CS-325 Computer Networks Lab

Assignment - 1

Submitted by -
Gyanendra Shukla
CSE 1
191112040

Study of different types of Network cables

There are 3 types of network cables primarily:

1. **Co-axial network cables**
   These cables are highly resistant to obstruction and work for long lengths of wire. However, they are complex to install.

   They are further divided into:

- Single core: made of single copper conductor
- Multi core: made of multiple strands

The structure of co-axial cables consists of the following (in order from center):

- Conductor: carries the EM signals
- Insulation: protects from interference and noise
- Braiding: protects from interference and noise
- Sheath: protects from physical damage

2. **Fiber optic cables**
   These are super fast cables, capable of carrying signals at speeds of over 100GBPS over almost 40km distance.

These are divided into:

- Single mode: carries only single beam of light. Higher bandwidth, but supports higher distance
- Multi mode: carries multiple beams of light. Higher speed, but supports shorter distance

The structure of fiber optic cable consists of the following (in order from center):

- Core: carries light signals
- Cladding: reflects light back into core
- Buffer protects light from leaking
- Jacket: prevents physical damage

3. **Twisted pair cable**
   This is also known as Ethernet cable. It consists of 4 color-coded insulated copper wire pairs

   It has 2 types: UTP (unshielded twisted pair cable), and STP (shielded twisted pair cable)

- It can transmit at speed ranging from 10MBPS - 10 GBPS
- STP is expensive (more material for shielding)
- STP gives more noise, and EMI resistance
- Max segment length is 100m

Study of different types of Networking Devices

Common network devices are:

- Hub
  Hub connects multiple deices together. It also acts as repeater and amplifies the signal. They do not perform address functions or packet filtering, only forwarding it to all devices. It has 2 types - single port and multi port.
- Switch
  It maintains limited routing information about internal network nodes. Strands of LAN are usually connected through switch. They improve the network efficiency and security. In can work as the *data link* or *network layer* of OSI model.
- Router
  Routers transmit packets to their destination through interconnected networks using different network topologies. Routers maintain tables about destinations and local connections and communicate using *Routing Information Protocol*, *Border Gateway Protocol* and *Open Shortest Path First Protocol*
- Bridge
  Bridge is used to connect two or more network segments together. Bridges store and forward frames between different segments connected by them. They work at the *physical* & *data link* layer of OSI model. Modern switches, called multiport brides have replaced bridges.
- Gateway
  They work at the *transport* & *session* layer of OSI model. It provides translation between technologies like OSI and TCP/IP. It performs all the functions of router, it is basically a router with translation functionality.
- Modem
  Modems, or *modulator-demodulator* are used to transmit digital signal over telephone lines. It converts digital signal to analog over different frequencies and transmits to modem at receiver station. They work on both *physical* and *data link* layer.
- Repeater
  Repeater is an analog device which receives a signal and transmits it at higher level or power so that the signal can cover greater lengths. It works on the *physical* layer.

Study of different types of Network Topologies

Different types of network topologies are:

- Bus Topology
  Every device on a network is connected to solo main cable line. Data is transmitted in single route, from one point to another.

  Benefit: cost effective, least cable length required
  Drawback: entire network fails if main cable collapses, lower performance with multiple nodes

- Ring Topology
  Each computer is connected together with other computers on both sides, forming a ring-like shape. Data is transmitted in sequential mode, i.e. bit by bit, and routes through each node in the network.

  Benefit: cheap installation and expansion, not affected by heavy traffic
  Drawback: difficult to troubleshoot, difficult to add/delete nodes

- Star Topology
  All nodes are connected to a single node called hub, which can be active or passive.

  Benefit: fast performance, due to low traffic, easy to troubleshoot & upgrade
  Drawbacks: high installation cost, all nodes dependent on hub

- Mesh Topology
  All nodes are connected to all other nodes. It is a point to point connection, and requires $\frac{n \times (n-1)}{2}$ network channels to connect =n= nodes. It uses rounding and flooding technique for transmission.

  Benefit: very robust, easy to diagnose fault, provides privacy and security
  Drawback: challenging to install, configuration is difficult, cable cost is very high

- Tree Topology
  All nodes are connected hierarchically to the root node. It is used mostly in WAN and is an extension of *star* and *bus* topology.

  Benefit: easy to expand network, easy to detect and troubleshoot errors
  Drawback: expensive to other technologies, entire networks collapses if root node fails

- Hybrid Topology
  It comprises of two or more different topologies and has the merits and demerits of all the topologies used.

  Benefit: flexible, durable
  Drawback: difficult to design

Computer Networks
CSE 325
Lab Assignment - 2

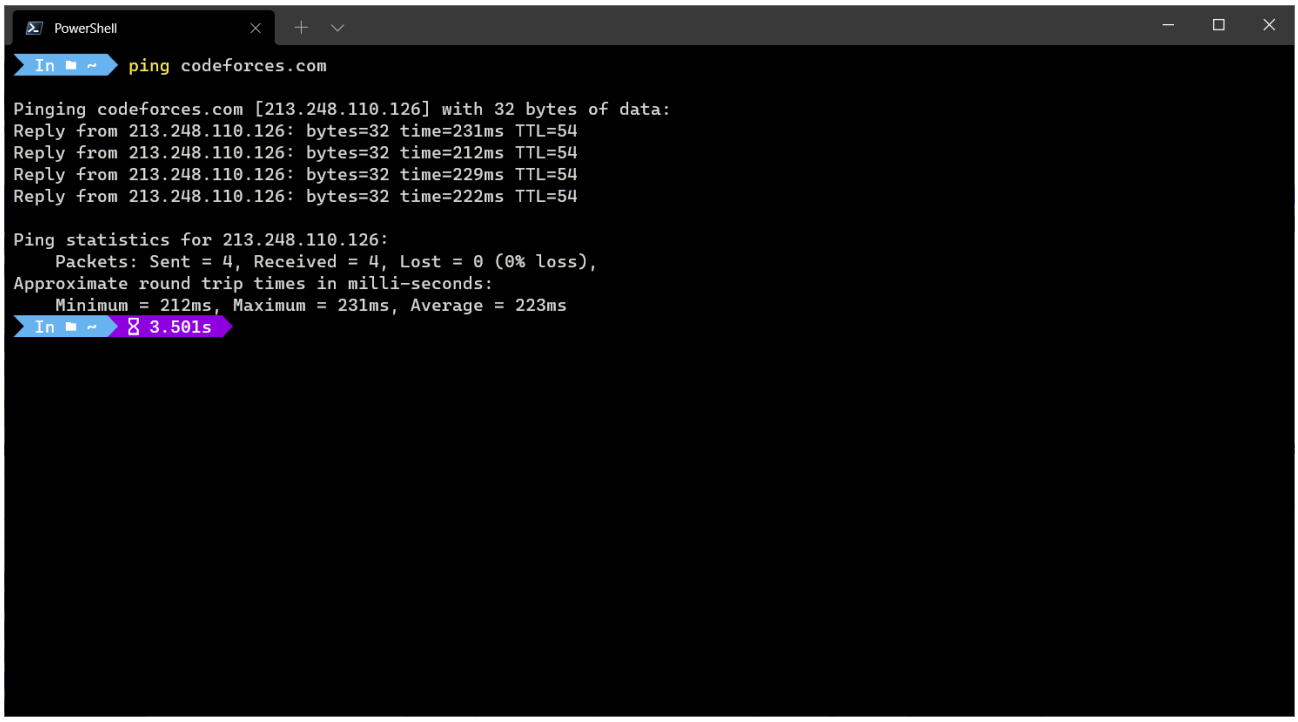
Gyanendra Shukla
CSE - 1
Scholar Number: **191112040**

MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL – 462 003 (INDIA)
January 2022

Contents

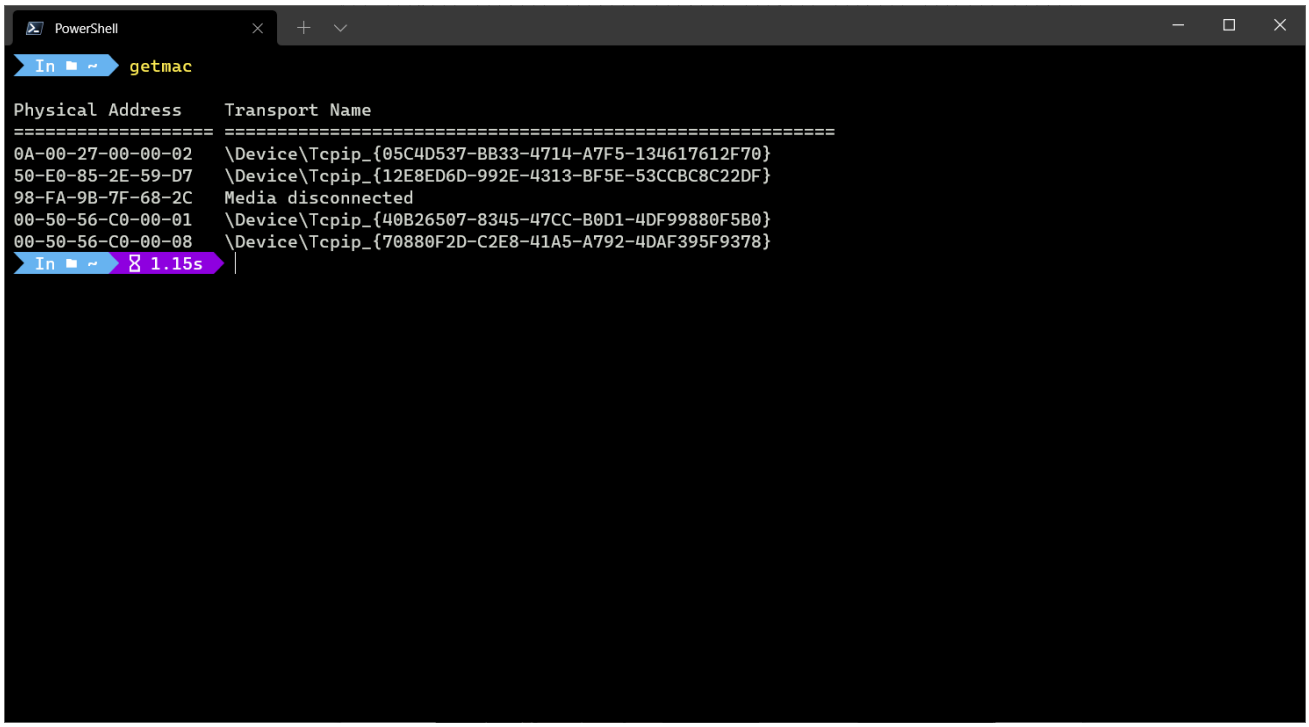
Basic commands in Computer Networking

1. Ping Command:



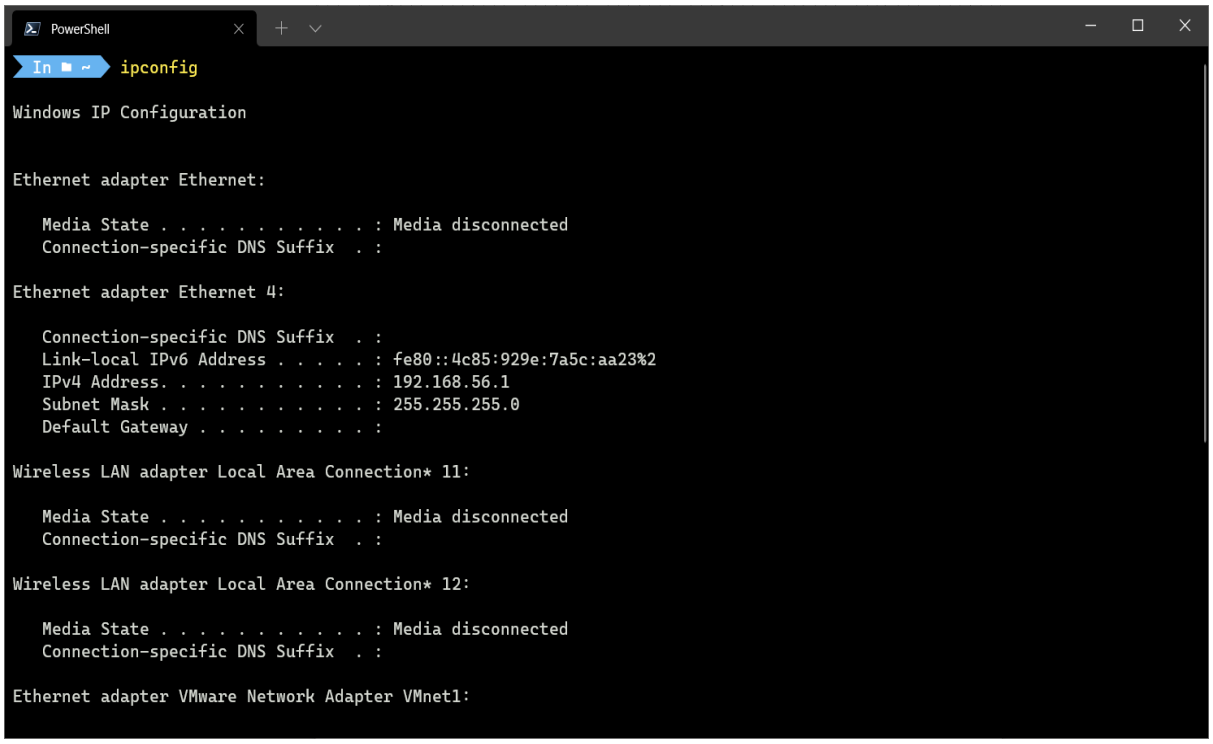
We can use the **ping** command to test whether or not we can make contact with another network device. It could be a device on our network (for instance, our network router) or to a website domain or internet IP address to test our internet connectivity.

2. GETMAC:



We use the getmac command o find the MAC address of the devices connected.

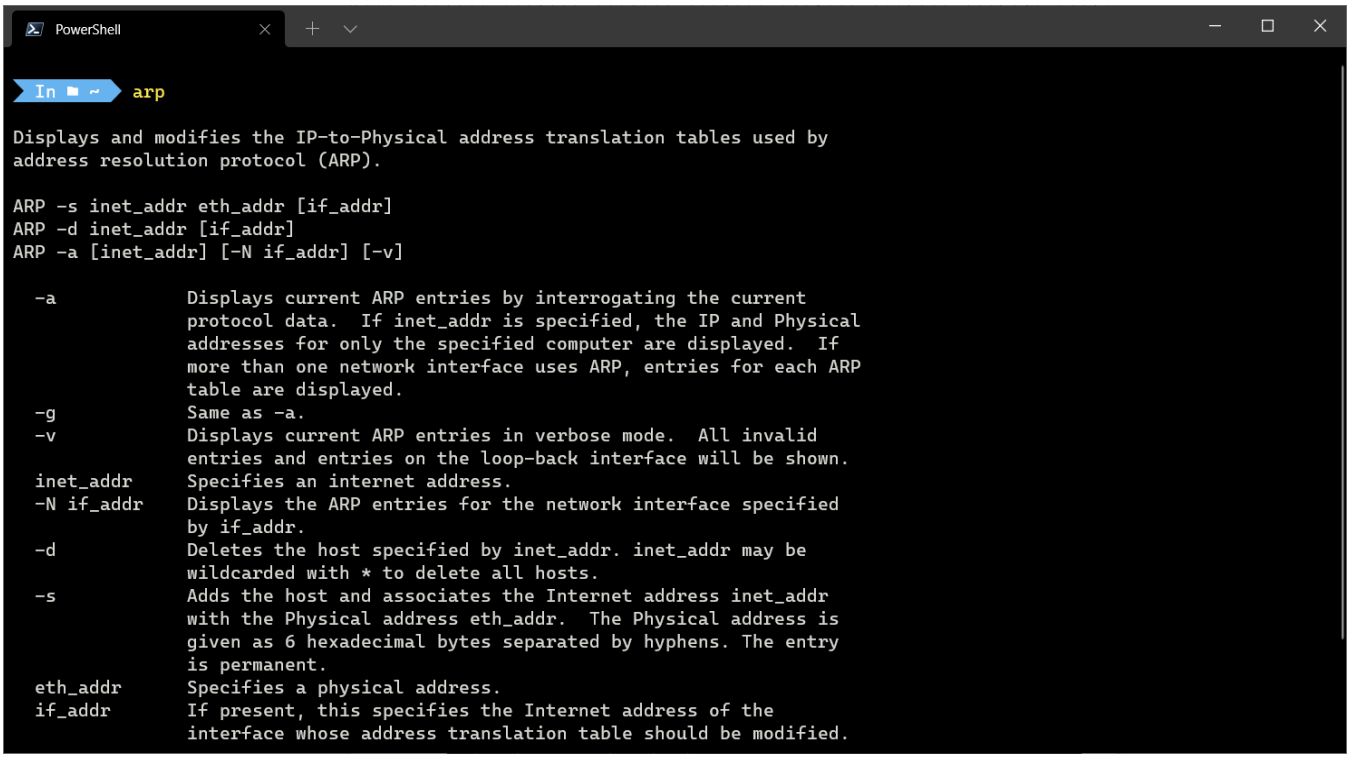
3. IPCONFIG:



Typing **ipconfig** at the terminal will list all available commands, but these include:

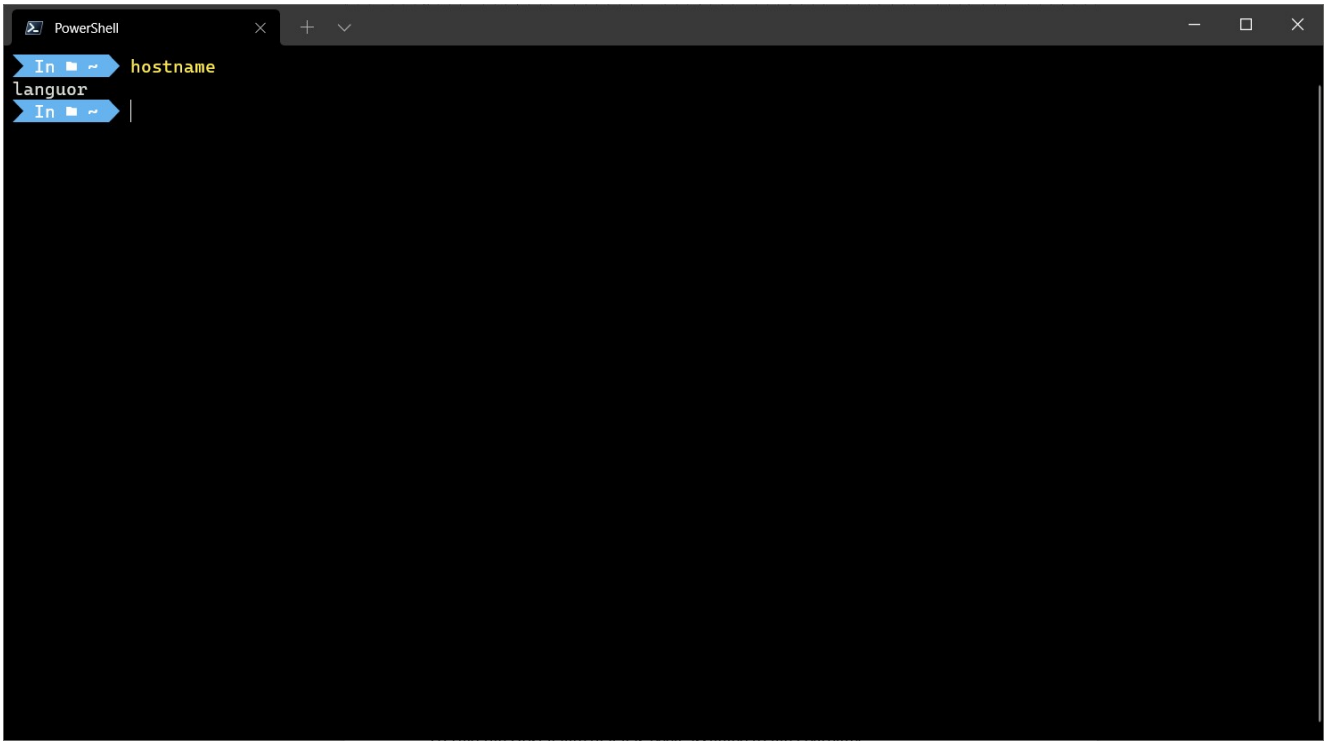
- To view your current network IP address:
  ipconfig getifaddr deviceid
- To view your current network DNS server:
  ipconfig getoption deviceid domain_name_server

4. ARP



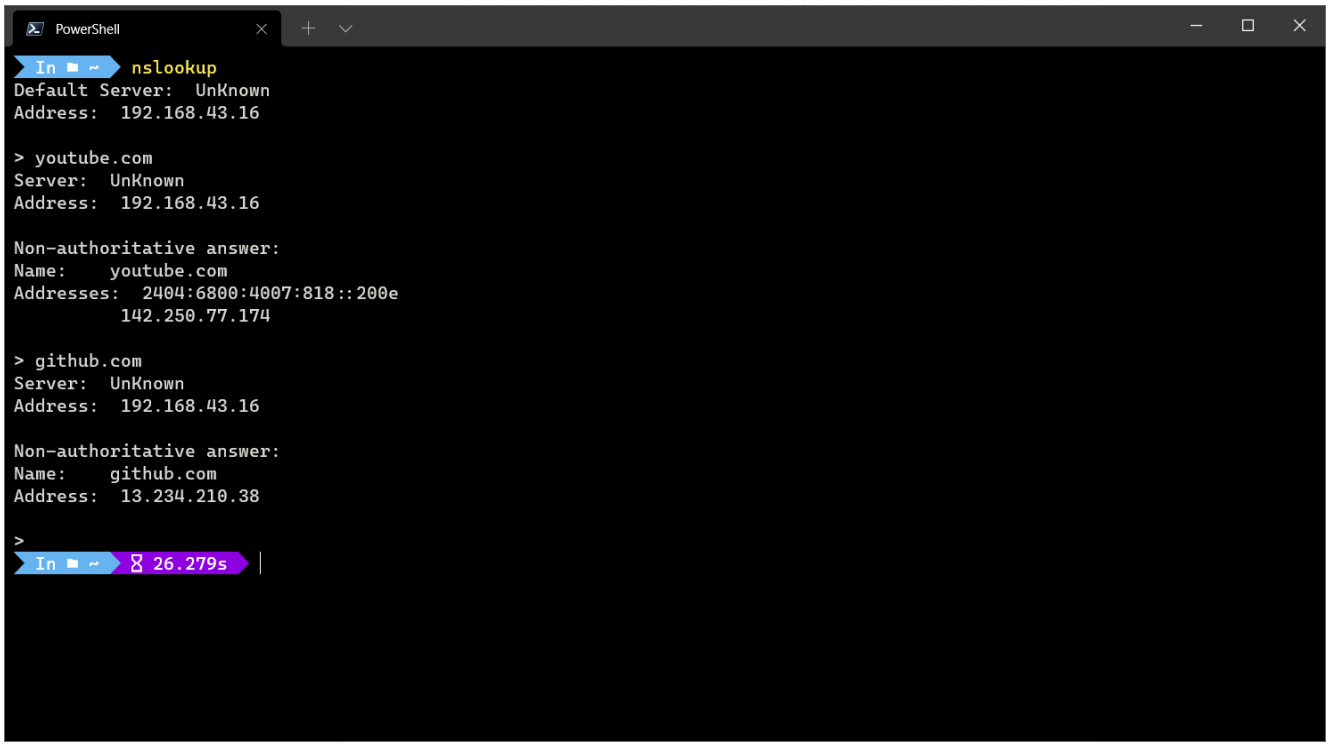
If you want to view a list of all active devices on a local network, you could use the **arp** tool. This will list the IP and MAC addresses for any devices that your Mac has detected on your network, based on the ARP (Address Resolution Protocol) broadcasts those devices have made.

5. HOSTNAME:



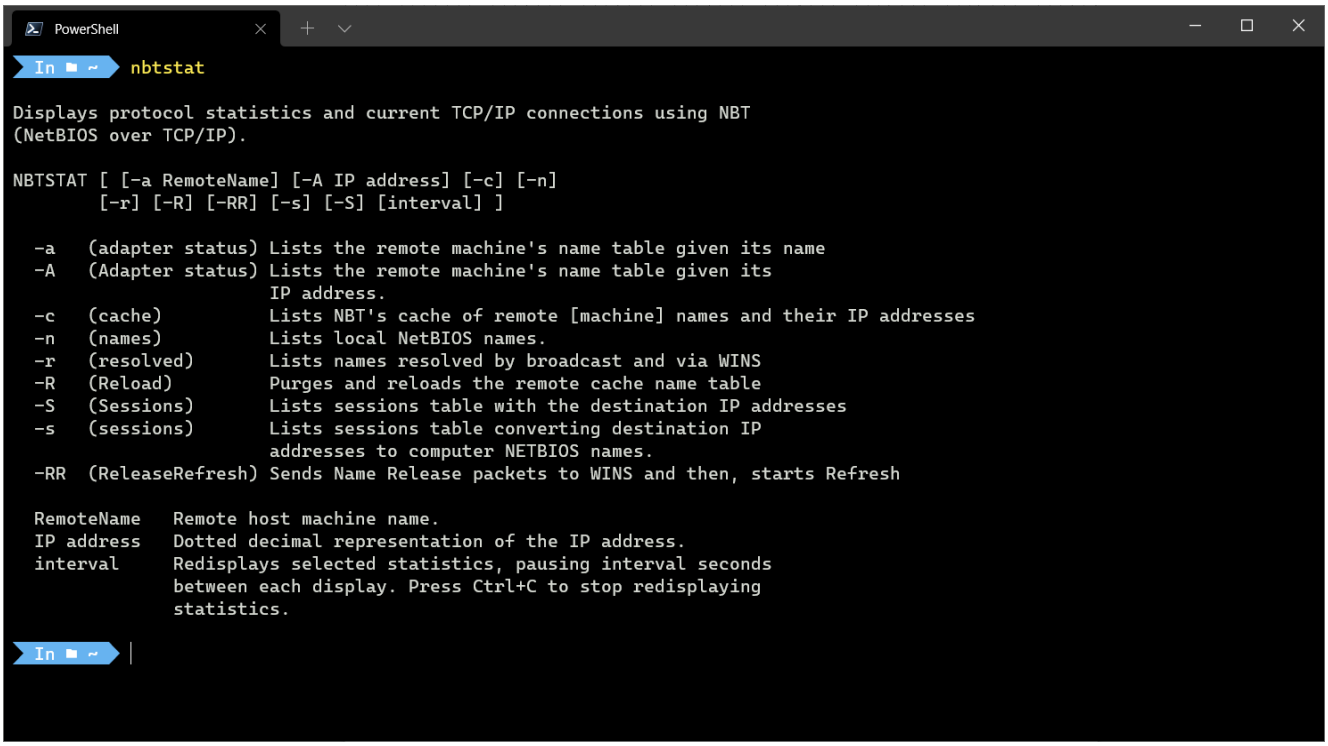
To find the Host name that has been assigned to the computer.

6. NSLOOKUP:



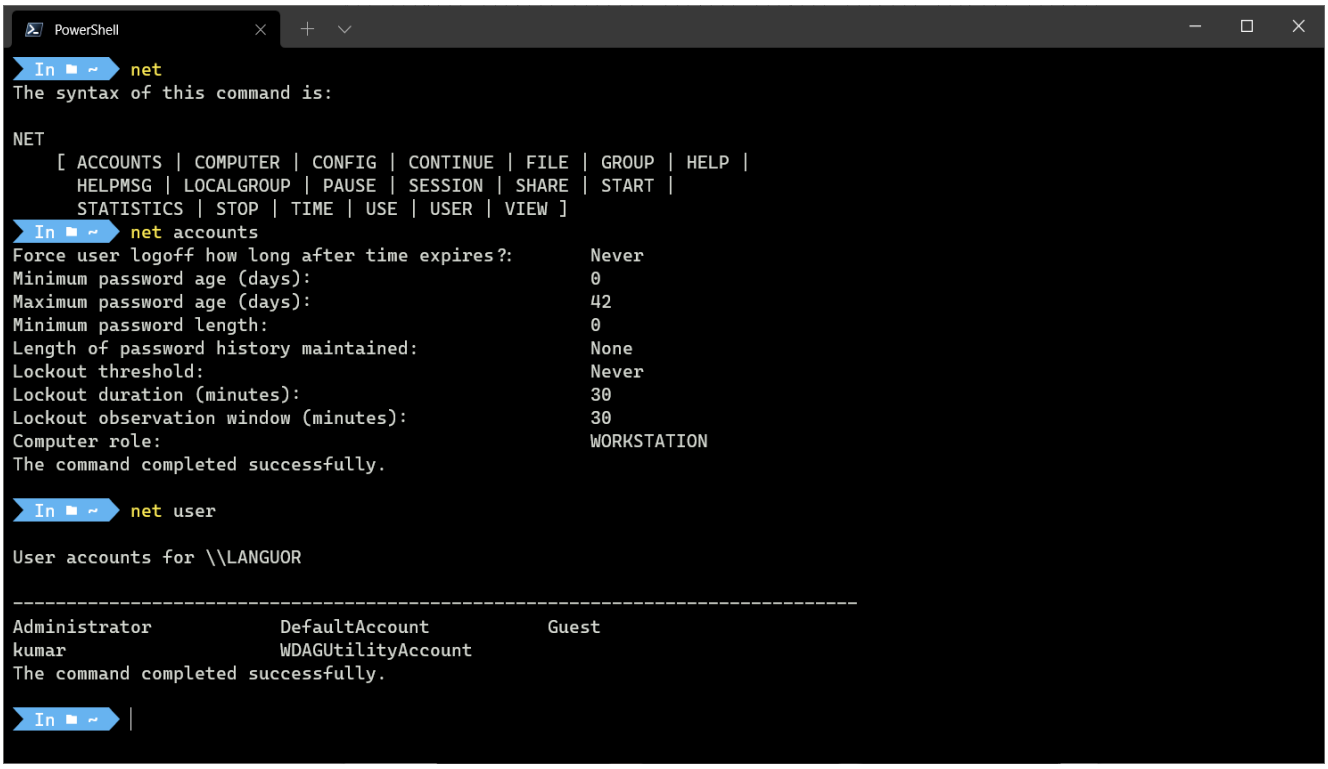
The **nslookup**, which stands for name server lookup command, is a network utility command used to obtain information about internet servers. It provides name server information for the DNS (Domain Name System), i.e. the default DNS server's name and IP Address.

7. NBSTAT:



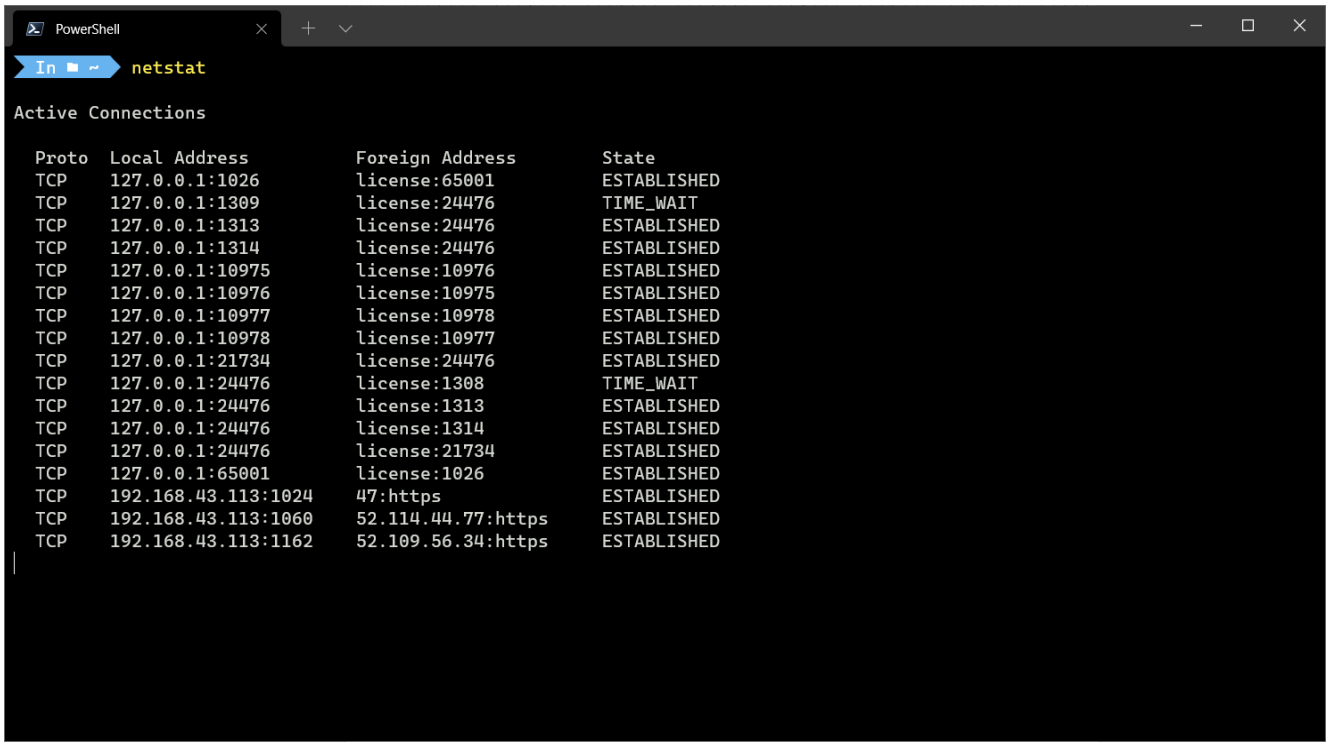
nbtstat command is used to help you diagnose and resolve these problems.

8. NET:



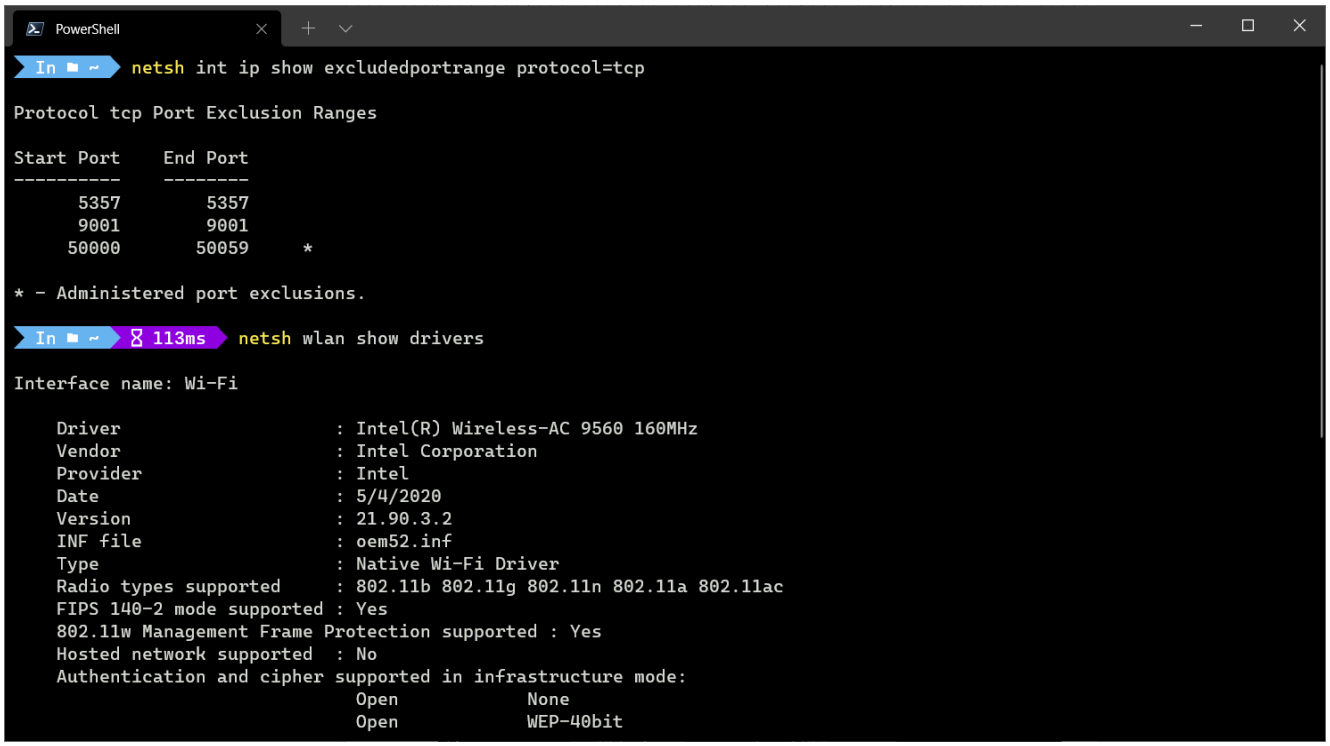
Used to manage many different aspects of a network and its settings such as network shares, users and print jobs.

9. NETSTAT:



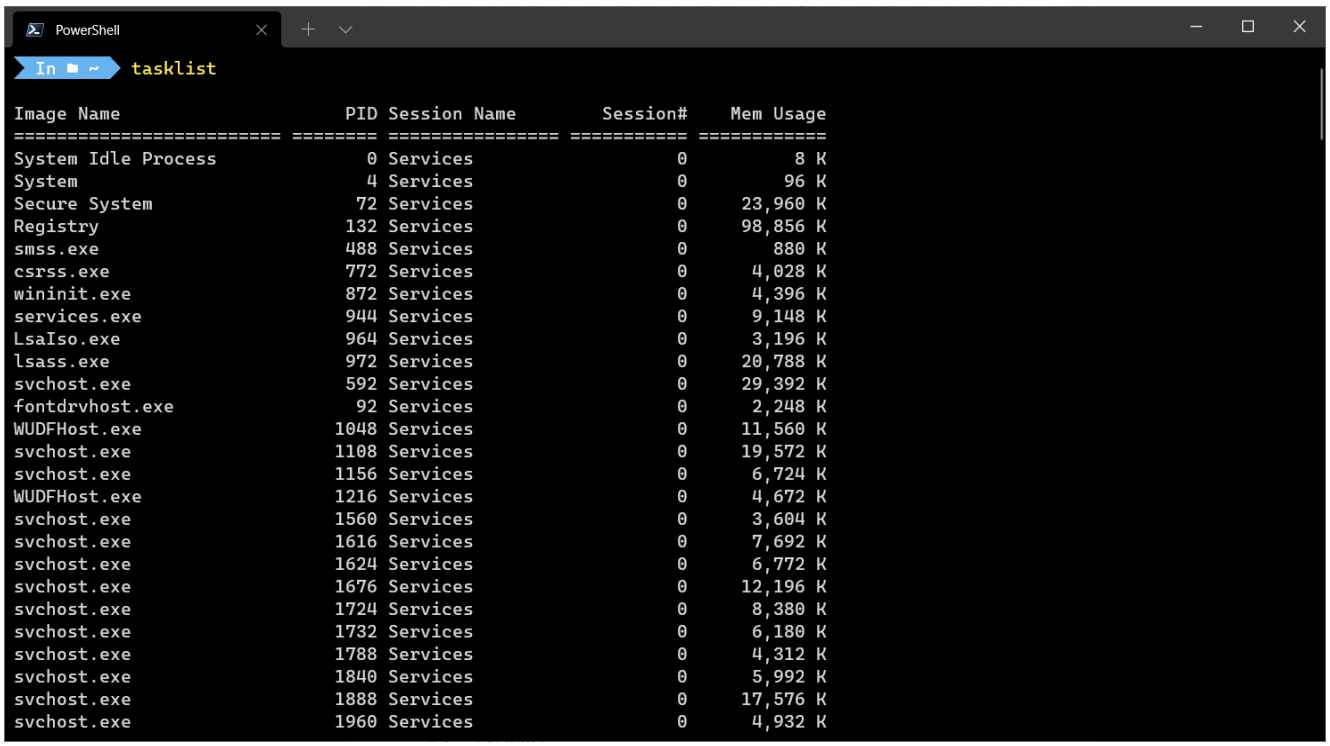
It shows the useful network summary for your device.

10. NETSH:



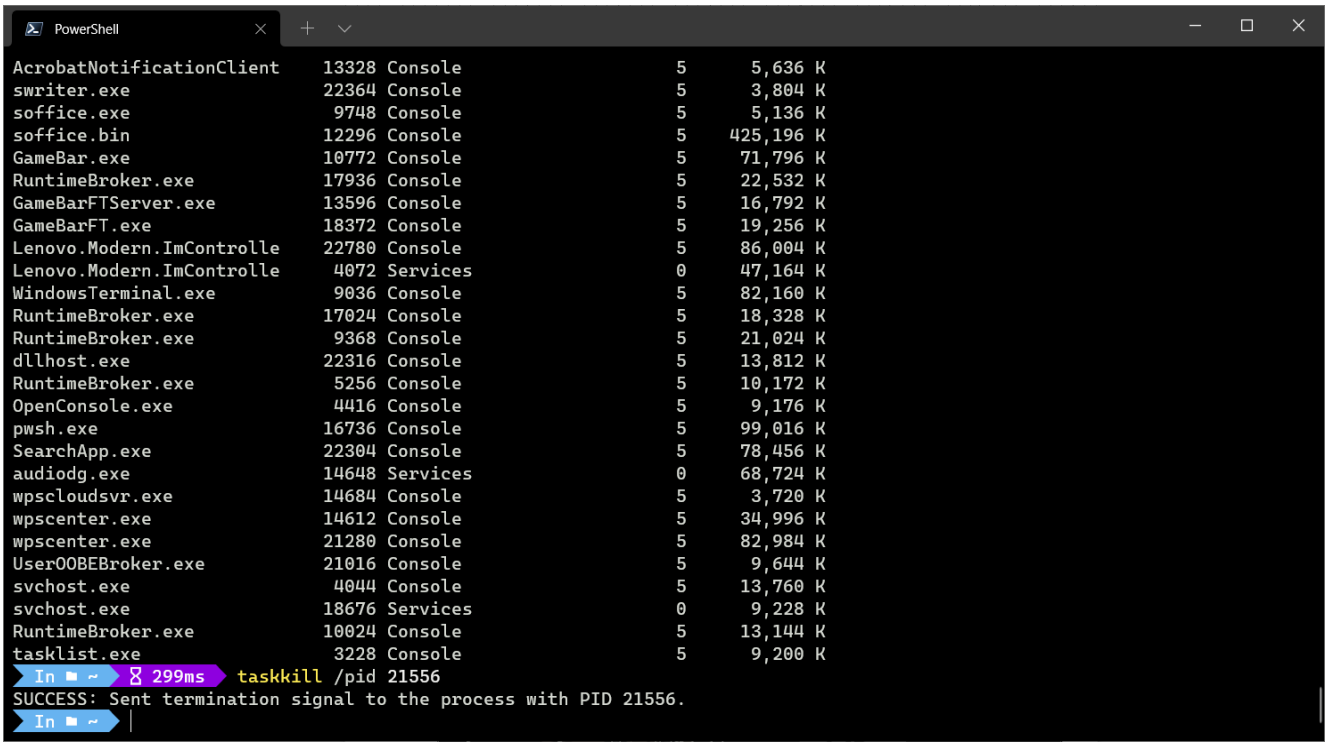
Used to view and configure almost all of the network adapters in your device in much greater detail compared with some other commands.

11. TASKLIST:



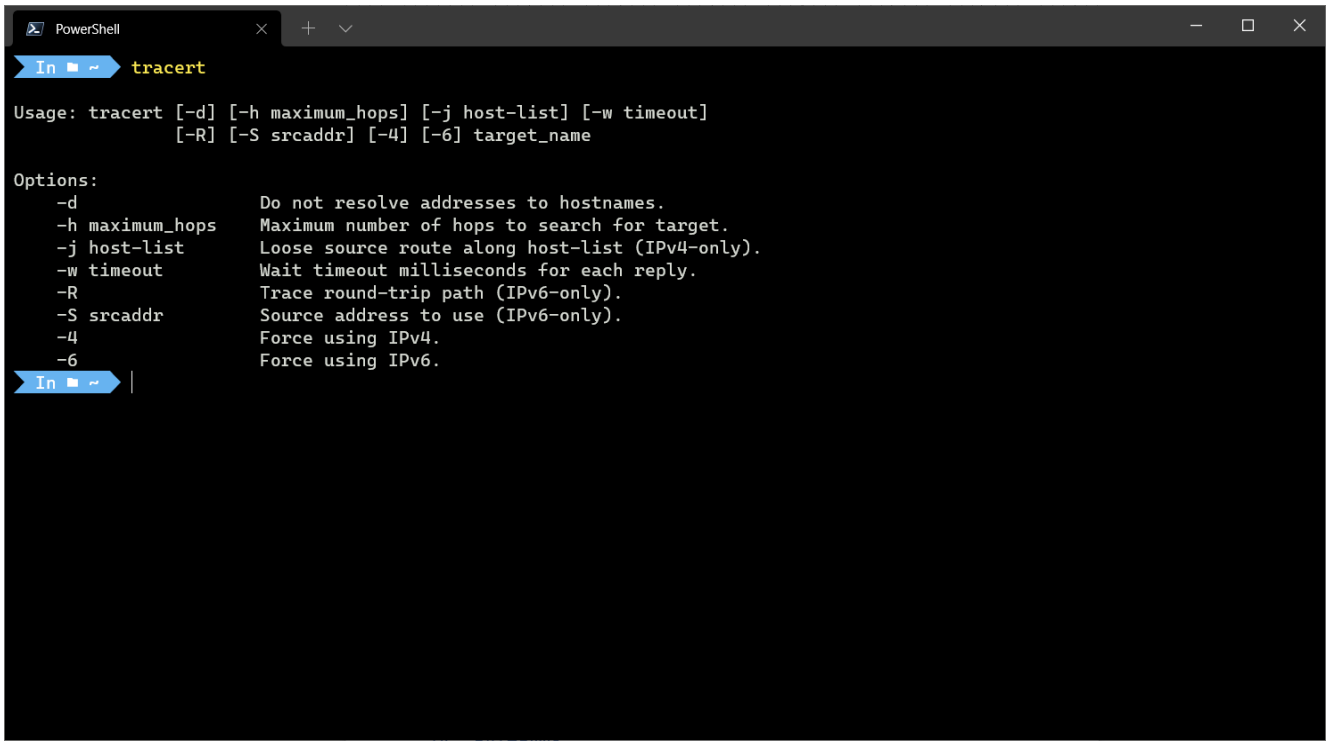
It will show all the running processes and their process id(PID).

12. TASKKILL:



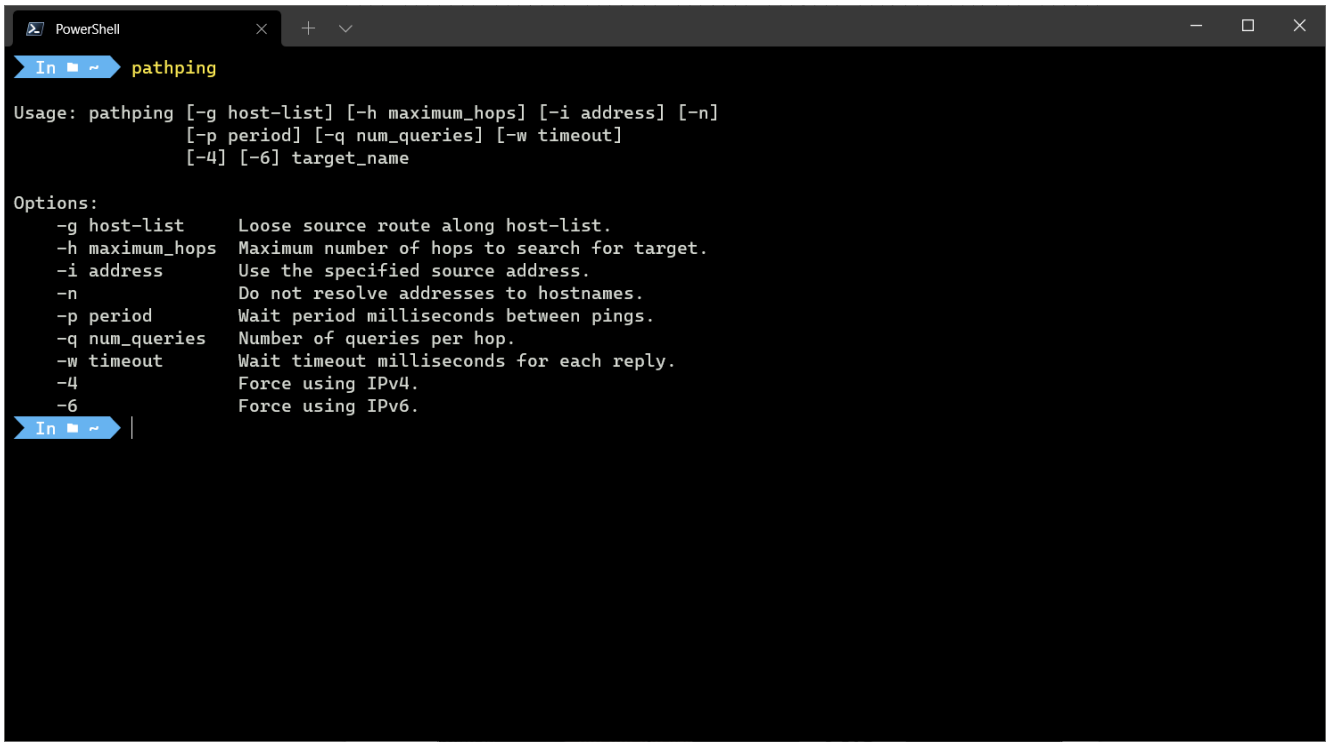
It is used to kill the task by name of its PID provided by TASKLIST command.

13. TRACERT:



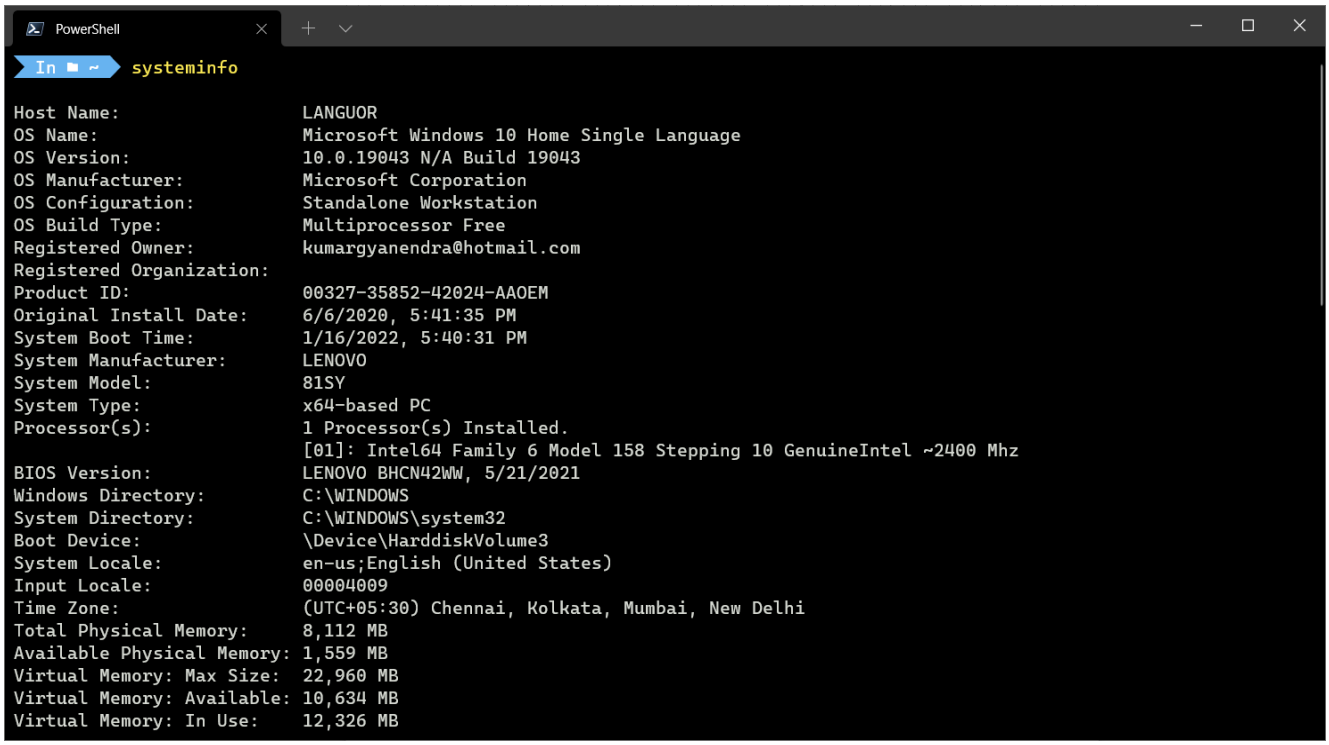
By using the tracert command you can trace the route a packet takes before reaching its destination, and see information on each “hop” along the route.

14. PATHPING:



It combines that best of both ping and tracert into a single utility.

15. SYSTEMINFO:



It shows the details of the processor used, the version of Windows, or what the boot device is configured as.

Computer Network Lab
CSE-325

Assignment - 3

Submitted by -
Gyanendra Shukla
CSE 1
191112040

To Write a Socket Program to implement ECHO

An echo server is a program that listens for a message from a client and then sends the same message back to the client.

I've implemented the server in python. The server listens on port 12345 for any message and returns the message with current timestamp prepended.

Server

```python
from datetime import datetime
import socket

class Server:
    def __init__(self, port) -> None:
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind(('', port))
        self.sock.listen()
        print('Server started on port {}'.format(port))

    def run(self) -> None:
        while True:
            conn, addr = self.sock.accept()
            print('Got connection from', addr)
            with conn:
                while True:
                    data = conn.recv(1024)
                    if not data:
                        break
                    else:
                        print(f'Got: {data.decode()}')
                    cur_time = datetime.now()
                    x = f'{cur_time} : {data.decode()}'
                    conn.sendall(x.encode())


    def __del__(self) -> None:
        self.sock.close()
        print('Server closed')

```

```
32
33 if __name__ == '__main__':
34     server = Server(12345)
35     server.run()
36
```

Client

```python
import socket

class Client:
    def __init__(self, host, port) -> None:
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((host, port))
        print('Connected to {}:{}'.format(host, port))

    def run(self) -> None:
        while True:
            data = input('> ')
            if not data:
                break
            self.sock.sendall(data.encode())
            data = self.sock.recv(1024)
            print(data.decode())

    def __del__(self) -> None:
        self.sock.close()
        print('Connection closed')


if __name__ == '__main__':
    client = Client('127.0.0.1', 12345)
    client.run()
```

Output

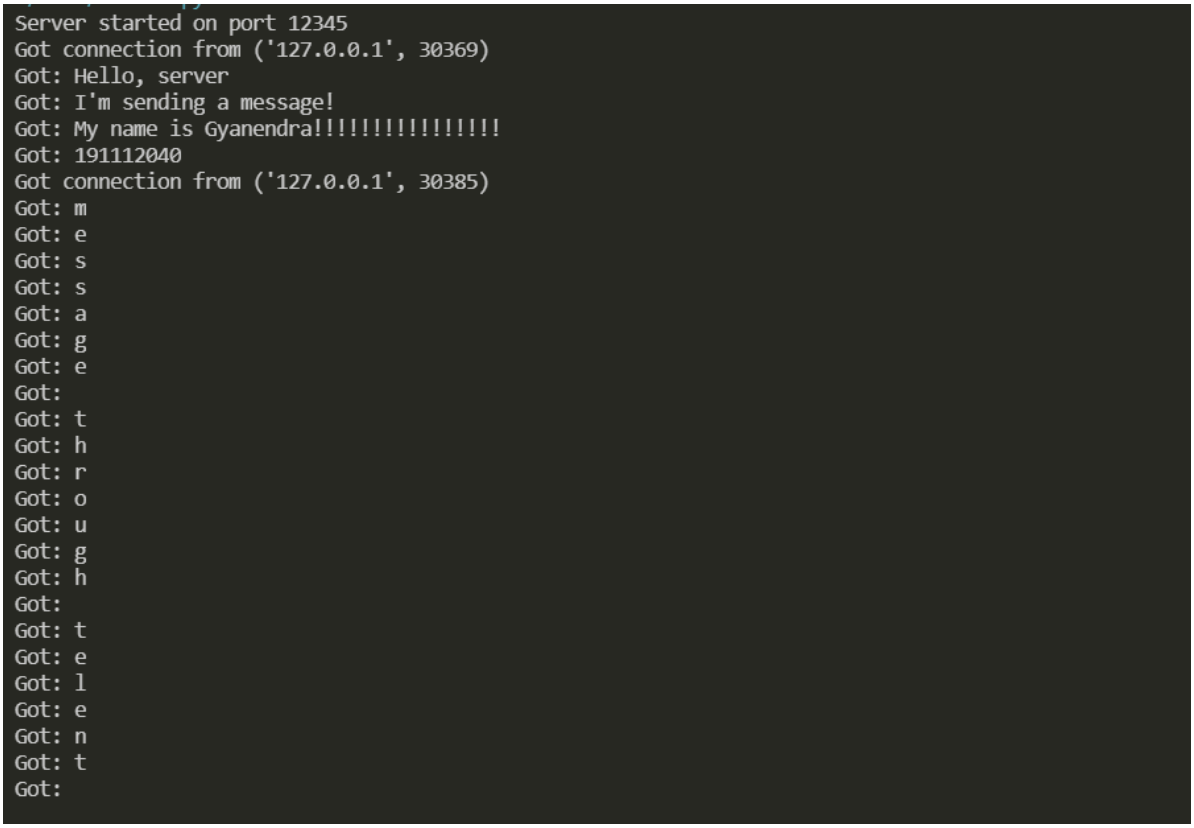


Fig: Messages received by the server

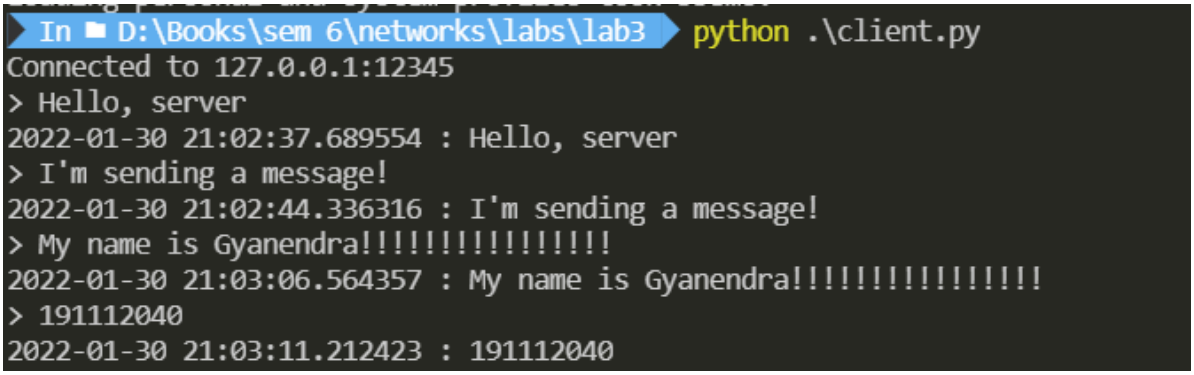


Fig: Messages sent by the client and then messages timestamped by the server returned to the client

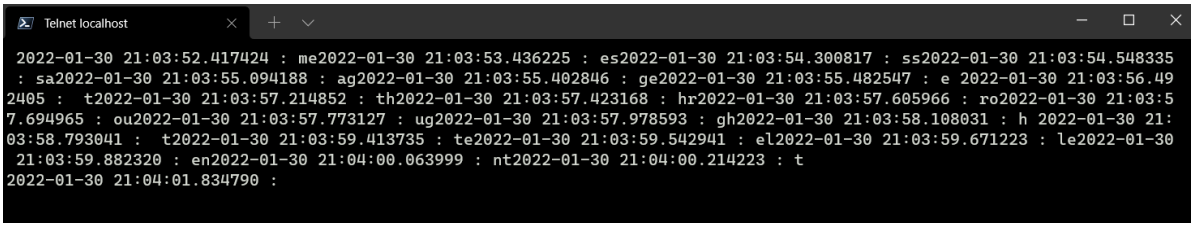


Fig: Connecting on server through telnet

Computer Network Lab
CSE-325

Assignment - 3

Submitted by -
Gyanendra Shukla
CSE 1
191112040

To write a Socket Program to implement CHAT between client & server

I implemented a program where multiple clients can connect to the server and send messages to each other. The Server listens to any incoming connection and sends the message to all the clients.

Chat Server

```
import socket
import select

IP = "127.0.0.1"
PORT = 12345

class ChatServer:
    HEADER_LENGTH = 10
    def __init__(self, ip, port) -> None:
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
        self.server_socket.bind((ip, port))
        self.server_socket.listen()

        self.socket_list = [self.server_socket]
        self.clients = {}
        print("Server started on {}:{}".format(ip, port))

    def receive_message(self, client_socket):
        try:
            message_header = client_socket.recv(self.HEADER_LENGTH)

            if not len(message_header):
                return False

            message_length = int(message_header.decode("utf-8").strip())

```

```
            return {'header': message_header, 'data':
                    client_socket.recv(message_length)}

        except:
            return False

    def start(self):
        while True:
            read_sockets, _, exception_sockets =
select.select(self.socket_list,
                                                                  [],
self.socket_list)
            for notified_socket in read_sockets:
                if notified_socket == self.server_socket:
                    client_socket, client_address =
self.server_socket.accept()

                    user = self.receive_message(client_socket)

                    if user is False:
                        continue

                    self.socket_list.append(client_socket)
                    self.clients[client_socket] = user

                    print("Accepted new connection from {}:{}, username: {}"
                            .format(*client_address,
user['data'].decode("utf-8")))

                else:
                    message = self.receive_message(notified_socket)

                    if message is False:
                        print("Closed connection from: {}"
                              .format(self.clients[notified_socket]['data']
                                      .decode("utf-8")))

                        self.socket_list.remove(notified_socket)
                        del self.clients[notified_socket]
                        continue

                    user = self.clients[notified_socket]
                    print("Received message from {}: {}"
                          .format(user['data'].decode("utf-8"),
message['data']
                                  .decode("utf-8")))

                    for client_socket in self.clients:
                        if client_socket != notified_socket:
                            client_socket.send(user['header'] + user['data']
                                               + message['header'] +
message['data'])

            for notified_socket in exception_sockets:
                # If we've got exceptional socket, probably it's broken one
                # so we need to remove it from socket list
                self.socket_list.remove(notified_socket)
                del self.clients[notified_socket]
```

```
80
81
82  if __name__ == "__main__":
83      server = ChatServer(IP, PORT)
84      server.start()
```

Chat Client

```
1   import socket
2   import select
3   import errno
4   import sys
5
6
7   IP = "127.0.0.1"
8   PORT = 12345
9
10  class ChatClient:
11      HEADER_LENGTH = 10
12      def __init__(self, ip, port, username) -> None:
13          self.client_socket = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
14          self.client_socket.connect((ip, port))
15          self.client_socket.setblocking(False)
16
17          self.username = username.encode("utf-8")
18          self.username_header = f"{len(self.username):
    <{self.HEADER_LENGTH}}".encode("utf-8")
19          self.client_socket.send(self.username_header + self.username)
20
21      def start(self):
22          while True:
23              message = input(f"{self.username}> ")
24              if message:
25                  message = message.encode("utf-8")
26                  message_header = f"{len(message):
    <{self.HEADER_LENGTH}}".encode("utf-8")
27                  self.client_socket.send(message_header + message)
28
29              try:
30                  while True:
31                      username_header =
    self.client_socket.recv(self.HEADER_LENGTH)
32                      if not len(username_header):
33                          print("Connection closed by the server")
34                          sys.exit()
35
36                      username_length = int(username_header.decode("utf-
    8").strip())
37                      username =
    self.client_socket.recv(username_length).decode("utf-8")
38
39                      message_header =
    self.client_socket.recv(self.HEADER_LENGTH)
40                      message_length = int(message_header.decode("utf-
    8").strip())
```

```
                    message =
self.client_socket.recv(message_length).decode("utf-8")

                    print(f"{username}> {message}")
            except IOError as err:
                if err.errno != errno.EAGAIN and err.errno !=
errno.EWOULDBLOCK:
                    print(f"Reading error: {str(err)}")
                    sys.exit()

                continue

            except Exception as e:
                print(f"Reading error: {str(e)}")
                sys.exit()

if __name__ == "__main__":
    chat_client = ChatClient(IP, PORT, str(sys.argv[1]))
    chat_client.start()
```

Output

```
In D:\Books\sem 6\networks\labs\lab4 107ms python .\ChatServer.py
Server started on 127.0.0.1:12345
Accepted new connection from 127.0.0.1:1076, username: gyan
Accepted new connection from 127.0.0.1:1246, username: al
Received message from al: Hello Gyan
Received message from gyan: Hi, al! This is some message.
Received message from al: some
Received message from al: more
Received message from al: messages
Received message from al: from al
Received message from gyan: ok bye!
Received message from al: bye!!
```

Fig: Chat Server

```
In D:\Books\sem 6\networks\labs\lab4 223ms python .\ChatClient.py gyan
b'gyan'>
al> Hello Gyan
b'gyan'> Hi, al! This is some message.
b'gyan'>
al> some
al> more
al> messages
al> from al
b'gyan'> ok bye!
b'gyan'>
```

Fig: Chat Client 1

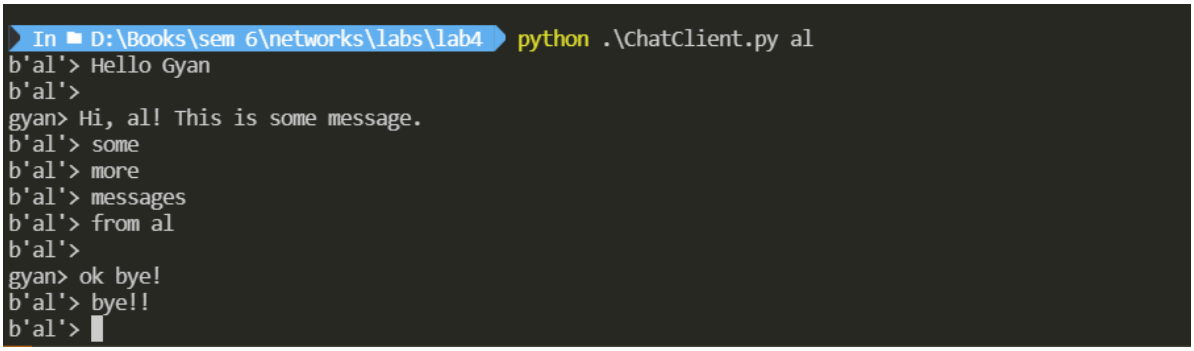


Fig: Chat Client 2

To write a Socket Program to implement File Transfer between client & server

I wrote a program to implement file transfer between client and server. The server receives the file from the client. The client sends the file it has to send to the server through command line args.

The received file has a `recv-` prefix.

File Transfer Server

```python
import socket
import os

IP = "127.0.0.1"
PORT = 12345

class FileServer:
    SEPARATOR = "<SEPARATOR>"
    BUFFER_SIZE = 4096

    def __init__(self, ip, port) -> None:
        self.sock = socket.socket()
        self.sock.bind((ip, port))
        self.sock.listen()
        print(f"Listening on {ip}:{port}")

    def receive(self):
        client_socket, address = self.sock.accept()
        received = client_socket.recv(self.BUFFER_SIZE).decode()
        filename, filesize = received.split(self.SEPARATOR)

        filename = "recv-" + os.path.basename(filename)
        filesize = int(filesize)

        with open(filename, "wb") as f:
            print(f"Incoming file, saving as {filename}")
            while True:
                bytes_read = client_socket.recv(self.BUFFER_SIZE)
                if not bytes_read:
                    # we've completed receiving files
                    break
```

```
                f.write(bytes_read)
        print(f"Done receiving {filename}")

        client_socket.close()
        self.sock.close()


if __name__ == "__main__":
    server = FileServer(IP, PORT)
    server.receive()

```

File Transfer Client

```
import socket
import sys
import os

class FileClient:
    SEPARATOR = "<SEPARATOR>"
    BUFFER_SIZE = 4096

    def __init__(self, ip, port) -> None:
        self.sock = socket.socket()
        print(f"Connecting to {ip}:{port}")
        self.sock.connect((ip, port))
        print(f"Connected to {ip}:{port}")

    def send(self, filename):
        if not os.path.isfile(filename):
            print(f"{filename} does not exist!")
            return
        filesize = os.path.getsize(filename)
        self.sock.send(f"{filename}{self.SEPARATOR}{filesize}".encode())

        with open(filename, "rb") as f:
            print(f"Sending {filename}")
            while True:
                bytes_read = f.read(self.BUFFER_SIZE)
                if not bytes_read:
                    # we've completed sending files
                    break
                self.sock.sendall(bytes_read)
        print(f"Done sending {filename}")

        self.sock.close()


if __name__ == "__main__":
    client = FileClient("127.0.0.1", 12345)
    filename = sys.argv[1]
    client.send(filename)
```

Output

```
In D:\Books\sem 6\networks\labs\lab4 Get-ChildItem

    Directory: D:\Books\sem 6\networks\labs\lab4

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          2/7/2022 10:57 AM           8712 191112040.md
-a---          2/7/2022 10:49 AM          12311 cclient1.png
-a---          2/7/2022 10:48 AM          11421 cclient2.png
-a---          2/7/2022  9:29 AM           2165 ChatClient.py
-a---          2/7/2022  9:28 AM           2999 ChatServer.py
-a---          2/7/2022 10:49 AM          22012 cserver.png
-a---          2/7/2022  9:06 AM           1119 FileClient.py
-a---          2/7/2022 10:56 AM           1168 FileServer.py
```

Fig: Files before sending

```
In D:\Books\sem 6\networks\labs\lab4 python .\FileServer.py
Listening on 127.0.0.1:12345
Incoming file, saving as recv-ChatClient.py
Done receiving recv-ChatClient.py
In D:\Books\sem 6\networks\labs\lab4 15.895s
```

Fig: File Transfer Server

```
In D:\Books\sem 6\networks\labs\lab4 python .\FileClient.py .\ChatClient.py
Connecting to 127.0.0.1:12345
Connected to 127.0.0.1:12345
Sending .\ChatClient.py
Done sending .\ChatClient.py
In D:\Books\sem 6\networks\labs\lab4 156ms
```

Fig: File Transfer Client

```
In D:\Books\sem 6\networks\labs\lab4 156ms Get-ChildItem

    Directory: D:\Books\sem 6\networks\labs\lab4

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          2/7/2022 10:57 AM           8712 191112040.md
-a---          2/7/2022 10:49 AM          12311 cclient1.png
-a---          2/7/2022 10:48 AM          11421 cclient2.png
-a---          2/7/2022  9:29 AM           2165 ChatClient.py
-a---          2/7/2022  9:28 AM           2999 ChatServer.py
-a---          2/7/2022 10:49 AM          22012 cserver.png
-a---          2/7/2022  9:06 AM           1119 FileClient.py
-a---          2/7/2022 10:56 AM           1168 FileServer.py
-a---          2/7/2022 11:00 AM           2165 recv-ChatClient.py
```

Fig: Files after sending (recv-ChatClient.py at the bottom)

Computer Network Lab

CSE-325

Assignment - 5

Submitted by -
Gyanendra Shukla
CSE 1
191112040

To implement Remote Command Execution(RCE)

I implemented a program where a client connects to a server and execute commands on the server. I take a command from the client and send it on the server. On the server, I use python's `subprocess` module to run the command and store the output in a pipe. This is equivalent to C's `popen` function. I then take the output from the pipe and send it back to the client.

RCE Server

```
import socket
import subprocess

HOST = "127.0.0.1"
PORT = 4204

class Server:
    def __init__(self) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.bind((HOST, PORT))
        self.socket.listen()

    def accept(self):
        """
        This function accpets a client connection and then
        if it recieves any data, it sends it to `execute`
        function and sends back the returned value back
        to the client.
        """
        conn, addr = self.socket.accept()
        print(f"Connection from {addr} has been established!")
        with conn:
            while True:
                data = conn.recv(1024)
```

```
                if not data:
                    break
                print(f"Received data: {data.decode('utf-8')}")
                conn.sendall(self.execute(data.decode('utf-8')))

    def execute(self, command):
        """
        This function takes a command as input and executes it,
        it then returns the output of the command.
        """
        print("Executing command: \n" + command)
        x = subprocess.run(command, shell=True, stdout=subprocess.PIPE)
        if x.returncode == 0 and len(x.stdout) > 0:
            return x.stdout
        return "".encode('utf-8')

if __name__ == "__main__":
    server = Server()
    server.accept()
```

Chat Client

```
import socket
import subprocess

HOST = "127.0.0.1"
PORT = 4204

class Client:
    def __init__(self) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((HOST, PORT))
        print("This program executes commands on the server and shows you the output.")

    def start(self):
        """
        This function takes input from the user and sends it to the server.
        The server then executes the command and returns the output.
        This function then displays that output.
        """
        while True:
            command = input("Enter command: ")
            self.socket.sendall(command.encode('utf-8'))
            data = self.socket.recv(1024)
            print(data.decode('utf-8'))
```

```python
if __name__ == "__main__":
    server = Client()
    server.start()
```

Output

```
In D:\Books\sem 6\networks\labs\lab5\server python server.py
Connection from ('127.0.0.1', 8706) has been established!
Received data: dir
Executing command:
dir
Received data:         print("Executing command: \n" + command)
Executing command:
        print("Executing command: \n" + command)
Received data: echo "echoing this on server, but the output will be shown here"
Executing command:
echo "echoing this on server, but the output will be shown here"
```

Fig: RCE Server. It takes in command, executes and sends output to server

```
In D:\Books\sem 6\networks\labs\lab5\client 444ms python client.py
This program executes commands on the server and shows you the output.
Enter command: dir
 Volume in drive D is Data
 Volume Serial Number is 7AFF-9A89

 Directory of D:\Books\sem 6\networks\labs\lab5\server

02/14/2022  12:14 PM    <DIR>          .
02/14/2022  12:14 PM    <DIR>          ..
02/14/2022  12:23 PM             1,018 server.py
               1 File(s)          1,018 bytes
               2 Dir(s)  411,846,053,888 bytes free

Enter command:         print("Executing command: \n" + command)
Unable to initialize device PRN

Enter command: echo "echoing this on server, but the output will be shown here"
"echoing this on server, but the output will be shown here"
```

Fig: RCE Client. It sends command to server and prints the output from server.

Computer Network Lab

CSE-325

Assignment - 6

Submitted by -
Gyanendra Shukla
CSE 1
191112040

To implement Client-Server application using UDP

I made an echo client and server using UDP protocol. The server socket is started with `SOCK_DGRAM` argument to start a UDP server. The client socket is started with `SOCK_DGRAM` argument to start a UDP client. I take in input in the client application in the `run` function. If the input is `exit`, I close the client socket and exit. In the server program, if I recieve an empty message, I close the server socket and exit.

UDP Server

```
import socket
import datetime

class UDPServer:
    def __init__(self, addr: str, port: int) -> None:
        """
        Starting a UDP server with SOCK_DGRAM on the given address and port.
        """
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind((addr, port))
        print(f"\nServer started on {addr}:{port}\n")


    def run(self) -> None:
        """
        Listen for incoming messages and echo them back to the client.
        """
        while True:
            data, addr = self.socket.recvfrom(1024)
            print(f"Got: {data.decode()}")

            if data:
                d = f"{datetime.datetime.now()} : {data.decode()}"
```

```
                self.socket.sendto(d.encode(), addr)
            else:
                self.socket.sendto("".encode(), addr)
                self.socket.close()
                break


if __name__ == '__main__':
    server = UDPServer("127.0.0.1", 1234)
    server.run()
```

UDP Client

```
import socket

class UDPClient:
    def __init__(self) -> None:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def send(self, message) -> None:
        try:
            sent = self.sock.sendto(message, ("localhost", 1234))
            data, server = self.sock.recvfrom(1024)
            print(data.decode())
        except Exception as e:
            print(e)

    def close(self) -> None:
        self.sock.close()

    def run(self) -> None:
        while True:
            message = input('> ')
            if (message=='exit'):
                self.close()
                break
            self.send(message.encode())

if __name__ == '__main__':
    client = UDPClient()
    client.run()
```

Output

```
In D:\Books\sem 6\networks\labs\lab6 python .\server.py

Server started on 127.0.0.1:1234

Got: hello
Got: this is a message
Got: from
Got: 191112040
Got: Gyanendra
Got: It is over UDP
Got: This is Computer Network Lab assignment 6.
Got:
In D:\Books\sem 6\networks\labs\lab6 3m 50.125s
```

Fig: UDP Server. It takes in command, adds timestamp and echoes back to the client.

```
In D:\Books\sem 6\networks\labs\lab6 python .\client.py
> hello
2022-02-20 22:27:49.090828 : hello
> this is a message
2022-02-20 22:27:53.198262 : this is a message
> from
2022-02-20 22:27:54.763092 : from
> 191112040
2022-02-20 22:27:57.187089 : 191112040
> Gyanendra
2022-02-20 22:27:59.485370 : Gyanendra
> It is over UDP
2022-02-20 22:28:03.317134 : It is over UDP
> This is Computer Network Lab assignment 6.
2022-02-20 22:29:08.233964 : This is Computer Network Lab assignment 6.
>

> exit
In D:\Books\sem 6\networks\labs\lab6 3m 47.683s
```

Fig: UDP Client. It sends command to server and prints the output from server.

Computer Network Lab
CSE-325

Assignment - 7

Submitted by -
Gyanendra Shukla
CSE 1
191112040

To write a Socket Program to implement CHAT between client & server

I implemented a chat program on the UDP protocol. The client enters the IP address and the port number of the server it wants to chat to.

Chat Server

```
import os
import socket
import sys
import threading
import datetime

class UDPChatServer:
    def __init__(self, name, client_addr=None, client_port=None) -> None:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(('127.0.0.1', 1234))
        self.name = name
        self.client_addr = "127.0.0.1"
        self.client_port = 1235

    def send(self):
        while True:
            message = input('>> ')
            if (message=='exit'):
                break
            message = f'{self.name} [{datetime.datetime.now()}]: {message}'
            self.socket.sendto(message.encode(), (self.client_addr,
self.client_port))
        self.socket.close()
        sys.exit(0)

    def recieve(self):
        while True:
            data, addr = self.socket.recvfrom(1024)
            print(data.decode())
            print("\n>> ", end="")
```

```
30
31
32
33 if __name__ == '__main__':
34     name = input("Enter your name: ")
35     server = UDPChatServer(name)
36     x1 = threading.Thread(target=server.send)
37     x2 = threading.Thread(target=server.recieve)
38     x1.start()
39     x2.start()
```

Chat Client

```
1  import os
2  import socket
3  import sys
4  import threading
5  import datetime
6
7  class UDPChatClient:
8      def __init__(self, name, client_addr, client_port) -> None:
9          self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10         self.socket.bind(('127.0.0.1', 1235))
11         self.name = name
12         self.client_addr = client_addr
13         self.client_port = client_port
14
15     def send(self):
16         while True:
17             message = input('>> ')
18             if (message=='exit'):
19                 break
20             message = f'{self.name} [{datetime.datetime.now()}]: {message}'
21             self.socket.sendto(message.encode(), (self.client_addr,
   self.client_port))
22         self.socket.close()
23         sys.exit(0)
24
25     def recieve(self):
26         while True:
27             data, addr = self.socket.recvfrom(1024)
28             print(data.decode())
29             print("\n>> ", end="")
30
31
32
33 if __name__ == '__main__':
34     name = input("Enter your name: ")
35     server = UDPChatClient(name, "127.0.0.1", 1234)
36     x1 = threading.Thread(target=server.send)
37     x2 = threading.Thread(target=server.recieve)
38     x1.start()
39     x2.start()
```

Output

```
In D:\Books\sem 6\networks\labs\lab7 python server.py
Enter your name: gyannn
>> albbbb [2022-02-24 20:55:42.618484]: hello!!

>> hi, my name is gyan
>> albbbb [2022-02-24 20:56:13.814814]: this is a message from client to the server

>> oooh nice!
>> a message from server to the client application
>> this is computer networks assignment 7
>> albbbb [2022-02-24 20:56:57.850013]: 7th? dang niceee

>> this program is made in python
>> albbbb [2022-02-24 20:57:18.661656]: python? isn't that slow?

>> yeah
>> but gets the work done
>> albbbb [2022-02-24 20:57:43.922148]: yeah that is true

>> albbbb [2022-02-24 20:57:58.429588]: also, this program is multithreaded

>> yeahh
>> albbbb [2022-02-24 20:58:12.966457]: its getting late now,,

>> yeah, let's talk later
>> albbbb [2022-02-24 20:58:28.678258]: yup! bye

>> exit
```

Fig: Chat Server

```
In D:\Books\sem 6\networks\labs\lab7 python client.py
Enter your name: albbbb
>> hello!!
>> gyannn [2022-02-24 20:55:53.243695]: hi, my name is gyan

>> this is a message from client to the server
>> gyannn [2022-02-24 20:56:24.956935]: oooh nice!

>> gyannn [2022-02-24 20:56:33.648784]: a message from server to the client application

>> gyannn [2022-02-24 20:56:50.968440]: this is computer networks assignment 7

>> 7th? dang niceee
>> gyannn [2022-02-24 20:57:09.906309]: this program is made in python

>> python? isn't that slow?
>> gyannn [2022-02-24 20:57:23.071198]: yeah

>> gyannn [2022-02-24 20:57:27.123467]: but gets the work done

>> yeah that is true
>> also, this program is multithreaded
>> gyannn [2022-02-24 20:58:05.570661]: yeahh

>> its getting late now,,
>> gyannn [2022-02-24 20:58:22.594984]: yeah, let's talk later

>> yup! bye
>> exit
```

Fig: Chat Client

Computer Networks
Lab Assignment - 8
CSE 325

4 April 2022

Gyanendra Kumar Shukla
CSE 1
191112040

Department of Computer Science
NIT, Bhopal

Contents

1 Program to write and read two messages using pipe.

1.1 CODE

```cpp
// Program to write and read two messages using pipe.
#include <iostream>
#include <unistd.h>
#include <array>

using std::cout;
int main()
{
    std::array<int, 2> pipe_file_descriptors;
    int returnstatus;
    char writemessages[2][20] = {"Message", "To Earth"};
    char readmessage[20];
    returnstatus = pipe(pipe_file_descriptors.data());

    if (returnstatus == -1){
        cout << "Unable to create pipe\n";
        return 1;
    }

    cout<<"Writing to pipe - Message 1 is "<<  writemessages[0] << "\n";
    write(pipe_file_descriptors[1], writemessages[0], sizeof(writemessages[0]));

    read(pipe_file_descriptors[0], readmessage, sizeof(readmessage));
    cout<<"Reading from pipe - Message 1 is " <<readmessage << "\n";

    cout<<"Writing to pipe - Message 2 is " << writemessages[1] <<"\n";
    write(pipe_file_descriptors[1], writemessages[1], sizeof(writemessages[1]));

    read(pipe_file_descriptors[0], readmessage, sizeof(readmessage));
    cout<<"Reading from pipe - Message 2 is "<< readmessage <<"\n";
}
```

1.1.1 OUTPUT

```
Writing to pipe - Message 1 is Message
Reading from pipe - Message 1 is Message
Writing to pipe - Message 2 is To Earth
Reading from pipe - Message 2 is To Earth
```

2 Program to write and read two messages through the pipe using the parent and the child processes.

2.1 CODE

```
// Program to write and read two messages through the pipe using the parent and the
↪  child processes.
#include <iostream>
#include <unistd.h>
#include  <array>

using std::cout;

int main() {
   std::array<int,2> pipe_file_descpritor;
   int returnstatus;
   int pid;

   char writemessages[2][20]={"Message", "To the World"};
   char readmessage[20];

   returnstatus = pipe(pipe_file_descpritor.data());
   if (returnstatus == -1) {
      cout<<"Unable to create pipe\n";
      return 1;
   }
   pid = fork();

   // Child process
   if (pid == 0) {
      read(pipe_file_descpritor[0], readmessage, sizeof(readmessage));
      cout << "Child Process - Reading from pipe - Message 1 is " << readmessage <<
      ↪  "\n";
      read(pipe_file_descpritor[0], readmessage, sizeof(readmessage));
      cout << "Child Process - Reading from pipe - Message 2 is " << readmessage <<
      ↪  "\n";
   } else { //Parent process
      cout << "Parent Process - Writing to pipe - Message 1 is " << writemessages[0]
      ↪  << "\n";
      write(pipe_file_descpritor[1], writemessages[0], sizeof(writemessages[0]));
      cout << "Parent Process - Writing to pipe - Message 2 is " << writemessages[1]
      ↪  << "\n";
      write(pipe_file_descpritor[1], writemessages[1], sizeof(writemessages[1]));
   }
   return 0;
}
```

2.1.1 OUTPUT

```
Parent Process - Writing to pipe - Message 1 is Message
Parent Process - Writing to pipe - Message 2 is To the World
Child Process - Reading from pipe - Message 1 is Message
Child Process - Reading from pipe - Message 2 is To the World
```