=32 time<1ms TTL=128
Reply from 192.168.10.2: bytes=32 time=8ms TTL=128
Reply from 192.168.10.2: bytes=32 time=8ms TTL=128
Reply from 192.168.10.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 8ms, Average = 4ms
```

RESULT:

The switch was successfully customised with additional network modules, expanding its interfaces and connectivity options in Cisco Packet Tracer. The switch powered on with new modules recognised and ready for use, enabling more flexible network design and testing.

EXERCISE 12

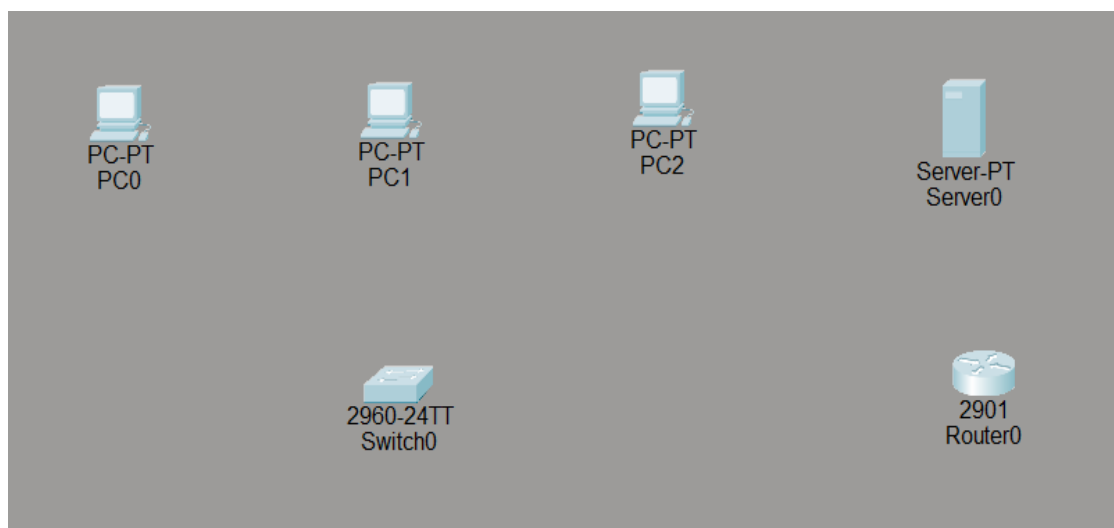
EXAMINE NETWORK ADDRESS TRANSLATION(NAT) USING CISCO PACKET TRACER

AIM:

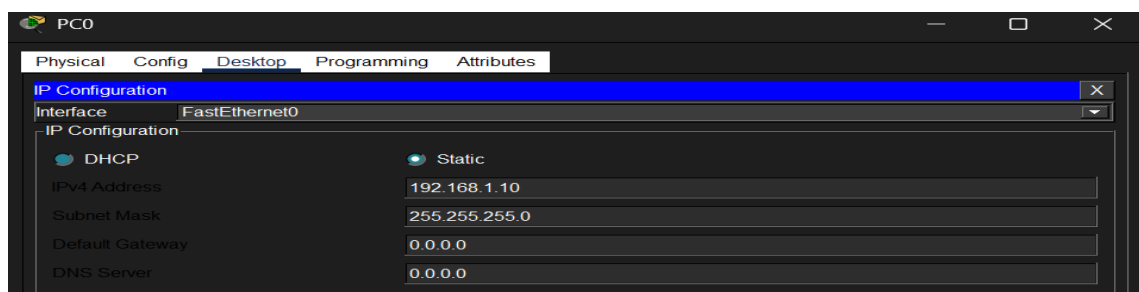
To demonstrate how NAT translates private IP addresses to public IP addresses, enabling private network devices to communicate with external networks. It also includes configuring and verifying different NAT types (Static, Dynamic, and PAT) in Cisco Packet Tracer.

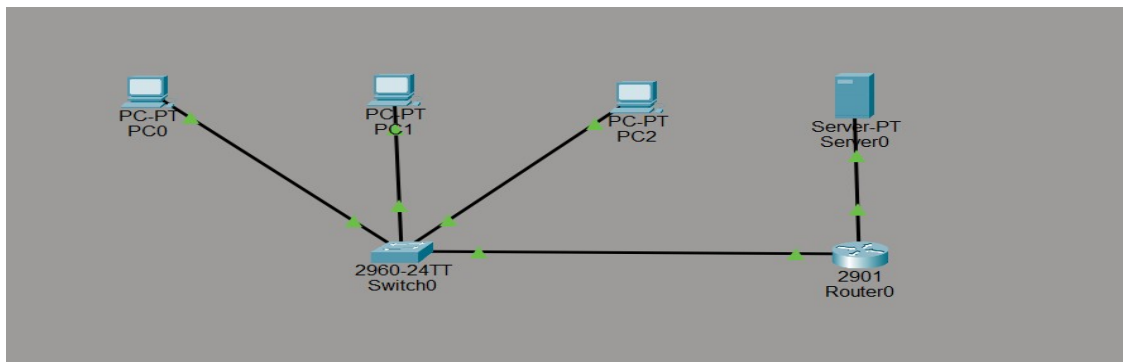
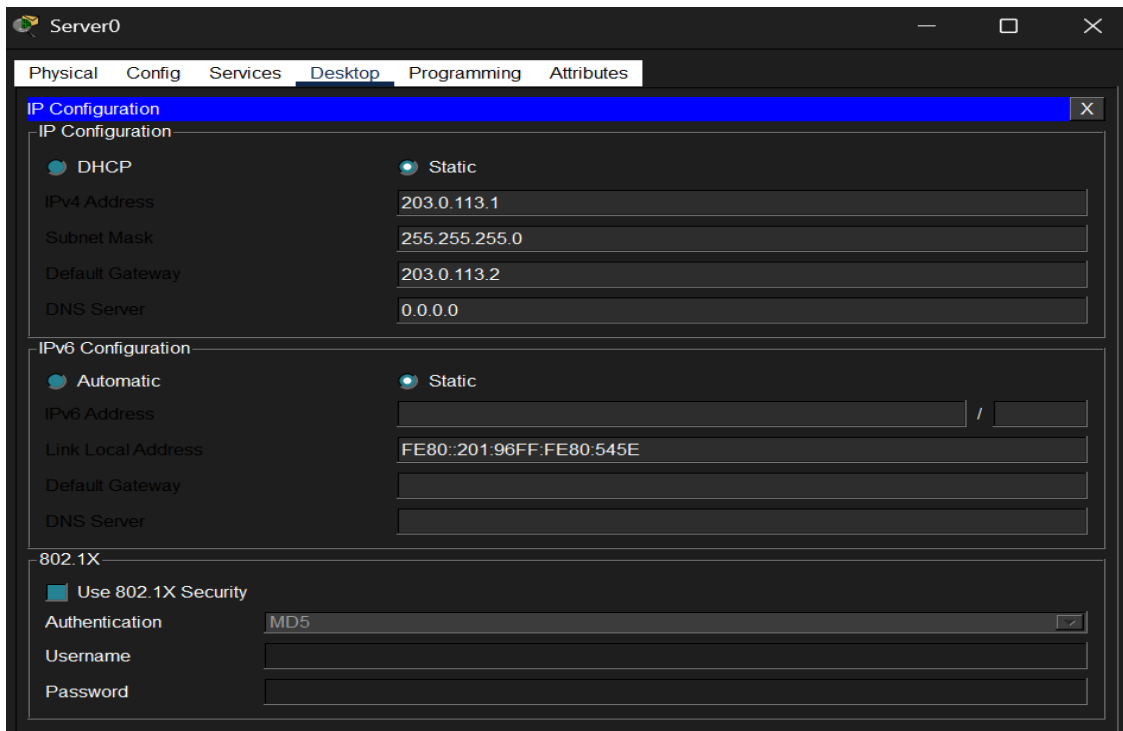
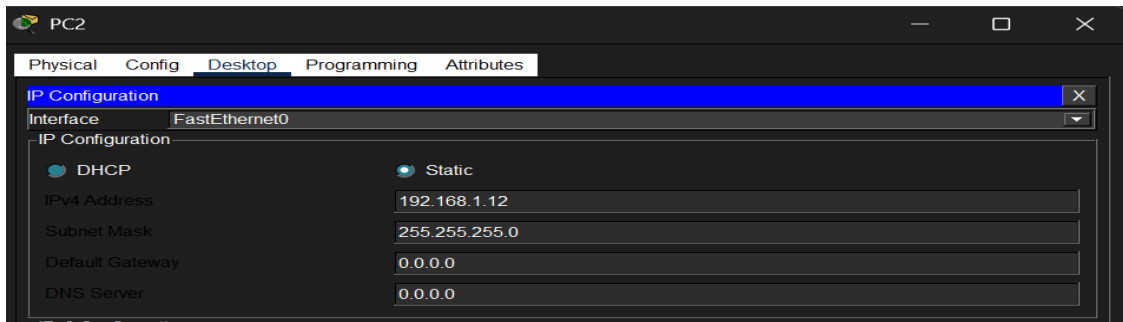
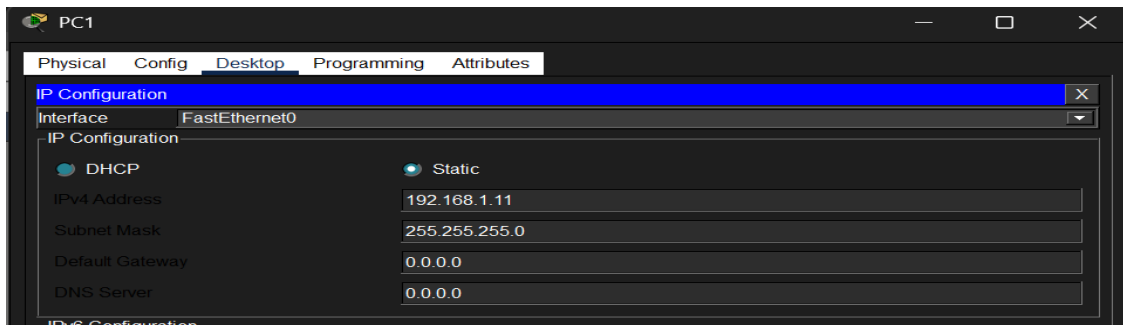
ALGORITHM:

1. Build a simple topology with a router connected to a private internal network and an external network.



2. Assign IP addresses to all devices and router interfaces.





3. Configure router interfaces as NAT inside (connected to private network) and NAT outside (connected to external network).
4. Define NAT translation rules:
 - Static NAT: Map a specific internal IP to a specific external IP.
 - Dynamic NAT: Use a pool of public IP addresses for translation.
 - PAT (Port Address Translation): Use a single public IP with different port numbers.
5. Create access control lists (ACLs) to match private IP addresses allowed to be translated.
6. Enable the NAT on the router with the appropriate commands.
7. Test connectivity by pinging from inside devices to external devices or servers.
8. Verify NAT translation tables using commands like show ip nat translations.

```

Router(config-if)#interface gigabitethernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
% 192.168.1.0 overlaps with GigabitEthernet0/1
Router(config-if)#no shutdown
% 192.168.1.0 overlaps with GigabitEthernet0/1
GigabitEthernet0/0: incorrect IP address assignment
Router(config-if)#exit
Router(config)#interface gigabitethernet0/0
Router(config-if)#
Router(config-if)#
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                     ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                     ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                     ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit192.168.1.0.0.0.0.255
                                     ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0 0.0.0.255
Router(config)#ip nat inside source list 1 interface
% Incomplete command.
Router(config)#ip nat inside source list 1 interface gigabitethernet0/1 overload
Router(config)#show ip nat translations
                                     ^
% Invalid input detected at '^' marker.

Router(config)#end
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#show ip nat translations
Router#

```

```

Router#show ip interface brief
Interface          IP-Address      OK? Method Status      Protocol
GigabitEthernet0/0 192.168.1.1     YES manual administratively down down
GigabitEthernet0/1 192.168.1.1     YES manual up      up
Vlan1              unassigned      YES unset  administratively down down
Router#

Router#
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#interface gigabitethernet0/1
Router(config-if)#ip address 203.0.113.2 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface gigabitethernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0, changed state to up
|

```

RESULT:

NAT translates private IPs to public IPs for internet access. Routers keep translation tables. NAT enables private devices to communicate externally, verified through Packet Tracer simulations and commands.

Roll No: 241901047

Name: Keerthisri D

Experiment: 13

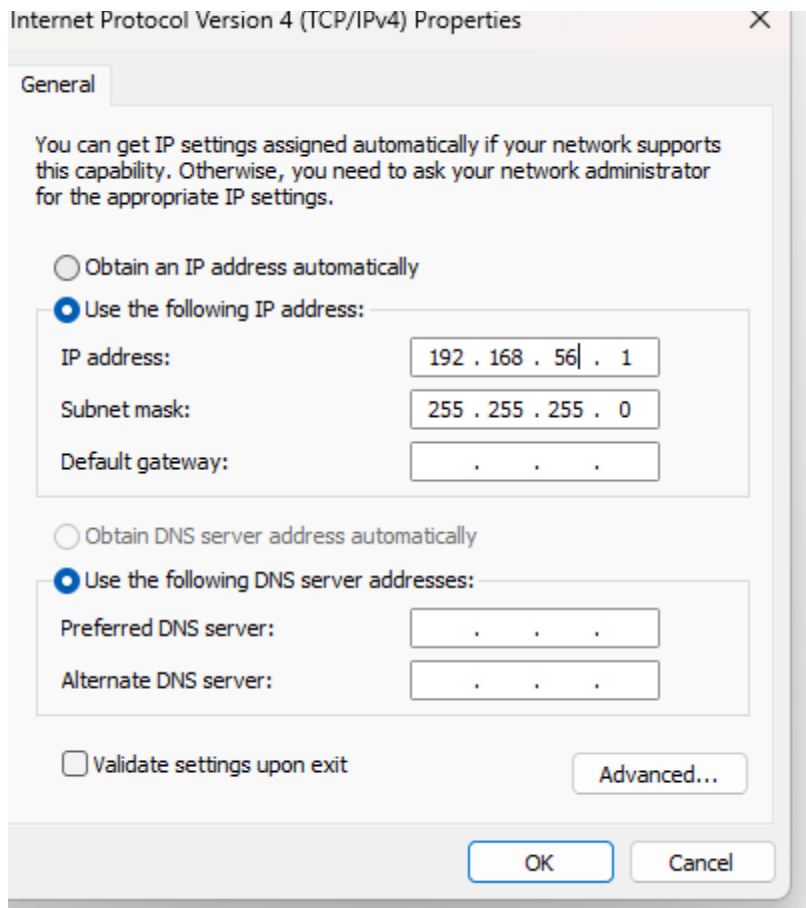
LEARNING AND ASSIGNMENT OF IP ADDRESS MANUALLY TO COMPUTERS

AIM:

To learn how to **assign IP addresses manually** to computers in a network using **Cisco Packet Tracer**.

ALGORITHM:

1. Open **Cisco Packet Tracer**.
2. **Create a simple network** by connecting two or more PCs with a switch.
3. **Select a PC**, click on it, and open the **Desktop tab**.
4. Click on **IP Configuration**.
5. **Manually enter**:
 - o IP Address (e.g., 192.168.1.2)
 - o Subnet Mask (e.g., 255.255.255.0)
 - o Default Gateway (if required, e.g., 192.168.1.1)
6. **Repeat** the process for all other PCs (assign different IPs in the same network).
7. **Test connectivity** using the **ping command** from one PC to another.
8. **Observe the results** and save the configuration.



OUTPUT:

BEFORE CHANGING

```
Microsoft Windows [Version 10.0.26100.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HDC0422193>cd..

C:\Users>cd..

C:\>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::b390:ace5:58a6:85f%13
    IPv4 Address. . . . . : 172.16.53.179
    Subnet Mask . . . . . : 255.255.254.0
    Default Gateway . . . . . : 172.16.52.1
```

AFTER CHANGING

```
C:\>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::b390:ace5:58a6:85f%13
    IPv4 Address. . . . . : 172.16.53.179
    Subnet Mask . . . . . : 255.255.254.0
    Default Gateway . . . . . : 172.16.52.1

Ethernet adapter Ethernet 2:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::2739:d7ac:d7d9:4c2f%5
    IPv4 Address. . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :
```

RESULT:

Thus, the system was configured and assigned an IP address manually to the system.