

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 devices 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. It 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 bridges 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



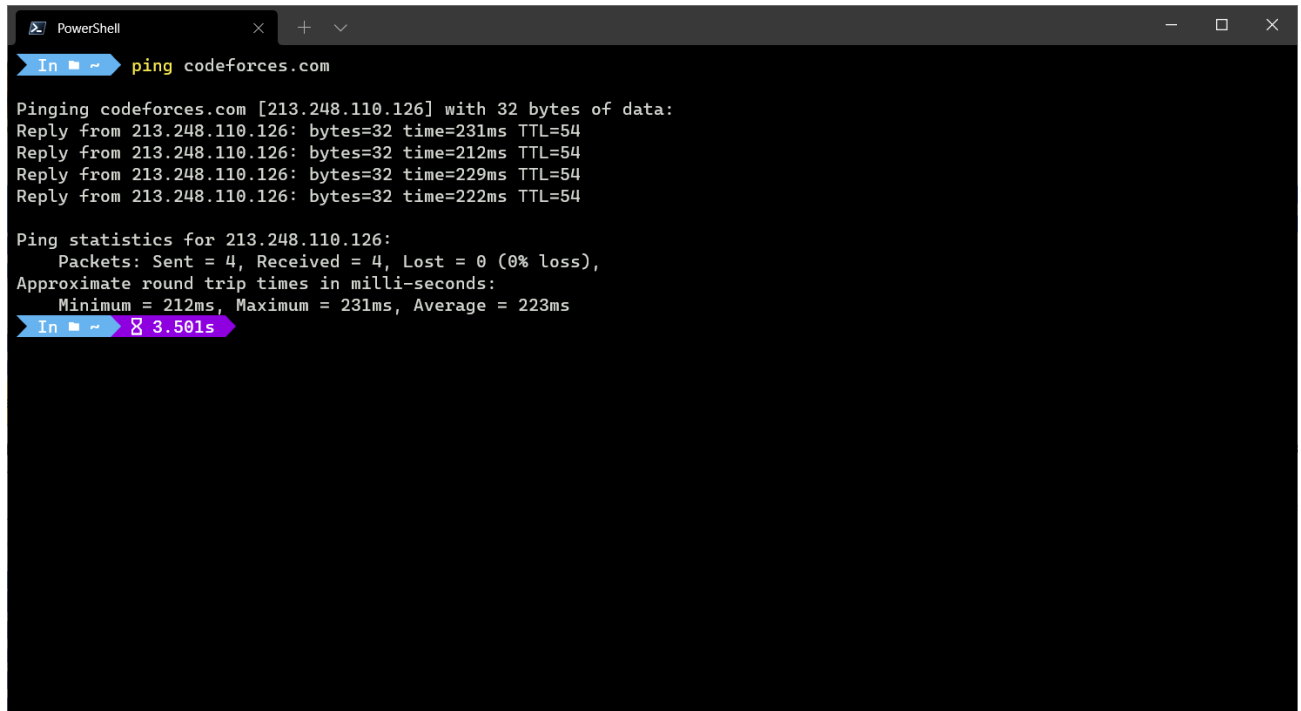
MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL – 462 003 (INDIA)
January 2022

Contents

1. Ping Command:	3
2. GETMAC:	4
3. IPCONFIG:	4
4. ARP	5
5. HOSTNAME:	5
6. NSLOOKUP:	6
7. NBSTAT:	6
8. NET:	7
9. NETSTAT:	7
10. NETSH:	8
11. TASKLIST:	8
12. TASKKILL:	9
13. TRACERT:	9
14. PATHPING:	10
15. SYSTEMINFO:	10

Basic commands in Computer Networking

1. Ping Command:



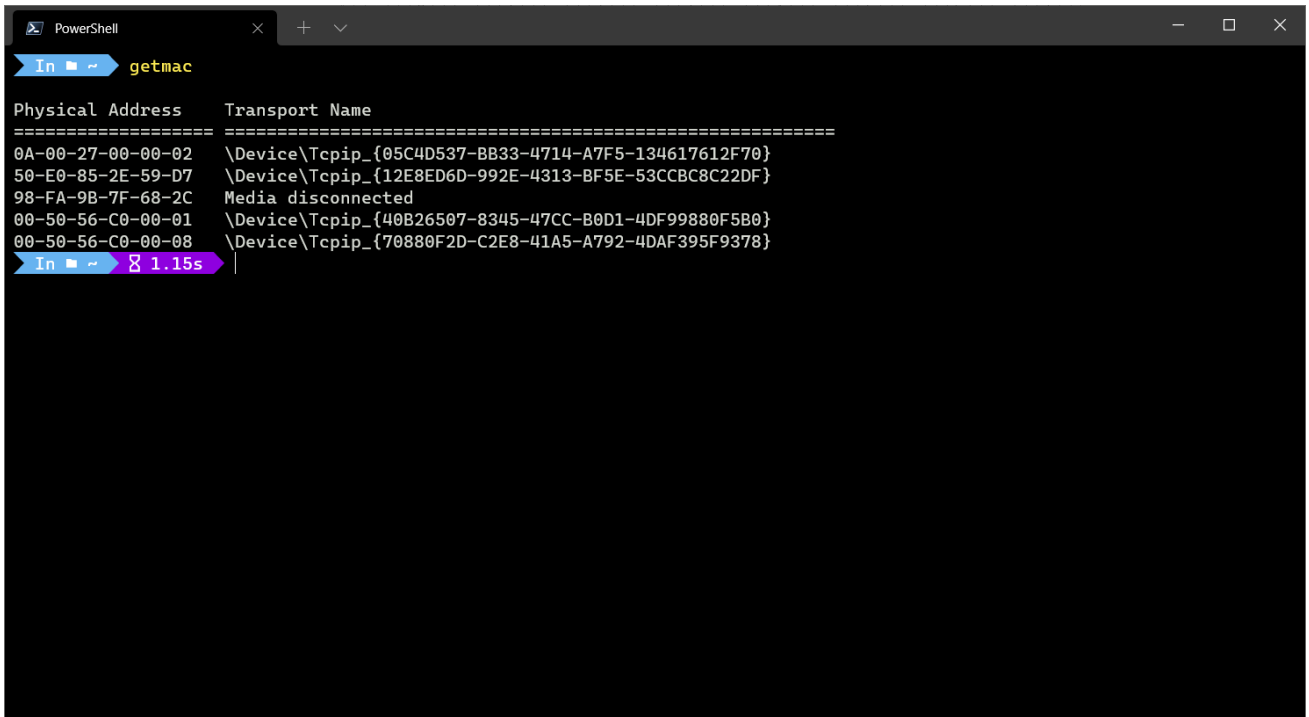
```
PowerShell
In ~ ~ ping codeforces.com

Pinging codeforces.com [213.248.110.126] with 32 bytes of data:
Reply from 213.248.110.126: bytes=32 time=231ms TTL=54
Reply from 213.248.110.126: bytes=32 time=212ms TTL=54
Reply from 213.248.110.126: bytes=32 time=229ms TTL=54
Reply from 213.248.110.126: bytes=32 time=222ms TTL=54

Ping statistics for 213.248.110.126:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 212ms, Maximum = 231ms, Average = 223ms
In ~ ~ 3.501s
```

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:



```
PowerShell
In ~ getmac

Physical Address      Transport Name
=====
0A-00-27-00-00-02     \Device\Tcpip_{05C4D537-BB33-4714-A7F5-134617612F70}
50-E0-85-2E-59-D7     \Device\Tcpip_{12E8ED6D-992E-4313-BF5E-53CCBC8C22DF}
98-FA-9B-7F-68-2C     Media disconnected
00-50-56-C0-00-01     \Device\Tcpip_{40B26507-8345-47CC-B0D1-4DF99880F5B0}
00-50-56-C0-00-08     \Device\Tcpip_{70880F2D-C2E8-41A5-A792-4DAF395F9378}
In ~ 1.15s
```

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

3. IPCONFIG:

```
PowerShell
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 4:

    Connection-specific DNS Suffix  . :
    Link-local IPv6 Address . . . . . : fe80::4c85:929e:7a5c:aa23%2
    IPv4 Address. . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :

Wireless LAN adapter Local Area Connection* 11:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 12:

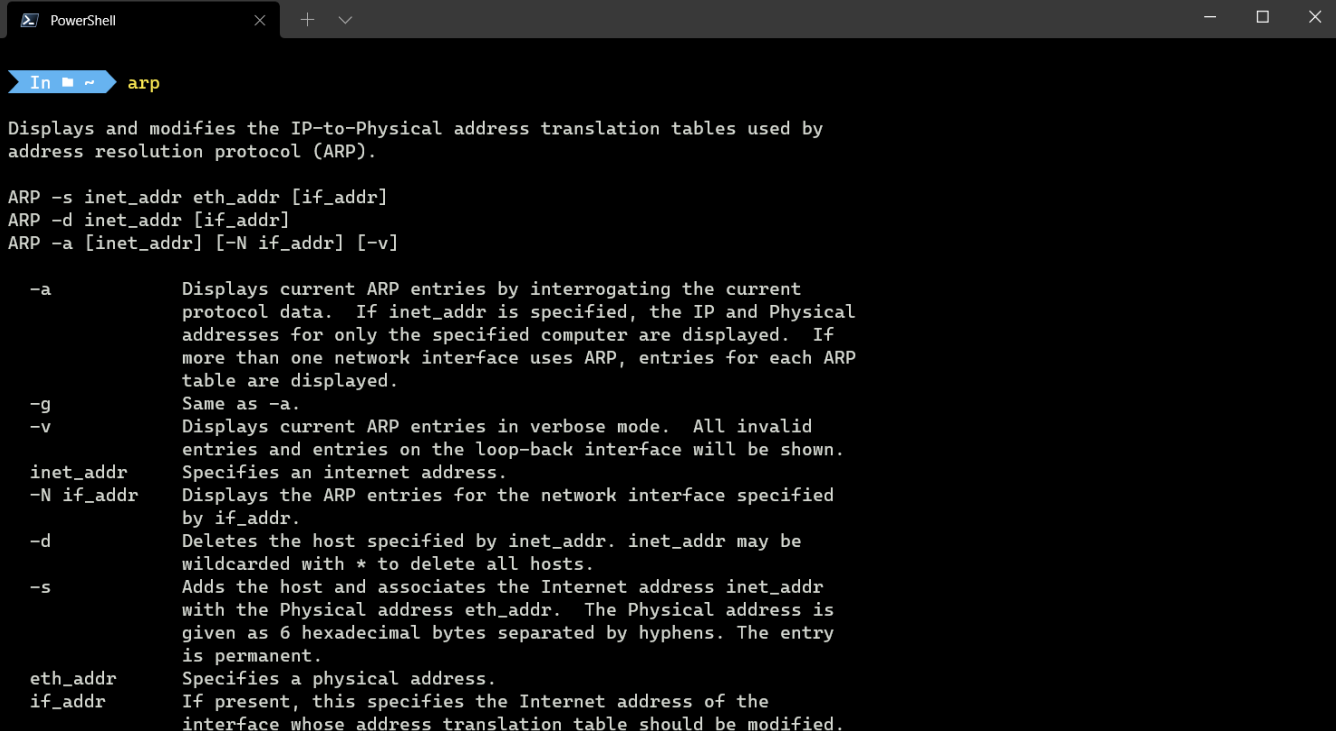
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter VMware Network Adapter VMnet1:
```

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



```
PowerShell
In ~> arp

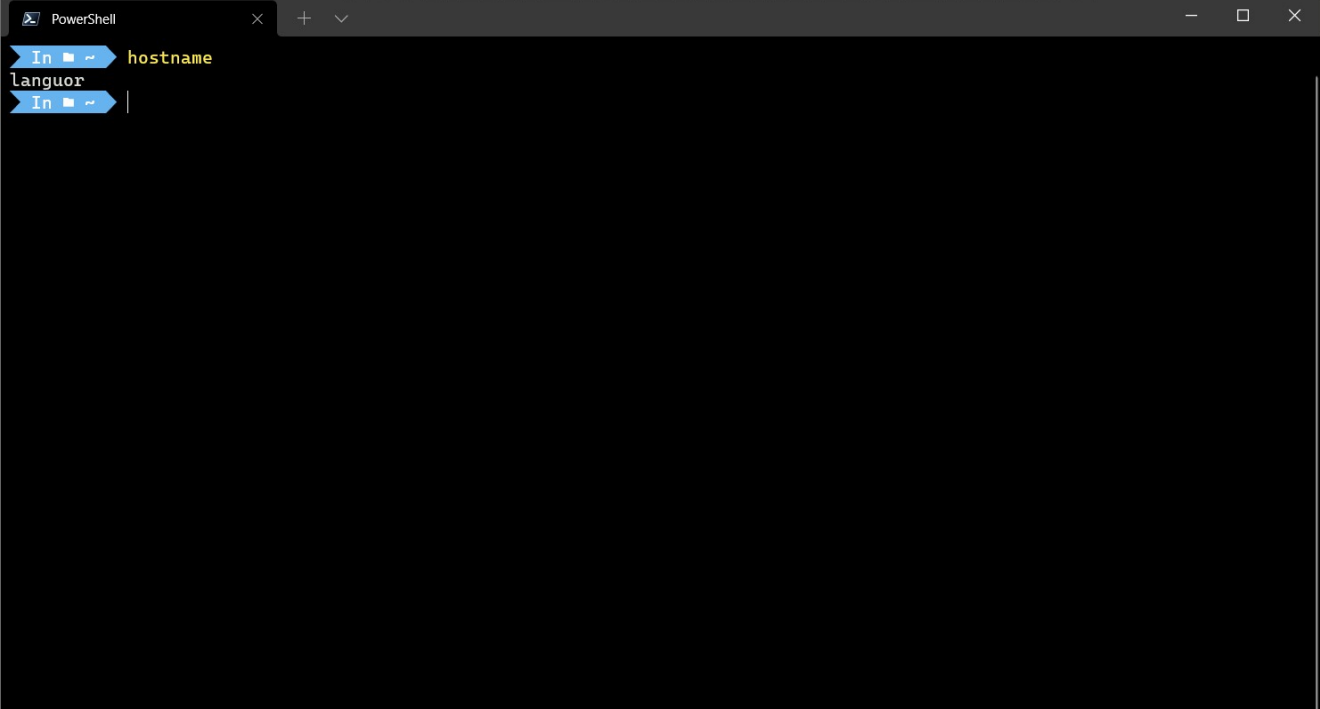
Displays and modifies the IP-to-Physical address translation tables used by
address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr] [-v]

-a          Displays current ARP entries by interrogating the current
            protocol data.  If inet_addr is specified, the IP and Physical
            addresses for only the specified computer are displayed.  If
            more than one network interface uses ARP, entries for each ARP
            table are displayed.
-g          Same as -a.
-v          Displays current ARP entries in verbose mode.  All invalid
            entries and entries on the loop-back interface will be shown.
inet_addr   Specifies an internet address.
-N if_addr  Displays the ARP entries for the network interface specified
            by if_addr.
-d          Deletes the host specified by inet_addr.  inet_addr may be
            wildcarded with * to delete all hosts.
-s          Adds the host and associates the Internet address inet_addr
            with the Physical address eth_addr.  The Physical address is
            given as 6 hexadecimal bytes separated by hyphens.  The entry
            is permanent.
eth_addr    Specifies a physical address.
if_addr     If present, this specifies the Internet address of the
            interface whose address translation table should be modified.
```

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:



```
PowerShell
In ~> hostname
languor
In ~> |
```

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

6. NSLOOKUP:

```
PowerShell
In ~ nslookup
Default Server: UnKnown
Address: 192.168.43.16

> youtube.com
Server: UnKnown
Address: 192.168.43.16

Non-authoritative answer:
Name: youtube.com
Addresses: 2404:6800:4007:818::200e
          142.250.77.174

> github.com
Server: UnKnown
Address: 192.168.43.16

Non-authoritative answer:
Name: github.com
Address: 13.234.210.38

>
In ~ 26.279s |
```

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:

```
PowerShell
In ~ nbtstat

Displays protocol statistics and current TCP/IP connections using NBT
(NetBIOS over TCP/IP).

NBTSTAT [ [-a RemoteName] [-A IP address] [-c] [-n]
          [-r] [-R] [-RR] [-s] [-S] [interval] ]

-a (adapter status) Lists the remote machine's name table given its name
-A (Adapter status) Lists the remote machine's name table given its
                    IP address.
-c (cache)          Lists NBT's cache of remote [machine] names and their IP addresses
-n (names)          Lists local NetBIOS names.
-r (resolved)       Lists names resolved by broadcast and via WINS
-R (Reload)         Purges and reloads the remote cache name table
-S (Sessions)       Lists sessions table with the destination IP addresses
-s (sessions)       Lists sessions table converting destination IP
                    addresses to computer NETBIOS names.
-RR (ReleaseRefresh) Sends Name Release packets to WINS and then, starts Refresh

RemoteName Remote host machine name.
IP address Dotted decimal representation of the IP address.
interval Redisplays selected statistics, pausing interval seconds
          between each display. Press Ctrl+C to stop redisplaying
          statistics.

In ~ |
```

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

8. NET:

```
PowerShell
In ~ net
The syntax of this command is:

NET
[ ACCOUNTS | COMPUTER | CONFIG | CONTINUE | FILE | GROUP | HELP |
  HELPMMSG | LOCALGROUP | PAUSE | SESSION | SHARE | START |
  STATISTICS | STOP | TIME | USE | USER | VIEW ]

In ~ net accounts
Force user logoff how long after time expires?: Never
Minimum password age (days): 0
Maximum password age (days): 42
Minimum password length: 0
Length of password history maintained: None
Lockout threshold: Never
Lockout duration (minutes): 30
Lockout observation window (minutes): 30
Computer role: WORKSTATION
The command completed successfully.

In ~ net user
User accounts for \\LANGUOR

-----
Administrator      DefaultAccount      Guest
kumar              WDAGUtilityAccount
The command completed successfully.

In ~ |
```

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

9. NETSTAT:

```
PowerShell
In ~ netstat

Active Connections

Proto Local Address          Foreign Address         State
TCP   127.0.0.1:1026          license:65001           ESTABLISHED
TCP   127.0.0.1:1309          license:24476           TIME_WAIT
TCP   127.0.0.1:1313          license:24476           ESTABLISHED
TCP   127.0.0.1:1314          license:24476           ESTABLISHED
TCP   127.0.0.1:10975         license:10976           ESTABLISHED
TCP   127.0.0.1:10976         license:10975           ESTABLISHED
TCP   127.0.0.1:10977         license:10978           ESTABLISHED
TCP   127.0.0.1:10978         license:10977           ESTABLISHED
TCP   127.0.0.1:21734         license:24476           ESTABLISHED
TCP   127.0.0.1:24476         license:1308            TIME_WAIT
TCP   127.0.0.1:24476         license:1313            ESTABLISHED
TCP   127.0.0.1:24476         license:1314            ESTABLISHED
TCP   127.0.0.1:24476         license:21734           ESTABLISHED
TCP   127.0.0.1:65001         license:1026            ESTABLISHED
TCP   192.168.43.113:1024      47:https               ESTABLISHED
TCP   192.168.43.113:1060     52.114.44.77:https      ESTABLISHED
TCP   192.168.43.113:1162     52.109.56.34:https      ESTABLISHED

|
```

It shows the useful network summary for your device.

10. NETSH:

```
PowerShell
In ~ netsh int ip show excludedportrange protocol=tcp

Protocol tcp Port Exclusion Ranges

Start Port      End Port
-----
5357           5357
9001           9001
50000          50059      *

* - Administered port exclusions.

In ~ 8 113ms netsh wlan show drivers

Interface name: Wi-Fi

Driver           : Intel(R) Wireless-AC 9560 160MHz
Vendor           : Intel Corporation
Provider         : Intel
Date             : 5/4/2020
Version          : 21.90.3.2
INF file         : oem52.inf
Type             : Native Wi-Fi Driver
Radio types supported : 802.11b 802.11g 802.11n 802.11a 802.11ac
FIPS 140-2 mode supported : Yes
802.11w Management Frame Protection supported : Yes
Hosted network supported : No
Authentication and cipher supported in infrastructure mode:
Open             None
Open             WEP-40bit
```

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

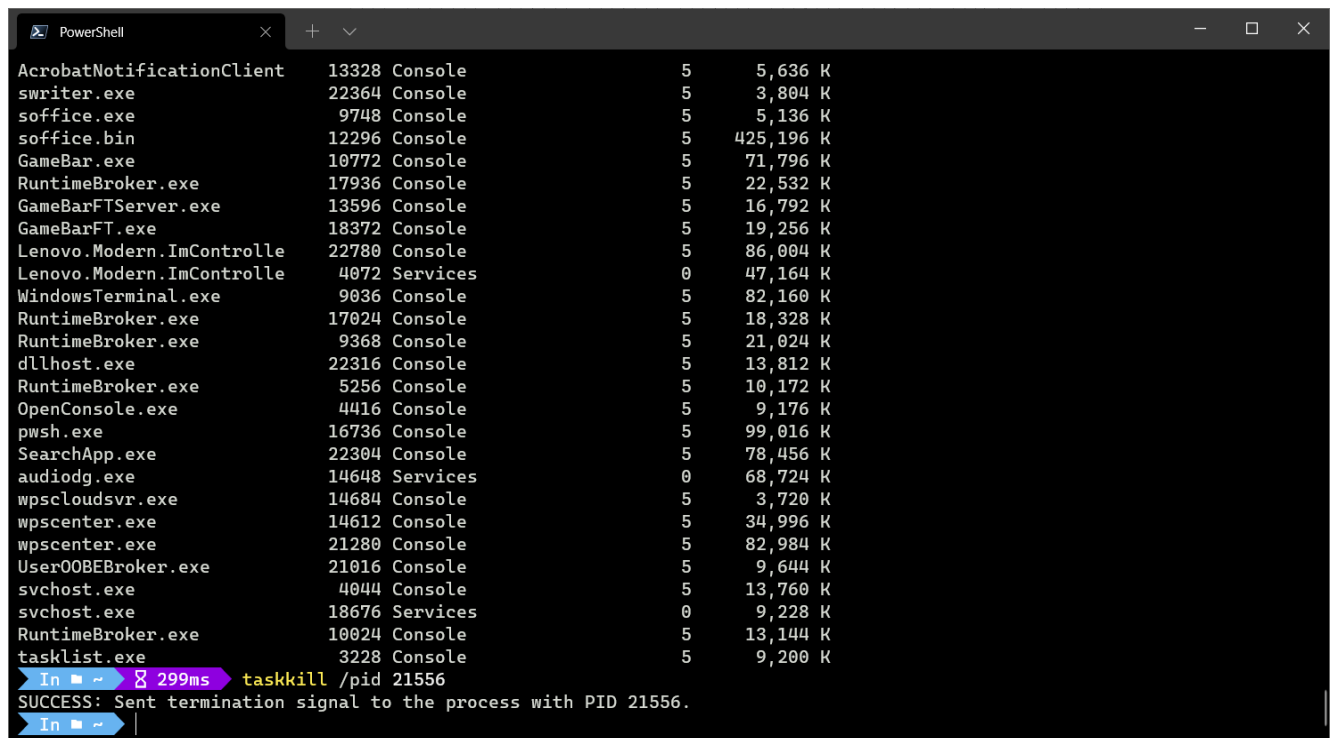
11. TASKLIST:

```
PowerShell
In ~ tasklist

Image Name          PID Session Name      Session#    Mem Usage
=====
System Idle Process    0 Services          0           8 K
System                4 Services          0          96 K
Secure System          72 Services          0       23,960 K
Registry              132 Services          0      98,856 K
smss.exe               488 Services          0         880 K
csrss.exe              772 Services          0       4,028 K
wininit.exe            872 Services          0       4,396 K
services.exe           944 Services          0       9,148 K
lsaiso.exe             964 Services          0       3,196 K
lsass.exe              972 Services          0      20,788 K
svchost.exe            592 Services          0      29,392 K
fontdrvhost.exe        92 Services          0       2,248 K
WUDFHost.exe          1048 Services          0      11,560 K
svchost.exe            1108 Services          0      19,572 K
svchost.exe            1156 Services          0       6,724 K
WUDFHost.exe           1216 Services          0       4,672 K
svchost.exe            1560 Services          0       3,604 K
svchost.exe            1616 Services          0       7,692 K
svchost.exe            1624 Services          0       6,772 K
svchost.exe            1676 Services          0      12,196 K
svchost.exe            1724 Services          0       8,380 K
svchost.exe            1732 Services          0       6,180 K
svchost.exe            1788 Services          0       4,312 K
svchost.exe            1840 Services          0       5,992 K
svchost.exe            1888 Services          0      17,576 K
svchost.exe            1960 Services          0       4,932 K
```

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

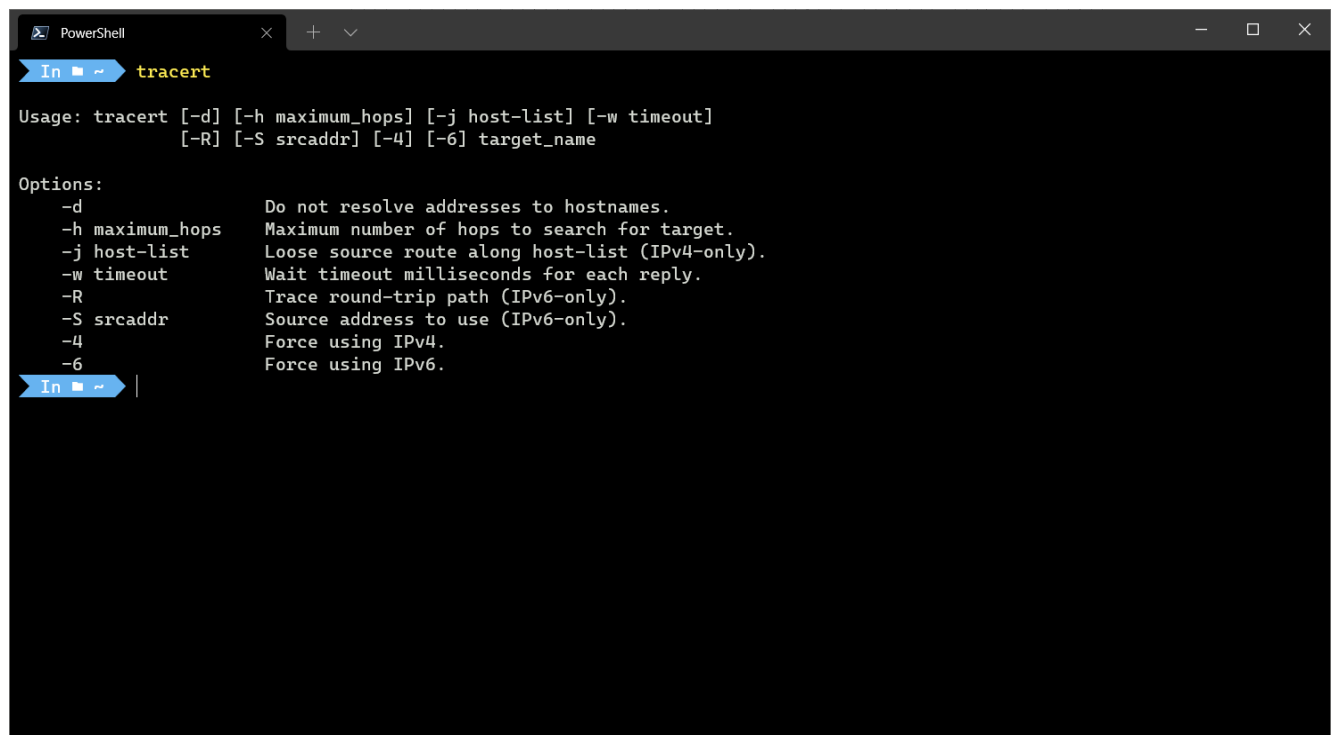
12. TASKKILL:



```
PowerShell
AcrobatNotificationClient 13328 Console 5 5,636 K
swriter.exe 22364 Console 5 3,804 K
soffice.exe 9748 Console 5 5,136 K
soffice.bin 12296 Console 5 425,196 K
GameBar.exe 10772 Console 5 71,796 K
RuntimeBroker.exe 17936 Console 5 22,532 K
GameBarFTServer.exe 13596 Console 5 16,792 K
GameBarFT.exe 18372 Console 5 19,256 K
Lenovo.Modern.ImControlle 22780 Console 5 86,004 K
Lenovo.Modern.ImControlle 4072 Services 0 47,164 K
WindowsTerminal.exe 9036 Console 5 82,160 K
RuntimeBroker.exe 17024 Console 5 18,328 K
RuntimeBroker.exe 9368 Console 5 21,024 K
dllhost.exe 22316 Console 5 13,812 K
RuntimeBroker.exe 5256 Console 5 10,172 K
OpenConsole.exe 4416 Console 5 9,176 K
pwsh.exe 16736 Console 5 99,016 K
SearchApp.exe 22304 Console 5 78,456 K
audiodg.exe 14648 Services 0 68,724 K
wpscloudsvr.exe 14684 Console 5 3,720 K
wpscenter.exe 14612 Console 5 34,996 K
wpscenter.exe 21280 Console 5 82,984 K
User00BEBroker.exe 21016 Console 5 9,644 K
svchost.exe 4044 Console 5 13,760 K
svchost.exe 18676 Services 0 9,228 K
RuntimeBroker.exe 10024 Console 5 13,144 K
tasklist.exe 3228 Console 5 9,200 K
In ~ 299ms taskkill /pid 21556
SUCCESS: Sent termination signal to the process with PID 21556.
In ~
```

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

13. TRACERT:



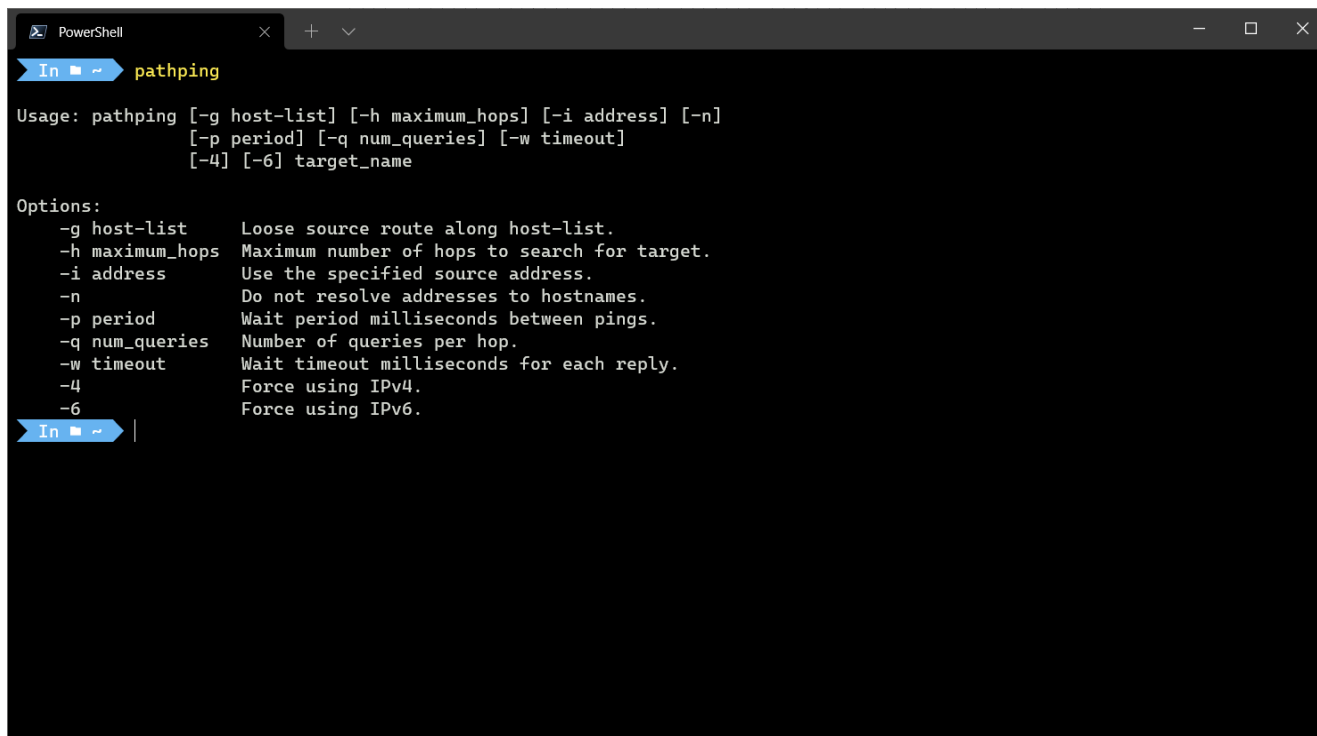
```
PowerShell
In ~ tracert

Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
           [-R] [-S srcaddr] [-4] [-6] target_name

Options:
  -d          Do not resolve addresses to hostnames.
  -h maximum_hops Maximum number of hops to search for target.
  -j host-list Loose source route along host-list (IPv4-only).
  -w timeout  Wait timeout milliseconds for each reply.
  -R          Trace round-trip path (IPv6-only).
  -S srcaddr  Source address to use (IPv6-only).
  -4          Force using IPv4.
  -6          Force using IPv6.
In ~
```

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:



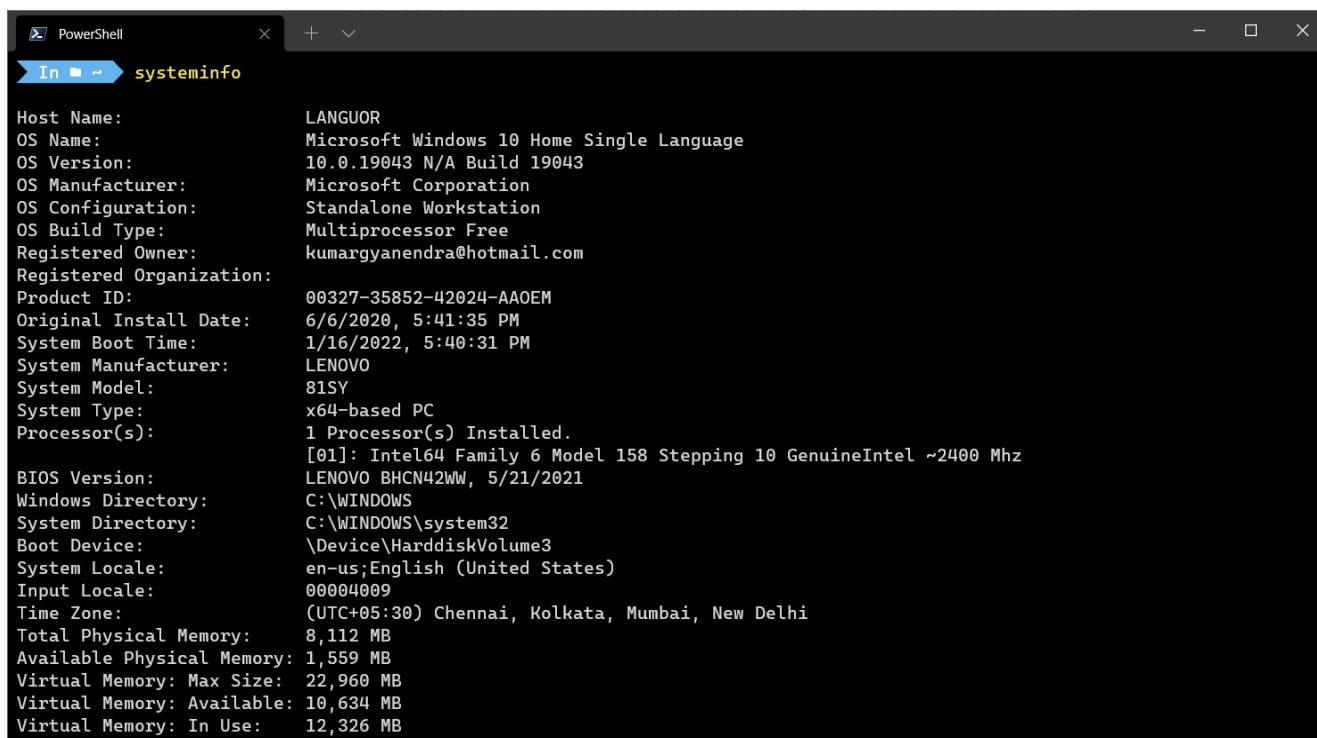
```
PowerShell
In ~ pathping

Usage: pathping [-g host-list] [-h maximum_hops] [-i address] [-n]
               [-p period] [-q num_queries] [-w timeout]
               [-4] [-6] target_name

Options:
  -g host-list      Loose source route along host-list.
  -h maximum_hops   Maximum number of hops to search for target.
  -i address        Use the specified source address.
  -n               Do not resolve addresses to hostnames.
  -p period         Wait period milliseconds between pings.
  -q num_queries    Number of queries per hop.
  -w timeout        Wait timeout milliseconds for each reply.
  -4               Force using IPv4.
  -6               Force using IPv6.
```

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

15. SYSTEMINFO:



```
PowerShell
In ~ systeminfo

Host Name:                LANGUOR
OS Name:                  Microsoft Windows 10 Home Single Language
OS Version:               10.0.19043 N/A Build 19043
OS Manufacturer:         Microsoft Corporation
OS Configuration:        Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:         kumargyanendra@hotmail.com
Registered Organization:
Product ID:                00327-35852-42024-AAOEM
Original Install Date:    6/6/2020, 5:41:35 PM
System Boot Time:         1/16/2022, 5:40:31 PM
System Manufacturer:      LENOVO
System Model:              81SY
System Type:               x64-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: Intel64 Family 6 Model 158 Stepping 10 GenuineIntel ~2400 Mhz
BIOS Version:              LENOVO BHCN42WW, 5/21/2021
Windows Directory:        C:\WINDOWS
System Directory:          C:\WINDOWS\system32
Boot Device:               \Device\HarddiskVolume3
System Locale:              en-us;English (United States)
Input Locale:              00004009
Time Zone:                 (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi
Total Physical Memory:     8,112 MB
Available Physical Memory: 1,559 MB
Virtual Memory: Max Size:  22,960 MB
Virtual Memory: Available: 10,634 MB
Virtual Memory: In Use:    12,326 MB
```

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

```
1  from datetime import datetime
2  import socket
3
4  class Server:
5      def __init__(self, port) -> None:
6          self.port = port
7          self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8          self.sock.bind(('', port))
9          self.sock.listen()
10         print('Server started on port {}'.format(port))
11
12     def run(self) -> None:
13         while True:
14             conn, addr = self.sock.accept()
15             print('Got connection from', addr)
16             with conn:
17                 while True:
18                     data = conn.recv(1024)
19                     if not data:
20                         break
21                     else:
22                         print(f'Got: {data.decode()}')
23                         cur_time = datetime.now()
24                         x = f'{cur_time} : {data.decode()}'
25                         conn.sendall(x.encode())
26
27
28     def __del__(self) -> None:
29         self.sock.close()
30         print('Server closed')
31
```



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

Client

```
1  import socket
2
3  class Client:
4      def __init__(self, host, port) -> None:
5          self.host = host
6          self.port = port
7          self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8          self.sock.connect((host, port))
9          print('Connected to {}:{}'.format(host, port))
10
11     def run(self) -> None:
12         while True:
13             data = input('> ')
14             if not data:
15                 break
16             self.sock.sendall(data.encode())
17             data = self.sock.recv(1024)
18             print(data.decode())
19
20     def __del__(self) -> None:
21         self.sock.close()
22         print('Connection closed')
23
24
25 if __name__ == '__main__':
26     client = Client('127.0.0.1', 12345)
27     client.run()
```

Output

```

Server started on port 12345
Got connection from ('127.0.0.1', 30369)
Got: Hello, server
Got: I'm sending a message!
Got: My name is Gyanendra!!!!!!!!!!!!!!
Got: 191112040
Got connection from ('127.0.0.1', 30385)
Got: m
Got: e
Got: s
Got: s
Got: a
Got: g
Got: e
Got:
Got: t
Got: h
Got: r
Got: o
Got: u
Got: g
Got: h
Got:
Got: t
Got: e
Got: l
Got: e
Got: n
Got: t
Got:

```

Fig: Messages received by the server

```

In D:\Books\sem 6\networks\labs\lab3 python .\client.py
Connected to 127.0.0.1:12345
> Hello, server
2022-01-30 21:02:37.689554 : Hello, server
> I'm sending a message!
2022-01-30 21:02:44.336316 : I'm sending a message!
> My name is Gyanendra!!!!!!!!!!!!!!
2022-01-30 21:03:06.564357 : My name is Gyanendra!!!!!!!!!!!!!!
> 191112040
2022-01-30 21:03:11.212423 : 191112040

```

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

```

Telnet localhost
2022-01-30 21:03:52.417424 : me2022-01-30 21:03:53.436225 : es2022-01-30 21:03:54.300817 : ss2022-01-30 21:03:54.548335
: sa2022-01-30 21:03:55.094188 : ag2022-01-30 21:03:55.402846 : ge2022-01-30 21:03:55.482547 : e 2022-01-30 21:03:56.49
2405 : t2022-01-30 21:03:57.214852 : th2022-01-30 21:03:57.423168 : hr2022-01-30 21:03:57.605966 : ro2022-01-30 21:03:5
7.694965 : ou2022-01-30 21:03:57.773127 : ug2022-01-30 21:03:57.978593 : gh2022-01-30 21:03:58.108031 : h 2022-01-30 21:
03:58.793041 : t2022-01-30 21:03:59.413735 : te2022-01-30 21:03:59.542941 : el2022-01-30 21:03:59.671223 : le2022-01-30
21:03:59.882320 : en2022-01-30 21:04:00.063999 : nt2022-01-30 21:04:00.214223 : t
2022-01-30 21:04:01.834790 :

```

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

```
1  import socket
2  import select
3
4  IP = "127.0.0.1"
5  PORT = 12345
6
7  class ChatServer:
8      HEADER_LENGTH = 10
9      def __init__(self, ip, port) -> None:
10         self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
11         self.server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
12         self.server_socket.bind((ip, port))
13         self.server_socket.listen()
14
15         self.socket_list = [self.server_socket]
16         self.clients = {}
17         print("Server started on {}:{}".format(ip, port))
18
19     def receive_message(self, client_socket):
20         try:
21             message_header = client_socket.recv(self.HEADER_LENGTH)
22
23             if not len(message_header):
24                 return False
25
26             message_length = int(message_header.decode("utf-8").strip())
27
```



```

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())

```

```

41         message =
self.client_socket.recv(message_length).decode("utf-8")
42
43         print(f"{username}> {message}")
44     except IOError as err:
45         if err.errno != errno.EAGAIN and err.errno !=
errno.EWOULDBLOCK:
46             print(f"Reading error: {str(err)}")
47             sys.exit()
48
49         continue
50
51     except Exception as e:
52         print(f"Reading error: {str(e)}")
53         sys.exit()
54
55 if __name__ == "__main__":
56     chat_client = ChatClient(IP, PORT, str(sys.argv[1]))
57     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

```

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

```

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

```

1  import socket
2  import os
3
4  IP = "127.0.0.1"
5  PORT = 12345
6
7  class FileServer:
8      SEPARATOR = "<SEPARATOR>"
9      BUFFER_SIZE = 4096
10
11     def __init__(self, ip, port) -> None:
12         self.sock = socket.socket()
13         self.sock.bind((ip, port))
14         self.sock.listen()
15         print(f"Listening on {ip}:{port}")
16
17     def receive(self):
18         client_socket, address = self.sock.accept()
19         received = client_socket.recv(self.BUFFER_SIZE).decode()
20         filename, filesize = received.split(self.SEPARATOR)
21
22         filename = "recv-" + os.path.basename(filename)
23         filesize = int(filesize)
24
25         with open(filename, "wb") as f:
26             print(f"Incoming file, saving as {filename}")
27             while True:
28                 bytes_read = client_socket.recv(self.BUFFER_SIZE)
29                 if not bytes_read:
30                     # we've completed receiving files
31                     break

```



```

32         f.write(bytes_read)
33         print(f"Done receiving {filename}")
34
35         client_socket.close()
36         self.sock.close()
37
38
39 if __name__ == "__main__":
40     server = FileServer(IP, PORT)
41     server.receive()
42

```

File Transfer Client

```

1  import socket
2  import sys
3  import os
4
5  class FileClient:
6      SEPARATOR = "<SEPARATOR>"
7      BUFFER_SIZE = 4096
8
9      def __init__(self, ip, port) -> None:
10         self.sock = socket.socket()
11         print(f"Connecting to {ip}:{port}")
12         self.sock.connect((ip, port))
13         print(f"Connected to {ip}:{port}")
14
15     def send(self, filename):
16         if not os.path.isfile(filename):
17             print(f"{filename} does not exist!")
18             return
19         filesize = os.path.getsize(filename)
20         self.sock.send(f"{filename}{self.SEPARATOR}{filesize}".encode())
21
22         with open(filename, "rb") as f:
23             print(f"Sending {filename}")
24             while True:
25                 bytes_read = f.read(self.BUFFER_SIZE)
26                 if not bytes_read:
27                     # we've completed sending files
28                     break
29                 self.sock.sendall(bytes_read)
30             print(f"Done sending {filename}")
31
32         self.sock.close()
33
34
35 if __name__ == "__main__":
36     client = FileClient("127.0.0.1", 12345)
37     filename = sys.argv[1]
38     client.send(filename)

```

Output

```
Loading personal and system profiles took 723ms.  
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

```
Done sending .\ChatClient.py  
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'))

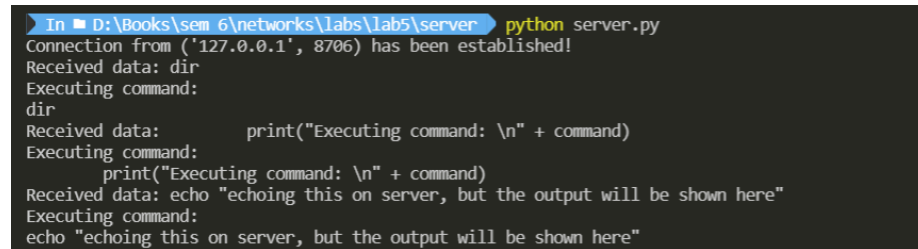
```

```

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

```

Output

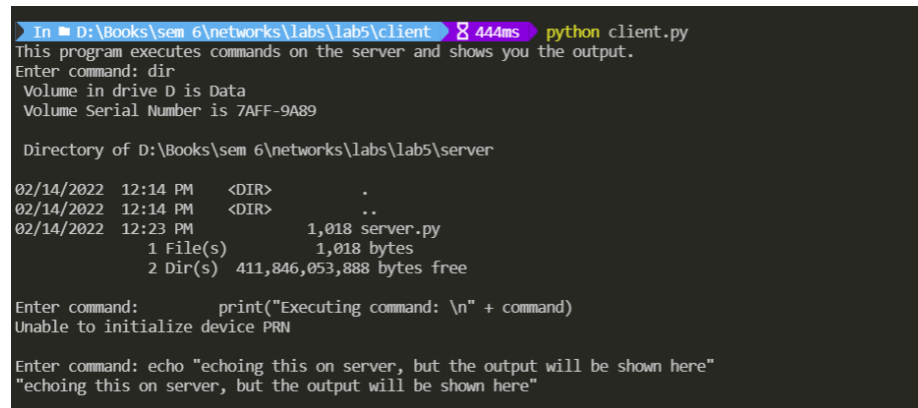


```

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 8 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 receive 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

```
1  import os
2  import socket
3  import sys
4  import threading
5  import datetime
6
7  class UDPChatServer:
8      def __init__(self, name, client_addr=None, client_port=None) -> None:
9          self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10         self.socket.bind(('127.0.0.1', 1234))
11         self.name = name
12         self.client_addr = "127.0.0.1"
13         self.client_port = 1235
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 = 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.	2
1.1	CODE	2
1.1.1	OUTPUT	3
2	Program to write and read two messages through the pipe using the parent and the child processes.	4
2.1	CODE	4
2.1.1	OUTPUT	5

1 Program to write and read two messages using pipe.

1.1 CODE

```
// 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
