

COLLEGE OF ENGINEERING, TRIVANDRUM

LAB REPORT

---

## Network Programming Lab

---

*Author:*  
Alan Anto

*Registration Number :*  
TVE16CS09

May 16, 2019

## Contents

<b>1</b>	<b>Basics of Network configurations files and Network- ing Commands</b>	<b>8</b>
1.1	AIM . . . . .	8
1.2	Theory . . . . .	8
1.2.1	ifconfig . . . . .	8
1.2.2	ifconfig -a . . . . .	9
1.2.3	ping . . . . .	9
1.2.4	ping -c . . . . .	9
1.2.5	tracert . . . . .	9
1.2.6	netstat . . . . .	10
1.2.7	netstat -r . . . . .	10
1.2.8	nslookup . . . . .	10
1.2.9	route . . . . .	11
1.2.10	dig . . . . .	11
1.2.11	host . . . . .	11
1.2.12	hostname . . . . .	12
1.2.13	ethtool . . . . .	12
1.3	Result . . . . .	12
<b>2</b>	<b>System Calls</b>	<b>13</b>
2.1	AIM . . . . .	13
2.2	Theory . . . . .	13
2.3	Process Control . . . . .	13
2.3.1	Create Process . . . . .	13
2.3.2	Exit Process . . . . .	14
2.4	Wait . . . . .	14
2.5	File Management . . . . .	15
2.5.1	Create File . . . . .	15
2.5.2	Write File . . . . .	15
2.5.3	Close Handle . . . . .	15
2.6	Communication . . . . .	16
2.6.1	Create Pipe . . . . .	16
2.6.2	Create File Mapping . . . . .	16
2.7	Information Maintenance . . . . .	17
2.7.1	Current Process ID . . . . .	17
2.7.2	SET TIMER . . . . .	17
2.7.3	Sleep . . . . .	17
2.8	Result . . . . .	17
<b>3</b>	<b>Process and Thread</b>	<b>18</b>

3.1	AIM . . . . .	18
3.2	Theory . . . . .	18
3.3	Algorithm . . . . .	19
3.4	Program . . . . .	19
3.5	Output . . . . .	20
3.6	Result . . . . .	20
<b>4</b>	<b>First Readers-Writers Problem</b>	<b>21</b>
4.1	AIM . . . . .	21
4.2	Theory . . . . .	21
	4.2.1 Introduction . . . . .	21
	4.2.2 Solution . . . . .	21
4.3	Algorithm . . . . .	22
4.4	Program . . . . .	22
4.5	Output . . . . .	24
4.6	Result . . . . .	24
<b>5</b>	<b>Network Simulator NS2</b>	<b>25</b>
5.1	AIM . . . . .	25
5.2	Theory . . . . .	25
	5.2.1 Installation . . . . .	25
	5.2.2 TCL . . . . .	26
	5.2.3 C++ . . . . .	26
5.3	Program . . . . .	26
	5.3.1 Wired Transmission . . . . .	26
	5.3.2 Wireless . . . . .	28
5.4	Result . . . . .	32
<b>6</b>	<b>Second Readers - Writers Problem</b>	<b>33</b>
6.1	AIM . . . . .	33
6.2	Theory . . . . .	33
6.3	Algorithm . . . . .	33
6.4	Program . . . . .	34
6.5	Output . . . . .	36
6.6	Result . . . . .	36
<b>7</b>	<b>Pipes,Message Queue and Shared Memory</b>	<b>37</b>
7.1	AIM . . . . .	37
7.2	Theory . . . . .	37
	7.2.1 Pipes . . . . .	37
	7.2.2 Ordinary Pipes . . . . .	37

7.2.3	Named Pipes . . . . .	37
7.2.4	Message Queues . . . . .	38
7.2.5	Shared Memory . . . . .	38
7.3	Program . . . . .	38
7.3.1	Pipes . . . . .	38
7.3.2	Message Queue . . . . .	39
7.3.3	Shared memory . . . . .	40
7.4	Output . . . . .	41
7.4.1	Pipes . . . . .	41
7.4.2	Message Queue . . . . .	41
7.4.3	Shared Memory . . . . .	41
7.5	Result . . . . .	41
<b>8</b>	<b>Client-Server Communication using TCP</b>	<b>42</b>
8.1	AIM . . . . .	42
8.2	Theory . . . . .	42
8.2.1	TCP . . . . .	42
8.2.2	Socket Programming . . . . .	42
8.3	Program . . . . .	44
8.3.1	Server Side . . . . .	44
8.3.2	Client Side . . . . .	45
8.4	Output . . . . .	45
8.4.1	Server . . . . .	45
8.4.2	Client . . . . .	46
8.5	Result . . . . .	46
<b>9</b>	<b>Client-Server Communication using UDP</b>	<b>47</b>
9.1	AIM . . . . .	47
9.2	Theory . . . . .	47
9.2.1	UDP Protocol . . . . .	47
9.2.2	Socket Programming . . . . .	49
9.3	Program . . . . .	50
9.3.1	Server Side . . . . .	50
9.3.2	Client Side . . . . .	51
9.4	Output . . . . .	52
9.4.1	Server . . . . .	52
9.4.2	Client . . . . .	52
9.5	Result . . . . .	52
<b>10</b>	<b>Multiuser Chat server</b>	<b>53</b>
10.1	AIM . . . . .	53

10.2	Theory . . . . .	53
10.2.1	TCP Protocol . . . . .	53
10.2.2	Client . . . . .	53
10.2.3	Server . . . . .	54
10.2.4	Socket . . . . .	54
10.2.5	Multi Threading . . . . .	54
10.2.6	Server Side Script . . . . .	54
10.2.7	Client Side Script . . . . .	54
10.3	Algorithm . . . . .	55
10.4	Server . . . . .	55
10.5	Client . . . . .	55
10.6	Program . . . . .	55
10.6.1	Server Side . . . . .	55
10.6.2	Client Side . . . . .	57
10.7	Output . . . . .	58
10.8	Result . . . . .	58
<b>11</b>	<b>Concurrent Time Server - UDP</b>	<b>60</b>
11.1	AIM . . . . .	60
11.2	Theory . . . . .	60
11.2.1	UDP Protocol . . . . .	60
11.2.2	Timer Server . . . . .	60
11.3	Client . . . . .	60
11.4	Server . . . . .	61
11.5	Socket . . . . .	61
11.6	Algorithm . . . . .	61
11.6.1	Server . . . . .	61
11.6.2	Client . . . . .	61
11.7	Program . . . . .	62
11.7.1	Server Side . . . . .	62
11.7.2	Client Side . . . . .	62
11.8	Output . . . . .	63
11.8.1	Server . . . . .	63
11.8.2	Client . . . . .	63
11.9	Result . . . . .	63
<b>12</b>	<b>Distance Vector Routing Protocol</b>	<b>64</b>
12.1	AIM . . . . .	64
12.2	Theory . . . . .	64
12.2.1	Distance Vector Routing Protocol . . . . .	64
12.3	Algorithm . . . . .	65

12.4	Program . . . . .	65
12.5	Output . . . . .	67
12.6	Result . . . . .	67
<b>13</b>	<b>Link state routing protocol</b>	<b>68</b>
13.1	AIM . . . . .	68
13.2	Theory . . . . .	68
13.2.1	Link state routing protocol . . . . .	68
13.3	Program . . . . .	68
13.4	Output . . . . .	71
13.5	Result . . . . .	71
<b>14</b>	<b>UDP in wireshark</b>	<b>72</b>
14.1	Aim . . . . .	72
14.2	Theory . . . . .	72
14.2.1	Wireshark . . . . .	72
14.2.2	Getting Wireshark . . . . .	72
14.2.3	UDP Protocol . . . . .	72
14.2.4	Capturing packets . . . . .	73
14.2.5	Filtering Packets . . . . .	73
14.3	Output . . . . .	74
14.4	Result . . . . .	75
<b>15</b>	<b>Three way hand shake connection termination</b>	<b>76</b>
15.1	Aim . . . . .	76
15.2	Theory . . . . .	76
15.2.1	Wireshark . . . . .	76
15.2.2	Getting Wireshark . . . . .	76
15.2.3	TCP Protocol . . . . .	76
15.2.4	Threeway handshake . . . . .	77
15.2.5	Capturing packets . . . . .	78
15.2.6	Filtering Packets . . . . .	78
15.3	Output . . . . .	79
15.4	Result . . . . .	79
<b>16</b>	<b>Packet capturing and filtering application</b>	<b>80</b>
16.1	Aim . . . . .	80
16.2	Theory . . . . .	80
16.2.1	Packet Capturing . . . . .	80
16.2.2	Algorithm . . . . .	80
16.2.3	Program . . . . .	81

16.3	Output . . . . .	82
16.4	Result . . . . .	83
<b>17</b>	<b>Simple Mail Transfer Protocol</b>	<b>84</b>
17.1	Aim . . . . .	84
17.2	Theory . . . . .	84
17.2.1	SMTP . . . . .	84
17.2.2	Working . . . . .	85
17.3	Program . . . . .	85
17.3.1	SMTP Server . . . . .	85
17.3.2	SMTP Client . . . . .	87
17.4	Output . . . . .	87
17.4.1	SMTP Server . . . . .	87
17.4.2	SMTP Client . . . . .	88
17.5	Result . . . . .	88
<b>18</b>	<b>Concurrent File Server</b>	<b>89</b>
18.1	Aim . . . . .	89
18.2	Theory . . . . .	89
18.2.1	FTP . . . . .	89
18.3	Program . . . . .	89
18.3.1	File Server . . . . .	89
18.3.2	File Client . . . . .	90
18.4	Output . . . . .	91
18.4.1	File Server . . . . .	91
18.4.2	File Client . . . . .	91
18.5	Result . . . . .	92
<b>19</b>	<b>Network with multiple subnets with wired and wireless LANs</b>	<b>93</b>
19.1	Aim . . . . .	93
19.2	Theory . . . . .	93
19.2.1	Network . . . . .	93
19.2.2	Subnet . . . . .	93
19.3	Configuring the Services . . . . .	94
19.3.1	Telnet . . . . .	94
19.3.2	SSH . . . . .	95
19.3.3	FTP Server . . . . .	95
19.3.4	Web Server . . . . .	96
19.3.5	File Server . . . . .	96
19.3.6	DHCP Server . . . . .	97
19.3.7	DNS Server . . . . .	98

19.4 Result . . . . . 99



# 1 Basics of Network configurations files and Networking Commands

## 1.1 AIM

Familiarising with Basics of Network configurations files and Networking Commands in Linux.

## 1.2 Theory

### 1.2.1 ifconfig

ifconfig stands for "interface configuration". It can be used for viewing and changing configurations of network interfaces on your system.

```

Alans-MacBook-Air:~ alan$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
    nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
XHC20: flags=0<> mtu 0
en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
    options=60<TS04,TS06>
    ether 9a:00:16:ee:b8:70
    media: autoselect <full-duplex>
    status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    ether 14:c2:13:01:b3:12
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect (<unknown type>)
    status: inactive
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=63<RXCSUM,TXCSUM,TS04,TS06>
    ether 9a:00:16:ee:b8:70
    Configuration:
        id 0:0:0:0:0:0 priority 0 hellotime 0 fwddelay 0
        maxage 0 holdcnt 0 proto stp maxaddr 100 timeout 1200
        root id 0:0:0:0:0:0 priority 0 ifcost 0 port 0
        ipfilter disabled flags 0x2
    member: en1 flags=3<LEARNING,DISCOVER>
        ifmaxaddr 0 port 5 priority 0 path cost 0
    nd6 options=201<PERFORMNUD,DAD>
    media: <unknown type>
    status: inactive
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
    ether 06:c2:13:01:b3:12
    media: autoselect
    status: inactive
awdl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1484
    ether 4e:11:ea:5c:85:96
    inet6 fe80::4c11:ea5c:8596%awdl0 prefixlen 64 scopeid 0x9
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect
    status: active
utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 2000
    inet6 fe80::902f:ed42:b855:8e2%utun0 prefixlen 64 scopeid 0xa
    nd6 options=201<PERFORMNUD,DAD>

```

### 1.2.2 ifconfig -a

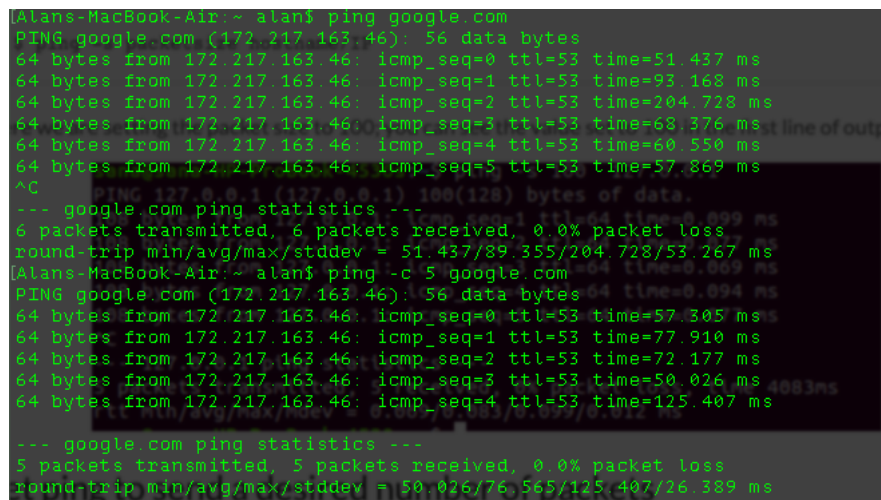
Used to view all network interfaces on the system.

### 1.2.3 ping

Ping stands for Packet Internet Groper . Its used to check the connectivity status between two, a source and a destination device . It uses ICMP(Internet Control Message Protocol) to sent and recieve between the source and destination systems.

### 1.2.4 ping -c

Used to specify the number of packets to be sent before exiting.

A terminal window showing the execution of the 'ping' command. The first command is 'ping google.com', which sends 6 packets. The output shows the IP address 172.217.163.46 and various statistics like icmp\_seq, ttl, and time. The second command is 'ping -c 5 google.com', which sends 5 packets. The output shows the same IP address and statistics. The terminal background is dark with green text.

```
[Alans-MacBook-Air:~ alan]$ ping google.com
PING google.com (172.217.163.46): 56 data bytes
64 bytes from 172.217.163.46: icmp_seq=0 ttl=53 time=51.437 ms
64 bytes from 172.217.163.46: icmp_seq=1 ttl=53 time=93.168 ms
64 bytes from 172.217.163.46: icmp_seq=2 ttl=53 time=204.728 ms
64 bytes from 172.217.163.46: icmp_seq=3 ttl=53 time=68.376 ms
64 bytes from 172.217.163.46: icmp_seq=4 ttl=53 time=60.550 ms
64 bytes from 172.217.163.46: icmp_seq=5 ttl=53 time=57.869 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 51.437/89.355/204.728/53.267 ms
[Alans-MacBook-Air:~ alan]$ ping -c 5 google.com
PING google.com (172.217.163.46): 56 data bytes
64 bytes from 172.217.163.46: icmp_seq=0 ttl=53 time=57.305 ms
64 bytes from 172.217.163.46: icmp_seq=1 ttl=53 time=77.910 ms
64 bytes from 172.217.163.46: icmp_seq=2 ttl=53 time=72.177 ms
64 bytes from 172.217.163.46: icmp_seq=3 ttl=53 time=50.026 ms
64 bytes from 172.217.163.46: icmp_seq=4 ttl=53 time=125.407 ms
--- google.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 50.026/76.565/125.407/26.389 ms
```

### 1.2.5 traceroute

It is used to track the route packets take to reach the destination or host.

```

Alans-MacBook-Air:~ alan$ traceroute google.com
traceroute to google.com (216.58.196.174): 64 hops max, 52 byte packets
 1  192.168.43.1 (192.168.43.1)  4.213 ms  2.670 ms  3.446 ms
 2  * * *
 3  10.72.51.19 (10.72.51.19)  40.389 ms  39.259 ms  50.105 ms
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  74.125.48.26 (74.125.48.26)  54.119 ms  49.641 ms  60.288 ms
 9  108.170.253.113 (108.170.253.113)  48.885 ms * 49.732 ms
10  216.239.43.239 (216.239.43.239)  49.794 ms
    74.125.253.16 (74.125.253.16)  121.242 ms
    216.239.43.239 (216.239.43.239)  45.904 ms
11  216.239.43.235 (216.239.43.235)  48.651 ms
    74.125.242.146 (74.125.242.146)  39.519 ms
    74.125.242.130 (74.125.242.130)  55.292 ms
12  maa03s31-in-f14.1e100.net (216.58.196.174)  50.349 ms  119.282 ms  45.628 ms

```

### 1.2.6 netstat

It is used to find network connections, routing tables etc.

### 1.2.7 netstat -r

It can be used for displaying routing table details.

```

Alans-MacBook-Air:~ alan$ netstat -r
Routing tables

Internet:
Destination Gateway Flags Refs Use Netif Expire
default 192.168.43.1 UGSc 53 105 en0
127 localhost UCS 0 0 lo0
localhost localhost graphics [UHLw.5](traceroute) 26284 lo0
169.254 link#6 UCS 0 0 en0 !
192.168.43 link#6 UCS 0 0 en0 !
192.168.43.1/32 link#6 UCS 1 0 en0 !
192.168.43.1 20:a6:c:b5:1b:97 UHLWIir 13 43 en0 1144
192.168.43.39/32 link#6 UCS 0 0 en0 !
224.0.0/4 link#6 used to find network connections, routing tables etc. 0 en0 !
224.0.0.251/1 1:0:5e:0:0:fb UHmLWI 0 0 en0
239.255.255.250 1:0:5e:7f:ff:fa UHmLWI 0 60 en0
255.255.255.255/32 link#6 UCS 0 0 en0 !

```

### 1.2.8 nslookup

It is used to find the information about an ip address or a domain. It will translate a domain name to IP address and vice-versa.

```

Alans-MacBook-Air:~ alan$ nslookup google.com
Server:                2405:204:d20b:85d2::91
Address:               2405:204:d20b:85d2::91#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.163.174

```

### 1.2.9 route

It is used to for manipulating the routing table.

### 1.2.10 dig

It stands for domain Information Groper . It is used to query DNS Name servers .

```

Alans-MacBook-Air:~ alan$ dig google.com
;; <<>> DiG 9.10.6<<>> google.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 63604
;; flags: qr rd ra, QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; OPT PSEUDOSECTION: redhat.com
;; EDNS: version: 0, flags: udp: 1280
;; QUESTION SECTION:
;google.com.      IN      A
;; ANSWER SECTION:
google.com.      64      IN      A      172.217.26.174
;; Query time: 130 msec
;; SERVER: 2405:204:d20b:85d2::91#53(2405:204:d20b:85d2::91)
;; WHEN: Mon Feb 04 17:33:17 IST 2019
;; MSG SIZE rcvd: 55

```

### 1.2.11 host

Its used for performing DNS Lookups. It can be used to change names to IP addresses and vice-versa.

```

Alans-MacBook-Air:~ alan$ host 216.58.197.78
216.58.197.78.in-addr.arpa domain name pointer maa03s21-
216.58.197.78.in-addr.arpa domain name pointer maa03s21-

```

### 1.2.12 hostname

It is used to find the hostname and domain name of the system.

```
Alans-MacBook-Air:~ alan$ hostname  
Alans-MacBook-Air.local
```

### 1.2.13 ethtool

It is used to manipulate Network Interface Card's settings. We can set the speed, port etc using this tool.

## 1.3 Result

The familiarization of basic networking commands used in Linux was completed successfully.

## **2 System Calls**

### **2.1 AIM**

Familiarizing and understanding the use and functioning of System Calls used for Operating system and network programming in Linux

### **2.2 Theory**

System calls are used by the programs to interact with the Operating System. Application Programme Interfaces (APIs) are used by the user programmes to access these system calls.

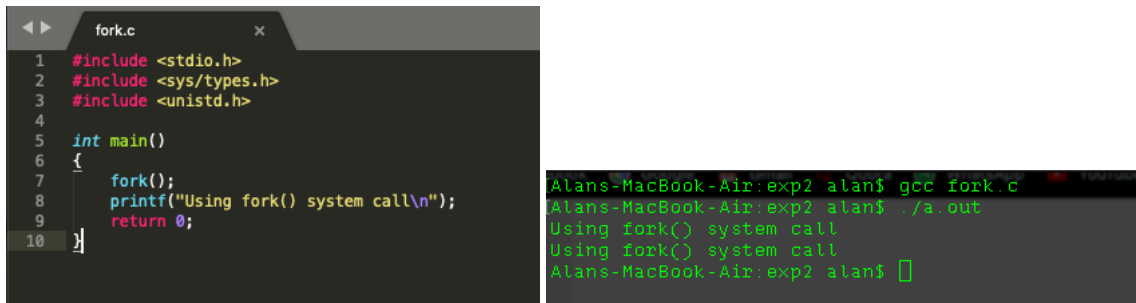
Services Provided by System Calls :

- 1.Process creation and management
- 2.Main memory management
- 3.File Access, Directory and File system management
- 4.Device handling(I/O)
- 5.Protection
- 6.Networking, etc.

### **2.3 Process Control**

#### **2.3.1 Create Process**

fork() is used to create a process . A child process, which is a copy of the parent process is created every time fork() is called.The fork() call returns the PID of the child process created if it is called successfully, else it returns a negative number.

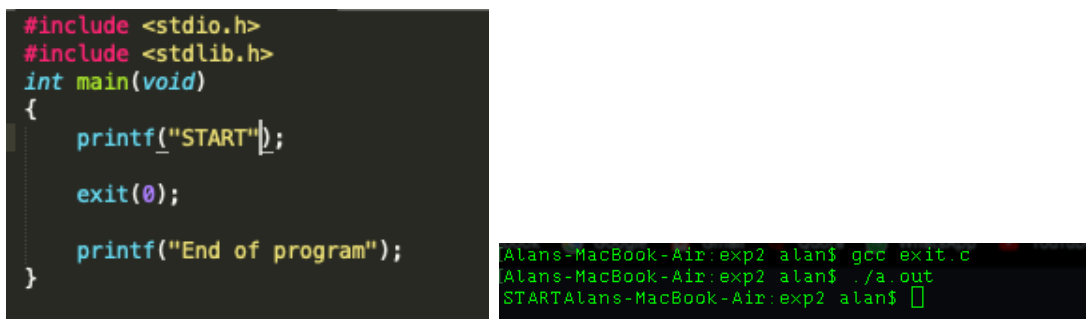


```
fork.c
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     fork();
8     printf("Using fork() system call\n");
9     return 0;
10 }
```

```
Alans-MacBook-Air:exp2 alan$ gcc fork.c
Alans-MacBook-Air:exp2 alan$ ./a.out
Using fork() system call
Using fork() system call
Alans-MacBook-Air:exp2 alan$
```

### 2.3.2 Exit Process

`exit()` is used to terminate a process. An exit code is also passed as an argument to the same.



```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("START\n");

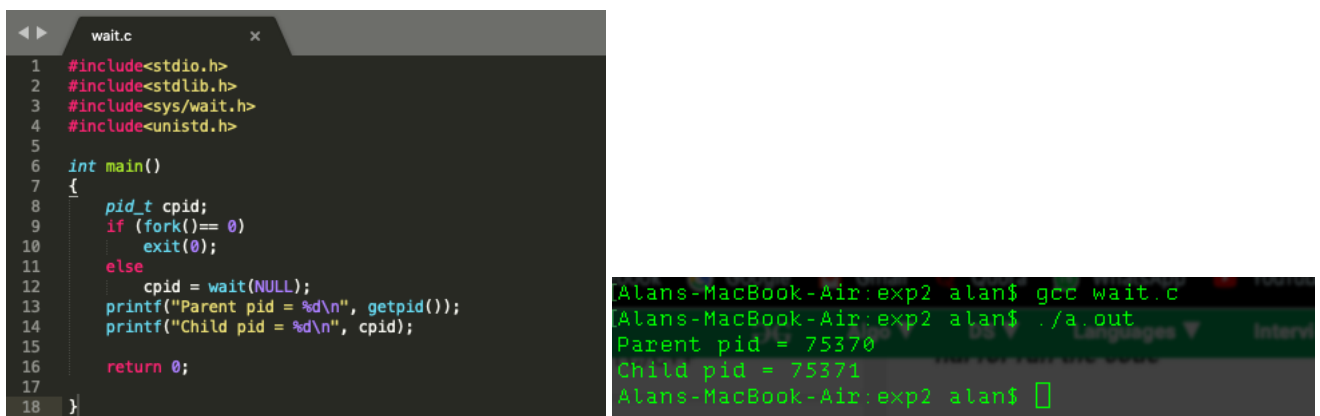
    exit(0);

    printf("End of program");
}
```

```
Alans-MacBook-Air:exp2 alan$ gcc exit.c
Alans-MacBook-Air:exp2 alan$ ./a.out
START
Alans-MacBook-Air:exp2 alan$
```

## 2.4 Wait

It is used for blocking the calling process until one of its child processes exits or a signal is received. The system call returns the PID of the terminated child on success and -1 on error.



```
wait.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<sys/wait.h>
4 #include<unistd.h>
5
6 int main()
7 {
8     pid_t cpid;
9     if (fork()== 0)
10         exit(0);
11     else
12         cpid = wait(NULL);
13     printf("Parent pid = %d\n", getpid());
14     printf("Child pid = %d\n", cpid);
15
16     return 0;
17 }
18 }
```

```
Alans-MacBook-Air:exp2 alan$ gcc wait.c
Alans-MacBook-Air:exp2 alan$ ./a.out
Parent pid = 75370
Child pid = 75371
Alans-MacBook-Air:exp2 alan$
```

## 2.5 File Management

### 2.5.1 Create File

The `open()` system call is used to create a new file or open an existing file to read data from or write data to. The system call returns the file descriptor of the new file or -1 if an error has occurred.

```
open.c
1 #include<stdio.h>
2 #include<fcntl.h>
3 #include<errno.h>
4 extern int errno;
5 int main()
6 {
7
8     int fd = open("sampletext.txt", O_RDONLY | O_CREAT);
9
10    printf("fd = %d/n", fd);
11
12    if (fd == -1)
13    {
14        printf("Error Number %d \n", errno);
15
16        perror("Program");
17    }
18    return 0;
19 }
20
21 }
```

```
~/Downloads/S6/Networking Lab/exp2
[Alans-MacBook-Air:exp2 alan$ gcc open.c
[Alans-MacBook-Air:exp2 alan$ ./a.out
fd = 3/n[Alans-MacBook-Air:exp2 alan$
```

### 2.5.2 Write File

The `write()` system call is used to write data from a file opened using the `open()` call. The system call returns the number of bytes successfully written to the file and the file pointer is also moved accordingly.

```
write.c
1 #include <unistd.h>
2
3 int main(void)
4 {
5     if (write(1, "This will be output to standard out\n", 36) != 36) {
6         write(2, "There was an error writing to standard out\n", 44);
7         return -1;
8     }
9
10    return 0;
11 }
```

```
[Alans-MacBook-Air:exp2 alan$ gcc write.c
[Alans-MacBook-Air:exp2 alan$ ./a.out
This will be output to standard out
[Alans-MacBook-Air:exp2 alan$
```

### 2.5.3 Close Handle

The `close()` system call is used to close a open file.The call returns zero on success and -1 on error. Once closed, all locks held on that file by that process are also

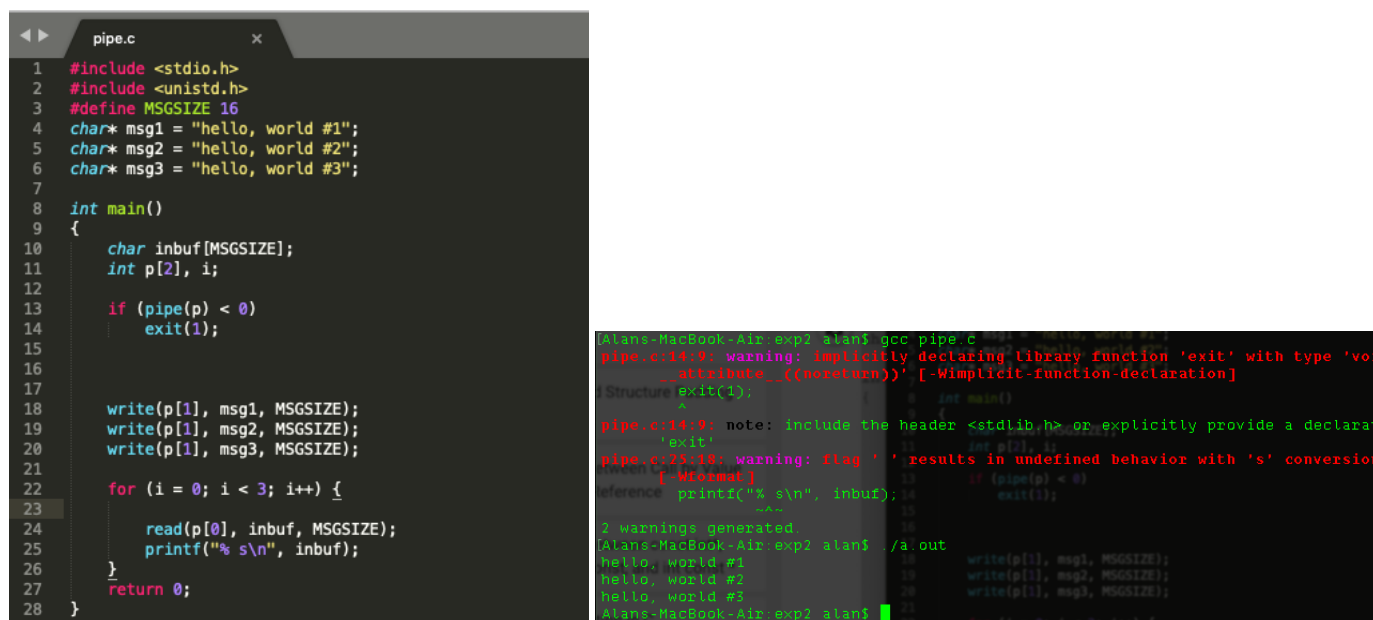


released.

## 2.6 Communication

### 2.6.1 Create Pipe

The `pipe()` system call is used to create an unnamed pipe. The created pipe is represented by two file descriptors, one to which data is written and one from which data is read. Thus, the pipe is unidirectional. The system call returns zero on success and -1 on error.



```

1  #include <stdio.h>
2  #include <unistd.h>
3  #define MSGSIZE 16
4  char* msg1 = "hello, world #1";
5  char* msg2 = "hello, world #2";
6  char* msg3 = "hello, world #3";
7
8  int main()
9  {
10     char inbuf[MSGSIZE];
11     int p[2], i;
12
13     if (pipe(p) < 0)
14         exit(1);
15
16
17     write(p[1], msg1, MSGSIZE);
18     write(p[1], msg2, MSGSIZE);
19     write(p[1], msg3, MSGSIZE);
20
21     for (i = 0; i < 3; i++) {
22
23         read(p[0], inbuf, MSGSIZE);
24         printf("%s\n", inbuf);
25     }
26     return 0;
27 }

```


```

[Alans-MacBook-Air exp2 alan$ gcc pipe.c
pipe.c:14:9: warning: implicitly declaring library function 'exit' with type 'void (*)(int)' [-Wimplicit-function-declaration]
14     exit(1);
    ~~~~~^
pipe.c:14:9: note: include the header <stdlib.h> or explicitly provide a declaration for 'exit'
pipe.c:25:18: warning: flag 's' results in undefined behavior with 's' conversion specifier [-Wformat]
25         printf("%s\n", inbuf);
                  ^
3 warnings generated
[Alans-MacBook-Air exp2 alan$ ./a.out
hello, world #1
hello, world #2
hello, world #3
[Alans-MacBook-Air exp2 alan$

```

### 2.6.2 Create File Mapping

The `shmget()` system call is used to create a shared memory segment. The memory segment has an associated key which is passed to the system call along with the size of the memory segment and some control flags. The system call returns a segment identifier on success and -1 on error.



```
shmget.cpp
1 #include <iostream>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <stdio.h>
5 using namespace std;
6
7 int main()
8 {
9
10     key_t key = ftok("shmfile",65);
11
12
13     int shmid = shmget(key,1024,0666|IPC_CREAT);
14
15     char *str = (char*) shmat(shmid,(void*)0,0);
16
17     cout<<"Write Data : ";
18     gets(str);
19
20     printf("Data written in memory: %s\n",str);
21
22     |
23     shmdt(str);
24
25     return 0;
26 }
27
```

```
Alans-MacBook-Air:exp2 alan$ g++ shmget.cpp
Alans-MacBook-Air:exp2 alan$ ./a.out
Warning: this program uses gets(), which is unsafe.
Write Data : hi
Data written in memory: hi
Alans-MacBook-Air:exp2 alan$
```

## 2.7 Information Maintenance

### 2.7.1 Current Process ID

The `getpid()` system call returns the pid of the process which called it.

### 2.7.2 SET TIMER

The `alarm()` system call is used to generate a signal after a specified amount of time has elapsed. The call returns the number of seconds remaining until any previously created alarm is due to be generated.

### 2.7.3 Sleep

The `sleep` system call causes the calling process to be suspended until the specified time in seconds has elapsed.

## 2.8 Result

The familiarization of system calls was completed successfully.

## 3 Process and Thread

### 3.1 AIM

Familiarizing and understanding the use and process and thread.

### 3.2 Theory

An executing instance of a program is called a process. Each process provides the resources needed to execute a program. A thread is a subset of the process. A thread is a path of execution within a process. A process can contain multiple threads. A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. A thread shares information like data segment, code segment, files etc. with its peer threads while it contains its own registers, stack, counter etc.

BASIS FOR COMPARISON	PROCESS	THREAD
Basic	Program in execution.	Lightweight process or part of it.
Memory sharing	Completely isolated and do not share memory.	Shares memory with each other.
Resource consumption	More	Less
Efficiency	Less efficient as compared to the process in the context of communication.	Enhances efficiency in the context of communication.
Time required for creation	More	Less
Context switching time	Takes more time.	Consumes less time.
Uncertain termination	Results in loss of process.	A thread can be reclaimed.

### 3.3 Algorithm

---

**Algorithm 1** Algorithm for creating N threads

---

```

1 START
2 Let NUMTHREAD=10
3 Procedure *MyFunction(void * tid)
4 Begin
5     Print("Thread_:_" tid)
6 End MyFunction
7 create pthread thread
8 f=fork() //create fork
9 IF (f==0)
10     PRINT "child_:_"pid"
11     Begin For i<NUMTHREAD
12         call pthread_create(&thread, NULL, MyFunction, tid)
13         i++
14     END FOR
15 ELSE
16     PRINT "parent_:_"pid"
17     Begin For i<NUMTHREAD
18         call pthread_create(&thread, NULL, MyFunction, tid)
19         i++
20     END FOR
21 ENDIF
22 STOP

```

---

### 3.4 Program

The following program was written based on the algorithm.

```

#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>

void *threadfunc(void *var)
{
    printf("Thread from function %d \n ", var);
    sleep(1);
    printf("Exiting from function %d \n", var);
}

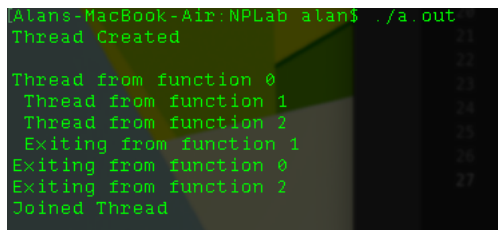
int main()
{

```

```
pthread_t id;  
printf("Thread Created \n \n");  
for(int i=0;i<3;i++)  
{  
pthread_create(&id,NULL,threadfunc,i);  
}  
pthread_join(id,NULL);  
printf("Joined Thread \n \n");  
  
exit(0);  
  
}
```

### 3.5 Output

The following output was obtained on running the program

A terminal window with a dark background and green text. The output shows the execution of a program with three threads. The first thread prints 'Thread Created' and then 'Thread from function 0'. The second thread prints 'Thread from function 1'. The third thread prints 'Thread from function 2'. After the threads finish, the program prints 'Exiting from function 1', 'Exiting from function 0', and 'Exiting from function 2'. Finally, it prints 'Joined Thread'.

```
Alan@Alan-MacBook-Air:NPLab alan$ ./a.out  
Thread Created  
Thread from function 0  
Thread from function 1  
Thread from function 2  
Exiting from function 1  
Exiting from function 0  
Exiting from function 2  
Joined Thread
```

### 3.6 Result

The familiarization of processes and thread was completed and the program executed successfully.

## 4 First Readers-Writers Problem

### 4.1 AIM

To implement first readers - writers problem.

### 4.2 Theory

#### 4.2.1 Introduction

The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object. It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. If the data is already being used by a reader say R1, and another reader R2 requests for access, it'll be forced to wait until the R1 finishes the reading. It is not efficient to wait till this happens . So in first readers-writers problem a constraint will be added so that no reader shall be kept waiting if it is opened for reading by some other reader .

#### 4.2.2 Solution

In this solution of the readers/writers problem, the first reader must lock the resource (shared file) if such is available. Once the file is locked from writers, it may be used by many subsequent readers without having them to re-lock it again.

Before entering the Critical Section , every process will now need to enter the ENTRY Section. Once it enters the ENTRY Section, it locks the ENTRY Section. ENTRY section shall remain locked until they are done with it. They're locking the ENTRY Section so that no other user can enter the section while they are using it. It will stop the race condition of users. The same is valid for EXIT section as well. Once they finishes execution, the EXIT section will also be marked. Only one user is allowed in EXIT Section at a time.

### 4.3 Algorithm

---

**Algorithm 1** Algorithm for creating N threads
 

---

```

1  START
2  Let NUMTHREAD=10
3  Procedure *MyFunction(void * tid)
4  Begin
5      Print ("Thread_:_"tid)
6  End   MyFunction
7  create pthread thread
8  f=fork() //create fork
9  IF (f==0)
10     PRINT "child_:_"pid"
11     Begin For i<NUMTHREAD
12         call pthread_create(&thread,NULL,MyFunction,tid)
13     i++
14     END FOR
15 ELSE
16     PRINT "parent_:_"pid"
17     Begin For i<NUMTHREAD
18         call pthread_create(&thread,NULL,MyFunction,tid)
19     i++
20     END FOR
21 ENDIF
22 STOP

```

---

### 4.4 Program

The following program was written based on the algorithm.

```

#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t mutex,writeblock;
int data = 0,rcount = 0;

```

```
void *reader(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
    printf("Reading by thread %i \n", arg);
    sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);

    printf("thread %i finished reading \n", arg);
}

void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&writeblock);
    data++;
    printf("Writing by thread %i \n", arg);
    sleep(1);
    sem_post(&writeblock);

    printf("thread %i finished writing \n", arg);
}

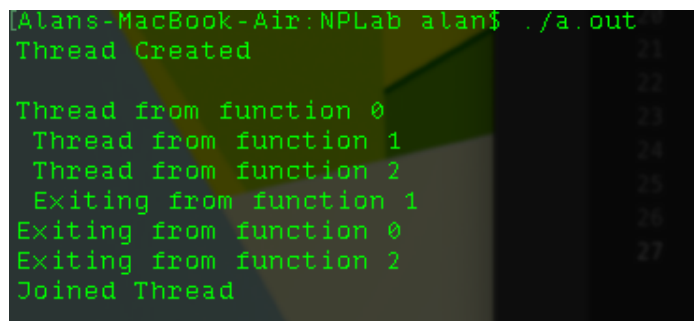
int main()
{
    int i, b;
    pthread_t rtid[5], wtid[5];
    sem_init(&mutex, 0, 1);
    sem_init(&writeblock, 0, 1);
    for (i=0; i<=4; i++)
```



```
{
    pthread_create(&wtid[i],NULL,writer,(void *)i);
    pthread_create(&rtid[i],NULL,reader,(void *)i);
}
for(i=0;i<=4;i++)
{
    pthread_join(wtid[i],NULL);
    pthread_join(rtid[i],NULL);
}
return 0;
}
```

## 4.5 Output

The following output was obtained on running the program

A terminal window with a dark background and green text. The prompt is '[Alans-MacBook-Air:NPLab alan\$ ./a.out'. The output consists of several lines: 'Thread Created', 'Thread from function 0', 'Thread from function 1', 'Thread from function 2', 'Exiting from function 1', 'Exiting from function 0', 'Exiting from function 2', and 'Joined Thread'. On the right side of the terminal, there is a vertical list of numbers from 21 to 27, each aligned with a line of output.

```
[Alans-MacBook-Air:NPLab alan$ ./a.out
Thread Created
Thread from function 0
Thread from function 1
Thread from function 2
Exiting from function 1
Exiting from function 0
Exiting from function 2
Joined Thread
```

## 4.6 Result

The familiarization of first readers-writers problem was completed and the program executed successfully.

## 5 Network Simulator NS2

### 5.1 AIM

Install network simulator NS-2 in any of the Linux operating system and simulate wired and wireless scenarios.

### 5.2 Theory

NS2 is an open-source simulation tool that runs on Linux. It is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks. It also supports various algorithms for routing and queuing. NS2 started as a variant of the REAL network simulator in 1989. This can be used for graphically showing network traffic.

#### 5.2.1 Installation

**Step 1 :**Download the NS2 file name " nsallinone-2.35.tar.gz " . Open the folder in terminal and extract the file using tar command.

```
tar -xvzf ns-allinone-2.35.tar.gz
```

**Step 2 :** We can built dependencies using the following commands.

```
sudo apt-get install build-essential autoconf automake libxmu-dev  
sudo apt-get install gcc-4.4
```

We need to specify the version of GCC used . It can be done using the following command.

```
sudo gedit ns-allinone-2.34/otcl-1.13/Makefile.in
```

**Step 3 :** We can now install the package using the following command.

```
sudo su cd ~/ns-allinone-2.35/./install
```

**Step 4 :**The final step is to tell the system,where the files for ns2 are installed or present. To do that, we have to set the environment path using the ".bashrc" file. We need to specify the path where we have installed and add it to the bottom of the file.

**Step 5 :Run NS2**

ns

**5.2.2 TCL**

Tcl is a radically simple open-source interpreted programming language. Tcl scripts are made up of commands separated by newlines or semicolons. Tcl stands for Tool Command Language. Tcl supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles.

**5.2.3 C++**

NS2 uses OTcl to create and configure a network, and uses C++ to run simulation.

**5.3 Program****5.3.1 Wired Transmission**

```
set ns [new Simulator]
set nt [open test1.tr w]ala
$ns trace-all $nt
set nf [open test1.nam w]
$ns namtrace-all $nf
$ns color 1 darkmagenta
$ns color 2 yellow
$ns color 3 blue
$ns color 4 green
$ns color 5 black
set totalNodes 3

for {set i 0} {$i < $totalNodes} {incr i} {
set node_($i) [$ns node]
}

set server 0
set router 1
set client 2
```

```
$ns duplex-link $node_($server) $node_($router) 2Mb 50ms DropTail
$ns duplex-link $node_($router) $node_($client) 2Mb 50ms DropTail

$ns duplex-link-op $node_($server) $node_($router) orient right
$ns duplex-link-op $node_($router) $node_($client) orient right

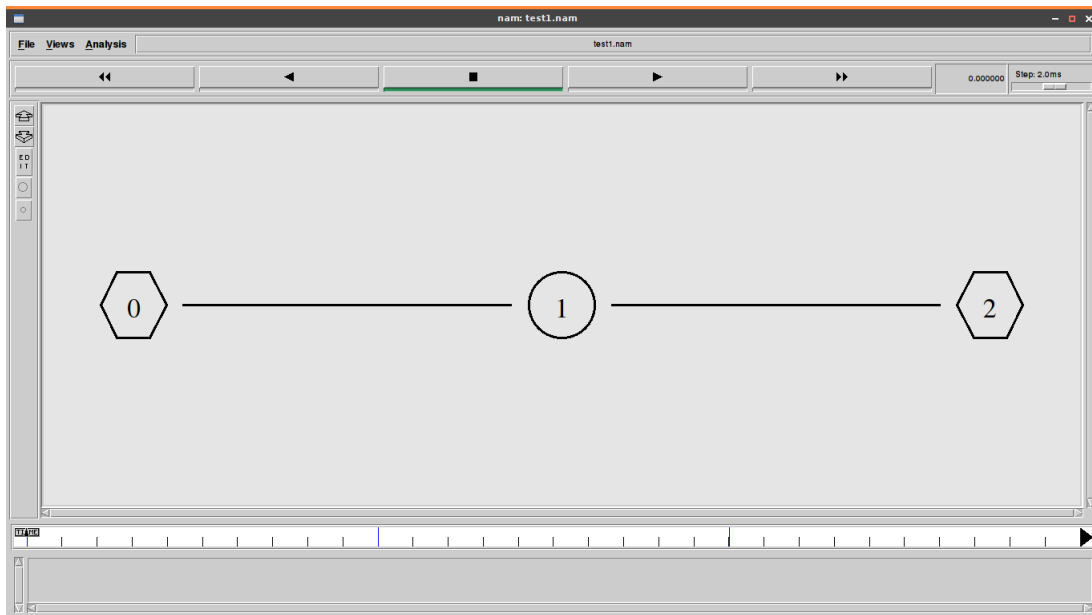
$ns at 0.0 "$node_($server) label Server"
$ns at 0.0 "$node_($router) label Router"
$ns at 0.0 "$node_($client) label Client"

$ns at 0.0 "$node_($server) color blue"
$ns at 0.0 "$node_($client) color blue"

$node_($server) shape hexagon
$node_($client) shape hexagon

proc finish {} {
  global ns nf nt
  $ns flush-trace
  close $nf
  close $nt
  puts "running nam"
  exec nam test1.nam &
  exit 0
}

#Calling finish procedure
$ns at 10.0 "finish"
$ns run
```



### 5.3.2 Wireless

```

        #initialize the variables
set val(chan) Channel/WirelessChannel ;
set val(prop) Propagation/TwoRayGround ;
set val(netif) Phy/WirelessPhy ;
set val(mac) Mac/802_11 ;
set val(ifq) Queue/DropTail/PriQueue ;
set val(ll) LL ;
set val(ant) Antenna/OmniAntenna ;
set val(ifqlen) 50 ;
set val(nn) 6 ;
set val(rp) AODV ;
set val(x) 500 ;# in metres
set val(y) 500 ;# in metres
#Adhoc OnDemand Distance Vector

#creation of Simulator
set ns [new Simulator]

#creation of Trace and namfile
set tracefile [open wireless.tr w]
$ns trace-all $tracefile

```

```
#Creation of Network Animation file
set namfile [open wireless.nam w]
$ns namtrace-all-wireless $namfile $val(x) $val(y)

#create topography
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#GOD Creation – General Operations Director
create-god $val(nn)

set channel1 [new $val(chan)]
set channel2 [new $val(chan)]
set channel3 [new $val(chan)]

#configure the node
$ns node-config -adhocRouting $val(rp) \
    -llType $val(ll) \
    -macType $val(mac) \
    -ifqType $val(ifq) \
    -ifqLen $val(ifqlen) \
    -antType $val(ant) \
    -propType $val(prop) \
    -phyType $val(netif) \
    -topoInstance $topo \
    -agentTrace ON \
    -macTrace ON \
    -routerTrace ON \
    -movementTrace ON \
    -channel $channel1

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n0 random-motion 0
$n1 random-motion 0
```

```
$n2 random-motion 0
$n3 random-motion 0
$n4 random-motion 0
$n5 random-motion 0
```

```
$ns initial_node_pos $n0 20
$ns initial_node_pos $n1 20
$ns initial_node_pos $n2 20
$ns initial_node_pos $n3 20
$ns initial_node_pos $n4 20
$ns initial_node_pos $n5 50
```

```
#initial coordinates of the nodes
```

```
$n0 set X_ 10.0
$n0 set Y_ 20.0
$n0 set Z_ 0.0
```

```
$n1 set X_ 210.0
$n1 set Y_ 230.0
$n1 set Z_ 0.0
```

```
$n2 set X_ 100.0
$n2 set Y_ 200.0
$n2 set Z_ 0.0
```

```
$n3 set X_ 150.0
$n3 set Y_ 330.0
$n3 set Z_ 0.0
```

```
$n4 set X_ 430.0
$n4 set Y_ 320.0
$n4 set Z_ 0.0
```

```
$n5 set X_ 270.0
$n5 set Y_ 120.0
$n5 set Z_ 0.0
```

```
#Dont mention any values above than 500 because in this example, we use X
```

```
#mobility of the nodes
```

```
#At what Time? Which node? Where to? at What Speed?
```

```
$ns at 1.0 "$n0 setdest 490.0 340.0 35.0"
$ns at 1.0 "$n1 setdest 490.0 340.0 5.0"
$ns at 1.0 "$n2 setdest 330.0 100.0 10.0"
$ns at 1.0 "$n3 setdest 300.0 100.0 8.0"
$ns at 1.0 "$n4 setdest 300.0 130.0 5.0"
$ns at 1.0 "$n5 setdest 190.0 440.0 15.0"
#the nodes can move any number of times at any location during the simulation
$ns at 20.0 "$n5 setdest 100.0 200.0 30.0"

#creation of agents
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"

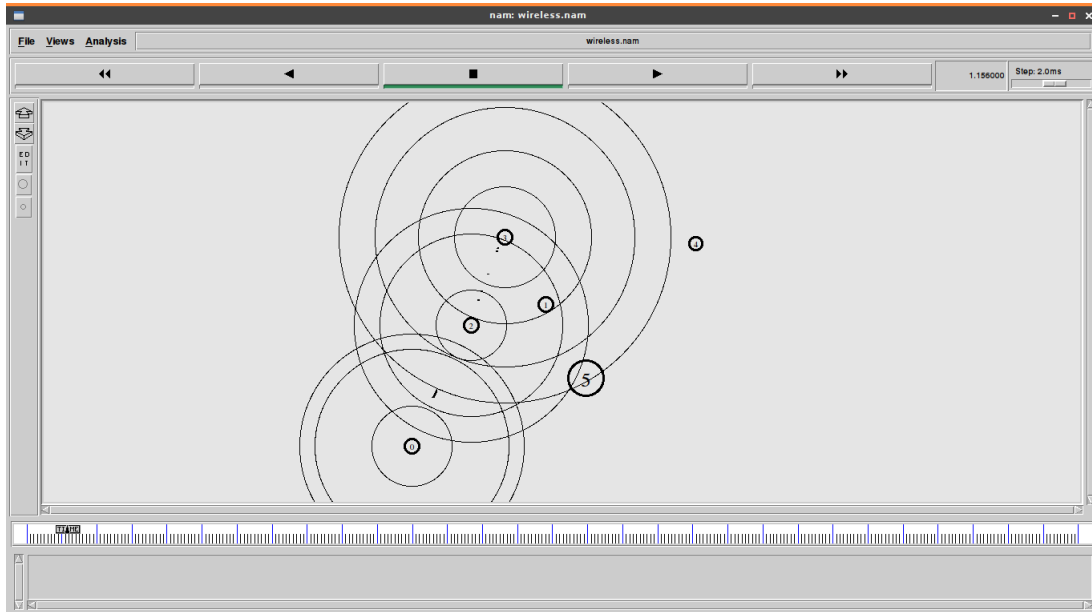
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n2 $udp
$ns attach-agent $n3 $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$ns at 1.0 "$cbr start"

$ns at 30.0 "finish"

proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exit 0
}

puts "Starting Simulation"
$ns run
```





## 5.4 Result

Installed NS2 and implemented wireless and wired simulations of on NS2 and outputs were verified.

## 6 Second Readers - Writers Problem

### 6.1 AIM

Implement the Second Readers-Writers problem

### 6.2 Theory

In second readers writers problem , no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is called writers-preference .In this solution, preference is given to the writers. This is accomplished by forcing every reader to lock and release the readtry semaphore individually. The writers on the other hand don't need to lock it individually. Only the first writer will lock the readtry and then all subsequent writers can simply use the resource as it gets freed by the previous writer. The very last writer must release the readtry semaphore, thus opening the gate for readers to try reading.

### 6.3 Algorithm

```
int readcount , writecount ;
semaphore rmutex , wmutex , readTry , resource ;

//READER
reader () {
<ENTRY Section>
    readTry.P();
    rmutex.P();
    readcount++;
    if (readcount == 1)
        resource.P();
    rmutex.V();
    readTry.V();

<CRITICAL Section>
    //reading is performed

<EXIT Section>
    rmutex.P();
```

```
    readcount--;
    if (readcount == 0)
        resource.V();
    rmutex.V();
}

//WRITER
writer() {
<ENTRY Section>
    wmutex.P();
    writecount++;
    if (writecount == 1)
        readTry.P();
    wmutex.V();
<CRITICAL Section>
    resource.P();
    //writing is performed
    resource.V();

<EXIT Section>
    wmutex.P();
    if (writecount == 0)
        readTry.V();
    wmutex.V();
}
```

## 6.4 Program

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t mutex, writeblock;
int data = 0, rcount = 0;

void *reader(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
```

```
    if (rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
    printf("Data read by the reader%d is %d\n",f,data);
    sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if (rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);
}

void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&writeblock);
    data++;
    printf("Data written by the writer%d is %d\n",f,data);
    sleep(1);
    sem_post(&writeblock);
}

main()
{
    int i,b;
    pthread_t rtid[5],wtid[5];
    sem_init(&mutex,0,1);
    sem_init(&writeblock,0,1);
    for (i=0;i<=2;i++)
    {
        pthread_create(&wtid[i],NULL,writer,(void *)i);
        pthread_create(&rtid[i],NULL,reader,(void *)i);
    }
    for (i=0;i<=2;i++)
    {
        pthread_join(wtid[i],NULL);
        pthread_join(rtid[i],NULL);
    }
}
```

## 6.5 Output

```
Alans-MacBook-Air:NPLab alan$ ./a.out
Data writen by the writer0 is 1
Data writen by the writer2 is 3
Data writen by the writer1 is 2
Data read by the reader0 is 2
Data read by the reader1 is 2
Data read by the reader2 is 3
Alans-MacBook-Air:NPLab alan$
```

## 6.6 Result

The second readers writers problem was implemented and the output was verified

## 7 Pipes, Message Queue and Shared Memory

### 7.1 AIM

Implement programs for Inter Process Communication using PIPE, Message Queue and Shared Memory.

### 7.2 Theory

#### 7.2.1 Pipes

A pipe acts as a conduit allowing two processes to communicate. Pipes were one of the first IPC mechanisms in early UNIX systems. They typically provide one of the simpler ways for processes to communicate with one another, although they also have some limitations. Two common types of pipes used on both UNIX and Windows systems: ordinary pipes and named pipes. Named pipes can be used to provide communication between processes on the same computer or between processes on different computers across a network, as in case of a distributed system. The Ordinary pipe in Operating Systems allows the two procedures to communicate in a standard way: the procedure writes on the one end (the write end) and reads on the consumer side or another end (the read-end).

#### 7.2.2 Ordinary Pipes

Ordinary pipes allow two processes to communicate in standard producer consumer fashion: the producer writes to one end of the pipe (the write-end) and the consumer reads from the other end (the read-end). Ordinary pipes are unidirectional. If two-way communication is required, two pipes must be used, with each pipe sending data in a different direction. On UNIX systems, ordinary pipes are constructed using the function

```
pipe( int fd[ 2 ] )
```

#### 7.2.3 Named Pipes

Named pipes provide a much more powerful communication tool. Communication can be bidirectional, and no parent-child relationship is required. Once a named pipe is established, several processes can use it for communication. In fact, in a typical

scenario, a named pipe has several writers. Additionally, named pipes continue to exist after communicating processes have finished.

#### 7.2.4 Message Queues

A message queue can be created by one process and used by multiple processes that read and/or write messages to the queue. Message queues have internal structure. With a named pipe, a writer is just pumping bits. For a reader, there's no distinction between different calls to `write()` from different writers. Message queues are priority-driven.

#### 7.2.5 Shared Memory

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

### 7.3 Program

#### 7.3.1 Pipes

```
#include <stdio.h>
#include <unistd.h>

int main() {
    int pipefds[2];
    int returnstatus;
    char writemessages[2][20] = { "Hi", "Hello" };
    char readmessage[20];
    returnstatus = pipe(pipefds);
```

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

printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Reading from pipe Message 1 is %s\n", readmessage);
printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Reading from pipe Message 2 is %s\n", readmessage);
return 0;
}
```

### 7.3.2 Message Queue

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;
    key = ftok("progfile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    gets(message.mesg_text);
}
```



```
    msgsnd(msgid, &message, sizeof(message), 0);
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

### 7.3.3 Shared memory

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    key_t key = ftok("shmfile", 65);

    int shmid = shmget(key, 1024, 0666|IPC_CREAT);

    char *str = (char*) shmat(shmid, (void*)0, 0);

    cout<<"Write Data : ";
    gets(str);

    printf("Data written in memory: %s\n", str);

    shmdt(str);

    return 0;
}
```

## 7.4 Output

### 7.4.1 Pipes

```
Alans-MacBook-Air:NPLab alan$ ./a.out
Writing to pipe - Message 1 is Hi
Reading from pipe - Message 1 is Hi
Writing to pipe - Message 2 is Hi
Reading from pipe - Message 2 is Hello
Alans-MacBook-Air:NPLab alan$
```

### 7.4.2 Message Queue

```
Alans-MacBook-Air:NPLab alan$ gcc msg.c
Alans-MacBook-Air:NPLab alan$ ./a.out
warning: this program uses gets(), which is unsafe.
Write Data : hi
Data send is : hi
Alans-MacBook-Air:NPLab alan$
```

### 7.4.3 Shared Memory

```
Alans-MacBook-Air:NPLab alan$ g++ sharedmem.cpp
Alans-MacBook-Air:NPLab alan$ ./a.out
warning: this program uses gets(), which is unsafe.
Write Data : hello
Data written in memory: hello
Alans-MacBook-Air:NPLab alan$
```

## 7.5 Result

Interprocess communication was implemented using pipes,message queue and Shared memory and the output was verified.

## 8 Client-Server Communication using TCP

### 8.1 AIM

To implement a Client-Server communication using Socket Programming and TCP as transport layer protocol

### 8.2 Theory

#### 8.2.1 TCP

Transmission Control Protocol(TCP) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data.

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

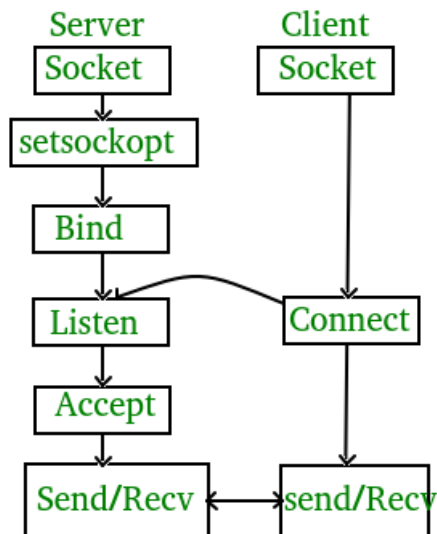
When loading a web page, the computer sends TCP packets to the web server's address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which the web browser stitches together to form the web page and display it to you. TCP enables two way communication — the remote system sends packets back to acknowledge it is received your packets.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors. Packets sent with TCP are tracked so no data is lost or corrupted in transit.

#### 8.2.2 Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other

socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



- Socket creation:

```
int sockfd = socket(domain, type, protocol)
```

- sockfd: socket descriptor, an integer (like a file-handle)
- domain: integer, communication domain e.g., AF\_INET (IPv4 protocol) , AF\_INET6 (IPv6 protocol)
- type: communication type

SOCK\_STREAM: TCP(reliable, connection oriented) SOCK\_DGRAM: UDP(unreliable, connectionless)

- protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man proto- cols for more details)

- Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure).

- Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

- Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

- Connect:

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t
addrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

## 8.3 Program

### 8.3.1 Server Side

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Created Socket ")
port= 8080

s.bind(('',port))
print("Binding Socket to ",port)

s.listen(5)
```

```
while True:
    c,addr=s.accept()
    message=c.recv(1024)
    print(message)
    print ('Got Connection from',addr)
    print('Thanks for connecting')
    c.close()
    exit(0)
```

### 8.3.2 Client Side

```
import socket
import sys

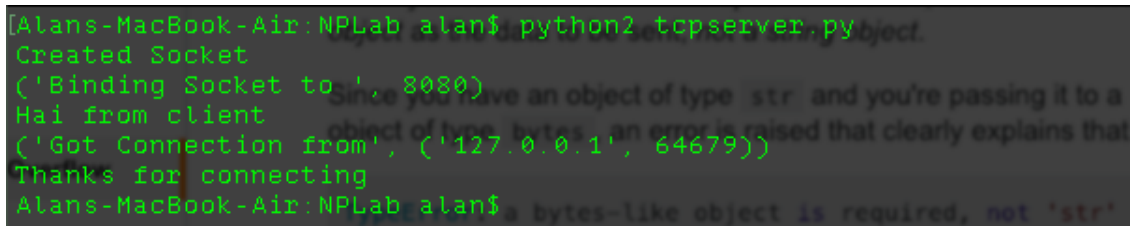
s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)

port = 8080

s.connect(('127.0.0.1',port))
str=("Hai from client")
byte = str.encode()
s.send(byte)
print (s.recv(1024))
print("Message sent from Client")
s.close()
```

## 8.4 Output

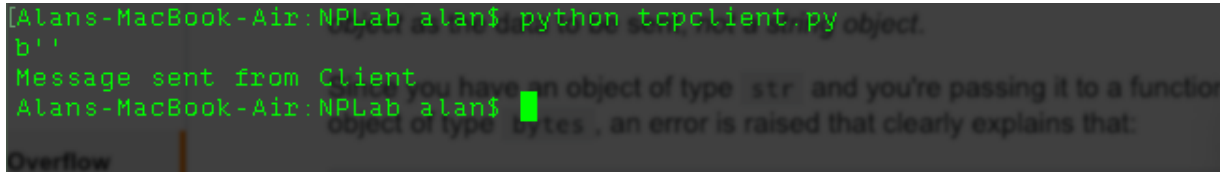
### 8.4.1 Server



```
[Alans-MacBook-Air:NPLab alan$ python2 tcpserver.py
Created Socket
('Binding Socket to', 8080)
Hai from client
('Got Connection from', ('127.0.0.1', 64679))
Thanks for connecting
Alans-MacBook-Air:NPLab alan$
```

Since you have an object of type 'str' and you're passing it to a object of type 'bytes', an error is raised that clearly explains that a bytes-like object is required, not 'str'

### 8.4.2 Client



```
Alans-MacBook-Air:NPLab alan$ python tcpclient.py
b''
Message sent from Client
Alans-MacBook-Air:NPLab alan$
```

## 8.5 Result

A Client-Server communication program has been implemented in python language using libraries for Socket Programming and TCP as transport layer Protocol. The program was compiled and runned using python compiler for python 3.7.0 in MacOS Mojave.

In the program, we use socket library from Python. This library is used to create sockets and the created sockets listen to port 8080 . The socket receives communications from the clients to through this port and prints the messages send. This is done in a while loop having true condition always. The client also creates a socket as above and sends in the message to the port , which 8080 here. It displays the message in server and closes the client socket.

## 9 Client-Server Communication using UDP

### 9.1 AIM

To implement a Client-Server communication using Socket Programming and UDP as trans- port layer protocol

### 9.2 Theory

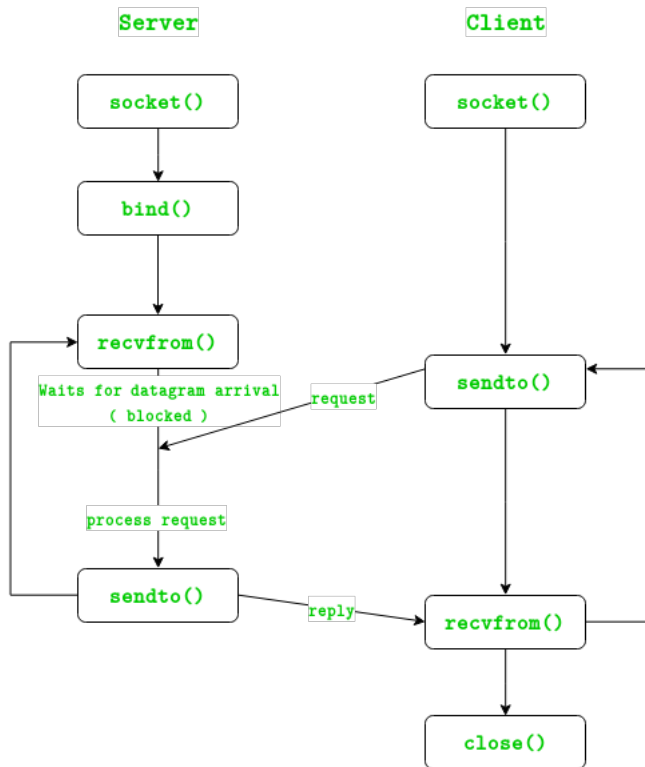
#### 9.2.1 UDP Protocol

UDP stands for User Datagram Protocol. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out.

When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet, it will just continue sending the next packets. There is no guarantee that all the packets are delivered and there is no way to ask for a packet again if it misses out, but losing all this overhead means the computers can communicate more quickly.

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.





UDP Server:

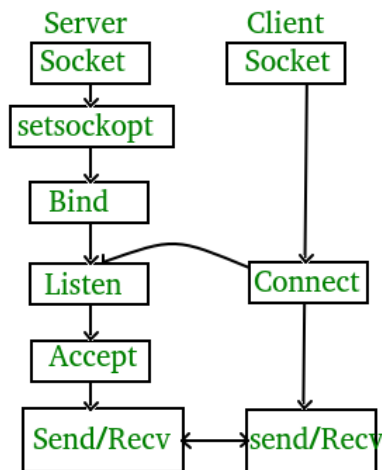
- Create UDP socket.
- Bind the socket to server address.
- Wait until datagram packet arrives from client.
- Process the datagram packet and send a reply to client.
- Go back to Step 3.

UDP Client:

- Create UDP socket.
- Send message to server.
- Wait until response from server is recieved.
- Process reply and go back to step 2, if necessary.
- Close socket descriptor and exit.

### 9.2.2 Socket Programming

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



- Socket creation:

```
int sockfd = socket(domain, type, protocol)
```

– sockfd: socket descriptor, an integer (like a file-handle)

– domain: integer, communication domain e.g., AF\_INET (IPv4 protocol) , AF\_INET6 (IPv6 protocol)

– type: communication type

SOCK\_STREAM: TCP(reliable, connection oriented) SOCK\_DGRAM: UDP(unreliable, connectionless)

– protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man proto- cols for more details)

- Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure).

- `sendto`

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen)
```

- `sockfd` – File descriptor of socket
- `buf` – Application buffer containing the data to be sent
- `len` – Size of buf application buffer
- `flags` – Bitwise OR of flags to modify socket behaviour
- `dest_addr` – Structure containing address of destination
- `addrlen` – Size of dest addr structure

- `recvfrom`

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen)
```

- `sockfd` – File descriptor of socket
- `buf` – Application buffer in which to receive data
- `len` – Size of buf application buffer
- `flags` – Bitwise OR of flags to modify socket behaviour
- `src_addr` – Structure containing source address is returned
- `addrlen` – Variable in which size of src addr structure is returned

## 9.3 Program

### 9.3.1 Server Side

```
import socket

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

port=8080

s.bind((' ', port))
```

```
msgFromServer      = "Hello UDP Client "

bytesToSend        = str.encode(msgFromServer)

while True:
    msg, addr=s.recvfrom(1024)
    print('Got Connection from', addr)
    print('Message from Client: ', msg)
    s.sendto(bytesToSend, addr)
    print("Thanks for connecting")
    exit(0)
```

### 9.3.2 Client Side

```
// Client side implementation of UDP client-server model

import socket
import sys

msgFromClient      = "Hello UDP Server\n"

bytesToSend        = str.encode(msgFromClient)

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

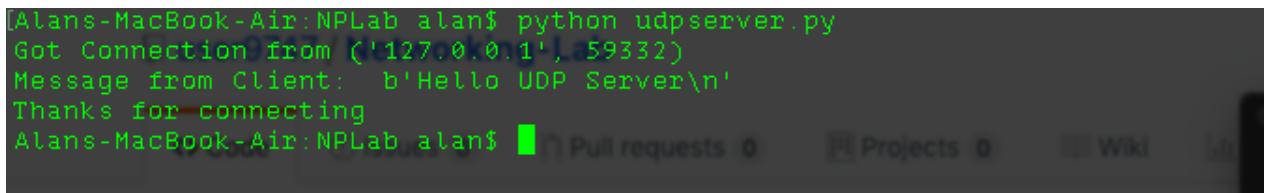
port = 8080

s.sendto(bytesToSend, ('127.0.0.1', port))
print("Sending message to ", port)
print("\n")
msgFromServer=s.recvfrom(1024)
# msg = "Message from Server:  {}".format(msgFromServer[0])

print(" Message from Server: ", msgFromServer[0])
print("\n ")
s.close()
```

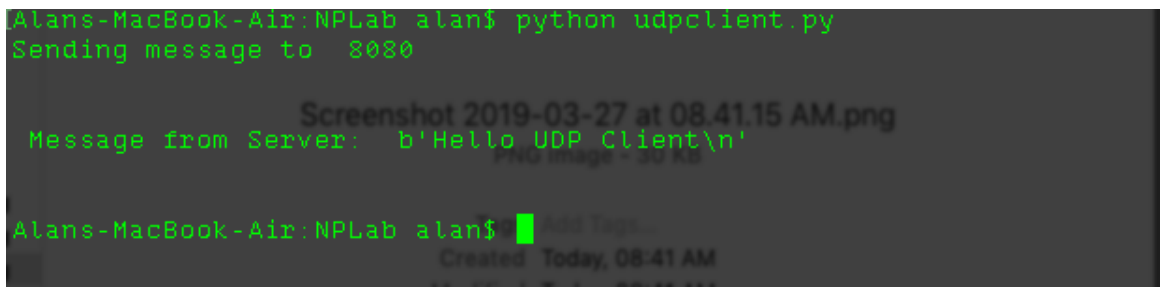
## 9.4 Output

### 9.4.1 Server

A terminal window on a Mac showing the execution of a Python UDP server. The prompt is 'Alans-MacBook-Air:NPLab alan\$'. The command 'python udpserver.py' is entered. The output shows a connection from '127.0.0.1' on port '59332', followed by the received message 'b'Hello UDP Server\\n'', and the response 'Thanks for connecting'.

```
Alans-MacBook-Air:NPLab alan$ python udpserver.py
Got Connection from ('127.0.0.1', 59332)
Message from Client:  b'Hello UDP Server\\n'
Thanks for connecting
Alans-MacBook-Air:NPLab alan$
```

### 9.4.2 Client

A terminal window on a Mac showing the execution of a Python UDP client. The prompt is 'Alans-MacBook-Air:NPLab alan\$'. The command 'python udpclient.py' is entered. The output shows the message 'Sending message to 8080'.

```
Alans-MacBook-Air:NPLab alan$ python udpclient.py
Sending message to 8080
```

## 9.5 Result

A Client-Server communication program has been implemented in python language using libraries for Socket Programming and UDP as transport layer Protocol. The program was compiled and excuted using python compiler for python 3.7.0 in MacOS Mojave.

The server code creates a UDP server and it will be running inside an infinite while loop. The server will be receiving message send from the UDP client. The socket is bind to port 8080. Similarly the client also sends message to the server and it is printed in the server. Then it prints the message that is sent from the server and the socket is closed.

## 10 Multiuser Chat server

### 10.1 AIM

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

### 10.2 Theory

#### 10.2.1 TCP Protocol

Transmission Control Protocol(TCP) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data.

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

When loading a web page, the computer sends TCP packets to the web server's address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which the web browser stitches together to form the web page and display it to you. TCP enables two way communication — the remote system sends packets back to acknowledge it is received your packets.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors. Packets sent with TCP are tracked so no data is lost or corrupted in transit.

#### 10.2.2 Client

A client is requester of this service. A client program request for some resources to the server and server responds to that request.

### 10.2.3 Server

A server is a software that waits for client requests and serves or processes them accordingly . A server may communicate with other servers inorder to serve the client requests.

### 10.2.4 Socket

Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

### 10.2.5 Multi Threading

A thread is sub process that runs a set of commands individually of any other thread. So, every time a user connects to the server, a separate thread is created for that user and communication from server to client takes place along individual threads based on socket objects created for the sake of identity of each client. We will require two scripts to establish the chat room. One to keep the serving running, and another that every client should run in order to connect to the server.

### 10.2.6 Server Side Script

The server side script will attempt to establish a socket and bind it to an IP address and port specified by the user. The script will then stay open and receive connection requests, and will append respective socket objects to a list to keep track of active connections. Every time a user connects, a separate thread will be created for that user. In each thread, the server awaits a message, and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.

### 10.2.7 Client Side Script

The client side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. If

the input is from the user, it sends the message that the users enters to the server for it to be broadcasted to other users. This is the client side script, that each user must use in order to connect to the server.

## **10.3 Algorithm**

### **10.4 Server**

- 1 START
- 2 Create TCP SOCKET
- 3 Bind SOCKET to a PORT
- 4 Start listening at the binded PORT for connection from CLIENTs
- 5 append new CLIENT to list clients
- 6 CREATE a new thread for each CLIENT which accepts messages from CLIENT and broadcast to all in list clients
- 7 STOP

### **10.5 Client**

- 1 START
- 2 Create TCP SOCKET
- 3 CONNECT to SERVER at IP ,PORT
- 4 CREATE a thread
- 5 WHILE true :
- 6 RECEIVE user input
- 7 SEND input to SERVER
- 8 WHILE true :
- 9 RECEIVE message from SERVER
- 10 PRINT message
- 11 STOP

## **10.6 Program**

### **10.6.1 Server Side**



```

import socket
import socket
import select
import sys
from _thread import *

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

port=8001

s.bind((' ',port))

s.listen(5)
list_of_clients = []
def clientThread(conn,addr):
    wel="welcome"
    welcome = wel.encode()
    conn.send(welcome)
    while True:
        message=conn.recv(1024)
        if message:
            print("[ " + str(addr[0]) + ":" + str(addr[1]) + "
] :")
            print(message)
            message = message.decode()
            message_to_send = "<" + addr[0] + " : " + str(addr[1]) + "> :"
            msg2 = message_to_send.encode()
            broadcast(msg2,conn)
        else:
            remove(conn)

def broadcast(message,conn):
    for client in list_of_clients:
        if client!=conn:
            client.send(message)

def remove(conn):
    if conn in list_of_clients:
        list_of_clients.remove(conn)

```

```
while True:
    conn,addr=s.accept()
    print ('Got Connection from ',addr)
    list_of_clients.append(conn)
    start_new_thread(clientThread,(conn,addr))
```

```
conn.close()
```

### 10.6.2 Client Side

```
import socket
import select
import sys
from _thread import *

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

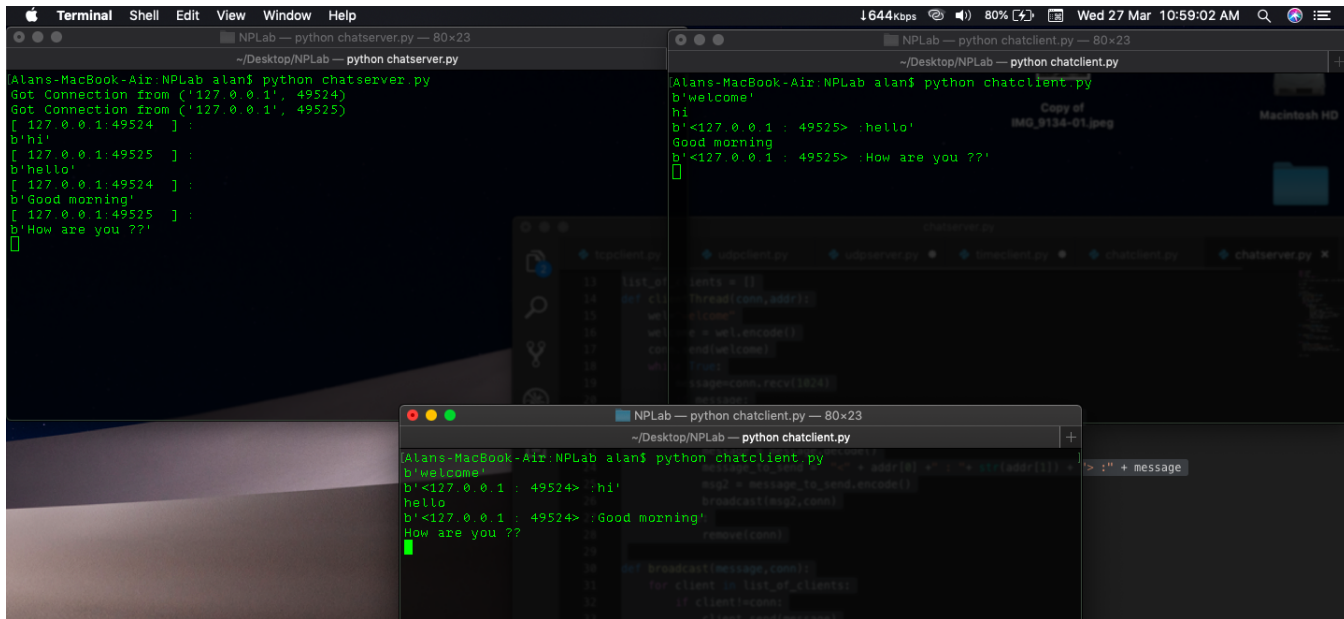
port = 8001
s.connect(('127.0.0.1',port))
def sendMessage(so):
    while True:
        message=input()
        msg = message.encode()
        so.send(msg)

start_new_thread(sendMessage,(s,))
while True:
    data = s.recv(1024)

    print (data)

s.close()
```

## 10.7 Output

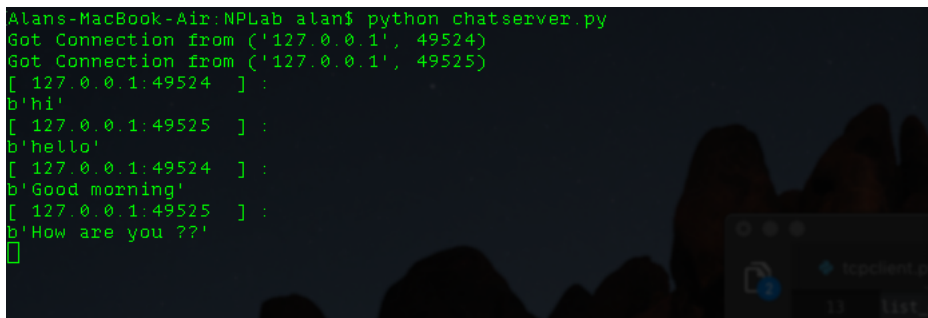


```

Alans-MacBook-Air:NPLab alan$ python chatserver.py
Got Connection from ('127.0.0.1', 49524)
Got Connection from ('127.0.0.1', 49525)
[ 127.0.0.1:49524 ] :
b'hi'
[ 127.0.0.1:49525 ] :
b'hello'
[ 127.0.0.1:49524 ] :
b'Good morning'
[ 127.0.0.1:49525 ] :
b'How are you ??'
[]

Alans-MacBook-Air:NPLab alan$ python chatclient.py
b'welcome'
hi
b'<127.0.0.1 : 49525> :hello'
Good morning
b'<127.0.0.1 : 49525> :How are you ??'
[]

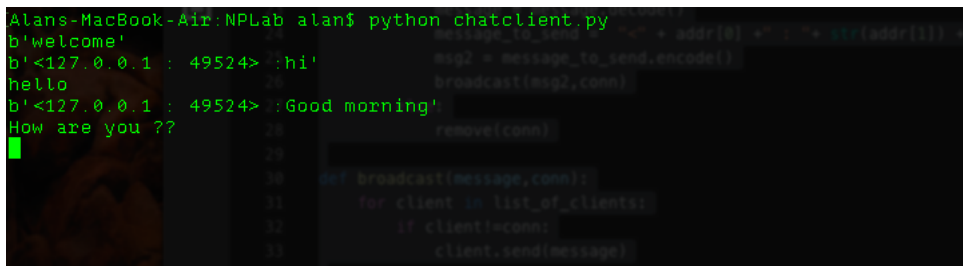
```



```

Alans-MacBook-Air:NPLab alan$ python chatserver.py
Got Connection from ('127.0.0.1', 49524)
Got Connection from ('127.0.0.1', 49525)
[ 127.0.0.1:49524 ] :
b'hi'
[ 127.0.0.1:49525 ] :
b'hello'
[ 127.0.0.1:49524 ] :
b'Good morning'
[ 127.0.0.1:49525 ] :
b'How are you ??'
[]

```



```

Alans-MacBook-Air:NPLab alan$ python chatclient.py
b'welcome'
b'<127.0.0.1 : 49524> :hi'
hello
b'<127.0.0.1 : 49524> :Good morning'
How are you ??
[]

```

## 10.8 Result

Implemented TCP Multi Client Chat Server on Python 3.7.0 and executed on macOS Majove and outputs were verified.

Server code creates a TCP socket using the socket library. Then binds the server to port 8080. The socket then listens for communications to this port. In an infinite while loop the server accepts the connection and address from connecting clients. These connections are appended to a list of clients. A thread is created for each of the clients which accepts messages from clients in its own thread and then broadcasts the message to all clients in the list of clients. Client code creates a TCP socket same as above. Then connects to the IP and port of the server. It then creates a thread for an infinite loop for getting user input and sending this message to server. The parent thread is also an infinite while loop that displays messages it receives from the server.

## 11 Concurrent Time Server - UDP

### 11.1 AIM

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

### 11.2 Theory

#### 11.2.1 UDP Protocol

UDP stands for User Datagram Protocol. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out.

When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet, it will just continue sending the next packets. There is no guarantee that all the packets are delivered and there is no way to ask for a packet again if it misses out, but losing all this overhead means the computers can communicate more quickly.

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

#### 11.2.2 Timer Server

Implement Concurrent Time Server application using UDP to execute the program at remote server. Client sends a time request to the server, server sends its system time back to the client. Client displays the result. We have a UDP based application which sends back current system time to the server indicating that some operation has been performed at the server.

### 11.3 Client

A client is requester of this service. A client program request for some resources to the server and server responds to that request.

## 11.4 Server

A server is a software that waits for client requests and serves or processes them accordingly . A server may communicate with other servers inorder to serve the client requests.

## 11.5 Socket

Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

## 11.6 Algorithm

### 11.6.1 Server

- 1 START
- 2 Create UDP SOCKET
- 3 Bind SOCKET t o a PORT
- 4 WHILE TRUE:
5.     RECEIVE time request and address ( IP ,PORT) from CLIENT
6.     SEND time t o CLIENT address
7. STOP

### 11.6.2 Client

1. START
- 2 CREATE UDP SOCKET
- 3 SEND time request to server at IP and PORT
- 4 RECEIVE time and address from server
- 5.STOP

## 11.7 Program

### 11.7.1 Server Side

```
import socket
import datetime

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

port=8080

s.bind(('', port))
while True:
    msg, addr=s.recvfrom(1024)
    print('Got Connection from ', addr)
    print('Message from Client: ', msg)
    now = datetime.datetime.now()
    time = now.strftime("%H:%M:%S")
    bytesToSend = str.encode(time)
    s.sendto(bytesToSend, addr)
    print("Thanks for connecting")
    exit(0)
```

### 11.7.2 Client Side

```
/import socket
import sys

msgFromClient = "Client Requesting Time...."
bytesToSend = str.encode(msgFromClient)
s= socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
port = 8080

s.sendto(bytesToSend, ('127.0.0.1', port))
msgFromServer=s.recvfrom(1024)
print("Time from Server: ", msgFromServer[0])
s.close()
```

## 11.8 Output

### 11.8.1 Server

```
Alans-MacBook-Air:NPLab alan$ python timeserver.py
Got Connection from ('127.0.0.1', 60058)
Message from Client:  b'Client Requesting Time....'
Thanks for connecting
Alans-MacBook-Air:NPLab alan$
```

### 11.8.2 Client

```
Alans-MacBook-Air:NPLab alan$ python timeclient.py
Time from Server:  b'09:45:06'
Alans-MacBook-Air:NPLab alan$ █
```

## 11.9 Result

UDP Concurrent time server program has been implemented in python 3.7.0 . The program was compiled and executed using python compiler for python 3.7.0 in MacOS Mojave and the output was verified.

Server code creates a udp socket using the socket library. Then binds the server to port 8080. In an infinite while loop the server receives time request and address from sending clients. Then it sends the current time to this address. Client code creates a UDP socket same as above. Then sends a time request to the ip and port of the server. It then receives the server time from the server. Then displays the time from the server. Then closes the socket.



## 12 Distance Vector Routing Protocol

### 12.1 AIM

Implement and simulate algorithm for Distance vector routing protocol.

### 12.2 Theory

#### 12.2.1 Distance Vector Routing Protocol

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as Bellman-Ford algorithm).

Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors' distance vectors. A distance-vector routing protocol in data networks determines the best route for data packets based on distance. Distance-vector routing protocols measure the distance by the number of routers a packet has to pass, one router counts as one hop. Some distance-vector protocols also take into account network latency and other factors that influence traffic on a given route. To determine the best route across a network routers, on which a distance-vector protocol is implemented, exchange information with one another, usually routing tables plus hop counts for destination networks and possibly other traffic information. Distance-vector routing protocols also require that a router informs its neighbours of network topology changes periodically.

Routers that use distance-vector protocol determine the distance between themselves and a destination. The best route for Internet Protocol packets that carry data across a data network is measured in terms of the numbers of routers (hops) a packet has to pass to reach its destination network. Additionally some distance-vector protocols take into account other traffic information, such as network latency. To establish the best route, routers regularly exchange information with neighbouring routers, usually their routing table, hop count for a destination network and possibly other traffic related information. Routers that implement distance-vector protocol rely purely on the information provided to them by other routers, and do not assess the network topology.

## 12.3 Algorithm

1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
  - 4 It receives a distance vector from a neighbor containing different information than before.
  - 5 It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

```
Dx(y) = Estimate of least cost from x to y
C(x,v) = Node x knows cost to each neighbor v
Dx = [Dx(y): y ∈ N ] = Node x maintains distance vector
Node x also maintains its neighbors' distance vectors
- For each neighbor v, x maintains Dv = [Dv(y): y ∈ N ]
```

## 12.4 Program

```
#include<stdio.h>
#include<iostream>
using namespace std;
struct node
{
    unsigned dist[6];
    unsigned from[6];
}DVR[10];
int main()
{
    cout<<"\n\n————— Distance Vector Routing Algorithm —————\n\n";
    int costmat[6][6];
    int nodes, i, j, k;
    cout<<"\n\n Enter the number of nodes : ";
    cin>>nodes; //Enter the nodes
    cout<<"\n Enter the cost matrix : \n" ;
    for(i = 0; i < nodes; i++)
```

```

{
    for(j = 0; j < nodes; j++)
    {
        cin>>costmat[i][j];
        costmat[i][i] = 0;
        DVR[i].dist[j] = costmat[i][j];
        DVR[i].from[j] = j;
    }
}

for(i = 0; i < nodes; i++)
for(j = i+1; j < nodes; j++)
for(k = 0; k < nodes; k++)
    if(DVR[i].dist[j] > costmat[i][k] + DVR[k].dist[j])
    {
        //We calculate the minimum distance
        DVR[i].dist[j] = DVR[i].dist[k] + DVR[k].dist[j];
        DVR[j].dist[i] = DVR[i].dist[j];
        DVR[i].from[j] = k;
        DVR[j].from[i] = k;
    }
for(i = 0; i < nodes; i++)
{
    cout<<"\n\n For router: "<<i+1;
    for(j = 0; j < nodes; j++)
        cout<<"\t\n node "<<j+1<<" via "<<DVR[i].from[j]+1<<
        " Distance "<<DVR[i].dist[j];
}
cout<<" \n\n ";
return 0;
}

```

## 12.5 Output

```
Alans-MacBook-Air:NPLab alan$ g++ -o dist distancevectrout.cpp
Alans-MacBook-Air:NPLab alan$ ./dist

----- Distance Vector Routing Algorithm-----

Enter the number of nodes : 3

Enter the cost matrix :
1
2
3
4
5
6
7
9
2

For router: 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3

For router: 2
node 1 via 1 Distance 4
node 2 via 2 Distance 0
node 3 via 3 Distance 6

For router: 3
node 1 via 1 Distance 7
node 2 via 2 Distance 9
node 3 via 3 Distance 0

Alans-MacBook-Air:NPLab alan$
```

## 12.6 Result

Implemented Distance Vectore routing algorithm in C++ and compiled using version 4.2.1 on macOS Mojave . The output was obtained and verified.

In this program the nodes find out the distance between each of the nodes initially from the router. Later on the shortest distance possible is calculated , this can be done by making use of any the paths possible. Later on the distance between each of the nodes, traversing through different routes is produced.

## 13 Link state routing protocol

### 13.1 AIM

Implement and simulate algorithm for Link state routing protocol.

### 13.2 Theory

#### 13.2.1 Link state routing protocol

Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation.

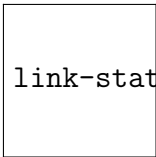
Features of link state routing protocols –

Link state packet – A small packet that contains routing information.

Link state database – A collection information gathered from link state packet.

Shortest path first algorithm (Dijkstra algorithm) – A calculation performed on the database results into shortest path

Routing table – A list of known paths and interfaces.



link-state-algo.jpg

### 13.3 Program

```
#include <stdio.h>
#include <string.h>
int main()
{
    int count, source, i, j, k, w, v, min;
```

```
int cost_matrix[100][100], distance[100], last[100];
int flag[100];
printf("Enter the no of routers : ");
scanf("%d",&count);
printf("Enter the cost matrix\n");
for(i=0;i<count;i++)
{
    for(j=0;j<count;j++)
    {
        printf("cost_matrix[%d][%d] : ",i,j);
        scanf("%d",&cost_matrix[i][j]);
        if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;
    }
}
printf("Enter the source router:");
scanf("%d",&source);
for(v=0;v<count;v++)
{
    flag[v]=0;
    last[v]=source;
    distance[v]=cost_matrix[source][v];
}
flag[source]=1;
for(i=0;i<count;i++)
{
    min=1000;
    for(w=0;w<count;w++)
    {
        if(!flag[w])
            if(distance[w]<min)
            {
                v=w;
                min=distance[w];
            }
    }
    flag[v]=1;
    for(w=0;w<count;w++)
    {
        if(!flag[w])
            if(min+cost_matrix[v][w]<distance[w])
```

```
        {
            distance[w]=min+cost_matrix[v][w];
            last[w]=v;
        }
    }
}
for ( i=0;i<count;i++)
{
    printf("\n%d==>%d——Path taken:%d \n",source,i,i);
    w=i;
    while (w!=source)
    {
        printf("<--%d",last[w]);w=last[w];
    }
    printf("\n Shortest path cost:%d \n",distance[i]);
}
}
```

## 13.4 Output

```
[Alans-MacBook-Air:NPLab alan$ gcc lsr.cpp
[Alans-MacBook-Air:NPLab alan$ ./a.out
Enter the no of routers : 4
Enter the cost matrix
cost_matrix[0][0] : 0
cost_matrix[0][1] : 2
cost_matrix[0][2] : 3
cost_matrix[0][3] : 4
cost_matrix[1][0] : 5
cost_matrix[1][1] : 0
cost_matrix[1][2] : 23
cost_matrix[1][3] : 2
cost_matrix[2][0] : 3
cost_matrix[2][1] : 4
cost_matrix[2][2] : 0
cost_matrix[2][3] : 3
cost_matrix[3][0] : 4
cost_matrix[3][1] : 6
cost_matrix[3][2] : 7
cost_matrix[3][3] : 0
Enter the source router:2

2==>0---Path taken:0
<--2
    Shortest path cost:3

2==>1---Path taken:1
<--2
    Shortest path cost:4

2==>2---Path taken:2

    Shortest path cost:0

2==>3---Path taken:3
<--2
    Shortest path cost:3
[Alans-MacBook-Air:NPLab alan$ █
```

## 13.5 Result

Implemented Link State routing algorithm in C++ and compiled using version 4.2.1 on macOS Mojave . The output was obtained and verified.



## 14 UDP in wireshark

### 14.1 Aim

Observing data transferred in client server communication using UDP using Wireshark and identifying the UDP datagram .

### 14.2 Theory

#### 14.2.1 Wireshark

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License. Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options.

#### 14.2.2 Getting Wireshark

You can download Wireshark for Windows or macOS from its official website. If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

#### 14.2.3 UDP Protocol

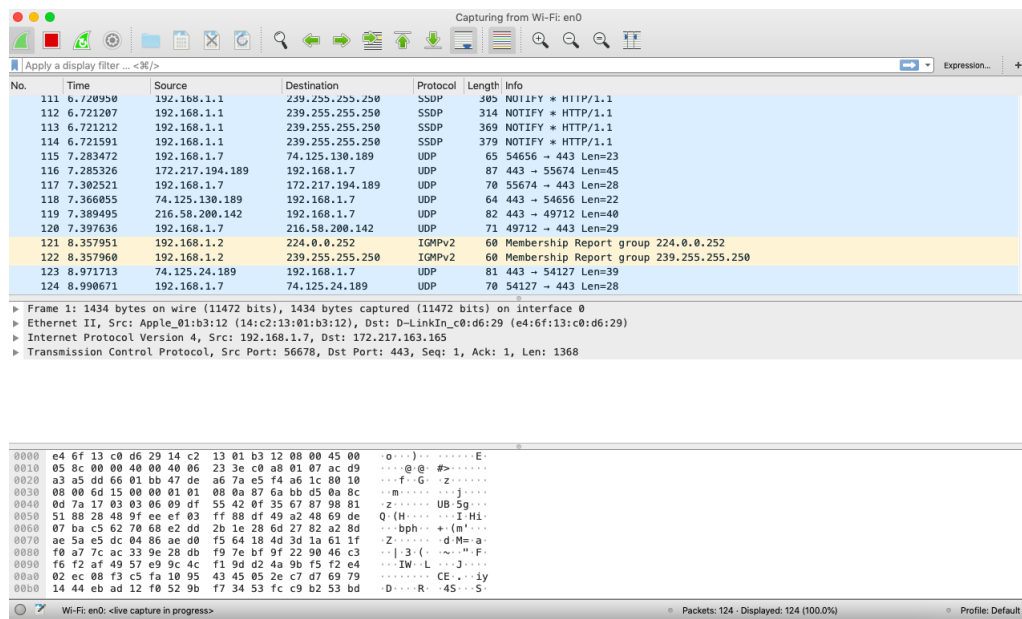
UDP stands for User Datagram Protocol. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out. When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet, it will just continue sending the next packets. There is no guarantee that all

the packets are delivered and there is no way to ask for a packet again if it misses out, but losing all this overhead means the computers can communicate more quickly.

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

### 14.2.4 Capturing packets

After downloading and installing Wireshark, you can launch it and double-click the name of a network interface under Capture to start capturing packets on that interface. For example, if you want to capture traffic on your wireless network, click your wireless interface.

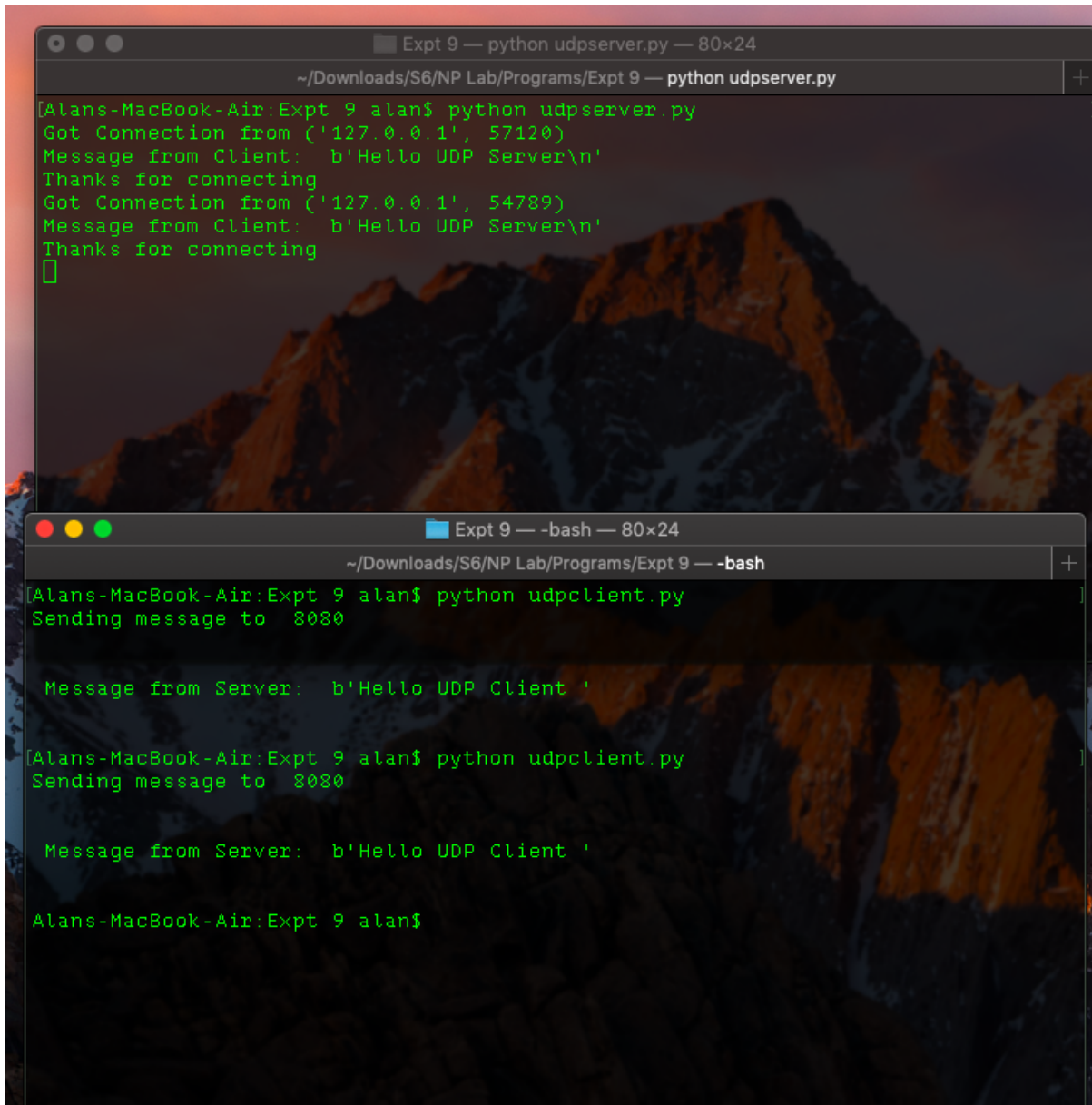


### 14.2.5 Filtering Packets

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type “dns” and you’ll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

Here we apply UDP to the filter and run a UDP server and client. Communication done between the server and client is captured by wireshark.

### 14.3 Output

The image shows two overlapping terminal windows on a macOS desktop. The top window, titled 'Expt 9 — python udpserver.py — 80x24', shows the execution of a Python UDP server script. It receives two connections from 127.0.0.1, each with a 'Hello UDP Server' message. The bottom window, titled 'Expt 9 — -bash — 80x24', shows the execution of a Python UDP client script. It sends two messages to port 8080, each receiving a 'Hello UDP Client' response from the server. The background of the desktop is a mountain landscape.

```
Expt 9 — python udpserver.py — 80x24
~/Downloads/S6/NP Lab/Programs/Expt 9 — python udpserver.py
[Alans-MacBook-Air:Expt 9 alan$ python udpserver.py
Got Connection from ('127.0.0.1', 57120)
Message from Client:  b'Hello UDP Server\n'
Thanks for connecting
Got Connection from ('127.0.0.1', 54789)
Message from Client:  b'Hello UDP Server\n'
Thanks for connecting
[]

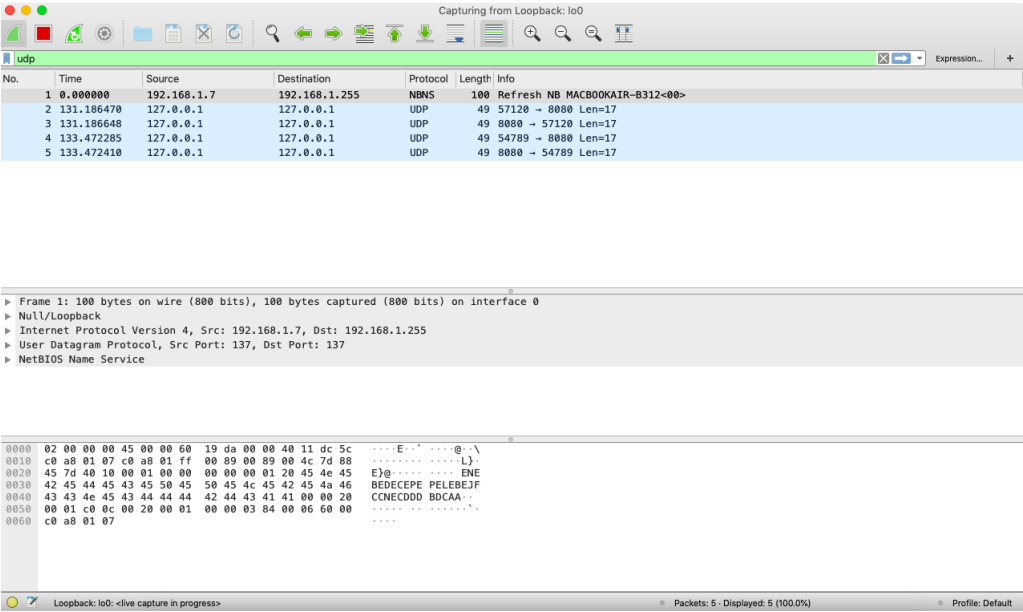
Expt 9 — -bash — 80x24
~/Downloads/S6/NP Lab/Programs/Expt 9 — -bash
[Alans-MacBook-Air:Expt 9 alan$ python udpclient.py
Sending message to 8080

Message from Server:  b'Hello UDP Client '

[Alans-MacBook-Air:Expt 9 alan$ python udpclient.py
Sending message to 8080

Message from Server:  b'Hello UDP Client '

Alans-MacBook-Air:Expt 9 alan$
```



14.4 Result

Wireshark was installed and Packet transfer using UDP was observed .

## **15 Three way hand shake connection termination**

### **15.1 Aim**

Using Wireshark observe Three Way Handshaking Connection Establishment, Data Transfer and Three Way Handshaking Connection Termination in client server communication using TCP.

### **15.2 Theory**

#### **15.2.1 Wireshark**

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License. Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options.

#### **15.2.2 Getting Wireshark**

You can download Wireshark for Windows or macOS from its official website. If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

#### **15.2.3 TCP Protocol**

Transmission Control Protocol(TCP) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data.

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

When loading a web page, the computer sends TCP packets to the web server's address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which the web browser stitches together to form the web page and display it to you. TCP enables two way communication — the remote system sends packets back to acknowledge it is received your packets.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors. Packets sent with TCP are tracked so no data is lost or corrupted in transit.

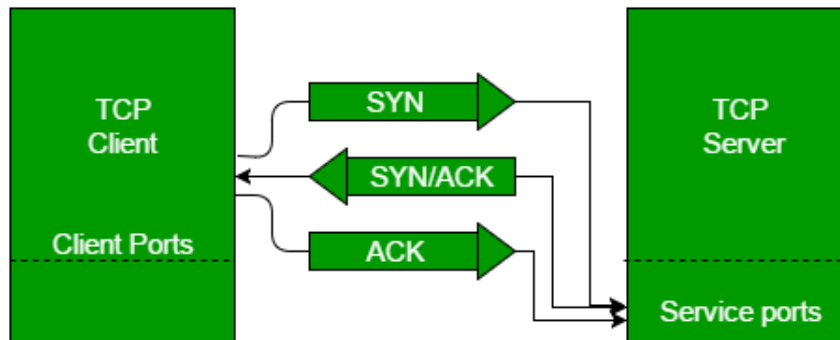
#### **15.2.4 Threeway handshake**

A three-way handshake is a method used in a TCP/IP network to create a connection between a local host/client and server. It is a three-step method that requires both the client and server to exchange SYN and ACK (acknowledgment) packets before actual data communication begins.

A three-way handshake is also known as a TCP handshake. A three-way handshake is primarily used to create a TCP socket connection. It works when:

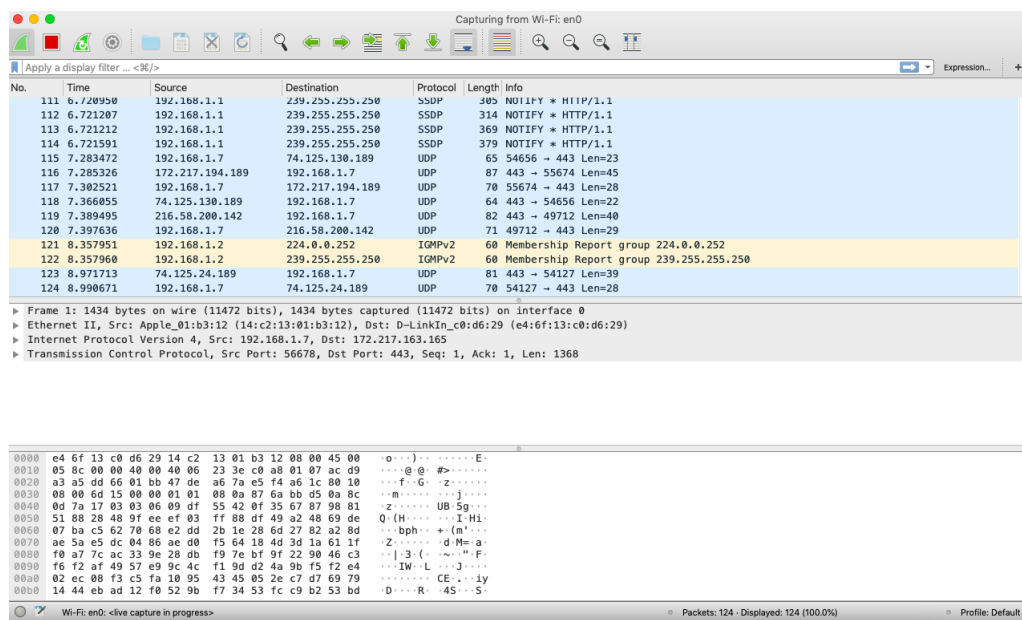
1. A client node sends a SYN data packet over an IP network to a server on the same or an external network. The objective of this packet is to ask/infer if the server is open for new connections.
2. The target server must have open ports that can accept and initiate new connections. When the server receives the SYN packet from the client node, it responds and returns a confirmation receipt – the ACK packet or SYN/ACK packet.
3. The client node receives the SYN/ACK from the server and responds with an ACK packet.

Upon completion of this process, the connection is created and the host and server can communicate.



### 15.2.5 Capturing packets

After downloading and installing Wireshark, you can launch it and double-click the name of a network interface under Capture to start capturing packets on that interface. For example, if you want to capture traffic on your wireless network, click your wireless interface.

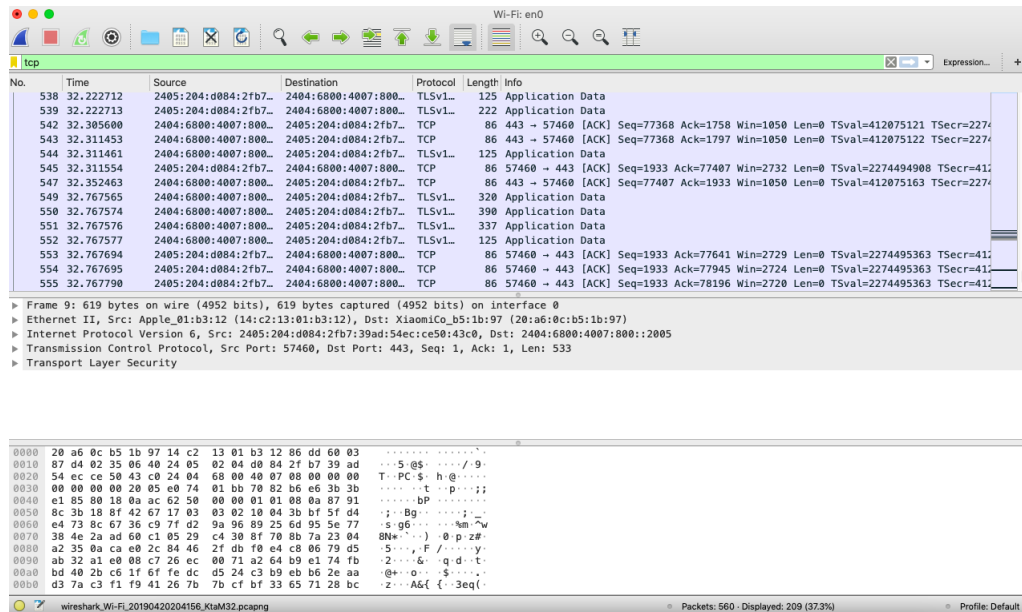


### 15.2.6 Filtering Packets

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type “dns”

and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

## 15.3 Output



## 15.4 Result

Wireshark was installed and three way handshaking protocol was observed.



## **16 Packet capturing and filtering application**

### **16.1 Aim**

Develop a packet capturing and filtering application using raw sockets .

### **16.2 Theory**

#### **16.2.1 Packet Capturing**

Packet Capture is a networking term for intercepting a data packet that is crossing a specific point in a data network. Once a packet is captured in real-time, it is stored for a period of time so that it can be analyzed, and then either archived or discarded. Packets are captured and examined to help diagnose and solve network problems such as:

1. Identifying security threats
2. Troubleshooting undesirable network behaviors
3. Identifying network congestion
4. Identifying data/packet loss
5. Forensic network analysis

Once a packet is captured, it is stored temporarily so that it can be analyzed. The packet is inspected to help diagnose and solve network problems and determine whether network security policies are being followed.

#### **16.2.2 Algorithm**

1. START
2. CREATE SOCKET
3. RECEIVE all packets
4. UNPACK the packets
5. FORMAT the packets
6. PRINT the filtered data

7.STOP

### 16.2.3 Program

```
import socket , sys
from struct import *

if (sys.argv[1]=="tcp"):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_RAW,
                           socket.IPPROTO_TCP)
    except socket.error ,msg:
        print 'Socket could not be created. Error Code : '
        + str(msg[0]) + ' Message ' + msg[1]
        sys.exit()
elif (sys.argv[1]=="udp"):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_RAW,
                           socket.IPPROTO_UDP)
    except socket.error , msg:
        print 'Socket could not be created. Error Code : '
        + str(msg[0]) + ' Message ' + msg[1]
        sys.exit()
else:
    print "Specify protocol"

# receive a packet
while True:
    packet = s.recvfrom(65565)

    packet = packet[0]

    ip_header = packet[0:20]

    iph = unpack( '!BBHHHBBH4s4s' , ip_header)

    version_ihl = iph[0]
    ihl = version_ihl & 0xF
```

```
iph_length = ihl * 4

s_addr = socket.inet_ntoa(iph[8]);
d_addr = socket.inet_ntoa(iph[9]);

print ' Source Address : ' + str(s_addr) + '
Destination Address : ' + str(d_addr)

tcp_header = packet[iph_length:iph_length+20]

tcph = unpack('!HLLBBHHH' , tcp_header)

source_port = tcph[0]
dest_port = tcph[1]
acknowledgement = tcph[3]
doff_reserved = tcph[4]
tcph_length = doff_reserved >> 4


print 'Source Port : ' + str(source_port) + ' Dest Port : ' +
str(dest_port) + ' Acknowledgement : ' + str(acknowledgement)

h_size = iph_length + tcph_length * 4
data_size = len(packet) - h_size

#get data from the packet
data = packet[h_size:]

print 'Data : ' + data
print
```

### 16.3 Output

 sniff.png

## 16.4 Result

Packet catching and filtering application was implemented using python and output was obtained.

## 17 Simple Mail Transfer Protocol

### 17.1 Aim

Implement Simple Mail Transfer Protocol

### 17.2 Theory

#### 17.2.1 SMTP

MTP is a push protocol and is used to send the mail whereas POP (post office protocol) or IMAP (internet message access protocol) are used to retrieve those mails at the receiver's side.

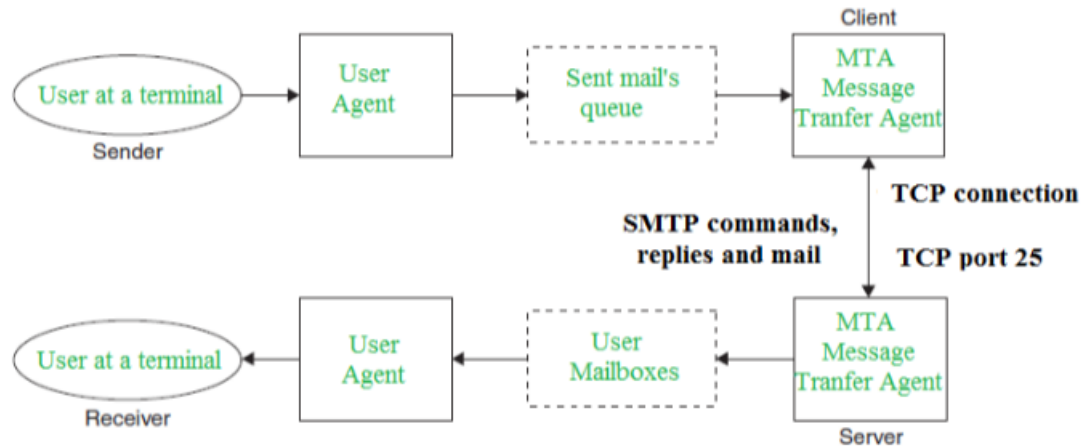
SMTP is an application layer protocol. The client who wants to send the mail opens a TCP connection to the SMTP server and then sends the mail across the connection. The SMTP server is always on listening mode. As soon as it listens for a TCP connection from any client, the SMTP process initiates a connection on that port (25). After successfully establishing the TCP connection the client process sends the mail instantly.

The SMTP model is of two type :

1. End-to- end method
2. Store-and- forward method

The end to end model is used to communicate between different organizations whereas the store and forward method is used within an organization. A SMTP client who wants to send the mail will contact the destination's host SMTP directly in order to send the mail to the destination. The SMTP server will keep the mail to itself until it is successfully copied to the receiver's SMTP.

The client SMTP is the one which initiates the session let us call it as client- SMTP and the server SMTP is the one which responds to the session request and let us call it as receiver-SMTP. The client- SMTP will start the session and the receiver-SMTP will respond to the request.



### 17.2.2 Working

1. SMTP server is always on a listening mode.
2. Client initiates a TCP connection with the SMTP server.
3. SMTP server listens for a connection and initiates a connection on that port.
4. The connection is established.
5. Client informs the SMTP server that it would like to send a mail.
6. Assuming the server is OK, client sends the mail to its mail server.
7. Client's mail server use DNS to get the IP Address of receiver's mail server.
8. Then, SMTP transfers the mail from sender's mail server to the receiver's mail server.

## 17.3 Program

### 17.3.1 SMTP Server

```
import socket
import datetime
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1",8080))
```

```
x=[" alananto@gmail.com", " hello@gmail.com", " example@gmail.com"]
s.listen(5)
conn, addr=s.accept()
while 1:
    mail=conn.recv(1024)
    tok=mail.split()
    print(mail)
    if tok[0]=='HELO':
        conn.send("250 mail.example.org")
    elif tok[0]=='MAILFROM: ':
        flag1=0
        for i in x: #checking if mail is in the list
            if tok[1]=="<"+i+">":
                conn.send("250 Sender "+tok[1]+" OK")
                flag1=1
                break
        if flag1==0:
            conn.send("421 Service Unavailable")
    elif tok[0]=='RCPTTO: ':
        flag2=0
        for i in x: #checking if mail is in the list
            if tok[1]=="<"+i+">":
                conn.send("250 Recipient "+tok[1]+" OK")
                flag2=1
                break
        if flag2==0:
            conn.send("421 Service Unavailable")
    elif tok[0]=='DATA':
        flag3=0
        if flag1==1 and flag2==1:
            flag3=1
            conn.send("354 Go Ahead, Enter data ending with
<CRLF>.<CRLF>")
            break
        else:
            conn.send("421 Service Unavailable")
```

```
if flag3==1:
    buff=conn.recv(2048)
    print("Message: "+buff)
    conn.send("250 OK; Message Accepted")
mail=conn.recv(1024)
tok=mail.split()
if tok[0]=='QUIT':
    conn.send("221 mail.example.org")
```

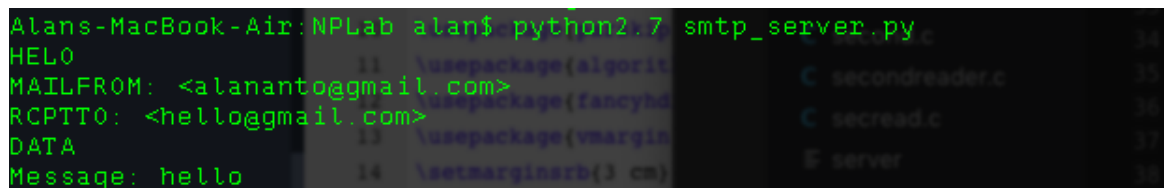
### 17.3.2 SMTP Client

```
import socket
import sys
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1",8080))
print("Waiting to connect...")
while 1:
    comm=raw_input()

    s.send(comm)
    print("Server: "+s.recv(1024))
    if comm=='QUIT':
        s.close()
        break
```

## 17.4 Output

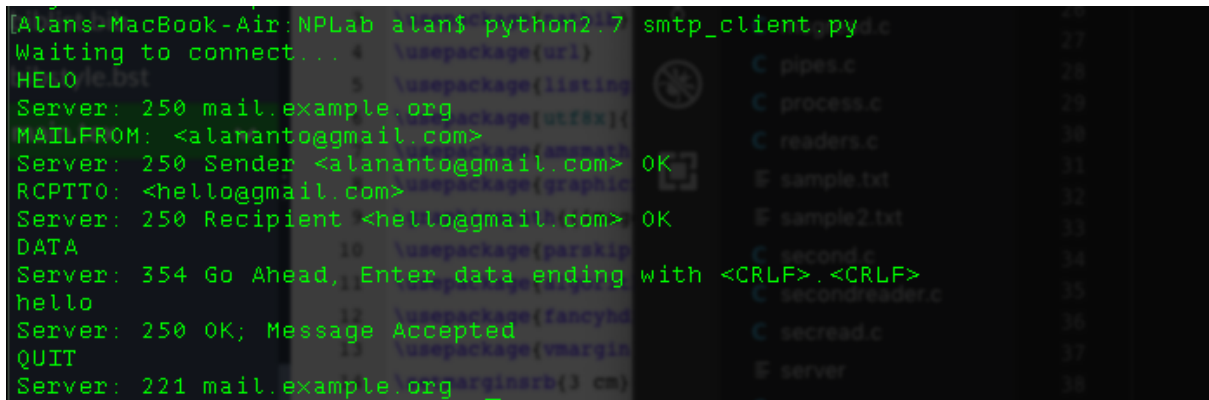
### 17.4.1 SMTP Server



```
Alans-MacBook-Air:NPLab alan$ python2.7 smtp_server.py
HELO
MAILFROM: <alananto@gmail.com>
RCPTTO: <hello@gmail.com>
DATA
Message: hello
```



### 17.4.2 SMTP Client



```
[Alans-MacBook-Air:NP Lab alan]$ python2.7 smtp_client.py d.c
Waiting to connect...
HELO le.bst
Server: 250 mail.example.org
MAILFROM: <alananto@gmail.com>
Server: 250 Sender <alananto@gmail.com> OK
RCPTTO: <hello@gmail.com>
Server: 250 Recipient <hello@gmail.com> OK
DATA
Server: 354 Go Ahead, Enter data ending with <CRLF>.<CRLF>
hello
Server: 250 OK; Message Accepted
QUIT
Server: 221 mail.example.org
```

### 17.5 Result

SMTP server and client was implemented using python and the required output was obtained.

## 18 Concurrent File Server

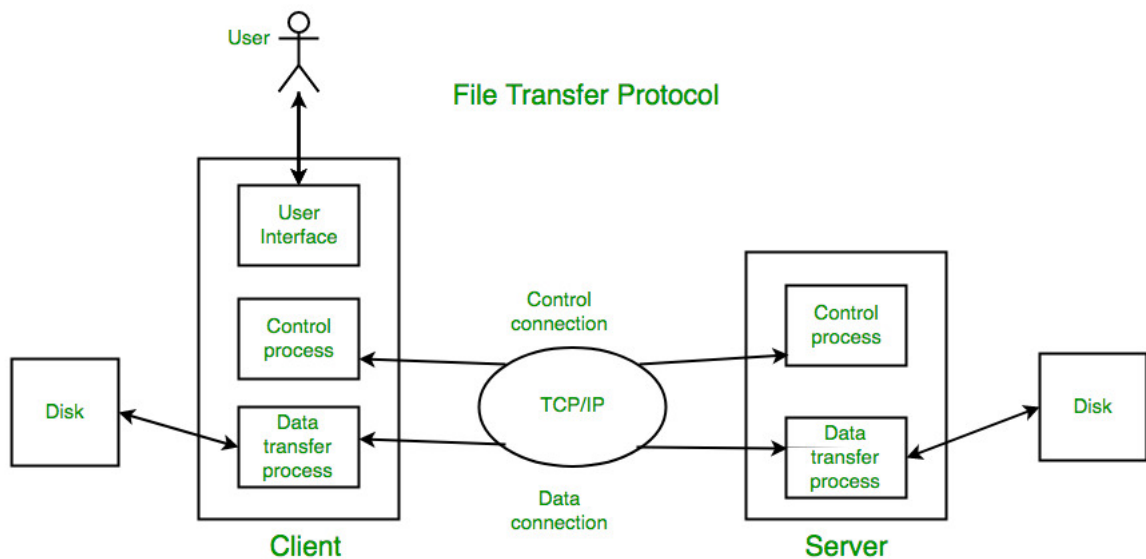
### 18.1 Aim

Implementing Concurrent server

### 18.2 Theory

#### 18.2.1 FTP

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture and uses separate control and data connections between the client and the server.[1] FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.



### 18.3 Program

#### 18.3.1 File Server

```
import socket

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

port=8080

s.bind(('', port))

s.listen(5)

while True:
    c, addr=s.accept()
    filename=c.recv(1024)
    print "Finding file : "+filename+" ....."
    print ('Got Connection from ', addr)
    try:
        file = open(filename, 'rb')
        c.send('Found')
        print "File Found"
        data = file.read(1024)
        print "Reading File ...."
        print "Sending ..."
        while(data):
            c.send(data)
            data = file.read(1024)
        file.close()
        print "File closed"
        break
    except:
        c.send('No File ')
        print "No file found"
        break
c.close()
```

### 18.3.2 File Client

```
import socket
import sys

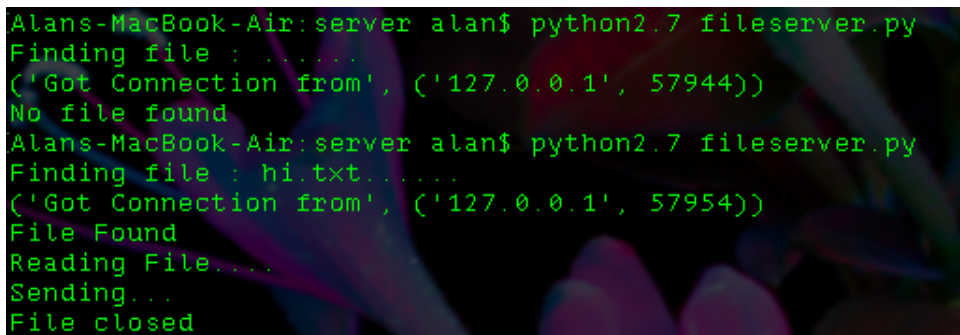
s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
port = 8080
s.connect(('localhost', port))
s.send(sys.argv[1])
response = s.recv(1024)
print response
if(response == 'Found'):
    file = open('recieve_'+ sys.argv[1], 'wb')
    print "Recieving File ....."
    while True:
        data = s.recv(1024)
        if not data:
            break
        file.write(data)
    file.close()
    print "File written at recieve_"+sys.argv[1]
else:
    print "File Not Found"

s.close()
```

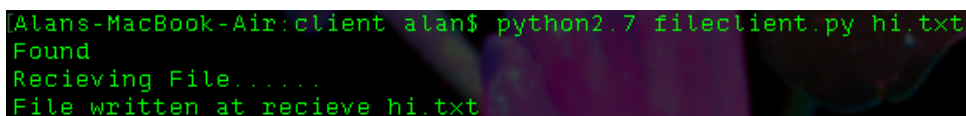
## 18.4 Output

### 18.4.1 File Server

A terminal window with a dark background and green text. The output shows the server script being executed twice. The first time, it says 'Finding file : .....', '("Got Connection from", ("127.0.0.1", 57944))', and 'No file found'. The second time, it says 'Finding file : hi.txt.....', '("Got Connection from", ("127.0.0.1", 57954))', 'File Found', 'Reading File....', 'Sending...', and 'File closed'.

```
Alans-MacBook-Air:server alan$ python2.7 fileserver.py
Finding file : .....
("Got Connection from", ("127.0.0.1", 57944))
No file found
Alans-MacBook-Air:server alan$ python2.7 fileserver.py
Finding file : hi.txt.....
("Got Connection from", ("127.0.0.1", 57954))
File Found
Reading File....
Sending...
File closed
```

### 18.4.2 File Client

A terminal window with a dark background and green text. The output shows the client script being executed, sending 'hi.txt' to the server. The output is 'Found', 'Recieving File.....', and 'File written at recieve hi.txt'.

```
Alans-MacBook-Air:client alan$ python2.7 fileclient.py hi.txt
Found
Recieving File.....
File written at recieve hi.txt
```

## 18.5 Result

File server and client was implemented using python and the required output was obtained.

## **19 Network with multiple subnets with wired and wireless LANs**

### **19.1 Aim**

Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure the following services in the network - TELNET, SSH, FTP server, Web server, File server, DHCP server and DNS server.

### **19.2 Theory**

#### **19.2.1 Network**

A computer network is a digital telecommunications network which allows nodes to share resources. In computer networks, computing devices exchange data with each other using connections (data links) between nodes. These data links are established over cable media such as wires or optic cables, or wireless media. Network computer devices that originate, route and terminate the data are called network nodes. Nodes are generally identified by network addresses, and can include hosts such as personal computers, phones, and servers, as well as networking hardware such as routers and switches. Two such devices can be said to be networked together when one device is able to exchange information with the other device, whether or not they have a direct connection to each other.

#### **19.2.2 Subnet**

A subnetwork or subnet is a logical subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting. Computers that belong to a subnet are addressed with an identical most-significant bitgroup in their IP addresses. This results in the logical division of an IP address into two fields, the network number or routing prefix and the rest field or host identifier. The rest field is an identifier for a specific host or network interface. A network may also be characterized by its subnet mask or netmask, which is the bitmask that when applied by a bitwise AND operation to any IP address in the network, yields the routing prefix.

## 19.3 Configuring the Services

The following shows how the different services can be configured in a linux pc :

### 19.3.1 Telnet

Telnet is a protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

Take the following steps to configure Telnet:

1. Install Telnet

```
sudo apt install telnet xinetd
```

2. Edit /etc/inetd.conf with root permission, add this line:

```
telnet stream tcp nowait telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

3. Edit /etc/xinetd.conf, copy the following configuration:

```
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/
defaults
{
# Please note that you need a log_type line to be able to use log_on_success
# and log_on_failure. The default is the following :
# log_type = SYSLOG daemon info
instances = 60
log_type = SYSLOG authpriv
log_on_success = HOST PID
log_on_failure = HOST
cps = 25 30
}
```

4. Change telnet port by using the following command in the terminal:

```
telnet 23/tcp
```

5. Then restart the service:

```
sudo /etc/init.d/xinetd restart
```

### 19.3.2 SSH

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line login and remote command execution, but any network service can be secured with SSH. SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server. Take the following steps to configure SSH:

- Install SSH:

```
sudo apt-get install openssh-server
```

(Installing the client can be done by replacing openssh-server by openssh-client)

- Configure SSH:

```
sudo nano /etc/ssh/sshd_config
```

Then make the changes you want to make.

- Restart SSH:

```
sudo systemctl restart ssh
```

We can login to the SSH server from an SSH client.

### 19.3.3 FTP Server

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture using separate control and data connections between the client and the server. FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

The following steps show setting up an FTP server on the computer:

- Install FTP daemon:

```
sudo apt install vsftpd
```

- Configuring FTP can be done by editing the following file:

```
/etc/vsftpd.conf
```



- Restart the service:

```
sudo systemctl restart vsftpd.service
```

### 19.3.4 Web Server

A web server is server software, or hardware dedicated to running said software, that can satisfy World Wide Web client requests. A web server can, in general, contain one or more websites. A web server processes incoming network requests over HTTP and several other related protocols.

A web server can be hosted on the localhost of the PC by following the following steps:

- Installing the server: The most common server on Linux systems and it is called the LAMP server. It can be installed on Ubuntu by:

```
sudo apt install lamp-server^
```

- Hosting a website: By creating a *.conf* file in the */etc/apache2/sites-available/* folder, we inform the server of the location of the code for our website.
- Enabling the website by using the command:

```
sudo a2ensite <nameOfFile.conf>
```

- By editing */etc/hosts* file, we can give the domain name for the website
- The configuration the server is in the file: */etc/apache2/apache2.conf*
- Restart the server by the command:

```
sudo systemctl restart apache2.service
```

### 19.3.5 File Server

A file server is a computer attached to a network that provides a location for shared disk access, i.e. shared storage of computer files (such as text, image, sound, video) that can be accessed by the workstations that are able to reach the computer that shares the access through a computer network.

The following steps can be followed to setup a file server:

- Installing Samba File Server:

```
sudo apt install samba
```

- Configuring the file server by editing */etc/samba/smb.conf*

First, edit the following key/value pairs in the [global] section of */etc/samba/smb.conf*:

```
workgroup = EXAMPLE
...
security = user
```

Create a new section at the bottom of the file, or uncomment one of the examples, for the directory to be shared:

```
[share]
comment = Ubuntu File Server Share
path = /srv/samba/share
browsable = yes
guest ok = yes
read only = no
create mask = 0755
```

- Make a directory for hosting files and setting permission for the directory:

```
sudo mkdir -p /srv/samba/share
sudo chown nobody:nogroup /srv/samba/share/
```

- Restart Samba service:

```
sudo systemctl restart smbd.service nmbd.service
```

### 19.3.6 DHCP Server

The Dynamic Host Configuration Protocol (DHCP) is a network management protocol used on UDP/IP networks whereby a DHCP server dynamically assigns an IP address and other network configuration parameters to each device on a network so they can communicate with other IP networks. A DHCP server enables computers to request IP addresses and networking parameters automatically from the Internet service provider (ISP), reducing the need for a network administrator or a user to manually assign IP addresses to all network devices.

The following steps shows how DHCP server can be run:

- Install DHCP server:

```
sudo apt-get install isc-dhcp-server
```

- Configure DHCP server, the config file is */etc/dhcp/dhcpd.conf*:

```
# Sample /etc/dhcpd.conf
# (add your comments here)
default-lease-time 600;
max-lease-time 7200;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.254;
option domain-name-servers 192.168.1.1, 192.168.1.2;
option domain-name "mydomain.example";

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.10 192.168.1.100;
    range 192.168.1.150 192.168.1.200;
}
```

- Starting and stopping services can be achieved using:

```
sudo service isc-dhcp-server restart
sudo service isc-dhcp-server start
sudo service isc-dhcp-server stop
```

After editing configuration files, we have to restart the service.

### 19.3.7 DNS Server

A name server is a computer application that implements a network service for providing responses to queries against a directory service. It translates an often humanly meaningful, text-based identifier to a system-internal, often numeric identification or addressing component. This service is performed by the server in response to a service protocol request. An example of a name server is the server component of the Domain Name System (DNS).

The Domain Name System (DNS) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. Most prominently, it translates more readily memorized domain names to the numerical IP addresses needed for locating and identifying computer services and devices with the underlying network protocols.

The following steps show the setup:

- Installing:

```
sudo apt install bind9
```

- The configuration is in the */etc/bind* folder
- Setting as a catching name server by editing the file */etc/bind/named.conf.options*:

```
forwarders {  
    1.2.3.4; # replace with the ip address  
    5.6.7.8; # of the name servers  
};
```

- BIND9 can be configured with the primary and the secondary master as a custom DNS server to access all the subnets.
- Restarting bind9:

```
sudo systemctl restart bind9.service
```

## 19.4 Result

For accessing the different nodes in the subnet, TELNET, SSH, FTP server, Web server, File server, DHCP server and DNS server have been configured and runs successfully in an Ubuntu 18.04 PC.