**Experiment No:1**

# NETWORK CONFIGURATION FILES AND NETWORKING COMMANDS

## NETWORK CONFIGURATION FILES

1. The **/etc/hosts** file contains the Internet Protocol (IP) host names and addresses for the local host and other hosts in the Internet network. This file is used to resolve a name into an address (that is, to translate a host name into its Internet address).

2. **resolv.conf** is usually located in the directory /etc of the file system. The file is either maintained manually, or when DHCP is used, it is usually updated with the utility resolvconf. In systemd based Linux distributions using systemd-resolved.

3. The **/etc/sysconfig/network** file is a global (across all network cards) configuration file. It allows us to define whether we want networking (NETWORKING=yes|no), what the hostname should be (HOSTNAME=) and which gateway to use (GATEWAY=).

4.The **/etc/nsswitch.conf** file defines the search order of the network databases. The Solaris
installation program creates a default /etc/switch.conf file for the local machine, based on the
name service you indicate during the installation process.

## NETWORKING COMMANDS

**ifconfig** stands for "interface configuration." It is used to view and change the configuration of the network interfaces on your system. Displays information about all network interfaces currently in operation.

**netstat :** Displays active TCP connections, ports on which the computer is listening, Ethernet
statistics, the IP routing table, IPv4 statistics (for the IP, ICMP, TCP, and UDP protocols), and IPv6 statistics (for the IPv6, ICMPv6, TCP over

IPv6, and UDP over IPv6 protocols). Used without parameters, this command displays active TCP connections.

**Ping** is a command-line utility, available on virtually any operating system with network
connectivity, that acts as a test to see if a networked device is reachable. The ping command sends a request over the network to a specific device. A successful ping results in a response from the computer that was pinged back to the originating computer.

**Experiment No:2**

## SYSTEM CALLS

**1. ps :**

Linux provides us a utility called ps for viewing information related with the processes on a
system which stands as abbreviation for “Process Status”. ps command is used to list the currently
running processes and their PIDs along with some other information depends on different options.

**2. fork()**

- Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process).

- After a new child process is created, both processes will execute the next instruction following the fork() system call.
- A child process uses the same pc(program counter), CPU registers, open files used by parent process.
- It takes no parameters and returns an integer valu
  - Negative: Creation of a child process was unsuccessful.
  - Zero: Returned to the newly created child process.
  - Positive: Returned to the parent process(caller process). The value contains Process ID (PID) of newly created child process.

**3. exec()**

The exec() family of functions replaces the current process image with a new process
image. It loads the program into the current process space and runs it from the entry point.
The exec() family consists of following function:
int execl()
int execlp()
int execle()
int execv()
int execvp()
int execvpe()

**4. wait()**

A call to wait() blocks the calling process until one of its child processes exits or a signal is
received. After child process terminates, parent continues its execution after wait() system call
instruction.

**5. exit()**

Syntax: void exit(int status) function call terminates the process normally.
exit() performs following operations.

- Flushes unwritten buffered data.
- Closes all open files.
- Removes temporary files.
- Returns an integer exit status to the operating system.

**6. open()**
Used to open the file for reading, writing or both.
Syntax: int open (const char* path, int flags [, int mode ]);
Parameters:

- path : path to file which you want to use
- flags : How you like to use
  - O_RDONLY: read only
  - O_WRONLY: write only
  - O_RDWR: read and write
  - O_CREAT: create file if it doesn't exist
  - O_EXCL: prevent creation if it already exists

**7. read()**
Read from the file indicated by the file descriptor fd, the cnt bytes of input into the memory area
indicated by buf. A successful read() updates the access time for the file.
Syntax:size_t read (int fd, void* buf, size_t cnt);
Parameters:

fd: file descripter
buf: buffer to read data from
cnt: length of buffer

Return value is the number of bytes read, returns 0 on reaching end of file, -1 on error/signal
interrupt

**8. write()**
Writes cnt bytes from buf to the file associated with fd. cnt should not be greater than INT_MAX
(defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any
other action.
Syntax: size_t write (int fd, void* buf, size_t cnt);
Parameters:

fd: file descripter
buf: buffer to write data to
cnt: length of buffer

Returns number bytes actually written on success, 0 on reaching end of file, -1 on error/signal
interrupt

**Creating a socket:**
socket() creates an endpoint for communication and returns a descriptor.
Syntax: int socket(int domain,int type,int protocol)

**Experiment No:3**

## CLIENT SERVER COMMUNICATION USING SOCKET PROGRAMMING AND TCP

TCP (Transmission Control Protocol) is one of the main protocols of the Internet protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between the different devices over a network. TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and

congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination

**PROGRAM**

**server**

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<netinet/in.h>
#include<arpa/inet.h>
void main()
{
	struct sockaddr_in server,client;
	int s,n,sock;
	char b1[100]=" ",b2[100]="";
	s=socket(AF_INET,SOCK_STREAM,0);
	server.sin_family=AF_INET;
	server.sin_port=2000;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	bind(s,(struct sockaddr *)&server,sizeof server);
	listen(s,1);
	printf("\nserver ready, waiting for client..\n");
	n=sizeof client;
	sock=accept(s,(struct sockaddr *)&client,&n);
	for(;;)
	{
		recv(sock,b1,sizeof b1,0);
		if(strcmp(b1,"end")==0)
			break;
		printf("\nClient %s",b1);
		printf("\nServer : ");
		gets(b2);
		send(sock,b2,sizeof b2,0);
		if(strcmp(b2,"end")==0)
			break;
	}
	close(sock);
```

```
	close(s);
}
```

**client**

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<netinet/in.h>
#include<arpa/inet.h>
void main()
{
	struct sockaddr_in server,client;
	int s,sock;
	char b1[100]=" ",b2[100]="cs";
	s=socket(AF_INET,SOCK_STREAM,0);
	server.sin_family=AF_INET;
	server.sin_port=2000;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	printf("\nClient ready....\n");
	connect(s,(struct sockaddr *)&server,sizeof server);
	for(;;)
	{
		printf("\nClient : ");
		gets(b2);
		send(s,b2,sizeof b2,0);
		if(strcmp(b2,"end")==0)
			break;
		recv(s,b1,sizeof b1,0);
		if(strcmp(b1,"end")==0)
			break;
		printf("\nServer %s",b1);
	}
	close(s);
}
```

**OUTPUT**



**Experiment No:4**

## CLIENT SERVER COMMUNICATION USING SOCKET PROGRAMMING AND UDP

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection prior to data transfer.

**PROGRAM**

**server**

```
#include<stdio.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
void main()
{
	struct sockaddr_in client,server;
	int s,n,num,fact=1,i;
	s=socket(AF_INET,SOCK_DGRAM,0);
	server.sin_family=AF_INET;
	server.sin_port=3000;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	bind(s,(struct sockaddr *)&server,sizeof(server));
```

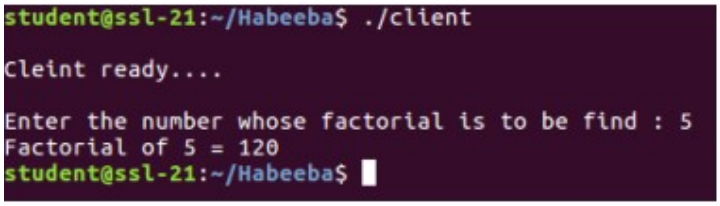
```
	printf("\nServer ready,waiting for client...\n");
	n=sizeof(client);
	recvfrom(s,&num,sizeof(int),0,(struct sockaddr *) &client,&n);
	if(num==0)
		fact=1;
	else
	{
		for(i=1;i<=num;i++)
			fact*=i;
	}
	sendto(s,&fact,sizeof(int),0,(struct sockaddr *) &client,n);
	printf("Factorial is calculated\n");
	close(s);
}
```

**client**

```
#include<stdio.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
void main()
{
	struct sockaddr_in client,server;
	int s,n,num,fact;
	s=socket(AF_INET,SOCK_DGRAM,0);
	server.sin_family=AF_INET;
	server.sin_port=3000;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	printf("\nCleint ready....\n");
	n=sizeof(server);
	printf("\nEnter the number whose factorial is to be find : ");
	scanf("%d",&num);
	sendto(s,&num,sizeof(int),0,(struct sockaddr *) &server,n);
	recvfrom(s,&fact,sizeof(int),0,NULL,NULL);
	printf("Factorial of %d = %d\n",num,fact);
	close(s);
```

}

## OUTPUT


## Experiment No:5

## SLIDING WINDOW FLOW CONTROL PROTOCOLS

### STOP-AND-WAIT PROTOCOL

Stop-and-wait Protocol is a flow control protocol used in the data link layer for transmission of data in noiseless channels. Sender keeps on sending messages to the Receiver. In order to prevent the receiver from overwhelming, there is a need to tell the sender to slow down the transmission of frames. We can make use of feedback from the receiver to the sender. Frames 0 sends to receiver, ACK 1 will be sent back to sender; frame 1 goes to receiver, ACK 0 will be back to sender, and so on.

### PROGRAM

**client**

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
void main(){
	struct sockaddr_in client,server;
	int s,n,sock,i,frame[100],temp;
	s=socket(AF_INET, SOCK_STREAM, 0);
	server.sin_family = AF_INET;
	server.sin_port=2022;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	printf("Client is ready\n");
	connect(s,(struct sockaddr *)&server, sizeof server);
	printf("\nClient\n");
	printf("Enter the number of frames: ");
	scanf("%d",&n);
	for(i=0;i<n+1;i++){
		frame[i]=i;
	}
	i=0;
	send(s,&n,sizeof(int),0);
```

```
	while(1)
	{
		printf("Frame %d sent\n",frame[i]);
		send(s,&frame[i],sizeof(int),0);
		recv(s,&temp,sizeof(int),0);
		if(temp==n)
		{
			printf("Acknowledgement %d Received Successfully\
n",temp);
			break;
		}
		if(temp==frame[i+1]){
			i++;
			printf("Acknowledgement %d Received Successfully\
n",temp);
		}
		else
		{
			sleep(5);
			printf("Timeout\n");
		}
	}
	close(s);
}
```

**server**

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include<math.h>
int main()
{
	struct sockaddr_in client,server;
	int s,sock,ack,frame,n,num;
	s=socket(AF_INET, SOCK_STREAM, 0);
```

```
        server.sin_family = AF_INET;
        server.sin_port=2022;
        server.sin_addr.s_addr=inet_addr("127.0.0.1");
        bind(s,(struct sockaddr *)&server,sizeof server);
        listen(s,1);
        printf("Server is ready\nWaiting for client...\n");
        n=sizeof client;
        sock=accept(s,(struct sockaddr *)&client,&n);
        recv(sock,&num,sizeof(int),0);
        int temp=rand()%num-1;
        printf("%d\n",temp);
        while(1)
        {
                recv(sock,&frame,sizeof(int),0);
                printf("Frame %d Received Successfully\n",frame);
                ack=frame+1;
                if(frame==temp)
                {
                        ack=100;
                        temp--;
                }
                printf("Acknowledgement %d Send\n",ack);
                send(sock,&ack,sizeof(int),0);
                if(ack==num)
                        break;
        }
        close(sock);
        close(s);
}
```

**OUTPUT**

**sender**

```
student@ssl-21:~$ ./a.out
Client is ready

Client
Enter the number of frames: 3
Frame 0 sent
Timeout
Frame 0 sent
Acknowledgement 1 Received Successfully
Frame 1 sent
Acknowledgement 2 Received Successfully
Frame 2 sent
Acknowledgement 3 Received Successfully
```

**receiver**

```
student@ssl-21:~$ ./a.out
Server is ready
Waiting for client...
0
Frame 0 Received Successfully
Acknowledgement 100 Send
Frame 0 Received Successfully
Acknowledgement 1 Send
Frame 1 Received Successfully
Acknowledgement 2 Send
Frame 2 Received Successfully
Acknowledgement 3 Send
```

**GO-BACK-N ARQ**

Go-Back-N ARQ is mainly a specific instance of Automatic Repeat Request (ARQ)
protocol where the sending process continues to send a number of frames as specified by
the window size even without receiving an acknowledgement (ACK) packet from the
receiver. The sender keeps a copy of each frame until the acknowledgement arrives.
This protocol is a practical approach to the sliding window.

- In Go-Back-N ARQ, the size of the sender window is N and the size of the receiver
  window is always 1.
- This protocol makes the use of cumulative acknowledgements means here the
  receiver maintains an acknowledgement timer.
- If the receiver receives a corrupted frame, then it silently discards that corrupted
  frame and the correct frame is retransmitted by the sender after the timeout timer
  expires.
- In case if the receiver receives the out of order frame then it simply discards all the
  frames.
- In case if the sender does not receive any acknowledgement then the frames in the
  entire window will be retransmitted again.

**PROGRAM**

**server**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<sys/time.h>
#include<netinet/in.h>
```

```
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
int main() {
	int s_sock, c_sock;
	s_sock = socket(AF_INET, SOCK_STREAM, 0);
	struct sockaddr_in server, other;
	memset(&server, 0, sizeof(server));
	memset(&other, 0, sizeof(other));
	server.sin_family = AF_INET;
	server.sin_port = htons(9009);
	server.sin_addr.s_addr = INADDR_ANY;
	socklen_t add;
	if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
		printf("Binding failed\n");
		return 0;
	}
	printf("\tServer Up\n Go back n (n=3) used to send 10 messages \n\
n");
	listen(s_sock, 10);
	add = sizeof(other);c_sock = accept(s_sock, (struct
sockaddr*)&other, &add);
	time_t t1,t2;
	char msg[50]="server message :";
	char buff[50];
	int flag=0;
	fd_set set1,set2,set3;
	struct timeval timeout1,timeout2,timeout3;
	int rv1,rv2,rv3;
	int i=-1;
	qq:
	i=i+1;
```

```
bzero(buff,sizeof(buff));
char buff2[60];
bzero(buff2,sizeof(buff2));
strcpy(buff2,"server message :");
buff2[strlen(buff2)]=i+'0';
buff2[strlen(buff2)]='\0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
usleep(1000);
i=i+1;
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));i=i+1;
usleep(1000);
qqq:
bzero(buff2,sizeof(buff2));
strcpy(buff2,msg);
buff2[strlen(msg)]=i+'0';
printf("Message sent to client :%s \n",buff2);
write(c_sock, buff2, sizeof(buff2));
FD_ZERO(&set1);
FD_SET(c_sock, &set1);
timeout1.tv_sec = 2;
timeout1.tv_usec = 0;
rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
if(rv1 == -1)
        perror("select error ");
else if(rv1 == 0){
        printf("Going back from %d:timeout \n",i);
        i=i-3;
        goto qq;
```

```
}
else{
        read(c_sock, buff, sizeof(buff));
        printf("Message from Client: %s\n", buff);
        i++;
        if(i<=9)
                goto qqq;
}
qq2:
FD_ZERO(&set2);
FD_SET(c_sock, &set2);
timeout2.tv_sec = 3;
timeout2.tv_usec = 0;
rv2 = select(c_sock + 1, &set2, NULL, NULL, &timeout2);
if(rv2 == -1)
        perror("select error "); // an error accured
else if(rv2 == 0){
        printf("Going back from %d:timeout on last 2\n",i-1);
        i=i-2;
        bzero(buff2,sizeof(buff2));
        strcpy(buff2,msg);
        buff2[strlen(buff2)]=i+'0';
        write(c_sock, buff2, sizeof(buff2));
        usleep(1000);
        bzero(buff2,sizeof(buff2));
        i++;
        strcpy(buff2,msg);
        buff2[strlen(buff2)]=i+'0';
        write(c_sock, buff2, sizeof(buff2));
        goto qq2;
} // a timeout occured
else{
        read(c_sock, buff, sizeof(buff));
```

```
            printf("Message from Client: %s\n", buff);
            bzero(buff,sizeof(buff));read(c_sock, buff, sizeof(buff));
            printf("Message from Client: %s\n", buff);
        }
        close(c_sock);
        close(s_sock);
        return 0;
}
```

**client**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/time.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
int main() {
        int c_sock,flag=1,flg=1;
        char msg1[50]="acknowledgement of :",msg2[50],buff[100];
        c_sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in client;
        memset(&client, 0, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(9009);
        client.sin_addr.s_addr = inet_addr("127.0.0.1");
        if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1){
            printf("Connection failed");
            return 0;
        }
```

```
	printf("\n\tClient -with individual acknowledgement scheme\n\n");
	for(int i=0;i<=9;i++) {
		flg=1;
		bzero(buff,sizeof(buff));
		bzero(msg2,sizeof(msg2));
		if(i==8&&flag==1){
			printf("here\n"); //simulating loss
			flag=0;
			read(c_sock,buff,sizeof(buff));
		}
		int n = read(c_sock, buff, sizeof(buff));
		if(buff[strlen(buff)-1]!=i+'0')
		{ //out of order
			printf("Discarded as out of order \n");
			i--;
		}
		else
		{
			printf("Message received from server : %s \t %d\
n",buff,i);
			printf("Acknowledgement sent for message \n");
			strcpy(msg2,msg1);
			msg2[strlen(msg2)]=i+'0';
			write(c_sock,msg2, sizeof(msg2));
		}
	}
	close(c_sock);
	return 0;
}
```

**OUTPUT**

**server**

```
student@ssl-21:~/Habeeba$ gcc gbns.c
student@ssl-21:~/Habeeba$ ./a.out
        Server Up
 Go back n (n=3) used to send 10 messages

Message sent to client :server message :0
Message sent to client :server message :1
Message sent to client :server message :2
Message from Client: acknowledgement of :0
Message sent to client :server message :3
Message from Client: acknowledgement of :1
Message sent to client :server message :4
Message from Client: acknowledgement of :2
Message sent to client :server message :5
Message from Client: acknowledgement of :3
Message sent to client :server message :6
Going back from 6:timeout
Message sent to client :server message :4
Message sent to client :server message :5
Message sent to client :server message :6
Message from Client: acknowledgement of :4
Message sent to client :server message :7
Message from Client: acknowledgement of :5
Message sent to client :server message :8
Message from Client: acknowledgement of :6
Message sent to client :server message :9
Message from Client: acknowledgement of :7
Going back from 9:timeout on last 2
Message from Client: acknowledgement of :8
Message from Client: acknowledgement of :9
student@ssl-21:~/Habeeba$
```

**client**

```
student@ssl-21:~/Habeeba$ gcc gbnc.c
student@ssl-21:~/Habeeba$ ./a.out

        Client -with individual acknowledgement scheme

Message received from server : server message :0         0
Acknowledgement sent for message
Message received from server : server message :1         1
Acknowledgement sent for message
Message received from server : server message :2         2
Acknowledgement sent for message
Message received from server : server message :3         3
Acknowledgement sent for message
Discarded as out of order
Discarded as out of order
Message received from server : server message :4         4
Acknowledgement sent for message
Message received from server : server message :5         5
Acknowledgement sent for message
Message received from server : server message :6         6
Acknowledgement sent for message
Message received from server : server message :7         7
Acknowledgement sent for message
here
Discarded as out of order
Message received from server : server message :8         8
Acknowledgement sent for message
Message received from server : server message :9         9
Acknowledgement sent for message
student@ssl-21:~/Habeeba$
```

## SELECTIVE REPEAT ARQ

Selective repeat protocol, also called Selective Repeat ARQ (Automatic Repeat reQuest), is a data link layer protocol that uses sliding window method for reliable delivery of data frames. Here, only the erroneous or lost frames are retransmitted, while the good frames are received and buffered. It uses two windows of equal size: a sending window that stores the frames to be sent and a receiving window that stores the frames receive by the receiver. The size is half the maximum sequence number of the frame.

## PROGRAM

**server**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<sys/time.h>
#include<netinet/in.h>
#include<string.h>
```

```
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
void rsendd(int ch,int c_sock){
	char buff2[60];
	bzero(buff2,sizeof(buff2));
	strcpy(buff2,"reserver message :");
	buff2[strlen(buff2)]=(ch)+'0';
	buff2[strlen(buff2)]='\0';
	printf("Resending Message to client :%s \n",buff2);
	write(c_sock, buff2, sizeof(buff2));
	usleep(1000);
}
int main() {
	int s_sock, c_sock;
	int rv1,rv2,rv3,tot=0,flag=0,ok[20];
	char msg[50]="server message :";
	char buff[50];
	struct timeval timeout1,timeout2,timeout3;
	s_sock = socket(AF_INET, SOCK_STREAM, 0);
	struct sockaddr_in server, other;
	memset(&server, 0, sizeof(server));memset(&other, 0,
sizeof(other));
	server.sin_family = AF_INET;
	server.sin_port = htons(9009);
	server.sin_addr.s_addr = INADDR_ANY;
	socklen_t add;
	if(bind(s_sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
		printf("Binding failed\n");
		return 0;
	}
	printf("\tServer Up\n Selective repeat scheme\n\n");
	listen(s_sock, 10);
```

```
	add = sizeof(other);
	c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
	time_t t1,t2;
	fd_set set1,set2,set3;
	memset(ok,0,sizeof(ok));
	while(tot<9){
		int toti=tot;
		for(int j=(0+toti);j<(3+toti);j++){
			bzero(buff,sizeof(buff));
			char buff2[60];
			bzero(buff2,sizeof(buff2));
			strcpy(buff2,"server message :");
			buff2[strlen(buff2)]=(j)+'0';
			buff2[strlen(buff2)]='\0';
			printf("Message sent to client :%s \t%d\t%d\
n",buff2,tot,j);
			write(c_sock, buff2, sizeof(buff2));
			usleep(1000);
		}
		for(int k=0+toti;k<(toti+3);k++){
			qq:
			FD_ZERO(&set1);
			FD_SET(c_sock, &set1);
			timeout1.tv_sec = 2;
			timeout1.tv_usec = 0;
			rv1 = select(c_sock + 1, &set1, NULL, NULL,
&timeout1);
			if(rv1 == -1)
				perror("select error ");
			else if(rv1 == 0){
				printf("Timeout for message :%d \n",k);
				rsendd(k,c_sock);
				goto qq;
```

```
                } // a timeout occured
                else{
                        read(c_sock, buff, sizeof(buff));
                        printf("Message from Client: %s\n", buff);
                        if(buff[0]=='n'){
                                printf(" corrupt message acknowledgement
(msg %d) \
        n",buff[strlen(buff)-1]-'0');
                                rsendd((buff[strlen(buff)-1]-'0'),c_sock);
                        goto qq;
                }
                else
                        tot++;
                }
            }
        }
        close(c_sock);
        close(s_sock);
        return 0;
}
```

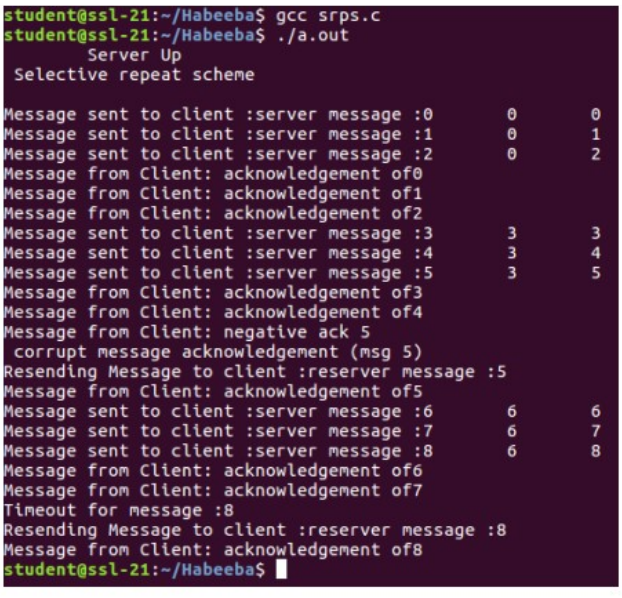
**client**

```
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/time.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
```

```
int isfaulty(){ //simulating corruption of message
	int d=rand()%4;
	return (d>2);
}
int main() {
	srand(time(0));
	int c_sock;
	c_sock = socket(AF_INET, SOCK_STREAM, 0);
	struct sockaddr_in client;
	memset(&client, 0, sizeof(client));
	client.sin_family = AF_INET;
	client.sin_port = htons(9009);
	client.sin_addr.s_addr = inet_addr("127.0.0.1");
	if(connect(c_sock, (struct sockaddr*)&client, sizeof(client)) == -1)
{
		printf("Connection failed");
		return 0;
	}
	printf("\n\tClient -with individual acknowledgement scheme\n\n");
	char msg1[50]="acknowledgement of";
	char msg3[50]="negative ack ";
	char msg2[50];
	char buff[100];
	int count=-1,flag=1;
	while(count<8){
		bzero(buff,sizeof(buff));
		bzero(msg2,sizeof(msg2));
		if(count==7&&flag==1){
			printf("here\n"); //simulate loss
			flag=0;
			read(c_sock,buff,sizeof(buff));
			continue;
		}
```

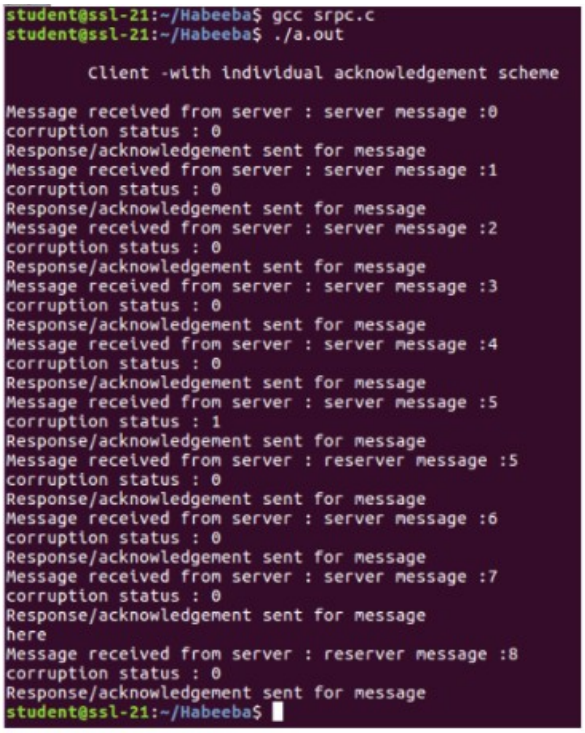
```
            int n = read(c_sock, buff, sizeof(buff));
            char i=buff[strlen(buff)-1];
            printf("Message received from server : %s \n",buff);
            int isfault=isfaulty();
            printf("corruption status : %d \n",isfault);
            printf("Response/acknowledgement sent for message \n");
            if(isfault){
                  strcpy(msg2,msg3);
            }
            else{
                  strcpy(msg2,msg1);
                  count++;
            }
            msg2[strlen(msg2)]=i;
            write(c_sock,msg2, sizeof(msg2));
      }
}
```

**OUTPUT**



```
student@ssl-21:~/Habeeba$ gcc srps.c
student@ssl-21:~/Habeeba$ ./a.out
        Server Up
 Selective repeat scheme

Message sent to client :server message :0        0       0
Message sent to client :server message :1        0       1
Message sent to client :server message :2        0       2
Message from Client: acknowledgement of0
Message from Client: acknowledgement of1
Message from Client: acknowledgement of2
Message sent to client :server message :3        3       3
Message sent to client :server message :4        3       4
Message sent to client :server message :5        3       5
Message from Client: acknowledgement of3
Message from Client: acknowledgement of4
Message from Client: negative ack 5
 corrupt message acknowledgement (msg 5)
Resending Message to client :reserver message :5
Message from Client: acknowledgement of5
Message sent to client :server message :6        6       6
Message sent to client :server message :7        6       7
Message sent to client :server message :8        6       8
Message from Client: acknowledgement of6
Message from Client: acknowledgement of7
Timeout for message :8
Resending Message to client :reserver message :8
Message from Client: acknowledgement of8
student@ssl-21:~/Habeeba$
```



```
student@ssl-21:~/Habeeba$ gcc srpc.c
student@ssl-21:~/Habeeba$ ./a.out

        Client -with individual acknowledgement scheme

Message received from server : server message :0
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :1
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :2
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :3
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :4
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :5
corruption status : 1
Response/acknowledgement sent for message
Message received from server : reserver message :5
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :6
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :7
corruption status : 0
Response/acknowledgement sent for message
here
Message received from server : reserver message :8
corruption status : 0
Response/acknowledgement sent for message
student@ssl-21:~/Habeeba$
```

**Experiment No:6**

## DISTANCE VECTOR ROUTING

Distance vector routing algorithms operate by having each router maintain a table (i.e, a vector)
giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors. The distance vector routing algorithm is sometimes called by Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

**PROGRAM**

```
#include<stdio.h>
struct node{
int dist[20];
int from[20];
}rt[10];
int main(){
	int costmatrix[20][20];
	int nodes,i,j,k,count=0;
	printf("\nEnter the number of nodes : ");
	scanf("%d",&nodes);
	printf("\nEnter the cost matrix :\n");
	for(i=0;i<nodes;i++){
		for(j=0;j<nodes;j++){
			scanf("%d",&costmatrix[i][j]);
			costmatrix[i][i]=0;
			rt[i].dist[j]=costmatrix[i][j];//initialize the distance equal
to cost matrix
```

```
                rt[i].from[j]=j; // initialize the source node
            }
        }
        do{
            count=0;
            for(i=0;i<nodes;i++){
                for(j=0;j<nodes;j++){
                    for(k=0;k<nodes;k++){
                        if(rt[i].dist[j]>costmatrix[i][k]+rt[k].dist[j]){

        rt[i].dist[j]=rt[i].dist[k+rt[k].dist[j];rt[i].from[j]=k;
                                count++;
                        }
                    }
                }
            }
        }while(count!=0);
        for(i=0;i<nodes;i++){
            printf("\n\nFor router %d\n",i+1);
            for(j=0;j<nodes;j++){
                printf("\t\nnode %d via %d Distance %d
",j+1,rt[i].from[j]+1,rt[i].dist[j]);
            }
        }
        printf("\n\n");
}
```

**OUTPUT**

```
Enter the number of nodes : 3

Enter the cost matrix :
0 2 7
2 0 1
7 1 0


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

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

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

**Experiment No:7**

# SIMPLE MAIL TRANSFER PROTOCOL(SMTP)

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 an always-on listening mode. As soon as it listens for a TCP connection from any client, the SMTP process initiates a connection through port 25. After successfully establishing a TCP connection the client process sends the mail instantly.

**PROGRAM**

**client**

```
#include<string.h>
#include<netdb.h>
#include<stdlib.h>
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#define BUF_SIZE 256
void main()
{
	struct sockaddr_in server,client;
	char str[50]="hi";
	char mail_f[50],mail_to[50],msg[20],c;
	int t=0,s,n;
	s=socket(AF_INET,SOCK_DGRAM,0);
	server.sin_family=AF_INET;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	server.sin_port=4100;
	printf("\nClient ready...\n");
	n=sizeof(server);
	printf("sending hi to server");
	sleep(10);
	sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
	recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
	printf("\nGreeting msg is %s",str);
```

```
if(strncmp(str,"220",3))
        printf("\nCan not established \n code 220 expected");
printf("\nSending HELLO");
strcpy(str,"HELLO 127.0.0.1");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
sleep(3);
printf("\nReceiving from server");
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
if(strncmp(str,"250",3))
        printf("\nOK not received from server");
printf("\nServer has send %s",str);
printf("\nEnter FROM address\n");
scanf("%s",mail_f);
strcpy(str,"MAIL FROM");
strcat(str,mail_f);
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
sleep(3);
printf("\nReceiving from server ");
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
if(strncmp(str,"250",3))
        printf("\nOK not received from server");
printf("%s\n",str);
printf("\nEnter TO address\n");
scanf("%s",mail_to);
strcpy(str,"RCPT TO ");
strcat(str,mail_to);
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
sleep(3);
printf("\nReceiving from server");
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
if(strncmp(str,"250",3))
        printf("\n OK not received from server");
```

```
printf(" %s\n",str);
printf("\nSending data to server");
strcpy(str,"DATA");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
sleep(3);
printf("\nReceiving from server");
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
if(strncmp(str,"354",3))
        printf("\nOK not received from server");
printf("\n%s",str);
printf("\nEnter mail body");
while(1)
{
        c=getchar();
        if(c=='$'){
                msg[t]='\0';
                break;
        }
        if(c=='\0')
                continue;
        msg[t++]=c;
}
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
sleep(3);
printf("\nSending QUIT to server");
strcpy(str,"QUIT");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&server,n);
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&server,&n);
if(strncmp(str,"221",3))
        printf("\nOK not received from server");
printf("\nServer has send GOODBYE.....Closing connection\n");
printf("\nBye");
close(s);
```

```
}
```

**server**

```
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<stdlib.h>
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#define BUF_SIZE 256
void main()
{
	struct sockaddr_in server, client;
	char str[50],msg[20];
	int s,n;
	s=socket(AF_INET,SOCK_DGRAM,0);
	server.sin_family=AF_INET;
	server.sin_addr.s_addr=inet_addr("127.0.0.1");
	server.sin_port=4100;
	bind(s,(struct sockaddr*)&server,sizeof(server));
	n=sizeof(client);
	printf("Server waiting...");
	recvfrom(s,str,100,0,(struct sockaddr*)&client,&n);
	printf("\nGot message from client:%s",str);
	printf("\nSending greeting msg to client");
	strcpy(str,"220 127.0.0.1");
	sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
	sleep(3);
	recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
	if(strncmp(str,"HELO",4))
		printf("\n 'HELO' expected from client...");
```

```
printf("\n%s",str);
printf("\nSending Response...");
strcpy(str,"250 ok");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
sleep(3);
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
if(strncmp(str,"MAIL FROM",9))
        printf("\nMAIL FROM expected from client...");
printf("\n%s",str);
printf("\nSending Response...");
strcpy(str,"250 ok");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
sleep(3);
printf("\nReceiving to address");
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
if(strncmp(str,"RCPT TO",7))
        printf("RCPT TO expected from client...");
printf("\n%s",str);
printf("\nSending Response...");
strcpy(str,"250 ok");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
sleep(3);
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
if(strncmp(str,"DATA",4))
        printf("DATA expected from client...");
printf("\n%s",str);
printf("\nSending Response...");
strcpy(str,"354 go ahead");
sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
printf("\nMail body received");
printf("\n%s",msg);
```

```
    recvfrom(s,str,sizeof(str),0,(struct sockaddr*)&client,&n);
    if(strncmp(str,"QUIT",4))
        printf("QUIT expected from client...");
    printf("sending quit...\n");
    strcpy(str,"221 ok");
    sendto(s,str,sizeof(str),0,(struct sockaddr*)&client,n);
    close(s);
}
```

## OUTPUT

**client**

```
student@ssl-21:~$ ./a.out

Client ready...
sending hi to server
Greeting msg is 220 127.0.0.1
Sending HELLO
Receiving from server
Server has send 250 ok
Enter FROM address
abc@gmail.com

Receiving from server 250 ok

Enter TO address
xyz@gmail.com

Receiving from server 250 ok

Sending data to server
Receiving from server
354 go ahead
Enter mail body hello $

Sending QUIT to server
Server has send GOODBYE......Closing connection

Byestudent@ssl-21:~$
```

**server**

```
student@ssl-21:~$ ./a.out
Server waiting...
Got message from client:hi
Sending greeting msg to client
 'HELO' expected from client...
HELLO 127.0.0.1
Sending Response...
MAIL FROM expected from client...
abc@gmail.com
Sending Response...
Receiving to address
RCPT TO xyz@gmail.com
Sending Response...
DATA
Sending Response...
Mail body received
sending quit...
student@ssl-21:~$
```

**Experiment No:8**

## FILE TRANSFER PROTOCOL

FTP stands for File transfer protocol. FTP is a standard internet protocol provided by TCP/IP used for transmitting the files from one host to another.There are two types of connections in FTP.The control connection is made between the control processes. The data connection is made between data transfer processes. The FTP client has three components – the user interface, control process and data transfer process. The server has two components – the server control process and the server data process. Only one file can be sent over one data connection. But control connection remains active throughout the user session.

## PROGRAM

**server**

```
#include <arpa/inet.h>
```

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 4444
int main()
{
	int sockfd, ret,n;
	char rcv[100],fileread[100];
	FILE *fp;
	struct sockaddr_in serverAddr;
	int clientSocket;
	struct sockaddr_in cliAddr;
	socklen_t addr_size;
	pid_t childpid;
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd < 0) {
		printf("Error in connection.\n");
		exit(1);
	}
	printf("Server Socket is created.\n");
	memset(&serverAddr, '\0',sizeof(serverAddr));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(PORT);
	serverAddr.sin_addr.s_addr= inet_addr("127.0.0.1");
```

```
	ret = bind(sockfd,(struct
sockaddr*)&serverAddr,sizeof(serverAddr));
	if (ret < 0) {
		printf("Error in binding.\n");
		exit(1);
	}
	if (listen(sockfd, 10) == 0)
	printf("Listening...\n\n");
	int cnt = 0;
	while (1) {
		clientSocket = accept(sockfd, (struct
sockaddr*)&cliAddr,&addr_size);
		if (clientSocket < 0)
			exit(1);
		printf("Connection accepted from %
		%d\n",inet_ntoa(cliAddr.sin_addr),ntohs(cliAddr.sin_port));
		printf("Clients connected: %d\n\n",++cnt);
		if ((childpid = fork()) == 0) {
			n=recv(clientSocket,rcv,100,0);
			rcv[n]='\0';
			fp=fopen(rcv,"r");
			if(fp==NULL)
			{
				send(clientSocket,"error",5,0);
				close(clientSocket);
			}
			else
			{
				while(fgets(fileread,sizeof(fileread),fp))
```

```
                        {

    if(send(clientSocket,fileread,sizeof(fileread),0)<0)
                                printf("Can't send file contents\n");
                            sleep(1);
                        }
                        send(clientSocket,"completed",100,0);
                    }
                    close(sockfd);
                }
    }
    close(clientSocket);
    return 0;
}
```

**client**

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 4444
int main()
{
    int clientSocket, ret,s;
    FILE *fp;
```

```
	struct sockaddr_in cliAddr;
	struct sockaddr_in serverAddr;
	char buffer[1024],name[100],fname[100],rcvg[100];
	clientSocket = socket(AF_INET,SOCK_STREAM, 0);
	if (clientSocket < 0) {
		printf("Error in connection.\n");
		exit(1);
	}
	printf("Client Socket is created.\n");
	memset(&cliAddr, '\0', sizeof(cliAddr));
	memset(buffer, '\0', sizeof(buffer));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(PORT);
	serverAddr.sin_addr.s_addr= inet_addr("127.0.0.1");
	ret = connect(clientSocket,(struct
sockaddr*)&serverAddr,sizeof(serverAddr));
	if (ret < 0) {
		printf("Error in connection.\n");
		exit(1);
	}
	printf("Connected to Server.\n");
	while (1) {
		printf("Enter the existing file name\t");
		scanf("%s",name);
		printf("Enter the new file name\t");
		scanf("%s",fname);
		fp=fopen(fname,"w");
		while(1){
			send(clientSocket,name,sizeof(name),0);
```

```
            s=recv(clientSocket,rcvg,100,0);
            if(s<0)
                printf("Error in receiving data");
            else
            {
                rcvg[s]='\0';
                if(strcmp(rcvg,"error")==0)
                    printf("File is not available\n");
                if(strcmp(rcvg,"completed")==0)
                {
                    printf("File is transferred.....\n");
                    break;
                }
                else
                {
                    fputs(rcvg,stdout);
                    fprintf(fp,"%s",rcvg);
                    bzero(rcvg,sizeof(rcvg));
                }
            }
        }
        fclose(fp);
        close(clientSocket);
        return 0;
    }
```

```
student@ssl-21:~$ gcc ftpserver.c
student@ssl-21:~$ ./a.out
Server Socket is created.
Listening...

Connection accepted from 0.0.0.0:0
Clients connected: 1

Connection accepted from 127.0.0.1:42302
Clients connected: 2
```

```
student@ssl-21:~$ gcc ftpclient.c -o c1
student@ssl-21:~$ ./c1
Client Socket is created.
Connected to Server.
Enter the existing file name    hello.txt
Enter the new file name hello1.txt
Hello, how are you?
File is transferred........
student@ssl-21:~$ gcc ftpclient.c -o c2
student@ssl-21:~$ ./c2
Client Socket is created.
Connected to Server.
Enter the existing file name    hii.txt
Enter the new file name hii1.txt
Have a nice day!
File is transferred........
student@ssl-21:~$
```

}

**OUTPUT**

**Experiment No:9**

## LEAKY BUCKET ALGORITHM

Leaky bucket algorithm is an open loop method. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite

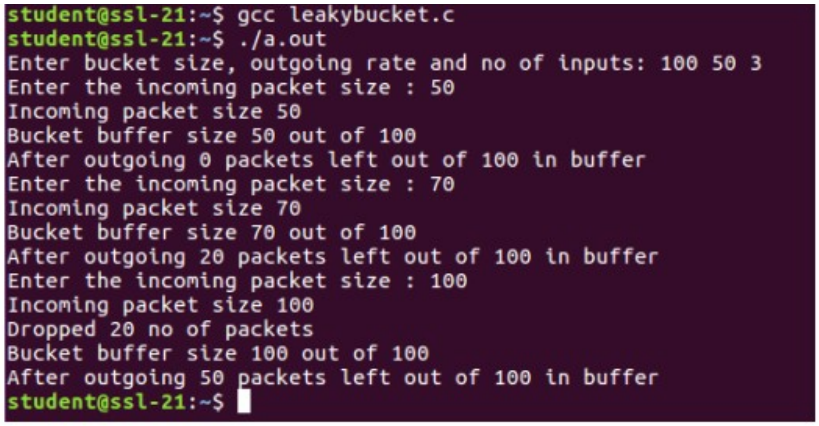
internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time. The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

**PROGRAM**

```
#include<stdio.h>
int main(){
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and no of inputs: ");
    scanf("%d %d %d", &buck_size, &outgoing, &n);
    while (n != 0) {
        printf("Enter the incoming packet size : ");
        scanf("%d", &incoming);
        printf("Incoming packet size %d\n", incoming);
        if (incoming <= (buck_size – store))
        {
            store += incoming;
            printf("Bucket buffer size %d out of %d\n", store,
buck_size);
        }
        else {
            printf("Dropped %d no of packets\n", incoming -
(buck_size - store));
            printf("Bucket buffer size %d out of %d\n", buck_size,
buck_size);
```

```
                store = buck_size;
            }
            store = store - outgoing;
            printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
            n--;
        }
}
```

**OUTPUT**



**Experiment No:10**

## NS2 SIMULATOR

Network simulation (NS) is one of the types of simulation, which is used to simulate the networks such as in MANETs, VANETs, etc. It provides simulation for routing and multicast protocols for both wired and wireless networks. NS is licensed for use under version 2 of the GNU (General Public License) and is popularly known as NS2. It is an object-oriented, discrete event-driven simulator written in C++ and Otcl/Tcl.

NS-2 can be used to implement network protocols such as TCP and UDP, traffic source behaviour such as FTP, Telnet, Web, CBR, and VBR, router queues management mechanism such as Drop Tail, RED, and CBQ, routing algorithms, and many more. In ns2, C++ is used for detailed protocol implementation and Otcl is used for the setup. The compiled C++ objects are made available to the Otcl interpreter and in this way, the ready-made C++ objects can be controlled from the OTcl level.

Features of NS2 :

- It’s a networking research discrete event simulator.

- It has a lot of features for simulating protocols including TCP, FTP, UDP, HTTPS, and DSR.
- It is capable of simulating both wired and wireless networks.
- It is mostly based on Unix.
- Its scripting language is TCL.
- Tclcl is a C++ and otcl linkage language.
- Scheduler for discrete events.

**set ns [new Simulator] :** generates an NS simulator object instance, and assigns it to variable ns

**$ns color fid color :** is to set color of the packets for a flow specified by the flow id (fid).

**$ns namtrace-all file-descriptor :** This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format.

**proc finish {} :** is called after this simulation is over by the command $ns at 5.0 "finish". In this function, post-simulation processes are specified.

**set n0 [$ns node] :** The member function node creates a node.

**$ns duplex-link node1 node2 bandwidth delay queue-type :** creates two simplex links of specified bandwidth and delay, and connects the two specified nodes.

**$ns queue-limit node1 node2 number :** This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified.

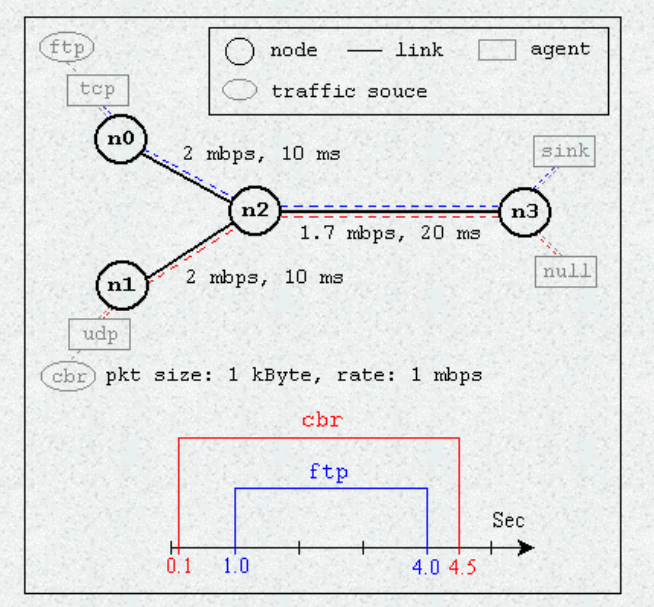
**set tcp [new Agent/TCP] :** This line shows how to create a TCP agent.

**$ns attach-agent node agent :** The attach-agent member function attaches an agent object created to a node object.

**$ns connect agent1 agent2 :** This line establishes a network connection by setting the destination address to each others' network and port address pair.

**$ns at time "string" :** This member function of a Simulator object makes the scheduler to

schedule the execution of the specified string at given simulation time.




**PROGRAM**

```
#Create a simulator object
set ns [new Simulator]


#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red


#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf


#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
```

```
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]

set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
```

```
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2



#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
```

```
#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

**Experiment No:11**

## LAN CONFIGURATION

### IP addresses

An IP address, or Internet Protocol address, is a series of numbers that identifies any device on a network. Computers use IP addresses to communicate with each other both over the internet as well as on other networks. An IP address serves two main functions: network interface identification and location addressing.

An IP address consists of two parts: the network ID and host ID. The Network ID indicates which network the device is on. The Host ID refers to the specific device on that network.There are two types of IP addresses: IPv4 and IPv6.

IPv4 addresses contain a series of four numbers, ranging from 0 (except the first one) to 255, each separated from the next by a period — such as 5.62.42.77. IPv6 addresses are represented as eight groups of four hexadecimal digits, with the groups separated by colons.

**Subnetting**

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 the same subnet are addressed with an identical most-significant bit-group in their IP addresses.

A subnet mask is a 32 bits address used to distinguish between a network address and a host address in IP address. A subnet mask identifies which part of an IP address is the network address and the host address. Class A networks use a default subnet mask of 255.0.0.0. Class B networks use a default subnet mask of 255.255.0.0. Class C networks use a default subnet mask of 255.255.255.0.

**Gateways**

A gateway is a network node that forms a passage between two networks operating with different transmission protocols. The network gateway operates at layer 3, i.e. network layer of the OSI (open systems interconnection) model. Depending upon the functionality, a gateway can operate at any of the seven layers of OSI model. It acts as the entry – exit point for a network since all traffic that flows across the networks should pass through the gateway. Only the internal traffic between the nodes of a LAN does not pass through the gateway.

**How to set LAN**

LAN (Local Area Network) is a data communication network that locally connects network devices such as workstations, servers, routers, etc. to share the resources within a small area such as a building or

campus. Ethernet and Wi-fi are the most important technologies of LAN. eg: Personal networks at home, school, office.

Requirements to set up LAN Network:

- Workstation/Personal devices: laptop, computer, mobile phones, etc.
- Network devices: router, switch, modem (if not already present in the router)
- Sharing resources: printers, disk drives, etc.
- Cables: Ethernet cables, wires for connecting other devices (in case of wired LAN)
- Internet connection: Wi-Fi (in case of wireless LAN)

Instructions to set up LAN Network:

Following steps should be followed to set up a LAN network:

1. Identify services: Identify the network services such as printers, disk drives, data, etc. that will be shared among workstations.
2. Identify devices: Identify devices such as computers, mobile phones, laptops, etc. with a unique address that will be connected to the network.
3. Plan connections: Design the network by laying out cable wires between network devices or by making wireless connections. Wired LAN is set up using Ethernet cables while wireless LAN is set up using Wi-Fi that connects network devices without making any physical connection. A wired LAN network is more secure than a wireless LAN network but it is difficult to relocate.
4. Select networking device: Select switch or router with enough ports to connect all workstations within the network. The choice of networking device is based on the requirements of the network.
5. Configure ports: Configure WAN ports according to the information provided by ISP (Internet Service Provider). Also, configure LAN ports of cable routers such that there are enough addresses available for all the workstations within the network. A cable router acts as DHCP (Dynamic Host Configuration Server) server that automatically allocates addresses to all the devices connected to the network.

6. Make connections: Connect all the devices using wires to configure a LAN network. Standard Ethernet cables are used to connect workstations and servers while Ethernet crossover cable is used to connect the switch to cable routers by connecting the standard port of the switch with router's LAN port. For wireless LAN, connect all the devices to Wi-Fi with SSID (Service Set Identifier) provided by the router or switch to configure the LAN network.
7. Test the network: Test each of the workstation connected to the network and ensure every workstation have access to network services.

## LAN Settings window in UBUNTU